Grimoire Assembleur 80x86

Première Edition

Avril 2010

Par

Sylvain Maltais

Numéro de publication : 00001

Annexe A

Voici une table des instructions du langage de programmation assembleur de la famille 80x86 :

Instruction	Description	CPU/MPU
AAA	Cette instruction permet d'adapter le résultat obtenu par l'addition de 2 valeurs en format *DCB*.	INTEL 8088+
AAD	Cette instruction permet de convertir une valeur de format *DCB* non compactée.	INTEL 8088+
AAM	Cette instruction offre la possibilité de convertir le produit de la multiplication de 2 valeurs de format *DCB* en un format *DCB*.	INTEL 8088+
AAS	Cette instruction permet d'adapter le résultat de la soustraction de nombre de format *DCB*.	INTEL 8088+
ADC	Cette instruction additionne 2 quantités numériques sur 8 ou 16 bits et ajoute ensuite la valeur de l'indicateur de retenue, lequel est contenu dans le drapeau *CF* (*CARRY FLAG)*, à la somme obtenu.	INTEL 8088+
ADD	Cette instruction additionne 2 quantités numériques sur 8 ou 16 bits.	INTEL 8088+
ADDPD	Cette instruction permet d'effectuer une addition de 2 paquets de valeurs réels de double précision d'un opérande source et d'un opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de double précision.	INTEL Pentium 4+
ADDPS	Cette instruction permet d'effectuer une addition *SIMD* de 4 paquets de valeurs réels de simple précision d'un opérande source et d'un opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de simple précision.	INTEL Pentium III (SSE)+
ADDSD	Cette instruction permet d'effectuer une addition de la partie basse d'une valeur réel de double précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de double précision.	INTEL Pentium 4+
ADDSS	Cette instruction permet d'effectuer une addition de la partie basse d'une valeur réel de simple précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de simple précision.	INTEL Pentium III (SSE)+
ADDSUBPD	Cette instruction permet d'effectuer une soustraction de la partie basse d'une valeur réel de double précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de double précision et d'effectuer une addition de la partie haute d'une valeur réel de double précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de double précision.	INTEL Pentium 4 (SSE3)+
ADDSUBPS	Cette instruction permet d'effectuer une soustraction de la partie basse d'un paquet de valeur réel de simple précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de simple précision et d'effectuer une addition de la partie haute d'un paquet de valeur réel de simple précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de simple précision.	INTEL Pentium 4 (SSE3)+
AND	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 opérandes spécifiés.	INTEL 8088+
ANDNPD	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 paquets d'opérandes de valeur de format réel de double précision et inverse chacun des bits du résultat.	INTEL Pentium 4+
ANDNPS	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 opérandes 128 bits spécifiés et inverse chacun des bits du résultat.	INTEL Pentium III (SSE)+
ANDPD	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 paquets	INTEL Pentium 4+

	d'opérandes de valeur de format réel de double précision.	
ANDPS	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 opérandes 128 bits spécifiés.	INTEL Pentium III (SSE)+
ARPL	Cette instruction permet de contrôler et corriger le niveau de privilège du segment de code en mode protégée.	INTEL 80286+
BOUND	Cette instruction permet de vérifier la validité pouvant exister entre avec un tableau et son indexation.	INTEL 80186+
BSF	Cette instruction permet d'effectuer une comparaison binaire de la droite vers la gauche.	INTEL 80386+
BSR	Cette instruction permet d'effectuer une comparaison binaire de la gauche vers la droite.	INTEL 80386+
BSWAP	Cette instruction permet d'inverse l'ordre des 4 octets d'un registre de taille de 32 bits.	INTEL 80486+
BT	Cette instruction permet de transférer une *Opérande* vers l'indicateur de retenue.	INTEL 80386+
BTC	Cette instruction transfère une *Opérande* vers l'indicateur de retenue le bit spécifié et inverser la valeur du bit correspondant.	INTEL 80386+
BTR	Cette instruction permet de transférer l'*Opérande* vers l'indicateur de retenue le bit spécifié puis met le bit correspondant de l'*Opérande* à 0.	INTEL 80386+
BTS	Cette instruction permet de transférer une *Opérande* vers l'indicateur de retenue le bit spécifié puis ensuite met le bit correspondant dans *Opérande* à 1.	INTEL 80386+
CALL	Cette instruction force le microprocesseur à exécuter les instructions du sous-programme indiqué par l'adresse d'appel avant de continuer.	INTEL 8088+
CBW	Cette instruction permet de convertir un nombre contenu dans le registre *AL* en un format sur 16 bits pour se retrouver dans le registre *AX* en appliquant une extension du signe.	INTEL 8088+
CDQ	Cette instruction permet de convertir le double mot en un quadruple mot, le résultat tient sur une taille de 64 bits.	INTEL 80386+
CDQE	Cette instruction permet de convertir le double mot du registre EAX en un quadruple mot RAX, le résultat tient sur une taille de 64 bits.	x86-64+
CLC	Cette instruction permet de mettre l'indicateur d'état de retenu *CF* à 0.	INTEL 8088+
CLD	Cette instruction met l'indicateur d'état *DF* à 0.	INTEL 8088+
CLFLUSH	Cette instruction permet de vider la ligne de cache d'une adresse linéaire.	INTEL Pentium 4 (SSE2)+
CLGI	Cette instruction permet d'effacer le drapeau global d'interruption (*GIF*). Quand le *GIF* est à 0, toutes les interruptions externes sont désactivées.	AMD-V
CLI	Cette instruction met l'indicateur d'état *IF* à 0.	INTEL 8088+
CLTS	Cette instruction permet de mettre à 0 le drapeau de l'indicateur de tâche (*Task-Switch*) du registre *CR0*.	INTEL 80286+
CMC	Cette instruction offre la possibilité d'inverser la valeur de l'indicateur de retenue. Si ce dernier vaut 1, elle le met à 0 et inversement.	INTEL 8088+
CMOV	Cette instruction copie des données d'une source (*valeur*) à une destination (*registre*) à la condition que la condition demandée soit remplie.	INTEL Pentium Pro+
CMP	Cette instruction offre la possibilité essentielle de comparer 2 registres ou emplacements de mémoire.	INTEL 8088+
CMPPS	Cette instruction permet d'effectuer une comparaison *SIMD* de 4 paquets de valeurs réels de simple précision d'un opérande source et d'un opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.	INTEL Pentium III (SSE)+

CMPS	Cette instruction permet d'effectuer la comparaison d'un octet, d'un mot ou double mot spécifié avec l'opérande source et destination spécifié et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.	INTEL 8088+
CMPSB	Cette instruction permet d'effectuer la comparaison d'un octet avec l'opérande source (DS:(R)SI) et destination (ES:(R)DI) et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.	INTEL 8086+
CMPSD	Cette instruction permet d'effectuer la comparaison d'un double mot avec l'opérande source (DS:(R)SI) et destination (ES:(R)DI) et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.	INTEL 80386+
CMPSQ	Cette instruction permet d'effectuer la comparaison d'un quadruple mot avec l'opérande source (DS:(R)SI) et destination (ES:(R)DI) et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.	x86-64+
CMPSS	Cette instruction permet d'effectuer une comparaison de la partie basse de valeurs réelles de simple précision d'un opérande source et d'un opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.	INTEL Pentium III (SSE)+
CMPSW	Cette instruction permet d'effectuer la comparaison d'un mot avec l'opérande source (DS:(R)SI) et destination (ES:(R)DI) et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.	INTEL 8086+
CMPXCHG	Cette instruction compare la destination avec l'accumulateur (*AL*, *AX* ou *EAX*) et les échanges si la condition est vraie.	INTEL Pentium+
CMPXCHG8B	Cette instruction compare un nombre de 64 bits et les échanges si la condition est vraie.	INTEL Pentium+
CMPXCHG16B	Cette instruction permet de comparer un nombre de 128 bits et les échanges si la condition est vraie.	INTEL x86-64+
COMISD	Cette instruction permet de comparer deux valeurs réel de double-précision dans la partie faible du quadruple mot de deux opérande et fixe les drapeaux de ZF, PF et FC du registre EFLAGS en fonction du résultat (non-ordonnée, supérieur à, inférieur ou égal).	INTEL Pentium III+
COMISS	Cette instruction permet de comparer deux valeurs réel de simple-précision dans la partie faible du quadruple mot de deux opérande et fixe les drapeaux de ZF, PF et FC du registre EFLAGS en fonction du résultat (non-ordonnée, supérieur à, inférieur ou égal).	INTEL Pentium III+
CPUID	Cette instruction retourne le code d'identification du microprocesseur.	INTEL Pentium+
CQO	Cette instruction permet de convertir le quadruple mot du registre EAX en deux quadruples mots RDX:RAX, le résultat tient sur une taille de 128 bits.	x86-64+
CVTDQ2PD	Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de double précision.	INTEL Pentium 4+
CVTDQ2PS	Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de simple précision.	INTEL Pentium 4+
CVTPD2DQ	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en double mot entier et fixe à 0 le reste du paquet.	INTEL Pentium 4+
CVTPD2PI	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en double mot entier.	INTEL Pentium 4+
CVTPD2PS	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en valeurs réel de simple précision et fixe à 0 le reste du paquet.	INTEL Pentium 4+
CVTPI2PD	Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de double précision compacté.	INTEL Pentium 4+
CVTPI2PS	Cette instruction permet de convertir un entier de format double mot en une valeur réel de simple précision compacté.	Pentium III (KNI/MMX2)+
CVTPS2DQ	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel	INTEL Pentium 4+

	de simple précision en paquet de doubles mots entiers.	
CVTPS2PD	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de simple précision en paquet de valeurs réel de double précision.	INTEL Pentium 4+
CVTPS2PI	Cette instruction permet de convertir une valeur réel de simple précision compacté en un entier de format double mot.	Pentium III (KNI/MMX2)+
CVTSD2SI	Cette instruction permet d'effectuer la conversion d'une valeur réel de simple précision en valeur entier de double mots.	INTEL Pentium 4+
CVTSD2SS	Cette instruction permet d'effectuer la conversion d'une valeur réel de double précision en valeur réel de simple précision.	INTEL Pentium 4+
CVTSI2SD	Cette instruction permet de convertir un entier de format double mot en une valeur réel de double précision.	INTEL Pentium 4+
CVTSI2SS	Cette instruction permet de convertir un entier de format double mot en une valeur réel de simple précision.	Pentium III (KNI/MMX2)+
CVTSS2SD	Cette instruction permet de convertir une valeur réel de simple précision en une valeur réel de double précision.	INTEL Pentium 4+
CVTSS2SI	Cette instruction permet de convertir une valeur réel de simple précision en un entier de format double mot.	Pentium III (KNI/MMX2)+
CVTTPD2DQ	Cette instruction permet de convertir un paquet de valeur réel de double précision avec en un paquet d'entier de format double mot tronquer et fixe à 0 le reste du paquet.	INTEL Pentium 4+
CVTTPD2PI	Cette instruction permet de convertir une valeur réel de double précision compacté avec tronquage en un entier de format double mot compacté.	INTEL Pentium 4+
CVTTPS2DQ	Cette instruction permet de convertir un paquet de valeur réel de simple précision avec en un paquet d'entier de format double mot tronquer.	INTEL Pentium 4+
CVTTPS2PI	Cette instruction permet de convertir une valeur réel de simple précision compacté avec tronquage en un entier de format double mot compacté.	Pentium III (KNI/MMX2)+
CVTTSD2SI	Cette instruction permet de convertir une valeur réel de double précision avec en un entier de format double mot tronquer.	INTEL Pentium 4+
CVTTSS2SI	Cette instruction permet de convertir un entier de format double mot avec tronquage en une valeur réel de simple précision.	Pentium III (KNI/MMX2)+
CWD	Cette instruction est l'alternative pour convertir le mot du registre *AX* en un double mot contenu dans le couple de registre *DX* et *AX* par extension du signe.	INTEL 8088+
CWDE	Cette instruction est l'alternative pour convertir le mot du registre *AX* en un double mot contenu dans le registre *EAX* par extension du signe.	INTEL 80386+
DAA	Cette instruction corrige après coup les retenues lors de la manipulation de valeur *DCB*.	INTEL 8088+
DAS	Cette instruction offre l'intéressante possibilité de corrige le résultat d'une soustraction de 2 nombres de format *DCB* compactées.	INTEL 8088+
DEC	Cette instruction décrément de 1 le registre ou l'adresse mémoire spécifié.	INTEL 8088+
DIV	Cette instruction permet d'effectuer une division non-signée (nombre naturel).	INTEL 8088+
DIVPD	Cette instruction permet d'effectuer une division de valeur réel de double précision de registre XMM ou d'emplacement mémoire de 128 bits.	INTEL Pentium 4+
DIVPS	Cette instruction permet d'effectuer une division de valeur réel de simple précision de registre XMM ou d'emplacement mémoire de 128 bits.	INTEL Pentium III (KNI/MMX2)+
DIVSD	Cette instruction permet d'effectuer une division scalaire de valeur réel de double précision de registre XMM ou d'emplacement mémoire de 128 bits.	INTEL Pentium 4+
DIVSS	Cette instruction permet d'effectuer une division scalaire de valeur réel de simple précision de registre XMM ou d'emplacement mémoire de 128 bits.	Pentium III (KNI/MMX2)+

EMMS	Cette instruction fixe les mots marqués du *MPU* de chacun un.	INTEL Pentium-MMX
ENTER	Cette instruction permet de créer les structures de paramètres nécessaires aux procédures des langages de haut niveau.	INTEL 80186+
ESC	Cette instruction active le coprocesseur, lequel se permettra d'utiliser les méthodes d'adressage du processeur principal (*CPU*).	INTEL 8088 au 80386
F2XM	Cette instruction s'applique à calculer 2 puissance le registre mathématique *ST(0)* moins 1 et enregistre son produit dans le registre mathématique *ST(0)*. Alias de *F2XM1*, mais déconseillé.	INTEL MPU 8087+
F2XM1	Cette instruction permet d'effectuer le calcul de 2 puissance le registre mathématique *ST(0)* moins 1 et enregistre son produit dans le registre mathématique *ST(0)*.	INTEL MPU 8087+
FABS	Cette instruction permet de convertir le nombre réel contenu dans le registre mathématique *ST(0)* en sa valeur absolue.	INTEL MPU 8087+
FADD	Cette instruction offre l'essentiel possibilité d'ajouter le nombre réel de valeur positive «*source*» à «*cible*» et enregistre la somme dans «*cible*».	INTEL MPU 8087+
FADDP	Cette instruction complémentaire ajoute le nombre réel *source* au nombre réel *cible* et enregistrer la somme dans *cible* puis prendre le registre mathématique *ST(0)* et le dépile.	INTEL MPU 8087+
FBLD	Cette instruction charge la valeur de format *DCB* compactée après le registre mathématique *ST(0)*.	INTEL MPU 8087+
FBSTP	Cette instruction extrait la valeur de format *DCB* compactée du registre mathématique *ST(0)*, l'enregistrer dans *cible* et prendre le registre mathématique *ST(0)* et sort de la pile.	INTEL MPU 8087+
FCHS	Cette instruction inverse tout simplement le signe du registre mathématique *ST(0)*.	INTEL MPU 8087+
FCLEX	Cette instruction efface toutes les exceptions contenu dans les drapeaux de registres du coprocesseur mathématique.	INTEL MPU 8087+
FCMOV	Cette instruction permet de déplacé des nombres réel (virgule flottante) si la condition en question est satisfaite.	INTEL P6+
FCOM	Cette instruction permet de comparer le nombre réel de valeur positive *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.	INTEL MPU 8087+
FCOMI	Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats.	INTEL Pentium III+
FCOMIP	Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats et désempile de la pile la valeur dans le registre.	INTEL Pentium III+
FCOMP	Cette instruction compare le nombre réel de valeur positive «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.	INTEL MPU 8087+
FCOMPP	Cette instruction permet de comparer les nombres réels du registre mathématique *ST(1)* avec le registre mathématique *ST(0)*, ensuite de mettre les indicateurs d'état *C0* à *C3* à la valeur 1 et prendre le couple de registre mathématique *ST(1)* et *ST(0)* dans la pile de registres (2 dépilages).	INTEL MPU 8087+
FCOS	Cette instruction calcule le cosinus du registre mathématique *ST(0)*.	INTEL MPU 80387+
FDECSTP	Cette instruction permet de décrémenter le pointeur de pile de registres.	INTEL MPU 8087+
FDISI	Cette instruction permettait de désactiver les interruptions.	INTEL MPU 8087+
FDIV	Cette instruction offre la possibilité d'effectuer des division de nombre réel de valeur positive.	INTEL MPU 8087+
FDIVP	Cette instruction offre la possibilité d'effectuer des division de nombre réel et	INTEL MPU 8087+

	place *ST(0)* dans la pile.	
FDIVR	Cette instruction offre la possibilité d'effectuer des division de nombre réel de valeur positive avec les registres inversés «*cible*» par «*source*».	INTEL MPU 8087+
FDIVRP	Cette instruction offre la possibilité d'effectuer des division de nombre réel avec les registres inversés «*cible*» par «*source*» et place *ST(0)* dans la pile.	INTEL MPU 8087+
FEMMS	Cette instruction permet d'effacer les états *MMX* après le passage d'instructions *MMX*.	AMD 3DNow!
FENI	Cette instruction permet d'autoriser les interruptions.	INTEL MPU 8087
FFREE	Cette instruction permet de libérer un des registres de la pile.	INTEL MPU 8087+
FFREEP	Cette instruction permet de libérer un des registres de la pile et d'ensuite de désempiler le registre de la pile.	INTEL MPU 80287+
FIADD	Cette instruction offre l'essentiel possibilité d'ajoute le nombre réel source à cible et enregistre la somme dans cible.	INTEL MPU 8087+
FICOM	Cette instruction permet de comparer le nombre réel *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.	INTEL MPU 8087+
FICOMP	Cette instruction compare le nombre réel «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.	INTEL MPU 8087+
FIDIV	Cette instruction offre la possibilité d'effectuer des division de nombre réel.	INTEL MPU 8087+
FIDIVR	Cette instruction offre la possibilité d'effectuer des division de nombre réel avec les registres inversés «*cible*» par «*source*».	INTEL MPU 8087+
FILD	Cette instruction permet de charger le nombre entier auprès du registre ST(0).	INTEL MPU 8087+
FIMUL	Cette instruction permet de multiplier le nombre réel «source» par «*cible*» et sauvegarde le résultat dans «*cible*».	INTEL MPU 8087+
FINCSTP	Cette instruction permet d'incrémenter le pointeur de pile de registres.	INTEL MPU 8087+
FINIT	Cette instruction permet d'effectuer l'initialisation du coprocesseur mathématique.	INTEL MPU 8087+
FIST	Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*».	INTEL MPU 8087+
FISTP	Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*» et prendre le registre *ST(0)* de la pile du registres.	INTEL MPU 8087+
FISUB	Cette instruction offre l'essentielle posibilité de soustraire le nombre «*source*» du registre *ST(0)*.	INTEL MPU 8087+
FISUBR	Cette instruction offre l'essentielle posibilité de soustraire le nombre du registre *ST(0)* de «*source*».	INTEL MPU 8087+
FLD	Cette instruction permet de charger le nombre réel du registre *ST(0)* dans la pile.	INTEL MPU 8087+
FLD1	Cette instruction permet de charger la constante 1,0 du registre *ST(0)* dans la pile.	INTEL MPU 8087+
FLDCW	Cette instruction permet de charger le «mot de contrôle» avec «*source*».	INTEL MPU 8087+
FLDENV	Cette instruction permet de charger l'environnement à partir de «*source*».	INTEL MPU 8087+
FLDL2E	Cette instruction permet de mettre le résultat du logarithme de «*e*» de la base 2 dans *ST(0)*.	INTEL MPU 8087+
FLDL2T	Cette instruction permet de mettre le résultat du logarithme de 10 de la base 2 dans *ST(0)*.	INTEL MPU 8087+
FLDLG2	Cette instruction permet de mettre le résultat du logarithme de 2 de la base 10 dans *ST(0)*.	INTEL MPU 8087+

FLDLN2	Cette instruction permet de mettre le résultat du logarithme de «*e*» de la base 10 dans *ST(0)*.	INTEL MPU 8087+
FLDPI	Cette instruction permet de mettre le résultat de la constante *PI* (Π) dans le registre *ST(0)*.	INTEL MPU 8087+
FLDZ	Cette instruction permet de mettre le résultat de la constante 0,0 dans le registre *ST(0)*.	INTEL MPU 8087+
FMUL	Cette instruction permet de multiplier le nombre réel de valeur positive «source» par «*cible*» et sauvegarde le résultat dans «*cible*».	INTEL MPU 8087+
FMULP	Cette instruction permet de multiplier le nombre réel de valeur positive «*source*» par «*cible*» et sauvegarde le résultat dans «*cible*» et prend ensuite *ST(0)* dans la pile de registres.	INTEL MPU 8087+
FNCLEX	Cette instruction permet d'éliminer et de gérer les exceptions non masquées.	INTEL MPU 8087+
FNDISI	Cette instruction permet de désactiver les interruptions et de gérer les exceptions non masquées.	INTEL MPU 8087
FNENI	Cette instruction permet d'activer les interruptions et de gérer les exceptions non masquées.	INTEL MPU 8087
FNINIT	Cette instruction permet d'initialiser le coprocesseur mathématique et de gérer les exceptions non masquées.	INTEL MPU 8087+
FNOP	Cette instruction permet de ne rien faire et de simplement passer à l'instruction suivante.	INTEL MPU 8087+
FNSAVE	Cette instruction permet de sauvegarder l'état courant du coprocesseur mathématique dans l'emplacement mémoire à partir de «*source*» et gérer les exceptions numériques non masquées.	INTEL MPU 8087+
FNSTCW	Cette instruction permet de sauvegarder le «mot de contrôle» dans «cible» et gérer les exceptions numériques non masquées.	INTEL MPU 8087+
FNSTENV	Cette instruction permet de copier l'environnement du coprocesseur mathématique vers une *cible* mais sans toutefois attendre que l'exception de nombre réel (virgule flottante) soit effacé.	INTEL MPU 8087+
FNSTSW	Cette instruction permet de copier le mot d'état du coprocesseur mathématique vers une *cible* mais sans toutefois attendre que l'exception de nombre réel (virgule flottante) soit effacé.	INTEL MPU 8087+
FPATAN	Cette instruction permet de calculer le résultat de l'arc tangente de *ST(1)* par *ST(0)* et de mettre le résultat dans le registre *ST(0)*.	INTEL MPU 8087+
FPREM	Cette instruction permet de diviser le registre *ST(0)* par le registre *ST(1)* et enregistre le reste dans registre *ST(0)*.	INTEL MPU 8087+
FPREM1	Cette instruction permet de diviser le registre *ST(0)* par le registre *ST(1)* et enregistre le reste *IEEE* dans registre *ST(0)*.	INTEL MPU 80387+
FPTAN	Cette instruction permet d'effectuer le calcule de la fonction trigonométrique de la tangente du registre *ST(0)* et enregistre son résultat dans le registre *ST(0)*.	INTEL MPU 8087+
FRICHOP	Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode *CHOP*.	Cyrix 486 avec FPU
FRINEAR	Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode du plus proche.	Cyrix 486 avec FPU
FRINT2	Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode du plus proche et si le nombre est exactement la moitié, alors l'instruction arrondit au signe de l'infini.	Cyrix 486 avec FPU
FRNDINT	Cette instruction permet d'arrondir le registre *ST(0)* à l'entier le plus proche et enregistre son résultat dans le registre *ST(0)*.	INTEL MPU 8087+
FRSTOR	Cette instruction permet de restituer l'état du coprocesseur précédamment sauvegardé dans l'emplacement mémoire spécifié par *source*.	INTEL MPU 8087+

FSAVE	Cette instruction permet de sauvegarder l'état courant du coprocesseur dans l'emplacement mémoire spécifié par *source*.	INTEL MPU 8087+
FSCALE	Cette instruction permet d'effectuer la multiplication du registre *ST(0)* par 2 puissance le registre *ST(1)* et sauvegarde le résultat dans le registre *ST(0)*.	INTEL MPU 8087+
FSETPM	Cette instruction permet de faire passer le coprocesseur mathématique en mode protégé.	INTEL MPU 80287
FSIN	Cette instruction permet d'effectuer le calcule de la fonction trigonométrique du sinus du registre *ST(0)* et le copie dans le registre *ST(1)*, le registre *ST(0)* prend la valeur 1,0.	INTEL MPU 80387+
FSINCOS	Cette instruction permet d'effectuer le calcule de la fonction trigonométrique du sinus et du cosinus du registre *ST(0)* puis copie le résultat du sinus dans le registre *ST(0)* et le cosinus dans le registre *ST(1)*.	INTEL MPU 80387+
FSQRT	Cette instruction permet d'extraire la racine carrée du registre *ST(0)* et de copier son résultat dans le registre *ST(0)*.	INTEL MPU 8087+
FST	Cette instruction permet de copier la valeur réel contenu le registre *ST(0)* vers une *cible*.	INTEL MPU 8087+
FSTCW	Cette instruction permet de copier le mot de contrôle *CW* vers une *cible*.	INTEL MPU 8087+
FSTENV	Cette instruction permet de copier l'environnement du coprocesseur mathématique vers une *cible*.	INTEL MPU 8087+
FSTP	Cette instruction permet de copier un nombre entier et prendre le contenu du registre *ST(0)* dans la pile de registres (dépilage).	INTEL MPU 8087+
FSTSW	Cette instruction permet de copier le mot d'état du coprocesseur mathématique vers une *cible*.	INTEL MPU 8087+
FSUB	Cette instruction permet de soustraire un nombre entier du registre *ST(0)*.	INTEL MPU 8087+
FSUBP	Cette instruction permet de soustraire un nombre entier du registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.	INTEL MPU 8087+
FSUBPP	Cette instruction permet de soustraire un nombre entier du registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres. Cette instruction est déconseillée.	INTEL MPU 8087+
FSUBR	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)*.	INTEL MPU 8087+
FSUBRP	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.	INTEL MPU 8087+
FTST	Cette instruction permet de comparer le sommet de la pile avec la valeur 0,0 et définit les indicateurs d'états C0 et C3 de façon approprié.	INTEL MPU 8087+
FUCOM	Cette instruction compare le registre *ST0* avec un opérande et fixe les drapeaux de façon approprié.	INTEL MPU 80387+
FUCOMI	Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et vérifie l'ordre des valeurs et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats.	INTEL Pentium III+
FUCOMIP	Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et vérifie l'ordre des valeurs et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats et désempile de la pile la valeur dans le registre.	INTEL Pentium III+
FUCOMP	Cette instruction compare le registre *ST0* avec un opérande et fixe les drapeaux de façon approprié et désempile la pile.	INTEL MPU 80387+
FUCOMPP	Cette instruction compare le registre *ST0* avec le registre *ST1* et fixe les drapeaux de façon approprié et désempile les registres *ST0* et *ST1* de la pile.	INTEL MPU 80387+
FWAIT	Cette instruction permet d'attendre la fin de l'exécution d'une commande avant de poursuivre.	INTEL MPU 8087+

FXAM	Cette instruction permet d'examiner le sommet de la pile et définit les indicateurs d'état C0 et C3 de façon approprié.	INTEL MPU 8087+
FXCH	Cette instruction permet d'échanger le contenu d'une expression avec celle du registre *ST(0)* s'il est omis.	INTEL MPU 8087+
FXRSTOR	Cette instruction permet de restituer rapidement 94 (en mode 16 bits) ou 108 (en mode 32 bits) octets d'un contexte de coprocesseur mathématique vers la mémoire.	INTEL MMX-2+
FXSAVE	Cette instruction permet de sauvegarder rapidement 94 (en mode 16 bits) ou 108 (en mode 32 bits) octets d'un contexte de coprocesseur mathématique de la mémoire.	INTEL MMX-2+
FXTRACT	Cette instruction permet de copier l'exposant du registre *ST(0)* dans le registre *ST(1)* comme nombre entier puis copie la mantisse dans la registre *ST(0)* comme nombre entier.	INTEL MPU 8087+
FYL2X	Cette instruction permet de calculer la multiplication du registre *ST(1)* par le logarithme du registre *ST(0)* dans la base 2 et copie le résultat dans le registre *ST(0)*.	INTEL MPU 8087+
FYL2XP1	Cette instruction permet de calculer la multiplication du registre *ST(1)* par le logarithme du registre *ST(0)* plus 1 dans la base 2 et copie le résultat dans le registre *ST(0)*.	INTEL MPU 8087+
HLT	Cette instruction permet de faire passer le microprocesseur en mode d'arrêt. Toutefois, le processeur peut quitter cet état lorsqu'une ligne matérielle RESET ou lorsqu'une interruption non-masquable (*NMI*) reçoit un signal.	INTEL 8088+
IBTS	Cette instruction permet de faire un tableau de bits du deuxième opérand et le met dans le premier.	INTEL 80386+
ICEBP	Cette instruction permet de passer en mode d'instruction *ICE* (*In-Circuit Emulator*) si le bit 12 du registre DR7 vaut 1 sinon il exécute l'interruption 1.	INTEL 80386 au Pentium Pro
IDIV	Cette instruction permet d'effectuer une division signée (nombre entier).	INTEL 8086+
IMUL	Cette instruction permet d'effectuer une multiplication signée (nombre entier).	INTEL 8086+
IN	Cette instruction permet de demander un octet, un mot ou un double mot provenant du port d'entrée/sortie et le retourne dans le registre accumulateur (AL, AX, EAX).	INTEL 8086+
INC	Cette instruction permet d'incrémenter un registre ou un emplacement mémoire.	INTEL 8086+
INS	Cette instruction permet de demander un octet, un mot ou un double mot du port d'entrée/sortie et retourne le résultat dans l'adresse ES:[DI] et incrémente/décrémente le registre DI en fonction de la taille de l'opérande cible et de l'état du drapeau de direction.	INTEL 8086+
INSB	Cette instruction permet de demander un octet du port d'entrée/sortie et retourne le résultat dans l'adresse ES:[DI] et incrémente/décrémente le registre DI de 1 en fonction de l'état du drapeau de direction.	INTEL 80286+
INSD	Cette instruction permet de demander un double mot du port d'entrée/sortie et retourne le résultat dans l'adresse ES:[DI] et incrémente/décrémente le registre DI de 4 en fonction de l'état du drapeau de direction.	INTEL 80386+
INSW	Cette instruction permet de demander un mot du port d'entrée/sortie et retourne le résultat dans l'adresse ES:[DI] et incrémente/décrémente le registre DI de 2 en fonction de l'état du drapeau de direction.	INTEL 80286+
INT	Cette instruction permet d'exécuter l'interruption avec le numéro spécifié.	INTEL 8086+
INTO	Cette instruction permet d'exécuter l'interruption numéro 4 si le drapeau de débordement (OF) est fixé sur 1	INTEL 8086+
INVD	Cette instruction permet de désactiver et de vider le tampon interne du micro-processeur.	INTEL 80486+
INVLPG	Cette instruction permet d'invalider les transferts du *TLB* (*Translation*	INTEL 80486+

	Lookaside Buffer) du micro-processeur	
INVLPGA	Cette instruction permet d'invalider la cartographie *TLB* (*Translation Lookaside Buffer*) pour une page virtual et un *ASID* spécifié.	x86-64+
IRET	Cette instruction permet d'effectuer un retour précédemment provoquer par une interruption.	INTEL 8086+
IRETD	Cette instruction permet d'effectuer un retour 32-bits précédemment provoquer par une interruption.	INTEL 80386+
IRETQ	Cette instruction permet d'effectuer un retour 64-bits précédemment provoquer par une interruption.	x86-64+
Jump if	Ces instructions permettent d'effectuer d'effectuer un branchement conditionnel à emplacement mémoire spécifié.	INTEL 8086+
JMP	Cette instruction permet d'effectuer un branchement à un emplacement mémoire spécifié.	INTEL 8086+
LAHF	Cette instruction permet de transférer les bits d'indicateurs du registre d'état vers le registre *AH*.	INTEL 8086+
LAR	Cette instruction permet de charger le registre des indicateurs d'un descripteur.	INTEL 80286+
LDDQU	Cette instruction permet de copier un quadruple mot d'une opérande source vers une opérande destination.	INTEL Pentium 4 (SSE3)+
LDMXCSR	Cette instruction permet d'effectuer le chargement du mot de contrôle et d'état du flux d'extension SIMD d'une opérande mémoire 32 bits.	INTEL Pentium III (KNI/MMX2)+
LDS	Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment DS et dans un registre d'offset spécifié.	INTEL 8086+
LEA	Cette instruction permet de copier l'offset d'une adresse de mémoire contenu dans un registre spécifié.	INTEL 8086+
LEAVE	Cette instruction permet de libérer une zone de mémoire attribué par l'instruction «*ENTER*» lorsqu'on utilise des procédures dans des langages de programmation de haut niveau.	INTEL 80286+
LES	Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment ES et dans un registre d'offset spécifié.	INTEL 8086+
LFENCE	Cette instruction permet d'agir comme une barrière pour forcer une priorité en mémoire (sérialisation) entre les instructions précédant le *LFENCE* et les instructions de chargement suivant le *LFENCE*.	INTEL Pentium 4 (SSE2)+
LFS	Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment FS et dans un registre d'offset spécifié.	INTEL 80386+
LGDT	Cette instruction permet de charger un descripteur de tables globale.	INTEL 80286+
LGS	Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment GS et dans un registre d'offset spécifié.	INTEL 80386+
LIDT	Cette instruction permet de charger un descripteur de tables d'interruption.	INTEL 80286+
LLDT	Cette instruction permet de charger un descripteur de tables local.	INTEL 80286+
LMSW	Cette instruction permet de copier 4 des bits d'une opérande vers les 4 bits de registre de contrôle *CR0*.	INTEL 80286+
LOADALL	Cette instruction permet d'effectuer le chargement de tous les registres de descripteur de cache.	INTEL 80286 ou 80386
LOCK	Cette instruction est utilisé comme préfixe avec d'autres instructions pour amener le microprocesseur à émettre un signal de vérouillage (*Lock*) sur le bus lors du traitement de l'instruction suivante.	INTEL 8086+
LODS	Cette instruction permet de copier un élément de l'adresse DS:SI dans le registre accumulateur et incrémente/décrémente le registre SI en fonction de	INTEL 8086+

	la taille de l'opérande source et de l'état du drapeau de direction.	
LODSB	Cette instruction permet de copier un élément de l'adresse DS:SI dans le registre accumulateur et incrémente/décrémente le registre SI de 1 en fonction de l'état du drapeau de direction.	INTEL 8086+
LODSD	Cette instruction permet de copier un élément de l'adresse DS:SI dans le registre accumulateur et incrémente/décrémente le registre SI de 4 en fonction de l'état du drapeau de direction.	INTEL 80386+
LODSQ	Cette instruction permet de copier un élément de l'adresse DS:(R)SI dans le registre accumulateur et incrémente/décrémente le registre (R)SI de 8 en fonction de l'état du drapeau de direction.	x86-64+
LODSW	Cette instruction permet de copier un élément de l'adresse DS:SI dans le registre accumulateur et incrémente/décrémente le registre SI de 2 en fonction de l'état du drapeau de direction.	INTEL 8086+
LOOP	Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0.	INTEL 8086+
LOOPD	Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0.	INTEL 80386+
LOOPE	Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0 et si le drapeau ZF vaut 1.	INTEL 8086+
LOOPED	Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0 et si le drapeau ZF vaut 1.	INTEL 80386+
LOOPNE	Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0 et si le drapeau ZF vaut 0.	INTEL 8086+
LOOPNED	Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0 et si le drapeau ZF vaut 0.	INTEL 80386+
LOOPNZ	Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0 et si le drapeau ZF vaut 0.	INTEL 8086+
LOOPNZD	Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0 et si le drapeau ZF vaut 0.	INTEL 80386+
LOOPZ	Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0 et si le drapeau ZF vaut 1.	INTEL 8086+
LOOPZD	Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0 et si le drapeau ZF vaut 1.	INTEL 80386+
LSL	Cette instruction permet de charger la limite de segment d'un descripteur de segment spécifié avec l'opérande source dans l'opérande de destination et fixe le drapeau *ZF* du registre *EFLAGS*.	INTEL 80286+
LSS	Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment *SS* (Segment de pile) et dans un registre d'offset spécifié.	INTEL 80386+
LTR	Cette instruction permet de charger l'opérande source dans le champ du sélecteur de segment du registre de tâche.	INTEL 80286+
LZCNT	Cette instruction permet de compter le nombre de bits à 0 dans un registre 16, 32 ou 64 bits contenu dans l'opérande source.	AMD K10 (SSE4a)+

MASKMOVDQU	Cette instruction permet d'entreposer les octets sélectionnés par l'opérande source dans une emplacement mémoire de 128 bits.	INTEL Pentium 4 (SSE2)+
MASKMOVQ	Cette instruction permet d'entreposer les octets sélectionner de l'opérande source dans un emplacement mémoire de 64 bits.	Pentium III (KNI/MMX2)+
MAXPS	Cette instruction permet de retourner la valeur maximal de chacune des paires de valeur entre l'opérande source et l'opérande de destination.	Pentium III (KNI/MMX2)+
MAXSS	Cette instruction permet de retourner la valeur maximal entre l'opérande source et destination.	Pentium III (KNI/MMX2)+
MFENCE	Cette instruction permet d'agir comme une barrière pour forcer une priorité en mémoire (sérialisation) entre les instructions précédant le *MFENCE* et les instructions de chargement et d'entreposage précédent le *MFENCE*.	INTEL Pentium 4 (SSE2)+
MINPS	Cette instruction permet de retourner la valeur minimal de chacune des paires de valeur entre l'opérande source et l'opérande de destination.	Pentium III (KNI/MMX2)+
MINSS	Cette instruction permet de retourner la valeur minimal entre l'opérande source et destination.	Pentium III (KNI/MMX2)+
MONITOR	Cette instruction permet d'indiquer au microprocesseur quel rangé d'adresse est à serveiller par l'instruction *STORE*.	INTEL Pentium 4+
MOV	Cette instruction permet de copier opérande source dans une opérande destinataire.	INTEL 8086+
MOVAPS	Cette instruction permet de copier le contenu de 4 paquets alignés de valeurs réel de simple précision (4 x 32 bits).	Pentium III (KNI/MMX2)+
MOVD	Cette instruction permet de copier l'opérande dans un registre *XMM* ou vice-verça.	INTEL Pentium MMX+
MOVHLPS	Cette instruction permet de copier le contenu du haut d'un paquet de valeurs réel de simple précision dans sa partie basse.	Pentium III (KNI/MMX2)+
MOVHPS	Cette instruction permet de copier le contenu du haut de deux paquets de valeurs réel de simple précision dans une opérande de destination.	Pentium III (KNI/MMX2)+
MOVLHPS	Cette instruction permet de copier le contenu du bas d'un paquet de valeurs réel de simple précision dans sa partie haute.	Pentium III (KNI/MMX2)+
MOVLPS	Cette instruction permet de copier le contenu du bas de deux paquets de valeurs réel de simple précision dans une opérande de destination.	Pentium III (KNI/MMX2)+
MOVNTPS	Cette instruction permet de copier le contenu de 4 paquets alignés de valeurs réel de simple précision sans utiliser la méthode temporel pour minimiser la pollution du cache.	Pentium III (KNI/MMX2)+
MOVMSKPD	Cette instruction permet de copier les bits de signes de deux paquets de valeurs réels de double précision d'un registre *XMM* dans les 2 bits les plus bas d'un registre 32 bits. Les autres bits du registres 32 bits sont fixés à 0.	INTEL Pentium 4 (SSE2)+
MOVMSKPS	Cette instruction permet de copier les bits de signes de quatre paquets de valeurs réels de simple précision d'un registre *XMM* dans les 4 bits les plus bas d'un registre 32 bits. Les autres bits du registres 32 bits sont fixés à 0.	INTEL Pentium 3 (SSE)+
MOVNTI	Cette instruction permet de copier une valeur 32 ou 64 bits dans un emplacement mémoire afin de minimiser la pollution du cache dans un processus léger.	INTEL Pentium 4 (SSE2)+
MOVNTQ	Cette instruction permet de copier une valeur 64 bits sans utiliser la méthode temporel pour minimiser la pollution du cache.	Pentium III (KNI/MMX2)+
MOVQ	Cette instruction permet de copier un quadruple mot d'une opérande source vers une opérande destination dans le cas des registres *XMM*.	INTEL Pentium MMX+
MOVS	Cette instruction permet de copier un élément de l'adresse DS:SI dans l'adresse ES:DI et incrémente/décrémente les registres DI et SI en fonction de la taille de l'opérande source et de l'état du drapeau de direction.	INTEL 8086+
MOVSB	Cette instruction permet de copier un octet de l'adresse DS:SI dans l'adresse	INTEL 8086+

	ES:DI et incrémente/décrémente les registres DI et SI de 1 en fonction de l'état du drapeau de direction.	
MOVSD	Cette instruction permet de copier un double mot de l'adresse DS:SI dans l'adresse ES:DI et incrémente/décrémente les registres DI et SI de 4 en fonction de l'état du drapeau de direction.	INTEL 80386+
MOVSQ	Cette instruction permet de copier un quadruple mot de l'adresse DS:(R)SI dans l'adresse ES:(R)DI et incrémente/décrémente les registres (R)DI et (R)SI de 8 en fonction de l'état du drapeau de direction.	x86-64+
MOVSS	Cette instruction permet de copier une valeur réel de simple précision d'une opérande source vers une opérande de destination.	Pentium III (KNI/MMX2)+
MOVSW	Cette instruction permet de copier un mot de l'adresse DS:SI dans l'adresse ES:DI et incrémente/décrémente les registres DI et SI de 2 en fonction de l'état du drapeau de direction.	INTEL 8086+
MOVSX	Cette instruction permet de copier un registre de taille inférieur dans un registre de plus grande taille en remplissant les bits supplémentaires par des 1.	INTEL 80386+
MOVSXD	Cette instruction permet de copier un registre de taille inférieur dans un registre 64 bits en remplissant les bits supplémentaires par des 1.	x86-64+
MOVUPS	Cette instruction permet de copier le contenu de 4 paquets désalignés de valeurs réel de simple précision (4 x 32 bits).	Pentium III (KNI/MMX2)+
MOVZX	Cette instruction permet de copier un registre de taille inférieur dans un registre de plus grande taille en remplissant les bits supplémentaires par des 0.	INTEL 80386+
MUL	Cette instruction permet d'effectuer une multiplication non-signée (nombre naturel).	INTEL 8086+
MULPS	Cette instruction permet d'effectuer la multiplication de chacune des paires de valeur de l'opérande source et l'opérande de destination.	Pentium III (KNI/MMX2)+
MULSS	Cette instruction permet d'effectuer la multiplication scalaire de l'opérande source et l'opérande de destination.	Pentium III (KNI/MMX2)+
MWAIT	Cette instruction permet d'indiquer au microprocesseur que l'état de l'alimentation de la ligne de cache est en attente d'écriture dans la plage d'adresse, mettant fin à la plupart des activités dans le noyau en le faisant.	INTEL Pentium 4+
NEG	Cette instruction permet d'effectuer le complément à 2 d'une opérande.	INTEL 8086+
NOP	Cette instruction ne fait rien.	INTEL 8086+
NOT	Cette instruction permet d'inverser la valeur de chacun des bits d'une opérande.	INTEL 8086+
OIO	Cette instruction permet de provoquer l'exécution d'un code indéfinie.	Cyrix Cx6x86/AMD Am5k86
OR	Cette instruction permet d'effectuer un *OU BINAIRE* sur les 2 opérandes spécifiés.	INTEL 8086+
ORPS	Cette instruction permet d'effectuer un *OU BINAIRE* de 128 bits sur les 2 opérandes spécifiés.	Pentium III (KNI/MMX2)+
OUT	Cette instruction permet d'envoyer un octet, un mot ou un double mot sur le port d'entrée/sortie.	INTEL 8086+
OUTS	Cette instruction permet d'envoyer un octet, un mot ou un double mot contenu dans l'adresse DS:[SI] du port d'entrée/sortie et incrémente/décrémente le registre SI en fonction de la taille de l'opérande cible et de l'état du drapeau de direction.	INTEL 80286+
OUTSB	Cette instruction permet d'envoyer un octet contenu dans l'adresse DS:[SI] du port d'entrée/sortie et incrémente/décrémente le registre SI de 1 en fonction de l'état du drapeau de direction.	INTEL 80286+
OUTSD	Cette instruction permet d'envoyer un double mot contenu dans l'adresse DS:[SI] du port d'entrée/sortie et incrémente/décrémente le registre SI de 4 en fonction de l'état du drapeau de direction.	INTEL 80386+

OUTSW	Cette instruction permet d'envoyer un mot contenu dans l'adresse DS:[SI] du port d'entrée/sortie et incrémente/décrémente le registre SI de 2 en fonction de l'état du drapeau de direction.	INTEL 80286+
PACKSSDW	Cette instruction permet de compacté 8 paquets de double mots en mots.	INTEL Pentium Pro+
PACKSSWB	Cette instruction permet de compacté 8 paquets d'entier en octets.	INTEL Pentium Pro+
PACKUSWB	Cette instruction permet de compacté 8 paquets de mots en octets.	INTEL Pentium Pro+
PADDB	Cette instruction permet d'effectuer une addition scalaire sur un paquet de 8 octets.	INTEL Pentium Pro+
PADDD	Cette instruction permet d'effectuer une addition scalaire sur un paquet de 2 double mots.	INTEL Pentium Pro+
PADDSB	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 8 octets signés.	INTEL Pentium Pro+
PADDSIW	Cette instruction permet d'effectuer l'addition du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans le registre *MMX*.	Cyrix 6x86MX (EMMX)+
PADDSW	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 4 entiers.	INTEL Pentium Pro+
PADDUSB	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 8 octets.	INTEL Pentium Pro+
PADDUSW	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 4 mots.	INTEL Pentium Pro+
PADDW	Cette instruction permet d'effectuer une addition scalaire sur un paquet de 4 mots.	INTEL Pentium Pro+
PAND	Cette instruction permet d'effectuer un «*ET BINAIRE*» d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.	INTEL Pentium MMX+
PANDN	Cette instruction permet d'effectuer un «*ET BINAIRE*» et une «*NEGATION*» de chacun des bits d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.	INTEL Pentium Pro+
PAUSE	Cette instruction permet d'améliorer les performances des boucles de «*SPIN*», en fournissant une indication pour le microprocesseur que le code courant est dans une boucle en «*SPIN*».	INTEL Pentium 4 (SSE2)+
PAVEB	Cette instruction permet d'effectuer le calcul de la moyenne des paquets spécifiés.	Cyrix 6x86MX (EMMX)+
PAVGUSB	Cette instruction permet d'effectuer le calcul de la moyenne de paquets de 8 octets spécifiés.	AMD 3DNow!
PCMPEQB	Cette instruction permet d'effectuer une comparaison sur un paquet d'octets et fixe la valeur de chacun d'eux à FFh s'ils sont égales sinon à 00h.	INTEL Pentium Pro+
PCMPEQD	Cette instruction permet d'effectuer une comparaison sur un paquet de double mots et fixe la valeur de chacun d'eux à FFFFFFFFh s'ils sont égales sinon à 00000000h.	INTEL Pentium Pro+
PCMPEQW	Cette instruction permet d'effectuer une comparaison sur un paquet de mots et fixe la valeur de chacun d'eux à FFFFh s'ils sont égales sinon à 0000h.	INTEL Pentium Pro+
PCMPGTB	Cette instruction permet d'effectuer une comparaison sur un paquet d'octets et fixe la valeur de chacun d'eux à FFh si l'opérande destinataire est supérieur à l'opérande source sinon à 00h.	INTEL Pentium Pro+
PCMPGTD	Cette instruction permet d'effectuer une comparaison sur un paquet de double mots et fixe la valeur de chacun d'eux à FFFFFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 00000000h.	INTEL Pentium Pro+
PCMPGTW	Cette instruction permet d'effectuer une comparaison sur un paquet de mots et fixe la valeur de chacun d'eux à FFFFh si l'opérande destinataire est	INTEL Pentium Pro+

	supérieur à l'opérande source sinon à 0000h.	
PDISTIB	Cette instruction permet d'effectuer le calcul de la distance entre des octets de deux opérandes, le résultat de l'addition d'octet de l'opérande de destination et la saturation du résultat.	Cyrix 6x86MX (EMMX)+
PEXTRW	Cette instruction permet de copier le mot de l'opérande source spécifié par une compteur d'opérande dans l'opérande de destination.	INTEL Pentium III (KNI/MMX2)+
PF2ID	Cette instruction permet de convertir un double mots d'un format réel à un format d'entier 32 bits.	AMD 3DNow!
PF2IW	Cette instruction permet de convertir un double mots d'un format réel à un format d'entier 16 bits.	AMD 3DNow!
PFACC	Cette instruction permet d'effectuer l'accumulation de double mots.	AMD 3DNow!
PFADD	Cette instruction permet d'effectuer l'addition de double mots.	AMD 3DNow!
PFCMPEQ	Cette instruction permet d'effectuer la comparaison d'égalité d'un paquet de double mots contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFCMPGE	Cette instruction permet d'effectuer la comparaison d'égalité ou de supériorité d'un paquet de double mots contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFCMPGT	Cette instruction permet d'effectuer la comparaison de supériorité d'un paquet de double mots contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFMAX	Cette instruction permet de retourner la valeur maximal de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFMIN	Cette instruction permet de retourner la valeur minimal de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFMUL	Cette instruction permet d'effectuer la multiplication de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFNACC	Cette instruction permet d'effectuer l'accumulation négative de double mots.	AMD 3DNow!
PFPNACC	Cette instruction permet d'effectuer un mixe d'accumulation positive et négative de double mots.	AMD 3DNow!
PFRCP	Cette instruction permet d'effectuer la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFRCPIT1	Cette instruction permet d'effectuer la première étape d'itération de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFRCPIT2	Cette instruction permet d'effectuer la deuxième étape d'itération de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFRSQIT1	Cette instruction permet d'effectuer la première étape d'itération de la racine carré de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFRSQRT	Cette instruction permet d'effectuer la racine carré approximative de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFSUB	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PFSUBR	Cette instruction permet d'effectuer la soustraction inversé de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	AMD 3DNow!
PI2FD	Cette instruction permet de convertir un paquet de format d'entier 32 bits à double mots d'un format réel.	AMD 3DNow!
PI2FW	Cette instruction permet de convertir un paquet de format d'entier 16 bits à double mots d'un format réel.	AMD 3DNow!

PINSRW	Cette instruction permet d'effectuer une copie d'une opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.	INTEL Pentium III (KNI/MMX2)+
PMACHRIW	Cette instruction permet d'effectuer une multiplication de 2 opérandes sources en utilisant la méthode décrite par *PMULHRW* et accumule le résultat avec la valeur dans un registre de destination en utilisant un arrondissement arithmétique.	Cyrix 6x86MX (EMMX)+
PMADDWD	Cette instruction permet d'effectuer une multiplication de 2 registres MMX contenant dans un paquet de mots d'un emplacement/mémoire.	INTEL Pentium Pro+
PMAGW	Cette instruction permet d'effectuer la comparaison de la valeur absolue d'un paquet de mots de l'opérande source et de l'ensemble des mots de destination et fixe l'opérande de destination avec la valeur de la plus large magnétude.	Cyrix 6x86MX (EMMX)+
PMAXSW	Cette instruction permet de retourner la valeur maximal de chacun des mots des paquets contenu dans deux opérandes.	INTEL Pentium III (KNI/MMX2)+
PMAXUB	Cette instruction permet de retourner la valeur maximal de chacun des octets des paquets contenu dans deux opérandes.	INTEL Pentium III (KNI/MMX2)+
PMINSW	Cette instruction permet de retourner la valeur minimal de chacun des mots des paquets contenu dans deux opérandes.	INTEL Pentium III (KNI/MMX2)+
PMINUB	Cette instruction permet de retourner la valeur minimal de chacun des octets des paquets contenu dans deux opérandes.	INTEL Pentium III (KNI/MMX2)+
PMOVMSKB	Cette instruction permet de copiés les masques de chacun des octets d'un paquet contenu dans une opérande.	INTEL Pentium III (KNI/MMX2)+
PMULHRIW	Cette instruction permet de retourner le résultat d'une matrice de 16x16 bits avec un arrondissement LSB avant de tronquer les 16 bits et place ce résultat dans un registre *MMX*.	Cyrix 6x86MX (EMMX)+
PMULHRW	Cette instruction permet de retourner le résultat d'une matrice de 16x16 bits avec un arrondissement LSB avant de tronquer les 16 bits.	Cyrix 6x86MX (EMMX)+
PMULHUW	Cette instruction permet d'effectuer la multiplication de la partie haute de chacun des entiers des 2 paquets d'opérandes.	INTEL Pentium III (KNI/MMX2)+
PMULHW	Cette instruction permet d'effectuer la multiplication de la partie haute de chacun des mots des 2 paquets d'opérandes.	INTEL Pentium Pro+
PMULLW	Cette instruction permet d'effectuer la multiplication de la partie basse de chacun des mots des 2 paquets d'opérandes.	INTEL Pentium Pro+
PMVGEZB	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination est supérieur à 0 et effectue se traitement pour chacun des 8 octets du paquet.	Cyrix 6x86MX (EMMX)+
PMVLZB	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination est inférieur à 0 et effectue se traitement pour chacun des 8 octets du paquet.	Cyrix 6x86MX (EMMX)+
PMVNZB	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination ne vaut pas 0 et effectue se traitement pour chacun des 8 octets du paquet.	Cyrix 6x86MX (EMMX)+
PMVZB	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination vaut 0 et effectue se traitement pour chacun des 8 octets du paquet.	Cyrix 6x86MX (EMMX)+
POP	Cette instruction permet de désempiler de la pile une mot ou un double mot et la met dans une opérande.	INTEL 8086+
POPA	Cette instruction permet de désempiler de la pile respectivement les registres DI, SI, BP, SP, BX, DX, CX et AX.	INTEL 80286+

POPAD	Cette instruction permet de désempiler de la pile respectivement les registres EDI, ESI, EBP, ESP, EBX, EDX, ECX et EAX.	INTEL 80386+
POPCNT	Cette instruction permet de compter le nombre de bits à 1 que possède une opérande source et place le résultat dans un registre destinataire.	AMD K10 (SSE4a)+
POPF	Cette instruction permet de désempiler de la pile le registre 16 bits de drapeau contenant les indicateurs d'état.	INTEL 8086+
POPFD	Cette instruction permet de désempiler de la pile le registre 32 bits de drapeau contenant les indicateurs d'état.	INTEL 80386+
POPFQ	Cette instruction permet de désempiler de la pile le registre 64 bits de drapeau (RFLAGS) contenant les indicateurs d'état.	x86-64+
POR	Cette instruction permet d'effectuer un «*OU BINAIRE*» d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.	INTEL Pentium MMX+
PREFETCH	Cette instruction permet d'effectuer le chargement de l'entrée d'une séquence d'alignement de mémoire de 64 bits de l'adresse mémoire spécifié dans le cache de données *L1* du microprocesseur.	AMD K6-2+
PREFETCH0	Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T0* du microprocesseur. Alias de *PREFETCHT0*.	INTEL Pentium III (SSE)+
PREFETCH1	Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T1* du microprocesseur. Alias de *PREFETCHT1*.	INTEL Pentium III (SSE)+
PREFETCH2	Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T2* du microprocesseur. Alias de *PREFETCHT2*.	INTEL Pentium III (SSE)+
PREFETCHNTA	Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *NTA* du microprocesseur.	INTEL Pentium III (SSE)+
PREFETCHT0	Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T0* du microprocesseur. Alias de *PREFETCH0*.	INTEL Pentium III (SSE)+
PREFETCHT1	Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T1* du microprocesseur. Alias de *PREFETCH1*.	INTEL Pentium III (SSE)+
PREFETCHT2	Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T2* du microprocesseur. Alias de *PREFETCH2*.	INTEL Pentium III (SSE)+
PREFETCHW	Cette instruction permet d'effectuer le chargement de l'entrée d'une séquence d'alignement de mémoire de 64 bits de l'adresse mémoire spécifié dans le cache de données *L1* du microprocesseur dans un état de modification.	AMD K6-2+
PROC	Cette directive permet de définir une procédure (sous-programme).	
PSLLD	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de double mots.	INTEL Pentium Pro+
PSLLQ	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un quadruple mot.	INTEL Pentium Pro+
PSLLW	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de mots.	INTEL Pentium Pro+
PSRAD	Cette instruction permet d'effectuer le décalage arithmétique de bits vers la droite d'un paquet de double mots.	INTEL Pentium Pro+
PSRAW	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de mots.	INTEL Pentium Pro+
PSRLD	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de double mots.	INTEL Pentium Pro+

PSRLQ	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un quadruple mot.	INTEL Pentium Pro+
PSRLW	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de mots.	INTEL Pentium Pro+
PSUBB	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des octets d'un paquet contenu dans des opérandes de 64 bits.	INTEL Pentium Pro+
PSUBD	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.	INTEL Pentium Pro+
PSUBSB	Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des octets signés d'un paquet contenu dans des opérandes de 64 bits.	INTEL Pentium Pro+
PSUBSIW	Cette instruction permet d'effectuer la soustraction du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans le registre *MMX*.	Cyrix 6x86MX (EMMX)+
PSUBSW	Cette instruction permet d'effectuer la soustraction du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans l'opérande destinataire.	INTEL Pentium Pro+
PSUBUSB	Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des octets d'un paquet contenu dans des opérandes de 64 bits.	INTEL Pentium Pro+
PSUBUSW	Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des mots d'un paquet contenu dans des opérandes de destinations.	INTEL Pentium Pro+
PSUBW	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des mots d'un paquet contenu dans des opérandes de 64 bits.	INTEL Pentium Pro+
PSWAPD	Cette instruction permet d'effectuer l'échange de la partie basse de 32 bits avec la partie haute de 32 bits dans des opérandes source de 64 bits et met le résultat dans une opérande 64 bits destinataire.	3DNow!
PUBLIC	Cette directive permet de rendre publique une expression pour des liaisons externe.	
PUNPCKHBW	Cette instruction permet de décompacter des octets en mots dans le haut d'un paquets de 64 bits.	INTEL Pentium Pro+
PUNPCKHDQ	Cette instruction permet de décompacter des doubles mots en quadruples mots dans le haut d'un paquets de 64 bits.	INTEL Pentium Pro+
PUNPCKHWD	Cette instruction permet de décompacter des mots en double mots dans le haut d'un paquets de 64 bits.	INTEL Pentium Pro+
PUNPCKLBW	Cette instruction permet de décompacter des octets en mots dans le bas d'un paquets de 64 bits.	INTEL Pentium Pro+
PUNPCKLDQ	Cette instruction permet de décompacter des doubles mots en quadruples mots dans le bas d'un paquets de 64 bits.	INTEL Pentium Pro+
PUNPCKLWD	Cette instruction permet de décompacter des mots en double mots dans le bas d'un paquets de 64 bits.	INTEL Pentium Pro+
PUSH	Cette instruction permet d'empiler une mot ou un double mot dans la pile.	INTEL 8086+
PUSHA	Cette instruction permet d'empiler respectivement les registres DI, SI, BP, SP, BX, DX, CX et AX dans la pile.	INTEL 80286+
PUSHAD	Cette instruction permet d'empiler respectivement les registres EDI, ESI, EBP, ESP, EBX, EDX, ECX et EAX dans la pile.	INTEL 80386+
PUSHF	Cette instruction permet d'empiler respectivement le registre 16 bits de drapeau des indicateurs d'état dans la pile.	INTEL 8086+
PUSHFD	Cette instruction permet d'empiler respectivement le registre 32 bits de drapeau des indicateurs d'état dans la pile.	INTEL 80386+

PUSHFQ	Cette instruction permet d'empiler respectivement le registre 64 bits de drapeau (RFLAGS) des indicateurs d'état dans la pile.	x86-64+
PXOR	Cette instruction permet d'effectuer un ou exclusif binaire d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.	INTEL Pentium MMX+
RCL	Cette instruction permet d'effectuer une rotation des bits vers la gauche en réinsérant le bit dans l'indicateur de retenue (*CF*) ainsi que dans le bit le plus à droite libéré.	INTEL 8088+
RCPPS	Cette instruction permet d'effectuer le calcul de la réciproque d'un paquet de valeurs de réel de simple précision.	INTEL Pentium III (KNI/MMX2)+
RCPSS	Cette instruction permet d'effectuer le calcul de la réciproque d'une valeur de réel de simple précision.	INTEL Pentium III (KNI/MMX2)+
RCR	Cette instruction permet d'effectuer une rotation des bits vers la droite en réinsérant le bit dans l'indicateur de retenue (*CF*) ainsi que dans le bit le plus à gauche libéré.	INTEL 8088+
RDMSR	Cette instruction permet de charger le contenu du modèle de registre 64-bits (*MSR*) indiquer par le registre ECX dans le couple de registre *EDX*:*EAX*.	INTEL Pentium+
RDPMC	Cette instruction permet d'effectuer la lecture du compteur du moniteur de performance.	INTEL Pentium Pro+
RDSHR	Cette instruction permet d'effectuer la lecture du registre de pointeur vers l'entête *SMM*.	Cyrix CX6x86MX+
RDTSC	Cette instruction permet de charger la valeur courante du compteur de temps du microprocesseur dans le couple de registres *EDX*:*EAX*.	INTEL Pentium+
RDTSCP	Cette instruction permet de charger la valeur courante du compteur de temps du microprocesseur dans le couple de registres *EDX*:*EAX* et la valeur du *TSC_AUX* dans le registre *ECX*.	x86-64+
REP	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions tant que CX ne vaut pas 0.	INTEL 8086+
REPE	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 0.	INTEL 8086+
REPNE	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 1.	INTEL 8086+
REPNZ	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 1.	INTEL 8086+
REPZ	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 0.	INTEL 8086+
RES3	Cette instruction permet d'effectuer le chargement de tous les registres de descripteur de cache pour les 386.	AMD Am386SXLV, Am386DXLV
RES4	Cette instruction permet d'effectuer le chargement de tous les registres de descripteur de cache pour les 486.	AMD Am486SXLV, Am486DXLV
RET	Cette instruction permet de quitter une procédure.	INTEL 8086+
RETF	Cette instruction permet de quitter une procédure ayant lieu avec un appel long (FAR CALL).	INTEL 8086+
RETN	Cette instruction permet de quitter une procédure ayant lieu avec un appel court (NEAR CALL).	INTEL 8086+
ROL	Cette instruction permet d'effectuer une rotation des bits vers la gauche en réinsérant le bit dans le bit le plus à droite libéré.	INTEL 8088+

ROR	Cette instruction permet d'effectuer une rotation des bits vers la droite en réinsérant le bit dans le bit le plus à gauche libéré.	INTEL 8088+
RSDC	Cette instruction permet d'effectuer le restauration de la structure des registres et des descripteurs.	Quelques 486
RSLDT	Cette instruction permet d'effectuer la restauration des LDTR et des descripteurs.	Quelques 486
RSM	Cette instruction permet de retourner le contrôle du programme du mode de gestion système (*SMM*) pour le programme d'application ou la procédure du système d'exploitation ayant été interrompu lorsque le microprocesseur à recu un signal *SSM*.	INTEL 80486+
RSQRTPS	Cette instruction permet de calculer la réciproque approximative de la racine carré d'un paquet de simple précision.	INTEL Pentium III (KNI/MMX2)+
RSQRTSS	Cette instruction permet de calculer la réciproque approximative de la racine carré d'un scalaire de simple précision.	INTEL Pentium III (KNI/MMX2)+
RSTS	Cette instruction permet d'effectuer le restauration des TR et des descripteurs.	Quelques 486
SAHF	Cette instruction permet de copier les bits du registre AH dans l'octet de poids faible dans le registre des drapeaux (les indicateurs d'état).	INTEL 8088+
SAL	Cette instruction permet d'effectuer une rotation des bits vers la gauche en réinsérant le bit dans l'indicateur de retenue (*CF*).	INTEL 8088+
SAR	Cette instruction permet d'effectuer une rotation des bits vers la droite en réinsérant le bit dans l'indicateur de retenue (*CF*).	INTEL 8088+
SBB	Cette instruction permet de soustraire avec l'indicateur de retenue (*CF*) une valeur à une opérande.	INTEL 8088+
SCAS	Cette instruction permet de comparer un octet, un mot ou un double mot avec la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI en fonction de la taille de l'opérande cible et de l'état du drapeau de direction.	INTEL 8086+
SCASB	Cette instruction permet de comparer le registre AL avec la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 1 en fonction de l'état du drapeau de direction.	INTEL 8086+
SCASD	Cette instruction permet de comparer le registre EAX avec la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 4 en fonction de l'état du drapeau de direction.	INTEL 80386+
SCASW	Cette instruction permet de comparer le registre AX avec la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 2 en fonction de l'état du drapeau de direction.	INTEL 8086+
SEG	Cette directive permet de demander la partie segment de l'adresse d'une opérande.	INTEL 8086+
SET	Ces instructions permettent de fixer la valeur d'une opérande 1 si la condition d'indicateur d'état du registre 32 bits de drapeau est vrai sinon fixe la valeur à 0.	INTEL 80386+
SETALC	Cette instruction permet de copier la valeur du drapeau de retenue dans le registre AL en la multipliant par 0FFh.	INTEL 8086+
SFENCE	Cette instruction permet d'agir comme une barrière pour forcer une priorité en mémoire (sérialisation) entre les instructions précédant emmagasiner du *SFENCE* et les instructions suivant le *SFENCE*.	Pentium III (SSE)+
SGDT	Cette instruction permet d'entreposer le sélecteur de segment dans le registre *GDTR* (registre de table global de descripteur) dans l'opérande de destination.	INTEL 80386+
SHL	Cette instruction permet d'effectuer une rotation des bits vers la gauche en réinsérant le bit dans l'indicateur de retenue (*CF*).	INTEL 8088+
SHLD	Cette instruction permet d'effectuer une rotation des bits d'un double mot vers la gauche en réinsérant le bit dans l'indicateur de retenue (*CF*).	INTEL 80386+

SHR	Cette instruction permet d'effectuer une rotation des bits vers la droite en réinsérant le bit dans l'indicateur de retenue (*CF*).	INTEL 8088+
SHRD	Cette instruction permet d'effectuer une rotation des bits d'un double mot vers la droite en réinsérant le bit dans l'indicateur de retenue (*CF*).	INTEL 80386+
SHUFPS	Cette instruction permet de copier 4 paquets de valeurs de format réel de simple précision dans une opérande destinataire dans la partie basse d'un quadruple mot de celle-ci et copie 2 des 4 paquets de valeurs de format réel de simple précision dans l'opérande source dans la partie haute d'un quadruple mot de l'opérande destinataire.	INTEL Pentium III (KNI/MMX2)+
SIDT	Cette instruction permet d'entreposer le registre de descripteur de table d'interruption (*IDTR*) dans l'opérande de destination.	INTEL 80386+
SKINIT	Cette instruction permet de réinitialiser de façon sécuritaire le microprocesseur et de démarrer un logiciel de confiance comme un *VMM*.	AMD-V
SLDT	Cette instruction permet d'entreposer le sélecteur de segment dans le registre *LDTR* (registre de table local de descripteur) dans l'opérande de destination.	INTEL 80386+
SMI	Cette instruction permet de gérer les interruptions systèmes en mode de déboguage.	AMD Am386SXLV,Am386DXLV
SMINT	Cette instruction permet de faire entrée le microprocesseur en mode *SMM* (*System Management Mode*).	P6, Cyrix
SMINTOLD	Cette instruction permet de faire entrée le microprocesseur en mode *SMM* (*System Management Mode*).	Quelques 486
SMSW	Cette instruction permet d'entreposer le mot des états (soit les bits de 0 à 15 du registre de contrôle *CR0*) à partir du registre de tâche (*TR*) dans l'opérande cible.	INTEL 80386+
SQRTPS	Cette instruction permet d'effectuer le calcul de la racine carré d'un paquet de valeur de simple précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de simple précision.	INTEL Pentium III (KNI/MMX2)+
SQRTSS	Cette instruction permet d'effectuer le calcul de la racine carré d'une valeur de simple précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de simple précision.	INTEL Pentium III (KNI/MMX2)+
SS:	Ce préfixe permet d'indiquer à une instruction d'utiliser le Segment de Pile (*Stack Segment*) pour adresse dans l'opérande.	INTEL 8088+
STC	Cette instruction permet de fixer l'indicateur de retenue (*CF*) à la valeur 1.	INTEL 8088+
STD	Cette instruction permet de fixer l'indicateur de direction (*DF*) à la valeur 1.	INTEL 8088+
STGI	Cette instruction permet de fixer les drapeaux global d'interruption (*GIF*). Quand le *GIF* est à 1, les interruptions externes sont réactivés.	AMD-V
STI	Cette instruction permet de fixer l'indicateur d'interruption (*IF*) à la valeur 1.	INTEL 8088+
STMXCSR	Cette instruction permet de copier le *MXCSR* dans un emplacement mémoire de 32 bits.	INTEL Pentium III (KNI/MMX2)+
STOS	Cette instruction permet de copier un octet, un mot ou un double mot dans la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI en fonction de la taille de l'opérande cible et de l'état du drapeau de direction.	INTEL 8086+
STOSB	Cette instruction permet de copier le registre AL dans la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 1 en fonction de l'état du drapeau de direction.	INTEL 8086+
STOSD	Cette instruction permet de copier le registre EAX dans la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 4 en fonction de l'état du drapeau de direction.	INTEL 80386+
STOSQ	Cette instruction permet de copier le registre RAX dans la cellule mémoire à l'adresse ES:[(R)DI] et incrémente/décrémente le registre (R)DI de 8 en fonction de l'état du drapeau de direction.	x86-64+

STOSW	Cette instruction permet de copier le registre AX dans la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 2 en fonction de l'état du drapeau de direction.	INTEL 8086+
STR	Cette instruction permet d'entreposer le sélecteur de segment à partir du registre de tâche (*TR*) à l'opérande cible	INTEL 80286+
SUB	Cette instruction permet de soustraire une valeur à une opérande.	INTEL 8088+
SUBPS	Cette instruction permet d'effectuer une soustraction de 4 paquets de valeurs réels de simple précision d'une opérande source et d'une opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de simple précision.	INTEL Pentium III (KNI/MMX2)+
SUBSS	Cette instruction permet d'effectuer une soustraction de la partie basse d'une valeur réel de simple précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de simple précision.	INTEL Pentium III (KNI/MMX2)+
SVDC	Cette instruction permet de sauvegarder le registre et le descripteur.	Quelques 486
SVLDT	Cette instruction permet de sauvegarder le LDTR et le descripteur.	Quelques 486
SVTS	Cette instruction permet de sauvegarder le TR et le descripteur.	Quelques 486
SWAPGS	Cette instruction permet de fournir une méthode à un logiciel système pour charger un pointeur sur une structure de données système.	x86-64+
SYSCALL	Cette instruction permet d'effectuer le transfert du contrôle d'un point d'entrée fixe au système d'exploitation.	AMD K6-2+
SYSENTER	Cette instruction permet d'effectuer le transférer du contrôle d'un point d'entrée au système d'exploitation.	INTEL Pentium Pro+
SYSEXIT	Cette instruction permet de retourner du système d'exploitation à une application.	INTEL Pentium Pro+
SYSRET	Cette instruction permet de retourner du système d'exploitation à une application.	AMD K6-2+
TEST	Cette instruction permet d'effectuer un «Et binaire» sur une opérande cible sans modifier sa valeur.	INTEL 8088+
TPASCAL	Cette directive permet la configurant du modèle de mémoire simplifié pour écrire des procédures appelable du *TURBO PASCAL*.	INTEL 8088+
UCOMISD	Cette instruction permet d'effectuer une comparaison désordonnée de valeurs réels de double précision dans la partie basse d'un double mot du premier opérande et du deuxième opérande, et fixe les drapeaux ZF, PF et FC dans le registre EFLAGS selon le résultat (non-ordonnée, supérieur à, inférieur ou égal)	INTEL Pentium III+
UCOMISS	Cette instruction permet d'effectuer une comparaison désordonnée de valeurs réels de simple précision dans la partie basse d'un double mot du premier opérande et du deuxième opérande, et fixe les drapeaux ZF, PF et FC dans le registre EFLAGS selon le résultat (non-ordonnée, supérieur à, inférieur ou égal)	INTEL Pentium III+
UD	Cette instruction permet de provoquer l'exécution d'un code indéfinie.	AMD Am5k86 (SSA/5, K5)
UD2	Cette instruction permet de provoquer l'exécution d'un code indéfinie.	INTEL Pentium Pro+
UMOV	Cette instruction permet de copier un opérande source dans une emplacement mémoire principal.	AMD Am386SXLV, Am386DXLV, 486s, IBM 486SLC2
UNPCKLPS	Cette instruction permet d'effectuer un dépaquetage de la partie basse d'un réel de simple précision d'une opérande source et destination et met le résultat dans l'opérande de destination.	INTEL Pentium III+ (KNI/MMX2)
VERR	Cette instruction permet de vérifier si le code ou le segment de données spécifié est en mode lecture à partir du niveau de privilège courant (*CPL*).	INTEL 80286+

VERW	Cette instruction permet de vérifier si le code ou le segment de données spécifié est en mode écriture à partir du niveau de privilège courant (*CPL*).	INTEL 80286+
VMLOAD	Cette instruction permet d'effectuer le chargement d'un sous-ensemble d'état de microprocesseur dans un *VMCB* spécifié par une adresse physique contenu dans le registre *RAX*.	AMD-V
VMMCALL	Cette instruction permet de fournir un mécanisme invité pour communiquer explicitement avec le *VMM* en générant un *#VMEXIT*.	AMD-V
VMRUN	Cette instruction permet de lancer l'exécution d'un flux d'instructions invité.	AMD-V
VMSAVE	Cette instruction permet d'entreposer un sous-ensemble d'état du microprocesseur dans un *VMCB* spécifié par une adresse physique contenu dans le registre *RAX*.	AMD-V
WAIT	Cette instruction permet de faire passer le microprocesseur en mode d'attente jusqu'à ce que la ligne de teste sur la carte mère s'active.	INTEL 8088+
WBINVD	Cette instruction permet de désactiver et de vider le tampon interne du micro-processeur.	INTEL 80486+
WRMSR	Cette instruction écrit les valeurs contenues dans le *MSR* (*Model-Specific Register*) en fonction du registre d'index ECX dans la paire des registres EDX:EAX.	INTEL Pentium+
WRSHR	Cette instruction écrit les valeurs contenues dans l'opérande source dans l'entête *SMM*.	Cyrix Cx6x86MX
XADD	Cette instruction permet d'échanger le premier opérande avec le deuxième opérande, et ensuite effectue la somme des valeurs dans le premier opérande.	INTEL 80486+
XBTS	Cette instruction permet de prendre des bits d'un opérande source et de les mettres dans une opérande destinataire.	INTEL 80386
XCHG	Cette instruction permet d'échanger la valeur de deux opérandes.	INTEL 8088+
XLAT	Cette instruction permet de remplacer le contenu du registre AL par un octet de la «*tablesource*».	INTEL 8088+
XLATB	Cette instruction permet de remplacer le contenu du registre AL par un octet de la «*tablesource*» sans opérande.	INTEL 8088+
XOR	Cette instruction permet d'effectuer un *OU exclusif BINAIRE* sur les 2 opérandes spécifiés.	INTEL 8086+
XORPS	Cette instruction permet d'effectuer un ou exclusif binaire de 4 paquets de valeurs réel de simple précision dans une opérande source et destination et de mettre le résultat dans l'opérande de destination.	INTEL Pentium III+ (KNI/MMX2)

Assembleur 80x86 — INTEL 8088+

AAA

Ascii Adjust after Addition

Syntaxe

AAA

Description

Cette instruction permet d'adapter le résultat obtenu par l'addition de 2 valeurs en format *DCB*. Puisque le processeur additionne ces valeurs comme s'il s'agissait de nombres normaux, des débordements peuvent se produire dans les sommes. Afin de remédier à ce problème, la commande *AAA* offre de convertir le résultat de l'addition de 2 valeurs de format *DCB* non compactées en une valeur de format *DCB* non compactée. Elle cherche la somme de l'addition dans le registre *AL*. Pour que cette instruction réagisse correctement, il faut qu'elle soit lancée directement après l'addition de 2 valeurs de format DCB, étant donné que la valeur des indicateurs du micro-processeur revêt une importance particulière lors de l'exécution de cette commande.

Algorithme

```
* Vaut 1 si c'est vrai
retenue d'AL ← AL > 0F9h
SI ( ( AL ∩ 0Fh > 9 ) U ( AF = 1 ) ALORS
   AL ← ( AL + 6 ) ∩ 0Fh
   AH ← AH + 1 + retenue d'AL
   Drapeau AF ← 1
   Drapeau CF ← 1
SINON
   AL ← 0Fh
   Drapeau AF ← 0
   Drapeau CF ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
AAA	37h	Cette instruction permet de créer un nombre dépaqueté BCD (invalide en mode 64-bits)

Cycles d'horloge

Instruction	Opcode	8086	8088	80186	80286	80386	i486	Pentium	Cx486 SLC	Cx486DX	IBM 486BL3X	UMC U5S
AAA	37h	4	4	8	3	4	3	3	4	4	4	1

Affectations des registres de drapeaux

Drapeau	Position	Description
CF	0	Fixé à 1 si la décimal est retenue, sinon il est fixé à 0.
PF	2	Indéfini
AF	4	Fixé à 1 si la décimale est retenue, sinon il est fixé à 0.
ZF	6	Indéfini
SF	7	Indéfini
TF	8	Non affecté
IF	9	Non affecté
DF	10	Non affecté
OF	11	Indéfini
IOPL	12 et 13	Non affecté
NT	14	Non affecté
RF	16	Non affecté
WM	17	Non affecté
AC	18	Non affecté
VIF	19	Non affecté
VIP	20	Non affecté
ID	21	Non affecté

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#**UD**(Opcode invalide)			X	Cette instruction est exécutée en mode 64-bits

Exemple

Cet exemple permet d'effectuer une addition BCD:

```
MOV AX,0005h
MOV CL,06h
ADD AL,CL
AAA
```

Ce programme retournera donc la valeur 0101h dans le registre AX.

Voir également

Instruction assembleur 80x86 - Instruction AAD
Instruction assembleur 80x86 - Instruction AAM
Instruction assembleur 80x86 - Instruction AAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 801
Le grand livre pour programmer en Assembleur sur PC, Holger Schakel, 1995, ISBN: 2-7429-0442-5, page 144.
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 53.

Assembleur 80x86	**AAD**
INTEL 8088+	*Ascii Adjust ax before Division*

Syntaxe

AAD **AAD** *immédiat 8*

Paramètres

Nom	Description
immédiat 8	Ce paramètre permet une valeur statique de 8 bits

Description

Grâce à cette instruction, on peut convertir une valeur de format *DCB* non compactée se trouvant dans le registre *AX* (un chiffre dans le registre *AH* tandis que l'autre est dans le registre *AL*) en un nombre binaire étant placée dans le registre *AL*. Pour la division, on utilise l'instruction *IDIV* en veillant auparavant à ce que le registre *AH* soit vide.

Algorithme

AL ← *AH* x *immédiat 8 bits* + *AL* *AH* ← 00h

Mnémonique

Instruction	**Opcode**	**Description**
AAD	D5h 0Ah	Ajuste 2 chiffres BCD dans les registres AL et AH (invalide en mode 64-bits)
aucun	D5h *ib*	Ajuste 2 chiffres BCD dans les registres AL et AH (invalide en mode 64-bits)

Cycles d'horloge

Instruction	Opcode	8086	8088	80186	80286	80386	i486	Pentium	Cx486SLC	Cx486DX	IBM 486BL 3X	UMC U5S
AAD	D5h 0Ah	60	60	15	14	19	14	10	4	4	15	11

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction est exécutée en mode 64-bits

Voir également

Instruction assembleur 80x86 - Instruction AAA
Instruction assembleur 80x86 - Instruction AAM
Instruction assembleur 80x86 - Instruction AAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 801
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 54.

Assembleur 80x86	**AAM**
INTEL 8088+	*Ascii adjust AX After Multiplication*

Syntaxe

AAM **AAM** *immédiat 8*

Paramètres

Nom	Description
immédiat 8	Ce paramètre permet une valeur statique de 8 bits

Description

Cette instruction offre la possibilité de convertir le produit de la multiplication de 2 valeurs de format *DCB* en un format *DCB*. Étant donné que le microprocesseur multiplie ces valeurs comme s'il s'agissait de nombres normaux, des débordements peuvent éventuellement se produire dans les résultats. L'instruction *AAM* rectifie cela. Lors de l'exécution de cette commande, le contenu du registre *AL* est divisé en 2 chiffres de format *DCB* nom compactés: le chiffre le plus élevé est placé dans le registre *AH* tandis que le moins élevé est mis dans le registre *AL*.

Algorithme

AH ← *AL* ÷ *immédiat 8* *AL* ← *AL* MOD *immédiat 8*

Mnémonique

Instruction	**Opcode**	**Description**
AAM	D4h 0Ah	Crée une paire des valeurs dépaqueter BCD dans les registres AL et AH (invalide en mode 64-bits)
aucun	D4h *ib*	Crée une paire des valeurs dépaqueter immédiatement dans l'octets de base (invalide en mode 64-bits)

Cycles d'horloge

Instruction	Opcode	8086	8088	80186	80286	80386	i486	Pentium	Cx486SLC	Cx486DX	IBM 486BL 3X	UMC U5S
AAM	D4h 0Ah	83	83	19	16	17	15	18	16	16	17	12

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#DE(Division par zéro)	X	X	X	Valeur immédiate 8 bits contient 0
#UD(Opcode invalide)			X	Cette instruction est exécutée en mode 64-bits

Voir également

Instruction assembleur 80x86 - Instruction AAA
Instruction assembleur 80x86 - Instruction AAD
Instruction assembleur 80x86 - Instruction AAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 802
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 55.

Assembleur 80x86	**AAS**
INTEL 8088+	*Ascii Adjust al after Subtraction*

Syntaxe

AAS

Description

A l'aide de cette instruction, on peut adapter le résultat de la soustraction de 2 nombre de format *DCB*. Étant donné que le processeur soustrait ces valeurs comme s'il s'agissait de chiffres normales, des débordements se produisent dans les résultats. La commande *AAS* rectifie cela. Cette instruction ne peut fonctionner qu'avec des nombres sur 8 bits. Elle transforme la valeur placée après la soustraction dans le registre *AL* en une valeur *DCB* acceptable.

Algorithme

```
* Vaut 1 si c'est vrai
AL inférieur ← AL < 6
SI ( AL ∩ 0Ah ) U ( AF = 1 ) ALORS
  AL ← ( AL - 6 ) ∩ 0Fh
  AH ← AH - 1 – AL inférieur
  Drapeau CF ← 1
  Drapeau AF ← 1
SINON
  AL ← AL ∩ 0Fh
  Drapeau CF ← 0
  Drapeau AF ← 0
FIN SI
```

Mnémonique

Instruction	**Opcode**	**Description**
AAS	3Fh	Crée un nombre dépacter BCD à partir du contenu du registre AL (invalide en mode 64-bits)

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction est exécuté en mode 64-bits

Voir également

Instruction assembleur 80x86 - Instruction AAA
Instruction assembleur 80x86 - Instruction AAD
Instruction assembleur 80x86 - Instruction AAM

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 802
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 56.

Assembleur 80x86	**ADC**
INTEL 8088+	*arithmetic Add with Carry*

Syntaxe

ADC *Opérande Cible,Opérande Source*

Paramètres

Nom	Description
Opérande Cible	Ce paramètre permet d'indiquer l'opérande reçevant le résultat
Opérande Source	Ce paramètre permet d'indiquer l'opérande à ajouter au résultat

Description

Cette instruction, des plus indispensable, additionne 2 quantités numériques sur 8 ou 16 bits et ajoute ensuite la valeur de l'indicateur de retenue, lequel est contenu dans le drapeau *CF* (*CARRY FLAG)*, à la somme obtenu. Le résultat pourra naturellement être placé en mémoire ou dans un registre. Cette opération tient donc compte de la retenue éventuelle ainsi que de l'addition des 16 bits de poids faible pour continuer l'addition des 16 bits de poids fort.

Algorithme

Opérande Cible ← *Opérande Cible* + *Quantité Source* + drapeau *CF*

Mnémonique

Instruction	**Opcode**	**Description**
ADC AL, *imm8*	14h *ib*	Ajout immédiate de 8 bits à AL + CF
ADC AX, *imm16*	15h *iw*	Ajout immédiate de 16 bits à AX + CF

ADC EAX, *imm32*	15h *id*	Ajout immédiate de 32 bits à EAX + CF
ADC RAX, *imm32*	15h *id*	Ajout immédiate de 32 bits à RAX + CF
ADC *reg/mem8, imm8*	80h /2 *ib*	Ajout immédiate de 8 bits à registre/mémoire + CF
ADC *reg/mem16, imm16*	81h /2 *iw*	Ajout immédiate de 16 bits à registre/mémoire 16 bits + CF
ADC *reg/mem32, imm32*	81h /2 *id*	Ajout immédiate de 32 bits à registre/mémoire 32 bits + CF
ADC *reg/mem64, imm32*	81h /2 *id*	Ajout immédiate de 32 bits entier à registre/mémoire 64 bits + CF
ADC *reg/mem16, imm8*	83h /2 *ib*	Ajout immédiate de 8 bits entier à registre/mémoire 16 bits + CF
ADC *reg/mem32, imm8*	83h /2 *ib*	Ajout immédiate de 8 bits entier à registre/mémoire 16 bits + CF
ADC *reg/mem64, imm8*	83h /2 *ib*	Ajout immédiate de 8 bits entier à registre/mémoire 64 bits + CF
ADC *reg/mem8, reg8*	10h /r	Ajout registre de 8 bits à registre/mémoire 8 bits + CF
ADC *reg/mem16, reg16*	11h /r	Ajout registre de 16 bits à registre/mémoire 16 bits + CF
ADC *reg/mem32, reg32*	11h /r	Ajout registre de 32 bits à registre/mémoire 32 bits + CF
ADC *reg/mem64, reg64*	11h /r	Ajout registre de 64 bits à registre/mémoire 64 bits + CF
ADC *reg8, reg/mem8*	12h /r	Ajout registre de 8 bits à registre/mémoire 8 bits + CF
ADC *reg16, reg/mem16*	13h /r	Ajout registre de 16 bits à registre/mémoire 16 bits + CF
ADC *reg32, reg/mem32*	13h /r	Ajout registre de 32 bits à registre/mémoire 32 bits + CF
ADC *reg64, reg/mem64*	13h /r	Ajout registre de 64 bits à registre/mémoire 64 bits + CF

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Exemple

Cette exemple permet d'effectuer l'addition de la retenue d'une addition du registre AL sur le registre AH:

```
MOV AX,0001h
ADD AL,FFh
ADC AH,00h
```

Ce programme retournera donc la valeur 0100h dans le registre AX.

Voir également

Instruction assembleur 80x86 - Instruction ADD
Instruction assembleur 80x86 - Instruction SBB
Instruction assembleur 80x86 - Instruction SUB

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 802
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 126, 401
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 57 à 58.

Assembleur 80x86	**ADD**
INTEL 8088+	*arithmetic ADDition*

Syntaxe

ADD *Opérande Cible, Opérande Source*

Paramètres

Nom	Description
Opérande Cible	Ce paramètre permet d'indiquer l'opérande reçevant le résultat
Opérande Source	Ce paramètre permet d'indiquer l'opérande à ajouter au résultat

Description

Cette instruction additionne 2 quantités numériques sur 8 ou 16 bits. La somme pourra être placée soit en mémoire ou encore dans un registre.

Algorithme

Opérande Cible ← *Opérande Cible* + *Quantité Source* drapeau *CF* ← retenue

Mnémonique

Instruction	**Opcode**	**Description**
ADD AL, *imm8*	04h *ib*	Ajout immédiate de 8 bits à AL
ADD AX, *imm16*	05h *iw*	Ajout immédiate de 16 bits à AX
ADD EAX, *imm32*	05h *id*	Ajout immédiate de 32 bits à EAX
ADD RAX, *imm32*	05h *id*	Ajout d'un entier immédiate de 32 bits à RAX
ADD *reg/mem8, imm8*	80h /0 *ib*	Ajout immédiate de 8 bits à registre/mémoire 8 bits

ADD *reg/mem16*, *imm16*	81h /0 *iw*	Ajout immédiate de 16 bits à registre/mémoire 16 bits
ADD *reg/mem32*, *imm32*	81h /0 *id*	Ajout immédiate de 32 bits à registre/mémoire 32 bits
ADD *reg/mem64*, *imm32*	81h /0 *id*	Ajout entier immédiate de 32 bits à registre/mémoire 64 bits
ADD *reg/mem16*, *imm8*	83h /0 *ib*	Ajout entier immédiate de 8 bits à registre/mémoire 16 bits
ADD *reg/mem32*, *imm8*	83h /0 *ib*	Ajout entier immédiate de 8 bits à registre/mémoire 32 bits
ADD *reg/mem64*, *imm8*	83h /0 *ib*	Ajout entier immédiate de 8 bits à registre/mémoire 64 bits
ADD *reg/mem8*, *reg8*	00h /r	Ajout registre de 8 bits à registre/mémoire 8 bits
ADD *reg/mem16*, *reg16*	01h /r	Ajout registre de 16 bits à registre/mémoire 16 bits
ADD *reg/mem32*, *reg32*	01h /r	Ajout registre de 32 bits à registre/mémoire 32 bits
ADD *reg/mem64*, *reg64*	01h /r	Ajout registre de 64 bits à registre/mémoire 64 bits
ADD *reg8*, *reg/mem8*	02h /r	Ajout registre de 8 bits à registre/mémoire 8 bits
ADD *reg16*, *reg/mem16*	03h /r	Ajout registre de 16 bits à registre/mémoire 16 bits
ADD *reg32*, *reg/mem32*	03h /r	Ajout registre de 32 bits à registre/mémoire 32 bits
ADD *reg64*, *reg/mem64*	03h /r	Ajout registre de 64 bits à registre/mémoire 64 bits

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction ADC
Instruction assembleur 80x86 - Instruction SBB
Instruction assembleur 80x86 - Instruction SUB

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 803
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 401
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 59 à 60.

Assembleur 80x86 INTEL Pentium 4+

ADDPD

Add Packed Double-Precision Floating-Point Values

Syntaxe

ADDPD *dest,source*

Description

Cette instruction permet d'effectuer une addition de 2 paquets de valeurs réels de double précision d'une opérande source et d'une opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de double précision.

Algorithme

dest(0..63) ← *dest*(0..63) + *source*(0..63) *dest*(64..12) ← *dest*(64..12) + *source*(64..127)

Mnémonique

Instruction	Opcode	Description
ADDPD *xmm1, xmm2/m128*	66h 0Fh 58h /r	Cette instruction permet d'effectuer une addition de 2 paquets de valeurs réels de double précision d'une opérande source et d'une opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de double précision.

Assembleur 80x86 — ADDPS

INTEL Pentium III (SSE)+ — *Packed Single-Precision Floating-Point Add*

Syntaxe

ADDPS *xmm1*, *xmm2*/*m128*

Description

Cette instruction permet d'effectuer une addition *SIMD* de 4 paquets de valeurs réels de simple précision d'une opérande source et d'une opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de simple précision.

Algorithme

xmm1(31..0) ← *xmm1*(31..0) + *xmm2*/*m128*(31..0) *xmm1*(63..32) ← *xmm1*(63..32) + *xmm2*/*m128*(63..32) *xmm1*(95..64) ← *xmm1*(95..64) + *xmm2*/*m128*(95..64) *xmm1*(127..96) ← *xmm1*(127..96) + *xmm2*/*m128*(127..96)

Assembleur 80x86 INTEL Pentium 4+

ADDSD

Add Scalar Double-Precision Floating-Point Values

Syntaxe

ADDSD *dest,source*

Description

Cette instruction permet d'effectuer une addition de la partie basse d'une valeur réel de double précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de double précision.

Algorithme

dest(0..63) ← *dest*(0..63) + *source*(0..63)

Mnémonique

Instruction	Opcode	Description
ADDSD *xmm1, xmm2/m128*	F2h 0Fh 58h /r	Cette instruction permet d'effectuer une addition de la partie basse d'une valeur réel de double précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de double précision.

Assembleur 80x86 INTEL Pentium III (SSE)+

ADDSS

Scalar Single-Precision Floating-Point Add

Syntaxe

ADDSS *xmm1, xmm2/m32*

Description

Cette instruction permet d'effectuer une addition de la partie basse d'une valeur réel de simple précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de simple précision.

Algorithme

xmm1(31..0) ← *xmm1*(31..0) + *xmm2*/*m32*(31..0)

Assembleur 80x86 ADDSUBPD

INTEL Pentium 4 (SSE3)+ *Packed Double-FP Add/Subtract*

Syntaxe

ADDSUBPD *dest,source*

Description

Cette instruction permet d'effectuer une soustraction de la partie basse d'une valeur réel de double précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de double précision et d'effectuer une addition de la partie haute d'une valeur réel de double précision d'une opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de double précision.

Algorithme

dest(0..63) ← *dest*(0..63) - *source*(0..63) *dest*(64..127) ← *dest*(64..127) + *source*(64..127)

Mnémonique

Instruction	Opcode	Description
ADDSUBPD *xmm1, xmm2/m128*	66h 0Fh D0h /r	Cette instruction permet d'effectuer une soustraction de la partie basse d'une valeur réel de double précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de double précision et d'effectuer une addition de la partie haute d'une valeur réel de double précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de double précision.

Assembleur 80x86 — ADDSUBPS

INTEL Pentium 4 (SSE3)+ — *Packed Single-FP Add/Subtract*

Syntaxe

ADDSUBPS *dest,source*

Description

Cette instruction permet d'effectuer une soustraction de la partie basse d'un paquet de valeur réel de simple précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de simple précision et d'effectuer une addition de la partie haute d'un paquet de valeur réel de simple précision d'une opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de simple précision.

Algorithme

dest(0..31) ← *dest*(0..31) - *source*(0..31) *dest*(32..63) ← *dest*(32..63) + *source*(32..63) *dest*(64..95) ← *dest*(64..95) - *source*(64..95) *dest*(96..127) ← *dest*(96..127) + *source*(96..127)

Mnémonique

Instruction	Opcode	Description
ADDSUBPS *xmm1, xmm2/m128*	F2h 0Fh D0h /r	Cette instruction permet d'effectuer une soustraction de la partie basse d'un paquet de valeur réel de simple précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de simple précision et d'effectuer une addition de la partie haute d'un paquet de valeur réel de simple précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de simple précision.

Assembleur 80x86	**AND**
INTEL 8088+	*AND binary*

Syntaxe

AND *Opérande Cible, Opérande Source*

Paramètres

Nom	Description
Opérande Cible	Ce paramètre permet d'indiquer l'opérande reçevant le résultat
Opérande Source	Ce paramètre permet d'indiquer l'opérande effectuant le masque binaire sur le résultat

Description

L'instruction *AND* effectue un *ET BINAIRE* sur les 2 opérandes spécifiés, le calcul est placé dans la première opérande, c'est-à-dire l'*Opérande Cible*. Rappelons qu'un *ET BINAIRE* donne le résultat 1 si les 2 opérandes sont 1 et donne 0 dans les autres cas et cela sur chacun des bits de l'opérande.

Algorithme

Opérande Cible ← *Opérande Cible* ← *Opérande Source* drapeau *CF* ← 0 drapeau *OF* ← 0

Mnémonique

Instruction	**Opcode**	**Description**
AND AL, *imm8*	24h *ib*	Et binaire sur le contenu du registre AL avec une valeur immédiate 8 bits et met le résultat dans AL
AND AX, *imm16*	25h *iw*	Et binaire sur le contenu du registre AX avec une valeur immédiate 16 bits et met le résultat dans AX
AND EAX, *imm32*	25h *iw*	Et binaire sur le contenu du registre EAX avec une valeur immédiate 32 bits et met le résultat dans EAX

AND RAX, *imm32*	25h *id*	Et binaire sur le contenu du registre RAX avec une valeur entière immédiate 32 bits et met le résultat dans RAX
AND *reg/mem8*, *imm8*	80h /4 *ib*	Et binaire sur le contenu du registre/mémoire 8 bits avec une valeur immédiate 8 bits
AND *reg/mem16*, *imm16*	81h /4 *iw*	Et binaire sur le contenu du registre/mémoire 16 bits avec une valeur immédiate 16 bits
AND *reg/mem32*, *imm32*	81h /4 *id*	Et binaire sur le contenu du registre/mémoire 32 bits avec une valeur immédiate 32 bits
AND *reg/mem64*, *imm32*	81h /4 *id*	Et binaire sur le contenu du registre/mémoire 64 bits avec une valeur entière immédiate 32 bits
AND *reg/mem16*, *imm8*	83h /4 *ib*	Et binaire sur le contenu du registre/mémoire 16 bits avec une valeur entière immédiate 8 bits
AND *reg/mem32*, *imm8*	83h /4 *ib*	Et binaire sur le contenu du registre/mémoire 32 bits avec une valeur entière immédiate 8 bits
AND *reg/mem64*, *imm8*	83h /4 *ib*	Et binaire sur le contenu du registre/mémoire 64 bits avec une valeur entière immédiate 8 bits
AND *reg/mem8*, *reg8*	20h /r	Et binaire sur le contenu du registre/mémoire 8 bits avec un registre 8 bits
AND *reg/mem16*, *reg16*	21h /r	Et binaire sur le contenu du registre/mémoire 16 bits avec un registre 16 bits
AND *reg/mem32*, *reg32*	21h /r	Et binaire sur le contenu du registre/mémoire 32 bits avec un registre 32 bits
AND *reg/mem64*, *reg64*	21h /r	Et binaire sur le contenu du registre/mémoire 64 bits avec un registre 64 bits
AND *reg8*, *reg/mem8*	22h /r	Et binaire sur le contenu du registre 8 bits avec un registre/mémoire 8 bits
AND *reg16*, *reg/mem16*	23h /r	Et binaire sur le contenu du registre 16 bits avec un registre/mémoire 16 bits
AND *reg32*, *reg/mem32*	23h /r	Et binaire sur le contenu du registre 32 bits avec un registre/mémoire 32 bits
AND *reg64*, *reg/mem64*	23h /r	Et binaire sur le contenu du registre 64 bits avec un registre/mémoire 64 bits

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas

				canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction OR
Instruction assembleur 80x86 - Instruction NOT
Instruction assembleur 80x86 - Instruction NEG
Instruction assembleur 80x86 - Instruction TEST
Instruction assembleur 80x86 - Instruction XOR

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 803
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 401
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 61 à 62.

Assembleur 80x86 INTEL Pentium 4+

ANDNPD

Bitwise Logical AND NOT of Packed Double-Precision Floating-Point Values

Syntaxe

ANDNPD *dest,source*

Description

Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 paquets d'opérandes de valeur de format réel de double précision et inverse chacun des bits du résultat.

Algorithme

dest ← ¬ *dest* ∩ *source*

Mnémonique

Instruction	Opcode	Description
ANDNPD *xmm1, xmm2/m128*	66h 0Fh 55h /r	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 paquets d'opérandes de valeur de format réel de double précision et inverse chacun des bits du résultat.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

ANDNPS

Bitwise Logical AND NOT

Syntaxe

ANDNPS *dest,source*

Description

Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 opérandes 128 bits spécifiés et inverse chacun des bits du résultat.

Algorithme

dest ← ¬ *dest* ∩ *source*

Mnémonique

Instruction	Opcode	Description
ANDNPS *xmmreg,r/m128*	0Fh 55h /r	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 opérandes 128 bits spécifiés et inverse chacun des bits du résultat.

Assembleur 80x86 — ANDPD

INTEL Pentium 4+ — *Bitwise Logical AND of Packed Double-Precision Floating-Point Values*

Syntaxe

ANDPD *dest,source*

Description

Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 paquets d'opérandes de valeur de format réel de double précision.

Algorithme

dest ← *dest* ∩ *source*

Mnémonique

Instruction	Opcode	Description
ANDPD *xmm1*, *xmm2/m128*	66h 0Fh 54h /r	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 paquets d'opérandes de valeur de format réel de double précision.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

ANDPS

Bitwise Logical AND

Syntaxe

ANDPS *dest,source*

Description

Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 opérandes 128 bits spécifiés.

Algorithme

dest ← *dest* ∩ *source*

Mnémonique

Instruction	Opcode	Description
ANDPS *xmmreg,r/m128*	0Fh 54h /r	Cette instruction permet d'effectuer un *ET BINAIRE* sur les 2 opérandes 128 bits spécifiés.

Assembleur 80x86	**ARPL**
INTEL 80286+ (Mode protégé)	*Adjust RPL field of selector*

Syntaxe

ARPL *Opérande Cible, Opérande Source*

Description

Avec cette instruction, on pourra contrôler et corriger le niveau de privilège du segment de code en mode protégée.

Algorithme

SI RPL bits(0,1) **DE** *Opérande Cible* < RPL bits(0,1) **DE** *Opérande Source* **ALORS**
  drapeau *ZF* ← 1
  RPL bits(0,1) **DE** *Opérande Cible* ← RPL bits(0,1) **DE** *Opérande Source*
SINON
  drapeau *ZF* ← 0
FIN SI

Mnémonique

Instruction	**Opcode**	**Description**
ARPL *reg/mem16, reg16*	63h /r	Ajuste le RPL du sélecteur de segment de destination au niveau inférieur du RPL du sélecteur de segment spécifié dans le registre source 16 bits. Invalide en mode 64 bits.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#UD(Opcode invalide)		X	X	Cette instruction est seulement reconnu en mode protégé «legacy» et en mode de compatibilité.

#SS(Pile)			X	Une adresse mémoire dépasse la limite du segment de pile.
#GP(Protection général)			X	Une adresse mémoire dépasse la limite du segment de données
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un sélecteur de segment nulle est utilisé comme référence mémoire
#PF(Faute de page)			X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)			X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction LAR
Instruction assembleur 80x86 - Instruction LSL
Instruction assembleur 80x86 - Instruction VERR
Instruction assembleur 80x86 - Instruction VERW

Assembleur 80x86	**BOUND**
INTEL 80186+	*check array index against BOUNDs*

Syntaxe

BOUND *registre, mémoire*

Description

Cette instruction, ayant fait sont apparition sur un processeur méconnue du public, offre la possibilité professionnel de vérifier la validité pouvant exister entre avec un tableau et son indexation.

Algorithme

SI (Source De Gauche < [Source De Droite]) U (Source De Gauche > [Source De Droite + Taille Opérande ÷ 8]) **ALORS**
  Interruption 5
FIN SI

Mnémonique

Instruction	**Opcode**	**Description**
BOUND *reg16, mem16&mem16*	62h /r	Test avec un index de tableau 16 bits spécifié deux valeurs 16 bits (Invalide en mode 64 bits)
BOUND *reg32, mem32&mem32*	62h /r	Test avec un index de tableau 32 bits spécifié deux valeurs 32 bits (Invalide en mode 64 bits)

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#BR(Débordement de rang)	X	X	X	La limite du rang est dépassé.
#UD(Opcode invalide)	X	X	X	La source de

				l'opérande est un registre
			X	Cette instruction est exécuté en mode 64-bits
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction INT
Instruction assembleur 80x86 - Instruction INTO

Assembleur 80x86	**BSF**
INTEL 80386+	*Bit Scan Forward*

Syntaxe

BSF *Opérande Cible, Opérande Source*

Description

La combinaison binaire contenue dans l'*Opérande Source* est analysée de la droite vers la gauche jusqu'à ce que l'on rencontre un bit égal à 1. Le bit correspondant dans *Opérande Cible* sera alors également mis à 1 et tous les autres équivaudront à 0. Les opérandes peuvent toutefois être 2 registres ou un registre et un emplacement mémoire. Encore un dernier détail, ils doivent naturellement avoir tous les 2 le même type; soit 16 ou 32 bits.

Algorithme

```
SI Opérande Source = 0 ALORS
  drapeau ZF ← 1
  Opérande Cible ← Non définie
SINON
  drapeau ZF ← 0
  Index ← 0
  FAIRE TANT QUE bit ( Source , Index ) = 0
    Index ← Index + 1
    Opérande Cible ← Index
  FIN FAIRE
FIN SI
```

Mnémonique

Instruction	**Opcode**	**Description**
BSF *reg16, reg/mem16*	0Fh BCh /r	Balayage de bit vers l'avant du contenu du registre/mémoire 16 bits
BSF *reg32, reg/mem32*	0Fh BCh /r	Balayage de bit vers l'avant du contenu du registre/mémoire 32 bits
BSF *reg64, reg/mem64*	0Fh BCh /r	Balayage de bit vers l'avant du contenu du registre/mémoire 64 bits

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BSR

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 803
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 65.

Assembleur 80x86	**BSR**
INTEL 80386+	*Bit Scan Reverse*

Syntaxe

BSR *Opérande Cible, Opérande Source*

Description

La combinaison binaire contenue dans l'*Opérande Source* est analysée de la gauche vers la droite jusqu'à ce que l'on rencontre un bit égal à 1. Le bit correspondant dans *Opérande Cible* sera ensuite mis à 1 et tous les autres équivaudront à 0. Les opérandes peuvent toutefois être 2 registres ou un registre et un emplacement mémoire. Les 2 opérandes doivent naturellement avoir le même type; soit 16 ou 32 bits.

Algorithme

```
SI ( Registre Destination DANS [ AX , BX, CX , DX , SI , DI , BP , SP ] ) ALORS
  Début ← 15
SINON
  Début ← 31
FIN SI
SI Source = 0 ALORS
  drapeau ZF ← 1
  Destination ← Non définie
SINON
  drapeau ZF ← 0
  Index ← Début
  FAIRE TANT QUE bit ( Source , Index ) = 0
    Index ← Index - 1
    Destination ← Index
  FIN FAIRE
FIN SI
```

Mnémonique

Instruction	Opcode	Description
BSR *reg16, reg/mem16*	0Fh BDh /r	Balayage de bit vers l'arrière du contenu du registre/mémoire 16 bits
BSR *reg32, reg/mem32*	0Fh BDh /r	Balayage de bit vers l'arrière du contenu du registre/mémoire 32 bits
BSR *reg64, reg/mem64*	0Fh BDh /r	Balayage de bit vers l'arrière du contenu du registre/mémoire 64 bits

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulles est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BSF

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 804
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 66.

Assembleur 80x86	**BSWAP**
INTEL 80486+	*Byte Swap*

Syntaxe

BSWAP *Opérande*

Description

L'instruction tardive *BSWAP* inverse l'ordre des 4 octets d'un registre de taille de 32 bits. L'octet de poids le plus fort devient celui de poids le plus faible et on affecte les 2 autres octets de la même manière en les inversant eux aussi.

Algorithme

Temporaire ← *Opérande* *Opérande* (0 à 7) ← *Temporaire* (24 à 31) *Opérande* (8 à 15) ← *Temporaire* (16 à 23) *Opérande* (16 à 23) ← *Temporaire* (8 à 15) *Opérande* (24 à 31) ← *Temporaire* (0 à 7)

Mnémonique

Instruction	Opcode	Description
BSWAP *reg32*	0Fh C8h +rd	Inverse l'ordre des octets du registre 32 bits
BSWAP *reg64*	0Fh C8h +rq	Inverse l'ordre des octets du registre 64 bits

Cycles d'horloge

Instruction	Opcode	8086	8088	80186	80286	80386	i486	Pentium	Cx486S / Cx486SLC	Cx486DX	486BL3	UMC U5S
BSWAP *reg32*	0Fh C8h +rd						1	1	4	4	9	2

Exceptions

Aucune

Voir également

Instruction assembleur 80x86 - Instruction XCHG

Assembleur 80x86	**BT**
INTEL 80386+	*Bit Test*

Syntaxe

BT *Opérande Source 1, Opérande Source 2*

Description

Cette instruction permet de transférer de l'*Opérande Source 1* vers l'indicateur de retenue, le bit spécifié par l'*Opérande Source 2*. Les opérandes utilisés peuvent être soit 16 ou 32 bits.

Algorithme

drapeau *CF* ← BIT [*Opérande Source 1, Opérande Source 2*]

Remarque

Processeur INTEL: Quand vous faites des accès en bits dans la mémoire, le microprocesseur fabriquer par le constructeur INTEL effectue des accès en mémoire d'adressage sous la forme suivante:

Adresse ← *Segment* + (4 x (Bit d'offset ÷ 32))

Tandis que dans le cas d'une opérande de taille de 32 bits, ou lorsque vous faîtes des accès mémoire de 2 octets (16 bits), il utilise plutôt la forme suivante:

Adresse ← *Segment* + (2 x (Bit d'offset ÷ 16))

Également, lorsqu'il y a 1 octets, il faut appliquer l'opération en tenant compte du 8 bits:

Adresse ← *Segment* + (1 x (Bit d'offset ÷ 8))

Il y a donc au proportionnalité de la division en fonction de la dimension de l'accès en mémoire.

Mnémonique

Instruction	Opcode	Description
BT *reg/mem16, reg16*	0Fh A3h /r	Copie la valeur de sélection de bit dans le drapeau de retenu
BT *reg/mem32, reg32*	0Fh A3h /r	Copie la valeur de sélection de bit dans le drapeau de retenu
BT *reg/mem64, reg64*	0Fh A3h /r	Copie la valeur de sélection de bit dans le drapeau de retenu
BT *reg/mem16, imm8*	0Fh BAh /4 *ib*	Copie la valeur de sélection de bit dans le drapeau de retenu
BT *reg/mem32, imm8*	0Fh BAh /4 *ib*	Copie la valeur de sélection de bit dans le drapeau de retenu
BT *reg/mem64, imm8*	0Fh BAh /4 *ib*	Copie la valeur de sélection de bit dans le drapeau de retenu

Cycles d'horloge

Instruction	Opcode	8086	8088	80186	80286	80386	i486	Pentium	Cx486SLC	Cx486DX	486BL3	UMC U5S
BT *reg16, imm8*	0Fh BAh /4 *ib*					3	3	?	?	?	?	?
BT *mem16, imm8*	0Fh BAh /4 *ib*					12	6	?	?	?	?	?
BT *reg16, reg16*	0Fh A3h /r					3	3	?	?	?	?	?
BT *mem16, reg16*	0Fh A3h /r					12	12	?	?	?	?	?

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire

#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BTC
Instruction assembleur 80x86 - Instruction BTR
Instruction assembleur 80x86 - Instruction BTS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 804
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 68 à 69.

Assembleur 80x86	**BTC**
INTEL 80386+	*Bit Test and Complement*

Syntaxe

BTC *Opérande Source 1, Opérande Source 2*

Description

Cette instruction offre la possibilité de transférer tout d'abord de l'*Opérande Source 1* vers l'indicateur de retenue les bits spécifiés par l'*Opérande Source 2* puis d'ensuite inverser la valeur du bit correspondant dans l'*Opérande Source 1*. Les opérandes utilisés peuvent être soit sur 16 ou 32 bits. Il est possible d'associer 2 registres, un registre à un emplacement mémoire, un registre ou un emplacement mémoire à un registre ou à une valeur immédiate.

Algorithme

drapeau *CF* ← BIT [*Opérande Source 1, Opérande Source 2*] BIT [*Opérande Source 1, Opérande Source 2*] ← Pas BIT [*Opérande Source 1, Opérande Source 2*]

Remarque

- **Processeur INTEL:** Quand vous faites des accès en bits dans la mémoire, le microprocesseur fabriquer par le constructeur INTEL effectue des accès en mémoire d'adressage sous la forme suivante:

Adresse ← *Segment* + (4 x (Bit d'offset ÷ 32))

Tandis que dans le cas d'un opérande de taille de 32 bits, ou lorsque vous faîtes des accès mémoire de 2 octets (16 bits), il utilise plutôt la forme suivante:

Adresse ← *Segment* + (2 x (Bit d'offset ÷ 16))

Également, lorsqu'il y a 1 octets, il faut appliquer l'opération en tenant compte du 8 bits:

Adresse ← *Segment* + (1 x (Bit d'offset ÷ 8))

Il y a donc au proportionnalité de la division en fonction de la dimension de l'accès en mémoire.

Mnémonique

Instruction	Opcode	Description
BTC *reg/mem16, reg16*	0Fh BBh /r	Copie la valeur du bit de sélection dans le drapeau de retenu et complémente le bit de sélection.
BTC *reg/mem32, reg32*	0Fh BBh /r	Copie la valeur du bit de sélection dans le drapeau de retenu et complémente le bit de sélection.
BTC *reg/mem64, reg64*	0Fh BBh /r	Copie la valeur du bit de sélection dans le drapeau de retenu et complémente le bit de sélection.
BTC *reg/mem16, imm8*	0Fh BBh /r	Copie la valeur du bit de sélection dans le drapeau de retenu et complémente le bit de sélection.
BTC *reg/mem16, imm8*	0Fh BAh /7 ib	Copie la valeur du bit de sélection dans le drapeau de retenu et complémente le bit de sélection.
BTC *reg/mem32, imm8*	0Fh BAh /7 *ib*	Copie la valeur du bit de sélection dans le drapeau de retenu et complémente le bit de sélection.
BTC *reg/mem64, imm8*	0Fh BAh /7 *ib*	Copie la valeur du bit de sélection dans le drapeau de retenu et complémente le bit de sélection.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire

#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BT
Instruction assembleur 80x86 - Instruction BTR
Instruction assembleur 80x86 - Instruction BTS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 805
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 70 à 71.

Assembleur 80x86	**BTR**
INTEL 80386+	*Bit Test and Reset*

Syntaxe

BTR *Opérande Source 1, Opérande Source 2*

Description

Cette instruction permet de transférer tout d'abord de l'*Opérande Source 1* vers l'indicateur de retenue les bits spécifiés par l'*Opérande Source 2* puis met le bit correspondant de l'*Opérande Source 1* à 0. Les opérandes utilisés peuvent être soit sur 16 ou 32 bits.

Algorithme

drapeau *CF* ← BIT [*Opérande Source 1, Opérande Source 2*] BIT [*Opérande Source 1, Opérande Source 2*] ← 0

Remarque

Processeur INTEL: Quand vous faites des accès en bits dans la mémoire, le microprocesseur fabriquer par le constructeur INTEL effectue des accès en mémoire d'adressage sous la forme suivante:

Adresse ← *Segment* + (4 x (Bit d'offset ÷ 32))

Tandis que dans le cas d'une opérande de taille de 32 bits, ou lorsque vous faîtes des accès mémoire de 2 octets (16 bits), il utilise plutôt la forme suivante:

Adresse ← *Segment* + (2 x (Bit d'offset ÷ 16))

Également, lorsqu'il y a 1 octets, il faut appliquer l'opération en tenant compte du 8 bits:

Adresse ← *Segment* + (1 x (Bit d'offset ÷ 8))

Il y a donc au proportionnalité de la division en fonction de la dimension de l'accès en mémoire.

Mnémonique

Instruction	Opcode	Description
BTR *reg/mem16, reg16*	0Fh B3h /r	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.
BTR *reg/mem32, reg32*	0Fh B3h /r	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.
BTR *reg/mem32, reg32*	0Fh B3h /r	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.
BTR *reg/mem64, reg64*	0Fh B3h /r	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.
BTR *reg/mem64, reg64*	0Fh B3h /r	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.
BTR *reg/mem16, imm8*	0Fh B3h /r	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.
BTR *reg/mem16, imm8*	0Fh BAh /6 *ib*	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.
BTR *reg/mem32, imm8*	0Fh BAh /6 *ib*	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.
BTR *reg/mem64, imm8*	0Fh BAh /6 *ib*	Copie la valeur du bit de sélection dans le drapeau de retenu et efface le bit de sélection.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BT
Instruction assembleur 80x86 - Instruction BTC
Instruction assembleur 80x86 - Instruction BTS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 805
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 72.

Assembleur 80x86	**BTS**
INTEL 80386+	*Bit Test and Set*

Syntaxe

BTS *Opérande Source 1, Opérande Source 2*

Description

Cette instruction permet de transférer tout d'abord de *Opérande Source1* vers l'indicateur de retenue le bits spécifié par *Opérande Source 2* puis ensuite met le bits correspondant dans *Opérande Source 1* à 1. Les opérandes utilisés peuvent être sur soit 16 ou 32 bits.

Algorithme

drapeau *CF* ← BIT (*Opérande Source 1, Opérande Source 2*) BIT (*Opérande Source 1, Opérande Source 2*) ← 1

Remarque

- **Processeur INTEL:** Quand vous faites des accès en bits dans la mémoire, le microprocesseur fabriquer par le constructeur INTEL effectue des accès en mémoire d'adressage sous la forme suivante:

Adresse ← *Segment* + (4 x (Bit d'offset ÷ 32))

Tandis que dans le cas d'un opérande de taille de 32 bits, ou lorsque vous faîtes des accès mémoire de 2 octets (16 bits), il utilise plutôt la forme suivante:

Adresse ← *Segment* + (2 x (Bit d'offset ÷ 16))

Également, lorsqu'il y a 1 octet, il faut appliquer l'opération en tenant compte du 8 bits:

Adresse ← *Segment* + (1 x (Bit d'offset ÷ 8))

Il y a donc au proportionnalité de la division en fonction de la dimension de l'accès en mémoire.

Mnémonique

Instruction	Opcode	Description
BTS *reg/mem16*, *reg16*	0Fh ABh /r	Copie la valeur du bit de sélection dans le drapeau de retenu et fixe le bit de sélection.
BTS *reg/mem32*, *reg32*	0Fh ABh /r	Copie la valeur du bit de sélection dans le drapeau de retenu et fixe le bit de sélection.
BTS *reg/mem64*, *reg64*	0Fh ABh /r	Copie la valeur du bit de sélection dans le drapeau de retenu et fixe le bit de sélection.
BTS *reg/mem16*, *imm8*	0Fh BAh /5 *ib*	Copie la valeur du bit de sélection dans le drapeau de retenu et fixe le bit de sélection.
BTS *reg/mem32*, *imm8*	0Fh BAh /5 *ib*	Copie la valeur du bit de sélection dans le drapeau de retenu et fixe le bit de sélection.
BTS *reg/mem64*, *imm8*	0Fh BAh /5 *ib*	Copie la valeur du bit de sélection dans le drapeau de retenu et fixe le bit de sélection.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulles est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BT
Instruction assembleur 80x86 - Instruction BTC
Instruction assembleur 80x86 - Instruction BTR

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 805
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 72 à 73.

Assembleur 80x86	**CALL**
INTEL 8088+	*CALL*

Syntaxe

CALL *adresse*

Description

Cette instruction force le microprocesseur à exécuter les instructions du sous-programme indiqué par l'adresse d'appel avant de continuer. Dès que la routine est terminée, l'exécution reprendra son cours à l'instruction suivant le CALL.

Algorithme

Appel long	Appel court
PUSH CS PUSH IP CS:IP ← *Destination*	PUSH IP IP ← *Destination*

Mnémonique

Instruction	**Opcode**	**Description**
CALL *rel16off*	E8h *iw*	Appel court avec une destination spécifié par une adresse 16 bits
CALL *rel32off*	E8h *id*	Appel court avec une destination spécifié par une adresse 32 bits
CALL *reg/mem16*	FFh /2	Appel court avec une destination spécifié par un registre/mémoire 16 bits
CALL *reg/mem32*	FFh /2	Appel court avec une destination spécifié par un registre/mémoire 32 bits. Il n'y a pas de préfixe pour l'encoder en mode 64 bits.
CALL *reg/mem64*	FFh /2	Appel court avec une destination spécifié par un registre/mémoire 64 bits
CALL FAR *pntr16:16*	9Ah *cd*	Appel long direct, avec un destination spécifié par un pointeur long. Invalide en mode 64 bits.
CALL FAR *pntr16:32*	9Ah *cp*	Appel long direct, avec un destination spécifié par un pointeur long.

		Invalide en mode 64 bits.
CALL FAR *mem16:16*	FFh /3	Appel long indirect, avec un destination spécifié par un pointeur long en mémoire.
CALL FAR *mem16:32*	FFh /3	Appel long indirect, avec un destination spécifié par un pointeur long en mémoire.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
	X	X	X	L'opérande de destination dépasse la limite du segment de code ou n'est pas canonique.
			X	Un segment de données nulle est utilisé comme référence mémoire
#GP(Sélecteur)			X	Le sélecteur de segment de code de destination est un sélecteur nulle.
			X	Un code, d'un appel de pont, de tâche de pont, ou un descripteur *TSS* dépasse la limite du descripteur de table.
			X	Un sélecteur de segment de bit TI est fixer mais le sélecteur *LDT* est un sélecteur nulle.
			X	Le descripteur de segment spécifié par l'instruction n'est pas un segment de code, une tâche de pont, un appel de pont, ou un TSS disponible en mode *legacy*, ou un segment de code n'étant en mode 64 bits ou un appel de pont 64 bits dans un mode long.
			X	Le RPL de sélecteur de segment de code spécifié est non-conforme à une instruction plus grande que le *CPL*, ou le *DPL* n'est pas égale au *CPL*.
			X	Le DPL du descripteur de segment de code spécifié est conforme au

				instruction plus grande que le *CPL*.
			X	Le DPL de l'appel de pont, de tâche de pont, ou le descripteur TSS spécifié par l'instruction est inférieur au *CPL* ou *RPL*.
			X	Le descripteur de segment spécifié par l'appel de pont ou la tâche de pont est un sélecteur nulle.
			X	Le descripteur de segment spécifié par l'appel de pont n'est pas en segment de code en mode *legacy* pi n'est pas un segment de code 64 bits dans le mode long.
			X	Le DPL du du descripteur de segment spécifié par l'appel de pont est plus grand que le *CPL*.
			X	L'appel d'un pont 64 bits avec des bits d'attribues étendues n'est pas à 0.
			X	Le *TSS* du descripteur est présent dans le *LDT*.
#NP(Segment non présent)			X	L'accès au segment de code, appel d'un pont, tâche de pont, ou TSS n'est pas présent.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique, et n'est pas l'échange de pile spécifié.
#SS(Sélecteur)			X	Après l'échange de pile, un accès mémoire dépasse la limite de segment de pile ou est non-canonique.
			X	Après l'échange de pile, le registre SS n'est pas chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non-présent.
#TS(Sélecteur TSS invalide)			X	Lors d'une partie de l'échange entre la pile, la destination du segment de la pile ou du registre RSP dans le TSS est en dehors des limites TSS.

			X	Lors d'une partie de l'échange entre la pile, la destination du segment de la pile ou du registre RSP dans le TSS est dans un sélecteur nulle.
			X	Lors d'une partie de l'échange entre la pile, la destination du sélecteur de pile du bit TI est fixé, mais le sélecteur LDT n'est pas un sélecteur nulle.
			X	Lors d'une partie de l'échange entre la pile, la destination du sélecteur de segment de pile dans le *TSS* est en dehors des limites de la table du descripteur *GDT* ou *LDT*.
			X	Lors d'une partie de l'échange entre la pile, la destination du sélecteur de segment de pile dans le *TSS* contient un *RPL* n'est pas égale au *DPL*.
			X	Lors d'une partie de l'échange entre la pile, la destination du sélecteur de segment de pile dans le *TSS* contient un *DPL* n'est pas égale au *CPL* du sélecteur de segment de code.
			X	Lors d'une partie de l'échange entre la pile, la destination du sélecteur de segment de pile dans le *TSS* n'est pas dans un segment écrivable.
#UD(Opcode invalide)	X	X	X	L'appel long indirect avec le Opcode (FFh /3) est un opérande de registre.
			X	L'appel long direct avec le Opcode (9Ah) est effectué en mode 64 bits.

Voir également

Instruction assembleur 80x86 - Instruction RET
Instruction assembleur 80x86 - Instruction RETF
Instruction assembleur 80x86 - Instruction RETN

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 806
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 401
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 76 à 83.

Assembleur 80x86	**CBW**
INTEL 8088+	*Convert Byte To Word*

Syntaxe

CBW

Description

Cette instruction permet de convertir le nombre contenu dans le registre *AL* en un format sur 16 bits pour se retrouver dans le registre *AX* en appliquant une extension du signe. L'application des signes est appliquée de la façon suivante: le bit de signe de *AL* est copié dans les 8 bits du registre *AH*. Cette commande est destinée au conversion de valeur 8 à 16 bits uniquement.

Algorithme

SI *AL* ∩ 80h > 0 **ALORS**
  AH ← 080h
  AL ← *AL* ∩ 7Fh
SINON
  AH ← 0
FIN SI

Mnémonique

Instruction	Opcode	Description
CBW	98h	Étendre les signes de AL dans AX

Exceptions

Aucune

Voir également

Instruction assembleur 80x86 - Instruction CWD
Instruction assembleur 80x86 - Instruction CWDE

Instruction assembleur 80x86 - Instruction CDQ

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 806*
Assembleur Facile, *Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 402*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 84.*

Assembleur 80x86	**CDQ**
INTEL 80386+	*Convert Doubleword to Quadword*

Syntaxe

CDQ

Description

L'instruction *CDQ* convertit le double mot (32 bits) se trouvant dans le registre *EAX* en un quadruple mot, le résultat tient sur une taille de 64 bits. Pour réussir à fournir la réponse en entier, on procédera comme suit: Le double mot de poids le plus fort est placé dans *EDX*, celui de poids le plus faible dans le registre *EAX*. Les 32 bits du registre *EDX* sont positionnés à la valeur du bit de poids le plus fort du registre *EAX*.

Algorithme

EDX ← extension des signes de *EAX*

Mnémonique

Instruction	**Opcode**	**Description**
CQO	99h	Étendre les signes de EAX dans EDX:EAX

Exceptions

Aucune

Assembleur 80x86	**CDQE**
x86-64+	*Convert Doubleword to Quadword Extended*

Syntaxe

CDQE

Description

Cette instruction permet de convertir le double mot du registre EAX en un quadruple mot RAX, le résultat tient sur une taille de 64 bits.

Algorithme

RAX ← extension des signes de *EAX*

Mnémonique

Instruction	Opcode	Description
CDQE	98h	Étendre les signes de EAX dans RAX

Exceptions

Aucune

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 187 à 188.

Assembleur 80x86	**CLC**
INTEL 8088+	*Clear Carry Flag*

Syntaxe

```
CLC
```

Description

Cette instruction permet de mettre l'indicateur d'état de retenu *CF* à 0.

Algorithme

```
drapeau CF ← 0
```

Remarque

A titre purement informatif, sachez que ce registre d'état de retenue est toujours égal à 1 lorsqu'une opération génère une retenue arithmétique et qu'il est par conséquent impossible de représenter le résultat qu'avec l'aide des bits disponibles. Dans ce éventualité, l'indicateur de retenue est considéré comme $17^{ième}$ ou $9^{ième}$ bit du résultat. C'est surtout dans la perspective des instructions *ADC* et *SBB* que le positionnement à 0 de l'indicateur de retenue s'avère utile. Dans les autres cas, l'interruption 21h du système d'exploitation sans servira pour indiquer s'il y a eu une erreur d'exécution, toutefois ce n'est pas une situation relier directement au microprocesseur mais une programmation logiciel.

Mnémonique

Instruction	Opcode	Description
CLC	F8h	Efface le drapeau de retenu (CF) à 0

Exceptions

Aucune

Voir également

Instruction assembleur 80x86 - Instruction STC
Instruction assembleur 80x86 - Instruction CMC

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 807
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 402
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 86.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 189 à 189.

Assembleur 80x86	**CLD**
INTEL 8088+	*Clear Direction Flag*

Syntaxe

```
CLD
```

Description

Cette instruction met l'indicateur d'état *DF* à 0. Les opérations sur les chaînes de caractères, tel *CMPSB*, *CMPSD*, *CMPSQ*, *CMPSW*, *LODSB*, *LODSD*, *LODSQ*, *LODSW*, *MOVSB*, *MOVSD*, *MOVSQ*, *MOVSW*, *OUTSB*, *OUTSD*, *OUTSW*, *STOSB*, *STOSD*, *STOSQ* et *STOSW* par exemple, incrémenteront maintenant les registres *SI* et *DI*.

Algorithme

```
drapeau DF ← 0
```

Mnémonique

Instruction	Opcode	Description
CLD	FCh	Efface le drapeau de direction (DF) à 0

Exceptions

Aucune

Voir également

Instruction assembleur 80x86 - Instruction CMPS
Instruction assembleur 80x86 - Instruction INS
Instruction assembleur 80x86 - Instruction LODS
Instruction assembleur 80x86 - Instruction MOVS
Instruction assembleur 80x86 - Instruction OUTS
Instruction assembleur 80x86 - Instruction SCAS
Instruction assembleur 80x86 - Instruction STD
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 807
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 402
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 87.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 190 à 190.

Assembleur 80x86 CLFLUSH

INTEL Pentium 4 (SSE2)+ *Load Unaligned Integer 128 bits*

Syntaxe

CLFLUSH *mem8*

Description

Cette instruction permet de vider la ligne de cache d'une adresse linéaire.

Mnémonique

Instruction	Opcode	Description
CLFLUSH *mem8*	0Fh AEh /7	Vide la ligne de cache contenu dans *mem8*.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique, et n'est pas l'échange de pile spécifié.
#UD(Opcode	X	X	X	Cette instruction n'est pas supporté, lequel est indiqué par le

invalide)				registre EDX, bits 19 de l'instruction CPUID, fonction 0000_0001h.

Voir également

Instruction assembleur 80x86 - Instruction INVD
Instruction assembleur 80x86 - Instruction WBINVD

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 191 à 192.*

Assembleur 80x86 — CLGI

AMD-V

Clear Global Interrupt Flag

Syntaxe

CLGI

Description

Cette instruction permet d'effacer les drapeaux global d'interruption (*GIF*). Quand le *GIF* est à 0, toutes les interruptions externes sont désactivés.

Mnémonique

Instruction	Opcode	Description
CLGI	0Fh 01h DDh	Efface le drapeau d'interruption global (*GIF*).

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Les instructions SVM ne sont pas supportés, comme indiqué par le bit 2 du registre ECX de la fonction 8000_00001h de l'instruction *CPUID*.
			X	La machine virtuel sécurisé n'est pas activé (EFER.SVME=0).
	X	X		Cette instruction est seulement reconnu en mode protégé.
#GP(Protection général)	X	X	X	Le CPL ne vaut pas 0.

Voir également

Instruction assembleur 80x86 - Instruction STGI

Assembleur 80x86 — CLI

INTEL 8088+ — *Clear Interrupt Flag*

Syntaxe

```
CLI
```

Description

Cette instruction met l'indicateur d'état *IF* à 0. Après avoir exécuter cette instruction, aucune interruption ne sera admise tant que l'instruction *STI* n'est pas rencontrée.

Algorithme

drapeau *IF* ← 0

Mnémonique

Instruction	Opcode	Description
CLI	FAh	Efface le drapeau d'interruption (IF) et le met à 0.

Cycles d'horloge

Instruction	Opcode	8086	8088	80186	80286	80386	i486	Pentium	Cx486SLC	Cx486DX	IBM 486BL3X	UMC U5S
CLI	FAh	2	?	?	2	3	5	7	7	7	?	?

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)		X		Le CPL est supérieur au IOCPL et les extensions du mode virtuel sont désactivés (CR4.VME=0).
			X	Le CPL est supérieur au IOCPL et le CPL n'est pas 3 ou l'interruption du mode protégé virtuel n'est pas activé (CR4.PVI=0).

Remarque

Afin que les choses soit bien claire, sachez que même si ce registre d'état d'interruption est mit à 0, le microprocesseur ne masque pas les interruptions de type *NMI* (*Non masquable Interruption*). La commande *CLI* vise donc à faire en sorte que toutes les interruptions masquables ne soient plus exécutées. Cette interdiction peut être levée à l'aide de la commande *STI*.

Exemple

Cet exemple, destiné au mode réel, permet de retourner 1 dans le registre AL si la touche «*ALT*» du clavier est enfoncé ou 0 s'il ne l'est pas :

```
1. XOR AX,AX
2. MOV ES,AX
3. CLI
4. MOV AL,ES:[417h]
5. STI
6. AND AL,8
7. JZ @@End
8. MOV AL,1
9. @@End:
```

Voir également

Instruction assembleur 80x86 - Instruction STI

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 807
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 402
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 255.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 193 à 195.

Assembleur 80x86	**CLTS**
INTEL 80286+ (Mode protégé)	*CLear Task-Switched flags in CR0*

Syntaxe

```
CLTS
```

Description

Cette instruction permet d'effacer ou si vous préférez de mettre à 0 le drapeau de l'indicateur de tâche (*Task-Switch*) du registre *CR0*.

Algorithme

```
drapeau TS ← 0
```

Mnémonique

Instruction	Opcode	Description
CLTS	0Fh 06h	Efface le drapeau d'échangeur de tâche (TS) du registre *CR0*

Cycles d'horloge

Instruction	Opcode	8086	8088	80186	80286	80386	i486	Pentium	Cx486SLC	Cx486DX	IBM 486BL3X	UMC U5S
CLTS	0Fh 06h				2	5	7	?	?	?	?	?

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)		X	X	Le CPL ne vaut pas 0.

Voir également

Instruction assembleur 80x86 - Instruction LMSW

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 196 à 197.*

Assembleur 80x86	**CMC**
INTEL 8088+	*Complement Carry Flag*

Syntaxe

```
CMC
```

Description

Cette instruction offre la possibilité d'inverser la valeur de l'indicateur de retenue. Si ce dernier vaut 1, elle le met à 0 et inversement. On utilise cette instruction surtout dans le cas des additions et des soustractions.

Algorithme

drapeau *CF* ← **PAS** drapeau *CF*

Mnémonique

Instruction	Opcode	Description
CMC	F5h	Complément du drapeau de retenue (CF)

Exceptions

Aucune

Voir également

Instruction assembleur 80x86 - Instruction CLC
Instruction assembleur 80x86 - Instruction STC

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 807
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 90.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 198 à 198.

Assembleur 80x86	**CMOV**
INTEL P6+	*Conditionnel MOVe*

Syntaxe

```
CMOVcc registre, valeur
```

Description

C'est l'équivalent d'un *MOV* conditionnel. Cette instruction copie des données d'une source (*valeur*) à une destination (*registre*) à la condition que la condition (*cc*) demandé soit remplie.

Algorithme

```
SI condition ALORS
  registre ← valeur
FIN SI
```

Remarque

Avant de lancer cette instruction, il est important que vous sachiez que cette instruction n'est pas supporté par tous les *Pentium Pro*, l'instruction *CPUID* retourne un indicateur confirmant ou niant que cette instruction est supportée.

Mnémonique

Instruction	Opcode	Description
CMOVO *reg16, reg/mem16*	0Fh 40h /r	Copie si débordement (OF = 1)
CMOVO *reg32, reg/mem32*	0Fh 40h /r	Copie si débordement (OF = 1)
CMOVO *reg64, reg/mem64*	0Fh 40h /r	Copie si débordement (OF = 1)
CMOVNO *reg16, reg/mem16*	0Fh 41h /r	Copie si pas de débordement (OF = 0)
CMOVNO *reg32, reg/mem32*	0Fh 41h /r	Copie si pas de débordement (OF = 0)
CMOVNO *reg64, reg/mem64*	0Fh 41h /r	Copie si pas de débordement (OF = 0)
CMOVB *reg16, reg/mem16*	0Fh 42h /r	Copie si retenu (CF = 1)

CMOVB *reg32*, *reg/mem32*	0Fh 42h /r	Copie si retenu (CF = 1)
CMOVB *reg64*, *reg/mem64*	0Fh 42h /r	Copie si retenu (CF = 1)
CMOVC *reg16*, *reg/mem16*	0Fh 42h /r	Copie si retenu (CF = 1)
CMOVC *reg32*, *reg/mem32*	0Fh 42h /r	Copie si retenu (CF = 1)
CMOVC *reg64*, *reg/mem64*	0Fh 42h /r	Copie si retenu (CF = 1)
CMOVNAE *reg16*, *reg/mem16*	0Fh 42h /r	Copie si retenu (CF = 1)
CMOVNAE *reg32*, *reg/mem32*	0Fh 42h /r	Copie si retenu (CF = 1)
CMOVNAE *reg64*, *reg/mem64*	0Fh 42h /r	Copie si retenu (CF = 1)
CMOVNB *reg16*,*reg/mem16*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVNB *reg32*,*reg/mem32*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVNB *reg64*,*reg/mem64*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVNC *reg16*,*reg/mem16*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVNC *reg32*,*reg/mem32*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVNC *reg64*,*reg/mem64*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVAE *reg16*, *reg/mem16*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVAE *reg32*, *reg/mem32*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVAE *reg64*, *reg/mem64*	0Fh 43h /r	Copie si pas retenu (CF = 0)
CMOVZ *reg16*, *reg/mem16*	0Fh 44h /r	Copie si drapeau du zéro (ZF = 1)
CMOVZ *reg32*, *reg/mem32*	0Fh 44h /r	Copie si drapeau du zéro (ZF = 1)
CMOVZ *reg64*, *reg/mem64*	0Fh 44h /r	Copie si drapeau du zéro (ZF = 1)
CMOVE *reg16*, *reg/mem16*	0Fh 44h /r	Copie si drapeau du zéro (ZF = 1)
CMOVE *reg32*, *reg/mem32*	0Fh 44h /r	Copie si drapeau du zéro (ZF = 1)
CMOVE *reg64*, *reg/mem64*	0Fh 44h /r	Copie si drapeau du zéro (ZF = 1)
CMOVNZ *reg16*, *reg/mem16*	0Fh 45h /r	Copie si drapeau de pas de zéro (ZF = 0)
CMOVNZ *reg32*, *reg/mem32*	0Fh 45h /r	Copie si drapeau de pas de zéro (ZF = 0)
CMOVNZ *reg64*, *reg/mem64*	0Fh 45h /r	Copie si drapeau de pas de zéro (ZF = 0)
CMOVNE *reg16*, *reg/mem16*	0Fh 45h /r	Copie si drapeau de pas de zéro (ZF = 0)
CMOVNE *reg32*, *reg/mem32*	0Fh 45h /r	Copie si drapeau de pas de zéro (ZF = 0)
CMOVNE *reg64*, *reg/mem64*	0Fh 45h /r	Copie si drapeau de pas de zéro (ZF = 0)
CMOVBE *reg16*, *reg/mem16*	0Fh 46h /r	Copie si drapeau de zéro (ZF = 1) ou drapeau de retenu (CF = 1)
CMOVBE *reg32*, *reg/mem32*	0Fh 46h /r	Copie si drapeau de zéro (ZF = 1) ou drapeau de retenu (CF = 1)

CMOVBE *reg64*, *reg/mem64*	0Fh 46h /r	Copie si drapeau de zéro (ZF = 1) ou drapeau de retenu (CF = 1)
CMOVNA *reg16*, *reg/mem16*	0Fh 46h /r	Copie si drapeau de zéro (ZF = 1) ou drapeau de retenu (CF = 1)
CMOVNA *reg32*, *reg/mem32*	0Fh 46h /r	Copie si drapeau de zéro (ZF = 1) ou drapeau de retenu (CF = 1)
CMOVNA *reg64*, *reg/mem64*	0Fh 46h /r	Copie si drapeau de zéro (ZF = 1) ou drapeau de retenu (CF = 1)
CMOVNBE *reg16*, *reg/mem16*	0Fh 47h /r	Copie si drapeau de pas de zéro (ZF = 0) et drapeau de pas de retenu (CF = 0)
CMOVNBE *reg32*, *reg/mem32*	0Fh 47h /r	Copie si drapeau de pas de zéro (ZF = 0) et drapeau de pas de retenu (CF = 0)
CMOVNBE *reg64*, *reg/mem64*	0Fh 47h /r	Copie si drapeau de pas de zéro (ZF = 0) et drapeau de pas de retenu (CF = 0)
CMOVA *reg16*, *reg/mem16*	0Fh 47h /r	Copie si drapeau de pas de zéro (ZF = 0) et drapeau de pas de retenu (CF = 0)
CMOVA *reg32*, *reg/mem32*	0Fh 47h /r	Copie si drapeau de pas de zéro (ZF = 0) et drapeau de pas de retenu (CF = 0)
CMOVA *reg64*, *reg/mem64*	0Fh 47h /r	Copie si drapeau de pas de zéro (ZF = 0) et drapeau de pas de retenu (CF = 0)
CMOVS *reg16*, *reg/mem16*	0Fh 48h /r	Copie si drapeau de signe (SF = 1)
CMOVS *reg32*, *reg/mem32*	0Fh 48h /r	Copie si drapeau de signe (SF = 1)
CMOVS *reg64*, *reg/mem64*	0Fh 48h /r	Copie si drapeau de signe (SF = 1)
CMOVNS *reg16*, *reg/mem16*	0Fh 49h /r	Copie si drapeau de pas de signe (SF = 0)
CMOVNS *reg32*, *reg/mem32*	0Fh 49h /r	Copie si drapeau de pas de signe (SF = 0)
CMOVNS *reg64*, *reg/mem64*	0Fh 49h /r	Copie si drapeau de pas de signe (SF = 0)
CMOVP *reg16*, *reg/mem16*	0Fh 4Ah /r	Copie si drapeau de parité (PF = 1)
CMOVP *reg32*, *reg/mem32*	0Fh 4Ah /r	Copie si drapeau de parité (PF = 1)
CMOVP *reg64*, *reg/mem64*	0Fh 4Ah /r	Copie si drapeau de parité (PF = 1)
CMOVPE *reg16*, *reg/mem16*	0Fh 4Ah /r	Copie si drapeau de parité (PF = 1)
CMOVPE *reg32*, *reg/mem32*	0Fh 4Ah /r	Copie si drapeau de parité (PF = 1)
CMOVPE *reg64*, *reg/mem64*	0Fh 4Ah /r	Copie si drapeau de parité (PF = 1)
CMOVNP *reg16*, *reg/mem16*	0Fh 4Bh /r	Copie si drapeau de pas de parité (PF = 0)
CMOVNP *reg32*, *reg/mem32*	0Fh 4Bh /r	Copie si drapeau de pas de parité (PF = 0)
CMOVNP *reg64*, *reg/mem64*	0Fh 4Bh /r	Copie si drapeau de pas de parité (PF = 0)
CMOVPO *reg16*, *reg/mem16*	0Fh 4Bh /r	Copie si drapeau de pas de parité (PF = 0)

CMOVPO *reg32*, *reg/mem32*	0Fh 4Bh /r	Copie si drapeau de pas de parité (PF = 0)
CMOVPO *reg64*, *reg/mem64*	0Fh 4Bh /r	Copie si drapeau de pas de parité (PF = 0)
CMOVL *reg16*, *reg/mem16*	0Fh 4Ch /r	Copie si SF <> OF
CMOVL *reg32*, *reg/mem32*	0Fh 4Ch /r	Copie si SF <> OF
CMOVL *reg64*, *reg/mem64*	0Fh 4Ch /r	Copie si SF <> OF
CMOVNGE *reg16*, *reg/mem16*	0Fh 4Ch /r	Copie si SF <> OF
CMOVNGE *reg32*, *reg/mem32*	0Fh 4Ch /r	Copie si SF <> OF
CMOVNGE *reg64*, *reg/mem64*	0Fh 4Ch /r	Copie si SF <> OF
CMOVNL *reg16*, *reg/mem16*	0Fh 4Dh /r	Copie si SF = OF
CMOVNL *reg32*, *reg/mem32*	0Fh 4Dh /r	Copie si SF = OF
CMOVNL *reg64*, *reg/mem64*	0Fh 4Dh /r	Copie si SF = OF
CMOVGE *reg16*, *reg/mem16*	0Fh 4Dh /r	Copie si SF = OF
CMOVGE *reg32*, *reg/mem32*	0Fh 4Dh /r	Copie si SF = OF
CMOVGE *reg64*, *reg/mem64*	0Fh 4Dh /r	Copie si SF = OF
CMOVLE *reg16*, *reg/mem16*	0Fh 4Eh /r	Copie si ZF = 1 ou SF <> OF
CMOVLE *reg32*, *reg/mem32*	0Fh 4Eh /r	Copie si ZF = 1 ou SF <> OF
CMOVLE *reg64*, *reg/mem64*	0Fh 4Eh /r	Copie si ZF = 1 ou SF <> OF
CMOVNG *reg16*, *reg/mem16*	0Fh 4Eh /r	Copie si ZF = 1 ou SF <> OF
CMOVNG *reg32*, *reg/mem32*	0Fh 4Eh /r	Copie si ZF = 1 ou SF <> OF
CMOVNG *reg64*, *reg/mem64*	0Fh 4Eh /r	Copie si ZF = 1 ou SF <> OF
CMOVNLE *reg16*, *reg/mem16*	0Fh 4Fh /r	Copie si ZF = 0 et SF = OF
CMOVNLE *reg32*, *reg/mem32*	0Fh 4Fh /r	Copie si ZF = 0 et SF = OF
CMOVNLE *reg64*, *reg/mem64*	0Fh 4Fh /r	Copie si ZF = 0 et SF = OF
CMOVG *reg16*, *reg/mem16*	0Fh 4Fh /r	Copie si ZF = 0 et SF = OF
CMOVG *reg32*, *reg/mem32*	0Fh 4Fh /r	Copie si ZF = 0 et SF = OF
CMOVG *reg64*, *reg/mem64*	0Fh 4Fh /r	Copie si ZF = 0 et SF = OF

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Cette instruction n'est pas supporté. C'est instruction est supporté si l'indicateur

				de bit 15 du registre EDX de l'instruction CPUID avec la fonction 0000_0001h ou 8000_0001h.
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOV

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 199 à 204.

Assembleur 80x86 — CMP

INTEL 8088+

Compare two operands

Syntaxe

CMP Opérande Cible, Opérande Source

Description

Cette instruction offre la possibilité essentielle de comparer 2 registres ou emplacements de mémoire. Le résultat de la comparaison est indiqué par les indicateurs (registre drapeau du microprocesseur).

Algorithme

Drapeau ← opérande1 = opérande2

Mnémonique

Instruction	Opcode	Description
CMP AL, *imm8*	3Ch *ib*	Compare une valeur 8 bits immédiate avec le contenu du registre AL.
CMP AX, *imm16*	3Dh *iw*	Compare une valeur 16 bits immédiate avec le contenu du registre AX.
CMP EAX, *imm32*	3Dh *iw*	Compare une valeur 32 bits immédiate avec le contenu du registre EAX.
CMP RAX, *imm32*	3Dh *id*	Compare une valeur 32 bits immédiate avec le contenu du registre RAX.
CMP *reg/mem8*, *imm8*	80h /7 *ib*	Compare une valeur 8 bits immédiate avec le contenu d'une opérande mémoire ou d'un registre 8 bits.
CMP *reg/mem16*, *imm16*	81h /7 *iw*	Compare une valeur 16 bits immédiate avec le contenu d'une opérande mémoire ou d'un registre 16 bits.
CMP *reg/mem32*, *imm32*	81h /7 *id*	Compare une valeur 32 bits immédiate avec le contenu d'une opérande mémoire ou d'un registre 32 bits.
CMP *reg/mem64*, *imm32*	81h /7 *id*	Compare une valeur 32 bits immédiate avec le contenu d'une opérande mémoire ou d'un registre 64 bits.

CMP *reg/mem16*, *imm8*	83h /7 *ib*	Compare une valeur 8 bits immédiate avec le contenu d'une opérande mémoire ou d'un registre 16 bits.
CMP *reg/mem32*, *imm8*	83h /7 *ib*	Compare une valeur 8 bits immédiate avec le contenu d'une opérande mémoire ou d'un registre 32 bits.
CMP *reg/mem64*, *imm8*	83h /7 *ib*	Compare une valeur 8 bits immédiate avec le contenu d'une opérande mémoire ou d'un registre 64 bits.
CMP *reg/mem8*, *reg8*	38h /r	Compare le contenu d'une opérande mémoire ou d'un registre 8 bits avec un registre 8 bits.
CMP *reg/mem16*, *reg16*	39h /r	Compare le contenu d'une opérande mémoire ou d'un registre 16 bits avec un registre 16 bits.
CMP *reg/mem32*, *reg32*	39h /r	Compare le contenu d'une opérande mémoire ou d'un registre 32 bits avec un registre 32 bits.
CMP *reg/mem64*, *reg64*	39h /r	Compare le contenu d'une opérande mémoire ou d'un registre 64 bits avec un registre 64 bits.
CMP *reg8*, *reg/mem8*	39h /r	Compare le contenu d'une opérande mémoire ou d'un registre 8 bits avec un registre 8 bits.
CMP *reg16*, *reg/mem16*	3Bh /r	Compare le contenu d'une opérande mémoire ou d'un registre 16 bits avec un registre 16 bits.
CMP *reg32*, *reg/mem32*	3Bh /r	Compare le contenu d'une opérande mémoire ou d'un registre 32 bits avec un registre 32 bits.
CMP *reg64*, *reg/mem64*	3Bh /r	Compare le contenu d'une opérande mémoire ou d'un registre 64 bits avec un registre 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Exemple

L'exemple suivant permet de retourner 1 si le caractère spécifié par la variable «*Chr*» est une lettre de l'alphabet (majuscule ou minucule) ou 0 s'il ne l'est pas :

```
1. XOR AL,AL
2. MOV CL,Chr
3. AND CL,0DFh
4. CMP CL,'A'
5. JB @1
6. CMP CL,'Z'
7. JA @1
8. INC AL
9. @1:
```

Voir également

Instruction assembleur 80x86 - Instruction SUB
Instruction assembleur 80x86 - Instruction CMPS
Instruction assembleur 80x86 - Instruction SCAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 808
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 403
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 94 à 96.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 206 à 208.

Assembleur 80x86 — CMPPD

INTEL Pentium 4 (SSE2)+

Compare Packed Double-Precision Floating-Point Values

Syntaxe

CMPPD *dest*,*source*,*immediat*

Description

Cette instruction permet d'effectuer une comparaison *SIMD* de 2 paquets de valeurs réels de double précision d'un opérande source et d'un opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.

Algorithme

EVALUER CAS *immediat* **DE**
CAS 0: OPERATION ← EQ
CAS 1: OPERATION ← LT
CAS 2: OPERATION ← LE
CAS 3: OPERATION ← UNORD
CAS 4: OPERATION ← NEQ
CAS 5: OPERATION ← NLT
CAS 6: OPERATION ← NLE
CAS 7: OPERATION ← ORD
FIN SI
CMP0 ← *dest*(63..0) OPERATION *source*(63..0)
CMP1 ← *dest*(127..64) OPERATION *source*(127..64)
SI CMP0 **ALORS**
  dest(63..0) ← FFFFFFFFFFFFFFFFh
SINON
  dest(63..0) ← 0000000000000000h
FIN SI
SI CMP1 **ALORS**
  dest(127..64) ← FFFFFFFFFFFFFFFFh
SINON
  dest(127..64) ← 0000000000000000h

FIN SI

Mnémonique

Instruction	Opcode	Description
CMPPD *xmm1,xmm2/m128,imm8*	66h 0Fh C2h /r *ib*	Cette instruction permet d'effectuer une comparaison *SIMD* de 2 paquets de valeurs réels de double précision d'un opérande source et d'un opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 209 à 213.*

Assembleur 80x86 — CMPPS

INTEL Pentium III (SSE)+ — *Compare Packed Single-Precision Floating-Point Values*

Syntaxe

CMPPS *dest*, *source*, *imm8*

Description

Cette instruction permet d'effectuer une comparaison *SIMD* de 4 paquets de valeurs réels de simple précision d'une opérande source et d'une opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.

Algorithme

```
EVALUER CAS (imm8) DE
  CAS 0: OPERATION EQ
  CAS 1: OPERATION LT
  CAS 2: OPERATION LE
  CAS 3: OPERATION UNORD
  CAS 4: OPERATION NE
  CAS 5: OPERATION NLT
  CAS 6: OPERATION NLE
  CAS 7: OPERATION ORD
FIN EVALUER CAS
CMP0 ← dest(31..0) OPERATION source(31..0)
CMP1 ← dest(63..32) OPERATION source(63..32)
CMP2 ← dest(95..64) OPERATION source(95..64)
CMP3 ← dest(127..96) OPERATION source(127..96)
SI CMP0 est vrai ALORS
  dest(31..0) ← FFFFFFFFh
SINON
  dest(31..0) ← 00000000h
FIN SI
```

SI CMP1 est vrai **ALORS**
  dest(63..32) ← FFFFFFFFh
SINON
  dest(63..32) ← 00000000h
FIN SI
SI CMP2 est vrai **ALORS**
  dest(95..64) ← FFFFFFFFh
SINON
  dest(95..64) ← 00000000h
FIN SI
SI CMP3 est vrai **ALORS**
  dest(127..96) ← FFFFFFFFh
SINON
  dest(127..96) ← 00000000h
FIN SI

Mnémonique

Instruction	Opcode	Description
CMPPS *xmm1,xmm2/m128,imm8*	0Fh C2h /r imm8	Cette instruction permet d'effectuer une comparaison *SIMD* de 4 paquets de valeurs réels de simple précision d'une opérande source et d'une opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 214 à 218.

Assembleur 80x86 — CMPS

INTEL 8088+ — *Compare String*

Syntaxe

CMPS *destination,source*

Description

Cette instruction permet d'effectuer la comparaison d'un octet, d'un mot, double mot ou du quadruple mot spécifié avec l'opérande source et destination spécifié et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.

Algorithme

```
Fixe les drapeaux d'états de (destination = source)
SI (comparison d'octet) ALORS
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 1
    (E)DI ← (E)DI + 1
  SINON
    (E)SI ← (E)SI - 1
    (E)DI ← (E)DI - 1
  FIN SI
SINON SI (comparison d'un mot) ALORS
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 2
    (E)DI ← (E)DI + 2
  SINON
    (E)SI ← (E)SI - 2
    (E)DI ← (E)DI - 2
  FIN SI
SINON SI (comparaison d'un double mot) ALORS
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 4
```

```
    (E)DI ← (E)DI + 4
  SINON
    (E)SI ← (E)SI - 4
    (E)DI ← (E)DI - 4
  FIN SI
SINON SI (comparaison d'un quadruple mot) ALORS
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 8
    (E)DI ← (E)DI + 8
  SINON
    (E)SI ← (E)SI - 8
    (E)DI ← (E)DI - 8
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CMPS *mem8*, *mem8*	A6h	Compare l'octet à l'adresse DS:(R)SI avec l'octet à l'adresses ES:(R)DI et incrémente ou décrémente les registres (R)SI et (R)DI
CMPS *mem16*, *mem16*	A7h	Compare le mot à l'adresse DS:(R)SI avec le mot à l'adresses ES:(R)DI et incrémente ou décrémente les registres (R)SI et (R)DI
CMPS *mem32*, *mem32*	A7h	Compare le double mot à l'adresse DS:(R)SI avec le double mot à l'adresses ES:(R)DI et incrémente ou décrémente les registres (R)SI et (R)DI
CMPS *mem64*, *mem64*	A7h	Compare le quadruple mot à l'adresse DS:(R)SI avec le quadruple mot à l'adresses ES:(R)DI et incrémente ou décrémente les

		registres (R)SI et (R)DI

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement

				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CMP
Instruction assembleur 80x86 - Instruction SCAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 808
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 97.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 219 à 224.

Assembleur 80x86 — CMPSB

INTEL 8086+ — *Compare String Byte*

Syntaxe

```
CMPSB
```

Description

Cette instruction permet d'effectuer la comparaison d'un octet avec l'opérande source (DS:(R)SI) et destination (ES:(R)DI) et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.

Algorithme

```
Fixe les drapeaux d'états de ([(E)DI] = [(E)SI])
SI DF = 0 ALORS
  (E)SI ← (E)SI + 1
  (E)DI ← (E)DI + 1
SINON
  (E)SI ← (E)SI - 1
  (E)DI ← (E)DI - 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CMPSB	A6h	Compare l'octet à l'adresse DS:(R)SI avec l'octet à l'adresses ES:(R)DI et incrémente ou décrémente les registres (R)SI et (R)DI

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué

				quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CMP
Instruction assembleur 80x86 - Instruction SCAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 808
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 403
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 97.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 219 à 224.

Assembleur 80x86 — CMPSD

INTEL 80386+ — *Compare String Double Word*

Syntaxe

```
CMPSD
```

Description

Cette instruction permet d'effectuer la comparaison d'un double mot avec l'opérande source (DS:(R)SI) et destination (ES:(R)DI) et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.

Algorithme

```
Fixe les drapeaux d'états de ([(E)DI] = [(E)SI])
SI DF = 0 ALORS
  (E)SI ← (E)SI + 4
  (E)DI ← (E)DI + 4
SINON
  (E)SI ← (E)SI - 4
  (E)DI ← (E)DI - 4
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CMPSD	A7h	Compare le double mot à l'adresse DS:(R)SI avec le double mot à l'adresses ES:(R)DI et incrémente ou décrémente les registres (R)SI et

		(R)DI

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement

				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CMP
Instruction assembleur 80x86 - Instruction SCAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 809
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 97.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 219 à 224.

Assembleur 80x86 — CMPSD

INTEL Pentium 4 (SSE2)+

Compare Scalar Double-Precision Floating-Point Values

Syntaxe

CMPSD *dest,source,immediat*

Description

Cette instruction permet d'effectuer une comparaison de la partie basse de valeurs réelles de double précision d'un opérande source et d'un opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.

Algorithme

```
EVALUER CAS immediat DE
  CAS 0: OPERATION ← EQ
  CAS 1: OPERATION ← LT
  CAS 2: OPERATION ← LE
  CAS 3: OPERATION ← UNORD
  CAS 4: OPERATION ← NEQ
  CAS 5: OPERATION ← NLT
  CAS 6: OPERATION ← NLE
  CAS 7: OPERATION ← ORD
FIN EVALUER CAS
CMP0 ← dest(63..0) OPERATION source(63..0)
SI CMP0 ALORS
  dest(63..0) ← FFFFFFFFFFFFFFFFh
SINON
  dest(63..0) ← 0000000000000000h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CMPSD *xmm1,xmm2/m64, imm8*	F2h 0Fh C2h /r *ib*	Cette instruction permet d'effectuer une comparaison de la partie basse de valeurs réelles de double précision d'un opérande source et d'un opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 225 à 228.

Assembleur 80x86 CMPSQ

x86-64+

Compare String Quadword

Syntaxe

```
CMPSQ
```

Description

Cette instruction permet d'effectuer la comparaison d'un quadruple mot avec l'opérande source (DS:(R)SI) et destination (ES:(R)DI) et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.

Algorithme

```
Fixe les drapeaux d'états de ([(E)DI] = [(E)SI])
SI DF = 0 ALORS
  (E)SI ← (E)SI + 8
  (E)DI ← (E)DI + 8
SINON
  (E)SI ← (E)SI - 8
  (E)DI ← (E)DI - 8
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CMPSQ	A7h	Compare le quadruple mot à l'adresse DS:(R)SI avec le quadruple mot à l'adresses ES:(R)DI et incrémente ou décrémente les

		registres (R)SI et (R)DI

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement

				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CMP
Instruction assembleur 80x86 - Instruction SCAS

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 219 à 224.*

Assembleur 80x86 — CMPSS

INTEL Pentium III (SSE)+

Compare Scalar Single-Precision Floating-Point Values

Syntaxe

CMPSS *dest*, *source*, *imm8*

Description

Cette instruction permet d'effectuer une comparaison de la partie basse de valeurs réels de simple précision d'une opérande source et d'une opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.

Algorithme

```
EVALUER CAS (imm8) DE
  CAS 0: OPERATION EQ
  CAS 1: OPERATION LT
  CAS 2: OPERATION LE
  CAS 3: OPERATION UNORD
  CAS 4: OPERATION NEQ
  CAS 5: OPERATION NLT
  CAS 6: OPERATION NLE
  CAS 7: OPERATION ORD
FIN EVALUER CAS
CMP0 ← dest(31..0) OPERATION source(31..0)
SI CMP0 est vrai ALORS
  dest(31..0) ← FFFFFFFFh
SINON
  dest(31..0) ← 00000000h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CMPSS *xmm1,xmm2/m128,imm8*	F3h 0Fh C2h /r imm8	Cette instruction permet d'effectuer une comparaison de la partie basse de valeurs réels de simple précision d'une opérande source et d'une opérande destination et entrepose le résultat de la comparaison dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 229 à 232.*

Assembleur 80x86 — CMPSW

INTEL 8086+ — *Compare String Word*

Syntaxe

```
CMPSW
```

Description

Cette instruction permet d'effectuer la comparaison d'un mot avec l'opérande source (DS:(R)SI) et destination (ES:(R)DI) et fixe l'état des drapeaux du registres *EFLAGS* en fonction des résultats de la comparaison.

Algorithme

```
Fixe les drapeaux d'états de ([(E)DI] = [(E)SI])
SI DF = 0 ALORS
  (E)SI ← (E)SI + 2
  (E)DI ← (E)DI + 2
SINON
  (E)SI ← (E)SI - 2
  (E)DI ← (E)DI - 2
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CMPSW	A7h	Compare le mot à l'adresse DS:(R)SI avec le mot à l'adresses ES:(R)DI et incrémente ou décrémente les registres (R)SI et (R)DI

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué

				quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CMP
Instruction assembleur 80x86 - Instruction SCAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 809
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 403
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 97.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 219 à 224.

Assembleur 80x86	**CMPXCHG**
INTEL 80486+	*CoMPare and eXCHanGe*

Syntaxe

CMPXCHG *Opérande Destination*, *Opérande Source*

Description

A l'aide de l'instruction *CMPXCHG*, on peut comparer la destination avec l'accumulateur (*AL*, *AX* ou *EAX*). S'ils sont égaux, l'opérande de destination reçoit le contenu de l'opérande source, tandis que s'ils sont différents, l'accumulation (*AL*, *AX* ou *EAX*) reçoit le contenu de l'opérande de destination.

Algorithme

SI *accumulateur* = *Opérande Destination* **ALORS**
  drapeau *ZF* ← 1
  Opérande Destination ← *Opérande Source*
SINON
  drapeau *ZF* ← 0
  accumulateur ← *Opérande Destination*
FIN SI

Mnémonique

Instruction	Opcode	Description
CMPXCHG *reg/mem8*, *reg8*	0Fh B0h /r	Compare de registre AL avec un emplacement 8 bits de registre ou mémoire. S'ils sont égale, le deuxième opérande est copié dans le première opérande.
CMPXCHG *reg/mem16*, *reg16*	0Fh B1h /r	Compare de registre AX avec un emplacement 16 bits de registre ou

		mémoire. S'ils sont égale, le deuxième opérande est copié dans le première opérande.
CMPXCHG *reg/mem32*, *reg32*	0Fh B1h /r	Compare de registre EAX avec un emplacement 32 bits de registre ou mémoire. S'ils sont égale, le deuxième opérande est copié dans le première opérande.
CMPXCHG *reg/mem64*, *reg64*	0Fh B1h /r	Compare de registre RAX avec un emplacement 64 bits de registre ou mémoire. S'ils sont égale, le deuxième opérande est copié dans le première opérande.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de

				destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CMPXCHG8B
Instruction assembleur 80x86 - Instruction CMPXCHG16B

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 233 à 235.*

Assembleur 80x86	**CMPXCHG8B**
INTEL Pentium	*CoMPare and eXCHanGe 8 Bytes*

Syntaxe

CMPXCHG8B *registre, mémoire*

Description

L'instruction *CMPXCHG8B* compare un nombre de 64 bits formée par le couple registre de *EDX:EAX* avec une valeur 64 bits se trouvant en mémoire. Si ces valeurs sont équivalente, les 64 bits se trouvant en mémoire sont remplacés par les 64 bits formés par le couple de registre *ECX:EBX*. Dans le cas contraire, les 64 bits formés par les registres *EDX:EAX* sont remplacés par les 64 bits se trouvant en mémoire.

Algorithme

SI *EDX:EAX = Opérande Destination* **ALORS**
  drapeau *ZF* ← 1
  Opérande Destination ← *ECX:EBX*
SINON
  drapeau *ZF* ← 0
  EDX:EAX ← *Opérande Destination*
FIN SI

Mnémonique

Instruction	Opcode	Description
CMPXCHG8B *mem64*	0Fh C7h /1 m64	Compare la paire de registres EDX:EAX à l'emplacement mémoire de 64 bits. S'ils sont égale, alors fixe le drapeau de zéro à 1 et copie ECX:EBX dans l'emplacement

		mémoire. Autrement, copie l'emplacement mémoire dans EDX:EAX et efface le drapeau de zéro.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	L'instruction CMPXCHG8B n'est pas supporté, lequel est indiquer par le bit 8 du registre EDX de l'instruction CPUID, fonction 0000_0001h ou 8000_0001h.
			X	L'opérande est un registre.
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CMPXCHG

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 236 à 238.*

Assembleur 80x86 CMPXCHG16B

x86-64+

Compare and Exchange Sixteen Bytes

Syntaxe

CMPXCHG16B *dest*

Description

Cette instruction permet de comparer un nombre de 128 bits et les échanges si la condition est vrai.

Algorithme

SI RDX:RAX = *dest* **ALORS**
  ZF ← 1
  dest ← RCX:RBX
SINON
  ZF ← 0
  RDX:RAX ← *dest*
FIN SI

Mnémonique

Instruction	Opcode	Description
CMPXCHG16B *mem128*	0Fh C7h /1 *m128*	Compare la paire de registres EDX:EAX à l'emplacement mémoire de 128 bits. S'ils sont égale, alors fixe le drapeau de zéro à 1 et copie RCX:RBX dans l'emplacement mémoire. Autrement, copie l'emplacement mémoire dans

		RDX:RAX et efface le drapeau de zéro.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	L'instruction CMPXCHG16B n'est pas supporté, lequel est indiquer par le bit 13 du registre ECX de l'instruction CPUID, fonction 0000_0001h.
			X	L'opérande est un registre.
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou

				n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
			X	L'opérande mémoire pour l'instruction CMPXCHG16B n'est pas aligné avec une limite de 16 octets.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement

				est activé

Exemple

Cet exemple permet de tester la valeur d'un octet de deux adresses :

```
1. MOVZX RBX,Byte Ptr [RAX]
2. MOVZX RCX,Byte Ptr [RAX]
3. MOVZX RDX,Byte Ptr [RAX]
4. MOVZX RAX,Byte Ptr [RDI]
5. CMPXCHG16B OWord Ptr [RAX]
```

Voir également

Instruction assembleur 80x86 - Instruction CMPXCHG

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 236 à 238.

Assembleur 80x86 — COMISD

INTEL Pentium III+

Compare Scalar Ordered Double-Precision Floating-Point Values and Set EFLAGS

Syntaxe

COMISD *destination,source*

Description

Cette instruction permet de comparer deux valeurs réel de double-précision dans la partie faible du quadruple mot de deux opérande et fixe les drapeaux de ZF, PF et FC du registre EFLAGS en fonction du résultat (non-ordonnée, supérieur à, inférieur ou égal).

Algorithme

```
EVALUER CAS (OrderedCompare(destination(63..0) <> source(63..0))) DE
  CAS UNORDERED: ZF,PF,CF ← 111b
  CAS GREATER_THAN: ZF,PF,CF ← 000b
  CAS LESS_THAN: ZF,PF,CF ← 001b
  CAS EQUAL: ZF,PF,CF ← 100b
FIN EVALUER CAS
OF,AF,SF ← 0
```

Mnémonique

Instruction	Opcode	Description
COMISD *xmm1,xmm2/m64*	66h 0Fh 2Fh /r	Cette instruction permet de comparer deux valeurs réel de double-précision dans la partie faible du quadruple mot de deux opérande et fixe les drapeaux de ZF, PF et FC du

		registre EFLAGS en fonction du résultat (non-ordonnée, supérieur à, inférieur ou égal).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 239 à 241.

Assembleur 80x86

INTEL Pentium III+

COMISS

Compare Scalar Ordered Single-Precision Floating-Point Values and Set EFLAGS

Syntaxe

COMISS *destination,source*

Description

Cette instruction permet de comparer deux valeurs réel de simple-précision dans la partie faible du quadruple mot de deux opérande et fixe les drapeaux de ZF, PF et FC du registre EFLAGS en fonction du résultat (non-ordonnée, supérieur à, inférieur ou égal).

Algorithme

```
EVALUER CAS (OrderedCompare(destination(31..0) <> source(31..0))) DE
  CAS UNORDERED: ZF,PF,CF ← 111b
  CAS GREATER_THAN: ZF,PF,CF ← 000b
  CAS LESS_THAN: ZF,PF,CF ← 001b
  CAS EQUAL: ZF,PF,CF ← 100b
FIN EVALUER CAS
OF,AF,SF ← 0
```

Mnémonique

Instruction	Opcode	Description
COMISS *xmm1,xmm2/m32*	0Fh 2Fh /r	Cette instruction permet de comparer deux valeurs réel de simple-précision dans la partie faible du quadruple mot de deux opérande et fixe les drapeaux de ZF, PF et FC du

		registre EFLAGS en fonction du résultat (non-ordonnée, supérieur à, inférieur ou égal).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 242 à 244.*

Assembleur 80x86	**CPUID**
INTEL Pentium	*Central Processor Unit IDentificator*

Syntaxe

CPUID

Description

Cette instruction retourne le code d'identification du microprocesseur dans les 4 registres suivants: *EAX*, *EBX*, *ECX*, *EDX*. Tout d'abord, si EAX = 0 lors de l'appel alors EAX retourne le nombre maximal d'entrée d'*EAX* et *EBX:EDX:ECX* retourne la chaîne de caractères "*GenuineIntel*" si s'est un microprocesseur du fabricant *INTEL* (soit EBX = "*Genu*", EDX = "*ineI*", ECX = "*ntel*"). Si *EAX* = 1 lors de l'appel alors *EAX* contient le drapeau de fourniture fixée si les instructions *CMPXCHG8B*, *CMOV*, *FCMOV*, *FCMOVB* sont supportée. Le bit 23 est fixée si les instructions *MMX* sont supportés. Si *EAX* = 2 lors de l'appel alors les 4 registres: *EAX*, *EBX*, *ECX* et *EDX* contient les informations concernant le cache et les *TLBs* (*Translation Lookahead Buffers*).

Algorithme

```
ÉVALUER EAX
  CAS 0:
    EAX ← hv
    EBX ← chaîne de caractères d'identification du fabricant :
      Intel: 756E6547h ("Genu")
      AMD: 68747541h ("Auth")
      Cyrix: 69727943h ("Cyri")
    EDX ← chaîne de caractères d'identification du fabricant :
      Intel: 49656E69h ("ineI")
      AMD: 69746E65h ("enti")
      Cyrix: 736E4978h ("xIns")
    ECX ← chaîne de caractères d'identification du fabricant :
      Intel: 6C65746Eh ("ntel")
      AMD: 444D4163h ("cAMD")
      Cyrix: 64616574h ("tead")
```

CAS 1:
EAX[0 à 3] ← Identification de niveau
SI *Intel* ou *AMD* **ALORS**
EAX[4 à 7] ← Modèle
EAX[8 à 11] ← Famille
EAX[12 à 31] ← réservée
SINON Cyrix **ALORS**
EAX[8 à 15] ← 6
FIN SI
EBX ← 0
ECX ← Drapeau de fourniture supplémentaire :
Bit[13] ← Instruction CMPXCHG16B supporté?
Bit[27] ← Instruction OSXSAVE (XGETBV, XRSTOR, XSAVE, XSETBV) supporté?
EDX ← Drapeau de fourniture:
Bit[0] ← FPU: Coprocesseur mathématique présent
Bit[1] ← *CYRIX*: VME: Supporte le mode *Virtuel 8086*
Bit[2] ← DE: Point d'arrêt aux *Entrées/Sorties*
Bit[3] ← *CYRIX*: PSE: Extension de taille de page
Bit[4] ← TSC: Compteur "*Time Stamp*"
Bit[5] ← MSR: *CPU* de style *Pentium*
Bit[6] ← *CYRIX*: PTE: Extension d'adresse physique, *INTEL*: PAE: Extension d'adresse physique
Bit[7] ← MCE: Vérification d'exception machine
Bit[8] ← CX8: *CMPXCHG8B* supportée?
Bit[9] ← *CYRIX*: *APIC*
Bit[11] ← SEP: Fourniture de d'appel rapide système (Pentium Pro+)
Bit[12] ← MTRR: *CYRIX*: Type de registres de rangé mémoire
Bit[13] ← PGE: Bit global *PTE*
Bit[14] ← CYRIX: MCA: Vérification de l'architecture machine supporté?
Bit[15] ← CMOV: Les instructions *CMOV*, *FCMOV*,... sont supportés?
Bit[16] ← PAT: Table d'attribut de page
Bit[17] ← PSE36: Microprocesseur support des pages de 4 Mo pour l'accès à la haute mémoire 2 Go
Bit[18] ← SN: Support le numéro de série du microprocesseur
Bit[19] ← Instruction *CLFLUSH* supporté?
Bit[23] ← MMX: Instruction *MMX* supporté?
Bit[24] ← FXSR: Sauvegarde/restauration rapide du microprocesseur (IA MMX-2)
Bit[25] ← SIMD: Extension de flux SIMD (IA MMX-2)
Bit[26] ← SSE2: Instruction LFENCE, MFENCE et MOVNTI supporté?
CAS 2:
SI (Pentium III ou plus) ou (Cyrix MediaGX MMX Enhanced) **ALORS**

EAX, *EBX*, *ECX*, *EDX* ← Les 4 type de cache de l'architecture du microprocesseur:
00h: Aucun
01h: Instruction TLB, 4 Ko par page, 4 routes, 64 entrées
02h: Instruction TLB, 4 Mo par page, 4 routes, 4 entrées
03h: Donnée TLB, 4 Ko par page, 4 routes, 64 entrées
04h: Donnée TLB, 4 Mo par page, 4 routes, 8 entrées
06h: Cache d'instruction, 8 Ko, 4 routes, 32 octets par ligne
08h: Cache d'instruction, 16 Ko, 4 routes, 32 octets par ligne
0Ah: Cache d'instruction, 8 Ko, 2 routes, 32 octets par ligne
0Ch: Cache d'instruction, 16 Ko, 2 routes, 32 octets par ligne
40h: Pas de cache L2
41h: Cache L2 unifié, 32 octets par ligne, 4 routes, 128 Ko
42h: Cache L2 unifié, 32 octets par ligne, 4 routes, 256 Ko
43h: Cache L2 unifié, 32 octets par ligne, 4 routes, 512 Ko
44h: Cache L2 unifié, 32 octets par ligne, 4 routes, 1 Mo
45h: Cache L2 unifié, 32 octets par ligne, 4 routes, 2 Mo
SI Cyrix MediaGX MMX Enhanced **ALORS**
70h: Entrée 32 bits TLB, 4 routes, Cache de 4 Ko
80h: Cache de 4 routes associative L1, 16 octets par ligne
FIN SI
SINON
EAX ← non-défini
EBX ← non-défini
ECX ← non-défini
EDX ← non-défini
FIN SI
CAS 3:
SI (Pentium III ou plus) **ALORS**
EDX:ECX ← Partie basse de 64 bits du numéro de série de 96 bits du microprocesseur
SINON
EAX ← non-défini
EBX ← non-défini
ECX ← non-défini
EDX ← non-défini
FIN SI
CAS 80000000h:
SI (AMD K5 sauf SSA/5, AMD K6 ou Cyrix GXm) ou (Pentium 4 ou plus) **ALORS**
EAX ← Valeur maximal de la fonction contenu dans le registre *EAX* reconnu par l'instruction CPUID (Exemple AMD 5k86 (K5) = 80000005h)
SINON
EAX ← non-défini

FIN SI
EBX ← non-défini
ECX ← non-défini
EDX ← non-défini
CAS 80000001h:
SI (AMD K5, AMD K6, Cyrix GXm ou IDT Winchip 2) **ALORS**
EAX ← Signature du microprocesseur AMD
0000051Xh: AMD 5k86 (K5 sauf SSA/5)
0000066Xh: AMD 6k86 (K6)
EDX ← Drapeaux des fournitures étendues:
Bit[0] ← FPU: Coprocesseur arithmétique include dans la puce du microprocesseur
Bit[1] ← VME: Extension de mode Virtual présent
Bit[2] ← DE: Extensions de déboguage
Bit[3] ← PSE: CPU support la tailles de pages de 4 Mo
Bit[4] ← TSC: TSC présent (Voir l'instruction **RDTSC**)
Bit[5] ← MSR: CPU est compatible MSR du K5 ou compatibilité MSR du Cyrix
Bit[6] ← PAE: Extensions de page d'adresse
Bit[7] ← MCE: Vérification d'exception machine
Bit[8] ← CX8: Support l'instruction **CMPXCHG8B**
Bit[9] ← APIC: CPU a un APIC local (doit être activé)
Bit[10] ← Réservé
Bit[11] ← Instruction **SYSCALL** et **SYSRET**
Bit[12] ← MTRR: Registres de type de rang mémoire (*Memory Type Range Registers*)
Bit[13] ← Extensions de pagination global (PTE-PGE)
Bit[14] ← MCA: Architecture de vérification machine
Bit[15] ← CMOV: Le CPU support les instructions **CMOV**
Bit[16] ← FCMOV: Le CPU support les instructions **FCMOV**
Bit[17] ← PSE: Extension de taille de Page
Bit[21..18] ← Réservé
Bit[22] ← MMXE: CPU Support les extensions d'instructions MMX (AMD Athlon)
Bit[23] ← MMX: CPU support IA MMX
Bit[24] ← (Cyrix) Cyrix supporte l'extension MMX
Bit[24] ← (AMD) Support les instructions FXSAVE/FXRSTOR
Bit[29..25] ← Réservé
Bit[30] ← CPU support les instructions étendue 3DNow!
Bit[31] ← Support l'AMD 3DNow!
SINON
EAX ← non-défini
EDX ← non-défini
FIN SI
EBX ← non-défini

ECX ← Drapeau de fourniture supplémentaire:
Bit[0] ← Instruction **LAHF** est supporté en mode 64 bits?
Bit[2] ← Les instructions SVM (**CLGI**,...) sont supportés?
Bit[5] ← Instruction **LZCNT** supporté?
Bit[8] ← Instruction **PREFETCH** du «3DNow! Prefetch» supporté?
CAS 80000002h,80000003h,80000004h:
SI (AMD K5, AMD K6 ou Cyrix GXm) **ALORS**
EAX, EBX, ECX, EDX ← Nom du CPU sous forme d'une chaine de caractères (si AMD K5, en format BIG-Endian)
SINON
EAX ← non-défini
EBX ← non-défini
ECX ← non-défini
EDX ← non-défini
FIN SI
CAS 80000008h:
EAX ← Demande des informations sur la taille des adresses et le compteur de coeur physique:
Bit[23..16] ← Demande la taille maximal physique invité d'octets en bits
Bit[15..8] ← Demande la taille maximal linéaire d'une adresse d'octets en bits
Bit[7..0] ← Demande la taille maximal physique d'octets en bits
ECX ← NC: Demande le nombre de coeur que comprendre le microprocesseur - 1.
AUTREMENT:
EAX ← non-défini
EBX ← non-défini
ECX ← non-défini
EDX ← non-défini
FIN ÉVALUER

Remarque

Voici la table des fournitures standard retourner par le registre EAX=1 de l'instruction *CPUID* en fonction des différents fabricants:

Registre EDX	**Nom**	**Cyrix 6x**	**Cyrix 6x**	**Cyrix GXm**	**Cyrix 6x86M**	**Cyrix GXm**	**Cyrix GXm**	**AMD DC**	**AMD DC**	**AMD DK**	**AMD DK**	**AMD DK6**	**AMD DK6**	**AMD Athl**	**INTEL Penti**	**INTEL Penti**	**INTEL Penti**	**INTEL Penti**	**INTEL Penti**	**INTEL Penti**

		x86	**86L**	**X**	**X**	**II**	**Xm**	**6**	**2**	**5**	**6**	**-2**	**-III**	**on**	**um**	**umMMX**	**umPro**	**umII**	**umIII**	**umCeleron**
Bit[0]	FPU	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Bit[1]	VME	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Bit[2]	DE	-	+	-	+	+	-	+	+	+	+	+	+	+	+	+	+	+	+	+
Bit[3]	PSE	-	-	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+
Bit[4]	TSC	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Bit[5]	MSR	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Bit[6]	PAE	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	+	+	+	+
Bit[7]	MCE	-	-	-	-	-	-	+	+	+	+	+	+	+	+	+	+	+	+	+
Bit[8]	CX8	-	+	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Bit[9	API	-	-	-	-	-	-	-	-	-	-	-	x	+	(1)	(1)	(1)	(1)	(1)	(1)

]	C																			
Bit[10]	res	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	?	-
Bit[11]	SEP	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	+	+	+	+
Bit[12]	MTRR	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	+	+	+	+
Bit[13]	PGE	-	-	-	+	+	-	-	-	-	-	-	-	+	-	-	+	+	+	+
Bit[14]	MCA	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	+	+	+	+
Bit[15]	CMOV	-	-	-	+	+	+	-	-	-	-	-	-	+	-	-	+	+	+	+
Bit[16]	PAT	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	(2)	+	+
Bit[17]	PSE36	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(2)	+	+
Bit[18]	SN	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
Bit[23]	MM	-	-	-	+	+	+	+	+	-	+	+	+	+	+	+	+	+	+	+

	X																			
Bit[24]	FXSR	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	(2)	+	+
Bit[25]	SIMD	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-

(1) = Intel produit des puces avec APIC et sans APIC.

(2) = Il commence à supporter les coeurs «*Deschutes*»

Mnémonique

Instruction	Opcode	Description
CPUID	0Fh A2h	Retourne les informations à propos du microprocesseur et sa compatiblité. Le registre EAX correspond à un numéro de fonction et les données sont retournées dans les registres EAX, EBX, ECX et EDX.

Exceptions

Aucune

Exemple

Cet exemple permet de tester la présence de l'instruction «*CMPXCHG16B*» avant de l'utiliser :

```
1.  MOV EAX,1
2.  CPUID
3.  BT ECX, 13  ; CPUID.1:ECX.CMPXCHG16B[bit 13] = 1: Est-
    ce que le CMPXCHG16B est disponible et actif pour l'utilisation d'application ?
4.  JNC @not_supported
5.   ; Mettre quelques instructions avant ici
```

```
    ; ...
    CMPXCHG16B OWord Ptr [RAX]
    ; Mettre quelques instructions après ici
    ; ...
@not_supported:
```

Références

AMD CPUID Specification, Edition Advanced Micro Devices, Revision 2.28, April 2008, Publication No. 25481.
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 103 à 104.
The Undocumented PC: A programmer's Guide to I/O, CPUs, and Fixed Memory Areas - Second Edition, Edition Addison-Wesley, Frank van Gilluwe, 1997, ISBN: 0-201-47950-8, page 64 à 67.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 245 à 279.

Assembleur 80x86	**CQO**
x86-64+	*Convert Quadword to Octoword*

Syntaxe

```
CQO
```

Description

Cette instruction permet de convertir le quadruple mot du registre EAX en deux quadruple mot RDX:RAX, le résultat tient sur une taille de 128 bits.

Algorithme

RDX:RAX ← extension des signes de *RAX*

Mnémonique

Instruction	Opcode	Description
CQO	99h	Étendre les signes de EAX dans RAX

Exceptions

Aucune

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 349 à 350.

Assembleur 80x86 CRC32

Nehalem (SSE4.2)+

Accumulate CRC32 Value

Syntaxe

CRC32 *dest*,*source*

Description

Cette instruction débute avec une valeur initiale dans l'opérande de destination, accumule une valeur de code CRC-32 (polynomial de 11EDC6F41h) dans l'opérande source et enregistre le résultat dans l'opérande de destination.

Algorithme

```
MODULE BIT_REFLECT64
  dest(63..0) ← source(0..63)

MODULEMODULE BIT_REFLECT32
  dest(31..0) ← source(0..31)

MODULE BIT_REFLECT16
  dest(15..0) ← source(0..15)

MODULE BIT_REFLECT8
  dest(7..0) ← source(0..7)

SI taille de l'opérande source = 64 bits ET taille de l'opérande de destination = 64 bits ALORS
  TEMP1(63..0) ← BIT_REFLECT64 (source(63..0))
  TEMP2(31..0) ← BIT_REFLECT32 (dest(31..0))
  TEMP3(95..0) ← TEMP1(63..0) << 32
  TEMP4(95..0) ← TEMP2(31..0) << 64
  TEMP5(95..0) ← TEMP3(95..0) XOR TEMP4(95..0)
  TEMP6(31..0) ← TEMP5(95..0) MOD2 11EDC6F41h
  DEST(31..0) ← BIT_REFLECT (TEMP6(31..0))
```

```
  DEST(63..32) ← 00000000h
SINON SI taille de l'opérande source = 32 bits ET taille de l'opérande de destination = 32 bits
ALORS
  TEMP1(31..0) ← BIT_REFLECT32 (source(31..0))
  TEMP2(31..0) ← BIT_REFLECT32 (dest(31..0))
  TEMP3(63..0) ← TEMP1(31..0) << 32
  TEMP4(63..0) ← TEMP2(31..0) << 32
  TEMP5(63..0) ← TEMP3(63..0) XOR TEMP4(63..0)
  TEMP6(31..0) ← TEMP5(63..0) MOD2 11EDC6F41h
  DEST(31..0) ← BIT_REFLECT (TEMP6(31..0))
SINON SI taille de l'opérande source = 16 bits ET taille de l'opérande de destination = 32 bits
ALORS
  TEMP1[15-0] ← BIT_REFLECT16 (source(15..0))
  TEMP2[31-0] ← BIT_REFLECT32 (dest(31..0))
  TEMP3[47-0] ← TEMP1(15..0) << 32
  TEMP4[47-0] ← TEMP2(31..0) << 16
  TEMP5[47-0] ← TEMP3(47..0) XOR TEMP4(47..0)
  TEMP6[31-0] ← TEMP5(47..0) MOD2 11EDC6F41h
  DEST[31-0] ← BIT_REFLECT (TEMP6(31..0))
SINON SI taille de l'opérande source = 8 bits ET taille de l'opérande de destination = 64 bits
ALORS
  TEMP1[7-0] ← BIT_REFLECT8(source(7..0))
  TEMP2[31-0] ← BIT_REFLECT32 (dest(31..0))
  TEMP3[39-0] ← TEMP1(7..0) << 32
  TEMP4[39-0] ← TEMP2(31..0) << 8
  TEMP5[39-0] ← TEMP3(39..0) XOR TEMP4(39..0)
  TEMP6[31-0] ← TEMP5(39..0) MOD2 11EDC6F41h
  DEST[31-0] ← BIT_REFLECT (TEMP6(31..0))
  DEST[63-32] ← 00000000h
SINON SI taille de l'opérande source = 8 bits ET taille de l'opérande de destination = 32 bits
ALORS
  TEMP1[7-0] ← BIT_REFLECT8(source(7..0))
  TEMP2[31-0] ← BIT_REFLECT32 (dest(31..0))
  TEMP3[39-0] ← TEMP1(7..0) << 32
  TEMP4[39-0] ← TEMP2(31..0) << 8
  TEMP5[39-0] ← TEMP3(39..0) XOR TEMP4(39..0)
  TEMP6[31-0] ← TEMP5(39..0) MOD2 11EDC6F41h
  DEST[31-0] ← BIT_REFLECT (TEMP6(31..0))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CRC32 *r32*, *r/m8*	F2h 0Fh 38h F0h /r	Cette instruction débute avec une valeur initiale dans l'opérande de destination, accumule une valeur de code CRC-32 (polynomial de 11EDC6F41h) dans l'opérande source et enregistre le résultat dans l'opérande de destination.
CRC32 *r32*, *r/m8*	F2h *REX* 0Fh 38h F0h /r	Cette instruction débute avec une valeur initiale dans l'opérande de destination, accumule une valeur de code CRC-32 (polynomial de 11EDC6F41h) dans l'opérande source et enregistre le résultat dans l'opérande de destination.
CRC32 *r32*, *r/m16*	F2h 0Fh 38h F1h /r	Cette instruction débute avec une valeur initiale dans l'opérande de destination, accumule une valeur de code CRC-32 (polynomial de 11EDC6F41h) dans l'opérande source et enregistre le résultat dans l'opérande de destination.
CRC32 *r32*, *r/m32*	F2h 0Fh 38h F1h /r	Cette instruction débute avec une valeur initiale dans l'opérande de destination, accumule une valeur de code CRC-32 (polynomial de 11EDC6F41h) dans l'opérande source et enregistre le résultat dans l'opérande de destination.
CRC32 *r64*, *r/m8*	F2h *REX.W* 0Fh 38h F0h /r	Cette instruction débute avec une valeur initiale dans l'opérande de destination, accumule une valeur de code CRC-32 (polynomial de

		11EDC6F41h) dans l'opérande source et enregistre le résultat dans l'opérande de destination.
CRC32 *r64, r/m64*	F2h *REX.W* 0Fh 38h F1h /r	Cette instruction débute avec une valeur initiale dans l'opérande de destination, accumule une valeur de code CRC-32 (polynomial de 11EDC6F41h) dans l'opérande source et enregistre le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 280 à 283.*

Assembleur 80x86 — CVTDQ2PD

INTEL Pentium 4+

Convert Packed Doubleword Integers to Packed Double-Precision Floating-Point Values

Syntaxe

CVTDQ2PD *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de double précision.

Algorithme

dest(0..63) ← ConvertIntegerToDouble(*source*(0..31))
dest(64..127) ← ConvertIntegerToDouble(*source*(32..63))

Mnémonique

Instruction	Opcode	Description
CVTDQ2PD *xmm1, xmm2/m64*	F3h 0Fh E6h	Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 284 à 285.

Assembleur 80x86 — CVTDQ2PS

INTEL Pentium 4+

Convert Packed Doubleword Integers to Packed Single-Precision Floating-Point Values

Syntaxe

CVTDQ2PS *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de simple précision.

Algorithme

dest(0..31) ← ConvertIntegerToFloat(*source*(0..31))
dest(32..63) ← ConvertIntegerToFloat(*source*(32..63))
dest(64..95) ← ConvertIntegerToFloat(*source*(64..95))
dest(96..127) ← ConvertIntegerToFloat(*source*(96..127))

Mnémonique

Instruction	Opcode	Description
CVTDQ2PS *xmm1, xmm2/m128*	0Fh 5Bh /r	Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 286 à 288.

Assembleur 80x86 — CVTPD2DQ

INTEL Pentium 4+

Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

Syntaxe

CVTPD2DQ *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en double mot entier et fixe à 0 le reste du paquet.

Algorithme

dest(0..31) ← ConvertDoubleToInteger(Source(0..63))
dest(32..63) ← ConvertDoubleToInteger(Source(64..127))
dest(64..127) ← 0

Mnémonique

Instruction	Opcode	Description
CVTPD2DQ *xmm1, xmm2/m128*	F2h 0Fh E6h	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en double mot entier et fixe à 0 le reste du paquet.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 289 à 291.

Assembleur 80x86 — CVTPD2PI

INTEL Pentium 4+

Convert Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

Syntaxe

CVTPD2PI *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en double mot entier.

Algorithme

dest(0..31) ← ConvertDoubleToInteger(Source(0..63))
dest(32..63) ← ConvertDoubleToInteger(Source(64..127))

Mnémonique

Instruction	Opcode	Description
CVTPD2PI *mm, xmm/m128*	66h 0Fh 2Dh /r	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en double mot entier.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 292 à 294.

Assembleur 80x86 — CVTPD2PS

INTEL Pentium 4+

Convert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point Values

Syntaxe

CVTPD2PS *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en valeurs réel de simple précision et fixe à 0 le reste du paquet.

Algorithme

dest(0..31) ← ConvertDoubleToFloat(Source(0..63))
dest(32..63) ← ConvertDoubleToFloat(Source(64..127))
dest(64..127) ← 0

Mnémonique

Instruction	Opcode	Description
CVTPD2PS *xmm1, xmm2/m128*	66h 0Fh 5Ah /r	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de double précision en valeurs réel de simple précision et fixe à 0 le reste du paquet.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 295 à 297.

Assembleur 80x86 — CVTPI2PD

INTEL Pentium 4+

Convert Packed Double-Precision Floating-Point Values to Packed Single-Precision Floating-Point Values

Syntaxe

CVTPI2PD *dest*,*source*

Description

Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de double précision compacté.

Algorithme

dest(0..63) ← ConvertIntegerToDouble(*source*(0..31))
dest(64..127) ← ConvertIntegerToDouble(*source*(32..63))

Mnémonique

Instruction	Opcode	Description
CVTPI2PD *xmm, mm/m64*	66h 0Fh 2Ah /r	Cette instruction permet d'effectuer la conversion d'un paquet de double mot entier en valeurs réel de double précision compacté.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 297 à 300.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

CVTPI2PS

Convert Packed Integer to Packed Single

Syntaxe

CVTPI2PS *dest,source*

Description

Cette instruction permet de convertir un entier de format double mot en une valeur réel de simple précision compacté.

Algorithme

dest(31..0) ← (réel) *source*(31..0)
dest(63..32) ← (réel) *source*(63..32)

Mnémonique

Instruction	Opcode	Description
CVTPI2PS *xmm,mm/m64*	0Fh 2Ah /r	Cette instruction permet de convertir un entier de format double mot en une valeur réel de simple précision compacté.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 301 à 302.

Assembleur 80x86 — CVTPS2DQ

INTEL Pentium 4+

Convert Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

Syntaxe

CVTPS2DQ *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de simple précision en paquet de double mots entiers.

Algorithme

dest(0..31) ← ConvertFloatToInteger(*source*(0..31))
dest(32..63) ← ConvertFloatToInteger(*source*(32..63))
dest(64..95) ← ConvertFloatToInteger(*source*(64..95))
dest(96..127) ← ConvertFloatToInteger(*source*(96..127))

Mnémonique

Instruction	Opcode	Description
CVTPS2DQ *xmm1, xmm2/m128*	66h 0Fh 5Bh /r	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de simple précision en paquet de double mots entiers.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 303 à 306.

Assembleur 80x86 — CVTPS2PD

INTEL Pentium 4+

Convert Packed Single-Precision Floating-Point Values to Packed Double-Precision Floating-Point Values

Syntaxe

CVTPS2PD *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de simple précision en paquet de valeurs réel de double précision.

Algorithme

dest(0..63) ← ConvertFloatToDouble(*source*(0..31))
dest(64..127) ← ConvertFloatToDouble(*source*(32..63))

Mnémonique

Instruction	Opcode	Description
CVTPS2PD *xmm1, xmm2/m64*	0Fh 5Ah /r	Cette instruction permet d'effectuer la conversion d'un paquet de valeurs réel de simple précision en paquet de valeurs réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 307 à 309.

Assembleur 80x86 — CVTPS2PI

INTEL Pentium III+ (KNI/MMX2)+

Convert Packed Single to Packed Integer

Syntaxe

CVTPS2PI *dest,source*

Description

Cette instruction permet de convertir une valeur réel de simple précision compacté en un entier de format double mot.

Algorithme

dest(31..0) ← (entier) *source*(31..0) *dest*(63..32) ← (entier) *source*(63..32)

Mnémonique

Instruction	Opcode	Description
CVTPS2PI *mm,xmm/m64*	0Fh 2Dh /r	Cette instruction permet de convertir une valeur réel de simple précision compacté en un entier de format double mot.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 310 à 312.

Assembleur 80x86 — CVTSD2SI

INTEL Pentium 4+

Convert Scalar Double-Precision Floating-Point Value to Doubleword Integer

Syntaxe

CVTSD2SI *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'une valeur réel de simple précision en valeur entier de double mots.

Algorithme

dest(0..31) ← ConvertDoubleToInteger(*source*(0..63))

Mnémonique

Instruction	Opcode	Description
CVTSD2SI *r32, xmm/m64*	F2h 0Fh 2Dh /r	Cette instruction permet d'effectuer la conversion d'une valeur réel de simple précision en valeur entier de double mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 313 à 315.

Assembleur 80x86

CVTSD2SS

INTEL Pentium 4+

Convert Scalar Double-Precision Floating-Point Value to Scalar Single-Precision Floating-Point Value

Syntaxe

CVTSD2SS *dest,source*

Description

Cette instruction permet d'effectuer la conversion d'une valeur réel de double précision en valeur réel de simple précision.

Algorithme

dest(0..31) ← ConvertDoubleToFloat(*source*(0..63))

Mnémonique

Instruction	Opcode	Description
CVTSD2SS xmm1, xmm2/m64	F2h 0Fh 5Ah /r	Cette instruction permet d'effectuer la conversion d'une valeur réel de double précision en valeur réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 316 à 318.

Assembleur 80x86

CVTSI2SD

INTEL Pentium 4+

Convert Doubleword Integer to Scalar Double-Precision Floating-Point Value

Syntaxe

CVTSI2SD *dest,source*

Description

Cette instruction permet de convertir un entier de format double mot en une valeur réel de double précision.

Algorithme

dest(0..63) ← ConvertIntegerToDouble(*source*(0..31))

Mnémonique

Instruction	Opcode	Description
CVTSI2SD *xmm, r/m32*	F2h 0Fh 2Ah /r	Cette instruction permet de convertir un entier de format double mot en une valeur réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 319 à 321.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

CVTSI2SS

Convert Doubleword Integer to Scalar Single-Precision Floating-Point Value

Syntaxe

CVTSI2SS *dest,source*

Description

Cette instruction permet de convertir un entier de format double mot en une valeur réel de simple précision.

Algorithme

dest(31..0) ← (réel) *source*(31..0)

Mnémonique

Instruction	Opcode	Description
CVTSI2SS *xmm,r/m32*	F3h 0Fh 2Ah /r	Cette instruction permet de convertir un entier de format double mot en une valeur réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 322 à 324.

Assembleur 80x86

INTEL Pentium 4+

CVTSS2SD

Convert Scalar Single-Precision Floating-Point Value to Scalar Double-Precision Floating-Point Value

Syntaxe

CVTSS2SD *dest,source*

Description

Cette instruction permet de convertir une valeur réel de simple précision en une valeur réel de double précision.

Algorithme

dest(0..63) ← ConvertFloatToDouble(*source*(0..31))

Mnémonique

Instruction	Opcode	Description
CVTSS2SD *xmm1, xmm2/m32*	F3h 0Fh 5Ah /r	Cette instruction permet de convertir une valeur réel de simple précision en une valeur réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 325 à 327.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

CVTSS2SI

Convert Single-Precision Floating-Point to Scalar Integer

Syntaxe

CVTSS2SI *dest,source*

Description

Cette instruction permet de convertir une valeur réel de simple précision en un entier de format double mot.

Algorithme

dest(31..0) ← (entier) *source*(31..0)

Mnémonique

Instruction	Opcode	Description
CVTSS2SI *r32,xmm/m32*	F3h 0Fh 2Dh /r	Cette instruction permet de convertir une valeur réel de simple précision en un entier de format double mot.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 328 à 330.

Assembleur 80x86

CVTTPD2DQ

INTEL Pentium 4+

Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

Syntaxe

CVTTPD2DQ *dest,source*

Description

Cette instruction permet de convertir un paquet de valeur réel de double précision avec en un paquet d'entier de format double mot tronquer et fixe à 0 le reste du paquet.

Algorithme

dest(0..31) ← ConvertDoubleToIntegerTruncate(*source*(0..63))
dest(32..63) ← ConvertDoubleToIntegerTruncate(*source*(64..127))
dest(64..127) ← 0

Mnémonique

Instruction	Opcode	Description
CVTTPD2DQ *xmm1, xmm2/m128*	66h 0Fh E6h /r	Cette instruction permet de convertir un paquet de valeur réel de double précision avec en un paquet d'entier de format double mot tronquer et fixe à 0 le reste du paquet.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 331 à 333.

Assembleur 80x86 — CVTTPD2PI

INTEL Pentium 4+

Convert with Truncation Packed Double-Precision Floating-Point Values to Packed Doubleword Integers

Syntaxe

CVTTPD2PI *dest*,*source*

Description

Cette instruction permet de convertir une valeur réel de double précision compacté avec tronquage en un entier de format double mot compacté.

Algorithme

dest(0..31) ← ConvertDoubleToIntegerTruncate(*source*(0..63))
dest(32..63) ← ConvertDoubleToIntegerTruncate(*source*(64..127))

Mnémonique

Instruction	Opcode	Description
CVTTPD2PI *mm, xmm/m128*	66h 0Fh 2Ch /r	Cette instruction permet de convertir une valeur réel de double précision compacté avec tronquage en un entier de format double mot compacté.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 334 à 336.

Assembleur 80x86

INTEL Pentium 4+

CVTTPS2DQ

Convert with Truncation Packed Single-Precision Floating-Point Values to Packed Doubleword Integers

Syntaxe

CVTTPS2DQ *dest,source*

Description

Cette instruction permet de convertir un paquet de valeur réel de simple précision avec en un paquet d'entier de format double mot tronquer.

Algorithme

dest(0..31) ← ConvertFloatToIntegerTruncate(*source*(0..31))
dest(32..63) ← ConvertFloatToIntegerTruncate(*source*(32..63))
dest(64..95) ← ConvertFloatToIntegerTruncate(*source*(64..95))
dest(96..127) ← ConvertFloatToIntegerTruncate(*source*(96..127))

Mnémonique

Instruction	Opcode	Description
CVTTPS2DQ *xmm1, xmm2/m128*	F3h 0Fh 5Bh /r	Cette instruction permet de convertir un paquet de valeur réel de simple précision avec en un paquet d'entier de format double mot tronquer.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 337 à 339.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)+

CVTTPS2PI

Convert Packed Single with truncation to Packed Integer

Syntaxe

CVTTPS2PI *dest*,*source*

Description

Cette instruction permet de convertir une valeur réel de simple précision compacté avec tronquage en un entier de format double mot compacté.

Algorithme

dest(31..0) ← (entier) *source*(31..0)
dest(63..32) ← (entier) *source*(63..32)

Mnémonique

Instruction	Opcode	Description
CVTTPS2PI *mm,xmm/m64*	0Fh 2Ch /r	Cette instruction permet de convertir une valeur réel de simple précision compacté avec tronquage en un entier de format double mot compacté.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 340 à 342.

Assembleur 80x86 — CVTTSD2SI

INTEL Pentium 4+

Convert with Truncation Scalar Double-Precision Floating-Point Value to Signed Doubleword Integer

Syntaxe

CVTTSD2SI *dest,source*

Description

Cette instruction permet de convertir une valeur réel de double précision avec en un entier de format double mot tronquer.

Algorithme

dest(0..31) ← ConvertDoubleToIntegerTruncate(*source*(0..63))

Mnémonique

Instruction	Opcode	Description
CVTTSD2SI *r32, xmm/m64*	F2h 0Fh 2Ch /r	Cette instruction permet de convertir une valeur réel de double précision avec en un entier de format double mot tronquer.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 343 à 345.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

CVTTSS2SI

Convert Single-Precision Floating-Point with truncation to Scalar Integer

Syntaxe

CVTTSS2SI *dest,source*

Description

Cette instruction permet de convertir un entier de format double mot avec tronquage en une valeur réel de simple précision.

Algorithme

dest(31..0) ← (entier) *source*(31..0)

Mnémonique

Instruction	Opcode	Description
CVTTSS2SI *r32,xmm/m32*	F3h 0Fh 2Ch /r	Cette instruction permet de convertir un entier de format double mot avec tronquage en une valeur réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 345 à 348.

Assembleur 80x86	**CWD**
INTEL 8088+	*Convert Word to Double word*

Syntaxe

```
CWD
```

Description

Cette instruction est l'alternative pour convertir le mot du registre *AX* en un double mot contenu dans le couple de registre *DX* et *AX* par extension du signe. Le bit de signe du registre *AX* est recopié dans les 16 bits du registre *DX*. En somme, cette commande ne présente un intérêt particulier que lorsque l'on manipule des valeurs binaires signées étant donné que le signe du registre *AX* est conservé dans le double mot du couple de registre *DX*:*AX*.

Algorithme

```
SI AX < 8000h ALORS
  DX ← 00000h
FIN SI
SI AX >= 8000h ALORS
  DX ← 0FFFFh
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CWD	99h	Étendre les signes de AX dans DX:AX

Exceptions

Aucune

Voir également

Instruction assembleur 80x86 - Instruction CBW
Instruction assembleur 80x86 - Instruction CWDE
Instruction assembleur 80x86 - Instruction CDQ

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 809
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 404
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 85.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 349 à 350.

Assembleur 80x86	**CWDE**
INTEL 80386+	*Convert WorD to Extended (double word)*

Syntaxe

```
CWDE
```

Description

S'appuyant sur la technique utilisé par son frère *CWD*, l'instruction *CWDE* permet de convertir le mot se trouvant dans *AX* en un double mot, le résultat étant placé dans le registre 32 bits *EAX*. Les 16 bits de poids forts du registre *EAX* sont positionnés à la valeur du bit de poids le plus fort du registre *AX*.

Algorithme

```
SI EAX < 8000h ALORS
  EAX ← EAX ∩ 00000FFFFh
FIN SI
SI EAX >= 8000h ALORS
  EAX ← EAX U 0FFFF0000h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
CWDE	66h 98h	Étendre les signes de AX dans EAX

Exceptions

Aucune

Voir également

Instruction assembleur 80x86 - Instruction CWD
Instruction assembleur 80x86 - Instruction CBW
Instruction assembleur 80x86 - Instruction CDQ

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 187 à 188.

Assembleur 80x86	**DAA**
INTEL 8088+	*Decimal Adjust for Addition*

Syntaxe

```
DAA
```

Description

Cette instruction corrige après coup les retenues lors de la manipulation de valeur *DCB*. Pour y parvenir, la commande convertit le contenu dans le registre *AL* en une valeur *DCB* compactée (même chose dans le registre *AL*).

Algorithme

```
SI ( ( AL ∩ 0Fh ) > 9 U AF = 1 ) ALORS
  AL ← AL + 6
  drapeau AF ← 1
SINON
  drapeau AF ← 0
  SI ( ( AL > 9Fh) U CF = 1 ) ALORS
    AL ← AL + 60h
    drapeau CF ← 1
  SINON
    drapeau CF ← 0
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
DAA	27h	Ajuste les décimal du registre AL (invalide en mode 64 bits)

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction est exécuté en mode 64-bits

Voir également

Instruction assembleur 80x86 - Instruction DAS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 810
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 105.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 351 à 352.

Assembleur 80x86	**DAS**
INTEL 8088+	*Decimal Adjust AL After Soustraction*

Syntaxe

```
DAS
```

Description

Cette instruction offre l'intéressante possibilité de corrige le résultat d'une soustraction de 2 nombres de format *DCB* compactées. Étant donné que le microprocesseur soustrait ces valeurs comme s'il s'agissait de chiffres normales, des débordements se produisent dans les résultats. La commande *DAS* convertit le résultat de la soustraction de 2 valeurs *DCB* compactées en une valeur de format *DCB* compactée.

Algorithme

```
SI ( AF = 1 ) U ( AL ∩ 0Fh ) > 9 ALORS
  AL ← AL - 6
  drapeau AF ← 1
SINON
  drapeau AF ← 0
FIN SI
SI ( CF = 1 ) U ( AL > 9Fh ) ALORS
  AL ← AL - 60h
  drapeau CF ← 1
SINON
  drapeau CF ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
DAS	2Fh	Ajuste les décimal du registre AL après soustraction (invalide en mode

		64 bits)

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction est exécuté en mode 64-bits

Voir également

Instruction assembleur 80x86 - Instruction DAA

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 810
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 106.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 353 à 354.

Assembleur 80x86	**DEC**
INTEL 8088+	*Decrement*

Syntaxe

DEC *opérande*

Description

Cette instruction est un quelque sorte un raccourci de l'instruction *SUB*, en effet celle-ci décrément de 1 le registre ou l'adresse mémoire spécifié, le résultat obtenu est donc identique à celui d'un *SUB* avec 1 comme 2^{ième} opérande.

Algorithme

opérande ← *opérande* - 1

Mnémonique

Instruction	Opcode	Description
DEC *reg/mem8*	FEh /1	Décrémente de 1 le contenu d'un emplacement de registre ou mémoire 8 bits
DEC *reg/mem16*	FFh /1	Décrémente de 1 le contenu d'un emplacement de registre ou mémoire 16 bits
DEC *reg/mem32*	FFh /1	Décrémente de 1 le contenu d'un emplacement de registre ou mémoire 32 bits
DEC *reg/mem64*	FFh /1	Décrémente de 1 le contenu d'un emplacement de registre ou mémoire

		64 bits
DEC *reg16*	(48h + rw)	Décrémente de 1 le contenu d'un registre 16 bits
DEC *reg32*	(48h +rd)	Décrémente de 1 le contenu d'un registre 32 bits

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est

				utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction INC
Instruction assembleur 80x86 - Instruction SUB

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 810
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 404
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 107 à 108.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 355 à 357.

Assembleur 80x86	**DIV**
INTEL 8088+	*Division*

Syntaxe

DIV *Opérande*

Description

L'instruction *DIV* permet d'effectuer une division non-signée (nombre naturel). Le dividende est implicite; il est ajuster en fonction de la taille du diviseur. Le restant est toujours plus petit que le diviseur. Le type de diviseur détermine quel registre l'instruction utilisera:

Taille	Dividende	Diviseur	Quotient	Restant
Octet	AX	*Opérande*	AL	AH
Mot	DX:AX	*Opérande*	AX	DX
Double mot	EDX:EAX	*Opérande*	EAX	EDX

Algorithme

```
SI Opérande = 0 ALORS
  Interruption 0
SINON
  SI Opérande 8 bits ALORS
    AL ← AX ÷ Opérande
    AH ← AX MOD Opérande
  FIN SI

  SI Opérande 16 bits ALORS
    AX ← ( ( DX x 65536 ) + AX ) ÷ Opérande
    DX ← ( ( DX x 65536 ) + AX ) MOD Opérande
  FIN SI
  SI Opérande 32 bits ALORS
```

```
    EAX ← ( ( EDX x 65536 x 65536 ) + EAX ) ÷ Opérande
    EDX ← ( ( EDX x 65536 x 65536 ) + EAX ) MOD Opérande
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
DIV *reg/mem8*	F6h /6	Division naturel de AX par le contenu d'un emplacement mémoire ou registre 8 bits et entrepose le quotient dans AL et restant dans AH.
DIV *reg/mem16*	F7h /6	Division naturel de DX:AX par le contenu d'un emplacement mémoire ou registre 16 bits et entrepose le quotient dans AX et restant dans DX.
DIV *reg/mem32*	F7h /6	Division naturel de EDX:EAX par le contenu d'un emplacement mémoire ou registre 32 bits et entrepose le quotient dans EAX et restant dans EDX.
DIV *reg/mem64*	F7h /6	Division naturel de RDX:RAX par le contenu d'un emplacement mémoire ou registre 64 bits et entrepose le quotient dans RAX et restant dans RDX.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#DE(Division par zéro)	X	X	X	L'opérande de diviseur vaut

				0.
	X	X	X	Le quotient est trop large pour le registre désigné.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie		X	X	Un désalignement

l'alignement)				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MUL
Instruction assembleur 80x86 - Instruction IDIV

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 810
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 404
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 109.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 358 à 360.

Assembleur 80x86 — DIVPD

INTEL Pentium 4+ — *Divide Packed Double-Precision Floating-Point Values*

Syntaxe

DIVPD *dest*,*source*

Description

Cette instruction permet d'effectuer une division de valeur réel de double précision de registre XMM ou d'emplacement mémoire de 128 bits.

Algorithme

dest(0..63) ← *dest*(0..63) / *source*(0..63)
dest(64..127) ← *dest*(64..127) / *source*(64..127)

Mnémonique

Instruction	Opcode	Description
DIVPD *xmm1, xmm2/m128*	66h 0Fh 5Eh /r	Cette instruction permet d'effectuer une division de valeur réel de double précision de registre XMM ou d'emplacement mémoire de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 361 à 364.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

DIVPS

Packed Single-Precision Floating-Point Divide

Syntaxe

DIVPS *dest,source*

Description

Cette instruction permet d'effectuer une division de valeur réel de simple précision de registre XMM ou d'emplacement mémoire de 128 bits.

Algorithme

dest(31..0) ← *dest*(31..0) / *source*(31..0) *dest*(63..32) ← *dest*(63..32) / *source*(63..32) *dest*(95..64) ← *dest*(95..64) / *source*(95..64) *dest*(127..96) ← *dest*(127..96) / *source*(127..96)

Mnémonique

Instruction	Opcode	Description
DIVPS *xmm1,xmm2/m128*	0Fh 5Eh /r	Cette instruction permet d'effectuer une division de valeur réel de simple précision de registre XMM ou d'emplacement mémoire de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 365 à 366.

Assembleur 80x86 — DIVSD

INTEL Pentium 4+ — *Divide Scalar Double-Precision Floating-Point Values*

Syntaxe

DIVSD *dest,source*

Description

Cette instruction permet d'effectuer une division scalaire de valeur réel de double précision de registre XMM ou d'emplacement mémoire de 128 bits.

Algorithme

dest(0..63) = *dest*(0..63) / *source*(0..63)

Mnémonique

Instruction	Opcode	Description
DIVSD *xmm1, xmm2/m64*	F2h 0Fh 5Eh /r	Cette instruction permet d'effectuer une division scalaire de valeur réel de double précision de registre XMM ou d'emplacement mémoire de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 367 à 370.

Assembleur 80x86 — DIVSS

INTEL Pentium III (KNI/MMX2)+

Scalar Single-Precision Floating-Point Divide

Syntaxe

DIVSS *dest,source*

Description

Cette instruction permet d'effectuer une division scalaire de valeur réel de simple précision de registre XMM ou d'emplacement mémoire de 128 bits.

Algorithme

dest(127..0) ← *dest*(127..0) / *source*(127..0)

Mnémonique

Instruction	Opcode	Description
DIVSS *xmm1,xmm2/m128*	F3h 0Fh 5Eh /r	Cette instruction permet d'effectuer une division scalaire de valeur réel de simple précision de registre XMM ou d'emplacement mémoire de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 371 à 372.

Assembleur 80x86 DPPD

SSE4.1

Dot Product of Packed Double Precision Floating-Point Values

Syntaxe

DPPD *dest*,*source*,*immediat*

Description

Cette instruction permet d'effectuer une multiplication conditionnel d'un paquet de valeurs réel de double précision dans l'opérande de destination avec le paquet de valeurs réel de l'opérande source et effectue un masque des bits extrait d'un opérande immédiat.

Algorithme

```
SI immediat(4) = 1 ALORS
  Temp1(63..0) ← dest(63..0) x source(63..0)
SINON
  Temp1(63..0) ← +0,0
FIN SI
SI immediat(5) = 1 ALORS
  Temp1(127..64) ← dest(127..64) x source(127..64)
SINON
  Temp1(127..64) ← +0,0
FIN SI
Temp2(63..0) ← Temp1(63..0) + Temp1(127..64)
SI immediat(0) = 1 ALORS
  dest(63..0) ← Temp2(63..0)
SINON
  dest(63..0) ← +0,0
FIN SI
SI immediat(1) = 1 ALORS
  dest(127..64) ← Temp2(63..0)
SINON
```

dest(127..64) ← +0,0
FIN SI

Mnémonique

Instruction	Opcode	Description
DPPD *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 41h /r *ib*	Cette instruction permet d'effectuer une multiplication conditionnel d'un paquet de valeurs réel de double précision dans l'opérande de destination avec le paquet de valeurs réel de l'opérande source et effectue un masque des bits extrait d'un opérande immédiat.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 374 à 377.

Assembleur 80x86 — DPPS

SSE4.1 — *Dot Product of Packed Single Precision Floating-Point Values*

Syntaxe

```
DPPS dest,source,immediat
```

Description

Cette instruction permet d'effectuer une multiplication conditionnel d'un paquet de valeurs réel de simple précision dans l'opérande de destination avec le paquet de valeurs réel de l'opérande source et effectue un masque des bits extrait d'un opérande immédiat.

Algorithme

```
SI immediat(4) = 1 ALORS
  Temp1(31..0) ← dest(31..0) x source(31..0)
SINON
  Temp1(31..0) ← +0.0
FIN SI
SI immediat(5) = 1 ALORS
  Temp1(63..32) ← dest(63..32) x source(63..32)
SINON
  Temp1(63..32) ← +0.0
FIN SI
SI immediat(6) = 1 ALORS
  Temp1(95..64) ← dest(95..64) x source(95..64)
SINON
  Temp1(95..64) ← +0.0
FIN SI
SI immediat(7) = 1 ALORS
  Temp1(127..96) ← dest(127..96) x source(127..96)
SINON
  Temp1(127..96) ← +0.0
FIN SI
Temp2(31..0) ← Temp1(31..0) + Temp1(63..32)
Temp3(31..0) ← Temp1(95..64) + Temp1(127..96)
Temp4(31..0) ← Temp2(31..0) + Temp3(31..0)
```

```
SI immediat(0) = 1 ALORS
  dest(31..0) ← Temp4(31..0)
SINON
  dest(31..0) ← +0.0
FIN SI
SI immediat(1) = 1 ALORS
  dest(63..32) ← Temp4(31..0)
SINON
  dest(63..32) ← +0.0
FIN SI
SI immediat(2) = 1 ALORS
  dest(95..64) ← Temp4(31..0)
SINON
  dest(95..64) ← +0.0
FIN SI
SI immediat(3) = 1 ALORS
  dest(127..96) ← Temp4(31..0)
SINON
  dest(127..96) ← +0.0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
DPPS *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 40h /r *ib*	Cette instruction permet d'effectuer une multiplication conditionnel d'un paquet de valeurs réel de simple précision dans l'opérande de destination avec le paquet de valeurs réel de l'opérande source et effectue un masque des bits extrait d'un opérande immédiat.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 378 à 380.

Assembleur 80x86	**EMMS**
INTEL Pentium-MMX	*Empty MMX State*

Syntaxe

```
EMMS
```

Description

Cette instruction fixe les mots marqués du *MPU* (soit les registres à virgule flottante disponible) de chacun un. Naturellement, ceci réclame que tous les registres du *MPU* soit disponible à cette usage. Il devrait être utilisé avant l'exécution les instructions *MMX* et par conséquent les opérations à virgule flottante.

Algorithme

```
FloatPointTagWord ← FFFFh
```

Mnémonique

Instruction	Opcode	Description
EMMS	0Fh 77h	Cette instruction permet de fixer les mots marqués du *MPU* de chacun un.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 381 à 383.

Assembleur 80x86	**ENTER**
INTEL 80186+	*Enter*

Syntaxe

```
ENTER taille,niveau
```

Description

Cette instruction permet de gérer des langages de programmation de haut niveau. La commande *ENTER* permet de créer les structures de paramètres nécessaires aux procédures des langages de haut niveau. *taille* indique le nombre d'octet devant être réserves dans la pile pour la procédure. *niveau* indique le nombre niveau d'imbrication de la procédure, c'est-à-dire que le nombre de structures de paramètres devant être imbriquées dans la pile. Cette commande n'est disponible qu'à partir des microprocesseurs 80186 ou postérieur.

Algorithme

```
NestingLevel ← NestingLevel ∩ 1Fh
SI StackSize = 32 ALORS
  PUSH EBP
  FrameTemp ESP
SINON
  PUSH BP
  FrameTemp SP
FIN SI
SI NestingLevel > 0 ALORS
  BOUCLE POUR i ← 1 JUSQU'A (NestingLevel - 1) FAIRE
    SI OperandSize = 32 ALORS
      SI StackSize 32 ALORS
        EBP ← EBP - 4
        PUSH [EBP]
      SINON
        BP ← BP - 4
        PUSH [BP]
      FIN SI
    SINON
```

```
      SI StackSize = 32 ALORS
        EBP ← EBP - 2
        PUSH [EBP]
      SINON
        BP ← BP - 2
        PUSH [BP]
      FIN SI
    FIN SI
  FIN BOUCLE POUR
  SI OperandSize = 32 ALORS
    PUSH FrameTemp
  SINON
    PUSH FrameTemp
  FIN SI
FIN SI
SI StackSize = 32 ALORS
  EBP ← FrameTemp
  ESP ← EBP - Size
SINON
  BP ← FrameTemp
  SP ← BP - Size
FIN SI
```

Mnémonique

Instruction	Opcode	Description
ENTER *imm16*, 0	C8h *iw* 00h	Crée une procédure de cadre de pile
ENTER *imm16*, 1	C8h *iw* 01h	Crée un cadre de pile pour une procédure
ENTER *imm16*, *imm8*	C8h *iw ib*	Crée un cadre de pile pour une procédure

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de pile ou n'est pas canonique
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction LEAVE

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 811
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 111 à 112.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 384 à 390.

Assembleur 80x86	**ESC**
INTEL 8088 au 80386 et quelque hybride 80486	*ESCape*

Syntaxe

ESC *Code Opération,Adresse*

Description

Cette instruction active le coprocesseur, lequel se permettra d'utiliser les méthodes d'adressage du processeur principal (*CPU*). Le microprocesseur effectue un accès mémoire sur le coprocesseur et place les informations sur le bus de données. La technique dont le coprocesseur fait preuve pour traiter la commande *Code Opération* dépend ensuite de sa conception interne.

Mnémonique

Instruction	**Opcode**	**Description**
ESC [*xxyy*],*imm8*	D8h 06h yyh xxh *imm8*	Cette instruction active le coprocesseur, lequel se permettra d'utiliser les méthodes d'adressage du processeur principal (*CPU*).

Cycles d'horloge

Instruction	Opcode	8086	8088	80186	80286	80386	i486	Pentium	Cx486SLC	Cx486DX	IBM 486BL3X	UMC U5S
ESC [*xxyy*],*imm8*	D8h 06h yyh xxh	8/12+EA	8/12+EA	9-20								

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 811

Assembleur 80x86	**F2XM**
INTEL MPU 8087+	*Float TO eXposant Multiplication*

Syntaxe

```
F2XM
```

Description

Cette instruction s'applique à calculer 2 puissance le registre mathématique *ST(0)* moins 1 et enregistre son produit dans le registre mathématique *ST(0)*.

Algorithme

$ST(0) \leftarrow 2^{ST(0) - 1}$

Remarque

- L'utilisation de cette instruction est déconseillée, il est préférable d'utiliser plutôt «F2XM1» car cette instruction est très peu supporté par les compilateurs.

Mnémonique

Instruction	Opcode	Description
F2XM	D9h F0h	Cette instruction permet de remplacer ST(0) avec 2 à la (ST(0) - 1).

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 841

Assembleur 80x86	**F2XM1**
INTEL MPU 8087+	*Float TO eXposant Multiplication - 1*

Syntaxe

```
F2XM1
```

Description

Cette instruction s'applique à calculer 2 puissance le registre mathématique *ST(0)* moins 1 et enregistre son produit dans le registre mathématique *ST(0)*.

Algorithme

$ST(0) \leftarrow 2^{ST(0) - 1}$

Mnémonique

Instruction	Opcode	Description
F2XM1	D9h F0h	Cette instruction permet de remplacer ST(0) avec 2 à la (ST(0) - 1).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 391 à 392.*

Assembleur 80x86	**FABS**
INTEL MPU 8087+	*Float ABSolute*

Syntaxe

```
FABS
```

Description

Cette instruction permet de convertir le nombre réel contenu dans le registre mathématique *ST(0)* en sa valeur absolue.

Algorithme

```
ST(0) ← | ST(0) |
```

Mnémonique

Instruction	Opcode	Description
FABS	D9h E1h	Cette instruction permet de convertir le nombre réel contenu dans le registre mathématique *ST(0)* en sa valeur absolue.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 841
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 393 à 394.

Assembleur 80x86	**FADD**
INTEL MPU 8087+	*Float ADDition*

Syntaxe

FADD *source*, *cible*

Description

Cette instruction offre l'essentiel possibilité d'ajouter le nombre réel de valeur positive *source* à *cible* et enregistre la somme dans *cible*. Si *cible* n'est pas défini, le registre mathématique ST(0) est sera utilisé par défaut. Si *source* et *cible* ne sont pas spécifiés, le couple de registre mathématique *ST(1)* et *ST(0)* seront exploités.

Algorithme

Cible ← *Cible* + *Source*

Mnémonique

Instruction	Opcode	Description
FADD *mem32*	D8h /0	Cette instruction permet d'ajouter le nombre réel de valeur positive «*source*» à «*cible*» et enregistre la somme dans «*cible*».
FADD *mem64*	DCh /0	Cette instruction permet d'ajouter le nombre réel de valeur positive «*source*» à «*cible*» et enregistre la somme dans «*cible*».
FADD *fpureg*	D8h (C0h+r)	Cette instruction permet d'ajouter le nombre réel de valeur positive «*source*» à «*cible*» et enregistre la somme dans «*cible*».
FADD ST0,*fpureg*	D8h (C0+r)	Cette instruction permet d'ajouter le nombre réel de valeur positive «*source*» à «*cible*» et enregistre la

		somme dans «*cible*».
FADD TO *fpureg*	DCh (C0h+r)	Cette instruction permet d'ajouter le nombre réel de valeur positive «*source*» à «*cible*» et enregistre la somme dans «*cible*».
FADD *fpureg*,ST0	DCh (C0h+r)	Cette instruction permet d'ajouter le nombre réel de valeur positive «*source*» à «*cible*» et enregistre la somme dans «*cible*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 841
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 395 à 398.

Assembleur 80x86	**FADDP**
INTEL MPU 8087+	*Float ADDition and Pop*

Syntaxe

FADDP *source*, *cible*

Description

Cette instruction complémentaire ajoute le nombre réel *source* au nombre réel *cible* et enregistrer la somme dans *cible* puis prendre le registre mathématique *ST(0)* et le dépile. Si *cible* n'est pas défini, le registre mathématique *ST(0)* est utilisé par défaut. Si *source* et *cible* ne sont pas spécifiés, le couple de registre mathématique *ST(1)* et *ST(0)* sont exploitée.

Algorithme

Cible ← *Cible* + *Source*
POP *ST*

Mnémonique

Instruction	Opcode	Description
FADDP *fpureg*	DEh (C0h+r)	Cette instruction complémentaire ajoute le nombre réel *source* au nombre réel *cible* et enregistrer la somme dans *cible* puis prendre le registre mathématique *ST(0)* et le dépile.
FADDP *fpureg*,ST0	DEh (C0h+r)	Cette instruction complémentaire ajoute le nombre réel *source* au nombre réel *cible* et enregistrer la somme dans *cible* puis prendre le registre mathématique *ST(0)* et le dépile.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 842
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 395 à 398.

Assembleur 80x86	**FBLD**
INTEL MPU 8087+	*Float dcB Load*

Syntaxe

FBLD *source*

Description

Cette instruction charge la valeur de format *DCB* compactée après le registre mathématique *ST(0)*.

Algorithme

Décrémentation du pointeur de pile *FPU*
ST(0) ← *Source*

Mnémonique

Instruction	Opcode	Description
FBLD *mem80*	DFh /4	Cette instruction charge la valeur de format *DCB* compactée après le registre mathématique *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 842
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 399 à 400.

Assembleur 80x86	**FBSTP**
INTEL MPU 8087+	*Float dcB load ST Pop st*

Syntaxe

FBSTP *cible*

Description

Cette instruction extrait la valeur de format *DCB* compactée du registre mathématique *ST(0)*, l'enregistrer dans *cible* et prendre le registre mathématique *ST(0)* et sort de la pile.

Algorithme

```
cible ← ST(0)
POP ST
```

Mnémonique

Instruction	Opcode	Description
FBSTP *mem80*	DFh /6	Cette instruction extrait la valeur de format *DCB* compactée du registre mathématique *ST(0)*, l'enregistrer dans *cible* et prendre le registre mathématique *ST(0)* et sort de la pile.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 842
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 401 à 403.

Assembleur 80x86	**FCHS**
INTEL MPU 8087+	*Float CHange Signed*

Syntaxe

```
FCHS
```

Description

Cette instruction inverse tout simplement le signe du registre mathématique *ST(0)*. S'il est positive il deviendra négatif et s'il est négatif et deviendra positif.

Algorithme

```
ST(0) ← -ST(0)
```

Mnémonique

Instruction	Opcode	Description
FCHS	D9h E0h	Cette instruction permet d'inverser le signe du registre mathématique *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 842
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 404 à 405.

Assembleur 80x86	**FCLEX**
INTEL MPU 8087+	*Float CLear EXecption*

Syntaxe

```
FCLEX
```

Description

Cette instruction efface toutes les exceptions contenu dans les drapeaux de registres du coprocesseur mathématique.

Algorithme

```
SW(bit 0 à 7) ← 0
SW(bit 15) ← 0
```

Mnémonique

Instruction	Opcode	Description
FCLEX	9Bh DBh E2h	Cette instruction permet d'effacer toutes les exceptions contenu dans les drapeaux de registres du coprocesseur mathématique.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 842
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 406 à 407.

Assembleur 80x86	**FCMOV**
INTEL P6+	*Float Conditionnel Move*

Syntaxe

FCMOVB *registre*
FCMOVB *ST(0),registre*
FCMOVBE *registre*
FCMOVBE *ST(0),registre*
FCMOVE *registre*
FCMOVE *ST(0),registre*
FCMOVNB *registre*
FCMOVNB *ST(0),registre*
FCMOVNBE *registre*
FCMOVNBE *ST(0),registre*
FCMOVNE *registre*
FCMOVNE *ST(0),registre*
FCMOVNU *registre*
FCMOVNU *ST(0),registre*
FCMOVU *registre*
FCMOVU *ST(0),registre*

Description

Cette instruction permet de déplacé des nombres réel (virgule flottante) si la condition en question est satisfaite.

Algorithme

```
SI condition est vrai ALORS
  ST(0) ← ST(i)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FCMOVB ST,ST*i*	DAh (C0h+*i*)	Cette instruction permet de déplacé des nombres réel si inférieur.
FCMOVE ST,ST*i*	DAh (C8h+*i*)	Cette instruction permet de déplacé des nombres réel si égale.
FCMOVBE ST,ST*i*	DAh (D0h+*i*)	Cette instruction permet de déplacé des nombres réel si inférieur ou égale.
FCMOVU ST,ST*i*	DAh (D8h+*i*)	Cette instruction permet de déplacé des nombres réel si désordonné.
FCMOVNB ST,ST*i*	DBh (C0h+*i*)	Cette instruction permet de déplacé des nombres réel si pas inférieur.
FCMOVNE ST,ST*i*	DBh (C8h+*i*)	Cette instruction permet de déplacé des nombres réel si pas égale.
FCMOVNBE ST,ST*i*	DBh (D0h+*i*)	Cette instruction permet de déplacé des nombres réel si pas inférieur ou égale.
FCMOVNU ST,ST*i*	DBh (D8h+*i*)	Cette instruction permet de déplacé des nombres réel si pas désordonné.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 408 à 409.

Assembleur 80x86	**FCOM**
INTEL MPU 8087+	*Float Compare*

Syntaxe

FCOM *source*

Description

Cette instruction permet de comparer le nombre réel de valeur positive *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.

Algorithme

```
EVALUER CAS (relation de l'operande) DE
  CAS ST > source:
    C3 ← 0
    C2 ← 0
    C0 ← 0
  CAS ST < source:
    C3 ← 0
    C2 ← 0
    C0 ← 1
  CAS ST = source:
    C3 ← 1
    C2 ← 0
    C0 ← 0
FIN EVALUER CAS
SI ST(0) ou source = NaN OU format non supporté ALORS
  EXCEPTION #IA
  SI FPUControlWord.IM = 1 ALORS
    C3 ← 1
    C2 ← 1
    C0 ← 1
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FCOM *mem32*	D8h /2	Cette instruction permet de comparer le nombre réel de valeur positive *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.
FCOM *mem64*	DCh /2	Cette instruction permet de comparer le nombre réel de valeur positive *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.
FCOM *fpureg*	D8h D0h+r	Cette instruction permet de comparer le nombre réel de valeur positive *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.
FCOM ST0,*fpureg*	D8h D0h+r	Cette instruction permet de comparer le nombre réel de valeur positive *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 842
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 410 à 413.

Assembleur 80x86 — FCOMI

INTEL Pentium III+ — *Compare Real and Set EFLAGS*

Syntaxe

```
FCOMI ST, ST(i)
```

Description

Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats.

Algorithme

```
EVALUER CAS (relation de l'opérande) DE
  CAS ST(0) > ST(i):
    ZF ← 0
    PF ← 0
    CF ← 0
  CAS ST(0) < ST(i):
    ZF ← 0
    PF ← 0
    CF ← 1
  CAS ST(0) ST(i):
    ZF ← 1
    PF ← 0
    CF ← 0
FIN EVALUER CAS
SI ST(0) OU ST(i) NaN OU format non supporté ALORS
  EXCEPTION #IA
  SI FPUControlWord.IM = 1 ALORS
    ZF ← 1
    PF ← 1
    CF ← 1
  FIN SI
```

FIN SI

Mnémonique

Instruction	Opcode	Description
FCOMI ST0,ST*i*	DBh (F0h+*i*)	Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 414 à 416.*

Assembleur 80x86 — FCOMIP

INTEL Pentium III+ — *Compare Real and Set EFLAGS*

Syntaxe

```
FCOMIP ST, ST(i)
```

Description

Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats et désempile de la pile la valeur dans le registre.

Algorithme

```
EVALUER CAS (relation de l'opérande) DE
  CAS ST(0) > ST(i):
    ZF ← 0
    PF ← 0
    CF ← 0
  CAS ST(0) < ST(i):
    ZF ← 0
    PF ← 0
    CF ← 1
  CAS ST(0) ST(i):
    ZF ← 1
    PF ← 0
    CF ← 0
FIN EVALUER CAS
SI ST(0) OU ST(i) NaN OU format non supporté ALORS
  EXCEPTION #IA
  SI FPUControlWord.IM = 1 ALORS
    ZF ← 1
    PF ← 1
    CF ← 1
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FCOMIP ST0,ST*i*	DFh (F0h+*i*)	Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats et désempile de la pile la valeur dans le registre.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel, Mars 2010, Publication No. 253666-034US, page 414 à 416.*

Assembleur 80x86	**FCOMP**
INTEL MPU 8087+	*Float Compare Pop st(0)*

Syntaxe

FCOMP *source*

Description

Cette instruction compare le nombre réel de valeur positive «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.

Algorithme

```
EVALUER CAS (relation de l'operande) DE
  CAS ST > source:
    C3 ← 0
    C2 ← 0
    C0 ← 0
  CAS ST < source:
    C3 ← 0
    C2 ← 0
    C0 ← 1
  CAS ST = source:
    C3 ← 1
    C2 ← 0
    C0 ← 0
FIN EVALUER CAS
SI ST(0) ou source = NaN OU format non supporté ALORS
  EXCEPTION #IA
  SI FPUControlWord.IM = 1 ALORS
    C3 ← 1
    C2 ← 1
    C0 ← 1
  FIN SI
FIN SI
```

PopRegisterStack

Mnémonique

Instruction	Opcode	Description
FCOMP *mem32*	D8h /3	Cette instruction permet de comparer le nombre réel de valeur positive «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.
FCOMP *mem64*	DCh /3	Cette instruction permet de comparer le nombre réel de valeur positive «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.
FCOMP *fpureg*	D8h D8h+r	Cette instruction permet de comparer le nombre réel de valeur positive «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.
FCOMP ST0,*fpureg*	D8h D8h+r	Cette instruction permet de comparer le nombre réel de valeur positive «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 842
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 410 à 413.

Assembleur 80x86	**FCOMPP**
INTEL MPU 8087+	*Float Compare Pop Pop*

Syntaxe

```
FCOMPP
```

Description

Cette instruction permet de comparer les nombres réels du registre mathématique *ST(1)* avec le registre mathématique *ST(0)*, ensuite de mettre les indicateurs d'état *C0* à *C3* à la valeur 1 et prendre le couple de registre mathématique *ST(1)* et *ST(0)* dans la pile de registres (2 dépilages).

Algorithme

```
EVALUER CAS (relation de l'operande) DE
  CAS ST > source:
    C3 ← 0
    C2 ← 0
    C0 ← 0
  CAS ST < source:
    C3 ← 0
    C2 ← 0
    C0 ← 1
  CAS ST = source:
    C3 ← 1
    C2 ← 0
    C0 ← 0
FIN EVALUER CAS
SI ST(0) ou source = NaN OU format non supporté ALORS
  EXCEPTION #IA
  SI FPUControlWord.IM = 1 ALORS
    C3 ← 1
    C2 ← 1
    C0 ← 1
  FIN SI
FIN SI
```

PopRegisterStack PopRegisterStack

Mnémonique

Instruction	Opcode	Description
FCOMPP	DEh D9h	Cette instruction permet de comparer les nombres réels du registre mathématique *ST(1)* avec le registre mathématique *ST(0)*, ensuite de mettre les indicateurs d'état *C0* à *C3* à la valeur 1 et prendre le couple de registre mathématique *ST(1)* et *ST(0)* dans la pile de registres (2 dépilages).

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 843
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 410 à 413.

Assembleur 80x86	**FCOS**
INTEL MPU 80387+	*Float COSinus*

Syntaxe

```
FCOS
```

Description

Cette instruction calcule le cosinus du registre mathématique *ST(0)* et l'enregistrer dans le registre mathématique *ST(1)*; ensuite met dans le registre mathématique *ST(0)* à la valeur 1,0. Attention, cette commande est disponible uniquement sur les *80387* ou postérieur.

Algorithme

```
SI opérande dans le rang ALORS
  C2 ← 0
  ST ← COS(ST)
SINON
  C2 ← 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FCOS	D9h FFh	Cette instruction permet de calculer le cosinus du registre mathématique *ST(0)* et l'enregistrer dans le registre mathématique *ST(1)*; ensuite met dans le registre mathématique *ST(0)* à la valeur 1,0.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 843
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 417 à 418.

Assembleur 80x86	**FDECSTP**
INTEL MPU 8087+	*Float Decrement STack Position*

Syntaxe

```
FDECSTP
```

Description

Cette instruction permet de décrémenter le pointeur de pile de registres.

Algorithme

```
SI TOP = 0 ALORS
  TOP ← 7
SINON
  TOP ← TOP - 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FDECSTP	D9h F6h	Cette instruction permet de décrémenter le pointeur de pile de registres.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 843
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 419 à 421.

Assembleur 80x86	**FDISI**
INTEL MPU 8087	*Float Desactive Interrupt Set Internal?*

Syntaxe

```
FDISI
```

Description

Cette instruction permettait de désactiver les interruptions; toutefois cette commande n'est disponible que sur le coprocesseur mathématique 8087. Elle n'a pas été de retour dans les coprocesseur suivant.

Algorithme

```
IF ← 0
```

Mnémonique

Instruction	Opcode	Description
FDISI	9Bh DBh E1h	Cette instruction permet de désactiver les interruptions.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 843

Assembleur 80x86	**FDIV**
INTEL MPU 8087+	*Float Division*

Syntaxe

FDIV *source*, *cible*

Description

Cette instruction offre la possibilité d'effectuer des divisions de nombre réel de valeur positive *source* par *cible* et enregistrer le résultat dans *cible*. Si *cible* n'est pas spécifié, le registre mathématique *ST(0)* est inutilisé. Si *source* et *cible* ne sont pas défini, le couple de registre mathématique *ST(1)* et *ST(0)* sont utilisés.

Algorithme

source ← *source* ÷ *cible*

Mnémonique

Instruction	Opcode	Description
FDIV *mem32*	D8h /6	Cette instruction permet d'effectuer des divisions de nombre réel de valeur positive *source* par *cible* et enregistrer le résultat dans *cible*.
FDIV *mem64*	DCh /6	Cette instruction permet d'effectuer des divisions de nombre réel de valeur positive *source* par *cible* et enregistrer le résultat dans *cible*.
FDIV *fpureg*	D8h (F0h+r)	Cette instruction permet d'effectuer des divisions de nombre réel de valeur positive *source* par *cible* et

		enregistrer le résultat dans *cible*.
FDIV ST0,*fpureg*	D8h (F0h+r)	Cette instruction permet d'effectuer des divisions de nombre réel de valeur positive *source* par *cible* et enregistrer le résultat dans *cible*.
FDIV TO *fpureg*	DCh (F8h+r)	Cette instruction permet d'effectuer des divisions de nombre réel de valeur positive *source* par *cible* et enregistrer le résultat dans *cible*.
FDIV *fpureg*,ST0	DCh (F8h+r)	Cette instruction permet d'effectuer des divisions de nombre réel de valeur positive *source* par *cible* et enregistrer le résultat dans *cible*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 843
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 422 à 424.

Assembleur 80x86	**FDIVP**
INTEL MPU 8087+	*Float Division & Position*

Syntaxe

FDIVP *source, cible*

Description

Cette instruction offre la possibilité d'effectuer des division de nombre réel et place ST(0) dans la pile.

Algorithme

source ← *source* ÷ *cible*
POP *ST*

Mnémonique

Instruction	Opcode	Description
FDIVP *fpureg*	DEh (F8h+r)	Cette instruction permet d'effectuer des division de nombre réel et place ST(0) dans la pile.
FDIVP *fpureg*,ST0	DEh (F8h+r)	Cette instruction permet d'effectuer des division de nombre réel et place ST(0) dans la pile.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 843
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 422 à 424.

Assembleur 80x86	**FDIVR**
INTEL MPU 8087+	*Float Division Reverse*

Syntaxe

FDIVR *cible*, *source*

Description

Cette instruction offre la possibilité d'effectuer des division de nombre réel de valeur positive avec les registres inversés «*cible*» par «*source*».

Algorithme

source ← *source* ÷ *cible*

Mnémonique

Instruction	Opcode	Description
FDIVR *mem32*	D8h /7	Cette instruction permet d'effectuer des division de nombre réel de valeur positive avec les registres inversés «*cible*» par «*source*».
FDIVR *mem64*	DCh /7	Cette instruction permet d'effectuer des division de nombre réel de valeur positive avec les registres inversés «*cible*» par «*source*».
FDIVR *fpureg*	D8h (F8h+r)	Cette instruction permet d'effectuer des division de nombre réel de valeur positive avec les registres inversés «*cible*» par «*source*».

FDIVR ST0,*fpureg*	D8h (F8h+r)	Cette instruction permet d'effectuer des division de nombre réel de valeur positive avec les registres inversés «*cible*» par «*source*».
FDIVR TO *fpureg*	DCh (F0h+r)	Cette instruction permet d'effectuer des division de nombre réel de valeur positive avec les registres inversés «*cible*» par «*source*».
FDIVR *fpureg*,ST0	DCh (F0h+r)	Cette instruction permet d'effectuer des division de nombre réel de valeur positive avec les registres inversés «*cible*» par «*source*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 843
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 425 à 428.

Assembleur 80x86	**FDIVRP**
INTEL MPU 8087+	*Float Division Reverse & Position*

Syntaxe

FDIVRP *cible*, *source*

Description

Cette instruction offre la possibilité d'effectuer des divisions de nombre réel avec les registres inversés «*cible*» par «*source*» et place ST(0) dans la pile.

Algorithme

source ← *source* ÷ *cible*
POP *ST*

Mnémonique

Instruction	Opcode	Description
FDIVRP *fpureg*	DEh (F0h+r)	Cette instruction permet d'effectuer des divisions de nombre réel avec les registres inversés «*cible*» par «*source*» et place ST(0) dans la pile.
FDIVRP *fpureg*,ST0	DEh (F0h+r)	Cette instruction permet d'effectuer des divisions de nombre réel avec les registres inversés «*cible*» par «*source*» et place ST(0) dans la pile.
FDIVRP	DEh F1h	Cette instruction permet d'effectuer des division de nombre réel de valeur

		positive avec les registres inversés «*cible*» par «*source*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 844
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 425 à 428.

Assembleur 80x86 — FEMMS

AMD 3DNow!+ — *Faster EMMS*

Syntaxe

```
FEMMS
```

Description

Cette instruction permet d'effacer les états *MMX* après le passage d'instructions *MMX*.

Algorithme

```
FPU.TAG ← FFFFh
```

Mnémonique

Instruction	Opcode	Description
FEMMS	0Fh 0Eh	Cette instruction permet d'effacer les états *MMX* après le passage d'instructions *MMX*.

Assembleur 80x86	**FENI**
INTEL MPU 8087	*Float Enable Numeric Interrupt*

Syntaxe

```
FENI
```

Description

Cette instruction permet d'autoriser les interruptions.

Algorithme

```
IF ← 1
```

Mnémonique

Instruction	Opcode	Description
FENI	9Bh DBh E0h	Cette instruction permet d'autoriser les interruptions.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 844

Assembleur 80x86	FFREE
INTEL MPU 8087+	*Float Free*

Syntaxe

FFREE *cible*

Description

Cette instruction permet de libérer le registre «*cible*» devant représenté un des registres de la pile.

Algorithme

TAG(*cible*) ← 11b

Mnémonique

Instruction	Opcode	Description
FFREE *fpureg*	DDh (C0h+r)	Cette instruction permet de libérer le registre «*cible*» devant représenté un des registres de la pile.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 844
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 429 à 429.

Assembleur 80x86	**FFREEP**
INTEL MPU 8087+	*Float Free and Pop*

Syntaxe

FFREEP *cible*

Description

Cette instruction permet de libérer un des registres de la pile et d'ensuite de désempiler le registre de la pile.

Algorithme

TAG(*cible*) ← 11b
POP *ST*

Mnémonique

Instruction	**Opcode**	**Description**
FFREEP *fpureg*	DFh (C0h+r)	Cette instruction permet de libérer un des registres de la pile et d'ensuite de désempiler le registre de la pile.

Assembleur 80x86	**FIADD**
INTEL MPU 8087+	*Float Integer ADDition*

Syntaxe

FIADD *source,cible*

Description

Cette instruction offre l'essentiel possibilité d'ajouter le nombre réel *source* à *cible* et enregistre la somme dans *cible*. Si *cible* n'est pas défini, le registre mathématique ST(0) est sera utilisé par défaut. Si *source* et *cible* ne sont pas spécifiés, le couple de registre mathématique *ST(1)* et *ST(0)* seront exploités.

Algorithme

cible ← *cible* + *source*

Mnémonique

Instruction	Opcode	Description
FIADD *mem16*	DEh /0	Cette instruction permet d'ajouter le nombre réel *source* à *cible* et enregistre la somme dans *cible*.
FIADD *mem32*	DAh /0	Cette instruction permet d'ajouter le nombre réel *source* à *cible* et enregistre la somme dans *cible*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 844
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 395 à 398.

Assembleur 80x86	**FICOM**
INTEL MPU 8087+	*Float Integer COMpare*

Syntaxe

FICOM *source*

Description

Cette instruction permet de comparer le nombre réel *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.

Algorithme

```
EVALUER CAS relation de l'opérande
  CAS non comparable:
    C3 ← 1
    C2 ← 1
    C0 ← 1
  CAS ST > source:
    C3 ← 0
    C2 ← 0
    C0 ← 0
  CAS ST < source:
    C3 ← 0
    C2 ← 0
    C0 ← 1
  CAS ST = source:
    C3 ← 1
    C2 ← 0
    C0 ← 0
FIN EVALUER CAS
```

Mnémonique

Instruction	Opcode	Description
FICOM *mem16*	DEh /2	Cette instruction permet de comparer le nombre réel *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.
FICOM *mem32*	DAh /2	Cette instruction permet de comparer le nombre réel *source* avec le registre mathématique *ST(0)* et mettre les indicateurs d'état *C0* à *C3* avec la valeur 1.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 844
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 430 à 431.

Assembleur 80x86	**FICOMP**
INTEL MPU 8087+	*Float Integer COMpare & Position*

Syntaxe

```
FICOMP source
```

Description

Cette instruction compare le nombre réel «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.

Algorithme

```
EVALUER CAS relation de l'opérande
  CAS non comparable:
    C3 ← 1
    C2 ← 1
    C0 ← 1
  CAS ST > source:
    C3 ← 0
    C2 ← 0
    C0 ← 0
  CAS ST < source:
    C3 ← 0
    C2 ← 0
    C0 ← 1
  CAS ST = source:
    C3 ← 1
    C2 ← 0
    C0 ← 0
FIN EVALUER CAS
POP ST
```

Mnémonique

Instruction	Opcode	Description
FICOMP *mem16*	DEh /3	Cette instruction compare le nombre réel «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.
FICOMP *mem32*	DAh /3	Cette instruction compare le nombre réel «*source*» avec le registre mathématique *ST(0)*, mettre les indicateurs d'état *C0* à *C3* avec la valeur 1 et dépile le registre mathématique *ST(0)* de la pile de registres.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 844
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 430 à 431.

Assembleur 80x86	**FIDIV**
INTEL MPU 8087+	*Float Integer Division*

Syntaxe

```
FIDIV source,cible
```

Description

Cette instruction offre la possibilité d'effectuer des divisions de nombre réel *source* par *cible* et enregistrer le résultat dans *cible*. Si *cible* n'est pas spécifié, le registre mathématique *ST(0)* est inutilisé. Si *source* et *cible* ne sont pas défini, le couple de registre mathématique *ST(1)* et *ST(0)* sont utilisés.

Algorithme

```
source ← source ÷ cible
```

Mnémonique

Instruction	Opcode	Description
FIDIV *mem16*	DEh /6	Cette instruction permet d'effectuer des divisions de nombre réel *source* par *cible* et enregistrer le résultat dans *cible*.
FIDIV *mem32*	DAh /6	Cette instruction permet d'effectuer des divisions de nombre réel *source* par *cible* et enregistrer le résultat dans *cible*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 844
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 422 à 424.

Assembleur 80x86	**FIDIVR**
INTEL MPU 8087+	*Float Integer Divsion Reverse*

Syntaxe

FIDIVR *source,cible*

Description

Cette instruction offre la possibilité d'effectuer des division de nombre réel avec les registres inversés «*cible*» par «*source*».

Algorithme

source ← *source* ÷ *cible*

Mnémonique

Instruction	Opcode	Description
FIDIVR *mem16*	DEh /7	Cette instruction offre la possibilité d'effectuer des division de nombre réel avec les registres inversés «*cible*» par «*source*».
FIDIVR *mem32*	DAh /7	Cette instruction offre la possibilité d'effectuer des division de nombre réel avec les registres inversés «*cible*» par «*source*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 845
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 425 à 428.

Assembleur 80x86	**FILD**
INTEL MPU 8087+	*Float Integer LoaD*

Syntaxe

FILD *source*

Description

Cette instruction permet de charger le nombre entier auprès du registre ST(0).

Algorithme

Décrémentation de la pile FPU
ST(0) ← source

Mnémonique

Instruction	Opcode	Description
FILD *mem16*	DFh /0	Cette instruction permet de charger le nombre entier auprès du registre ST(0).
FILD *mem32*	DBh /0	Cette instruction permet de charger le nombre entier auprès du registre ST(0).
FILD *mem64*	DFh /5	Cette instruction permet de charger le nombre entier auprès du registre ST(0).

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 845
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 432 à 434.

Assembleur 80x86	**FIMUL**
INTEL MPU 8087+	*Float Integer Multplication*

Syntaxe

FIMUL *source*,*cible*

Description

Cette instruction permet de multiplier le nombre réel «source» par «*cible*» et sauvegarde le résultat dans «*cible*». Si la «*cible*» n'est pas définit, le registre *ST(0)* est utilisé. Si «source» et «cible» ne sont pas définit, les registres *ST(1)* et *ST(0)* sont utilisés.

Algorithme

source ← *source* x *cible*

Mnémonique

Instruction	Opcode	Description
FIMUL *mem16*	DEh /1	Cette instruction permet de multiplier le nombre réel «source» par «*cible*» et sauvegarde le résultat dans «*cible*». Si la «*cible*» n'est pas définit, le registre *ST(0)* est utilisé. Si «source» et «cible» ne sont pas définit, les registres *ST(1)* et *ST(0)* sont utilisés.
FIMUL *mem32*	DAh /1	Cette instruction permet de multiplier le nombre réel «source» par «*cible*» et sauvegarde le résultat dans «*cible*». Si la «*cible*» n'est pas définit, le registre *ST(0)* est utilisé. Si «source» et «cible» ne sont pas définit, les registres *ST(1)* et *ST(0)* sont utilisés.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 845
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 455 à 459.

Assembleur 80x86	**FINCSTP**
INTEL MPU 8087+	*Float Increment STack Pointer*

Syntaxe

```
FINCSTP
```

Description

Cette instruction permet d'incrémenter le pointeur de pile de registres.

Algorithme

```
SI TOP = 7 ALORS
  TOP ← 0
SINON
  TOP ← TOP + 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FINCSTP	D9h F7h	Cette instruction permet d'incrémenter le pointeur de pile de registres.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 845
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 435 à 436.

Assembleur 80x86	**FINIT**
INTEL MPU 8087+	*Float Initialize*

Syntaxe

```
FINIT
```

Description

Cette instruction permet d'effectuer l'initialisation du coprocesseur mathématique.

Algorithme

```
CW ← 37Fh
SW ← 0
TW ← 0FFFFh
FEA ← 0
FDS ← 0
FIP ← 0
FOP ← 0
FCS ← 0
```

Mnémonique

Instruction	Opcode	Description
FINIT	9Bh DBh E3h	Cette instruction permet d'effectuer l'initialisation du coprocesseur mathématique.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 845
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 437 à 437.

Assembleur 80x86	**FIST**
INTEL MPU 8087+	*Float Integer in ST*

Syntaxe

FISTP *cible*

Description

Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*».

Algorithme

cible ← *ST(0)*

Mnémonique

Instruction	Opcode	Description
FIST *mem16*	DFh /2	Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*».
FIST *mem32*	DBh /2	Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 845
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 438 à 442.

Assembleur 80x86	**FISTP**
INTEL MPU 8087+	*Float Integer in ST & Pop*

Syntaxe

FISTP *cible*

Description

Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*» et prendre le registre *ST(0)* de la pile du registres.

Algorithme

cible ← *ST(0)*
POP *ST*

Mnémonique

Instruction	Opcode	Description
FISTP *mem16*	DFh /3	Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*» et prendre le registre *ST(0)* de la pile du registres.
FISTP *mem32*	DBh /3	Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*» et prendre le registre *ST(0)* de la pile du registres.
FISTP *mem64*	DFh /7	Cette instruction permet de sauvegarder le nombre du registre *ST(0)* dans «*cible*» et prendre le registre *ST(0)* de la pile du registres.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 845
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 438 à 442.

Assembleur 80x86 FISTTP

SSE3 *Store Integer with Truncation*

Syntaxe

FISTTP *dest*

Description

Cette instruction permet de convertir la valeur contenu dans le registre ST dans un entier en utilisant on tronquage, de type *chop*, lequel est un mode d'arrondissement, et transfert le résultat dans la destination et ensuite effectuer un désempilage de ST.

Algorithme

dest ← ST
pop ST

Mnémonique

Instruction	Opcode	Description
FISTTP *m16int*	DFh /1	Cette instruction permet de convertir la valeur contenu dans le registre ST dans un entier en utilisant on tronquage, de type *chop*, lequel est un mode d'arrondissement, et transfert le résultat dans la destination et ensuite effectuer un désempilage de ST.
FISTTP *m32int*	DBh /1	Cette instruction permet de convertir la valeur contenu dans le registre ST

		dans un entier en utilisant on tronquage, de type *chop*, lequel est un mode d'arrondissement, et transfert le résultat dans la destination et ensuite effectuer un désempilage de ST.
FISTTP *m64int*	DDh /1	Cette instruction permet de convertir la valeur contenu dans le registre ST dans un entier en utilisant on tronquage, de type *chop*, lequel est un mode d'arrondissement, et transfert le résultat dans la destination et ensuite effectuer un désempilage de ST.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 443 à 444.

Assembleur 80x86	**FISUB**
INTEL MPU 8087+	*Float Integer Subtract*

Syntaxe

FISUB *source*

Description

Cette instruction offre l'essentielle posibilité de soustraire le nombre «*source*» du registre *ST(0)*.

Algorithme

ST ← ST - source

Mnémonique

Instruction	Opcode	Description
FISUB *mem16*	DEh /4	Cette instruction permet de soustraire le nombre «*source*» du registre *ST(0)*.
FISUB *mem32*	DAh /4	Cette instruction permet de soustraire le nombre «*source*» du registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 845
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 503 à 506.

Assembleur 80x86	**FISUBR**
INTEL MPU 8087+	*Float Integer Subtract Reverse*

Syntaxe

FISUBR *source*

Description

Cette instruction offre l'essentielle possibilité de soustraire le nombre du registre *ST(0)* de «*source*».

Algorithme

source ← *source - ST*

Mnémonique

Instruction	Opcode	Description
FISUBR *mem16*	DEh /5	Cette instruction permet de soustraire le nombre du registre *ST(0)* de «*source*».
FISUBR *mem32*	DAh /5	Cette instruction permet de soustraire le nombre du registre *ST(0)* de «*source*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 507 à 510.

Assembleur 80x86	**FLD**
INTEL MPU 8087+	*Float LoaD*

Syntaxe

FLD source

Description

Cette instruction permet de charger le nombre réel du registre *ST(0)* dans la pile.

Algorithme

Décrémentation de la pile du FPU
ST(0) ← source

Mnémonique

Instruction	Opcode	Description
FLD *mem32*	D9h /0	Cette instruction permet de charger le nombre réel du registre *ST(0)* dans la pile.
FLD *mem64*	DDh /0	Cette instruction permet de charger le nombre réel du registre *ST(0)* dans la pile.
FLD *mem80*	DBh /5	Cette instruction permet de charger le nombre réel du registre *ST(0)* dans la pile.
FLD *fpureg*	D9h (C0h+r)	Cette instruction permet de charger le nombre réel du registre *ST(0)* dans la pile.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 446 à 448.

Assembleur 80x86	**FLD1**
INTEL MPU 8087+	*Float LoaD 1*

Syntaxe

```
FLD1
```

Description

Cette instruction permet de charger la constante 1,0 du registre *ST(0)* dans la pile.

Algorithme

Décrémentation de la pile du FPU
ST(0) ← 1

Mnémonique

Instruction	Opcode	Description
FLD1	D9h E8h	Cette instruction permet de charger la constante 1,0 du registre *ST(0)* dans la pile.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 449 à 450.

Assembleur 80x86 — FLDCW

INTEL MPU 8087+ — *Float Load Control Word*

Syntaxe

FLDCW *source*

Description

Cette instruction permet de charger le «mot de contrôle» avec «*source*».

Algorithme

CW ← *source*

Mnémonique

Instruction	Opcode	Description
FLDCW *mem16*	D9h /5	Cette instruction permet de charger le «mot de contrôle» avec «*source*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 451 à 452.

Assembleur 80x86	**FLDENV**
INTEL MPU 8087+	*Float LoaD ENVironment*

Syntaxe

FLDENV *source*

Description

Cette instruction permet de charger l'environnement à partir de «*source*».

Algorithme

environnement FPU ← *source*

Mnémonique

Instruction	**Opcode**	**Description**
FLDENV *mem*	D9h /4	Cette instruction permet de charger l'environnement à partir de «*source*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 453 à 454.

Assembleur 80x86	**FLDL2E**
INTEL MPU 8087+	*Float LoaD Logarithmic 2 in E*

Syntaxe

FLDL2E

Description

Cette instruction permet de mettre le résultat du logarithme de «*e*» de la base 2 dans *ST(0)*.

Algorithme

Décrémentation de la pile du FPU
ST(0) ← 2^e

Mnémonique

Instruction	Opcode	Description
FLDL2E	D9h EAh	Cette instruction permet de mettre le résultat du logarithme de «*e*» de la base 2 dans *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 449 à 450.

Assembleur 80x86	**FLDL2T**
INTEL MPU 8087+	

Syntaxe

FLDL2T

Description

Cette instruction permet de mettre le résultat du logarithme de 10 de la base 2 dans *ST(0)*.

Algorithme

Décrémentation de la pile du FPU *ST(0)* ← 10^e

Mnémonique

Instruction	**Opcode**	**Description**
FLDL2T	D9h E9h	Cette instruction permet de mettre le résultat du logarithme de 10 de la base 2 dans *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 449 à 450.

Assembleur 80x86	**FLDLG2**
INTEL MPU 8087+	*Float LoaD LoGarithmic 2*

Syntaxe

FLDLG2

Description

Cette instruction permet de mettre le résultat du logarithme de 2 de la base 10 dans *ST(0)*.

Algorithme

Décrémentation de la pile du FPU *ST(0)* ← LOG 2^{10}

Mnémonique

Instruction	**Opcode**	**Description**
FLDLG2	D9h ECh	Cette instruction permet de mettre le résultat du logarithme de 2 de la base 10 dans *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 449 à 450.

Assembleur 80x86	**FLDLN2**
INTEL MPU 8087+	*Float Load Logarithmic Natural 2*

Syntaxe

```
FLDLN2
```

Description

Cette instruction permet de mettre le résultat du logarithme de «*e*» de la base 10 dans *ST(0)*.

Mnémonique

Instruction	Opcode	Description
FLDLN2	D9h EDh	Cette instruction permet de mettre le résultat du logarithme de «*e*» de la base 10 dans *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 846
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 449 à 450.

Assembleur 80x86	**FLDPI**
INTEL MPU 8087+	*Float LoaD PI*

Syntaxe

```
FLDPI
```

Description

Cette instruction permet de mettre le résultat de la constante *PI* (Π) dans le registre *ST(0)*.

Algorithme

```
ST(0) ← Π
```

Mnémonique

Instruction	Opcode	Description
FLDPI	D9h EBh	Cette instruction permet de mettre le résultat de la constante *PI* (Π) dans le registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 847
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 449 à 450.

Assembleur 80x86	**FLDZ**
INTEL MPU 8087+	*Float LoaD Zero*

Syntaxe

```
FLDZ
```

Description

Cette instruction permet de mettre le résultat de la constante 0,0 dans le registre *ST(0)*.

Algorithme

```
ST(0) ← 0,0
```

Mnémonique

Instruction	Opcode	Description
FLDZ	D9h EEh	Cette instruction permet de mettre le résultat de la constante 0,0 dans le registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 847
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 449 à 450.

Assembleur 80x86	**FMUL**
INTEL MPU 8087+	*Float MULtiplication*

Syntaxe

FMUL *source,cible*

Description

Cette instruction permet de multiplier le nombre réel de valeur positive «source» par «*cible*» et sauvegarde le résultat dans «*cible*». Si la «*cible*» n'est pas définit, le registre *ST(0)* est utilisé. Si «source» et «cible» ne sont pas définit, les registres *ST(1)* et *ST(0)* sont utilisés.

Algorithme

source ← *source* x *cible*

Mnémonique

Instruction	Opcode	Description
FMUL *mem32*	D8h /1	Cette instruction permet de multiplier le nombre réel de valeur positive «source» par «*cible*» et sauvegarde le résultat dans «*cible*».
FMUL *mem64*	DCh /1	Cette instruction permet de multiplier le nombre réel de valeur positive «source» par «*cible*» et sauvegarde le résultat dans «*cible*».
FMUL *fpureg*	D8h (C8h+r)	Cette instruction permet de multiplier le nombre réel de valeur positive «source» par «*cible*» et

		sauvegarde le résultat dans «*cible*».
FMUL ST0,*fpureg*	D8h (C8h+r)	Cette instruction permet de multiplier le nombre réel de valeur positive «source» par «*cible*» et sauvegarde le résultat dans «*cible*».
FMUL TO *fpureg*	DCh (C8h+r)	Cette instruction permet de multiplier le nombre réel de valeur positive «source» par «*cible*» et sauvegarde le résultat dans «*cible*».
FMUL *fpureg*,ST0	DCh (C8h+r)	Cette instruction permet de multiplier le nombre réel de valeur positive «source» par «*cible*» et sauvegarde le résultat dans «*cible*».

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 847
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 455 à 459.

Assembleur 80x86 — FMULP

INTEL MPU 8087+ — *Float MULtiplication in Pointer*

Syntaxe

FMULP source,cible

Description

Cette instruction permet de multiplier le nombre réel de valeur positive «*source*» par «*cible*» et sauvegarde le résultat dans «*cible*» et prend ensuite *ST(0)* dans la pile de registres. Si «source» et «cible» ne sont pas définit, les registres *ST(1)* et *ST(0)* sont utilisés.

Algorithme

source ← source x cible POP ST(0)

Mnémonique

Instruction	Opcode	Description
FMULP *fpureg*	DEh (C8h+r)	Cette instruction permet de multiplier le nombre réel de valeur positive «*source*» par «*cible*» et sauvegarde le résultat dans «*cible*» et prend ensuite *ST(0)* dans la pile de registres.
FMULP *fpureg*,ST0	DEh (C8h+r)	Cette instruction permet de multiplier le nombre réel de valeur positive «*source*» par «*cible*» et sauvegarde le résultat dans «*cible*» et prend ensuite *ST(0)* dans la pile de registres.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 847

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 455 à 459.

Assembleur 80x86 — FNCLEX

INTEL MPU 8087+ — *Float Numeric CLear EXecption*

Syntaxe

```
FNCLEX
```

Description

Cette instruction permet d'éliminer et de gérer les exceptions non masquées.

Algorithme

```
FPUStatusWord(0..7) ← 0
FPUStatusWord(15) ← 0
```

Mnémonique

Instruction	Opcode	Description
FNCLEX	DBh E2h	Cette instruction permet d'éliminer et de gérer les exceptions non masquées.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 847
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 406 à 407.

Syntaxe

FNDISI

Description

Cette instruction permet de désactiver les interruptions et de gérer les exceptions non masquées.

Mnémonique

Instruction	Opcode	Description
FNDISI	DBh E1h	Cette instruction permet de désactiver les interruptions et de gérer les exceptions non masquées.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 847

Syntaxe

FNENI

Description

Cette instruction permet d'activer les interruptions et de gérer les exceptions non masquées.

Mnémonique

Instruction	Opcode	Description
FNENI	DBh E0h	Cette instruction permet d'activer les interruptions et de gérer les exceptions non masquées.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 847

Assembleur 80x86 — FNINIT

INTEL MPU 8087+ — *Float & Numeric Initialize*

Syntaxe

FNINIT

Description

Cette instruction permet d'initialiser le coprocesseur mathématique et de gérer les exceptions non masquées.

Algorithme

```
FPUControlWord ← 037Fh
FPUStatusWord ← 0
FPUTagWord ← FFFFh
FPUDataPointer ← 0
FPUInstructionPointer ← 0
FPULastInstructionOpcode ← 0
```

Mnémonique

Instruction	Opcode	Description
FNINIT	DBh E3h	Cette instruction permet d'initialiser le coprocesseur mathématique et de gérer les exceptions non masquées.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 848
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 437 à 437.

Assembleur 80x86	**FNOP**
INTEL MPU 8087+	*Float No OPeration*

Syntaxe

```
FNOP
```

Description

Cette instruction permet de ne rien faire et de simplement passer à l'instruction suivante.

Mnémonique

Instruction	Opcode	Description
FNOP	D9h D0h	Cette instruction permet de ne rien faire et de simplement passer à l'instruction suivante.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 848
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 460 à 460.

Syntaxe

FNSAVE *source*

Description

Cette instruction permet de sauvegarder l'état courant du coprocesseur mathématique dans l'emplacement mémoire à partir de «*source*» et gérer les exceptions numériques non masquées.

Algorithme

source[FPUControlWord] ← FPUControlWord
source[FPUStatusWord] ← FPUStatusWord
source[FPUTagWord] ← FPUTagWord
source[FPUDataPointer) ← FPUDataPointer
source[FPUInstructionPointer] ← FPUInstructionPointer
source[FPULastInstructionOpcode) ← FPULastInstructionOpcode
source[ST(0)] ← ST(0)
source[ST(1)] ← ST(1)
source[ST(2)] ← ST(2)
source[ST(3)] ← ST(3)
source[ST(4)] ← ST(4)
source[ST(5)] ← ST(5)
source[ST(6)] ← ST(6)
source[ST(7)] ← ST(7)
FPUControlWord ← 037Fh
FPUStatusWord ← 0
FPUTagWord ← FFFFh
FPUDataPointer ← 0
FPUInstructionPointer ← 0
FPULastInstructionOpcode ← 0

Mnémonique

Instruction	Opcode	Description
FNSAVE *mem*	DDh /6	Cette instruction permet de sauvegarder l'état courant du coprocesseur mathématique dans l'emplacement mémoire à partir de «*source*» et gérer les exceptions numériques non masquées.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 848
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 478 à 481.

Assembleur 80x86	**FNSTCW**
INTEL MPU 8087+	

Syntaxe

FNSTCW *cible*

Description

Cette instruction permet de sauvegarder le «mot de contrôle» dans «cible» et gérer les exceptions numériques non masquées.

Algorithme

cible ← FPUControlWord

Mnémonique

Instruction	Opcode	Description
FNSTCW *mem16*	D9h /7	Cette instruction permet de sauvegarder le «mot de contrôle» dans «cible» et gérer les exceptions numériques non masquées.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 848
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 494 à 496.

Assembleur 80x86 FNSTENV

INTEL MPU 8087+ *Float Not Waiting ST in ENVironment*

Syntaxe

FNSTENV *cible*

Description

Cette instruction permet de copier l'environnement du coprocesseur mathématique vers une *cible* mais sans toutefois attendre que l'exception de nombre réel (virgule flottante) soit effacé.

Algorithme

cible(FPUControlWord) ← FPUControlWord *cible*(FPUStatusWord) ← FPUStatusWord *cible*(FPUTagWord) ← FPUTagWord *cible*(FPUDataPointer) ← FPUDataPointer *cible*(FPUInstructionPointer) ← FPUInstructionPointer *cible*(FPULastInstructionOpcode) ← FPULastInstructionOpcode

Mnémonique

Instruction	Opcode	Description
FNSTENV *mem*	D9h /6	Cette instruction permet de copier l'environnement du coprocesseur mathématique vers une *cible* mais sans toutefois attendre que l'exception de nombre réel (virgule flottante) soit effacé.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 848

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 497 à 499.

Assembleur 80x86	**FNSTSW**
INTEL MPU 8087+	

Syntaxe

FNSTSW *cible*

Description

Cette instruction permet de copier le mot d'état du coprocesseur mathématique vers une *cible* mais sans toutefois attendre que l'exception de nombre réel (virgule flottante) soit effacé.

Algorithme

cible ← FPUStatusWord

Mnémonique

Instruction	Opcode	Description	Prérequis
FNSTSW *mem16*	DDh /7	Cette instruction permet de copier le mot d'état du coprocesseur mathématique vers une *cible* mais sans toutefois attendre que l'exception de nombre réel (virgule flottante) soit effacé.	INTEL 8087+
FNSTSW AX	DFh E0h	Cette instruction permet de copier le	INTEL 80287+

		mot d'état du coprocesseur mathématique vers le registre AX mais sans toutefois attendre que l'exception de nombre réel (virgule flottante) soit effacé.	

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 848
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 500 à 502.

Assembleur 80x86	FPATAN
INTEL MPU 8087+	

Syntaxe

```
FPATAN
```

Description

Cette instruction permet de calculer le résultat de l'arc tangente de *ST(1)* par *ST(0)* et de mettre le résultat dans le registre *ST(0)*.

Algorithme

```
ST(1) ← ARCTAN(ST(1) / ST(0))
Désempile la register de pile FPU
```

Mnémonique

Instruction	Opcode	Description
FPATAN	D9h F3h	Cette instruction permet de calculer le résultat de l'arc tangente de *ST(1)* par *ST(0)* et de mettre le résultat dans le registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 848
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 461 à 463.

Assembleur 80x86 FPREM

INTEL MPU 8087+

Syntaxe

```
FPREM
```

Description

Cette instruction permet de diviser le registre *ST(0)* par le registre *ST(1)* et enregistre le reste dans registre *ST(0)*.

Algorithme

```
D ← exponent(ST(0)) - exponent(ST(1))
SI D < 64 ALORS
  Q ← [ TruncateTowardZero(ST(0) / ST(1)) ]
  ST(0) ← ST(0) - (ST(1) * Q)
  C2 ← 0
  C0, C3, C1 ← LeastSignificantBits(Q)
SINON
  C2 ← 1
  N ← un implémentation d'un nombre indépendant entre 32 et 63
  QQ ← [ Tronque autour de zéro((ST(0) / ST(1)) / 2(D-N))) ]
  ST(0) ← ST(0) - (ST(1) * QQ * 2(D-N))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FPREM	D9h F8h	Cette instruction permet de diviser le registre *ST(0)* par le registre *ST(1)* et enregistre le reste dans registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 849

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 464 à 466.

Syntaxe

FPREM1

Description

Cette instruction permet de diviser le registre *ST(0)* par le registre *ST(1)* et enregistre le reste *IEEE* dans registre *ST(0)*.

Mnémonique

Instruction	Opcode	Description
FPREM1	D9h F5h	Cette instruction permet de diviser le registre *ST(0)* par le registre *ST(1)* et enregistre le reste *IEEE* dans registre *ST(0)*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 467 à 469.

Syntaxe

```
FPTAN
```

Description

Cette instruction permet de diviser le registre *ST(0)* par le registre *ST(1)* et enregistre le reste dans registre *ST(0)*.

Algorithme

```
SI ST(0) < 263 ALORS
  C2 ← 0
  ST(0) ← TAN(ST(0))
  TOP ← TOP - 1
  ST(0) ← 1.0
SINON
  C2 ← 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FPTAN	D9h F2h	Cette instruction permet de diviser le registre *ST(0)* par le registre *ST(1)* et enregistre le reste dans registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 849
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 470 à 472.

Syntaxe

FRICHOP

Description

Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode *CHOP*.

Algorithme

ST ← arrondissement(ST,CHOP)

Mnémonique

Instruction	Opcode	Description
FRICHOP	DDh FCh	Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode *CHOP*.

Syntaxe

FRINEAR

Description

Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode du plus proche.

Algorithme

ST ← arrondissement(ST,NEAREST)

Mnémonique

Instruction	Opcode	Description
FRINEAR	DFh FCh	Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode du plus proche.

Assembleur 80x86 — FRINT2

Cyrix 486 avec FPU — *FPU: Round to Integer Near*

Syntaxe

FRINT2

Description

Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode du plus proche et si le nombre est exactement la moitié, alors l'instruction arrondit au signe de l'infini.

Algorithme

```
SI exactement la moitié ALORS
  ST ← SIGN(ST) x arrondissement( | ST | + 0,5 ; NEAREST)
SINON
  ST ← arrondissement( ST,NEAREST )
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FRINT2	DBh FCh	Cette instruction permet d'effectuer le calcul de l'arrondissement de *ST* avec 0 décimal selon la méthode du plus proche et si le nombre est exactement la moitié, alors l'instruction arrondit au signe de l'infini.

Assembleur 80x86	**FRNDINT**
INTEL MPU 8087+	*Float RouND INTeger*

Syntaxe

```
FRNDINT
```

Description

Cette instruction permet d'arrondir le registre *ST(0)* à l'entier le plus proche et enregistre son résultat dans le registre *ST(0)*.

Algorithme

```
ST(0) ← [ ST(0) ]
```

Mnémonique

Instruction	Opcode	Description
FRNDINT	D9h FCh	Cette instruction permet d'arrondir le registre *ST(0)* à l'entier le plus proche et enregistre son résultat dans le registre *ST(0)*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 473 à 474.

Syntaxe

FRSTOR *source*

Description

Cette instruction permet de restituer l'état du coprocesseur précédamment sauvegardé dans l'emplacement mémoire spécifié par *source*.

Algorithme

```
FPUControlWord ← source[FPUControlWord)
FPUStatusWord ← source[FPUStatusWord)
FPUTagWord ← source[FPUTagWord)
FPUDataPointer ← source[FPUDataPointer)
FPUInstructionPointer ← source[FPUInstructionPointer)
FPULastInstructionOpcode ← source[FPULastInstructionOpcode)
ST(0) ← source[ST(0)]
ST(1) ← source[ST(1)]
ST(2) ← source[ST(2)]
ST(3) ← source[ST(3)]
ST(4) ← source[ST(4)]
ST(5) ← source[ST(5)]
ST(6) ← source[ST(6)]
ST(7) ← source[ST(7)]
```

Mnémonique

Instruction	Opcode	Description
FRSTOR *mem*	DDh /4	Cette instruction permet de restituer l'état du coprocesseur précédamment sauvegardé dans l'emplacement mémoire spécifié par *source*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 849
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 475 à 477.

Syntaxe

FSAVE *source*

Description

Cette instruction permet de sauvegarder l'état courant du coprocesseur dans l'emplacement mémoire spécifié par *source*.

Algorithme

```
source(FPUControlWord) ← FPUControlWord
source(FPUStatusWord) ← FPUStatusWord
source(FPUTagWord) ← FPUTagWord
source(FPUDataPointer) ← FPUDataPointer
source(FPUInstructionPointer) ← FPUInstructionPointer
source(FPULastInstructionOpcode) ← FPULastInstructionOpcode
source(ST(0)) ← ST(0)
source(ST(1)) ← ST(1)
source(ST(2)) ← ST(2)
source(ST(3)) ← ST(3)
source(ST(4)) ← ST(4)
source(ST(5)) ← ST(5)
source(ST(6)) ← ST(6)
source(ST(7)) ← ST(7)
FPUControlWord ← 037Fh
FPUStatusWord ← 0
FPUTagWord ← FFFFh
FPUDataPointer ← 0
FPUInstructionPointer ← 0
FPULastInstructionOpcode ← 0
```

Mnémonique

Instruction	Opcode	Description
FSAVE *mem*	9Bh DDh /6	Cette instruction permet de sauvegarder l'état courant du

		coprocesseur dans l'emplacement mémoire spécifié par *source*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 849
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 478 à 481.

Assembleur 80x86	FSCALE
INTEL MPU 8087+	*Float Scale*

Syntaxe

FSCALE

Description

Cette instruction permet d'effectuer la multiplication du registre *ST(0)* par 2 puissance le registre *ST(1)* et sauvegarde le résultat dans le registre *ST(0)*.

Algorithme

$ST(0) \leftarrow ST(0) \times 2^{ST(1)}$

Mnémonique

Instruction	Opcode	Description
FSCALE	D9h FDh	Cette instruction permet d'effectuer la multiplication du registre *ST(0)* par 2 puissance le registre *ST(1)* et sauvegarde le résultat dans le registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 849
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 482 à 483.

Assembleur 80x86	FSETPM
INTEL MPU 80287	*Float SETting Protected Mode*

Syntaxe

```
FSETPM
```

Description

Cette instruction permet de faire passer le coprocesseur mathématique en mode protégé.

Mnémonique

Instruction	Opcode	Description
FSETPM	DBh E4h	Cette instruction permet de faire passer le coprocesseur mathématique en mode protégé.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 849

Syntaxe

```
FSIN
```

Description

Cette instruction permet d'effectuer le calcule de la fonction trigonométrique du sinus du registre *ST(0)* et le copie dans le registre *ST(1)*, le registre *ST(0)* prend la valeur 1,0.

Algorithme

```
SI ST(0) < 2^63 ALORS
  C2 ← 0
  ST(0) ← sin(ST(0))
SINON
  C2 ← 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FSIN	D9h FEh	Cette instruction permet d'effectuer le calcule de la fonction trigonométrique du sinus du registre *ST(0)* et le copie dans le registre *ST(1)*, le registre *ST(0)* prend la valeur 1,0.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 850
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 484 à 485.

Assembleur 80x86 — FSINCOS

INTEL MPU 8087+ — *Float SINus COSinus*

Syntaxe

```
FSINCOS
```

Description

Cette instruction permet d'effectuer le calcule de la fonction trigonométrique du sinus et du cosinus du registre *ST(0)* puis copie le résultat du sinus dans le registre *ST(0)* et le cosinus dans le registre *ST(1)*.

Algorithme

```
SI ST(0) < 263 ALORS
  C2 ← 0
  TEMP ← COS(ST(0))
  ST(0) ← SIN(ST(0))
  TOP ← TOP - 1
  ST(0) ← TEMP
SINON
  C2 ← 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
FSINCOS	D9h FBh	Cette instruction permet d'effectuer le calcule de la fonction trigonométrique du sinus et du cosinus du registre *ST(0)* puis copie le résultat du sinus dans le registre *ST(0)* et le cosinus dans le registre *ST(1)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 850

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 486 à 489.

Assembleur 80x86	**FSQRT**
INTEL MPU 8087+	*Float SQuaRed rooT*

Syntaxe

```
FSQRT
```

Description

Cette instruction permet d'extraire la racine carrée du registre *ST(0)* et de copier son résultat dans le registre *ST(0)*.

Algorithme

```
ST(0) ← √ (ST(0))
```

Mnémonique

Instruction	**Opcode**	**Description**
FSQRT	D9h FAh	Cette instruction permet d'extraire la racine carrée du registre *ST(0)* et de copier son résultat dans le registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 850
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 489 à 490.

Assembleur 80x86	FST
INTEL MPU 8087+	*Float in ST(0)*

Syntaxe

FST *cible*

Description

Cette instruction permet de copier la valeur réel contenu le registre *ST(0)* vers une *cible*.

Algorithme

cible ← ST(0)

Mnémonique

Instruction	Opcode	Description
FST *mem32*	D9h /2	Cette instruction permet de copier la valeur réel contenu le registre *ST(0)* vers une *cible*.
FST *mem64*	DDh /2	Cette instruction permet de copier la valeur réel contenu le registre *ST(0)* vers une *cible*.
FST *fpureg*	DDh (D0h+r)	Cette instruction permet de copier la valeur réel contenu le registre *ST(0)* vers une *cible*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 850
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 491 à 493.

Assembleur 80x86 — FSTCW

INTEL MPU 8087+ — *Float ST in Control Word*

Syntaxe

FSTCW *cible*

Paramètres

Nom	Description
cible	Ce paramètre permet d'indiquer l'opérande reçevant le résultat du mot de contrôle *CW*

Description

Cette instruction permet de copier le mot de contrôle *CW* vers une *cible*.

Algorithme

cible ← FPUControlWord

Mnémonique

Instruction	Opcode	Description
FSTCW *mem16*	9Bh D9h /7	Cette instruction permet de copier le mot de contrôle *CW* vers une *cible*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 850
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 494 à 496.

Assembleur 80x86	FSTENV
INTEL MPU 8087+	*Float ST in ENVironment*

Syntaxe

FSTENV *cible*

Description

Cette instruction permet de copier l'environnement du coprocesseur mathématique vers une *cible*.

Algorithme

cible(FPUControlWord) ← FPUControlWord
cible(FPUStatusWord) ← FPUStatusWord
cible(FPUTagWord) ← FPUTagWord
cible(FPUDataPointer) ← FPUDataPointer
cible(FPUInstructionPointer) ← FPUInstructionPointer
cible(FPULastInstructionOpcode) ← FPULastInstructionOpcode

Mnémonique

Instruction	Opcode	Description
FSTENV *mem*	9Bh D9h /6	Cette instruction permet de copier l'environnement du coprocesseur mathématique vers une *cible*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 850
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 497 à 499.

Assembleur 80x86	**FSTP**
INTEL MPU 8087+	*Float in ST & Position*

Syntaxe

FSTP *cible*

Description

Cette instruction permet de copier un nombre entier et prendre le contenu du registre *ST(0)* dans la pile de registres (dépilage).

Algorithme

cible ← ST(0)
Desempile le registre de pile

Mnémonique

Instruction	Opcode	Description
FSTP *mem32*	D9h /3	Cette instruction permet de copier un nombre entier et prendre le contenu du registre *ST(0)* dans la pile de registres (dépilage).
FSTP *mem64*	DDh /3	Cette instruction permet de copier un nombre entier et prendre le contenu du registre *ST(0)* dans la pile de registres (dépilage).
FSTP *mem80*	DBh /7	Cette instruction permet de copier un nombre entier et prendre le contenu du registre *ST(0)* dans la pile de registres (dépilage).
FSTP *fpureg*	DDh (D8h+r)	Cette instruction permet de copier un nombre entier et prendre le contenu du registre *ST(0)* dans la pile de registres (dépilage).

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 850
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 491 à 493.

Assembleur 80x86	FSTSW
INTEL MPU 8087+	

Syntaxe

FSTSW *cible*

Description

Cette instruction permet de copier le mot d'état du coprocesseur mathématique vers une *cible*.

Algorithme

cible ← FPUStatusWord

Mnémonique

Instruction	Opcode	Description	Prérequis
FSTSW *mem16*	9Bh DDh /7	Cette instruction permet de copier le mot d'état du coprocesseur mathématique vers une *cible*.	INTEL 8087+
FSTSW AX	9Bh DFh E0h	Cette instruction permet de copier le mot d'état du coprocesseur mathématique vers le registre AX.	INTEL 80287+

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 851
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 500 à 502.

Assembleur 80x86	**FSUB**
INTEL MPU 8087+	*Float Subtract*

Syntaxe

FSUB *source*

Description

Cette instruction permet de soustraire un nombre entier du registre *ST(0)*.

Algorithme

ST ← ST - source

Mnémonique

Instruction	**Opcode**	**Description**
FSUB *mem32*	D8h /4	Cette instruction permet de soustraire un nombre entier du registre *ST(0)*.
FSUB *mem64*	DCh /4	Cette instruction permet de soustraire un nombre entier du registre *ST(0)*.
FSUB *fpureg*	D8h (E0h+r)	Cette instruction permet de soustraire un nombre entier du registre *ST(0)*.
FSUB ST0,*fpureg*	D8h (E0h+r)	Cette instruction permet de soustraire un nombre entier du

		registre *ST(0)*.
FSUB TO *fpureg*	DCh (E8h+r)	Cette instruction permet de soustraire un nombre entier du registre *ST(0)*.
FSUB *fpureg*,ST0	DCh (E8h+r)	Cette instruction permet de soustraire un nombre entier du registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 851
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 503 à 506.

Assembleur 80x86	**FSUBP**
INTEL MPU 8087+	*Float Subtract & Pointer*

Syntaxe

FSUBP *source*

Description

Cette instruction permet de soustraire un nombre entier du registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.

Algorithme

ST ← *ST* - *source*
Désempile le registre de pile

Mnémonique

Instruction	Opcode	Description
FSUBP *fpureg*	DEh (E8h+r)	Cette instruction permet de soustraire un nombre entier du registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.
FSUBP *fpureg*,ST0	DEh (E8h+r)	Cette instruction permet de soustraire un nombre entier du registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 503 à 506.

Syntaxe

FSUBPP *source*

Description

Cette instruction permet de soustraire un nombre entier du registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.

Remarque

- L'utilisation de cette instruction est déconseillée, il est préférable d'utiliser plutôt «FSUBP» car cette instruction est très peu supportée par les compilateurs.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 851

Syntaxe

FSUBR *source*

Description

Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)*.

Algorithme

ST ← source - ST

Mnémonique

Instruction	Opcode	Description
FSUBR *mem32*	D8h /5	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)*.
FSUBR *mem64*	DCh /5	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)*.
FSUBR *fpureg*	D8h (E8h+r)	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)*.
FSUBR ST0,*fpureg*	D8h (E8h+r)	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)*.
FSUBR TO *fpureg*	DCh (E0h+r)	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)*.
FSUBR *fpureg*,ST0	DCh (E0h+r)	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 851
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 507 à 510.

Assembleur 80x86	FSUBRP
INTEL MPU 8087+	

Syntaxe

FSUBRP *source*

Description

Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.

Algorithme

ST ← source - ST
Désempile le registre de pile

Mnémonique

Instruction	Opcode	Description
FSUBRP *fpureg*	DEh (E0h+r)	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.
FSUBRP *fpureg*,ST0	DEh (E0h+r)	Cette instruction permet de soustraire le registre *ST(0)* du nombre entier et place le résultat dans le registre *ST(0)* et copie le registre *ST(0)* dans la pile de registres.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 851
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 507 à 510.

Syntaxe

FTST

Description

Cette instruction permet de comparer le sommet de la pile avec la valeur 0,0 et définit les indicateurs d'états C0 et C3 de façon approprié.

Algorithme

```
EVALUER CAS relation de l'opérande DE
  CAS Pas comparable:
    C3 ← 1
    C2 ← 1
    C0 ← 1
  CAS ST(0) > 0.0:
    C3 ← 0
    C2 ← 0
    C0 ← 0
  CAS ST(0) < 0.0:
    C3 ← 0
    C2 ← 0
    C0 ← 1
  CAS ST(0) 0.0:
    C3 ← 1
    C2 ← 0
    C0 ← 0
FIN EVALUER CAS
```

Mnémonique

Instruction	Opcode	Description
FTST	D9h E4h	Cette instruction permet de comparer le sommet de la pile avec la valeur 0,0 et définit les indicateurs d'états C0 et C3 de façon approprié.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 851
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 511 à 512.

Assembleur 80x86	**FUCOM**
INTEL MPU 80387+	*Float Unordered Compare*

Syntaxe

FUCOM *fpureg*
FUCOM ST0,*fpureg*

Description

Cette instruction compare le registre *ST0* avec un opérande et fixe les drapeaux de façon approprié.

Algorithme

```
EVALUER CAS (relation de l'opérande) DE
  CAS ST(0) > ST(i):
    ZF ← 0
    PF ← 0
    CF ← 0
  CAS ST(0) < ST(i):
    ZF ← 0
    PF ← 0
    CF ← 1
  CAS ST(0) = ST(i):
    ZF ← 1
    PF ← 0
    CF ← 0
FIN EVALUER CAS
```

Mnémonique

Instruction	Opcode	Description
FUCOM *fpureg*	DDh (E0h+r)	Cette instruction compare le registre *ST0* avec un opérande et fixe les drapeaux de façon approprié.
FUCOM ST0,*fpureg*	DDh (E0h+r)	Cette instruction compare le registre *ST0* avec un opérande et fixe les drapeaux de façon approprié.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 513 à 515.

Assembleur 80x86 — FUCOMI

INTEL Pentium III+ — *Compare Real and Set EFLAGS*

Syntaxe

FUCOMI ST, ST(*i*)

Description

Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et vérifie l'ordre des valeurs et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats.

Mnémonique

Instruction	Opcode	Description
FUCOMI ST0,ST*i*	DBh (E8h+*i*)	Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et vérifie l'ordre des valeurs et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 414 à 416.

Assembleur 80x86 — FUCOMIP

INTEL Pentium III+ — *Compare Real and Set EFLAGS*

Syntaxe

FUCOMIP ST, ST(*i*)

Description

Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et vérifie l'ordre des valeurs et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats et désempile de la pile la valeur dans le registre.

Mnémonique

Instruction	Opcode	Description
FUCOMIP ST0,ST*i*	DFh (E8h+*i*)	Cette instruction permet d'effectuer la comparaison de ST(0) avec ST(i) et vérifie l'ordre des valeurs et fixe la valeur drapeaux *ZF*, *PF* et *CF* du registre *EFLAGS* en fonction des résultats et désempile de la pile la valeur dans le registre.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 414 à 416.

Assembleur 80x86	**FUCOMP**
INTEL MPU 80387+	*Float Unordered Compare Pop Stack*

Syntaxe

FUCOMP *fpureg*
FUCOMP ST0,*fpureg*

Description

Cette instruction compare le registre *ST0* avec un opérande et fixe les drapeaux de façon approprié et désempile la pile.

Mnémonique

Instruction	**Opcode**	**Description**
FUCOMP *fpureg*	DDh (E8h+r)	Cette instruction compare le registre *ST0* avec un opérande et fixe les drapeaux de façon approprié et désempile la pile.
FUCOMP ST0,*fpureg*	DDh (E8h+r)	Cette instruction compare le registre *ST0* avec un opérande et fixe les drapeaux de façon approprié et désempile la pile.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 513 à 515.

Assembleur 80x86	**FUCOMPP**
INTEL MPU 80387+	*Float Unordered Compare Pop & Pop Stack*

Syntaxe

FUCOMPP

Description

Cette instruction compare le registre *ST0* avec le registre *ST1* et fixe les drapeaux de façon approprié et désempile les registres *ST0* et *ST1* de la pile.

Mnémonique

Instruction	**Opcode**	**Description**
FUCOMPP	DAh E9h	Cette instruction compare le registre *ST0* avec le registre *ST1* et fixe les drapeaux de façon approprié et désempile les registres *ST0* et *ST1* de la pile.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 513 à 515.

Assembleur 80x86	FWAIT
INTEL MPU 8087+	*Float WAIT*

Syntaxe

```
FWAIT
```

Description

Cette instruction permet d'attendre la fin de l'exécution d'une commande avant de poursuivre.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 851

Assembleur 80x86	**FXAM**
INTEL MPU 8087+	*Float eXAMine*

Syntaxe

FXAM

Description

Cette instruction permet d'examiner le sommet de la pile et définit les indicateurs d'état C0 et C3 de façon approprié.

Algorithme

```
C1 ← signe du bit de ST * 0 pour positif, 1 pour négatif
EVALUER CAS (valeur de la classe ou nombre dans ST(0)) DE
  CAS Non supporté:
    C3 ← 0
    C2 ← 0
    C0 ← 0
  CAS NaN:
    C3 ← 0
    C2 ← 0
    C0 ← 1
  CAS Normal:
    C3 ← 0
    C2 ← 1
    C0 ← 0
  CAS Infini:
    C3 ← 0
    C2 ← 1
    C0 ← 1
  CAS Zéro:
    C3 ← 1
    C2 ← 0
    C0 ← 0
```

CAS Vide:
C3 ← 1
C2 ← 0
C0 ← 1
CAS Anormal:
C3 ← 1
C2 ← 1
C0 ← 0
FIN EVALUER CAS

Mnémonique

Instruction	Opcode	Description
FXAM	D9h E5h	Cette instruction permet d'examiner le sommet de la pile et définit les indicateurs d'état C0 et C3 de façon approprié.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 852
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 516 à 517.

Assembleur 80x86	**FXCH**
INTEL MPU 8087+	*Float eXchange*

Syntaxe

FXCH
FXCH *fpureg*
FXCH *fpureg*,ST0
FXCH ST0,*fpureg*

Description

Cette instruction permet d'échanger le contenu d'une expression avec celle du registre *ST(0)* s'il est omis.

Algorithme

```
SI nombre d'opérande est 1 ALORS
  temp ← ST(0)
  ST(0) ← fpureg
  fpureg ← temp
SINON
  temp ← ST(0)
  ST(0) ← ST(1)
  ST(1) ← temp
FIN SI
```

Mnémonique

Instruction	Opcode	Description

FXCH	D9h C9h	Cette instruction permet d'échanger le contenu d'une expression avec celle du registre *ST(0)* s'il est omis.
FXCH *fpureg*	D9h (C8h+r)	Cette instruction permet d'échanger le contenu d'une expression avec celle du registre *ST(0)* s'il est omis.
FXCH *fpureg*	D9h C9h	Cette instruction permet d'échanger le contenu d'une expression avec celle du registre *ST(0)* s'il est omis.
FXCH *fpureg*,ST0	D9h (C8h+r)	Cette instruction permet d'échanger le contenu d'une expression avec celle du registre *ST(0)* s'il est omis.
FXCH ST0,*fpureg*	D9h C8h+r	Cette instruction permet d'échanger le contenu d'une expression avec celle du registre *ST(0)* s'il est omis.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 852
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 518 à 519.

Assembleur 80x86 — FXRSTOR

INTEL MMX-2+ — *Fast Restore*

Syntaxe

FXRSTOR *source*

Description

Cette instruction permet de restituer rapidement 94 (en mode 16 bits) ou 108 (en mode 32 bits) octets d'un contexte de coprocesseur mathématique vers la mémoire.

Mnémonique

Instruction	Opcode	Description
FXRSTOR *mem512byte*	0Fh AEh *mm*001*mmm*	Cette instruction permet de restituer rapidement 94 (en mode 16 bits) ou 108 (en mode 32 bits) octets d'un contexte de coprocesseur mathématique vers la mémoire.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 520 à 523.

Assembleur 80x86 FXSAVE

INTEL MMX-2+ *Fast Save*

Syntaxe

FXSAVE *source*

Description

Cette instruction permet de sauvegarder rapidement 94 (en mode 16 bits) ou 108 (en mode 32 bits) octets d'un contexte de coprocesseur mathématique de la mémoire.

Mnémonique

Instruction	Opcode	Description
FXSAVE *mem512byte*	0Fh AEh *mm*000*mmm*	Cette instruction permet de sauvegarder rapidement 94 (en mode 16 bits) ou 108 (en mode 32 bits) octets d'un contexte de coprocesseur mathématique de la mémoire.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 524 à 534.

Assembleur 80x86	FXTRACT
INTEL MPU 8087+	

Syntaxe

```
FXTRACT
```

Description

Cette instruction permet de copier l'exposant du registre *ST(0)* dans le registre *ST(1)* comme nombre entier puis copie la mantisse dans le registre *ST(0)* comme nombre entier.

Algorithme

```
TEMP ← Signification(ST(0))
ST(0) ← Exponent(ST(0))
TOP ← TOP - 1
ST(0) ← TEMP
```

Mnémonique

Instruction	Opcode	Description
FXTRACT	D9h F4h	Cette instruction permet de copier l'exposant du registre *ST(0)* dans le registre *ST(1)* comme nombre entier puis copie la mantisse dans la registre *ST(0)* comme nombre entier.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 852
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 535 à 536.

Assembleur 80x86	**FYL2X**
INTEL MPU 8087+	

Syntaxe

FYL2X

Description

Cette instruction permet de calculer la multiplication du registre *ST(1)* par le logarithme du registre *ST(0)* dans la base 2 et copie le résultat dans le registre *ST(0)*.

Algorithme

$LOG\ b^x \leftarrow (LOG\ 2^b)\text{-}1 * LOG\ 2^x$

Mnémonique

Instruction	Opcode	Description
FYL2X	D9h F1h	Cette instruction permet de calculer la multiplication du registre *ST(1)* par le logarithme du registre *ST(0)* dans la base 2 et copie le résultat dans le registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 852
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 537 à 538.

Assembleur 80x86	**FYL2XP1**
INTEL MPU 8087+	

Syntaxe

FYL2XP1

Description

Cette instruction permet de calculer la multiplication du registre *ST(1)* par le logarithme du registre *ST(0)* plus 1 dans la base 2 et copie le résultat dans le registre *ST(0)*.

Algorithme

$ST(1) \leftarrow ST(1) \times \log 2^{ST(0)}$ Désempile le registre de pile

Mnémonique

Instruction	**Opcode**	**Description**
FYL2XP1	D9h F9h	Cette instruction permet de calculer la multiplication du registre *ST(1)* par le logarithme du registre *ST(0)* plus 1 dans la base 2 et copie le résultat dans le registre *ST(0)*.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 852
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 539 à 540.

Assembleur 80x86 HADDPD

SSE4.1 *Packed Double-FP Horizontal Add*

Syntaxe

HADDPD *dest,source*

Description

Cette instruction permet d'effectuer l'addition horizontale de paquet de nombre réel de double précision des deux opérandes.

Algorithme

dest(63..0) ← *dest*(63..0) + *dest*(127..64) *dest*(127..64) ← *source*(63..0) + *source*(127..64)

Mnémonique

Instruction	Opcode	Description
HADDPD *xmm1,xmm2/m128*	66h 0Fh 7Ch /r	Cette instruction permet d'effectuer l'addition horizontal de paquet de nombre réel de double précision des deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 541 à 544.

Assembleur 80x86 HADDPS

SSE4.1 *Packed Single-FP Horizontal Add*

Syntaxe

HADDPS *dest,source*

Description

Cette instruction permet d'effectuer l'addition horizontale de paquet de nombre réel de simple précision des deux opérandes.

Algorithme

dest(31..0) ← *dest*(31..0) + *dest*(63..32) *dest*(63..32) ← *dest*(95..64) + *dest*(127..96) *dest*(95..64) ← *source*(31..0) + *source*(63..32) *dest*(127..96) ← *source*(95..64) + *source*(127..96)

Mnémonique

Instruction	Opcode	Description
HADDPS *xmm1,xmm2/m128*	F2h 0Fh 7Ch /r	Cette instruction permet d'effectuer l'addition horizontale de paquet de nombre réel de simple précision des deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 545 à 548.

Assembleur 80x86	HLT
INTEL 8088+	*HaLT*

Syntaxe

```
HLT
```

Description

Cette instruction permet de faire passer le microprocesseur en mode d'arrêt. Toutefois, le processeur peut quitter cet état lorsqu'une ligne matérielle RESET ou lorsqu'une interruption non-masquable (*NMI*) reçoit un signal.

Mnémonique

Instruction	Opcode	Description
HLT	F4h	Exécute l'instruction d'arrêt

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)		X	X	Le CPL ne vaut pas 0.

Voir également

Instruction assembleur 80x86 - Instruction STI
Instruction assembleur 80x86 - Instruction CLI

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 811
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 258.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 549 à 550.

Assembleur 80x86 — HSUBPD

INTEL Pentium 4 (SSE3)+ — *Packed Double-FP Horizontal Subtract*

Syntaxe

HSUBPD *destination*, *source*

Description

Cette instruction permet d'effectuer la soustraction horizontale de paquet de nombre réel de double précision des deux opérandes.

Algorithme

destination(0..63) ← *destination*(0..63) - *destination*(64..127)
destination(64..127) ← *source*(0..63) - *source*(64..127)

Mnémonique

Instruction	Opcode	Description
HSUBPD *xmm1*, *xmm2/m128*	66h 0Fh 7Dh /r	Cette instruction permet de soustraire horizontalement les paquets de simple précisions de *xmm2/m128* à *xmm1*

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 551 à 555.

Assembleur 80x86 HSUBPS

INTEL Pentium 4 (SSE3)+ *Packed Single-FP Horizontal Subtract*

Syntaxe

HSUBPS *destination*, *source*

Description

Cette instruction permet d'effectuer la soustraction horizontale de paquet de nombre réel de double précision des deux opérandes.

Algorithme

destination(0..31) ← *destination*(0..31) - *destination*(32..63) *destination*(32..63) ← *destination*(64..95) - *destination*(96..127) *destination*(64..95) ← *source*(0..31) - *source*(32..63) *destination*(96..127) ← *source*(64..95) - *source*(96..127)

Mnémonique

Instruction	Opcode	Description
HSUBPS *xmm1*, *xmm2/m128*	F2h 0Fh 7Dh /r	Cette instruction permet d'effectuer la soustraction horizontal de paquet de nombre réel de double précision des deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 555 à 558.

Assembleur 80x86	**IBTS**
INTEL 80386+	*Insert Bit String*

Syntaxe

IBTS *valeur,reg16*
IBTS *valeur,reg32*

Description

Cette instruction permet de faire un tableau de bits du deuxième opérand et le met dans le premier.

Mnémonique

Instruction	**Opcode**	**Description**
IBTS *r/m16,reg16*	0Fh A7h /r	Cette instruction permet de faire un tableau de bits du deuxième opérand et le met dans le premier.
IBTS *r/m32,reg32*	66h 0Fh A7h /r	Cette instruction permet de faire un tableau de bits du deuxième opérand et le met dans le premier.

ICEBP

In-Circuit Emulator

Syntaxe

```
ICEBP
```

Description

Cette instruction permet de passer en mode d'instruction *ICE* (*In-Circuit Emulator*) si le bit 12 du registre DR7 vaut 1 sinon il exécute l'interruption 1.

Algorithme

```
SI DR7(12) = 1 ALORS
   passe au mode d'instruction ICE
SINON
   INT 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
ICEBP	F1h	Cette instruction permet de passer en mode d'instruction *ICE* (*In-Circuit Emulator*) si le bit 12 du registre DR7 vaut 1 sinon il exécute l'interruption 1.

Assembleur 80x86	**IDIV**
INTEL 8088+	*Division entière*

Syntaxe

IDIV *Opérande*

Description

Cette instruction permet d'effectuer une division signée (nombre entier). Le dividende est implicite; il est ajuster en fonction de la taille du diviseur. Le restant est toujours plus petit que le diviseur. Le type de diviseur détermine quel registre l'instruction utilisera:

Taille	Dividende	Diviseur	Quotient	Restant
Octet	AX	*Opérande*	AL	AH
Mot	DX:AX	*Opérande*	AX	DX
Double mot	EDX:EAX	*Opérande*	EAX	EDX

Algorithme

SI *Opérande* = 0 **ALORS**
  Interruption 0
SINON
  SI *Opérande* 8 bits **ALORS**
    AL ← *AX* ÷ *Opérande*
    AH ← *AX* MOD *Opérande*
  FIN SI
  SI *Opérande* 16 bits **ALORS**
    AX ← ((*DX* x 65536) + *AX*) ÷ *Opérande*
    DX ← ((*DX* x 65536) + *AX*) MOD *Opérande*
  FIN SI
  SI *Opérande* 32 bits **ALORS**
    EAX ← ((*EDX* x 65536 x 65536) + *EAX*) ÷ *Opérande*

```
    EDX ← ( ( EDX x 65536 x 65536 ) + EAX ) MOD Opérande
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
IDIV *reg/mem8*	F6h /7	Division entière de AX par le contenu d'un emplacement mémoire ou registre 8 bits et entrepose le quotient dans AL et restant dans AH.
IDIV *reg/mem16*	F7h /7	Division entière de DX:AX par le contenu d'un emplacement mémoire ou registre 16 bits et entrepose le quotient dans AX et restant dans DX.
IDIV *reg/mem32*	F7h /7	Division entière de EDX:EAX par le contenu d'un emplacement mémoire ou registre 32 bits et entrepose le quotient dans EAX et restant dans EDX.
IDIV *reg/mem64*	F7h /7	Division entière de RDX:RAX par le contenu d'un emplacement mémoire ou registre 64 bits et entrepose le quotient dans RAX et restant dans RDX.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#DE(Division par zéro)	X	X	X	L'opérande de diviseur vaut

				0.
	X	X	X	Le quotient est trop large pour le registre désigné.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie		X	X	Un désalignement

l'alignement)				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction IMUL
Instruction assembleur 80x86 - Instruction DIV

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 811
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 404
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 113 à 114.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 559 à 562.

Assembleur 80x86	**IMUL**
INTEL 8088+	*Multiplication entière*

Syntaxe

IMUL *Opérande*	INTEL 8088+
IMUL *dest*, *src*	INTEL 80286+
IMUL *dest*, *src*, *imm*	INTEL 80386+

Description

Cette instruction permet d'effectuer une multiplication signée (nombre entier). Le multiplicateur est implicite; il est ajuster en fonction de la taille de la base. Le produit est toujours plus grand que le multiplicateur. Le type de multiplication détermine quel registre l'instruction utilisera:

Taille	Base	Multiplicateur	Résultat
Octet	AL	*Opérande*	AX
Mot	AX	*Opérande*	DX:AX
Double mot	EAX	*Opérande*	EDX:EAX

Algorithme

```
SI Nombre d'opérande = 1 ALORS
  SI Taille de l'opérande en bits = 8 ALORS
    AX ← AL x Opérande
    SI (AH = 00h) OU (AH = FFh) ALORS
      CF ← 0
      OF ← 0
    SINON
      CF ← 1
```

```
    OF ← 1
  FIN SI
SINON SI Taille de l'opérande en bits = 16 ALORS
    DX:AX ← AX x Opérande
    SI (DX = 0000h) OU (DX = FFFFh) ALORS
      CF ← 0
      OF ← 0
    SINON
      CF ← 1
      OF ← 1
    FIN SI
  SINON
    EDX:EAX ← EAX x Opérande
    SI ((EDX = 00000000h) OU (EDX = FFFFFFFFh)) ALORS
      CF ← 0
      OF ← 0
    SINON
      CF ← 1
      OF ← 1
    FIN SI
  FIN SI
SINON SI Nombre d'opérande = 2 ALORS
  temp ← dest x src
  dest ← dest x src
  SI temp = dest ALORS
    CF ← 1
    OF ← 1
  SINON
    CF ← 0
    OF ← 0
  FIN SI
SINON
  temp ← dest x src
  dest ← dest x src
  SI temp = dest ALORS
    CF ← 1
    OF ← 1
  SINON
    CF ← 0
    OF ← 0
  FIN SI
```

```
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
IMUL *reg/mem8*	F6h /5	Multiplie le contenu du registre AL avec une opérande mémoire ou registre de 8 bits et met le résultat entier dans le registre AX.
IMUL *reg/mem16*	F7h /5	Multiplie le contenu du registre AX avec une opérande mémoire ou registre de 16 bits et met le résultat entier dans le registre DX:AX.
IMUL *reg/mem32*	F7h /5	Multiplie le contenu du registre EAX avec une opérande mémoire ou registre de 32 bits et met le résultat entier dans le registre EDX:EAX.
IMUL *reg/mem64*	F7h /5	Multiplie le contenu du registre RAX avec une opérande mémoire ou registre de 64 bits et met le résultat entier dans le registre RDX:RAX.
IMUL *reg16*, *reg/mem16*	0Fh AFh /r	Multiplie le contenu d'un registre de destination 16 bits avec une opérande mémoire ou registre de 16 bits et met le résultat entier dans le registre de destination 16 bits.
IMUL *reg32*, *reg/mem32*	0Fh AFh /r	Multiplie le contenu d'un registre de destination 32 bits avec une opérande mémoire ou registre de 32 bits et met le résultat entier dans le registre de destination 32 bits.

IMUL *reg64*, *reg/mem64*	0Fh AFh /r	Multiplie le contenu d'un registre de destination 64 bits avec une opérande mémoire ou registre de 64 bits et met le résultat entier dans le registre de destination 64 bits.
IMUL *reg16*, *reg/mem16*, *imm8*	6Bh /r *ib*	Multiplie le contenu d'une opérande mémoire ou registre de 16 bits par une valeur entière immédiate de 8 bits et met le résultat entier dans le registre 16 bits.
IMUL *reg32*, *reg/mem32*, *imm8*	6Bh /r *ib*	Multiplie le contenu d'une opérande mémoire ou registre de 32 bits par une valeur entière immédiate de 8 bits et met le résultat entier dans le registre 32 bits.
IMUL *reg64*, *reg/mem64*, *imm8*	6Bh /r *ib*	Multiplie le contenu d'une opérande mémoire ou registre de 64 bits par une valeur entière immédiate de 8 bits et met le résultat entier dans le registre 64 bits.
IMUL *reg16*, *reg/mem16*,*imm16*	69h /r *iw*	Multiplie le contenu d'une opérande mémoire ou registre de 16 bits par une valeur entière immédiate de 16 bits et met le résultat entier dans le registre 16 bits.
IMUL *reg32*, *reg/mem32*,*imm32*	69h /r *id*	Multiplie le contenu d'une opérande mémoire ou registre de 32 bits par une valeur entière immédiate de 32 bits et met le résultat entier dans le registre 32 bits.
IMUL *reg64*, *reg/mem64*,*imm32*	69h /r *id*	Multiplie le contenu d'une opérande mémoire ou registre de 64 bits par une valeur entière immédiate de 32 bits et met le résultat entier dans le

		registre 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique

Voir également

Instruction assembleur 80x86 - Instruction IDIV
Instruction assembleur 80x86 - Instruction MUL

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 812
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 405
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 115.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 563 à 567.

Assembleur 80x86	**IN**
INTEL 8088+	*INput*

Syntaxe

IN *accumulateur*,*imm*
IN *accumulateur*,DX

Description

Cette instruction permet de demander un octet ou un mot provenant du port d'entrée/sortie et le retourne dans le registre accumulateur (AL, AX, EAX).

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    accumulateur ← Port(adresse)
  FIN SI
SINON
  accumulateur ← Port(adresse)
FIN SI
```

Mnémonique

Instruction	Opcode	Description

IN AL, *imm8*	E4h *ib*	Lit un octet dans un port d'entrée/sortie à l'adresse spécifié par l'*imm8* et met le résultat dans le registre AL.
IN AX, *imm8*	E5h *ib*	Lit un mot dans un port d'entrée/sortie à l'adresse spécifié par l'*imm8* et met le résultat dans le registre AX.
IN EAX, *imm8*	E5h *ib*	Lit un double mot dans un port d'entrée/sortie à l'adresse spécifié par l'*imm8* et met le résultat dans le registre EAX.
IN AL, DX	ECh	Lit un octet dans un port d'entrée/sortie à l'adresse spécifié par le registre DX et met le résultat dans le registre AL.
IN AX, DX	EDh	Lit un mot dans un port d'entrée/sortie à l'adresse spécifié par le registre DX et met le résultat dans le registre AX.
IN EAX, DX	EDh	Lit un mot dans un port d'entrée/sortie à l'adresse spécifié par le registre DX et met le résultat dans le registre EAX.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)		X		Un ou plusieurs bits de permission d'entrée/sortie

				sont fixer par le *TSS* pour un accès au port.
			X	Le *CPL* est plus grand que le IOPL et une ou plusieurs bits de permission sont fixer par le *TSS* pour un accès au port.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 812
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 117.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 568 à 569.

Assembleur 80x86	**INC**
INTEL 8088+	*Increment*

Syntaxe

INC *registre*
INC *mémoire*

Description

Cette instruction permet d'incrémenter un registre ou un emplacement mémoire.

Algorithme

operande ← *operande* + 1

Mnémonique

Instruction	**Opcode**	**Description**
INC *reg/mem8*	FEh /0	Incrémente le contenu d'un emplacement registre ou mémoire de 8 bits de 1.
INC *reg/mem16*	FFh /0	Incrémente le contenu d'un emplacement registre ou mémoire de 16 bits de 1.
INC *reg/mem32*	FFh /0	Incrémente le contenu d'un emplacement registre ou mémoire de 32 bits de 1.

INC *reg/mem64*	FFh /0	Incrémente le contenu d'un emplacement registre ou mémoire de 64 bits de 1.
INC *reg16*	40h +rw	Incrémente le contenu d'un registre 16 bits de 1.
INC *reg32*	40h +rd	Incrémente le contenu d'un registre 32 bits de 1.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment

				de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction ADD
Instruction assembleur 80x86 - Instruction DEC

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 812
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 405
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 118.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 570 à 572.

Assembleur 80x86	**INS**
INTEL 80286+	*INput String*

Syntaxe

INS *opérande_cible*,DX

Description

Cette instruction permet de demander un octet, un mot ou un double mot du port d'entrée/sortie et retourne le résultat dans l'adresse ES:[DI] et incrémente/décrémente le registre DI en fonction de la taille de l'opérande cible et de l'état du drapeau de direction. L'adresse du port est contenue obligatoirement dans le registre DX.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    accumulateur ← Port(adresse)
  FIN SI
SINON
  accumulateur ← Port(adresse)
FIN SI
SI (taille de l'opérande_cible = octet) ALORS
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 1
  SINON
    (E)DI ← (E)DI - 1
  FIN SI
SINON SI (taille de l'opérande_cible = mot) ALORS
  SI DF = 0 ALORS
```

```
    (E)DI ← (E)DI + 2
  SINON
    (E)DI ← (E)DI - 2
  FIN SI
SINON
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 4
  SINON
    (E)DI ← (E)DI - 4
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
INS *mem8*, DX	6Ch	Lecture d'un octet dans le port spécifié par le registre DX, et met le résultat dans l'emplacement mémoire spécifié par ES:(R)DI. Enfin, incrémente et décrémente le registre (R)DI.
INS *mem16*, DX	6Dh	Lecture d'un mot dans le port spécifié par le registre DX, et met le résultat dans l'emplacement mémoire spécifié par ES:(R)DI. Enfin, incrémente et décrémente le registre (R)DI.
INS *mem32*, DX	6Dh	Lecture d'un double mot dans le port spécifié par le registre DX, et met le résultat dans l'emplacement mémoire spécifié par ES:(R)DI. Enfin, incrémente et décrémente le registre (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Un ou plusieurs bits de permission d'entrée/sortie sont fixé par le *TSS* pour un accès au port.
		X		Une ou plusieurs permissions bit d'entrée/sortie sont fixées dans le TSS d'un accès de port.
			X	Le *CPL* est plus grand que le IOPL et une ou plusieurs bits de permission sont fixer par le *TSS* pour un accès au port.
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulles est utilisé

				comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activée

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 813
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 120.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 573 à 577.

Assembleur 80x86	**INSB**
INTEL 80286+	*INput String Byte*

Syntaxe

```
INSB
```

Description

Cette instruction permet de demander un octet du port d'entrée/sortie et retourne le résultat dans l'adresse ES:[DI] et incrémente/décrémente le registre DI de 1 en fonction de l'état du drapeau de direction. L'adresse du port est contenue obligatoirement dans le registre DX.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    accumulateur ← Port(adresse)
  FIN SI
SINON
  accumulateur ← Port(adresse)
FIN SI
SI DF = 0 ALORS
  (E)DI ← (E)DI + 1
SINON
  (E)DI ← (E)DI - 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
INSB	6Ch	Lecture d'un octet dans le port spécifié par le registre DX, et met le résultat dans l'emplacement mémoire spécifié par ES:(R)DI. Enfin, incrémente et décrémente le registre (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Un ou plusieurs bits de permission d'entrée/sortie sont fixé par le *TSS* pour un accès au port.
		X		Une ou plusieurs permissions bit d'entrée/sortie sont fixées dans le TSS d'un accès de port.
			X	Le *CPL* est plus grand que l'IOPL et une ou plusieurs bits de permission sont fixés par le *TSS* pour un

				accès au port.
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulles est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 813*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System

Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 120.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 573 à 577.

Assembleur 80x86	INSD
INTEL 80386+	*INput String Double word*

Syntaxe

INSD

Description

Cette instruction permet de demander un double mot du port d'entrée/sortie et retourne le résultat dans l'adresse ES:[DI] et incrémente/décrémente le registre DI de 4 en fonction de l'état du drapeau de direction. L'adresse du port est contenue obligatoirement dans le registre DX.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    accumulateur ← Port(adresse)
  FIN SI
SINON
  accumulateur ← Port(adresse)
FIN SI
SI DF = 0 ALORS
  (E)DI ← (E)DI + 4
SINON
  (E)DI ← (E)DI - 4
FIN SI
```

Mnémonique

Instruction	Opcode	Description
INSD	6Dh	Lecture d'un double mot dans le port spécifié par le registre DX, et met le résultat dans l'emplacement mémoire spécifié par ES:(R)DI. Enfin, incrémente et décrémente le registre (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Un ou plusieurs bits de permission d'entrée/sortie sont fixé par le *TSS* pour un accès au port.
		X		Une ou plusieurs permission bit d'entrée/sortie sont fixer dans le TSS d'un accès de port.
			X	Le *CPL* est plus grand que l'IOPL et une ou plusieurs bits de permission sont fixé par le *TSS* pour un accès au port.

			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulles est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 814
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 120.

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 573 à 577.

Assembleur 80x86 — INSERTPS

SSE4.1 — *Insert Packed Single Precision Floating-Point Value*

Syntaxe

INSERTPS *dest*,*source*,*immediat*

Description

Cette instruction permet d'insérer une valeur réel de simple précision d'un opérande source dans l'emplacement d'un registre d'opérande de destination et met des 0 dans les données en dehors du masque de destination.

Algorithme

```
SI source est un registre ALORS
  COUNT_S ← immediat(7..6)
SINON
  COUNT_S ← 0
FIN SI
COUNT_D ← immediat(5..4)
ZMASK ← immediat(3..0)
EVALUER CAS COUNT_S DE
  CAS 0: TMP ← source(31..0)
  CAS 1: TMP ← source(63..32)
  CAS 2: TMP ← source(95..64)
  CAS 3: TMP ← source(127..96)
FIN EVALUER CAS
EVALUER CAS COUNT_D DE
  CAS 0:
    TMP2(31..0) ← TMP
    TMP2(127..32) ← dest(127..32)
  CAS 1:
    TMP2(63..32) ← TMP
```

```
    TMP2(31..0) ← dest(31..0)
    TMP2(127..64) ← dest(127..64)
  CAS 2:
    TMP2(95..64) ← TMP
    TMP2(63..0) ← dest(63..0)
    TMP2(127..96) ← dest(127..96)
  CAS 3:
    TMP2(127..96) ← TMP
    TMP2(95..0) ← dest(95..0)
FIN EVALUER CAS
SI ZMASK(0) = 1 ALORS
  dest(31..0) ← 00000000h
SINON
  dest(31..0) ← TMP2(31..0)
  SI ZMASK(1) = 1 ALORS
    dest(63..32) ← 00000000h
  SINON
    dest(63..32) ← TMP2(63..32)
    SI ZMASK[2] = 1 ALORS
      dest(95..64) ← 00000000h
    SINON
      dest(95..64) ← TMP2(95..64)
      SI ZMASK[3] = 1 ALORS
        dest(127..96) ← 00000000h
      SINON
        dest(127..96) ← TMP2(127..96)
      FIN SI
    FIN SI
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
INSERTPS *xmm1,xmm2/m32,imm8*	66h 0Fh 3Ah 21h /r *ib*	Cette instruction permet d'insérer une valeur réel de simple précision d'un opérande source dans l'emplacement d'un registre

		d'opérande de destination et met des 0 dans les données en dehors du masque de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 578 à 580.

Assembleur 80x86	**INSW**
INTEL 80286+	*INput String Word*

Syntaxe

```
INSW
```

Description

Cette instruction permet de demander un mot du port d'entrée/sortie et retourne le résultat dans l'adresse ES:[DI] et incrémente/décrémente le registre DI de 2 en fonction de l'état du drapeau de direction. L'adresse du port est contenu obligatoirement dans le registre DX.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    accumulateur ← Port(adresse)
  FIN SI
SINON
  accumulateur ← Port(adresse)
FIN SI
SI DF = 0 ALORS
  (E)DI ← (E)DI + 2
SINON
  (E)DI ← (E)DI - 2
FIN SI
```

Mnémonique

Instruction	Opcode	Description
INSW	6Dh	Lecture d'un mot dans le port spécifié par le registre DX, et met le résultat dans l'emplacement mémoire spécifié par ES:(R)DI. Enfin, incrémente et décrémente le registre (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Un ou plusieurs bits de permission d'entrée/sortie sont fixer par le *TSS* pour un accès au port.
		X		Une ou plusieurs permission bit d'entrée/sortie sont fixer dans le TSS d'un accès de port.
			X	Le *CPL* est plus grand que le IOPL et une ou plusieurs bits de permission sont fixer par le *TSS* pour un accès au port.

			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 813
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 120.

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 573 à 577.

Assembleur 80x86 INT

INTEL 8088+ *Interrupt*

Syntaxe

INT *numéro*

Description

Cette instruction permet d'exécuter l'interruption avec le numéro spécifié.

Algorithme

drapeaux préservé SP ← SP - 2 drapeau IF ← 0 drapeau TF ← 0 CS préservé SP ← SP - 2 IP préservé CS ← Donnée à l'adresse mémoire (*numéro* x 4 + 2) IP ← Donnée à l'adresse mémoire (*numéro* x 4)

Mnémonique

Instruction	Opcode	Description
INT 03h	CCh	Cette instruction permet d'exécuter l'interruption avec le numéro 03h.
INT *imm8*	CDh *ib*	Appel une routine de service d'interruption spécifié par le vecteur

		imm8.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#TS(Sélecteur)		X	X	Une partie de l'échangeur de pile, la destination du sélecteur de segment de pile ou le registre (R)SP dans le *TSS* est en dehors des limites du *TSS*.
		X	X	Une partie de l'échangeur de pile, la destination du sélecteur de segment de pile ou le registre (R)SP dans le *TSS* a sélecteur nulle.
		X	X	Une partie de l'échangeur de pile, la destination du sélecteur du bit *TS* est fixé, mais le sélecteur *LDT* a un sélecteur

				nulle.
		X	X	Une partie de l'échangeur de pile, la destination du sélecteur du *TSS* est en dehors des limites des tables de descripteur *GDT* ou *LDT*.
		X	X	Une partie de l'échangeur de pile, la destination du sélecteur du *TSS* contient un *RPL* n'étant pas égale au *DPL*.
		X	X	Une partie de l'échangeur de pile, la destination du sélecteur du *TSS* contient un *DPL* n'étant pas égale au *CPL* du sélecteur de code segment.
		X	X	Une partie de l'échangeur de pile, la destination du

				sélecteur du *TSS* n'est pas dans un segment écrivable.
#NP(Segment non présent)		X	X	L'accès au segment de code, le pont d'interruption, le pont de la trappe, le pont de la tâche ou le *TSS* ne sont pas présent.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Sélecteur)		X	X	Après un échange de pile, une adresse mémoire dépasse les limites du segment de pile ou est non-canonique.
		X	X	Après un échange de pile, le

				registre *SS* est chargé dans un sélecteur de segment non-nulle et le segment est marqué comme non présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
	X	X	X	La destination de déplacement dépasse la limite du segment de code ou n'est pas canonique.
		X		Le IOPL est inférieur à 3 et le CR4.VME vaut 0.
		X		Le IOPL est inférieur à 3, le CR4.VME vaut 1 et le bit correspondant à la

				redirection de la cartographie d'interruption *VME* vaut 1.
#GP(Sélecteur)	X	X	X	Le vecteur d'interruption est en dehors des limites du *IDT*.
		X	X	Le descripteur d'*IDT* n'est pas une interruption, une trappe ou une tâche de pont dans un mode «*legacy*» ou n'est pas une interruption 64 bits ou un pont de trappe pour un mode «*long*».
		X	X	Le *DPL* de l'interruption, de la trappe, du descripteur de pont de tâche est inférieur au *CPL*.
		X	X	Le sélecteur de segment

				spécifié par l'interruption ou la trappe du pont est un bit TI fixé, mais le sélecteur *LDT* a un sélecteur nulle.
		X	X	Le sélecteur de segment spécifié par l'interruption ou la trappe du pont dépasse la limite de la table de descripteur ou a un sélecteur nulle.
		X	X	Le descripteur de segment spécifié par l'interruption ou le pont de la trappe n'est pas un segment de code dans un mode «*legacy*», ou un segment de code 64 bits dans un mode «*long*».
			X	Le segment de

				DPL spécifié par l'interruption ou le pont de la trappe est supérieur au *CPL*.
		X		Le segment de *DPL* spécifié par l'interruption ou le pont ne pointe pas sur 0 ou est dans un segment conforme.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Exemples

L'exemple suivant permet d'afficher un message «Bonjour» sur la console du système d'exploitation DOS :

```
1. Message DB 'Bonjour$'
2. ; Place des instructions ici
3. MOV AH,09h
4. MOV DX,Offset Message
5. INT 21h
```

On obtiendra le résultat suivant :

Voir également

Liste des interruptions
Instruction assembleur 80x86 - Instruction INTO
Instruction assembleur 80x86 - Instruction IRET

Références

Le livre d'Or PC, *Martin Althaus*, *1992*, *ISBN: 2-7361-0934-1*, *page 814*
Assembleur Facile, *Philippe Mercier*, *1990*, *ISBN: 2-501-01176-7*, *page 405*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices*, *Revision 3.14*, *September 2007*, *Publication No. 24594*, *page 122 à 128.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, *Edition Intel*, *Mars 2010*, *Publication No. 253666-034US*, *page 581 à 595.*

Assembleur 80x86	**INTO**
INTEL 8086+	*Interrupt Overflow*

Syntaxe

```
INTO
```

Description

Cette instruction permet d'exécuter l'interruption numéro 4 si le drapeau de débordement (OF) est fixé sur 1. Parcontre, si le drapeau de débordement (OF) est égal à 0, l'exécution se poursuit tout simplement à l'instruction suivante.

Algorithme

```
SI OF = 1 ALORS
  SP ← SP - 2
  Les drapeaux sont préservé
  SP ← SP - 2
  CS est préservé
  SP ← SP - 2
  IP est préservé
  CS ← Donnée à la position mémoire 12h
  IP ← Donnée à la position mémoire 10h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
INTO	CEh	Appel une exception de débordement si le drapeau de débordement vaut 1. Invalide en

		mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#OF(Débordement)	X	X	X	Cette instruction est exécuté avec OF fixé à 1.
#UD(Opcode invalide)			X	Cette instruction est exécuté en mode 64-bits

Voir également

Instruction assembleur 80x86 - Instruction INT
Instruction assembleur 80x86 - Instruction BOUND

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 814
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 129.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 581 à 595.

Assembleur 80x86	**INVD**
INTEL 80486+	*Invalidate Internal Cache*

Syntaxe

INVD

Description

Cette instruction permet de désactiver et de vider le tampon interne du micro-processeur.

Algorithme

Vide le cache interne (*Cela signifie que toutes les lignes de caches internes sont définit comme invalide*) *Envoi un signal au cache externe à vider*

Mnémonique

Instruction	**Opcode**	**Description**
INVD	0Fh 08h	Invalide le cache interne et les déclencheurs de cache externe invalide

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#GP(Protection général)		X	X	Le CPL ne vaut pas 0.

Voir également

Instruction assembleur 80x86 - Instruction WBINVD
Instruction assembleur 80x86 - Instruction CLFLUSH

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 596 à 597.

Assembleur 80x86 INVEPT

INTEL VMX (Virtualisation)

Invalidate Translations Derived from EPT

Syntaxe

INVEPT *destination*, *source*

Description

Cette instruction permet d'invalider la cartographie dans translation des tampons (*TLB*) et des caches de pagination structuré dérivé des tables de pages étendue (*EPT*).

Algorithme

```
SI (pas dans opération VMX ) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON SI dans VMX mais pas l'opération racine ALORS
  quitte le VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON
  INVEPT_TYPE ← valeur de l'opérande de registre
  SI IA32_VMX_EPT_VPID_CAP MSR inqieue que le processus ne support pas INVEPT_TYPE
ALORS
    VMfail(Opération invalide à INVEPT/INVVPID)
  SINON
    INVEPT_DESC ← valeur d'opérande mémoire
    EPTP ← INVEPT_DESC(63..0)
    EVALUER CAS INVEPT_TYPE
      CAS 1:
        SI entrée VM avec l'active «EPT» VM contrôle d'exécution fixé à 1 en échec à cause de la
valeur EPTP ALORS
          VMfail(Opérande invalide à INVEPT/INVVPID)
        SINON
          Cartographie invalide associé avec EPTP(51..12)
```

```
            succès VM
          FIN SI
        CAS 2:
          Cartographie invalide associé avec tous les EPTPs
          succès VM
      FIN EVALUER CAS
    FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
INVEPT *r64, m128*	66h 0Fh 38h 80h	Cette instruction permet d'invalider la cartographie dans translation des tampons (*TLB*) et des caches de pagination structuré dérivé des tables de pages étendue (*EPT*).
INVEPT *r32, m128*	66h 0Fh 38h 80h	Cette instruction permet d'invalider la cartographie dans translation des tampons (*TLB*) et des caches de pagination structuré dérivé des tables de pages étendue (*EPT*).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 570 à 572.

Assembleur 80x86	**INVLPG**
INTEL 80486+	*Invalidate TLB Entry*

Syntaxe

```
INVLPG
```

Description

Cette instruction permet d'invalider les transferts du *TLB* (*Translation Lookaside Buffer*) du micro-processeur.

Mnémonique

Instruction	Opcode	Description
INVLPG *mem8*	0Fh 01h /7	Invalide l'entrée TLB pour la page contenant en emplacement mémoire spécifié.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)		X	X	Le CPL ne vaut pas 0.

Voir également

Instruction assembleur 80x86 - Instruction INVLPGA

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 598 à 599.

Assembleur 80x86 INVLPGA

x86-64+ *Invalidate TLB Entry in a Specified ASID*

Syntaxe

```
INVLPGA RAX, ECX
```

Description

Cette instruction permet d'invalider la cartographie *TLB* (*Translation Lookaside Buffer*) pour une page virtual et un *ASID* spécifié.

Mnémonique

Instruction	Opcode	Description
INVLPGA RAX, ECX	0Fh 01h DFh	Cette instruction invalide le cartographie TLB pour la page virtuel spécifié dans le registre RAX et l'ASID spécifié par le registre ECX.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Les instructions SVM ne sont pas supportés, comme indiqué par le bit 2 du registre ECX de la fonction

				8000_00001h de l'instruction *CPUID*.
			X	La machine virtuel sécurisé n'est pas activé (EFER.SVME=0).
	X	X		Cette instruction est seulement reconnu en mode protégé.
#GP(Protection général)	X	X	X	Le CPL ne vaut pas 0.

Voir également

Instruction assembleur 80x86 - Instruction INVLPG

Assembleur 80x86 INVVPID

INTEL VMX (Virtualisation)

Invalidate Translations Based on VPID

Syntaxe

INVVPID *destination*, *source*

Description

Cette instruction permet d'invalider la cartographie dans translation des tampons (*TLB*) et des caches de pagination structuré basé sur l'identificateur de processeur virtuel (*VPID*).

Algorithme

```
SI (pas dans opération VMX ) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON mais pas l'opération racine ALORS
  quitte VM
SINON CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON
  INVVPID_TYPE ← valeur de l'opérande registre
  SI IA32_VMX_EPT_VPID_CAP MSR indique que le processeur ne support pas INVVPID_TYPE
ALORS
    VMfail(Opération invalide à INVEPT/INVVPID)
  SINON
    INVVPID_DESC ← valeur de l'opérande mémoire
    SI INVVPID_DESC(63..16) ← 0 ALORS
      VMfail(Opération invalide à INVEPT/INVVPID)
    SINON
      EVALUER CAS INVVPID_TYPE
        CAS 0:
          VPID ← INVVPID_DESC(15..0)
          SI VPID = 0 ALORS
            VMfail(Opération invalide à INVEPT/INVVPID)
```

```
        SINON
          GL_ADDR ← INVVPID_DESC(127..64)
          SI GL_ADDR n'a pas une forme canonique ALORS
            VMfail(Opération invalide à INVEPT/INVVPID)
          SINON
            Cartographie invalide pour la balise GL_ADDR avec VPID
            succès VM
          FIN SI
        FIN SI
      CAS 1:
        VPID_CTX ← INVVPID_DESC(15..0)
        SI VPID = 0 ALORS
          VMfail(Opération invalide à INVEPT/INVVPID)
        SINON
          Toute la cartographie avec VPID
          succès VM
        FIN SI
      CAS 2:
        Toute la cartographie invalide avec tous les VPID non zéros
        succès VM
      CAS 3:
        VPID ← INVVPID_DESC(15..0)
        SI VPID = 0 ALORS
          VMfail(Opération invalide à INVEPT/INVVPID)
        SINON
          Toute la cartographique invalide avec une exception VPID de translation global
          succès VM
        FIN SI
    FIN EVALUER CAS
  FIN SI
 FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
INVVPID *r64*, *m128*	66h 0Fh 38h 81h	Cette instruction permet d'invalider la cartographie dans translation des

		tampons (*TLB*) et des caches de pagination structuré basé sur l'identificateur de processeur virtuel (*VPID*).
INVVPID *r32, m128*	66h 0Fh 38h 81h	Cette instruction permet d'invalider la cartographie dans translation des tampons (*TLB*) et des caches de pagination structuré basé sur l'identificateur de processeur virtuel (*VPID*).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 573 à 576.

Assembleur 80x86	**IRET**
INTEL 8086+	*Interrupt Return*

Syntaxe

```
IRET
```

Description

Cette instruction permet d'effectuer un retour précédemment provoquer par une interruption.

Algorithme

```
IP est retiré de la pile
SP ← SP + 2
CS est retiré de la pile
SP ← SP + 2
Les drapeaux sont retiré de la pile
SP ← SP + 2
```

Mnémonique

Instruction	Opcode	Description
IRET	CFh	Cette instruction permet d'effectuer un retour précédemment provoquer par une interruption.

Voir également

Instruction assembleur 80x86 - Instruction INT
Instruction assembleur 80x86 - Instruction INTO

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 814
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 405
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 129.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 600 à 610.

Assembleur 80x86 IRETD

INTEL 80386+

Interrupt Return Doubleword

Syntaxe

```
IRETD
```

Description

Cette instruction permet d'effectuer un retour 32-bits précédemment provoquer par une interruption.

Mnémonique

Instruction	Opcode	Description
IRETD	66h CFh	Cette instruction permet d'effectuer un retour 32-bits précédemment provoquer par une interruption.

Voir également

Instruction assembleur 80x86 - Instruction INT
Instruction assembleur 80x86 - Instruction INTO

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 600 à 610.

Assembleur 80x86 IRETQ

x86-64+

Interrupt Return Quadword

Syntaxe

```
IRETQ
```

Description

Cette instruction permet d'effectuer un retour 64-bits précédemment provoquer par une interruption.

Mnémonique

Instruction	Opcode	Description
IRETQ	CFh	Cette instruction permet d'effectuer un retour 64-bits précédemment provoquer par une interruption.

Voir également

Instruction assembleur 80x86 - Instruction INT
Instruction assembleur 80x86 - Instruction INTO

Références

AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 265.

Assembleur 80x86	**J??**
INTEL 8086+	*Jump if ??*

Syntaxe

J?? *destination*

Description

Cette instruction permet d'effectuer un branchement conditionnel à un emplacement mémoire spécifié.

Il est a noter que les combinaisons de lettres permettant de représenter la condition sont indiquées dans le tableau suivant:

Instruction	Condition	Description	**CPU/MPU**
JA	A	Supérieur	INTEL 8086+
JAE	AE	Supérieur ou égal	INTEL 8086+
JB	B	Inférieur	INTEL 8086+
JBE	BE	Inférieur ou égal	INTEL 8086+
JC	C	Indicateur de retenue à égal à 1	INTEL 8086+
JCXZ	CXZ	Si CX = 0	INTEL 8086+
JE	E	Zéro (Egal)	INTEL 8086+
JECXZ	ECXZ	Si ECX = 0	INTEL 80386+
JG	G	Supérieur	INTEL 8086+

JGE	GE	Supérieur ou égal	INTEL 8086+
JL	L	Inférieur	INTEL 8086+
JLE	LE	Inférieur ou égal	INTEL 8086+
JNA	NA	Pas supérieur	INTEL 8086+
JNAE	NAE	Ni supérieur ni égal	INTEL 8086+
JNB	NB	Pas inférieur	INTEL 8086+
JNC	NC	Indicateur de retenue égal à 0	INTEL 8086+
JNE	NE	Pas égal (Zéro)	INTEL 8086+
JNG	NG	Pas supérieur	INTEL 8086+
JNGE	NGE	Ni supérieur ni égal	INTEL 8086+
JNL	NL	Pas inférieur	INTEL 8086+
JNLE	NLE	Ni inférieur ni égal	INTEL 8086+
JNO	NO	Pas de débordement (indicateur de débordement égal à 0)	INTEL 8086+
JNP	NP	Pas de parité	INTEL 8086+
JNS	NS	Pas de signe	INTEL 8086+
JNZ	NZ	Pas zéro	INTEL 8086+
JO	O	Débordement (indicateur de débordement égal à 1)	INTEL 8086+
JP	P	Parité	INTEL 8086+
JPE	PE	Parité paire	INTEL 8086+

JPO	PO	Parité impaire	INTEL 8086+
JRCXZ	RCXZ	Si RCX = 0	x86-64+
JS	S	Signe	INTEL 8086+
JZ	Z	Zéro	INTEL 8086+

Algorithme

SI *condition* **ALORS**
  EIP ← EIP + SignExtend(*destination*)
  SI taille de l'opérande = 16 bits **ALORS**
    EIP ← EIP ∩ 0000FFFFh
  FIN SI
FIN SI

Mnémonique

Instruction	Opcode	Description
JO *rel8off*	70h *cb*	Saut si drapeau de débordement vaut 1 (OF = 1).
JO *rel16off*	0Fh 80h *cw*	Saut si drapeau de débordement vaut 1 (OF = 1).
JO *rel32off*	0Fh 80h *cd*	Saut si drapeau de débordement vaut 1 (OF = 1).
JNO *rel8off*	71h *cb*	Saut si drapeau de débordement vaut 0 (OF = 0).
JNO *rel16off*	0Fh 81h *cw*	Saut si drapeau de débordement vaut 0 (OF = 0).

JNO *rel32off*	0Fh 81h *cd*	Saut si drapeau de débordement vaut 0 (OF = 0).
JB *rel8off*	72h *cb*	Saut si drapeau de retenue vaut 1 (CF = 1).
JB *rel16off*	0Fh 82h *cw*	Saut si drapeau de retenue vaut 1 (CF = 1).
JB *rel32off*	0Fh 82h *cd*	Saut si drapeau de retenue vaut 1 (CF = 1).
JC *rel8off*	72h *cb*	Saut si drapeau de retenue vaut 1 (CF = 1).
JC *rel16off*	0Fh 82h *cw*	Saut si drapeau de retenue vaut 1 (CF = 1).
JC *rel32off*	0Fh 82h *cd*	Saut si drapeau de retenue vaut 1 (CF = 1).
JNAE *rel8off*	72h *cb*	Saut si drapeau de retenue vaut 1 (CF = 1).
JNAE *rel16off*	0Fh 82h *cw*	Saut si drapeau de retenue vaut 1 (CF = 1).
JNAE *rel32off*	0Fh 82h *cd*	Saut si drapeau de retenue vaut 1 (CF = 1).
JNB *rel8off*	73h *cb*	Saut si drapeau de retenue vaut 0 (CF = 0).
JNB *rel16off*	0Fh 83h *cw*	Saut si drapeau de retenue vaut 0 (CF = 0).
JNB *rel32off*	0Fh 83h *cd*	Saut si drapeau de retenue vaut 0 (CF = 0).

JNC *rel8off*	73h *cb*	Saut si drapeau de retenue vaut 0 (CF = 0).
JNC *rel16off*	0Fh 83h *cw*	Saut si drapeau de retenue vaut 0 (CF = 0).
JNC *rel32off*	0Fh 83h *cd*	Saut si drapeau de retenue vaut 0 (CF = 0).
JAE *rel8off*	73h *cb*	Saut si drapeau de retenue vaut 0 (CF = 0).
JAE *rel16off*	0Fh 83h *cw*	Saut si drapeau de retenue vaut 0 (CF = 0).
JAE *rel32off*	0Fh 83h *cd*	Saut si drapeau de retenue vaut 0 (CF = 0).
JZ *rel8off*	74h *cb*	Saut si drapeau du zéro vaut 1 (ZF = 1).
JZ *rel16off*	0Fh 84h *cw*	Saut si drapeau du zéro vaut 1 (ZF = 1).
JZ *rel32off*	0Fh 84h *cd*	Saut si drapeau du zéro vaut 1 (ZF = 1).
JE *rel8off*	74h *cb*	Saut si drapeau du zéro vaut 1 (ZF = 1).
JE *rel16off*	0Fh 84h *cw*	Saut si drapeau du zéro vaut 1 (ZF = 1).
JE *rel32off*	0Fh 84h *cd*	Saut si drapeau du zéro vaut 1 (ZF = 1).
JNZ *rel8off*	75h *cb*	Saut si drapeau du zéro vaut 0 (ZF = 0).

JNZ *rel16off*	0Fh 85h *cw*	Saut si drapeau du zéro vaut 0 (ZF = 0).
JNZ *rel32off*	0Fh 85h *cd*	Saut si drapeau du zéro vaut 0 (ZF = 0).
JNE *rel8off*	75h *cb*	Saut si drapeau du zéro vaut 0 (ZF = 0).
JNE *rel16off*	0Fh 85h *cw*	Saut si drapeau du zéro vaut 0 (ZF = 0).
JNE *rel32off*	0Fh 85h *cd*	Saut si drapeau du zéro vaut 0 (ZF = 0).
JBE *rel8off*	76h *cb*	Saut si drapeau du zéro vaut 1 ou drapeau de retenue vaut 1 (CF = 1 ou ZF = 1).
JBE *rel16off*	0Fh 86h *cw*	Saut si drapeau du zéro vaut 1 ou drapeau de retenue vaut 1 (CF = 1 ou ZF = 1).
JBE *rel32off*	0Fh 86h *cd*	Saut si drapeau du zéro vaut 1 ou drapeau de retenue vaut 1 (CF = 1 ou ZF = 1).
JNA *rel8off*	76h *cb*	Saut si drapeau du zéro vaut 1 ou drapeau de retenue vaut 1 (CF = 1 ou ZF = 1).
JNA *rel16off*	0Fh 86h *cw*	Saut si drapeau du zéro vaut 1 ou drapeau de retenue vaut 1 (CF = 1 ou ZF = 1).
JNA *rel32off*	0Fh 86h *cd*	Saut si drapeau du zéro vaut 1 ou drapeau de retenue vaut 1 (CF = 1 ou ZF = 1).

JNBE *rel8off*	77h *cb*	Saut si drapeau du zéro vaut 0 et drapeau de retenue vaut 0 (CF = 0 et ZF = 0).
JNBE *rel16off*	0Fh 87h *cw*	Saut si drapeau du zéro vaut 0 et drapeau de retenue vaut 0 (CF = 0 et ZF = 0).
JNBE *rel32off*	0Fh 87h *cd*	Saut si drapeau du zéro vaut 0 et drapeau de retenue vaut 0 (CF = 0 et ZF = 0).
JA *rel8off*	77h *cb*	Saut si drapeau du zéro vaut 0 et drapeau de retenue vaut 0 (CF = 0 et ZF = 0).
JA *rel16off*	0Fh 87h *cw*	Saut si drapeau du zéro vaut 0 et drapeau de retenue vaut 0 (CF = 0 et ZF = 0).
JA *rel32off*	0Fh 87h *cd*	Saut si drapeau du zéro vaut 0 et drapeau de retenue vaut 0 (CF = 0 et ZF = 0).
JS *rel8off*	78h *cb*	Saut si drapeau de signe vaut 1 (SF = 1).
JS *rel16off*	0Fh 88h *cw*	Saut si drapeau de signe vaut 1 (SF = 1).
JS *rel32off*	0Fh 88h *cd*	Saut si drapeau de signe vaut 1 (SF = 1).
JNS *rel8off*	79h *cb*	Saut si drapeau de signe vaut 0 (SF = 0).
JNS *rel16off*	0Fh 89h *cw*	Saut si drapeau de signe vaut 0 (SF = 0).

JNS *rel32off*	0Fh 89h *cd*	Saut si drapeau de signe vaut 0 (SF = 0).
JP *rel8off*	7Ah *cb*	Saut si drapeau de parité vaut 1 (PF = 1).
JP *rel16off*	0Fh 8Ah *cw*	Saut si drapeau de parité vaut 1 (PF = 1).
JP *rel32off*	0Fh 8Ah *cd*	Saut si drapeau de parité vaut 1 (PF = 1).
JPE *rel8off*	7Ah *cb*	Saut si drapeau de parité vaut 1 (PF = 1).
JPE *rel16off*	0Fh 8Ah *cw*	Saut si drapeau de parité vaut 1 (PF = 1).
JPE *rel32off*	0Fh 8Ah *cd*	Saut si drapeau de parité vaut 1 (PF = 1).
JNP *rel8off*	7Bh *cb*	Saut si drapeau de parité vaut 0 (PF = 0).
JNP *rel16off*	0Fh 8Bh *cw*	Saut si drapeau de parité vaut 0 (PF = 0).
JNP *rel32off*	0Fh 8Bh *cd*	Saut si drapeau de parité vaut 0 (PF = 0).
JPO *rel8off*	7Bh *cb*	Saut si drapeau de parité vaut 0 (PF = 0).
JPO *rel16off*	0Fh 8Bh *cw*	Saut si drapeau de parité vaut 0 (PF = 0).
JPO *rel32off*	0Fh 8Bh *cd*	Saut si drapeau de parité vaut 0 (PF = 0).

JL *rel8off*	7Ch *cb*	Saut si SF <> OF.
JL *rel16off*	0Fh 8Ch *cw*	Saut si SF <> OF.
JL *rel32off*	0Fh 8Ch *cd*	Saut si SF <> OF.
JNGE *rel8off*	7Ch *cb*	Saut si SF <> OF.
JNGE *rel16off*	0Fh 8Ch *cw*	Saut si SF <> OF.
JNGE *rel32off*	0Fh 8Ch *cd*	Saut si SF <> OF.
JNL *rel8off*	7Dh *cb*	Saut si SF = OF.
JNL *rel16off*	0Fh 8Dh *cw*	Saut si SF = OF.
JNL *rel32off*	0Fh 8Dh *cd*	Saut si SF = OF.
JGE *rel8off*	7Dh *cb*	Saut si SF = OF.
JGE *rel16off*	0Fh 8Dh *cw*	Saut si SF = OF.
JGE *rel32off*	0Fh 8Dh *cd*	Saut si SF = OF.
JLE *rel8off*	7Eh *cb*	Saut si ZF = 1 ou SF <> OF.
JLE *rel16off*	0Fh 8Eh *cw*	Saut si ZF = 1 ou SF <> OF.
JLE *rel32off*	0Fh 8Eh *cd*	Saut si ZF = 1 ou SF <> OF.
JNG *rel8off*	7Eh *cb*	Saut si ZF = 1 ou SF <> OF.
JNG *rel16off*	0Fh 8Eh *cw*	Saut si ZF = 1 ou SF <> OF.
JNG *rel32off*	0Fh 8Eh *cd*	Saut si ZF = 1 ou SF <> OF.
JNLE *rel8off*	7Fh *cb*	Saut si ZF = 0 et SF = OF.

JNLE *rel16off*	0Fh 8Fh *cw*	Saut si ZF = 0 et SF = OF.
JNLE *rel32off*	0Fh 8Fh *cd*	Saut si ZF = 0 et SF = OF.
JG *rel8off*	7Fh *cb*	Saut si ZF = 0 et SF = OF.
JG *rel16off*	0Fh 8Fh *cw*	Saut si ZF = 0 et SF = OF.
JG *rel32off*	0Fh 8Fh *cd*	Saut si ZF = 0 et SF = OF.
JCXZ *rel8off*	E3h *cb*	Saut si le registre CX vaut zéro.
JECXZ *rel8off*	E3h *cb*	Saut si le registre ECX vaut zéro.
JRCXZ *rel8off*	E3h *cb*	Saut si le registre RCX vaut zéro.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique

Voir également

Instruction assembleur 80x86 - Instruction JMP

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 815
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 405 à 409
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 130 à 134.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 611 à 618.

Assembleur 80x86	**JMP**
INTEL 8086+	*Jump*

Syntaxe

JMP *destination*

Description

Cette instruction permet d'effectuer un branchement à un emplacement mémoire spécifié.

Algorithme

```
SI near jump ALORS
  SI saut relatif court ALORS
    tempEIP ← EIP + destination
  SINON
    tempEIP ← destination
  FIN SI
  SI tempEIP est en dehors des limites du code segment ALORS
    EXCEPTION #GP(0)
  FIN SI
  SI taille de l'opérande = 32 bits ALORS
    EIP ← tempEIP
  SINON
    EIP ← tempEIP ∩ 0000FFFFh
  FIN SI
FIN SI
SI saut long ET (PE = 0 OU (PE = 1 ET VM = 1)) ALORS * Adresse réel ou mode virtuel 8086
  tempEIP ← destination(offset)
  SI tempEIP est en dehors des limites du code de segment ALORS
    EXCEPTION #GP(0)
  FIN SI
  CS ← destination(Sélecteur de segment)
  SI taille de l'opérande = 32 bits ALORS
    EIP ← tempEIP
```

```
  SINON
    EIP ← tempEIP ∩ 0000FFFFh
  FIN SI
FIN SI
SI far jump ET (PE = 1 ET VM = 0) ALORS * Mode protégé, pas mode virtuel 8086
  SI l'adresse effective dans les segments CS, DS, ES, FS, GS ou SS est illégale OU sélecteur de
segment dans une destination d'opérande nulle ALORS
    EXCEPTION #GP(0)
  FIN SI
  SI index de sélecteur de segment n'est pas dans la limite de descripteur de table ALORS
    EXCEPTION #GP(nouveau sélecteur)
  FIN SI
  Lecture du type et de l'accès correct du descripteur de segment
  SI type de segment n'est pas conforme OU code de segment non conforme, appel le pont,
tâche du pont, ou TSS ALORS
    EXCEPTION #GP(sélecteur de segment)
  FIN SI
  Type de dépendance et accès correct
  ALLER A CONFORMING-CODE-SEGMENT
  ALLER A NONCONFORMING-CODE-SEGMENT
  ALLER A CALL-GATE
  ALLER A TASK-GATE
  ALLER A TASK-STATE-SEGMENT
SINON
  EXCEPTION #GP(Sélecteur de segment)
FIN SI

CONFORMING-CODE-SEGMENT:
  SI DPL > CPL ALORS
    EXCEPTION #GP(Sélecteur de segment)
  FIN SI
  SI segment pas présent ALORS
    EXCEPTION #NP(Sélecteur de segment)
  FIN SI
    tempEIP ← destination(Offset)
  SI Taille de l'opérande = 16 bits ALORS
    tempEIP ← tempEIP ∩ 0000FFFFh
  FIN SI
  SI tempEIP n'est pas dans la limite du code segment ALORS
    EXCEPTION #GP(0)
  FIN SI
```

```
  CS ← destination(Sélecteur de segment)
  CS(RPL) ← CPL
  EIP ← tempEIP
  FIN

NONCONFORMING-CODE-SEGMENT:
  SI (RPL > CPL) OU (DPL = CPL) ALORS
    EXCEPTION #GP(Sélecteur de code segment)
  FIN SI
  SI segment n'est pas présent ALORS
    EXCEPTION #NP(Sélecteur de segment)
  FIN SI
  SI pointeur d'instruction est en dehors des limites du segment ALORS
    EXCEPTION #GP(0)
  FIN SI
  tempEIP ← destination(offset)
  SI taille de l'opérande = 16 bits ALORS
    tempEIP ← tempEIP ∩ 0000FFFFh
  FIN SI
  SI tempEIP n'est pas dans les limites du code segment ALORS
    EXCEPTION #GP(0)
  FIN SI
  CS ← destination(Sélecteur de segment)
  CS(RPL) ← CPL
  EIP ← tempEIP
  FIN

CALL-GATE:
  SI appel de pont DPL < CPL OU appel de pont DPL < appel de pont de sélecteur de segment RPL
ALORS
    EXCEPTION #GP(sélecteur d'appel de pont)
  FIN SI
  SI appel d'un pont non présent ALORS
    EXCEPTION #NP(Sélecteur d'appel de pont)
  FIN SI
  SI appel d'un pont du sélecteur de code segment est nulle ALORS
    EXCEPTION #GP(0)
  FIN SI
  SI appel d'un pont d'index de sélecteur de code de segment est en dehors des limites du
descripteur de table ALORS
    EXCEPTION #GP(sélecteur de code segment)
```

FIN SI
Lecture du descripteur de code segment
SI descripteur de code segment n'est pas indiquer dans le code segment **OU** descripteur de code segment est conforme et DPL > CPL **OU** decripteur de segment de code n'est pas conforme **ET** DPL = CPL **ALORS**
EXCEPTION #GP(Sélecteur de code segment)
FIN SI
SI code segment n'est pas présent **ALORS**
EXCEPTION #NP(sélecteur de code segment)
FIN SI
SI pointeur d'instruction n'est pas dans les limites du code segment **ALORS**
EXCEPTION #GP(0)
FIN SI
tempEIP ← *destination*(offset)
SI Taille du pont = 16 bits **ALORS**
tempEIP ← tempEIP ∩ 0000FFFFh
FIN SI
SI tempEIP n'est pas dans les limites du code segment **ALORS**
EXCEPTION #GP(0)
FIN SI
CS ← *destination*(Sélecteur de segment)
CS(RPL) ← CPL
EIP ← tempEIP
FIN

TASK-GATE:
SI tâche du pont DPL < CPL **OU** tâche du pont DPL < sélecteur de segment du pont de tâche RPL **ALORS**
EXCEPTION #GP(Sélecteur de tâche du pont)
FIN SI
SI pont de tâche n'est pas présent **ALORS**
EXCEPTION #NP(Sélecteur de pont)
FIN SI
Lecteur du sélecteur de segment TSS dans le descripteur de pont de tâche
SI bit de sélecteur de segment TSS local/global est fixer à local **OU** index n'est dans les limites du GDT **OU** que le descripteur TSS specifié est occupé **ALORS**
EXCEPTION #GP(Sélecteur TSS)
FIN SI
SI TSS n'est pas présent **ALORS**
EXCEPTION #NP(Sélecteur TSS)
FIN SI

```
  SWITCH-TASKS à TSS
  SI EIP n'est dans les limites de code de segment ALORS
    EXCEPTION #GP(0)
  FIN SI
  FIN

TASK-STATE-SEGMENT:
  SI TSS DPL < CPL OU TSS DPL < sélecteur de segment TSS RPL OU descripteur TSS indique que le TSS n'est pas disponible ALORS
    EXCEPTION #GP(Sélecteur TSS)
  FIN SI
  SI TSS n'est pas présent ALORS
    EXCEPTION #NP(Sélecteur TSS)
  FIN SI
  SWITCH-TASKS à TSS
  SI EIP n'est pas dans les limites de code segment ALORS
    EXCEPTION #GP(0)
  FIN SI
  FIN
```

Mnémonique

Instruction	Opcode	Description
JMP *rel8off*	EBh *cb*	Saut court à la destination spécifié par un déplacement signé de 8 bits
JMP *rel16off*	E9h *cw*	Saut court à la destination spécifié par un déplacement signé de 16 bits
JMP *rel32off*	E9h *cd*	Saut court à la destination spécifié par un déplacement signé de 32 bits
JMP *reg/mem16*	FFh /4	Saut court à la destination spécifié par un le registre/mémoire 16 bits
JMP *reg/mem32*	FFh /4	Saut court à la destination spécifié par un le registre/mémoire 32 bits

JMP *reg/mem64*	FFh /4	Saut court à la destination spécifié par un le registre/mémoire 64 bits
JMP FAR *pntr16:16*	EAh *cd*	Saut long direct avec la destination spécifié par un long pointeur contenu dans l'instruction. Cette instruction est invalide en mode 64 bits.
JMP FAR *pntr16:32*	EAh *cp*	Saut long direct avec la destination spécifié par un long pointeur contenu dans l'instruction. Cette instruction est invalide en mode 64 bits.
JMP FAR *mem16:16*	FFh /5	Saut long indirect avec la destination spécifié par un long pointeur contenu en mémoire. Cette instruction est invalide en mode 64 bits.
JMP FAR *mem16:32*	FFh /5	Saut long indirect avec la destination spécifié par un long pointeur contenu en mémoire. Cette instruction est invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Cette instruction est exécuté en mode 64-bits avec un code un opcode indirect JMP (FFh /5) dans un opérande de registre.

			X	Cette instruction est exécuté en mode 64-bits avec un code un opcode direct JMP (EAh)
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#NP(Sélecteur)			X	L'accès au segment de code, au pont d'appel, un pont de tâche, ou a un TSS non présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable

			X	Un segment de données nulle est utilisé comme référence mémoire
#GP(Sélecteur)			X	La destination de sélecteur de segment de code est un sélecteur nulle.
			X	Un code, un pont d'appel, une tâche de pont, ou un descripteur *TSS* dépasse la limite de descripteur de table.
			X	Un sélecteur de segment de bit TI est fixé, mais le sélecteur *LDT* est un sélecteur nulle.
			X	Le descripteur de segment spécifié par l'instruction n'est pas un segment de code, une

				tâche de pont, un pont d'appel, ou un *TSS* disponible dans le mode «*legacy*», ou n'est pas un segment de code 64 bits ou un appel 64 bits long d'un pont dans le mode «*long*».
			X	Le *RPL* d'un sélecteur de segment de code non-conforme spécifié par l'instruction est supérieur au *CPL*, ou le *DPL* n'est pas égale au *CPL*.
			X	Le *DPL* du descripteur de segment de code spécifié est conforme quand l'instruction est supérieur au *CPL*.
			X	Le *DPL* est un pont d'appel, une tâche de

				pont, un descripteur *TSS* spécifié quand l'instruction est inférieur au *CPL* ou au *RPL*.
			X	Le sélecteur de segment spécifié par le pont d'appel ou le pont de tâche est un sélecteur nulle.
			X	Le descripteur de segment spécifié par l'appel de pont n'est pas un segment de code en mode «*legacy*» ou segment de code 64 bits dans un mode «*long*».
			X	Le *DPL* du descripteur de segment spécifié par le pont d'appel est supérieur au *CPL* et dans un segment

				conforme.
			X	Le *DPL* du descripteur de segment spécifié par le pont d'appel n'est pas égale au *CPL* et il n'est pas dans un segment conforme.
			X	L'attribut étendue du pont d'appel 64 bits n'est pas à 0.
			X	Le descripteur *TSS* n'est pas un *LDT*.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction *Jump if* (JA, JB, JC, JCX, JE,...)

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 816*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 135.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 619 à 629.

Assembleur 80x86	**LAHF**
INTEL 8088+	*Load AH in Flags*

Syntaxe

```
LAHF
```

Description

Cette instruction permet de transférer les bits d'indicateurs du registre d'état vers le registre *AH*.

Algorithme

```
AH ← Registre de drapeaux(7..0)
```

Mnémonique

Instruction	Opcode	Description
LAHF	9Fh	Charge les drapeaux SF, ZF, AF, PF et CF dans le registre AH

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction n'est pas supporté en mode 64-

				bits comme indiquer par le bit 0 du registre ECX par la fonction 8000_0001h de l'instruction CPUID.

Voir également

Instruction assembleur 80x86 - Instruction SAHF

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 816
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 409
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 142.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 630 à 631.

Intel 80286+ (Mode Protégé, Mode Réel Niveau 0)

Syntaxe

LAR *dest*,*source*

Description

Cette instruction permet de charger le registre des indicateurs d'un descripteur.

Algorithme

SI *source*(Offset) > limite du descripteur de table **ALORS**
  ZF ← 0
FIN SI
Lecteur du descripteur de segment
SI type de Descripteur de segment n'est pas conforme au code segment **ET** (CPL > DPL) **OU** (RPL > DPL) **OU** type de segment n'est pas valide pour l'instruction **ALORS**
  ZF ← 0
SINON
  SI la taille de l'opérande = 32 bits **ALORS**
    dest(*source*) ⋂ 00FxFF00h
  SINON
    dest(*source*) ⋂ FF00h
  FIN SI
FIN SI

Mnémonique

Instruction	Opcode	Description
LAR *r16,r/m16*	0Fh 02h /r	Cette instruction permet de charger le registre des indicateurs d'un

		descripteur avec un masque de FF00h.
LAR *r32,r/m32*	0Fh 02h /r	Cette instruction permet de charger le registre des indicateurs d'un descripteur avec un masque de 00FxFF00h.

Voir également

Instruction assembleur 80x86 - Instruction ARPL
Instruction assembleur 80x86 - Instruction LSL
Instruction assembleur 80x86 - Instruction VERR
Instruction assembleur 80x86 - Instruction VERW

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 632 à 634.

Assembleur 80x86 LDDQU

INTEL Pentium 4 (SSE3)+ *Load Unaligned Integer 128 bits*

Syntaxe

LDDQU *dest*, *source*

Description

Cette instruction permet de copier un quadruple mot d'une opérande source vers une opérande destination. Les opérandes doivent être des registres *XMM*.

Algorithme

dest(0..127) ← *dest*

Mnémonique

Instruction	Opcode	Description
LDDQU *xmm*, *mem*	F2h 0Fh F0h /r	Cette instruction permet de copier un quadruple mot d'une opérande source vers une opérande destination. Les opérandes doivent être des registres *XMM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 635 à 638.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

LDMXCSR

Load Streaming SIMD Extension Control/Status

Syntaxe

LDMXCSR *source*

Description

Cette instruction permet d'effectuer le chargement du mot de contrôle et d'état du flux d'extension SIMD d'une opérande mémoire 32 bits.

Algorithme

MXCSR ← *source*

Mnémonique

Instruction	Opcode	Description
LDMXCSR *m32*	0Fh AEh /2	Cette instruction permet d'effectuer le chargement du mot de contrôle et d'état du flux d'extension SIMD d'une opérande mémoire 32 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 639 à 641.

Assembleur 80x86	**LDS**
INTEL 8088+	*Load in DS*

Syntaxe

LDS *opérande_cible*,*opérande_source*

Description

Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment DS et dans un registre d'offset spécifié.

Algorithme

Registre d'offset ← Source
Registre de segment *DS* ← Source + 2

Mnémonique

Instruction	**Opcode**	**Description**
LDS *reg16*, *mem16:16*	C5h /r	Charge le DS:*reg16* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.
LDS *reg32*, *mem16:32*	C5h /r	Charge le DS:*reg32* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	L'opérande source est un registre.
			X	Cette instruction est exécuté en mode 64-bits.
#NP(Sélecteur)			X	Le registre DS, ES, FS ou GS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non présent.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non

				présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite du descripteur de la table.
	X	X	X	Un registre de segment est chargé et le bit *TI* de sélecteur de segment est fixé, mais le sélecteur *LDT* est un sélecteur nulle.

	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment nulle dans un mode 64-bits ou avec *CPL* = 3.
	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment *RPL* et le descripteur de segment *DPL* n'est pas égale au *CPL*.
	X	X	X	Le registre *SS* est chargé et le pointeur de segment n'est pas dans un segment de données écrivable.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code non-conforme, mais le *RPL* ou

				CPL est supérieur au *DPL*.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code en lecture.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 817
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 409
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 143.

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 642 à 646.

Assembleur 80x86	**LEA**
INTEL 8088+	*Load Effective Address*

Syntaxe

LEA *opérande_cible,opérande_source*

Description

Cette instruction permet d'incrémenter un registre ou un emplacement mémoire.

Algorithme

Destination ← Adresse (source)

Mnémonique

Instruction	**Opcode**	**Description**
LEA *reg16*, *mem*	8Dh /r	Entrepose l'adresse effective dans un registre 16 bits.
LEA *reg32*, *mem*	8Dh /r	Entrepose l'adresse effective dans un registre 32 bits.
LEA *reg64*, *mem*	8Dh /r	Entrepose l'adresse effective dans un registre 64 bits.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**

#UD(Opcode invalide)	X	X	X	L'opérande source est un registre.

Voir également

Instruction assembleur 80x86 - Instruction MOV

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 817
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 409
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 145.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 647 à 651.

Assembleur 80x86	LEAVE
INTEL 80286+	

Syntaxe

```
LEAVE
```

Description

Cette instruction permet de libérer une zone de mémoire attribué par l'instruction «*ENTER*» lorsqu'on utilise des procédures dans des langages de programmation de haut niveau.

Algorithme

```
SI taille de l'adresse de pile = 32 bits ALORS
  ESP ← EBP
SINON
  SP ← BP
FIN SI
SI taille l'opérande = 32 bits ALORS
  EBP ← POP()
SINON
  BP ← POP()
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LEAVE	C9h	Fixe le pointeur de pile du registre *SP* à la valeur du registre *BP* et «*POP BP*».

LEAVE	C9h	Fixe le pointeur de pile du registre *ESP* à la valeur du registre *EBP* et «*POP EBP*». Pas de préfixe pour le mode 64 bits.
LEAVE	C9h	Fixe le pointeur de pile du registre *RSP* à la valeur du registre *RBP* et «*POP RBP*».

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction ENTER

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 817
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 147.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 651 à 652.

Assembleur 80x86 LES

INTEL 8088+

Load in ES

Syntaxe

LES *opérande_cible,opérande_source*

Description

Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment ES et dans un registre d'offset spécifié.

Algorithme

Registre d'offset ← Source
Registre de segment *ES* ← Source + 2

Mnémonique

Instruction	Opcode	Description
LES *reg16*, *mem16:16*	C4h /r	Charge le ES:*reg16* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.
LES *reg32*, *mem16:32*	C4h /r	Charge le ES:*reg32* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	L'opérande source est un registre.
			X	Cette instruction est exécuté en mode 64-bits.
#NP(Sélecteur)			X	Le registre DS, ES, FS ou GS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non présent.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non

				présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite du descripteur de la table.
	X	X	X	Un registre de segment est chargé et le bit *TI* de sélecteur de segment est fixé, mais le sélecteur *LDT* est un sélecteur nulle.

	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment nulle dans un mode 64-bits ou avec *CPL* = 3.
	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment *RPL* et le descripteur de segment *DPL* n'est pas égale au *CPL*.
	X	X	X	Le registre *SS* est chargé et le pointeur de segment n'est pas dans un segment de données écrivable.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code non-conforme, mais le *RPL* ou

				CPL est supérieur au *DPL*.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code en lecture.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Exemple

Cette exemple permet de retourner dans la paire de registres DX:AX, l'heure de l'horloge étant incrémenté à tous les 18,2 par seconde. Pour se faire il utilise une méthode d'accès directe à la mémoire et prend soit de ne pas être interrompu par un IRQ. L'astuce de l'utilisation de cette instruction consiste au fait qu'il utilise une seule instruction pour un chargé double mot de 32 bits (l'instruction *LES*) plutôt que deux instructions *MOV* beaucoup plus longue en cycle d'horloge et en consommation d'octets :

```
1. PUSH 0
2. POP ES
3. XOR AX,AX
4. MOV ES,AX
5. CLI
6. LES AX,ES:[$46C]
7. STI
8. MOV DX,ES
```

Cette routine est utilisé dans le coeur de la bibliothèque «*RLL*» de l'application «*MonsterBook*» des «*Chevaliers de Malte/Développeur*» par soucis de performance lorsqu'on tant de faire jouer des «*.MOD*» en arrière plan en même temps que le «*MonsterBook*» fonctionne.

Voir également

Instruction assembleur 80x86 - Instruction LDS
Instruction assembleur 80x86 - Instruction LFS
Instruction assembleur 80x86 - Instruction LGS
Instruction assembleur 80x86 - Instruction LSS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 817
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 410
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 143.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 642 à 646.

Assembleur 80x86 LFENCE

INTEL Pentium 4 (SSE2)+ *Load Fence*

Syntaxe

LFENCE

Description

Cette instruction permet d'agir comme une barrière pour forcer une priorité en mémoire (sérialisation) entre les instructions précédant le *LFENCE* et les instructions de chargement suivant le *LFENCE*.

Mnémonique

Instruction	Opcode	Description
LFENCE	0Fh AEh E8h	Force l'ordre de sérialisation de chargement des opérations

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction n'est pas supporté et est indiqué par le bit 26 du registre EDX de la fonction

				0000_0001h de l'instruction CPUID.

Voir également

Instruction assembleur 80x86 - Instruction MFENCE
Instruction assembleur 80x86 - Instruction SFENCE

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 653 à 653.

Assembleur 80x86	LFS
INTEL 80386+	*Load in FS*

Syntaxe

LFS *opérande_cible*,*opérande_source*

Description

Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment FS et dans un registre d'offset spécifié.

Algorithme

Registre d'offset ← Source
Registre de segment *FS* ← Source + 2

Mnémonique

Instruction	Opcode	Description
LFS *reg16*, *mem16:16*	0Fh B4h /r	Charge le FS:*reg16* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.
LFS *reg32*, *mem16:32*	0Fh B4h /r	Charge le FS:*reg32* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	L'opérande source est un registre.
			X	Cette instruction est exécuté en mode 64-bits.
#NP(Sélecteur)			X	Le registre DS, ES, FS ou GS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non présent.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non

				présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite du descripteur de la table.
	X	X	X	Un registre de segment est chargé et le bit *TI* de sélecteur de segment est fixé, mais le sélecteur *LDT* est un sélecteur nulle.

	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment nulle dans un mode 64-bits ou avec *CPL* = 3.
	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment *RPL* et le descripteur de segment *DPL* n'est pas égale au *CPL*.
	X	X	X	Le registre *SS* est chargé et le pointeur de segment n'est pas dans un segment de données écrivable.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code non-conforme, mais le *RPL* ou

				CPL est supérieur au *DPL*.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code en lecture.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 642 à 646.

Assembleur 80x86	**LGDT**
INTEL 80286+, Mode protégé, mode réel niveau 0	*Load Global Descriptor Tables*

Syntaxe

LGDT *mémoire*

Description

Cette instruction permet de charger un descripteur de tables globale.

Algorithme

```
SI taille de l'opérande = 16 bits ALORS
  GDTR(Limit) mémoire(0..15)
  GDTR(Base) mémoire(16..47) ∩ 00FFFFFFh
SINON
  GDTR(Limit) mémoire(0..15)
  GDTR(Base) mémoire(16..47)
FIN SI
```

Mnémonique

Instruction	**Opcode**	**Description**
LGDT *m16/32*	0Fh 01h /2	Cette instruction permet de charger un descripteur de tables globale.

Voir également

Instruction assembleur 80x86 - Instruction LLDT
Instruction assembleur 80x86 - Instruction LIDT
Instruction assembleur 80x86 - Instruction LTR

Instruction assembleur 80x86 - Instruction SGDT
Instruction assembleur 80x86 - Instruction SIDT
Instruction assembleur 80x86 - Instruction SLDT
Instruction assembleur 80x86 - Instruction STR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 654 à 656.

Assembleur 80x86	LGS
INTEL 80386+	*Load in GS*

Syntaxe

LGS *opérande_cible,opérande_source*

Description

Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment FS et dans un registre d'offset spécifié.

Algorithme

Registre d'offset ← Source
Registre de segment *GS* ← Source + 2

Mnémonique

Instruction	Opcode	Description
LGS *reg16*, *mem16:16*	0Fh B5h /r	Charge le GS:*reg16* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.
LGS *reg32*, *mem16:32*	0Fh B5h /r	Charge le GS:*reg32* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	L'opérande source est un registre.
			X	Cette instruction est exécuté en mode 64-bits.
#NP(Sélecteur)			X	Le registre DS, ES, FS ou GS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non présent.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non

				présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite du descripteur de la table.
	X	X	X	Un registre de segment est chargé et le bit *TI* de sélecteur de segment est fixé, mais le sélecteur *LDT* est un sélecteur nulle.

	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment nulle dans un mode 64-bits ou avec *CPL* = 3.
	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment *RPL* et le descripteur de segment *DPL* n'est pas égale au *CPL*.
	X	X	X	Le registre *SS* est chargé et le pointeur de segment n'est pas dans un segment de données écrivable.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code non-conforme, mais le *RPL* ou

				CPL est supérieur au *DPL*.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code en lecture.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 642 à 646.

Assembleur 80x86	**LIDT**
INTEL 286+, Mode protégé, mode réel niveau 0	*Load Interrupt Descriptor Tables*

Syntaxe

LIDT *mémoire*

Description

Cette instruction permet de charger un descripteur de tables d'interruption.

Algorithme

SI taille de l'opérande = 16 bits **ALORS**
  IDTR(Limit) ← *mémoire*(0..15)
  IDTR(Base) ← *mémoire*(16..47) ∩ 00FFFFFFh
SINON
  IDTR(Limit) ← *mémoire*(0..15)
  IDTR(Base) ← *mémoire*(16..47)
FIN SI

Mnémonique

Instruction	Opcode	Description
LIDT *mem*	0Fh 01h /3	Cette instruction permet de charger un descripteur de tables d'interruption.

Voir également

Instruction assembleur 80x86 - Instruction LGDT
Instruction assembleur 80x86 - Instruction LLDT

Instruction assembleur 80x86 - Instruction LTR
Instruction assembleur 80x86 - Instruction SGDT
Instruction assembleur 80x86 - Instruction SIDT
Instruction assembleur 80x86 - Instruction SLDT
Instruction assembleur 80x86 - Instruction STR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 654 à 656.

Assembleur 80x86	**LLDT**
INTEL 286+, Mode protégé, mode réel niveau 0	*Load Local Descriptor Tables*

Syntaxe

LLDT *mémoire*

Description

Cette instruction permet de charger un descripteur de tables local.

Algorithme

SI *mémoire*(Offset) > limite de descripteur de table **ALORS**
  EXCEPTION #GP(Sélecteur de segment)
FIN SI
Lecture du descripteur de segment
SI type de segment de descriptor n'est pas égale à LDT **ALORS**
  EXCEPTION #GP(Sélecteur de segment)
FIN SI
SI descripteur de segment n'est pas présent **ALORS**
  EXCEPTION #NP(Sélecteur de segment)
FIN SI
LDTR(Sélecteur de segment) ← *mémoire*
LDTR(Descripteur de segment) ← Descripteur de segment GDT

Mnémonique

Instruction	**Opcode**	**Description**
LLDT *r/m16*	0Fh 00h /2	Cette instruction permet de charger un descripteur de tables local.

Voir également

Instruction assembleur 80x86 - Instruction LGDT
Instruction assembleur 80x86 - Instruction LIDT
Instruction assembleur 80x86 - Instruction LTR
Instruction assembleur 80x86 - Instruction SGDT
Instruction assembleur 80x86 - Instruction SIDT
Instruction assembleur 80x86 - Instruction SLDT
Instruction assembleur 80x86 - Instruction STR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 657 à 659.

Assembleur 80x86	**LMSW**
INTEL 286+, Mode protégé, mode réel niveau 0	*Load/Store Machine Status Word*

Syntaxe

LMSW *opérande*

Description

Cette instruction permet de copier 4 des bits d'une opérande vers les 4 bits de registre de contrôle *CR0*.

Algorithme

CR0(0..3) ← *opérande*(0..3)

Mnémonique

Instruction	**Opcode**	**Description**
LMSW *r/m16*	0Fh 01h /6	Cette instruction permet de copier 4 des bits d'une opérande vers les 4 bits de registre de contrôle *CR0*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 660 à 661.

Assembleur 80x86

INTEL 80286 ou 80386 seulement

LOADALL

Load All

Syntaxe

LOADALL

Description

Cette instruction permet d'effectuer le chargement de tous les registres de descripteur de cache.

Algorithme

SI INTEL 80286 **ALORS**
  (00800h)(6) ← 0
  (00806h)(2) ← MSW
  (00808h)(14) ← 0
  (00816h)(2) ← TR
  (00818h)(2) ← Registre de drapeaux 16 bits
  (0081Ah)(2) ← IP
  (0081Ch)(2) ← LDTR
  (0081Eh)(2) ← DS
  (00820h)(2) ← SS
  (00822h)(2) ← CS
  (00824h)(2) ← ES
  (00826h)(2) ← DI
  (00828h)(2) ← SI
  (0082Ah)(2) ← BP
  (0082Ch)(2) ← SP
  (0082Eh)(2) ← BX
  (00830h)(2) ← DX
  (00832h)(2) ← CX
  (00834h)(2) ← AX
  (00836h)(6) ← Descripteur de segment ES
  (0083Ch)(6) ← Descripteur de segment CS
  (00842h)(6) ← Descripteur de segment SS
  (00848h)(6) ← Descripteur de segment DS
  (0084Eh)(6) ← GDT

(00854h)(6) ← LDT
(0085Ah)(6) ← IDT
(00860h)(6) ← TSS
c**FIN SI**
SI INTEL 80386 **ALORS**
(ES:EDI+0000h)(4) ← CR0
(ES:EDI+0004h)(4) ← EFLAGS
(ES:EDI+0008h)(4) ← EIP
(ES:EDI+000Ch)(4) ← EDI
(ES:EDI+0010h)(4) ← ESI
(ES:EDI+0014h)(4) ← EBP
(ES:EDI+0018h)(4) ← ESP
(ES:EDI+001Ch)(4) ← EBX
(ES:EDI+0020h)(4) ← EDX
(ES:EDI+0024h)(4) ← ESX
(ES:EDI+0028h)(4) ← EAX
(ES:EDI+002Ch)(4) ← DR6
(ES:EDI+0030h)(4) ← DR7
(ES:EDI+0034h)(4) ← TR (16 bits complété par des 0)
(ES:EDI+0038h)(4) ← LDT
(ES:EDI+003Ch)(4) ← GS (16 bits complété par des 0)
(ES:EDI+0040h)(4) ← FS (16 bits complété par des 0)
(ES:EDI+0044h)(4) ← DS (16 bits complété par des 0)
(ES:EDI+0048h)(4) ← SS (16 bits complété par des 0)
(ES:EDI+004Ch)(4) ← CS (16 bits complété par des 0)
(ES:EDI+0050h)(4) ← ES (16 bits complété par des 0)
(ES:EDI+0054h)(4) ← Attribut de TSS
(ES:EDI+0058h)(4) ← Base de TSS
(ES:EDI+005Ch)(4) ← Limite de TSS
(ES:EDI+0060h)(4) ← 0s
(ES:EDI+0064h)(4) ← Base de IDT
(ES:EDI+0068h)(4) ← Limite de IDT
(ES:EDI+006Ch)(4) ← 0s
(ES:EDI+0070h)(4) ← Base de GDT
(ES:EDI+0074h)(4) ← Limite de GDT
(ES:EDI+0078h)(4) ← Attribut de LDT
(ES:EDI+007Ch)(4) ← Base de LDT
(ES:EDI+0080h)(4) ← Limite de LDT
(ES:EDI+0084h)(4) ← Attribut de GS
(ES:EDI+0088h)(4) ← Base de GS
(ES:EDI+008Ch)(4) ← Limite de GS
(ES:EDI+0090h)(4) ← Attribut de FS
(ES:EDI+0094h)(4) ← Base de FS
(ES:EDI+0098h)(4) ← Limite de FS
(ES:EDI+009Ch)(4) ← Attribut de DS
(ES:EDI+00A0h)(4) ← Base de DS

```
  (ES:EDI+00A4h)(4) ← Limite de DS
  (ES:EDI+00A8h)(4) ← Attribut de SS
  (ES:EDI+00ACh)(4) ← Base de SS
  (ES:EDI+00B0h)(4) ← Limite de SS
  (ES:EDI+00B4h)(4) ← Attribut de CS
  (ES:EDI+00B8h)(4) ← Base de CS
  (ES:EDI+00BCh)(4) ← Limite de CS
  (ES:EDI+00C0h)(4) ← Attribut de ES
  (ES:EDI+00C4h)(4) ← Base de ES
  (ES:EDI+00C8h)(4) ← Limite de ES
  (ES:EDI+00CCh)(4) ← Longueur de la table
  (ES:EDI+00D0h)(30h) ← Non utilisé
  (ES:EDI+0100h)(4) ← Registre temporaire IST
  (ES:EDI+0104h)(4) ← Registre temporaire I
  (ES:EDI+0108h)(4) ← Registre temporaire H
  (ES:EDI+010Ch)(4) ← Registre temporaire G
  (ES:EDI+0110h)(4) ← Registre temporaire F
  (ES:EDI+0114h)(4) ← Registre temporaire E
  (ES:EDI+0118h)(4) ← Registre temporaire D
  (ES:EDI+011Ch)(4) ← Registre temporaire C
  (ES:EDI+0120h)(4) ← Registre temporaire B
  (ES:EDI+0124h)(4) ← Registre temporaire A
FIN SI
```

Mnémonique

Instruction	Opcode	Description	Prérequis
LOADALL	0Fh 05h	Cette instruction permet d'effectuer la lecture des données à l'adresse 00800 de 00866 contenu dans les registres de segment	80286
LOADALL	0Fh 07h	Cette instruction permet d'effectuer la lecture des données à l'adresse ES:EDI.	80386

INTEL 8088+

Syntaxe

```
LOCK
```

Description

Cette instruction est utilisé comme préfixe avec d'autres instructions pour amener le microprocesseur à émettre un signal de vérouillage (*Lock*) sur le bus lors du traitement de l'instruction suivante.

Mnémonique

Instruction	Opcode	Description
LOCK	F0h	Cette instruction est utilisé comme préfixe avec d'autres instructions pour amener le microprocesseur à émettre un signal de vérouillage (*Lock*) sur le bus lors du traitement de l'instruction suivante.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 818
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 662 à 663.

Assembleur 80x86	**LODS**
INTEL 8088+	*LOaD String*

Syntaxe

LODS *chainesource*

Description

Cette instruction permet de copier un élément de l'adresse DS:SI dans le registre accumulateur et incrémente/décrémente le registre SI en fonction de la taille de l'opérande source et de l'état du drapeau de direction.

Algorithme

```
SI (charge un octet) ALORS
  AL ← chainesource
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 1
  SINON
    (E)SI ← (E)SI - 1
  FIN SI
SINON SI (charge un mot) ALORS
  AX ← chainesource
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 2
  SINON
    (E)SI ← (E)SI - 2
  FIN SI
SINON SI (charge un double mot) ALORS
  EAX ← chainesource
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 4
  SINON
    (E)SI ← (E)SI - 4
  FIN SI
```

```
SINON
  RAX ← chainesource
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 8
  SINON
    (E)SI ← (E)SI - 8
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LODS *mem8*	ACh	Charge l'octet de DS:(R)SI dans AL et incrémente ou décrement (R)SI.
LODS *mem16*	ADh	Charge le mot de DS:(R)SI dans AX et incrémente ou décrement (R)SI.
LODS *mem32*	ADh	Charge le double mot de DS:(R)SI dans EAX et incrémente ou décrement (R)SI.
LODS *mem64*	ADh	Charge le quadruple mot de DS:(R)SI dans RAX et incrémente ou décrement (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est

				pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVS
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 818
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 149.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 664 à 667.

Assembleur 80x86	**LODSB**
INTEL 8088+	*LOad String Byte*

Syntaxe

```
LODSB
```

Description

Cette instruction permet de copier un élément de l'adresse DS:SI dans le registre accumulateur et incrémente/décrémente le registre SI en fonction de la taille de l'opérande source et de l'état du drapeau de direction.

Algorithme

```
AL ← chainesource
SI DF = 0 ALORS
  (E)SI ← (E)SI + 1
SINON
  (E)SI ← (E)SI - 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LODSB	ACh	Charge l'octet de DS:(R)SI dans AL et incrémente ou décrement (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description

#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVS
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 818
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 410
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 149.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 664 à 667.

Assembleur 80x86	**LODSD**
INTEL 80386+	*LOaD String Double word*

Syntaxe

```
LODSD
```

Description

Cette instruction permet de copier un élément de l'adresse DS:SI dans le registre accumulateur et incrémente/décrémente le registre SI de 4 en fonction de l'état du drapeau de direction.

Algorithme

```
EAX ← chainesource
SI DF = 0 ALORS
  (E)SI ← (E)SI + 4
SINON
  (E)SI ← (E)SI - 4
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LODSD	ADh	Charge le double mot de DS:(R)SI dans EAX et incrémente ou décrement (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification

				d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVS
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 818
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 149.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 664 à 667.

Assembleur 80x86 LODSQ

x86-64+ *Load String Quadword*

Syntaxe

```
LODSQ
```

Description

Cette instruction permet de copier un élément de l'adresse DS:(R)SI dans le registre accumulateur et incrémente/décrémente le registre (R)SI de 8 en fonction de l'état du drapeau de direction.

Algorithme

```
RAX ← chainesource
SI DF = 0 ALORS
  (E)SI ← (E)SI + 8
SINON
  (E)SI ← (E)SI - 8
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LODSQ	ADh	Charge le quadruple mot de DS:(R)SI dans RAX et incrémente ou décrement (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification

				d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVS
Instruction assembleur 80x86 - Instruction STOS

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 664 à 667.

Assembleur 80x86	**LODSW**
INTEL 8088+	*LOaD String Word*

Syntaxe

```
LODSW
```

Description

Cette instruction permet de copier un élément de l'adresse DS:SI dans le registre accumulateur et incrémente/décrémente le registre SI de 2 en fonction de l'état du drapeau de direction.

Algorithme

```
AX ← chainesource
SI DF = 0 ALORS
  (E)SI ← (E)SI + 2
SINON
  (E)SI ← (E)SI - 2
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LODSW	ADh	Charge le mot de DS:(R)SI dans AX et incrémente ou décrement (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description

#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVS
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 819
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 410
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 149.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 664 à 667.

INTEL 8088+

Syntaxe

LOOP *étiquette*

Description

Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0.

Algorithme

```
CX ← CX - 1
SI CX <> 0 ALORS
  IP ← IP + Offset
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LOOP *rel8off*	E2h *cb*	Décrémente (R)CX, si (R)CX ne vaut pas 0 alors effectuer un saut court.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection	X	X	X	Une

général)				adresse mémoire dépasse la limite du segment de données ou n'est pas canonique

Voir également

Instruction assembleur 80x86 - Instruction LOOPE
Instruction assembleur 80x86 - Instruction LOOPNE
Instruction assembleur 80x86 - Instruction LOOPNZ
Instruction assembleur 80x86 - Instruction LOOPZ

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 819
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 410
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 151.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 668 à 669.

Assembleur 80x86 — LOOPD

INTEL 80386+ — *Loop Double*

Syntaxe

LOOPD *etiquette*

Description

Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0.

Algorithme

```
ECX ← ECX - 1
SI ECX <> 0 ALORS
   IP ← IP + Offset
FIN SI
```

Assembleur 80x86	**LOOPE**
INTEL 8088+	*LOOP Equal*

Syntaxe

LOOPE *étiquette*

Description

Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0 et si le drapeau ZF vaut 1.

Algorithme

```
CX ← CX - 1
SI CX <> 0 ET ZF = 1 ALORS
  IP ← IP + Offset
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LOOPE *rel8off*	E1h *cb*	Décrémente (R)CX, si (R)CX ne vaut pas 0 et ZF vaut 1 alors effectuer un saut court.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description

#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 819
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 410
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 151.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 668 à 669.

Assembleur 80x86 — LOOPED

INTEL 80386+ — *Loop Equal Double*

Syntaxe

LOOPED *etiquette*

Description

Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0 et si le drapeau ZF vaut 1.

Algorithme

```
ECX ← ECX - 1
SI ECX <> 0 ET ZF = 1 ALORS
   IP ← IP + Offset
FIN SI
```

Assembleur 80x86	LOOPNE
INTEL 8088+	*LOOP No Equal*

Syntaxe

LOOPNE *étiquette*

Description

Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0 et si le drapeau ZF vaut 0.

Algorithme

```
CX ← CX - 1
SI CX <> 0 ET ZF = 0 ALORS
  IP ← IP + Offset
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LOOPNE *rel8off*	E0h *cb*	Décrémente (R)CX, si (R)CX ne vaut pas 0 et que ZF vaut 0 alors effectuer un saut court.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description

#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 820
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 411
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 151.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 668 à 669.

Assembleur 80x86 — LOOPNED

INTEL 80386+ — *Loop Not Equal Double*

Syntaxe

LOOPNED *etiquette*

Description

Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0 et si le drapeau ZF vaut 0.

Algorithme

```
ECX ← ECX - 1
SI ECX <> 0 ET ZF = 0 ALORS
   IP ← IP + Offset
FIN SI
```

Assembleur 80x86	**LOOPNZ**
INTEL 8088+	*LOOP Non Zero*

Syntaxe

LOOPNZ *étiquette*

Description

Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0 et si le drapeau ZF vaut 0.

Algorithme

```
CX ← CX - 1
SI CX <> 0 ET ZF = 0 ALORS
  IP ← IP + Offset
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LOOPNZ *rel8off*	E0h *cb*	Décrémente (R)CX, si (R)CX ne vaut pas 0 et ZF ne vaut pas 0 alors effectuer un saut court.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description

#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 820
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 411
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 151.

Assembleur 80x86 — LOOPNZD

INTEL 80386+ — *Loop Not Zero Double*

Syntaxe

LOOPNZD *etiquette*

Description

Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0 et si le drapeau ZF vaut 0.

Algorithme

```
ECX ← ECX - 1
SI ECX <> 0 ET ZF = 0 ALORS
   IP ← IP + Offset
FIN SI
```

Assembleur 80x86	**LOOPZ**
INTEL 8088+	*LOOP Zero*

Syntaxe

LOOPZ *étiquette*

Description

Cette instruction de boucle permet de décrémenter le registre CX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre CX ne vaut pas 0 et si le drapeau ZF vaut 1.

Algorithme

```
CX ← CX - 1
SI CX <> 0 ET ZF = 1 ALORS
  IP ← IP + Offset
FIN SI
```

Mnémonique

Instruction	Opcode	Description
LOOPZ *rel8off*	E1h *cb*	Décrémente (R)CX, si (R)CX ne vaut pas 0 et que ZF vaut 1 alors effectuer un saut court.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description

#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 820
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 411
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 151.

Assembleur 80x86 — LOOPZD

INTEL 80386+ — *Loop Double Zero*

Syntaxe

LOOPZD *etiquette*

Description

Cette instruction de boucle permet de décrémenter le registre ECX (compteur de boucle) de 1 et par la suite de donner le contrôle à une *étiquette* destinataire tant que le registre ECX ne vaut pas 0 et si le drapeau ZF vaut 1.

Algorithme

```
ECX ← ECX - 1
SI ECX <> 0 ET ZF = 1 ALORS
   IP ← IP + Offset
FIN SI
```

Assembleur 80x86 — LSL

INTEL 80286+ — *Load Segment Limit*

Syntaxe

LSL *operande1,operande2*

Description

Cette instruction permet de charger la limite de segment d'un descripteur de segment spécifié avec l'opérande source dans l'opérande de destination et fixe le drapeau *ZF* du regsistre *EFLAGS*.

Algorithme

SI *SRC*(Offset) > limite du descripteur de table **ALORS**
  ZF ← 0
FIN SI
Lecture du descripteur de segment
SI type de descripteur de segment est conforme au code segment **ET** (CPL > DPL) **OU** (RPL > DPL)
OU type de segment n'est pas valide pour l'instruction **ALORS**
  ZF ← 0
SINON
  temp ← limite de segment de *SRC*
  SI G = 1 **ALORS**
    temp ← DécalageDesBitsVersLaGauche(12, temp) U 00000FFFh
  FIN SI
  SI taille de l'opérande = 32 bits **ALORS**
    DEST ← temp
  SINON
    DEST ← temp ∩ FFFFh
  FIN SI
FIN SI

Mnémonique

Instruction	Opcode	Description
LSL *reg16, reg/mem16*	0Fh 03h /r	Cette instruction permet de charger la limite de segment d'un descripteur de segment spécifié avec l'opérande source 16 bits dans l'opérande de destination de 16 bits et fixe le drapeau *ZF* du regsistre *EFLAGS*.
LSL *reg32, reg/mem16*	0Fh 03h /r	Cette instruction permet de charger la limite de segment d'un descripteur de segment spécifié avec l'opérande source 16 bits dans l'opérande de destination de 32 bits et fixe le drapeau *ZF* du regsistre *EFLAGS*.
LSL *reg64, reg/mem16*	0Fh 03h /r	Cette instruction permet de charger la limite de segment d'un descripteur de segment spécifié avec l'opérande source 16 bits dans l'opérande de destination de 64 bits et fixe le drapeau *ZF* du regsistre *EFLAGS*.

Voir également

Instruction assembleur 80x86 - Instruction ARPL
Instruction assembleur 80x86 - Instruction LAR
Instruction assembleur 80x86 - Instruction VERR
Instruction assembleur 80x86 - Instruction VERW

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 670 à 674.

Assembleur 80x86 — LSS

INTEL 80386+ — *Load Stack Segment*

Syntaxe

LSS *operande_cible,operande_source*

Description

Cette instruction permet de copier une adresse de mémoire contenu sur 32 bits dans la paire de registre de segment *SS* (Segment de pile) et dans un registre d'offset spécifié.

Algorithme

Registre d'offset ← Source
Registre de segment *SS* ← Source + 2

Mnémonique

Instruction	Opcode	Description
LSS *reg16, mem16:16*	0Fh B2h /r	Charge le SS:*reg16* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.
LSS *reg32, mem16:32*	0Fh B2h /r	Charge le SS:*reg32* avec un pointeur long dans la mémoire. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	L'opérande source est un registre.
			X	Cette instruction est exécuté en mode 64-bits.
#NP(Sélecteur)			X	Le registre DS, ES, FS ou GS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non présent.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur de segment non-nulle et le segment est marqué comme non

				présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite du descripteur de la table.
	X	X	X	Un registre de segment est chargé et le bit *TI* de sélecteur de segment est fixé, mais le sélecteur *LDT* est un sélecteur nulle.

	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment nulle dans un mode 64-bits ou avec *CPL* = 3.
	X	X	X	Le registre *SS* est chargé avec un sélecteur de segment *RPL* et le descripteur de segment *DPL* n'est pas égale au *CPL*.
	X	X	X	Le registre *SS* est chargé et le pointeur de segment n'est pas dans un segment de données écrivable.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code non-conforme, mais le *RPL* ou

				CPL est supérieur au *DPL*.
	X	X	X	Le registre DS, ES, FS ou GS n'est pas chargé et le segment pointe sur des données ou un segment de code en lecture.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 642 à 646.

Assembleur 80x86 — LTR

INTEL 80286+ — *Load Task Register*

Syntaxe

LTR *operande*

Description

Cette instruction permet de charger l'opérande source dans le champ du sélecteur de segment du registre de tâche.

Algorithme

SI SRC(Offset) > limite du descripteur de table **OU** *operande*(type) <> global **ALORS**
  EXCEPTION #GP(Sélecteur de segment)
FIN SI
Lecture du descripteur de segment
SI descripteur de segment n'est pas disponible dans TSS **ALORS**
  EXCEPTION #GP(Sélecteur de segment)
FIN SI
SI descripteur de segment n'est pas présent **ALORS**
  EXCEPTION #NP(Sélecteur de segment)
FIN SI
Descripteur de segment TSS(occupé) ← 1
Registre de tâche(Sélecteur de segment) ← *operande*
Registre de tâche(Descripteur de segment) ← Descripteur de segment TSS

Mnémonique

Instruction	Opcode	Description

LTR *reg/mem16*	0Fh 00h /3	Cette instruction permet de charger le sélecteur de segment dans le registre de tâche et charge le descripteur *TSS* dans le *GDT*.

Voir également

Instruction assembleur 80x86 - Instruction LGDT
Instruction assembleur 80x86 - Instruction LIDT
Instruction assembleur 80x86 - Instruction LLDT
Instruction assembleur 80x86 - Instruction STR
Instruction assembleur 80x86 - Instruction SGDT
Instruction assembleur 80x86 - Instruction SIDT
Instruction assembleur 80x86 - Instruction SLDT

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 675 à 677.

Assembleur 80x86 — LZCNT

AMD K10 (SSE4a)+ — *Count Leading Zeros*

Syntaxe

LZCNT *regdest*, *source*

Description

Cette instruction permet de compter le nombre de bits à 0 dans un registre 16, 32 ou 64 bits contenu dans l'opérande source.

Mnémonique

Instruction	Opcode	Description
LZCNT *reg16*, *reg/mem16*	F3h 0Fh BDh /r	Compte le nombre de bit à 0 dans *reg/mem16*.
LZCNT *reg32*, *reg/mem32*	F3h 0Fh BDh /r	Compte le nombre de bit à 0 dans *reg/mem32*.
LZCNT *reg64*, *reg/mem64*	F3h 0Fh BDh /r	Compte le nombre de bit à 0 dans *reg/mem64*.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Assembleur 80x86 — MASKMOVDQU

INTEL Pentium 4 (SSE2)+ — *Mask Move of Double Quadword Unaligned*

Syntaxe

MASKMOVDQU *dest*, *source*

Description

Cette instruction permet d'entreposer les octets sélectionnés par l'opérande source dans une emplacement mémoire de 128 bits.

Fonction

```
SI MASK(7) = 1 ALORS
  dest((E)DI) ← source(7..0)
FIN SI
IF MASK(15) = 1 ALORS
  dest((E)DI+1) ← source(15..8)
FIN SI
SI MASK(127) = 1 ALORS
  dest(DI/EDI+15) ← source(127..120)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MASKMOVDQU *xmm1*, *xmm2*	66h 0Fh F7h /r	Cette instruction permet de sélectionner les octets à écrire du *xmm1* à l'emplacement mémoire utilisé par le masque d'octet du

		xmm2.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)			X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(0)			X	Une adresse mémoire dépasse la limite du segment CS, DS, ES, FS ou GS ou n'est pas canonique
#NM()	X		X	Si le bit TS du registre CR0 est fixé à 1.
#UD(Opcode invalide)	X		X	Si le bit EM du registre CR0 est fixé à 1.
	X		X	Si le

				OXFXSR dans CR4 vaut 0.
	X		X	Si le drapeau de fourniture SSE2 du CPUID vaut 0.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 678 à 679.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MASKMOVQ

Mask Move of Quadword

Syntaxe

MASKMOVQ *source*,*masque*

Description

Cette instruction permet d'entreposer les octets sélectionner de l'opérande source dans un emplacement mémoire de 64 bits.

Algorithme

```
MODULE moveByte(checkBit,moveBits)
  SI masque(checkBit) = 1 ALORS
    mem64(EDI)(moveBits) ← source(moveBits)
  SINON
    mem64(EDI)(moveBits) ← 0
  FIN SI
FIN MODULE

moveByte(7 , 7..0)
moveByte(15, 15..8)
moveByte(23, 23..16)
moveByte(31, 31..24)
moveByte(39, 39..32)
moveByte(47, 47..40)
moveByte(55, 55..48)
moveByte(63, 63..56)
```

Mnémonique

Instruction	Opcode	Description
MASKMOVQ *mm1,mm2*	0Fh F7h /r	Cette instruction permet d'entreposer les octets sélectionner de l'opérande source dans un emplacement mémoire de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 680 à 683.

Assembleur 80x86 — MAXPD

INTEL Pentium 4 (SSE2)+ — *Maximum Packed Double-Precision Floating-Point Values*

Syntaxe

MAXPD *dest,source*

Description

Cette instruction permet de retourner la valeur maximal de chacune des paires de valeur entre l'opérande source et l'opérande de destination de nombre réel de double précision.

Algorithme

SI ((*dest*(0..63) = 0,0 ∩ *source*(0..63) = 0,0) U IsSNaN(*dest*(0..63)) U IsSNaN(*source*(0..63)) U *dest*(0..63) <= *source*(0..63)) **ALORS**
  dest(0..63) ← *source*(0..63)
FIN SI
SI ((*dest*(64..127) = 0,0 ∩ *source*(64..127) = 0.0) U IsSNaN(*dest*(64..127)) U IsSNaN(*source*(64..127)) U *dest*(64..127) <= *source*(64..127)) **ALORS**
  dest(64..127) ← *source*(64..127)
FIN SI

Mnémonique

Instruction	Opcode	Description
MAXPD *xmm1*, *xmm2/m128*	66h 0Fh 5Fh /r	Cette instruction permet de retourner la valeur maximal de chacune des paires de valeur entre l'opérande source et l'opérande de destination de nombre réel de double

		précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 684 à 686.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MAXPS

Maximum Packed Double-Precision Floating-Point Values

Syntaxe

```
MAXPS dest,source
```

Description

Cette instruction permet de retourner la valeur maximal de chacune des paires de valeur entre l'opérande source et l'opérande de destination.

Algorithme

```
BOUCLE POUR i ← 0 JUSQU'A 3 FAIRE
  SI dest(octet i) n'est pas indéfinit ALORS
    dest(octet i) ← dest(octet i)
  FIN SI
  SI source(octet i) n'est pas indéfini ALORS
    dest(octet i) ← source(octet i)
  FIN SI
  SI source(i) >= dest(i) ALORS
    dest(octet i) ← source(octet i)
  FIN SI
FIN BOUCLE POUR
```

Mnémonique

Instruction	Opcode	Description

MAXPS *xmm1,xmm2/m128*	0Fh 5Fh /r	Cette instruction permet de retourner la valeur maximal de chacune des paires de valeur entre l'opérande source et l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 687 à 689.

Assembleur 80x86 — MAXSD

INTEL Pentium 4 (SSE2)+ — *Maximum Scalar Double-Precision Floating-Point Value*

Syntaxe

MAXSD *dest*,*source*

Description

Cette instruction permet de retourner la valeur maximal entre l'opérande source et destination de nombre réel de simple précision.

Algorithme

SI *dest*(0..63) = 0,0 ∩ *source*(0..63) = 0,0) U IsSNaN(*dest*[0..63]) U IsSNaN(*source*(0..63)) U *dest*[0..63] <= *source*(0..63)) **ALORS**
  dest(0..63) ← *source*(0..63)
FIN SI

Mnémonique

Instruction	Opcode	Description
MAXSD *xmm1*, *xmm2/m64*	F2h 0Fh 5Fh /r	Cette instruction permet de retourner la valeur maximal entre l'opérande source et destination de nombre réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 690 à 692.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MAXSS

Maximum Scalar Single-Precision Floating-Point Value

Syntaxe

MAXSS *dest,source*

Description

Cette instruction permet de retourner la valeur maximal entre l'opérande source et destination.

Algorithme

SI *dest* n'est pas indéfinit **ALORS**
  dest ← *dest*
FIN SI
SI *source* n'est pas indéfini **ALORS**
  dest ← *source*
FIN SI
SI *source*(*i*) >= dest(*i*) **ALORS**
  dest ← *source*
FIN SI

Mnémonique

Instruction	Opcode	Description
MAXSS *xmm1,xmm2/m32*	F3h 0Fh 5Fh /r	Cette instruction permet de retourner la valeur maximal entre

		l'opérande source et destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 693 à 694.

Assembleur 80x86 MFENCE

INTEL Pentium 4 (SSE2)+ *Memory Fence*

Syntaxe

MFENCE

Description

Cette instruction permet d'agir comme une barrière pour forcer une priorité en mémoire (sérialisation) entre les instructions précédant le *MFENCE* et les instructions de chargement et d'entreposage précédent le *MFENCE*.

Mnémonique

Instruction	Opcode	Description
MFENCE	0Fh AEh F0h	Force l'ordre de sérialisation dans l'opération de chargement et d'entreposage

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction n'est pas supporté comme l'indique le bit 26 du registre EDX

				de la fonction 0000_0000h dans l'instruction CPUID.

Voir également

Instruction assembleur 80x86 - Instruction LFENCE
Instruction assembleur 80x86 - Instruction SFENCE

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 695 à 696.

Assembleur 80x86 — MINPD

INTEL Pentium 4 (SSE2)+

Minimum Packed Double-Precision Floating-Point Values

Syntaxe

MINPD *dest*,*source*

Description

Cette instruction permet de retourner la valeur minimal de chacune des paires de valeur entre l'opérande source et l'opérande de destination de nombre réel de double précision.

Algorithme

SI ((*dest*(0..63) = 0,0 ∩ *source*(0..63) = 0,0) U IsSNaN(*dest*(0..63)) U IsSNaN(*source*(0..63)) U *dest*(0..63) >= *source*(0..63)) **ALORS**
  dest(0..63) ← *source*(0..63)
FIN SI
SI ((*dest*(64..127) = 0,0 ∩ *source*(64..127) = 0,0) U IsSNaN(*dest*(64..127)) U IsSNaN(*source*(64..127)) U *dest*(64..127) >= *source*(64..127)) **ALORS**
  dest(64..127) ← *source*(64..127)
FIN SI

Mnémonique

Instruction	Opcode	Description
MINPD *xmm1*, *xmm2/m128*	66h 0Fh 5Dh /r	Cette instruction permet de retourner la valeur minimal de chacune des paires de valeur entre l'opérande source et l'opérande de destination de nombre réel de double

		précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 697 à 700.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MINPS

Minimum Packed Double-Precision Floating-Point Values

Syntaxe

MINPS *dest,source*

Description

Cette instruction permet de retourner la valeur minimal de chacune des paires de valeur entre l'opérande source et l'opérande de destination.

Algorithme

```
BOUCLE POUR i ← 0 JUSQU'A 3 FAIRE
  SI dest(octet i) n'est pas indéfinit ALORS
    dest(octet i) ← dest(octet i)
  FIN SI
  SI source(octet i) n'est pas indéfini ALORS
    dest(octet i) ← source(octet i)
  FIN SI
  SI source(i) < dest(i) ALORS
    dest(octet i) ← source(octet i)
  FIN SI
FIN BOUCLE POUR
```

Mnémonique

Instruction	Opcode	Description

MINPS *xmm1,xmm2/m128*	0Fh 5Dh /r	Cette instruction permet de retourner la valeur minimal de chacune des paires de valeur entre l'opérande source et l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 701 à 703.

Assembleur 80x86 — MINSD

INTEL Pentium 4 (SSE2)+

Minimum Scalar Double-Precision Floating-Point Value

Syntaxe

MINSD *dest*,*source*

Description

Cette instruction permet de retourner la valeur minimal entre l'opérande source et destination de nombre réel de simple précision.

Algorithme

SI ((*dest*(0..63) = 0,0 ∩ *source*(0..63) = 0,0) U IsSNaN(*dest*(0..63)) U IsSNaN(*source*(0..63)) U *dest*(0..63) >= *source*(0..63)) **ALORS**
  dest(0..63) ← *source*(0..63)
FIN SI

Mnémonique

Instruction	Opcode	Description
MINSD *xmm1, xmm2/m64*	F2h 0Fh 5Dh /r	Cette instruction permet de retourner la valeur minimal entre l'opérande source et destination de nombre réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 704 à 706.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MINSS

Minimum Scalar Single-Precision Floating-Point Value

Syntaxe

MINSS *dest,source*

Description

Cette instruction permet de retourner la valeur minimal entre l'opérande source et destination.

Algorithme

```
SI dest n'est pas indéfinit ALORS
  dest ← dest
FIN SI
SI source n'est pas indéfini ALORS
  dest ← source
FIN SI
SI source(i) < dest(i) ALORS
  dest ← source
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MINSS *xmm1,xmm2/m32*	F3h 0Fh 5Dh /r	Cette instruction permet de retourner la valeur minimal entre

		l'opérande source et destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 707 à 709.

Assembleur 80x86 MONITOR

INTEL Pentium 4+ *Monitor*

Syntaxe

MONITOR
MONITOR EAX, ECX, EDX

Description

Cette instruction permet d'indiquer au microprocesseur quel rangé d'adresse est à surveiller par l'instruction *STORE*. Normalement, cette instruction est relié avec l'instruction MWAIT.

Mnémonique

Instruction	Opcode	Description
MONITOR	0Fh 01h C8h	Cette instruction permet d'indiquer au microprocesseur quel rangé d'adresse est à surveiller par l'instruction *STORE*.

Voir également

Instruction assembleur 80x86 - Instruction MWAIT

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 710 à 712.

Assembleur 80x86	**MOV**
INTEL 8088+	*MOVe*

Syntaxe

MOV *opérandecible,opérandesource*

Description

Cette instruction permet de copier opérande source dans une opérande destinataire.

Algorithme

opérandecible ← *opérandesource*

Mnémonique

Instruction	**Opcode**	**Description**
MOV *reg/mem8*, *reg8*	88h /r	Copie le contenu du registre 8 bits dans l'opérande de registre ou mémoire 8 bits.
MOV *reg/mem16*, *reg16*	89h /r	Copie le contenu du registre 16 bits dans l'opérande de registre ou mémoire 16 bits.
MOV *reg/mem32*, *reg32*	89h /r	Copie le contenu du registre 32 bits dans l'opérande de registre ou mémoire 32 bits.
MOV *reg/mem64*, *reg64*	89h /r	Copie le contenu du registre 64 bits dans l'opérande de registre ou

		mémoire 64 bits.
MOV *reg8*, *reg/mem8*	8Ah /r	Copie le contenu de l'opérande de registre ou mémoire 8 bits dans le registre 8 bits.
MOV *reg16*, *reg/mem16*	8Bh /r	Copie le contenu de l'opérande de registre ou mémoire 16 bits dans le registre 16 bits.
MOV *reg32*, *reg/mem32*	8Bh /r	Copie le contenu de l'opérande de registre ou mémoire 32 bits dans le registre 32 bits.
MOV *reg64*, *reg/mem64*	8Bh /r	Copie le contenu de l'opérande de registre ou mémoire 64 bits dans le registre 64 bits.
MOV *reg16/32/64/mem16,segReg*	8Ch /r	Copie le contenu du registre de segment dans l'opérande de registre ou mémoire 16, 32 ou 64 bits.
MOV *segReg*, *reg/mem16*	8Eh /r	Copie le contenu de l'opérande de registre ou mémoire 16 bits dans le registre de segment.
MOV AL,[*yyxx*h]	A0h *xx*h *yy*h	Copie le contenu de l'opérande de l'adresse mémoire 8 bits dans le registre AL.
MOV AX, [*yyxx*h]	A1h *xx*h *yy*h	Copie le contenu de l'opérande de l'adresse mémoire 16 bits dans le registre AX.
MOV EAX, *moffset32*	A1h *dw*	Copie le contenu de l'opérande de l'adresse mémoire 32 bits dans le registre EAX.

MOV RAX, *moffset64*	A1h *dw*	Copie le contenu de l'opérande de l'adresse mémoire 64 bits dans le registre RAX.
MOV *moffset8*, AL	A2h *dw*	Copie le registre AL dans l'opérande de l'adresse mémoire 8 bits.
MOV *moffset16*, AX	A3h *dw*	Copie le registre AX dans l'opérande de l'adresse mémoire 16 bits.
MOV *moffset32*, EAX	A3h *dw*	Copie le registre EAX dans l'opérande de l'adresse mémoire 32 bits.
MOV *moffset64*, RAX	A3h *dw*	Copie le registre RAX dans l'opérande de l'adresse mémoire 64 bits.
MOV *reg8*, *imm8*	(B0h+*rb*) *ib*	Copie une valeur immédiate de 8 bits dans un registre 8 bits.
MOV *reg16*, *imm16*	(B8h+*rw*) *iw*	Copie une valeur immédiate de 16 bits dans un registre 16 bits.
MOV *reg32*, *imm32*	(B8h+*rd*) *id*	Copie une valeur immédiate de 32 bits dans un registre 32 bits.
MOV *reg64*, *imm64*	(B8h+*rq*) *iq*	Copie une valeur immédiate de 64 bits dans un registre 64 bits.
MOV *reg/mem8*, *imm8*	C6h /0 *ib*	Copie une valeur immédiate 8 bits dans une opérande de registre ou mémoire 8 bits.
MOV *reg/mem16*, *imm16*	C7h /0 *iw*	Copie une valeur immédiate 16 bits dans une opérande de registre ou mémoire 16 bits.

MOV *reg/mem32*, *imm32*	C7h /0 *id*	Copie une valeur immédiate 32 bits dans une opérande de registre ou mémoire 32 bits.
MOV *reg/mem64*, *imm32*	C7h /0 *id*	Copie une valeur immédiate 32 bits dans une opérande de registre ou mémoire 64 bits.
MOV CR*n*, *reg32*	0Fh 22h /r	Copie un registre 32 bits dans un registre CR*n*.
MOV CR*n*, *reg64*	0Fh 22h /r	Copie un registre 64 bits dans un registre CR*n*.
MOV *reg32*, CR*n*	0Fh 20h /r	Copie un registre CR*n* dans un registre 32 bits.
MOV *reg64*, CR*n*	0Fh 20h /r	Copie un registre CR*n* dans un registre 64 bits.
MOV CR8, *reg32*	F0h 0Fh 22h/r	Copie un registre 32 bits dans le registre CR8.
MOV CR8, *reg64*	F0h 0Fh 22h/r	Copie un registre 64 bits dans le registre CR8.
MOV *reg32*, CR8	F0h 0Fh 20h/r	Copie un registre CR8 dans le registre 32 bits.
MOV *reg64*, CR8	F0h 0Fh 20h/r	Copie un registre CR8 dans le registre 64 bits.
MOV *reg32*, DR*n*	0Fh 21h /r	Copie un registre DR*n* dans le registre 32 bits.
MOV *reg64*, DR*n*	0Fh 21h /r	Copie un registre DR*n* dans le registre 64 bits.
MOV DR*n*, *reg32*	0Fh 23h /r	Copie un registre 32 bits dans le

		registre DR*n*.
MOV DR*n*, *reg64*	0Fh 23h /r	Copie un registre 64 bits dans le registre DR*n*.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé
#DB(Déboguage)		X	X	Un registre de déboguage est référencé quand une détection générale du bit GD du registre DR7 est fixé.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique

			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
		X	X	Le *CPL* ne vaut pas 0.
			X	La valeur 1 est écrite dans n'importe quel des 32 bits haut du registre DR6 ou DR7 en mode 64 bits.
	X		X	Une tentative de fixer le CR0.PG = 1 et le CR0.PE = 0.
	X		X	Une tentative de fixer le CR0.CD = 0 et le CR0.NW = 1.
	X		X	Bits réservé à 1 dans le pointeur de la table de page

				de répertoire (utilise le mode «*legacy*» d'adresse physique étendue) et l'instruction modifie le registre *CR0*, *CR3* ou *CR4*.
	X		X	Une tentative d'écrire 1 dans n'importe quel bit réservé du registre *CR0*, *CR3*, *CR4* ou *CR8*.
	X		X	Une tentative de fixé le *CR0*.PG en mode long est activé (EFER.LME = 1), mais l'adresse d'extensions de page est désactivé (CR4.PAE = 0).
			X	Une tentative d'effacer le CR4.PAE en mode «*long*» est activé (EFER.LMA =

				1).
#GP(Sélecteur)			X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite de la table du descripteur.
			X	Un registre de segment est chargé et le bit *TI* de sélecteur de segment est fixé, mais le sélecteur *LDT* est un sélecteur nulle.
			X	Le registre *SS* est chargé avec un sélecteur de segment nulle dans un mode non 64 bits ou avec CPL = 3.
			X	Le registre *SS* est chargé et le sélecteur de segment *RPL* et le descripteur de segment *DPL*

				n'est pas égale au *CPL*.
			X	Le registre *SS* est chargé et le segment pointe dans un segment de données non écrivable.
			X	Le registre DS, ES, FS ou GS est chargé et le segment pointe sur des données ou un segment de code non-conforme, mais le *RPL* ou *CPL* est supérieur au *DPL*.
			X	Le registre DS, ES, FS ou GS est chargé et le segment ne pointe pas sur un segment de données ou un segment de code en lecture.
#NP(Sélecteur)			X	Le registre DS, ES, FS ou GS n'est pas

				chargé avec une valeur non-nulle.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur non-nulle et le segment est marqué non présent.
#UD(Opcode invalide)			X	Une tentative de chargement dans le registre *CS*.
		X	X	Le registre *DR4* ou *DR5* est référencé quand le bit d'extensions de déboguage (DE) du

				registre *CR4* est fixé.
			X	Un registre de déboguage illégale (DR8 à DR15) est référencé.
#UD(Instruction invalide)	X	X	X	Un registre de contrôle illégale est référencé (CR1, CR6 à CR7, CR8 à CR15).
	X	X	X	L'utilisation d'un préfixe de verroue *LOCK* en lecture du registre CR8 n'est pas supporté, comme indiqué par le bit 4 du registre ECX de la fonction 8000_0001h de l'instruction *CPUID*.

Voir également

Instruction assembleur 80x86 - Instruction CLTS
Instruction assembleur 80x86 - Instruction LMSW
Instruction assembleur 80x86 - Instruction MOVD

Instruction assembleur 80x86 - Instruction MOVSX
Instruction assembleur 80x86 - Instruction MOVZX
Instruction assembleur 80x86 - Instruction MOVSXD
Instruction assembleur 80x86 - Instruction MOVS
Instruction assembleur 80x86 - Instruction SMSW

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 820
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 411
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 156.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 713 à 724.

Assembleur 80x86 — MOVAPD

INTEL Pentium 4 (SSE2)+ — *Move Aligned Packed Double-Precision Floating-Point Values*

Syntaxe

MOVAPD *destination*, *source*

Description

Cette instruction permet de copier le contenu de 2 paquets alignés de valeurs réel de double précision.

Algorithme

destination ← *source*

Mnémonique

Instruction	Opcode	Description
MOVAPD *xmm1*, *xmm2/m128*	66h 0Fh 28h /r	Cette instruction permet de copier le contenu de 2 paquets alignés de valeurs réel de double précision.
MOVAPD *xmm2/m128*, *xmm1*	66h 0Fh 29h /r	Cette instruction permet de copier le contenu de 2 paquets alignés de valeurs réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 725 à 728.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MOVAPS

Move Aligned Four Packed Single

Syntaxe

MOVAPS *dest,source*

Description

Cette instruction permet de copier le contenu de 4 paquets alignés de valeurs réel de simple précision (4 x 32 bits).

Algorithme

dest ← source

Mnémonique

Instruction	Opcode	Description
MOVAPS *xmm1,xmm2/m128*	0Fh 28h /r	Cette instruction permet de copier le contenu de 4 paquets alignés de valeurs réel de simple précision (4 x 32 bits).
MOVAPS *xmm2/m128,xmm1*	0Fh 29h /r	Cette instruction permet de copier le contenu de 4 paquets alignés de valeurs réel de simple précision (4 x 32 bits).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 729 à 731.

Assembleur 80x86 — MOVBE

SSE4.2+ — *Move Data After Swapping Bytes*

Syntaxe

MOVBE *dest*,*source*

Description

Cette instruction permet d'effectuer l'échange des octets lors de l'opération de copiage de l'opérande source dans l'opérande de destination.

Algorithme

SI taille de l'opérande = 16 bits **ALORS**
  dest(7..0) ← *source*(15..8)
  dest(15..8) ← *source*(7..0)
SINON SI taille de l'opérande = 32 bits **ALORS**
  dest(7..0) ← *source*(31..24)
  dest(15..8) ← *source*(23..16)
  dest(23..16) ← *source*(15..8)
  dest(31..23) ← *source*(7..0)
SINON SI taille de l'opérande = 64 bits **ALORS**
  dest(7..0) ← *source*(63..56)
  dest(15..8) ← *source*(55..48)
  dest(23..16) ← *source*(47..40)
  dest(31..24) ← *source*(39..32)
  dest(39..32) ← *source*(31..24)
  dest(47..40) ← *source*(23..16)
  dest(55..48) ← *source*(15..8)
  dest(63..56) ← *source*(7..0)
FIN SI

Mnémonique

Instruction	Opcode	Description
MOVBE *r16*, *m16*	0Fh 38h F0h /r	Cette instruction permet d'inverser l'ordre des octets d'un emplacement mémoire 16 bits lors de son copiage dans un registre 16 bits.
MOVBE *r32*, *m32*	0Fh 38h F0h /r	Cette instruction permet d'inverser l'ordre des octets d'un emplacement mémoire 32 bits lors de son copiage dans un registre 32 bits.
MOVBE *r64*, *m64*	(REX.W) 0Fh 38h F0h /r	Cette instruction permet d'inverser l'ordre des octets d'un emplacement mémoire 64 bits lors de son copiage dans un registre 64 bits.
MOVBE *m16*, *r16*	0Fh 38h F1h /r	Cette instruction permet d'inverser l'ordre des octets d'un registre 16 bits lors de son copiage dans un emplacement mémoire 16 bits.
MOVBE *m32*, *r32*	0Fh 38h F1h /r	Cette instruction permet d'inverser l'ordre des octets d'un registre 32 bits lors de son copiage dans un emplacement mémoire 32 bits.
MOVBE *m64*, *r64*	(REX.W) 0Fh 38h F1h /r	Cette instruction permet d'inverser l'ordre des octets d'un registre 64 bits lors de son copiage dans un emplacement mémoire 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 732 à 734.

Assembleur 80x86 — MOVD

INTEL Pentium MMX+ — *Move Doubleword or Quadword*

Syntaxe

MOVD *dest*, *source*

Description

Cette instruction permet de copier l'opérande dans un registre *XMM* ou vice-verça.

Algorithme

SI l'opérande de destination est un registre MMX **ALORS**
  dest(31..0) ← *source*
  dest(63..32) ← 00000000h
SINON
  dest(31..0) ← *source*
  dest(127..32) ← 00000000000000000000000000h
FIN SI
SI l'opérande source est un registre MMX ou XXM **ALORS**
  dest ← *source*(31..0)
FIN SI

Mnémonique

Instruction	Opcode	Description
MOVD *xmm, reg/mem32*	66h 0Fh 6Eh /r	Copie une valeur 32 bits de l'emplacement mémoire ou registre 32 bits dans un registre *XMM*.

MOVD *xmm*, *reg/mem64*	66h 0Fh 6Eh /r	Copie une valeur 64 bits de l'emplacement mémoire ou registre 64 bits dans un registre *XMM*.
MOVD *reg/mem32*, *xmm*	66h 0Fh 7Eh /r	Copie une valeur 32 bits du registre *XMM* dans l'emplacement mémoire ou registre 32 bits.
MOVD *reg/mem64*, *xmm*	66h 0Fh 7Eh /r	Copie une valeur 64 bits du registre *XMM* dans l'emplacement mémoire ou registre 64 bits.
MOVD *mmx*, *reg/mem32*	0Fh 6Eh /r	Copie une valeur 32 bits de l'emplacement mémoire ou registre 32 bits dans le registre *MMX*
MOVD *mmx*, *reg/mem64*	0Fh 6Eh /r	Copie une valeur 64 bits de l'emplacement mémoire ou registre 64 bits dans le registre *MMX*
MOVD *reg/mem32*, *mmx*	0Fh 7Eh /r	Copie une valeur 32 bits du registre *MMX* dans l'emplacement mémoire ou registre 32 bits
MOVD *reg/mem64*, *mmx*	0Fh 7Eh /r	Copie une valeur 64 bits du registre *MMX* dans l'emplacement mémoire ou registre 64 bits

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Cette instruction n'est pas supporté, comme indiqué par le

				bit 23 du registre EDX de la fonction 0000_00001h de l'instruction *CPUID*.
	X	X	X	Les ensembles d'instructions SSE2 ne sont pas supporté, comme indiqué par le bit 26 du registre EDX de la fonction 0000_0001h de l'instruction *CPUID*.
	X	X	X	Le bit d'émulation (EM) du registre CR0 est fixé à 1.
	X	X	X	L'instruction utilise les registres *XMM* avec le registre *CR4.OSFXSR*=0.
#NM(Périphérique non disponible)	X	X	X	Le bit d'échangeur de tâche (TS) du registre CR0 est fixé à 1.
#SS(Pile)	X	X	X	Une adresse

				mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#MF(x87 virgule flottante)	X	X	X	Un exception de nombre réel x87 s'est produite et l'instruction référence un registre *MMX*.
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVDQA
Instruction assembleur 80x86 - Instruction MOVDQU
Instruction assembleur 80x86 - Instruction MOVDQ2Q
Instruction assembleur 80x86 - Instruction MOVQ
Instruction assembleur 80x86 - Instruction MOVQ2DQ

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 735 à 737.

Assembleur 80x86 — MOVDDUP

INTEL Pentium 4 (SSE3)+ — *Move One Double-FP and Duplicate*

Syntaxe

MOVDDUP *destination*, *source*

Description

Cette instruction permet de copier les 64 bits d'une opérande dans la partie basse et haute d'une opérande de 128 bits.

Algorithme

destination(0..63) ← *source*(0..63) *destination*(64..127) ← *source*(0..63)

Mnémonique

Instruction	Opcode	Description
MOVDDUP *xmm1, xmm2/m64*	F2h 0Fh 12h /r	Cette instruction permet de copier les 64 bits d'une opérande dans la partie basse et haute d'une opérande de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 738 à 741.

Assembleur 80x86 — MOVDQ2Q

INTEL Pentium 4 (SSE2)+ — *Move Quadword from XMM to MMX Technology Register*

Syntaxe

MOVDQ2Q *destination*, *source*

Description

Cette instruction permet de copier la partie basse d'un quadruple mots d'un opérande source vers un opérande de destination.

Algorithme

destination ← *source*(0..63)

Mnémonique

Instruction	Opcode	Description
MOVDQ2Q *mm*, *xmm*	F2h 0Fh D6h /r	Cette instruction permet de copier la partie basse d'un quadruple mots d'un opérande source vers un opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 748 à 749.

Assembleur 80x86 MOVDQA

INTEL Pentium 4 (SSE2)+ *Move Aligned Double Quadword*

Syntaxe

MOVDQA *destination*, *source*

Description

Cette instruction permet de copier un double quadruple mots d'un opérande source vers un opérande de destination.

Algorithme

destination ← *source*

Mnémonique

Instruction	Opcode	Description
MOVDQA *xmm1*, *xmm2/m128*	66h 0Fh 6Fh /r	Cette instruction permet de copier un double quadruple mots d'un opérande source vers un opérande de destination.
MOVDQA *xmm2/m128*, *xmm1*	66h 0Fh 7Fh /r	Cette instruction permet de copier un double quadruple mots d'un opérande source vers un opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 742 à 744.

Assembleur 80x86 — MOVDQU

INTEL Pentium 4 (SSE2)+ — *Move Unaligned Double Quadword*

Syntaxe

MOVDQU *destination*, *source*

Description

Cette instruction permet de copier un double quadruple mots désaligné d'un opérande source vers un opérande de destination.

Algorithme

destination ← *source*

Mnémonique

Instruction	Opcode	Description
MOVDQU *xmm1*, *xmm2/m128*	F3h 0Fh 6Fh /r	Cette instruction permet de copier un double quadruple mots désaligné d'un opérande source vers un opérande de destination.
MOVDQU *xmm2/m128*, *xmm1*	F3h 0Fh 7Fh /r	Cette instruction permet de copier un double quadruple mots désaligné d'un opérande source vers un opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 745 à 747.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MOVHLPS

Move High to Low Packed Single

Syntaxe

MOVHLPS *dest,source*

Description

Cette instruction permet de copier le contenu du haut d'un paquet de valeurs réel de simple précision dans sa partie basse.

Algorithme

dest(63..32) ← *source*(127..96) *dest*(31..0) ← *source*(95..64)

Mnémonique

Instruction	Opcode	Description
MOVHLPS *xmm1,xmm2*	0Fh 12h /r	Cette instruction permet de copier le contenu du haut d'un paquet de valeurs réel de simple précision dans sa partie basse.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 750 à 751.

Assembleur 80x86 — MOVHPD

INTEL Pentium 4 (SSE2)+

Move High Packed Double-Precision Floating-Point Value

Syntaxe

MOVHPD *destination*, *source*

Description

Cette instruction permet de copier le contenu du haut de deux paquets de valeurs réel de double précision dans une opérande de destination.

Algorithme

destination(64..127) ← *source*
destination ← *source*(64..127)

Mnémonique

Instruction	Opcode	Description
MOVHPD *xmm*, *m64*	66 0F 16 /r	Cette instruction permet de copier le contenu du haut de deux paquets de valeurs réel de double précision dans une opérande de destination.
MOVHPD *m64*, *xmm*	66h 0Fh 17h /r	Cette instruction permet de copier le contenu du haut de deux paquets de valeurs réel de double précision dans une opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 752 à 753.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MOVHPS

Move High Packed Single-Precision Floating-Point Values

Syntaxe

MOVHPS *dest,source*

Description

Cette instruction permet de copier le contenu du haut de deux paquets de valeurs réel de simple précision dans une opérande de destination.

Algorithme

SI *dest* est mémoire 64 bits **ALORS**
  dest(63..32) ← *source*(127..96)
  dest(31..0) ← *source*(95..64)
FIN SI
SI *source* est mémoire 64 bits **ALORS**
  dest(127..96) ← *source*(63..32)
  dest(95..64) ← *source*(31..0)
FIN SI

Mnémonique

Instruction	Opcode	Description
MOVHPS *xmm,m64*	0Fh 16h /r	Cette instruction permet de copier le contenu du haut de deux paquets de valeurs réel de simple précision dans

		une opérande de destination.
MOVHPS *m64,xmm*	0Fh 17h /r	Cette instruction permet de copier le contenu du haut de deux paquets de valeurs réel de simple précision dans une opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 754 à 757.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MOVLHPS

Move Low to High Packed Single

Syntaxe

MOVLHPS *dest,source*

Description

Cette instruction permet de copier le contenu du bas d'un paquet de valeurs réel de simple précision dans sa partie haute.

Algorithme

dest(127..96) ← *source*(63..32)
dest(95..64) ← *source*(31..0)

Mnémonique

Instruction	Opcode	Description
MOVLHPS *xmm1,xmm2*	0Fh 16h /r	Cette instruction permet de copier le contenu du bas d'un paquet de valeurs réel de simple précision dans sa partie haute.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 758 à 759.

Assembleur 80x86 — MOVLPD

INTEL Pentium 4 (SSE2)+ — *Move Low Packed Double-Precision Floating-Point Value*

Syntaxe

MOVLPD *destination*, *source*

Description

Cette instruction permet de copier le contenu du bas de deux paquets de valeurs réel de double précision dans un opérande de destination.

Algorithme

```
SI IsXMM(Destination) ALORS
  destination(0..63) ← source
SINON
  destination ← source(0..63)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MOVLPD *xmm*, *m64*	66h 0Fh 12h /r	Cette instruction permet de copier le contenu du bas de deux paquets de valeurs réel de double précision dans un opérande de destination.
MOVLPD *m64*, *xmm*	66h 0Fh 13h /r	Cette instruction permet de copier le contenu du bas de deux paquets de valeurs réel de double précision dans

		un opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 760 à 762.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MOVLPS

Move Low Packed Single-Precision Floating-Point Values

Syntaxe

MOVLPS *dest,source*

Description

Cette instruction permet de copier le contenu du bas de deux paquets de valeurs réel de simple précision dans une opérande de destination.

Algorithme

dest(63..32) ← *source*(63..32)
dest(31..0) ← *source*(31..0)

Mnémonique

Instruction	Opcode	Description
MOVLPS *xmm,m64*	0Fh 12h /r	Cette instruction permet de copier le contenu du bas de deux paquets de valeurs réel de simple précision dans une opérande de destination.
MOVLPS *m64,xmm*	0Fh 13h /r	Cette instruction permet de copier le contenu du bas de deux paquets de valeurs réel de simple précision dans une opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 763 à 765.

Assembleur 80x86 — MOVMSKPD

INTEL Pentium 4 (SSE2)+

Extract Packed Double-Precision Floating-Point Sign Mask

Syntaxe

MOVMSKPD *registres32*, *registrexmm*

Description

Cette instruction permet de copier les bits de signes de deux paquets de valeurs réels de double précision d'un registre *XMM* dans les 2 bits les plus bas d'un registre 32 bits. Les autres bits du registres 32 bits sont fixés à 0.

Algorithme

registres32(0) ← *registrexmm*(63)
registres32(1) ← *registrexmm*(127)
registres32(3..2) ← 00b
registres32(31..4) ← 0000000h

Mnémonique

Instruction	Opcode	Description
MOVMSKPD *reg32*, *xmm*	66h 0Fh 50h /r	Copie les bits de signe 127 et 63 d'un registre XMM dans un registre 32 bits spécifié.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Les ensembles d'instructions SSE2 ne sont pas supporté, comme indiqué par le bit 26 du registre EDX de la fonction 0000_0001h de l'instruction *CPUID*.
	X	X	X	Le bit de support du système d'exploitation FXSAVE et FXRSTOR (OSFXSR) du registre CR4 sont effacés à 0.
	X	X	X	Le bit d'émulation (EM) du registre CR0 est fixé à 1.
#NM(Périphérique non disponible)	X	X	X	Le bit d'échangeur de tâche (TS) du registre CR0 est fixé à 1.

Voir également

Instruction assembleur 80x86 - Instruction MOVMSKPS

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 766 à 767.

Assembleur 80x86 — MOVMSKPS

INTEL Pentium 3 (SSE)+

Extract Packed Single-Precision Floating-Point Sign Mask

Syntaxe

MOVMSKPS *registres32*, *registrexmm*

Description

Cette instruction permet de copier les bits de signes de quatre paquets de valeurs réels de simple précision d'un registre *XMM* dans les 4 bits les plus bas d'un registre 32 bits. Les autres bits du registres 32 bits sont fixés à 0.

Algorithme

registres32(0) ← *registrexmm*(31)
registres32(1) ← *registrexmm*(63)
registres32(1) ← *registrexmm*(95)
registres32(1) ← *registrexmm*(127)
registres32(31..4) ← 000000h

Mnémonique

Instruction	Opcode	Description
MOVMSKPS *reg32*, *xmm*	0Fh 50h /r	Copie les bits de signe 127, 95, 63 et 31 du registre *XMM* au registre 32 bits spécifié.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Les ensembles d'instructions SSE2 ne sont pas supporté, comme indiqué par le bit 26 du registre EDX de la fonction 0000_0001h de l'instruction *CPUID*.
	X	X	X	Le bit de support du système d'exploitation FXSAVE et FXRSTOR (OSFXSR) du registre CR4 sont effacés à 0.
	X	X	X	Le bit d'émulation (EM) du registre CR0 est fixé à 1.
#NM(Périphérique non disponible)	X	X	X	Le bit d'échangeur de tâche (TS) du registre CR0 est fixé à 1.

Voir également

Instruction assembleur 80x86 - Instruction MOVMSKPD

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 768 à 768.

Assembleur 80x86 MOVNTI

INTEL Pentium 4 (SSE2)+

Move Non-Temporal Doubleword or Quadword

Syntaxe

MOVNTI *memoire*, *registre*

Description

Cette instruction permet de copier une valeur 32 ou 64 bits dans un emplacement mémoire afin de minimiser la pollution du cache dans un processus léger.

Algorithme

memoire ← *registre*

Mnémonique

Instruction	Opcode	Description
MOVNTI *mem32*, *reg32*	0Fh C3h /r	Entrepose une valeur de registre 32 bits spécifié dans un emplacement mémoire en minimisant la pollution du système de cache.
MOVNTI *mem64*, *reg64*	0Fh C3h /r	Entrepose une valeur de registre 64 bits spécifié dans un emplacement mémoire en minimisant la pollution du système de cache.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Les ensembles d'instructions SSE2 ne sont pas supporté, comme indiqué par le bit 26 du registre EDX de la fonction 0000_0001h de l'instruction *CPUID*.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment

				de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVNTDQ
Instruction assembleur 80x86 - Instruction MOVNTPD
Instruction assembleur 80x86 - Instruction MOVNTPS
Instruction assembleur 80x86 - Instruction MOVNTQ

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 776 à 778.

Assembleur 80x86 MOVNTDQ

SSE4.1+ *Store Double Quadword Using Non-Temporal Hint*

Syntaxe

MOVNTDQ *destination*, *source*

Description

Cette instruction permet de copier le contenu de double quadruple mots d'un opérande source vers un opérande de destination sans utiliser la méthode temporel pour minimiser la pollution du cache.

Algorithme

destination ← *source*

Mnémonique

Instruction	Opcode	Description
MOVNTDQ *m128*, *xmm*	66h 0Fh E7h /r	Cette instruction permet de copier le contenu de double quadruple mots d'un opérande source vers un opérande de destination sans utiliser la méthode temporel pour minimiser la pollution du cache.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 773 à 775.

Assembleur 80x86 MOVNTDQA

SSE4.1+

Load Double Quadword Non-Temporal Aligned Hint

Syntaxe

MOVNTDQA *dest,source*

Description

Cette instruction permet de copier le contenu aligné de double quadruple mots d'un opérande source vers un opérande de destination sans utiliser la méthode temporel pour minimiser la pollution du cache.

Mnémonique

Instruction	Opcode	Description
MOVNTDQA *xmm1, m128*	66h 0Fh 38h 2Ah /r	Cette instruction permet de copier le contenu aligné de double quadruple mots d'un opérande source vers un opérande de destination sans utiliser la méthode temporel pour minimiser la pollution du cache.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 770 à 772.

Assembleur 80x86 — MOVNTPD

SSE4.1+ — *Store Packed Double-Precision Floating-Point Values Using Non-Temporal Hint*

Syntaxe

MOVNTPD *destination*, *source*

Description

Cette instruction permet de copier le contenu de valeur réel de double précision d'un opérande source vers un opérande de destination sans utiliser la méthode temporel pour minimiser la pollution du cache.

Algorithme

destination ← *source*

Mnémonique

Instruction	Opcode	Description
MOVNTPD *m128*, *xmm*	66h 0Fh 2Bh /r	Cette instruction permet de copier le contenu de valeur réel de double précision d'un opérande source vers un opérande de destination sans utiliser la méthode temporel pour minimiser la pollution du cache.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 778 à 780.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MOVNTPS

Move Aligned Four Packed Single-FP Non Temporal

Syntaxe

MOVNTPS *dest,source*

Description

Cette instruction permet de copier le contenu de 4 paquets alignés de valeurs réel de simple précision sans utiliser la méthode temporel pour minimiser la pollution du cache.

Algorithme

dest ← source

Mnémonique

Instruction	Opcode	Description
MOVNTPS *m128,xmm*	0Fh 2Bh /r	Cette instruction permet de copier le contenu de 4 paquets alignés de valeurs réel de simple précision sans utiliser la méthode temporel pour minimiser la pollution du cache.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 781 à 783.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MOVNTQ

Move Quadword Non-Temporal

Syntaxe

MOVNTQ *dest,source*

Description

Cette instruction permet de copier une valeur 64 bits sans utiliser la méthode temporel pour minimiser la pollution du cache.

Algorithme

dest ← source

Mnémonique

Instruction	Opcode	Description
MOVNTQ *m64,mm*	0Fh E7h /r	Cette instruction permet de copier une valeur 64 bits sans utiliser la méthode temporel pour minimiser la pollution du cache.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 741 à 786.

Assembleur 80x86 — MOVQ

INTEL Pentium MMX+ — *Move Quadword*

Syntaxe

MOVQ *destination*, *source*

Description

Cette instruction permet de copier un quadruple mot d'une opérande source vers une opérande destination dans le cas des registres *XMM*.

Algorithme

destination ← *source*

Mnémonique

Instruction	Opcode	Description
MOVQ *mm*, *mm/m64*	0Fh 6Fh /r	Cette instruction permet de copier un quadruple mot d'une opérande source vers une opérande destination dans le cas des registres *XMM*.
MOVQ *mm/m64*, *mm*	0Fh 7Fh /r	Cette instruction permet de copier un quadruple mot d'une opérande source vers une opérande destination dans le cas des registres *XMM*.
MOVQ *xmm1*, *xmm2/m64*	F3h 0Fh 7Eh	Cette instruction permet de copier un quadruple mot d'une opérande

		source vers une opérande destination dans le cas des registres *XMM*.
MOVQ *xmm2/m64, xmm1*	66h 0Fh D6h	Cette instruction permet de copier un quadruple mot d'une opérande source vers une opérande destination dans le cas des registres *XMM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 735 à 737.

Assembleur 80x86 — MOVQ2DQ

INTEL Pentium 4 (SSE2)+ — *Move Quadword from MMX Technology to XMM Register*

Syntaxe

MOVQ2DQ *destination*, *source*

Description

Cette instruction permet de copier un quadruple mot d'un opérande source vers la partie basse d'un opérande destination dans le cas des registres *XMM*.

Algorithme

destination(0..63) ← *source*(0..63)
destination(64..127) ← 0

Mnémonique

Instruction	Opcode	Description
MOVQ2DQ *xmm*, *mm*	F3h 0Fh D6h	Cette instruction permet de copier un quadruple mot d'un opérande source vers la partie basse d'un opérande destination dans le cas des registres *XMM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 789 à 791.

Assembleur 80x86	**MOVS**
INTEL 8088+	*MOVe String*

Syntaxe

MOVS *opérandecible*,*opérandesource*

Description

Cette instruction permet de copier un élément de l'adresse DS:SI dans l'adresse ES:DI et incrémente/décrémente les registres DI et SI en fonction de la taille de l'opérande source et de l'état du drapeau de direction.

Algorithme

```
opérandecible ← opérandesource
SI opérande est un octet ALORS
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 1
    (E)DI ← (E)DI + 1
  SINON
    (E)SI ← (E)SI - 1
    (E)DI ← (E)DI - 1
  FIN SI
SINON SI opérande est un mot ALORS
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 2
    (E)DI ← (E)DI + 2
  SINON
    (E)SI ← (E)SI - 2
    (E)DI ← (E)DI - 2
  FIN SI
SINON SI opérande est un double mot ALORS
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 4
    (E)DI ← (E)DI + 4
```

```
  SINON
    (E)SI ← (E)SI - 4
    (E)DI ← (E)DI - 4
  FIN SI
SINON
  SI DF = 0 ALORS
    (E)SI ← (E)SI + 8
    (E)DI ← (E)DI + 8
  SINON
    (E)SI ← (E)SI - 8
    (E)DI ← (E)DI - 8
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MOVS *mem8*, *mem8*	A4h	Copie l'octet de DS:(R)SI à ES:(R)DI, et alors incrémente ou décrément le registre (R)SI et (R)DI.
MOVS *mem16*, *mem16*	A5h	Copie le mot de DS:(R)SI à ES:(R)DI, et alors incrémente ou décrément le registre (R)SI et (R)DI.
MOVS *mem32*, *mem32*	A5h	Copie le double mot de DS:(R)SI à ES:(R)DI, et alors incrémente ou décrément le registre (R)SI et (R)DI.
MOVS *mem64*, *mem64*	A5h	Copie le double mot de DS:(R)SI à ES:(R)DI, et alors incrémente ou décrément le registre (R)SI et (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie		X	X	Un désalignement

l'alignement)				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOV
Instruction assembleur 80x86 - Instruction LODS
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 821
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 411
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 168.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 792 à 796.

Assembleur 80x86	**MOVSB**
INTEL 8088+	*MOVe String Byte*

Syntaxe

```
MOVSB
```

Description

Cette instruction permet de copier un octet de l'adresse DS:SI dans l'adresse ES:DI et incrémente/décrémente les registres DI et SI de 1 en fonction de l'état du drapeau de direction.

Algorithme

```
opérandecible ← opérandesource
SI DF = 0 ALORS
  (E)SI ← (E)SI + 1
  (E)DI ← (E)DI + 1
SINON
  (E)SI ← (E)SI - 1
  (E)DI ← (E)DI - 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MOVSB	A4h	Copie l'octet de DS:(R)SI à ES:(R)DI, et alors incrémente ou décrément le registre (R)SI et (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie		X	X	Un désalignement

l'alignement)				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOV
Instruction assembleur 80x86 - Instruction LODS
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 821
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 168.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 792 à 796.

Assembleur 80x86	**MOVSD**
INTEL 80386+	*MOVe String Double word*

Syntaxe

MOVSD

Description

Cette instruction permet de copier un double mot de l'adresse DS:SI dans l'adresse ES:DI et incrémente/décrémente les registres DI et SI de 4 en fonction de l'état du drapeau de direction.

Algorithme

opérandecible ← *opérandesource*
SI DF = 0 **ALORS**
  (E)SI ← (E)SI + 4
  (E)DI ← (E)DI + 4
SINON
  (E)SI ← (E)SI - 4
  (E)DI ← (E)DI - 4
FIN SI

Mnémonique

Instruction	Opcode	Description
MOVSD	A5h	Copie le double mot de DS:(R)SI à ES:(R)DI, et alors incrémente ou décrément le registre (R)SI et (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie		X	X	Un désalignement

l'alignement)				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOV
Instruction assembleur 80x86 - Instruction LODS
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 822
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 168.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 792 à 796.

Assembleur 80x86 — MOVSD

INTEL Pentium 4 (SSE2)+ — *Move Scalar Double-Precision Floating-Point Value*

Syntaxe

MOVSD *dest,source*

Description

Cette instruction permet de copier un scalaire de valeur de double précision d'un opérande source vers un opérande destination.

Algorithme

dest(63..0) ← *source*(63..0)

Mnémonique

Instruction	Opcode	Description
MOVSD *xmm1,xmm2/m64*	F2h 0Fh 10h /r	Cette instruction permet de copier un scalaire de valeur de double précision d'un opérande source vers un opérande destination.
MOVSD *xmm2/m64,xmm1*	F2h 0Fh 11h /r	Cette instruction permet de copier un scalaire de valeur de double précision d'un opérande source vers un opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 797 à 799.

Assembleur 80x86 — MOVSHDUP

SSE4.1+ — *Move Packed Single-Float Point High and Duplicate*

Syntaxe

MOVSHDUP *destination*, *source*

Description

Cette instruction permet de copier la partie du haut de 32 bits d'une valeur 64 bits dans sa partie basse et haute d'une valeur réel de simple précision de 64 bits contenu dans un paquet de 128 bits.

Algorithme

destination(0..31) ← *source*(32..63)
destination(32..63) ← *source*(32..63)
destination(64..95) ← *source*(96..127)
destination(96..127) ← *source*(96..127)

Mnémonique

Instruction	Opcode	Description
MOVSHDUP *xmm1*, *xmm2/m128*	F3h 0Fh 16h /r	Cette instruction permet de copier la partie du haut de 32 bits d'une valeur 64 bits dans sa partie basse et haute d'une valeur réel de simple précision de 64 bits contenu dans un paquet de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 800 à 802.

Assembleur 80x86 — MOVSLDUP

SSE4.1+ — *Move Packed Single-Float Point Low and Duplicate*

Syntaxe

MOVSLDUP *destination*, *source*

Description

Cette instruction permet de copier la partie du basse de 32 bits d'une valeur 64 bits dans sa partie basse et haute d'une valeur réel de simple précision de 64 bits contenu dans un paquet de 128 bits.

Algorithme

destination(0..31) ← *source*(0..31)
destination(32..63) ← *source*(0..31)
destination(64..95) ← *source*(64..95)
destination(96..127) ← *source*(64..95)

Mnémonique

Instruction	Opcode	Description
MOVSLDUP *xmm1*, *xmm2/m128*	F3h 0Fh 12h /r	Cette instruction permet de copier la partie du basse de 32 bits d'une valeur 64 bits dans sa partie basse et haute d'une valeur réel de simple précision de 64 bits contenu dans un paquet de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 803 à 805.

Assembleur 80x86 — MOVSQ

x86-64+ — *Move String Quadword*

Syntaxe

```
MOVSQ
```

Description

Cette instruction permet de copier un quadruple mot de l'adresse DS:(R)SI dans l'adresse ES:(R)DI et incrémente/décrémente les registres (R)DI et (R)SI de 8 en fonction de l'état du drapeau de direction.

Algorithme

```
opérandecible ← opérandesource
SI DF = 0 ALORS
  (E)SI ← (E)SI + 8
  (E)DI ← (E)DI + 8
SINON
  (E)SI ← (E)SI - 8
  (E)DI ← (E)DI - 8
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MOVSQ	A5h	Copie le double mot de DS:(R)SI à ES:(R)DI, et alors incrémente ou décrément le registre (R)SI et (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOV
Instruction assembleur 80x86 - Instruction LODS
Instruction assembleur 80x86 - Instruction STOS

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 792 à 796.

Assembleur 80x86

MOVSS

INTEL Pentium MMX+

Move Scalar Single-Precision Floating-Point Values

Syntaxe

MOVSS *destination*, *source*

Description

Cette instruction permet de copier une valeur réel de simple précision d'une opérande source vers une opérande de destination.

Algorithme

destination(31..0) ← *source*(31..0)

Mnémonique

Instruction	Opcode	Description
MOVSS *xmm1,xmm2/m32*	F3h 0Fh 10h /r	Cette instruction permet de copier une valeur réel de simple précision d'une opérande source vers une opérande de destination.
MOVSS *xmm2/m32,xmm1*	F3h 0Fh 11h /r	Cette instruction permet de copier une valeur réel de simple précision d'une opérande source vers une opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 806 à 808.

Assembleur 80x86	**MOVSW**
INTEL 8088+	*MOV String Word*

Syntaxe

MOVSW

Description

Cette instruction permet de copier un mot de l'adresse DS:SI dans l'adresse ES:DI et incrémente/décrémente les registres DI et SI de 2 en fonction de l'état du drapeau de direction.

Algorithme

opérandecible ← *opérandesource*
SI DF = 0 **ALORS**
  (E)SI ← (E)SI + 2
  (E)DI ← (E)DI + 2
SINON
  (E)SI ← (E)SI - 2
  (E)DI ← (E)DI - 2
FIN SI

Mnémonique

Instruction	Opcode	Description
MOVSW	A5h	Copie le mot de DS:(R)SI à ES:(R)DI, et alors incrémente ou décrément le registre (R)SI et (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie		X	X	Un désalignement

l'alignement)				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOV
Instruction assembleur 80x86 - Instruction LODS
Instruction assembleur 80x86 - Instruction STOS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 822
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 168.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 792 à 796.

Assembleur 80x86 — MOVSX

INTEL 80386+ — *Move with Sign-Extension*

Syntaxe

MOVSX *operandedestination*, *operandesource*

Description

Cette instruction permet de copier un registre de taille inférieur dans un registre de plus grande taille en remplissant les bits supplémentaires par des 1.

Algorithme

```
SI taille de l'opérande source = 8 bits ALORS
  SI taille de l'opérande destinataire = 16 bits ALORS
    operandedestination(15..8) ← FFh
    operandedestination(7..0) ← operandesource
  SINON
    operandedestination(31..8) ← FFFFFFh
    operandedestination(7..0) ← operandesource
  FIN SI
SINON
  operandedestination(31..16) ← FFFFh
  operandedestination(15..0) ← operandesource
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MOVSX *reg16*, *reg/mem8*	0Fh BEh /r	Copie le contenu du registre ou d'un emplacement mémoire 8 bits dans un

		registre 16 bits avec un extension des signes.
MOVSX *reg32*, *reg/mem8*	0Fh BEh /r	Copie le contenu du registre ou d'un emplacement mémoire 8 bits dans un registre 32 bits avec un extension des signes.
MOVSX *reg64*, *reg/mem8*	0Fh BEh /r	Copie le contenu du registre ou d'un emplacement mémoire 8 bits dans un registre 64 bits avec un extension des signes.
MOVSX *reg32*, *reg/mem16*	0Fh BFh /r	Copie le contenu du registre ou d'un emplacement mémoire 16 bits dans un registre 32 bits avec un extension des signes.
MOVSX *reg64*, *reg/mem16*	0Fh BFh /r	Copie le contenu du registre ou d'un emplacement mémoire 16 bits dans un registre 64 bits avec un extension des signes.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVSXD
Instruction assembleur 80x86 - Instruction MOVZX

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 809 à 811.

Assembleur 80x86 MOVSXD

x86-64+

Move with Sign-Extend Doubleword

Syntaxe

MOVSXD *registres64*, *source*

Description

Cette instruction permet de copier un registre de taille inférieur dans un registre 64 bits en remplissant les bits supplémentaires par des 1.

Algorithme

SI taille de l'opérande source = 8 bits **ALORS**
  operandedestination(63..8) ← FFFFFFFFFFFFFFh
  operandedestination(7..0) ← *operandesource*
SINON SI taille de l'opérande source = 16 bits **ALORS**
  operandedestination(63..16) ← FFFFFFFFFFFFh
  operandedestination(15..0) ← *operandesource*
SINON SI taille de l'opérande source = 32 bits **ALORS**
  operandedestination(63..32) ← FFFFFFFFh
  operandedestination(31..0) ← *operandesource*
FIN SI

Mnémonique

Instruction	Opcode	Description
MOVSXD *reg64*, *reg/mem32*	63h /r	Copie le contenu du registre ou d'un emplacement mémoire 32 bits dans un registre 64 bits avec un extension

		des signes.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)			X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)			X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)			X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVSX
Instruction assembleur 80x86 - Instruction MOVZX

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 809 à 811.

Assembleur 80x86 — MOVUPD

INTEL Pentium 4 (SSE2)+ — *Move Unaligned Packed Double-Precision Floating-Point Values*

Syntaxe

MOVUPD *destination*, *source*

Description

Cette instruction permet de copier le contenu de 2 paquets désalignés de valeurs réel de double précision.

Algorithme

destination ← *source*

Mnémonique

Instruction	Opcode	Description
MOVUPD *xmm1*, *xmm2/m128*	66h 0Fh 10h /r	Cette instruction permet de copier le contenu de 2 paquets désalignés de valeurs réel de double précision.
MOVUPD *xmm2/m128*, *xmm*	66h 0Fh 11h /r	Cette instruction permet de copier le contenu de 2 paquets désalignés de valeurs réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 812 à 814.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MOVUPS

Move Unaligned Packed Double-Precision Floating-Point Values

Syntaxe

MOVUPS *dest,source*

Description

Cette instruction permet de copier le contenu de 4 paquets désalignés de valeurs réel de simple précision (4 x 32 bits).

Algorithme

dest ← source

Mnémonique

Instruction	Opcode	Description
MOVUPS *xmm1,xmm2/m128*	0Fh 10h /r	Cette instruction permet de copier le contenu de 4 paquets désalignés de valeurs réel de simple précision (4 x 32 bits).
MOVUPS *xmm2/m128,xmm1*	0Fh 11h /r	Cette instruction permet de copier le contenu de 4 paquets désalignés de valeurs réel de simple précision (4 x 32 bits).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 815 à 817.

Assembleur 80x86 — MOVZX

INTEL 80386+ — *Move with Zero-Extend*

Syntaxe

MOVZX *operandedestination*, *operandesource*

Description

Cette instruction permet de copier un registre de taille inférieur dans un registre de plus grande taille en remplissant les bits supplémentaires par des 0.

Algorithme

```
SI taille de l'opérande source = 8 bits ALORS
  SI taille de l'opérande destinataire = 16 bits ALORS
    operandedestination(15..8) ← 00h
    operandedestination(7..0) ← operandesource
  SINON
    operandedestination(31..8) ← 000000h
    operandedestination(7..0) ← operandesource
  FIN SI
SINON
  operandedestination(31..16) ← 0000h
  operandedestination(15..0) ← operandesource
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MOVZX *reg16*, *reg/mem8*	0Fh B6h /r	Copie le contenu d'une opérande de registre ou d'un emplacement

		mémoire 8 bits dans un registre 16 bits avec un extension des zéros.
MOVZX *reg32*, *reg/mem8*	0Fh B6h /r	Copie le contenu d'une opérande de registre ou d'un emplacement mémoire 8 bits dans un registre 32 bits avec un extension des zéros.
MOVZX *reg64*, *reg/mem8*	0Fh B6h /r	Copie le contenu d'une opérande de registre ou d'un emplacement mémoire 8 bits dans un registre 64 bits avec un extension des zéros.
MOVZX *reg32*, *reg/mem16*	0Fh B7h /r	Copie le contenu d'une opérande de registre ou d'un emplacement mémoire 16 bits dans un registre 32 bits avec un extension des zéros.
MOVZX *reg64*, *reg/mem16*	0Fh B7h /r	Copie le contenu d'une opérande de registre ou d'un emplacement mémoire 16 bits dans un registre 64 bits avec un extension des zéros.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction MOVSXD
Instruction assembleur 80x86 - Instruction MOVSX

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 818 à 819.

Assembleur 80x86 — MPSADBW

SSE4.1 — *Compute Multiple Packed Sums of Absolute Difference*

Syntaxe

MPSADBW *dest*,*source*,*immediat*

Description

Cette instruction permet d'effectuer le calcul de la somme de la différence absolue (*SAD*) d'une paire d'octets d'un groupe de 4 octets paires et produit 8 résultats *SAD* entreposé dans 8 entier contenu dans l'opérande de destination.

Algorithme

```
SRC_OFFSET ← immediat(1..0) x 32
DEST_OFFSET ← immediat(2) x 32
DEST_BYTE0 ← dest[DEST_OFFSET+7..DEST_OFFSET]
DEST_BYTE1 ← dest[DEST_OFFSET+15..DEST_OFFSET+8]
DEST_BYTE2 ← dest[DEST_OFFSET+23..DEST_OFFSET+16]
DEST_BYTE3 ← dest[DEST_OFFSET+31..DEST_OFFSET+24]
DEST_BYTE4 ← dest[DEST_OFFSET+39..DEST_OFFSET+32]
DEST_BYTE5 ← dest[DEST_OFFSET+47..DEST_OFFSET+40]
DEST_BYTE6 ← dest[DEST_OFFSET+55..DEST_OFFSET+48]
DEST_BYTE7 ← dest[DEST_OFFSET+63..DEST_OFFSET+56]
DEST_BYTE8 ← dest[DEST_OFFSET+71..DEST_OFFSET+64]
DEST_BYTE9 ← dest[DEST_OFFSET+79..DEST_OFFSET+72]
DEST_BYTE10 ← dest[DEST_OFFSET+87..DEST_OFFSET+80]
SRC_BYTE0 ← source[SRC_OFFSET+7..SRC_OFFSET]
SRC_BYTE1 ← source[SRC_OFFSET+15..SRC_OFFSET+8]
SRC_BYTE2 ← source[SRC_OFFSET+23..SRC_OFFSET+16]
SRC_BYTE3 ← source[SRC_OFFSET+31..SRC_OFFSET+24]

TEMP0 ← | DEST_BYTE0 - SRC_BYTE0 |
```

```
TEMP1 ← | DEST_BYTE1 - SRC_BYTE1 |
TEMP2 ← | DEST_BYTE2 - SRC_BYTE2 |
TEMP3 ← | DEST_BYTE3 - SRC_BYTE3 |
dest(15..0) ← TEMP0 + TEMP1 + TEMP2 + TEMP3
TEMP0 ← | DEST_BYTE1 - SRC_BYTE0 |
TEMP1 ← | DEST_BYTE2 - SRC_BYTE1 |
TEMP2 ← | DEST_BYTE3 - SRC_BYTE2 |
TEMP3 ← | DEST_BYTE4 - SRC_BYTE3 |
dest(31..16) ← TEMP0 + TEMP1 + TEMP2 + TEMP3
TEMP0 ← | DEST_BYTE2 - SRC_BYTE0 |
TEMP1 ← | DEST_BYTE3 - SRC_BYTE1 |
TEMP2 ← | DEST_BYTE4 - SRC_BYTE2 |
TEMP3 ← | DEST_BYTE5 - SRC_BYTE3 |
dest(47..32) ← TEMP0 + TEMP1 + TEMP2 + TEMP3
TEMP0 ← | DEST_BYTE3 - SRC_BYTE0 |
TEMP1 ← | DEST_BYTE4 - SRC_BYTE1 |
TEMP2 ← | DEST_BYTE5 - SRC_BYTE2 |
TEMP3 ← | DEST_BYTE6 - SRC_BYTE3 |
dest(63..48) ← TEMP0 + TEMP1 + TEMP2 + TEMP3
TEMP0 ← | DEST_BYTE4 - SRC_BYTE0 |
TEMP1 ← | DEST_BYTE5 - SRC_BYTE1 |
TEMP2 ← | DEST_BYTE6 - SRC_BYTE2 |
TEMP3 ← | DEST_BYTE7 - SRC_BYTE3 |
dest(79..64) ← TEMP0 + TEMP1 + TEMP2 + TEMP3
TEMP0 ← | DEST_BYTE5 - SRC_BYTE0 |
TEMP1 ← | DEST_BYTE6 - SRC_BYTE1 |
TEMP2 ← | DEST_BYTE7 - SRC_BYTE2 |
TEMP3 ← | DEST_BYTE8 - SRC_BYTE3 |
dest(95..80) ← TEMP0 + TEMP1 + TEMP2 + TEMP3
TEMP0 ← | DEST_BYTE6 - SRC_BYTE0 |
TEMP1 ← | DEST_BYTE7 - SRC_BYTE1 |
TEMP2 ← | DEST_BYTE8 - SRC_BYTE2 |
TEMP3 ← | DEST_BYTE9 - SRC_BYTE3 |
dest(111..96) ← TEMP0 + TEMP1 + TEMP2 + TEMP3
TEMP0 ← | DEST_BYTE7 - SRC_BYTE0 |
TEMP1 ← | DEST_BYTE8 - SRC_BYTE1 |
TEMP2 ← | DEST_BYTE9 - SRC_BYTE2 |
TEMP3 ← | DEST_BYTE10 - SRC_BYTE3 |
dest(127..112) ← TEMP0 + TEMP1 + TEMP2 + TEMP3
```

Mnémonique

Instruction	Opcode	Description
MPSADBW *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 42h /r *ib*	Cette instruction permet d'effectuer le calcul de la somme de la différence absolue (*SAD*) d'une paire d'octets d'un groupe de 4 octets paires et produit 8 résultats *SAD* entreposé dans 8 entier contenu dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 820 à 823.

Assembleur 80x86	**MUL**
INTEL 8088+	*Multiplication*

Syntaxe

MUL *Opérande*

Description

Cette instruction permet d'effectuer une multiplication non-signée (nombre naturel). Le multiplicateur est implicite; il est ajuster en fonction de la taille de la base. Le produit est toujours plus grand que le multiplicateur. Le type de multiplication détermine quel registre l'instruction utilisera:

Taille	Base	Multiplicateur	Résultat
Octet	AL	*Opérande*	AX
Mot	AX	*Opérande*	DX:AX
Double mot	EAX	*Opérande*	EDX:EAX

Algorithme

```
SI Nombre d'opérande = 1 ALORS
  SI Taille de l'opérande en bits = 8 ALORS
    AX ← AL x Opérande
    SI (AH = 00h) OU (AH = FFh) ALORS
      CF ← 0
      OF ← 0
    SINON
      CF ← 1
      OF ← 1
    FIN SI
  SINON SI Taille de l'opérande en bits = 16 ALORS
```

```
      DX:AX ← AX x Opérande
      SI (DX = 0000h) OU (DX = FFFFh) ALORS
        CF ← 0
        OF ← 0
      SINON
        CF ← 1
        OF ← 1
      FIN SI
    SINON
      EDX:EAX ← EAX x Opérande
      SI ((EDX = 00000000h) OU (EDX = FFFFFFFFh)) ALORS
        CF ← 0
        OF ← 0
      SINON
        CF ← 1
        OF ← 1
      FIN SI
    FIN SI
  SINON SI Nombre d'opérande = 2 ALORS
    temp ← dest x src
    dest ← dest x src
    SI temp = dest ALORS
      CF ← 1
      OF ← 1
    SINON
      CF ← 0
      OF ← 0
    FIN SI
  SINON
    temp ← dest x src
    dest ← dest x src
    SI temp = dest ALORS
      CF ← 1
      OF ← 1
    SINON
      CF ← 0
      OF ← 0
    FIN SI
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
MUL *reg/mem8*	F6h /4	Multiple l'opérande mémoire ou registre 8 bits par le contenu du registre AL et entrepose le résultat dans le registre AX.
MUL *reg/mem16*	F7h /4	Multiple l'opérande mémoire ou registre 16 bits par le contenu du registre AX et entrepose le résultat dans le registre DX:AX.
MUL *reg/mem32*	F7h /4	Multiple l'opérande mémoire ou registre 32 bits par le contenu du registre EAX et entrepose le résultat dans le registre EDX:EAX.
MUL *reg/mem64*	F7h /4	Multiple l'opérande mémoire ou registre 64 bits par le contenu du registre RAX et entrepose le résultat dans le registre RDX:RAX.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection	X	X	X	Une adresse mémoire

général)				dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction DIV
Instruction assembleur 80x86 - Instruction IMUL
Langage de programmation - Assembleur et Pascal - Opération Mathématique

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 822
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 411 à 412
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 173.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 824 à 826.

Assembleur 80x86 — MULPD

INTEL Pentium 4 (SSE2)+ — *Multiply Packed Double-Precision Floating-Point Values*

Syntaxe

MULPD *destination*, *source*

Description

Cette instruction permet d'effectuer la multiplication de chacune des paires de valeur de double précision de l'opérande source et l'opérande de destination.

Algorithme

destination(0..63) ← *destination*(0..63) x *source*(0..63)
destination(64..127) ← *destination*(64..127) x *source*(64..127)

Mnémonique

Instruction	Opcode	Description
MULPD *xmm1*, *xmm2/m128*	66h 0Fh 59h /r	Cette instruction permet d'effectuer la multiplication de chacune des paires de valeur de double précision de l'opérande source et l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 827 à 829.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MULPS

Packed Single-Precision Floating-Point Multiply

Syntaxe

MULPS *dest,source*

Description

Cette instruction permet d'effectuer la multiplication de chacune des paires de valeur de l'opérande source et l'opérande de destination.

Algorithme

dest(31..0) ← *dest*(31..0) x *source*(31..0) *dest*(63..32) ← *dest*(63..32) x *source*(63..32) *dest*(95..64) ← *dest*(95..64) x *source*(95..64) *dest*(127..96) ← *dest*(127..96) x *source*(127..96)

Mnémonique

Instruction	Opcode	Description
MULPS *xmm1,xmm2/m128*	0Fh 59h /r	Cette instruction permet d'effectuer la multiplication de chacune des paires de valeur de l'opérande source et l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 830 à 832.

Assembleur 80x86 — MULSD

INTEL Pentium 4 (SSE2)+ — *Multiply Scalar Double-Precision Floating-Point Values*

Syntaxe

MULSD *destination*, *source*

Description

Cette instruction permet d'effectuer la multiplication scalaire de valeurs réel de double précision de l'opérande source et l'opérande de destination.

Algorithme

destination(0..63) ← *destination*(0..63) x *source*(0..63)

Mnémonique

Instruction	Opcode	Description
MULSD *xmm1*, *xmm2/m64*	F2h 0Fh 59h /r	Cette instruction permet d'effectuer la multiplication scalaire de valeurs réel de double précision de l'opérande source et l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 833 à 835.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

MULSS

Scalar Single-FP Multiply

Syntaxe

MULSS *dest,source*

Description

Cette instruction permet d'effectuer la multiplication scalaire de l'opérande source et l'opérande de destination.

Algorithme

dest(31..0) ← *dest*(31..0) x *source*(31..0)

Mnémonique

Instruction	Opcode	Description
MULSS *xmm1, xmm2/m32*	F3h 0Fh 59h /r	Cette instruction permet d'effectuer la multiplication scalaire de l'opérande source et l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 836 à 838.

Assembleur 80x86 MWAIT

INTEL Pentium 4+

Monitor Wait

Syntaxe

MWAIT
MWAIT EAX, ECX

Description

Cette instruction permet d'indiquer au microprocesseur que l'état de l'alimentation de la ligne de cache est en attente d'écriture dans la plage d'adresse, mettant fin à la plupart des activités dans le noyau en le faisant.

Mnémonique

Instruction	Opcode	Description
MWAIT	0Fh 01h C9h	Cette instruction permet d'indiquer au microprocesseur que l'état de l'alimentation de la ligne de cache est en attente d'écriture dans la plage d'adresse, mettant fin à la plupart des activités dans le noyau en le faisant.

Voir également

Instruction assembleur 80x86 - Instruction MONITOR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2A: Instruction

Set Reference, A-M, Edition Intel, Mars 2010, Publication No. 253666-034US, page 839 à 842.

Assembleur 80x86	**NEG**
INTEL 8088+	*Negation*

Syntaxe

NEG *registre*
NEG *mémoire*

Description

Cette instruction permet d'effectuer le complément à 2 d'une opérande.

Algorithme

```
SI DEST = 0 ALORS
  CF ← 0
SINON
  CF ← 1
FIN SI
DEST ← - (DEST)
```

Mnémonique

Instruction	Opcode	Description
NEG *reg/mem8*	F6h /3	Effectue une négation de compléments de 2 dans une opérande de registre ou mémoire 8 bits.
NEG *reg/mem16*	F7h /3	Effectue une négation de compléments de 2 dans une opérande de registre ou mémoire 16

		bits.
NEG *reg/mem32*	F7h /3	Effectue une négation de compléments de 2 dans une opérande de registre ou mémoire 32 bits.
NEG *reg/mem64*	F7h /3	Effectue une négation de compléments de 2 dans une opérande de registre ou mémoire 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable

			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction AND
Instruction assembleur 80x86 - Instruction NEG
Instruction assembleur 80x86 - Instruction OR
Instruction assembleur 80x86 - Instruction XOR

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 823
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 412
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 175.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 10 à 12.

Assembleur 80x86	NOP
INTEL 8088+	*No OPeration*

Syntaxe

```
NOP
```

Description

Cette instruction ne fait rien. Elle est utilisé lors de déboguage ou pour créer des délais d'attente artificielle sans affectation des registres du processeurs.

Mnémonique

Instruction	Opcode	Description
NOP	90h	Efface aucune opération

Exception

Aucune

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 823
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 412
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 177.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 13 à 14.

Assembleur 80x86	**NOT**
INTEL 8088+	

Syntaxe

NOT *registre*
NOT *mémoire*

Description

Cette instruction permet d'inverser la valeur de chacun des bits d'une opérande.

Algorithme

dest ← (¬ *dest*)

Mnémonique

Instruction	Opcode	Description
NOT *reg/mem8*	F6h /2	Compléments les bits d'une opérande de registre ou mémoire 8 bits.
NOT *reg/mem16*	F7h /2	Compléments les bits d'une opérande de registre ou mémoire 16 bits.
NOT *reg/mem32*	F7h /2	Compléments les bits d'une opérande de registre ou mémoire 32 bits.
NOT *reg/mem64*	F7h /2	Compléments les bits d'une opérande de registre ou mémoire 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction AND
Instruction assembleur 80x86 - Instruction NEG
Instruction assembleur 80x86 - Instruction OR
Instruction assembleur 80x86 - Instruction XOR

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 823
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 412
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 178.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 15 à 15.

Assembleur 80x86

Cyrix Cx6x86/AMD Am5k86

OIO

Official Undefined Opcode

Syntaxe

```
OIO
```

Description

Cette instruction permet de provoquer l'exécution d'un code indéfinie.

Algorithme

```
EXCEPTION #UD
```

Mnémonique

Instruction	Opcode	Description
OIO	0Fh FFh	Cette instruction permet de provoquer l'exécution d'un code indéfinie.

Assembleur 80x86 — OR

INTEL 8088+ — *Or bitwise*

Syntaxe

OR *Opérande Cible*, *Opérande Source*

Description

L'instruction *OR* effectue un *OU BINAIRE* sur les 2 opérandes spécifiés, le calcul est placé dans la première opérande, c'est-à-dire l'*Opérande Cible*. Rappelons qu'un *OU BINAIRE* donne le résultat 1 si au moins une des 2 opérandes vaut 1 et donne 0 si les deux bits valent 0.

Algorithme

Opérande Cible ← *Opérande Cible* U *Opérande Source* drapeau *CF* ← 0 drapeau *OF* ← 0

Mnémonique

Instruction	Opcode	Description
OR AL, *imm8*	0Ch *ib*	Effectue un «Ou binaire» du contenu du registre AL avec une valeur immédiate de 8 bits.
OR AX, *imm16*	0Dh *iw*	Effectue un «Ou binaire» du contenu du registre AX avec une valeur immédiate de 16 bits.

OR EAX, *imm32*	0Dh *id*	Effectue un «Ou binaire» du contenu du registre EAX avec une valeur immédiate de 32 bits.
OR RAX, *imm32*	0Dh *id*	Effectue un «Ou binaire» du contenu du registre RAX avec une valeur immédiate de 32 bits.
OR *reg/mem8*, *imm8*	80h /1 *ib*	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 8 bits avec une valeur immédiate de 8 bits.
OR *reg/mem16*, *imm16*	81h /1 *iw*	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 16 bits avec une valeur immédiate de 16 bits.
OR *reg/mem32*, *imm32*	81h /1 *id*	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 32 bits avec une valeur immédiate de 32 bits.
OR *reg/mem64*, *imm32*	81h /1 *id*	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 64 bits avec une valeur immédiate entière de 32 bits.
OR *reg/mem16*, *imm8*	83h /1 *ib*	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 16 bits avec une valeur immédiate entière de 8 bits.
OR *reg/mem32*, *imm8*	83h /1 *ib*	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 32 bits avec une valeur immédiate entière de 8 bits.
OR *reg/mem64*, *imm8*	83h /1 *ib*	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre

		64 bits avec une valeur immédiate entière de 8 bits.
OR *reg/mem8*, *reg8*	08h /r	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 8 bits avec le contenu d'un registre de 8 bits.
OR *reg/mem16*, *reg16*	09h /r	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 16 bits avec le contenu d'un registre de 16 bits.
OR *reg/mem32*, *reg32*	09h /r	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 32 bits avec le contenu d'un registre de 32 bits.
OR *reg/mem64*, *reg64*	09h /r	Effectue un «Ou binaire» du contenu d'une opérande mémoire ou registre 64 bits avec le contenu d'un registre de 64 bits.
OR *reg8*, *reg/mem8*	0Ah /r	Effectue un «Ou binaire» du contenu d'un registre de 8 bits avec le contenu d'une opérande mémoire ou registre 8 bits.
OR *reg16*, *reg/mem16*	0Bh /r	Effectue un «Ou binaire» du contenu d'un registre de 16 bits avec le contenu d'une opérande mémoire ou registre 16 bits.
OR *reg32*, *reg/mem32*	0Bh /r	Effectue un «Ou binaire» du contenu d'un registre de 32 bits avec le contenu d'une opérande mémoire ou registre 32 bits.
OR *reg64*, *reg/mem64*	0Bh /r	Effectue un «Ou binaire» du contenu d'un registre de 64 bits avec le

		contenu d'une opérande mémoire ou registre 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de

				page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Exemple

Cet exemple permet de vérifier si AX vaut 0. Cette technique de programmation à l'apparence un peu étrange, est pourtant très commune chez les professionnels de l'assembleur, car il consomme moins d'octets en mémoire, que l'utilisation de l'instruction «*CMP AX,0*» :

```
1. OR AX,AX
2. JE @axvautzero
3.  ; AX ne vaut pas 0
4.  ; ...
5. @axvautzero
6.  ; AX vaut 0
7.  ; ...
```

Voir également

Instruction assembleur 80x86 - Instruction AND
Instruction assembleur 80x86 - Instruction NEG
Instruction assembleur 80x86 - Instruction NOT
Instruction assembleur 80x86 - Instruction XOR

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 823
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 412
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 179.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 16 à 19.

Assembleur 80x86 — ORPD

INTEL Pentium 4 (SSE2)+

Bitwise Logical OR of Double-Precision Floating-Point Values

Syntaxe

ORPD *destination, source*

Description

Cette instruction permet d'effectuer un *OU BINAIRE* de 128 bits sur les 2 opérandes réels de double précisions spécifiés.

Algorithme

destination(0..127) ← *destination*(0..127) U *source*(0..127)

Mnémonique

Instruction	Opcode	Description
ORPD *xmm1, xmm2/m128*	66h 0Fh 56h /r	Cette instruction permet d'effectuer un *OU BINAIRE* de 128 bits sur les 2 opérandes réels de double précisions spécifiés.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 20 à 21.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

ORPS

Bitwise Logical OR of Double-Precision Floating-Point Values

Syntaxe

ORPS *dest,source*

Description

Cette instruction permet d'effectuer un *OU BINAIRE* de 128 bits sur les 2 opérandes spécifiés.

Algorithme

dest(127..0) ← *dest*(127..0) U *source*(127..0)

Mnémonique

Instruction	Opcode	Description
ORPS *xmm1,xmm2/m128*	0Fh 56h /r	Cette instruction permet d'effectuer un *OU BINAIRE* de 128 bits sur les 2 opérandes spécifiés.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 22 à 23.

Assembleur 80x86	**OUT**
INTEL 8088+	*Output*

Syntaxe

OUT *adresse,accumulateur*

Paramètres

Nom	Description
adresse	Ce paramètre permet d'indiquer l'adresse du port d'entrée/sortie. Il peut s'agir d'une valeur entre 0 et FFFFh. Voir Référence des ports d'entrée/sortie 80x86 pour plus de détails.
accumulateur	Ce paramètre permet d'indiquer la valeur envoyer dans le port d'entrée/sortie. Il peut s'agir d'un des 3 registres AL, AX ou EAX. Même avec les microprocesseurs 64 bits, il n'est pas possible d'envoyer une valeur plus grande que 32 bits.

Description

Cette instruction permet d'envoyer un octet, un mot ou un double mot sur le port d'entrée/sortie.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    Port(adresse) ← accumulateur
```

```
  FIN SI
SINON
  Port(adresse) ← accumulateur
FIN SI
```

Mnémonique

Instruction	Opcode	Description
OUT *imm8*, AL	E6h *ib*	Met un octet contenu dans le registre AL dans le port spécifié par la valeur immédiate de 8 bits.
OUT *imm8*, AX	E7h *ib*	Met un mot contenu dans le registre AX dans le port spécifié par la valeur immédiate de 8 bits.
OUT *imm8*, EAX	E7h *ib*	Met un double mot contenu dans le registre EAX dans le port spécifié par la valeur immédiate de 8 bits.
OUT DX, AL	EEh	Met un octet contenu dans le registre AL dans le port spécifié par le registre DX.
OUT DX, AX	EFh	Met un mot contenu dans le registre AX dans le port spécifié par le registre DX.
OUT DX, EAX	EFh	Met un double mot contenu dans le registre EAX dans le port spécifié par le registre DX.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description

#GP(Protection général)		X		Un ou plusieurs bits de permission d'entrée/sortie sont fixer par le *TSS* pour un accès au port.
			X	Le *CPL* est plus grand que le IOPL et une ou plusieurs bits de permission sont fixer par le *TSS* pour un accès au port.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

Exemple

Cette exemple permet d'effectuer une commande de fin d'interruption (Port 0020h):

```
MOV AL,20h
OUT 20h,AL
```

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction IN
Instruction assembleur 80x86 - Instruction INS
Instruction assembleur 80x86 - Instruction OUTS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 824
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 181.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 24 à 26.

Assembleur 80x86	**OUTS**
INTEL 80286+	*Output String*

Syntaxe

OUTS *opérandesource*

Description

Cette instruction permet d'envoyer un octet, un mot ou un double mot contenu dans l'adresse DS:[SI] du port d'entrée/sortie et incrémente/décrémente le registre SI en fonction de la taille de l'opérande cible et de l'état du drapeau de direction.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    Port(adresse) ← accumulateur
  FIN SI
SINON
  Port(adresse) ← accumulateur
FIN SI
SI (taille de l'opérande_cible = octet) ALORS
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 1
  SINON
    (E)DI ← (E)DI - 1
  FIN SI
SINON SI (taille de l'opérande_cible = mot) ALORS
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 2
  SINON
    (E)DI ← (E)DI - 2
```

```
  FIN SI
SINON
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 4
  SINON
    (E)DI ← (E)DI - 4
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
OUTS DX, *mem8*	6Eh	Envoi un octet de l'adresse DS:(R)SI au port spécifié par DX et ensuite, incrémente ou décrémente le registre (R)SI.
OUTS DX, *mem16*	6Fh	Envoi un mot de l'adresse DS:(R)SI au port spécifié par DX et ensuite, incrémente ou décrémente le registre (R)SI.
OUTS DX, *mem32*	6Fh	Envoi un double mot de l'adresse DS:(R)SI au port spécifié par DX et ensuite, incrémente ou décrémente le registre (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de

				pile ou n'est pas canonique
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite de la table du descripteur.
			X	Un segment de données nulle est utilisé comme référence mémoire
		X		Un ou plusieurs bits de permission d'entrée/sortie sont fixer par le *TSS* pour un accès au port.
			X	Le *CPL* est plus grand que le IOPL et une ou plusieurs bits de permission sont fixer par le *TSS* pour un accès au port.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution

				de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT
Instruction assembleur 80x86 - Instruction IN
Instruction assembleur 80x86 - Instruction INS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 824
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 182.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 27 à 31.

Assembleur 80x86	OUTSB
INTEL 80286+	*Output String Byte*

Syntaxe

```
OUTSB
```

Description

Cette instruction permet d'envoyer un octet contenu dans l'adresse DS:[SI] du port d'entrée/sortie et incrémente/décrémente le registre SI de 1 en fonction de l'état du drapeau de direction.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    Port(adresse) ← accumulateur
  FIN SI
SINON
  Port(adresse) ← accumulateur
FIN SI
SI DF = 0 ALORS
  (E)DI ← (E)DI + 1
SINON
  (E)DI ← (E)DI - 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description

OUTSB	6Eh	Envoi un octet de l'adresse DS:(R)SI au port spécifié par DX et ensuite, incrémente ou décrémente le registre (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite de la table du descripteur.
			X	Un segment de données nulle est utilisé comme référence mémoire
		X		Un ou plusieurs bits de permission d'entrée/sortie sont fixer par

				le *TSS* pour un accès au port.
			X	Le *CPL* est plus grand que le IOPL et une ou plusieurs bits de permission sont fixer par le *TSS* pour un accès au port.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT
Instruction assembleur 80x86 - Instruction IN
Instruction assembleur 80x86 - Instruction INS

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 825*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 182.

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 27 à 31.

Assembleur 80x86	OUTSD
INTEL 80386+	*Output String Double word*

Syntaxe

```
OUTSD
```

Description

Cette instruction permet d'envoyer un double mot contenu dans l'adresse DS:[SI] du port d'entrée/sortie et incrémente/décrémente le registre SI de 4 en fonction de l'état du drapeau de direction.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    Port(adresse) ← accumulateur
  FIN SI
SINON
  Port(adresse) ← accumulateur
FIN SI
SI DF = 0 ALORS
  (E)DI ← (E)DI + 4
SINON
  (E)DI ← (E)DI - 4
FIN SI
```

Mnémonique

Instruction	Opcode	Description

OUTSD	6Fh	Envoi un double mot de l'adresse DS:(R)SI au port spécifié par DX et ensuite, incrémente ou décrémente le registre (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite de la table du descripteur.
			X	Un segment de données nulle est utilisé comme référence mémoire
		X		Un ou plusieurs bits de permission d'entrée/sortie sont fixer par

				le *TSS* pour un accès au port.
			X	Le *CPL* est plus grand que le IOPL et une ou plusieurs bits de permission sont fixer par le *TSS* pour un accès au port.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT
Instruction assembleur 80x86 - Instruction IN
Instruction assembleur 80x86 - Instruction INS

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 825*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 182.

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 27 à 31.

Assembleur 80x86	**OUTSW**
INTEL 80286+	*Output String Word*

Syntaxe

```
OUTSW
```

Description

Cette instruction permet d'envoyer un mot contenu dans l'adresse DS:[SI] du port d'entrée/sortie et incrémente/décrémente le registre SI de 2 en fonction de l'état du drapeau de direction.

Algorithme

```
SI ((PE = 1) ET ((CPL > IOPL) ET (VM = 1))) ALORS * Mode protégé avec CPL > IOPL ou mode virtual 8086
  SI (n'importe quel permission de bit d'E/S pour un port d'E/S à un accès = 1) ALORS
    EXCEPTION #GP(0)
  SINON
    Port(adresse) ← accumulateur
  FIN SI
SINON
  Port(adresse) ← accumulateur
FIN SI
SI DF = 0 ALORS
  (E)DI ← (E)DI + 2
SINON
  (E)DI ← (E)DI - 2
FIN SI
```

Mnémonique

Instruction	Opcode	Description

OUTSW	6Fh	Envoi un mot de l'adresse DS:(R)SI au port spécifié par DX et ensuite, incrémente ou décrémente le registre (R)SI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Sélecteur)	X	X	X	Un registre de segment est chargé, mais le descripteur de segment dépasse la limite de la table du descripteur.
			X	Un segment de données nulle est utilisé comme référence mémoire
		X		Un ou plusieurs bits de permission d'entrée/sortie sont fixer par

				le *TSS* pour un accès au port.
			X	Le *CPL* est plus grand que le IOPL et une ou plusieurs bits de permission sont fixer par le *TSS* pour un accès au port.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Référence des ports d'entrée/sortie 80x86
Instruction assembleur 80x86 - Instruction OUT
Instruction assembleur 80x86 - Instruction IN
Instruction assembleur 80x86 - Instruction INS

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 825*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 182.

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 27 à 31.

Assembleur 80x86 PABSB

SSSE3+ *Packed Absolute Byte*

Syntaxe

PABSB *dest*,*source*

Description

Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des octets d'un opérande source et entrepose le résultat dans un opérande destination.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  dest(7..0) ← | *source*(7..0) |
  dest(15..8) ← | *source*(15..8) |
  dest(23..16) ← | *source*(23..16) |
  dest(31..24) ← | *source*(31..24) |
  dest(39..32) ← | *source*(39..32) |
  dest(47..40) ← | *source*(47..40) |
  dest(55..48) ← | *source*(55..48) |
  dest(63..56) ← | *source*(63..56) |
SINON
  dest(7..0) ← | *source*(7..0) |
  dest(15..8) ← | *source*(15..8) |
  dest(23..16) ← | *source*(23..16) |
  dest(31..24) ← | *source*(31..24) |
  dest(39..32) ← | *source*(39..32) |
  dest(47..40) ← | *source*(47..40) |
  dest(55..48) ← | *source*(55..48) |
  dest(63..56) ← | *source*(63..56) |
  dest(71..64) ← | *source*(71..64) |
  dest(79..72) ← | *source*(79..72) |
  dest(87..80) ← | *source*(87..80) |

```
  dest(95..88) ← | source(95..88) |
  dest(103..96) ← | source(103..96) |
  dest(111..104) ← | source(111..104) |
  dest(119..112) ← | source(119..112) |
  dest(127..120) ← | source(127..120) |
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PABSB *mm1,mm2/m64*	0Fh 38h 1Ch /r	Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des octets d'un opérande source et entrepose le résultat dans un opérande destination.
PABSB *xmm1,xmm2/m128*	66h 0Fh 38h 1Ch /r	Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des octets d'un opérande source et entrepose le résultat dans un opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 33 à 36.

Assembleur 80x86 PABSD

SSSE3+ *Packed Absolute Doubleword*

Syntaxe

PABSD *dest,source*

Description

Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des doubles mots d'un opérande source et entrepose le résultat dans un opérande destination.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  dest(31..0) ← | *source*(31.0) |
  dest(63..32) ← | *source*(63..32) |
SINON
  dest(31..0) ← | *source*(31..0) |
  dest(63..32) ← | *source*(63..32) |
  dest(95..64) ← | *source*(95..64) |
  dest(127..96) ← | *source*(127..96) |
FIN SI

Mnémonique

Instruction	Opcode	Description
PABSD *mm1,mm2/m64*	0Fh 38h 1Eh /r	Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des doubles mots d'un opérande source et entrepose le résultat dans un opérande

		destination.
PABSD *xmm1,xmm2/m128*	66h 0Fh 38h 1Eh /r	Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des doubles mots d'un opérande source et entrepose le résultat dans un opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 33 à 36.

Assembleur 80x86 PABSW

SSSE3+ *Packed Absolute Word*

Syntaxe

PABSW *dest,source*

Description

Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des mots d'un opérande source et entrepose le résultat dans un opérande destination.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  dest(15..0) ← | *source*(15..0) |
  dest(31..16) ← | *source*(31..16) |
  dest(47..32) ← | *source*(47..32) |
  dest(63..48) ← | *source*(63..48) |
SINON
  dest(15..0) ← | *source*(15..0) |
  dest(31..16) ← | *source*(31..16) |
  dest(47..32) ← | *source*(47..32) |
  dest(63..48) ← | *source*(63..48) |
  dest(79..64) ← | *source*(79..64) |
  dest(95..80) ← | *source*(95..80) |
  dest(111..96) ← | *source*(111..96) |
  dest(127..112) ← | *source*(127..112) |
FIN SI

Mnémonique

Instruction	Opcode	Description
PABSW *mm1,mm2/m64*	0F 38 1D /r	Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des mots d'un opérande source et entrepose le résultat dans un opérande destination.
PABSW *xmm1,xmm2/m128*	66 0F 38 1D /r	Cette instruction permet d'effectuer le calcul de la valeur absolue de chacun des mots d'un opérande source et entrepose le résultat dans un opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 33 à 36.

Assembleur 80x86 PACKSSDW

INTEL Pentium Pro+

Pack with Signed Saturation dword to word

Syntaxe

PACKSSDW *dest,source*

Description

Cette instruction permet de compacté 8 paquets de double mots en mots.

Algorithme

dest(7..0) ← SaturateSignedWordToSignedByte *dest*(15..0)
dest(15..8) ← SaturateSignedWordToSignedByte *dest*(31..16)
dest(23..16) ← SaturateSignedWordToSignedByte *dest*(47..32)
dest(31..24) ← SaturateSignedWordToSignedByte *dest*(63..48)
dest(39..32) ← SaturateSignedWordToSignedByte *source*(15..0)
dest(47..40) ← SaturateSignedWordToSignedByte *source*(31..16)
dest(55..48) ← SaturateSignedWordToSignedByte *source*(47..32)
dest(63..56) ← SaturateSignedWordToSignedByte *source*(63..48)

Mnémonique

Instruction	Opcode	Description
PACKSSDW *mm1,mm2/m64*	0Fh 6Bh /r	Cette instruction permet de compacté 8 paquets de double mots en mots.
PACKSSDW *xmm1, xmm2/m128*	66h 0Fh 6Bh /r	Cette instruction permet de compacté 8 paquets de double mots

		en mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 37 à 41.

Assembleur 80x86 — PACKSSWB

INTEL Pentium Pro+ — *Pack with Signed Saturation word to Byte*

Syntaxe

PACKSSWB *dest,source*

Description

Cette instruction permet de compacté 8 paquets d'entier en octets.

Algorithme

dest(7..0) ← SaturateSignedWordToSignedByte *dest*(15..0)
dest(15..8) ← SaturateSignedWordToSignedByte *dest*(31..16)
dest(23..16) ← SaturateSignedWordToSignedByte *dest*(47..32)
dest(31..24) ← SaturateSignedWordToSignedByte *dest*(63..48)
dest(39..32) ← SaturateSignedWordToSignedByte *source*(15..0)
dest(47..40) ← SaturateSignedWordToSignedByte *source*(31..16)
dest(55..48) ← SaturateSignedWordToSignedByte *source*(47..32)
dest(63..56) ← SaturateSignedWordToSignedByte *source*(63..48)

Mnémonique

Instruction	Opcode	Description
PACKSSWB *mm1, mm2/m64*	0Fh 63h /r	Cette instruction permet de compacté 8 paquets d'entier en octets.
PACKSSWB *xmm1, xmm2/m128*	66h 0Fh 63h /r	Cette instruction permet de compacté 8 paquets d'entier en

		octets.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 37 à 41.

Assembleur 80x86 PACKUSDW

SSSE3+

Pack with Unsigned Saturation

Syntaxe

PACKUSDW *dest*,*source*

Description

Cette instruction permet de compacté 8 paquets de double mots naturel en mots naturel.

Algorithme

SI *dest*(31..0) < 0 **ALORS**
  TMP(15..0) ← 0
SINON
  TMP(15..0) ← *dest*(15..0)
FIN SI
SI *dest*(31..0) > FFFFh **ALORS**
  dest(15..0) ← FFFFh
SINON
  dest(15..0) ← TMP(15..0)
FIN SI
SI *dest*(63..32) < 0 **ALORS**
  TMP(31..16) ← 0
SINON
  TMP(31..16) ← *dest*(47..32)
FIN SI
SI *dest*(63..32) > FFFFh **ALORS**
  dest(31..16) ← FFFFh
SINON
  dest(31..16) ← TMP(31..16)
FIN SI
SI *dest*(95..64) < 0 **ALORS**

```
  TMP(47..32) ← 0
SINON
  TMP(47..32) ← dest(79..64)
FIN SI
SI dest(95..64) > FFFFh ALORS
  dest(47..32) ← FFFFh
SINON
  dest(47..32) ← TMP(47..32)
FIN SI
SI dest(127..96) < 0 ALORS
  TMP(63..48) ← 0
SINON
  dest(111..96)
FIN SI
SI dest(127..96) > FFFFh ALORS
  dest(63..48) ← FFFFh
SINON
  dest(63..48) ← TMP(63..48)
FIN SI
SI dest(127..96) < 0 ALORS
  TMP(63..48) ← 0
SINON
  dest(111..96)
FIN SI
SI dest(127..96) > FFFFh ALORS
  dest(63..48) ← FFFFh
SINON
  dest(63..48) ← TMP(63..48)
FIN SI
SI source(31..0) < 0 ALORS
  TMP(79..64) ← 0
SINON
  TMP(79..64) ← source(15..0)
FIN SI
SI source(31..0) > FFFFh ALORS
  dest(63..48) ← FFFFH
SINON
  dest(63..48) ← TMP(79..64)
FIN SI
SI source(63..32) < 0 ALORS
  TMP(95..80) ← 0
```

```
SINON
  source(47..32)
FIN SI
SI source(63..32) > FFFFh ALORS
  dest(95..80) ← FFFFh
SINON
  dest(95..80) TMP(95..80)
FIN SI
SI source(95..64) < 0 ALORS
  TMP(111..96) ← 0
SINON
  TMP(111..96) ← source(79..64)
FIN SI
SI source(95..64) > FFFFh ALORS
  dest(111..96) ← FFFFh
SINON
  dest(111..96) ← TMP(111..96)
FIN SI
SI source(127..96) < 0 ALORS
  TMP(127..112) ← 0
SINON
  TMP(127..112) ← source(111..96)
FIN SI
SI source(127..96) > FFFFh ALORS
  dest(128..112) ← FFFFh
SINON
  dest(128..112) ← TMP(127..112)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PACKUSDW *xmm1,xmm2/m128*	66h 0Fh 38h 2Bh /r	Cette instruction permet de compacté 8 paquets de double mots naturel en mots naturel.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 42 à 44.*

Assembleur 80x86 PACKUSWB

INTEL Pentium Pro+ *Pack with Unsigned Saturation*

Syntaxe

PACKUSWB *dest,source*

Description

Cette instruction permet de compacté 8 paquets de mots en octets.

Algorithme

dest(7..0) ← SaturateSignedWordToUnsignedByte *dest*(15..0)
dest(15..8) ← SaturateSignedWordToUnsignedByte *dest*(31..16)
dest(23..16) ← SaturateSignedWordToUnsignedByte *dest*(47..32)
dest(31..24) ← SaturateSignedWordToUnsignedByte *dest*(63..48)
dest(39..32) ← SaturateSignedWordToUnsignedByte *source*(15..0)
dest(47..40) ← SaturateSignedWordToUnsignedByte *source*(31..16)
dest(55..48) ← SaturateSignedWordToUnsignedByte *source*(47..32)
dest(63..56) ← SaturateSignedWordToUnsignedByte *source*(63..48)

Mnémonique

Instruction	Opcode	Description
PACKUSWB *mm, mm/m64*	0Fh 67h /r	Cette instruction permet de compacté 8 paquets de mots en octets.
PACKUSWB *xmm1, xmm2/m128*	66h 0Fh 67h /r	Cette instruction permet de compacté 8 paquets de mots en

		octets.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 45 à 48.

Assembleur 80x86 — PADDB

INTEL Pentium Pro+ — *Packed Add Bytes*

Syntaxe

PADDB *dest,source*

Description

Cette instruction permet d'effectuer une addition scalaire sur un paquet de 8 octets.

Algorithme

dest(7..0) ← *dest*(7..0) + *source*(7..0)
dest(15..8) ← *dest*(15..8) + *source*(15..8)
dest(23..16) ← *dest*(23..16) + *source*(23..16)
dest(31..24) ← *dest*(31..24) + *source*(31..24)
dest(39..32) ← *dest*(39..32) + *source*(39..32)
dest(47..40) ← *dest*(47..40) + *source*(47..40)
dest(55..48) ← *dest*(55..48) + *source*(55..48)
dest(63..56) ← *dest*(63..56) + *source*(63..56)

Mnémonique

Instruction	Opcode	Description
PADDB *mm, mm/m64*	0Fh FCh /r	Cette instruction permet d'effectuer une addition scalaire sur un paquet de 8 octets.
PADDB *xmm1,xmm2/m128*	66h 0Fh FCh /r	Cette instruction permet d'effectuer une addition scalaire sur un paquet

		de 8 octets.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 49 à 52.

Assembleur 80x86 — PADDD

INTEL Pentium Pro+ — *Packed Add Dwords*

Syntaxe

PADDD *dest,source*

Description

Cette instruction permet d'effectuer une addition scalaire sur un paquet de 2 double mots.

Algorithme

dest(31..0) ← *dest*(31..0) + *source*(31..0)
dest(63..32) ← *dest*(63..32) + *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PADDD *mm, mm/m64*	0Fh FEh /r	Cette instruction permet d'effectuer une addition scalaire sur un paquet de 2 double mots.
PADDD *xmm1, xmm2/m128*	66h 0Fh FEh /r	Cette instruction permet d'effectuer une addition scalaire sur un paquet de 2 double mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 49 à 52.

Assembleur 80x86 — PADDQ

INTEL Pentium 4 (SSE2)+ — *Add Packed Quadword Integers*

Syntaxe

PADDQ *destination*, *source*

Description

Cette instruction permet d'effectuer une addition scalaire sur un paquet de quadruple mots.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  Destination(0..63) ← Destination(0..63) + Source(0..63)
SINON
  Destination(0..63) ← Destination(0..63) + Source(0..63)
  Destination(64..127) ← Destination(64..127) + Source(64..127)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PADDQ *mm1,mm2/m64*	0Fh D4h /r	Cette instruction permet d'effectuer une addition scalaire sur un paquet de quadruple mots.
PADDQ *xmm1,xmm2/m128*	66h 0Fh D4h /r	Cette instruction permet d'effectuer une addition scalaire sur un paquet

		de quadruple mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 53 à 55.

Assembleur 80x86 — PADDSB

INTEL Pentium Pro+ — *Packed Add with Saturation Bytes*

Syntaxe

PADDSB *dest,source*

Description

Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 8 octets signés.

Algorithme

dest(7..0) ← SaturateToSignedByte(*dest*(7..0) + *source*(7..0))
dest(15..8) ← SaturateToSignedByte(*dest*(15..8) + *source*(15..8))
dest(23..16) ← SaturateToSignedByte(*dest*(23..16) + *source*(23..16))
dest(31..24) ← SaturateToSignedByte(*dest*(31..24) + *source*(31..24))
dest(39..32) ← SaturateToSignedByte(*dest*(39..32) + *source*(39..32))
dest(47..40) ← SaturateToSignedByte(*dest*(47..40) + *source*(47..40))
dest(55..48) ← SaturateToSignedByte(*dest*(55..48) + *source*(55..48))
dest(63..56) ← SaturateToSignedByte(*dest*(63..56) + *source*(63..56))

Mnémonique

Instruction	Opcode	Description
PADDSB *mm, mm/m64*	0Fh ECh /r	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 8 octets signés.
PADDSB *xmm1,xmm2/m128*	66h 0Fh ECh /r	Cette instruction permet d'effectuer

		une addition scalaire avec saturation sur un paquet de 8 octets signés.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 56 à 58.

Assembleur 80x86 — PADDSIW

Cyrix 6x86MX (EMMX)+ — *Packed Add with Saturation*

Syntaxe

PADDSIW *destination, source*

Description

Cette instruction permet d'effectuer l'addition du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans le registre *MMX*.

Algorithme

destination(15..0) ← SaturateToSignedWord(*destination*(15..0) + *source*(15..0)
destination(31..16) ← SaturateToSignedWord(*destination*(31..16) + *source*(31..16)
destination(47..32) ← SaturateToSignedWord(*destination*(47..32) + *source*(47..32)
destination(63..48) ← SaturateToSignedWord(*destination*(63..48) + *source*(63..48)

Mnémonique

Instruction	Opcode	Description
PADDSIW *mm,mm/m64*	0Fh 51h *PostByte*	Cette instruction permet d'effectuer l'addition du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans le registre *MMX*.

Assembleur 80x86 — PADDSW

INTEL Pentium Pro+ — *Packed Add with Saturation Words*

Syntaxe

PADDSW *dest,source*

Description

Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 4 entiers.

Algorithme

dest(15..0) ← SaturateToSignedWord(*dest*(15..0) + *source*(15..0))
dest(31..16) ← SaturateToSignedWord(*dest*(31..16) + *source*(31..16))
dest(47..32) ← SaturateToSignedWord(*dest*(47..32) + *source*(47..32))
dest(63..48) ← SaturateToSignedWord(*dest*(63..48) + *source*(63..48))

Mnémonique

Instruction	Opcode	Description
PADDSW *mm, mm/m64*	0Fh EDh /r	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 4 entiers.
PADDSW *xmm1, xmm2/m128*	66h 0Fh EDh /r	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 4 entiers.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 56 à 58.

Assembleur 80x86 — PADDUSB

INTEL Pentium Pro+ — *Packed Add Unsigned with Saturation Bytes*

Syntaxe

PADDUSB *dest*,*source*

Description

Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 8 octets.

Algorithme

dest(7..0) ← SaturateToUnsignedByte(*dest*(7..0) + *source*(7..0))
dest(15..8) ← SaturateToUnsignedByte(*dest*(15..8) + *source*(15..8))
dest(23..16) ← SaturateToUnsignedByte(*dest*(23..16) + *source*(23..16))
dest(31..24) ← SaturateToUnsignedByte(*dest*(31..24) + *source*(31..24))
dest(39..32) ← SaturateToUnsignedByte(*dest*(39..32) + *source*(39..32))
dest(47..40) ← SaturateToUnsignedByte(*dest*(47..40) + *source*(47..40))
dest(55..48) ← SaturateToUnsignedByte(*dest*(55..48) + *source*(55..48))
dest(63..56) ← SaturateToUnsignedByte(*dest*(63..56) + *source*(63..56))

Mnémonique

Instruction	Opcode	Description
PADDUSB *mm, mm/m64*	0Fh DCh /r	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 8 octets.
PADDUSB *xmm1, xmm2/m128*	66h 0Fh DCh /r	Cette instruction permet d'effectuer

		une addition scalaire avec saturation sur un paquet de 8 octets.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 59 à 63.

Assembleur 80x86 — PADDUSW

INTEL Pentium Pro+ — *Packed Add Unsigned with Saturation Words*

Syntaxe

PADDUSW *dest*,*source*

Description

Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 4 mots.

Algorithme

dest(15..0) ← SaturateToUnsignedWord(*dest*(15..0) + *source*(15..0))
dest(31..16) ← SaturateToUnsignedWord(*dest*(31..16) + *source*(31..16))
dest(47..32) ← SaturateToUnsignedWord(*dest*(47..32) + *source*(47..32))
dest(63..48) ← SaturateToUnsignedWord(*dest*(63..48) + *source*(63..48))

Mnémonique

Instruction	Opcode	Description
PADDUSW *mm*, *mm/m64*	0Fh DDh /r	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 4 mots.
PADDUSW *xmm1*, *xmm2/m128*	66h 0Fh DDh /r	Cette instruction permet d'effectuer une addition scalaire avec saturation sur un paquet de 4 mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 59 à 63.

Assembleur 80x86 — PADDW

INTEL Pentium Pro+ — *Packed Add Words*

Syntaxe

PADDW *dest,source*

Description

Cette instruction permet d'effectuer une addition scalaire sur un paquet de 4 mots.

Algorithme

dest(15..0) ← *dest*(15..0) + *source*(15..0)
dest(31..16) ← *dest*(31..16) + *source*(31..16)
dest(47..32) ← *dest*(47..32) + *source*(47..32)
dest(63..48) ← *dest*(63..48) + *source*(63..48)

Mnémonique

Instruction	Opcode	Description
PADDW *mm, mm/m64*	0Fh FDh /r	Cette instruction permet d'effectuer une addition scalaire sur un paquet de 4 mots.
PADDW *xmm1, xmm2/m128*	66h 0Fh FDh /r	Cette instruction permet d'effectuer une addition scalaire sur un paquet de 4 mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 49 à 52.*

Assembleur 80x86 PALIGNR

SSSE3+ *Packed Align Right*

Syntaxe

PALIGNR *dest*,*source*,*immediat*

Description

Cette instruction permet d'effectuer la concaténation de l'opérande de destination et de l'opérande source et effectue un décalage d'une constante immédiate de la composante de granulité d'octets, et enfin extrait le résultat aligné dans l'opérande de destination.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  temp1(127..0) ← CONCATENATE(*dest*,*source*) >> (*immediat* x 8)
  dest(63..0) ← temp1(63..0)
FIN SI
SI taille de l'opérande = 128 bits **ALORS**
  temp1(255..0) ← CONCATENATE(*dest*,*source*)>>(*immediat* x 8)
  dest(127..0) ← temp1(127..0)
FIN SI

Mnémonique

Instruction	Opcode	Description
PALIGNR *mm1,mm2/m64, imm8*	0Fh 3Ah 0Fh	Cette instruction permet d'effectuer la concaténation de l'opérande de destination et de l'opérande source et effectue un décalage d'une constante

		immédiate de la composante de granulité d'octets, et enfin extrait le résultat aligné dans l'opérande de destination.
PALIGNR *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 0Fh	Cette instruction permet d'effectuer la concaténation de l'opérande de destination et de l'opérande source et effectue un décalage d'une constante immédiate de la composante de granulité d'octets, et enfin extrait le résultat aligné dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 63 à 65.

Assembleur 80x86 — PAND

INTEL Pentium MMX+ — *Bitwise Logical AND*

Syntaxe

PAND *destination*, *source*

Description

Cette instruction permet d'effectuer un «*ET BINAIRE*» d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.

Algorithme

destination ← *destination* ∩ *source*

Mnémonique

Instruction	Opcode	Description
PAND *mm*, *mm/m64*	0Fh DBh /r	Cette instruction permet d'effectuer un «*ET BINAIRE*» d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.
PAND *xmm1*, *xmm2/m128*	66h 0Fh DBh /r	Cette instruction permet d'effectuer un «*ET BINAIRE*» d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 67 à 69.

Assembleur 80x86 — PANDN

INTEL Pentium Pro+ — *Bitwise Logical And Not*

Syntaxe

PANDN *destination,source*

Description

Cette instruction permet d'effectuer un «*ET BINAIRE*» et une «*NEGATION*» de chacun des bits d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.

Algorithme

destination ← (¬ *destination*) ∩ *source*

Mnémonique

Instruction	Opcode	Description
PANDN *mm, mm/m64*	0Fh DFh /r	Cette instruction permet d'effectuer un «*ET BINAIRE*» et une «*NEGATION*» de chacun des bits d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.
PANDN *xmm1, xmm2/m128*	66h 0Fh DFh /r	Cette instruction permet d'effectuer un «*ET BINAIRE*» et une «*NEGATION*» de chacun des bits d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des

		registres *XMM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 70 à 72.

Assembleur 80x86 PAUSE

INTEL Pentium 4 (SSE2)+ *Pause*

Syntaxe

```
PAUSE
```

Description

Cette instruction permet d'améliorer les performances des boucles de «*SPIN*», en fournissant une indication pour le microprocesseur que le code courant est dans une boucle en «*SPIN*».

Mnémonique

Instruction	Opcode	Description
PAUSE	F3h 90h	Fournit un délai de performance à l'exécution.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 73 à 74.

Assembleur 80x86 — PAVEB

Cyrix 6x86MX (EMMX)+ — *Packed Average Bits*

Syntaxe

PAVEB *destination, source*

Description

Cette instruction permet d'effectuer le calcul de la moyenne des paquets spécifiés.

Algorithme

destination(7..0) ← (*destination*(7..0) + *source*(7..0)) >> 1
destination(15..8) ← (*destination*(15..8) + *source*(15..8)) >> 1
destination(23..16) ← (*destination*(23..16) + *source*(23..16)) >> 1
destination(31..24) ← (*destination*(31..24) + *source*(31..24)) >> 1
destination(39..32) ← (*destination*(39..32) + *source*(39..32)) >> 1
destination(47..40) ← (*destination*(47..40) + *source*(47..40)) >> 1
destination(55..48) ← (*destination*(55..48) + *source*(55..48)) >> 1
destination(63..56) ← (*destination*(63..56) + *source*(63..56)) >> 1

Mnémonique

Instruction	Opcode	Description
PAVEB *mm,mm/m64*	0Fh 50h *PostByte*	Cette instruction permet d'effectuer le calcul de la moyenne des paquets spécifiés.

Assembleur 80x86 — PAVGB

INTEL Pentium 4 (SSE2)+ — *Packed Average Byte*

Syntaxe

PAVGB *destination*, *source*

Description

Cette instruction permet d'effectuer le calcul de la moyenne des paquets d'octets spécifiés.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  source(0..7) ← (*source*(0..7) + *destination*(0..7) + 1) >> 1
  source(8..15) ← (*source*(8..15) + *destination*(8..15) + 1) >> 1
  source(16..23) ← (*source*(16..23) + *destination*(16..23) + 1) >> 1
  source(24..31) ← (*source*(24..31) + *destination*(24..31) + 1) >> 1
  source(32..39) ← (*source*(32..39) + *destination*(32..39) + 1) >> 1
  source(40..47) ← (*source*(40..47) + *destination*(40..47) + 1) >> 1
  source(48..55) ← (*source*(48..55) + *destination*(48..55) + 1) >> 1
  source(56..63) ← (*source*(56..63) + *destination*(56..63) + 1) >> 1
SINON
  source(0..7) ← (*source*(0..7) + *destination*(0..7) + 1) >> 1
  source(8..15) ← (*source*(8..15) + *destination*(8..15) + 1) >> 1
  source(16..23) ← (*source*(16..23) + *destination*(16..23) + 1) >> 1
  source(24..31) ← (*source*(24..31) + *destination*(24..31) + 1) >> 1
  source(32..39) ← (*source*(32..39) + *destination*(32..39) + 1) >> 1
  source(40..47) ← (*source*(40..47) + *destination*(40..47) + 1) >> 1
  source(48..55) ← (*source*(48..55) + *destination*(48..55) + 1) >> 1
  source(56..63) ← (*source*(56..63) + *destination*(56..63) + 1) >> 1
  source(64..71) ← (*source*(64..71) + *destination*(64..71) + 1) >> 1
  source(72..79) ← (*source*(72..79) + *destination*(72..79) + 1) >> 1
  source(80..87) ← (*source*(80..87) + *destination*(80..87) + 1) >> 1

```
  source(88..95) ← (source(88..95) + destination(88..95) + 1) >> 1
  source(96..103) ← (source(96..103) + destination(96..103) + 1) >> 1
  source(104..111) ← (source(104..111) + destination(104..111) + 1) >> 1
  source(112..119) ← (source(112..119) + destination(112..119) + 1) >> 1
  source(120..127) ← (source(120..127) + destination(120..127) + 1) >> 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PAVGB *mm1, mm2/m64*	0Fh E0h /r	Cette instruction permet d'effectuer le calcul de la moyenne des paquets d'octets spécifiés.
PAVGB *xmm1, xmm2/m128*	66h 0Fh E0h /r	Cette instruction permet d'effectuer le calcul de la moyenne des paquets d'octets spécifiés.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 75 à 77.

Assembleur 80x86 — PAVGUSB

AMD 3DNow! — *Avarage of Unsigned packed 8*

Syntaxe

PAVGUSB *dest,source*

Description

Cette instruction permet d'effectuer le calcul de la moyenne de paquets de 8 octets spécifiés.

Algorithme

dest(7..0) ← (*dest*(7..0) + *source*(7..0)) / 2
dest(15..8) ← (*dest*(15..8) + *source*(15..8)) / 2
dest(23..16) ← (*dest*(23..16) + *source*(23..16)) / 2
dest(31..24) ← (*dest*(31..24) + *source*(31..24)) / 2
dest(39..32) ← (*dest*(39..32) + *source*(39..32)) / 2
dest(47..40) ← (*dest*(47..40) + *source*(47..40)) / 2
dest(55..48) ← (*dest*(55..48) + *source*(55..48)) / 2
dest(63..56) ← (*dest*(63..56) + *source*(63..56)) / 2

Mnémonique

Instruction	Opcode	Description
PAVGUSB *mm,mm/m64*	0Fh 0Fh BFh *Postbyte*	Cette instruction permet d'effectuer le calcul de la moyenne de paquets de 8 octets spécifiés.

Assembleur 80x86 — PAVGW

Intel Pentium III (SSE)+ — *Packed Average Word*

Syntaxe

PAVGW *destination*, *source*

Description

Cette instruction permet d'effectuer le calcul de la moyenne des paquets des mots spécifiés.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  source(0..15) ← (*source*(0..15) + *destination*(0..15) + 1) >> 1
  source(16..31) ← (*source*(16..31) + *destination*(16..31) + 1) >> 1
  source(32..47) ← (*source*(32..47) + *destination*(32..47) + 1) >> 1
  source(48..63) ← (*source*(48..63) + *destination*(48..63) + 1) >> 1
SINON
  source(0..15) ← (*source*(0..15) + *destination*(0..15) + 1) >> 1
  source(16..31) ← (*source*(16..31) + *destination*(16..31) + 1) >> 1
  source(32..47) ← (*source*(32..47) + *destination*(32..47) + 1) >> 1
  source(48..63) ← (*source*(48..63) + *destination*(48..63) + 1) >> 1
  source(64..79) ← (*source*(64..79) + *destination*(64..79) + 1) >> 1
  source(80..95) ← (*source*(80..95) + *destination*(80..95) + 1) >> 1
  source(96..111) ← (*source*(96..111) + *destination*(96..111) + 1) >> 1
  source(112..127) ← (*source*(112..127) + *destination*(112..127) + 1) >> 1
FIN SI

Mnémonique

Instruction	Opcode	Description
PAVGW *mm1, mm2/m64*	0Fh E3h /r	Cette instruction permet d'effectuer le calcul de la moyenne des paquets des mots spécifiés.
PAVGW *xmm1, xmm2/m128*	66h 0Fh E3h /r	Cette instruction permet d'effectuer le calcul de la moyenne des paquets des mots spécifiés.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 75 à 77.

Assembleur 80x86 PBLENDVB

SSE4.1+ *Variable Blend Packed Bytes*

Syntaxe

PBLENDVB *dest,source,masque*

Description

Cette instruction permet d'effectuer une copie conditionnel d'un élément d'octets d'un opérande source vers un opérande de destination en fonction des bits de masque définit dans un troisième opérande.

Algorithme

SI *masque*(7) = 1 **ALORS**
  dest(7..0) ← *source*(7..0)
SINON
  dest(7..0) ← *dest*(7..0)
FIN SI
SI *masque*(15) = 1 **ALORS**
  dest(15..8) ← *source*(15..8)
SINON
  dest(15..8) ← *dest*(15..8)
FIN SI
SI *masque*(23) = 1 **ALORS**
  dest(23..16) ← *source*(23..16)
SINON
  dest(23..16) ← *dest*(23..16)
FIN SI
SI *masque*(31) = 1 **ALORS**
  dest(31..24) ← *source*(31..24)
SINON
  dest(31..24) ← *dest*(31..24)
FIN SI

```
SI masque(39) = 1 ALORS
  dest(39..32) ← source(39..32)
SINON
  dest(39..32) ← dest(39..32)
FIN SI
SI masque(47) = 1 ALORS
  dest(47..40) ← source(47..40)
SINON
  dest(47..40) ← dest(47..40)
FIN SI
SI masque(55) = 1 ALORS
  dest(55..48) ← source(55..48)
SINON
  dest(55..48) ← dest(55..48)
FIN SI
SI masque(63) = 1 ALORS
  dest(63..56) ← source(63..56)
SINON
  dest(63..56) ← dest(63..56)
FIN SI
SI masque(71) = 1 ALORS
  dest(71..64) ← source(71..64)
SINON
  dest(71..64) ← dest(71..64)
FIN SI
SI masque(79) = 1 ALORS
  dest(79..72) ← source(79..72)
SINON
  dest(79..72) ← dest(79..72)
FIN SI
SI masque(87) = 1 ALORS
  dest(87..80) ← source(87..80)
SINON
  dest(87..80) ← dest(87..80)
FIN SI
SI masque(95) = 1 ALORS
  dest(95..88) ← source(95..88)
SINON
  dest(95..88) ← dest(95..88)
FIN SI
SI masque(103) = 1 ALORS
```

```
  dest(103..96) ← source(103..96)
SINON
  dest(103..96) ← dest(103..96)
FIN SI
SI masque(111) = 1 ALORS
  dest(111..104) ← source(111..104)
SINON
  dest(111..104) ← dest(111..104)
FIN SI
SI masque(119) = 1 ALORS
  dest(119..112) ← source(119..112)
SINON
  dest(119..112) ← dest(119..112)
FIN SI
SI masque(127) = 1 ALORS
  dest(127..120) ← source(127..120)
SINON
  dest(127..120) ← dest(127..120)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PBLENDVB *xmm1,xmm2/m128,XMM0*	66h 0Fh 38h 10h /r	Cette instruction permet d'effectuer une copie conditionnel d'un élément d'octets d'un opérande source vers un opérande de destination en fonction des bits de masque définit dans un troisième opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 78 à 82.

Assembleur 80x86 PBLENDW

SSE4.1+ *Blend Packed Words*

Syntaxe

PBLENDW *dest,source,immediat*

Description

Cette instruction permet d'effectuer une copie conditionnel d'un élément d'octets d'un opérande source vers un opérande de destination en fonction des bits de masque définit dans un troisième opérande.

Algorithme

SI *immediat*(0) = 1 **ALORS**
  dest(15..0) ← *source*(15..0)
SINON
  dest(15..0) ← *dest*(15..0)
FIN SI
SI *immediat*(1) = 1 **ALORS**
  dest(31..16) ← *source*(31..16)
SINON
  dest(31..16) ← *dest*(31..16)
FIN SI
SI *immediat*[2] = 1 **ALORS**
  dest(47..32) ← *source*(47..32)
SINON
  dest(47..32) ← *dest*(47..32)
FIN SI
SI *immediat*(3) = 1 **ALORS**
  dest(63..48) ← *source*(63..48)
SINON
  dest(63..48) ← *dest*(63..48)
FIN SI

```
SI immediat(4) = 1 ALORS
  dest(79..64) ← source(79..64)
SINON
  dest(79..64) ← dest(79..64)
FIN SI
SI immediat(5) = 1 ALORS
  dest(95..80) ← source(95..80)
SINON
  dest(95..80) ← dest(95..80)
FIN SI
SI immediat(6) = 1 ALORS
  dest(111..96) ← source(111..96)
SINON
  dest(111..96) ← dest(111..96)
FIN SI
SI immediat(7) = 1 ALORS
  dest(127..112) ← source(127..112)
SINON
  dest(127..112) ← dest(127..112)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PBLENDW *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 0Eh /r *ib*	Cette instruction permet d'effectuer une copie conditionnel d'un élément d'octets d'un opérande source vers un opérande de destination en fonction des bits de masque définit dans un troisième opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 83 à 85.

Assembleur 80x86 PCLMULQDQ

INTEL AES+

Carry-Less Multiplication Quadword

Syntaxe

PCLMULQDQ *dest*,*source*,*immediat*

Description

Cette instruction permet d'effectuer une multiplication de quadruple mots avec une retenue en sélectionnant l'opérande destination et source et une valeur immédiate pour sélectionner les critères.

Algorithme

```
SI immediat[0] = 0 ALORS
  TEMP1 ← dest(63..0)
SINON
  TEMP1 ← dest(127..64)
FIN SI
SI immediat(4) = 0 ALORS
  TEMP2 ← source (63..0)
SINON
  TEMP2 ← source (127..64)
FIN SI
BOUCLE POUR i ← 0 JUSQU'A 63 SAUT 1
  TmpB(i) ← (TEMP1(0) ∩ TEMP2(i))
  BOUCLE POUR j ← 1 JUSQU'A i SAUT 1
    TmpB(i) ← TmpB(i) XOR (TEMP1(j) ∩ TEMP2(i - j))
  FIN BOUCLE POUR
  dest(i) ← TmpB(i)
FIN BOUCLE POUR
BOUCLE POUR i ← 64 JUSQU'A 126 SAUT 1
  TmpB(i) ← 0;
  BOUCLE POUR j ← i - 63 JUSQU'A 63 SAUT 1
```

```
    TmpB(i) ← TmpB(i) XOR (TEMP1(j) ∩ TEMP2(i - j))
  FIN BOUCLE POUR
  dest(i) ← TmpB(i)
FIN BOUCLE POUR
dest(127) ← 0
```

Mnémonique

Instruction	Opcode	Description
PCLMULQDQ *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 44h /r *ib*	Cette instruction permet d'effectuer une multiplication de quadruple mots avec une retenue en sélectionnant l'opérande destination et source et une valeur immédiate pour sélectionner les critères.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 86 à 89.

Assembleur 80x86 — PCMPEQB

INTEL Pentium Pro+ — *Packed Compare for Equal Bytes*

Syntaxe

PCMPEQB *dest*,*source*

Description

Cette instruction permet d'effectuer une comparaison sur un paquet d'octets et fixe la valeur de chacun d'eux à FFh s'il sont égale sinon à 00h.

Algorithme

```
SI dest(7..0) = source(7..0) ALORS
  dest(7..0) ← FFh
SINON
  dest(7..0) ← 00h
FIN SI
SI dest(15..8) = source(15..8) ALORS
  dest(15..8) ← FFh
SINON
  dest(15..8) ← 00h
FIN SI
SI dest(23..16) = source(23..16) ALORS
  dest(23..16) ← FFh
SINON
  dest(23..16) ← 00h
FIN SI
SI dest(31..24) = source(31..24) ALORS
  dest(31..24) ← FFh
SINON
  dest(31..24) ← 00h
FIN SI
SI dest(39..32) = source(39..32) ALORS
```

```
  dest(39..32) ← FFh
SINON
  dest(39..32) ← 00h
FIN SI
SI dest(47..40) = source(47..40) ALORS
  dest(47..40) ← FFh
SINON
  dest(47..40) ← 00h
FIN SI
SI dest(55..48) = source(55..48) ALORS
  dest[55..48] ← FFh
SINON
  dest[55..48] ← 00h
FIN SI
SI dest[63..56] = source(63..56) ALORS
  dest(63..56) ← FFh
SINON
  dest(63..56) ← 00h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PCMPEQB *mm,mm/m64*	0Fh 74h *PostByte*	Cette instruction permet d'effectuer une comparaison sur un paquet d'octets et fixe la valeur de chacun d'eux à FFh s'il sont égale sinon à 00h.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 90 à 93.

Assembleur 80x86 — PCMPEQD

INTEL Pentium Pro+ — *Packed Compare for Equal Dwords*

Syntaxe

PCMPEQD *dest,source*

Description

Cette instruction permet d'effectuer une comparaison sur un paquet de double mots et fixe la valeur de chacun d'eux à FFFFFFFFh s'il sont égale sinon à 00000000h.

Algorithme

```
SI dest(31..0) = source(31..0) ALORS
  dest(31..0) ← FFFFFFFFh
SINON
  dest(31..0) ← 00000000h
FIN SI
SI dest[63..32] = source(63..32) ALORS
  dest(63..32) ← FFFFFFFFh
SINON
  dest(63..32) ← 00000000h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PCMPEQD *mm,mm/m64*	0Fh 76h /r	Cette instruction permet d'effectuer une comparaison sur un paquet de double mots et fixe la valeur de chacun d'eux à FFFFFFFFh s'il sont

		égale sinon à 00000000h.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 90 à 93.

Assembleur 80x86 PCMPEQQ

SSE4.1+ *Compare Packed Qword Data for Equal*

Syntaxe

PCMPEQQ *dest,source*

Description

Cette instruction permet d'effectuer une comparaison d'égalité *SIMD* d'un paquet de quadruple mots de l'opérande destination et de l'opérande source.

Algorithme

```
SI dest(63..0) = source(63..0) ALORS
  dest(63..0) ← FFFFFFFFFFFFFFFFh
SINON
  dest(63..0) ← 0
FIN SI
SI dest(127..64) = source(127..64) ALORS
  dest(127..64) ← FFFFFFFFFFFFFFFFh
SINON
  dest(127..64) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PCMPEQQ *xmm1,xmm2/m128*	66h 0Fh 38h 29h /r	Cette instruction permet d'effectuer une comparaison d'égalité *SIMD* d'un paquet de quadruple mots de l'opérande destination et de

		l'opérande source.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 94 à 95.

Assembleur 80x86 — PCMPEQW

INTEL Pentium Pro+ — *Packed Compare for Equal Words*

Syntaxe

PCMPEQW *dest*,*source*

Description

Cette instruction permet d'effectuer une comparaison sur un paquet de mots et fixe la valeur de chacun d'eux à FFFFh s'il sont égale sinon à 0000h.

Algorithme

```
SI dest(15..0) = source(15..0) ALORS
  dest(15..0) ← FFFFh
SINON
  dest(15..0) ← 0000h
FIN SI
SI dest(31..16) = source(31..16) ALORS
  dest(31..16) ← FFFFh
SINON
  dest(31..16) ← 0000h
FIN SI
SI dest(47..32) = source(47..32) ALORS
  dest(47..32) ← FFFFh
SINON
  dest(47..32) ← 0000h
FIN SI
SI dest(63..48) = source(63..48) ALORS
  dest(63..48) ← FFFFh
SINON
  dest(63..48) ← 0000h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PCMPEQW *mm,mm/m64*	07h 75h /r	Cette instruction permet d'effectuer une comparaison sur un paquet de mots et fixe la valeur de chacun d'eux à FFFFh s'il sont égale sinon à 0000h.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 90 à 93.

Assembleur 80x86 — PCMPESTRI

Nehalem (SSE4.2)+ — *Packed Compare Explicit Length Strings, Return Index*

Syntaxe

PCMPESTRI *dest,source,immediat*

Description

Cette instruction permet d'effectuer une comparaison ainsi qu'un processus de fragments de deux chaine de caractères basé sur une valeur d'encodage indiquer par un opérande immédiate et génère un index entreposé dans le registre *ECX*.

Mnémonique

Instruction	Opcode	Description
PCMPESTRI *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 61h /r *imm8*	Cette instruction permet d'effectuer une comparaison ainsi qu'un processus de fragments de deux chaine de caractères basé sur une valeur d'encodage indiquer par un opérande immédiate et génère un index entreposé dans le registre *ECX*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 96 à 98.

Assembleur 80x86 — PCMPESTRM

Nehalem (SSE4.2)+ — *Packed Compare Explicit Length Strings, Return Mask*

Syntaxe

PCMPESTRM *dest,source,immediat*

Description

Cette instruction permet d'effectuer une comparaison ainsi qu'un processus de fragments de deux chaine de caractères basé sur une valeur d'encodage indiquer par un opérande immédiate et génère un masque entreposé dans le registre *XMM0*.

Mnémonique

Instruction	Opcode	Description
PCMPESTRM *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 60h /r *imm8*	Cette instruction permet d'effectuer une comparaison ainsi qu'un processus de fragments de deux chaine de caractères basé sur une valeur d'encodage indiquer par un opérande immédiate et génère un masque entreposé dans le registre *XMM0*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 99 à 101.

Assembleur 80x86 — PCMPGTB

INTEL Pentium Pro+ — *Packed Compare for Greater Than Bytes*

Syntaxe

PCMPGTB *dest,source*

Description

Cette instruction permet d'effectuer une comparaison sur un paquet d'octets et fixe la valeur de chacun d'eux à FFh si l'opérande destinataire est supérieur à l'opérande source sinon à 00h.

Algorithme

```
SI dest(7..0) > source(7..0) ALORS
  dest(7..0) ← FFh
SINON
  dest(7..0) ← 00h
FIN SI
SI dest(15..8) > source(15..8) ALORS
  dest(15..8) ← FFh
SINON
  dest(15..8) ← 00h
FIN SI
SI dest(23..16) > source(23..16) ALORS
  dest(23..16) ← FFh
SINON
  dest(23..16) ← 00h
FIN SI
SI dest(31..24) > source(31..24) ALORS
  dest(31..24) ← FFh
SINON
  dest(31..24) ← 00h
FIN SI
```

```
SI dest(39..32) > source(39..32) ALORS
  dest(39..32) ← FFh
SINON
  dest(39..32) ← 00h
FIN SI
SI dest(47..40) > source(47..40) ALORS
  dest(47..40) ← FFh
SINON
  dest(47..40) ← 00h
FIN SI
SI dest(55..48) > source(55..48) ALORS
  dest(55..48) ← FFh
SINON
  dest(55..48) ← 00h
FIN SI
SI dest(63..56) > source(63..56) ALORS
  dest(63..56) ← FFh
SINON
  dest(63..56) ← 00h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PCMPGTB *mm,mm/m64*	0Fh 64h /r	Cette instruction permet d'effectuer une comparaison sur un paquet d'octets et fixe la valeur de chacun d'eux à FFh si l'opérande destinataire est supérieur à l'opérande source sinon à 00h.
PCMPGTB *xmm1, xmm2/m128*	66h 0Fh 64h /r	Cette instruction permet d'effectuer une comparaison sur un paquet d'octets et fixe la valeur de chacun d'eux à FFh si l'opérande destinataire est supérieur à l'opérande source sinon à 00h.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 102 à 106.

Assembleur 80x86 — PCMPGTD

INTEL Pentium Pro+

Packed Compare for Greater Than Dwords

Syntaxe

PCMPGTD *dest,source*

Description

Cette instruction permet d'effectuer une comparaison sur un paquet de double mots et fixe la valeur de chacun d'eux à FFFFFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 00000000h.

Algorithme

SI *dest*(31..0) > *source*(31..0) **ALORS**
  dest(31..0) ← FFFFFFFFh
SINON
  dest(31..0) ← 00000000h
FIN SI
SI dest(63..32) > *source*(63..32) **ALORS**
  dest(63..32) ← FFFFFFFFh
SINON
  dest(63..32) ← 00000000h
FIN SI

Mnémonique

Instruction	Opcode	Description
PCMPGTD *mm,mm/m64*	0Fh 66h /r	Cette instruction permet d'effectuer une comparaison sur un paquet de double mots et fixe la valeur de

		chacun d'eux à FFFFFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 00000000h.
PCMPGTD *xmm1, xmm2/m128*	66h 0Fh 66h /r	Cette instruction permet d'effectuer une comparaison sur un paquet de double mots et fixe la valeur de chacun d'eux à FFFFFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 00000000h.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 102 à 106.

Assembleur 80x86 — PCMPGTQ

Nehalem (SSE4.2)+ — *Compare Packed Data for Greater Than*

Syntaxe

PCMPGTQ *dest,source*

Description

Cette instruction permet d'effectuer une comparaison sur un paquet de quadruple mots et fixe la valeur de chacun d'eux à FFFFFFFFFFFFFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 0000000000000000h.

Algorithme

SI *dest*(63..0) > *source*(63..0) **ALORS**
  dest(63..0) ← FFFFFFFFFFFFFFFFh
SINON
  dest(63..0) ← 0
FIN SI
SI *dest*(127..64) > *source*(127..64) **ALORS**
  dest(127..64) ← FFFFFFFFFFFFFFFFh
SINON
  dest(127..64) ← 0
FIN SI

Mnémonique

Instruction	Opcode	Description
PCMPGTQ *xmm1,xmm2/m128*	66h 0Fh 38h 37h /r	Cette instruction permet d'effectuer une comparaison sur un paquet de quadruple mots et fixe la valeur de

		chacun d'eux à FFFFFFFFFFFFFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 0000000000000000h.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 107 à 109.

Assembleur 80x86 — PCMPGTW

INTEL Pentium Pro+

Packed Compare for Greater Than Words

Syntaxe

PCMPGTW *dest*,*source*

Description

Cette instruction permet d'effectuer une comparaison sur un paquet de mots et fixe la valeur de chacun d'eux à FFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 0000h.

Algorithme

```
SI dest(15..0) > source(15..0) ALORS
  dest(15..0) ← FFFFh
SINON
  dest(15..0) ← 0000h
SI dest(31..16) > source(31..16) ALORS
FIN SI
  dest(31..16) ← FFFFh
SINON
  dest(31..16) ← 0000h
FIN SI
SI dest(47..32) > source(47..32) ALORS
  dest(47..32) ← FFFFh
SINON
  dest(47..32) ← 0000h
FIN SI
SI dest(63..48) > source(63..48) ALORS
  dest(63..48) ← FFFFh
SINON
  dest(63..48) ← 0000h
```

FIN SI

Mnémonique

Instruction	Opcode	Description
PCMPGTW *mm, mm/m64*	0Fh 65h /r	Cette instruction permet d'effectuer une comparaison sur un paquet de mots et fixe la valeur de chacun d'eux à FFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 0000h.
PCMPGTW *xmm1, xmm2/m128*	66h 0Fh 65h /r	Cette instruction permet d'effectuer une comparaison sur un paquet de mots et fixe la valeur de chacun d'eux à FFFFFh si l'opérande destinataire est supérieur à l'opérande source sinon à 0000h.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 102 à 106.

Assembleur 80x86 — PCMPISTRI

Nehalem (SSE4.2)+ — *Packed Compare Implicit Length Strings, Return Index*

Syntaxe

PCMPISTRI *dest,source,immediat*

Description

Cette instruction permet d'effectuer une comparaison ainsi qu'un processus de fragments de deux chaines de caractères basé sur une valeur d'encodage implicite indiquer par un opérande immédiate et génère un index entreposé dans le registre *ECX*.

Mnémonique

Instruction	Opcode	Description
PCMPISTRI *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 63h /r *imm8*	Cette instruction permet d'effectuer une comparaison ainsi qu'un processus de fragments de deux chaines de caractères basé sur une valeur d'encodage implicite indiquer par un opérande immédiate et génère un index entreposé dans le registre *ECX*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 110 à 111.

Assembleur 80x86 — PCMPISTRM

Nehalem (SSE4.2)+ — *Packed Compare Implicit Length Strings, Return Mask*

Syntaxe

PCMPISTRM *dest,source,immediat*

Description

Cette instruction permet d'effectuer une comparaison ainsi qu'un processus de fragments de deux chaine de caractères basé sur une valeur d'encodage implicite indiquer par un opérande immédiate et génère un masque entreposé dans le registre *XMM0*.

Mnémonique

Instruction	Opcode	Description
PCMPISTRM *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 62h /r *imm8*	Cette instruction permet d'effectuer une comparaison ainsi qu'un processus de fragments de deux chaine de caractères basé sur une valeur d'encodage implicite indiquer par un opérande immédiate et génère un masque entreposé dans le registre *XMM0*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 113 à 115.

Assembleur 80x86 — PDISTIB

Cyrix 6x86MX (EMMX)+ — *Packed Distribution*

Syntaxe

PDISTIB *destination, source*

Description

Cette instruction permet d'effectuer le calcul de la distance entre des octets de deux opérandes, le résultat de l'addition d'octet de l'opérande de destination et la saturation du résultat.

Algorithme

destination(7..0) ← SaturateToUnsignedByte(*destination*(7..0) + ABS(*destination*(7..0) - *source*(7..0)))
destination(15..8) ← SaturateToUnsignedByte(*destination*(15..8) + ABS(*destination*(15..8) - *source*(15..8)))
destination(23..16) ← SaturateToUnsignedByte(*destination*(23..16) + ABS(*destination*(23..16) - *source*(23..16)))
destination(31..24) ← SaturateToUnsignedByte(*destination*(31..24) + ABS(*destination*(31..24) - *source*(31..24)))
destination(39..32) ← SaturateToUnsignedByte(*destination*(39..32) + ABS(*destination*(39..32) - *source*(39..32)))
destination(47..40) ← SaturateToUnsignedByte(*destination*(47..40) + ABS(*destination*(47..40) - *source*(47..40)))
destination(55..48) ← SaturateToUnsignedByte(*destination*(55..48) + ABS(*destination*(55..48) - *source*(55..48)))
destination(63..56) ← SaturateToUnsignedByte(*destination*(63..56) + ABS(*destination*(63..56) - *source*(63..56)))

Mnémonique

Instruction	Opcode	Description
PDISTIB *mm,m64*	0Fh 54h /r	Cette instruction permet d'effectuer le calcul de la distance entre des octets de deux opérandes,

		le résultat de l'addition d'octet de l'opérande de destination et la saturation du résultat.

Assembleur 80x86 PEXTRB

SSE4.1+

Paquet Extract Byte

Syntaxe

PEXTRB *dest*,*source*,*count*

Description

Cette instruction permet de copier l'octet de l'opérande source spécifié par un compteur d'opérande dans l'opérande de destination.

Algorithme

```
SEL ← count(3..0)
TEMP ← (source >> SEL x 8) ∩ FFh
SI dest = Mem8 ALORS
  Mem8 ← TEMP(7..0)
SINON SI mode 64 bits et registre 64 bits selectionné ALORS
  R64(7..0) ← TEMP(7..0)
  r64(63..8) ← 00000000000000h
SINON
  R32(7..0) ← TEMP(7..0)
  r32(31..8) ← 000000h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PEXTRB *reg/m8,xmm2, imm8*	66h 0Fh 3Ah 14h /r ib	Cette instruction permet de copier l'octet de l'opérande source spécifié par un compteur d'opérande dans

		l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 116 à 118.

Assembleur 80x86 PEXTRD

SSE4.1+ *Paquet Extract Doubleword*

Syntaxe

PEXTRD *dest,source,count*

Description

Cette instruction permet de copier le double mot de l'opérande source spécifié par un compteur d'opérande dans l'opérande de destination.

Algorithme

```
SEL ← count(1..0)
TEMP ← (source >> SEL x 32) ∩ FFFFFFFFh
dest ← TEMP
```

Mnémonique

Instruction	Opcode	Description
PEXTRD *r/m32,xmm2,imm8*	66h 0Fh 3Ah 16h /r *ib*	Cette instruction permet de copier le double mot de l'opérande source spécifié par un compteur d'opérande dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 116 à 118.

Assembleur 80x86 PEXTRQ

SSE4.1+

Paquet Extract Quadword

Syntaxe

PEXTRQ *dest,source,count*

Description

Cette instruction permet de copier le quadruple mot de l'opérande source spécifié par un compteur d'opérande dans l'opérande de destination.

Algorithme

SEL ← *count*(0)
TEMP ← (*source* >> SEL x 64)
dest ← TEMP

Mnémonique

Instruction	Opcode	Description
PEXTRQ *r/m64,xmm2,imm8*	66h (*REX.W*) 0Fh 3Ah 16h /r *ib*	Cette instruction permet de copier le quadruple mot de l'opérande source spécifié par un compteur d'opérande dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 116 à 118.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

PEXTRW

Extract Word

Syntaxe

PEXTRW *dest*,*source*,*count*

Description

Cette instruction permet de copier le mot de l'opérande source spécifié par un compteur d'opérande dans l'opérande de destination.

Algorithme

SI opérande source est 64 bits **ALORS**
  SEL ← *count* ∩ 3h
  TEMP ← (*source* >> (SEL x 16)) ∩ FFFFh
  dest(15..0) ← TEMP(15..0)
  dest(31..16) ← 0000h
SINON SI opérande source est 128 bits **ALORS**
  SEL ← *count* ∩ 7h
  TEMP ← (*source* >> (SEL x 16)) ∩ FFFFh
  dest(15..0) ← TEMP(15..0)
  dest(31..16) ← 0000h
FIN SI

Mnémonique

Instruction	Opcode	Description

PEXTRW *reg32,mmreg,imm8*	0Fh C5h /r *imm8*	Cette instruction permet de copier le mot de l'opérande source spécifié par un compteur d'opérande dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 118 à 121.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

PF2ID

Convert Packed Float Point to 32 bits

Syntaxe

PF2ID *dest,source*

Description

Cette instruction permet de convertir un double mots d'un format réel à un format d'entier 32 bits.

Algorithme

SI *source*(31..0) > 2^{31} **ALORS**
  dest(31..0) ← 7FFFFFFFh
FIN SI
SI *source*(31..0) <= -2^{31} **ALORS**
  dest(31..0) ← 80000000h
FIN SI
dest(31..0) ← truncate(*source*(31..0))
SI *source*(63..32) > 2^{31} **ALORS**
  dest(63..32) ← 7FFFFFFFh
FIN SI
SI *source*(63..32) <= -2^{31} **ALORS**
  dest(63..32) ← 80000000h
FIN SI
dest(63..32) ← truncate(*source*(63..32))

Mnémonique

Instruction	Opcode	Description
PF2ID *mm,mm/m64*	0Fh 0Fh 1Dh /r	Cette instruction permet de convertir un double mots d'un format réel à un format d'entier 32

		bits.

Assembleur 80x86 — PF2IW

AMD 3DNow!+

Packed Float Point to Integer Word conversion with sign extend

Syntaxe

PF2IW *dest*,*source*

Description

Cette instruction permet de convertir un double mots d'un format réel à un format d'entier 16 bits.

Algorithme

SI *source*(31..0) >= 2^{15} **ALORS**
  dest(31..0) ← 00007FFFh
SINON SI *source*(31..0) <= -2^{15} **ALORS**
  dest(31..0) ← FFFF8000h
SINON
  dest(31..0) ← [*source*(31..0)]
FIN SI
SI *source*(63..32) >= 2^{15} **ALORS**
  dest(63..32) ← 00007FFFh
SINON SI *source*(63..32) <= -2^{15} **ALORS**
  dest(63..32) ← FFFF8000h
SINON
  dest(63..32) ← [*source*(63..32)]
FIN SI

Mnémonique

Instruction	Opcode	Description

PF2IW *mm,mm/m64*	0Fh 0Fh 1Ch /r	Cette instruction permet de convertir un double mots d'un format réel à un format d'entier 16 bits.

Références

AMD Extensions to the 3DNow! and MMX Instruction Sets Manual, Edition Advanced Micro Devices, March 2000, Publication No. 22466D, page 12.

Assembleur 80x86 — PFACC

AMD 3DNow!+ — *Float Point Accumulate*

Syntaxe

PFACC *dest,source*

Description

Cette instruction permet d'effectuer l'accumulation de double mots.

Algorithme

dest(31..0) ← *dest*(31..0) + *dest*(63..32)
dest(63..32) ← *source*(31..0) + *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PFACC *mm,mm/m64*	0Fh 0Fh AEh /r	Cette instruction permet d'effectuer l'accumulation de double mots.

Assembleur 80x86 — PFADD

AMD 3DNow!+ — *Float Point Addition*

Syntaxe

PFADD *dest,source*

Description

Cette instruction permet d'effectuer l'addition de double mots.

Algorithme

dest(31..0) ← *dest*(31..0) + *source*(31..0)
dest(63..32) ← *dest*(63..32) + *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PFADD *mm,mm/m64*	0Fh 0Fh 9Eh /r	Cette instruction permet d'effectuer l'addition de double mots.

Assembleur 80x86 — PFCMPEQ

AMD 3DNow!+ — *Float Point Compare Equal*

Syntaxe

PFCMPEQ *dest,source*

Description

Cette instruction permet d'effectuer la comparaison d'égalité d'un paquet de double mots contenu dans des opérandes de 64 bits.

Algorithme

```
SI dest(31..0) = source(31..0) ALORS
  dest(31..0) ← FFFFFFFFh
SINON
  dest(31..0) ← 00000000h
FIN SI
SI dest(63..32) = source(63..32) ALORS
  dest(63..32) ← FFFFFFFFh
SINON
  dest(63..32) ← 00000000h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PFCMPEQ *mm,mm/m64*	0Fh 0Fh B0h /r	Cette instruction permet d'effectuer la comparaison d'égalité d'un paquet de double mots contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFCMPGE

AMD 3DNow!+ — *Float Point Greater or Equal to*

Syntaxe

PFCMPGE *dest,source*

Description

Cette instruction permet d'effectuer la comparaison d'égalité ou de supériorité d'un paquet de double mots contenu dans des opérandes de 64 bits.

Algorithme

```
SI dest(31..0) >= source(31..0) ALORS
  dest(31..0) ← FFFFFFFFh
SINON
  dest(31..0) ← 00000000h
FIN SI
SI dest(63..32) >= source(63..32) ALORS
  dest(63..32) ← FFFFFFFFh
SINON
  dest(63..32) ← 00000000h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PFCMPGE *mm,mm/m64*	0Fh 0Fh 90h /r	Cette instruction permet d'effectuer la comparaison d'égalité ou de supériorité d'un paquet de double mots contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFCMPGT

AMD 3DNow!+ — *Float Point Greater*

Syntaxe

PFCMPGT *dest,source*

Description

Cette instruction permet d'effectuer la comparaison de supériorité d'un paquet de double mots contenu dans des opérandes de 64 bits.

Algorithme

SI *dest*(31..0) > *source*(31..0) **ALORS**
  dest(31..0) ← FFFFFFFFh
SINON
  dest(31..0) ← 00000000h
FIN SI
SI *dest*(63..32) > *source*(63..32) **ALORS**
  dest(63..32) ← FFFFFFFFh
SINON
  dest(63..32) ← 00000000h
FIN SI

Mnémonique

Instruction	Opcode	Description
PFCMPGT *mm,mm/m64*	0Fh 0Fh A0h /r	Cette instruction permet d'effectuer la comparaison de supériorité d'un paquet de double mots contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFMAX

AMD 3DNow!+ — *Float Point Maximum*

Syntaxe

PFMAX *dest,source*

Description

Cette instruction permet de retourner la valeur maximal de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

SI *source*(31..0) > *dest*(31..0) **ALORS**
  dest(31..0) ← *source*(31..0)
FIN SI
SI *source*(63..32) > *dest*(63..32) **ALORS**
  dest(63..32) ← *source*(63..32)
FIN SI

Mnémonique

Instruction	Opcode	Description
PFMAX *mm,mm/m64*	0Fh 0Fh A4h /r	Cette instruction permet de retourner la valeur maximal de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFMIN

AMD 3DNow!+ — *Float Point Minimum*

Syntaxe

PFMIN *dest,source*

Description

Cette instruction permet de retourner la valeur minimal de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

SI *source*(31..0) < *dest*(31..0) **ALORS**
  dest(31..0) ← *source*(31..0)
FIN SI
SI *source*(63..32) < *dest*(63..32) **ALORS**
  dest(63..32) ← *source*(63..32)
FIN SI

Mnémonique

Instruction	Opcode	Description
PFMIN *mm,mm/m64*	0Fh 0Fh 94h /r	Cette instruction permet de retourner la valeur minimal de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFMUL

AMD 3DNow!+ — *Float Point Multiplication*

Syntaxe

PFMUL *dest,source*

Description

Cette instruction permet d'effectuer la multiplication de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

dest(31..0) ← *dest*(31..0) * *source*(31..0)
dest(63..32) ← *dest*(63..32) * *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PFMUL *mm,mm/m64*	0Fh 0Fh B4h /r	Cette instruction permet d'effectuer la multiplication de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFNACC

AMD 3DNow!+ — *Float Point Negative Accumulate*

Syntaxe

PFNACC *dest,source*

Description

Cette instruction permet d'effectuer l'accumulation négative de double mots.

Algorithme

dest(31..0) ← *dest*(31..0) - *dest*(63..32)
dest(63..32) ← *source*(31..0) - *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PFNACC *mm,mm/m64*	0Fh 0Fh 8Ah /r	Cette instruction permet d'effectuer l'accumulation négative de double mots.

Assembleur 80x86 — PFPNACC

AMD 3DNow!+ — *Packed Floating Point Mixed Positive*

Syntaxe

PFPNACC *dest,source*

Description

Cette instruction permet d'effectuer un mixe d'accumulation positive et négative de double mots.

Algorithme

dest(31..0) ← *dest*(31..0) - *dest*(63..32)
dest(63..32) ← *source*(31..0) + *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PFPNACC *mm,mm/m64*	0Fh 0Fh 8Eh /r	Cette instruction permet d'effectuer un mixe d'accumulation positive et négative de double mots.

Assembleur 80x86 — PFRCP

AMD 3DNow!+ — *Float Point Reciprocal Approximation*

Syntaxe

PFRCP *dest,source*

Description

Cette instruction permet d'effectuer la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

dest(31..0) ← Reciprocal(*source*(31..0))
dest(63..32) ← Reciprocal(*source*(63..32))

Mnémonique

Instruction	Opcode	Description
PFRCP *mm,mm/m64*	0Fh 0Fh 96h /r	Cette instruction permet d'effectuer la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFRCPIT1

AMD 3DNow!+ — *Float Point Reciprocal, First iteration Step*

Syntaxe

PFRCIT1 *dest,source*

Description

Cette instruction permet d'effectuer la première étape d'itération de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

```
dest(31..0) ← First_Step_Reciprocal(source(31..0))
dest(63..32) ← First_Step_Reciprocal(source(63..32))
```

Mnémonique

Instruction	Opcode	Description
PFRCIT1 *mm,mm/m64*	0Fh 0Fh A6h /r	Cette instruction permet d'effectuer la première étape d'itération de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86

AMD 3DNow!+

PFRCPIT2

Float Point Reciprocal, Second iteration Step

Syntaxe

PFRCPIT2 *dest,source*

Description

Cette instruction permet d'effectuer la deuxième étape d'itération de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

dest(31..0) ← Second_Step_Reciprocal(*source*(31..0))
dest(63..32) ← Second_Step_Reciprocal(*source*(63..32))

Mnémonique

Instruction	Opcode	Description
PFRCPIT2 *mm,mm/m64*	0Fh 0Fh B6h /r	Cette instruction permet d'effectuer la deuxième étape d'itération de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFRSQIT1

AMD 3DNow!+

Float Point Reciprocal Square Root, First iteration step

Syntaxe

PFRSQIT1 *dest,source*

Description

Cette instruction permet d'effectuer la première étape d'itération de la racine carré de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

dest(31..0) ← First_Step_Reciprocal_Square_Root(*source*(31..0))
dest(63..32) ← First_Step_Reciprocal_Square_Root(*source*(63..32))

Mnémonique

Instruction	Opcode	Description
PFRSQIT1 *mm,mm/m64*	0Fh 0Fh A7h /r	Cette instruction permet d'effectuer la première étape d'itération de la racine carré de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFRSQRT

AMD 3DNow!+ — *Float Point Reciprocal Square Root Approximation*

Syntaxe

PFRSQRT *dest,source*

Description

Cette instruction permet d'effectuer la racine carré approximative de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

```
dest(31..0) ← Reciprocal_Square_Root(source(31..0))
dest(63..32) ← Reciprocal_Square_Root(source(63..32))
```

Mnémonique

Instruction	Opcode	Description
PFRSQRT *mm,mm/m64*	0Fh 0Fh 97h /r	Cette instruction permet d'effectuer la racine carré approximative de la réciproque de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFSUB

AMD 3DNow!+ — *Float Point Subtraction*

Syntaxe

PFSUB *dest,source*

Description

Cette instruction permet d'effectuer la soustraction de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

dest(31..0) ← *dest*(31..0) - *source*(31..0)
dest(63..32) ← *dest*(63..32) - *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PFSUB *mm,mm/m64*	0Fh 0Fh 9Ah /r	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 — PFSUBR

AMD 3DNow!+ — *Float Point Reverse Subtraction*

Syntaxe

PFSUBR *dest,source*

Description

Cette instruction permet d'effectuer la soustraction inversé de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

dest(31..0) ← *source*(31..0) - *dest*(31..0)
dest(63..32) ← *source*(63..32) - *dest*(63..32)

Mnémonique

Instruction	Opcode	Description
PFSUBR *mm,mm/m64*	0Fh 0Fh AAh /r	Cette instruction permet d'effectuer la soustraction inversé de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Assembleur 80x86 PHADDD

SSSE3+

Packed Horizontal Add Doubleword

Syntaxe

PHADDD *dest,source*

Description

Cette instruction permet d'effectuer l'addition de double mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Algorithme

SI taille d'opérande = 64 bits **ALORS**
  dest(31..0) ← *dest*(63..32) + *dest*(31..0)
  dest(63..32) ← *source*(63..32) + *source*(31..0)
SINON SI taille d'opérande = 128 bits **ALORS**
  dest(31..0) ← *dest*(63..32) + *dest*(31..0)
  dest(63..32) ← *dest*(127..96) + *dest*(95..64)
  dest(95..64) ← *source*(63..32) + *source*(31..0)
  dest(127..96) ← *source*(127..96) + *source*(95..64)
FIN SI

Mnémonique

Instruction	Opcode	Description
PHADDD *mm1,mm2/m64*	0Fh 38h 02h /r	Cette instruction permet d'effectuer l'addition de double mot entier horizontallement adjacent d'un opérande source et opérande

		destination et met le résultat dans l'opérande destination.
PHADDD *xmm1,xmm2/m128*	66h 0Fh 38h 02h /r	Cette instruction permet d'effectuer l'addition de double mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 123 à 125.

Assembleur 80x86 PHADDSW

SSSE3+

Packed Horizontal Add and Saturate Word

Syntaxe

PHADDSW *dest*,*source*

Description

Cette instruction permet d'effectuer l'addition de mot entier horizontallement adjacent avec saturation d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Algorithme

SI taille d'opérande = 64 bits **ALORS**
dest(15..0) ← SaturateToSignedWord((*dest*(31..16) + *dest*(15..0))
dest(31..16) ← SaturateToSignedWord(*dest*(63..48) + *dest*(47..32))
dest(47..32) ← SaturateToSignedWord(*source*(31..16) + *source*(15..0))
dest(63..48) ← SaturateToSignedWord(*source*(63..48) + *source*(47..32))
SINON SI taille d'opérande = 128 bits **ALORS**
dest(15..0) ← SaturateToSignedWord(*dest*(31..16) + *dest*(15..0))
dest(31..16) ← SaturateToSignedWord(*dest*(63..48) + *dest*(47..32))
dest(47..32) ← SaturateToSignedWord(*dest*(95..80) + *dest*(79..64))
dest(63..48) ← SaturateToSignedWord(*dest*(127..112) + *dest*(111..96))
dest(79..64) ← SaturateToSignedWord(*source*(31..16) + *source*(15..0))
dest(95..80) ← SaturateToSignedWord(*source*(63..48) + *source*(47..32))
dest(111..96) ← SaturateToSignedWord(*source*(95..80) + *source*(79..64))
dest(127..112) ← SaturateToSignedWord(*source*(127..112) + *source*(111..96))
FIN SI

Mnémonique

Instruction	Opcode	Description
PHADDSW *mm1,mm2/m64*	0Fh 38h 03h /r	Cette instruction permet d'effectuer l'addition de mot entier horizontallement adjacent avec saturation d'un opérande source et opérande destination et met le résultat dans l'opérande destination.
PHADDSW *xmm1,xmm2/m128*	66h 0Fh 38h 03h /r	Cette instruction permet d'effectuer l'addition de mot entier horizontallement adjacent avec saturation d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 126 à 128.

Assembleur 80x86 PHADDW

SSSE3+

Packed Horizontal Add Word

Syntaxe

PHADDW *dest*,*source*

Description

Cette instruction permet d'effectuer l'addition de mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Algorithme

SI taille d'opérande = 64 bits **ALORS**
  dest(15..0) ← *dest*(31..16) + *dest*(15..0)
  dest(31..16) ← *dest*(63..48) + *dest*(47..32)
  dest(47..32) ← *source*(31..16) + *source*(15..0)
  dest(63..48) ← *source*(63..48) + *source*(47..32)
SINON SI taille d'opérande = 128 bits **ALORS**
  dest(15..0) ← *dest*(31..16) + *dest*(15..0)
  dest(31..16) ← *dest*(63..48) + *dest*(47..32)
  dest(47..32) ← *dest*(95..80) + *dest*(79..64)
  dest(63..48) ← *dest*(127..112) + *dest*(111..96)
  dest(79..64) ← *source*(31..16) + *source*(15..0)
  dest(95..80) ← *source*(63..48) + *source*(47..32)
  dest(111..96) ← *source*(95..80) + *source*(79..64)
  dest(127..112) ← *source*(127..112) + *source*(111..96)
FIN SI

Mnémonique

Instruction	Opcode	Description
PHADDW *mm1,mm2/m64*	0Fh 38h 01h /r	Cette instruction permet d'effectuer l'addition de mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.
PHADDW *xmm1,xmm2/m128*	66h 0Fh 38h 01h /r	Cette instruction permet d'effectuer l'addition de mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 123 à 125.

Assembleur 80x86 PHMINPOSUW

SSE4.1+

Packed Horizontal Word Minimum

Syntaxe

PHMINPOSUW *dest,source*

Description

Cette instruction permet de demander le minimum des valeurs de mot de l'opérande source et place dans la partie basse de l'opérande le mot.

Algorithme

```
INDEX ← 0
MIN ← source(15..0)
SI source(31..16) < MIN ALORS
  INDEX ← 1
  MIN ← SRC(31..16)
FIN SI
SI source(47..32) < MIN ALORS
  INDEX ← 2
  MIN ← source(47..32)
FIN SI
SI source(63..48) < MIN ALORS
  INDEX ← 3
  MIN ← source(63..48)
FIN SI
SI source(79..64) < MIN ALORS
  INDEX ← 4
  MIN ← source(79..64)
FIN SI
SI source(95..80) < MIN ALORS
  INDEX ← 5
  MIN ← source(95..80)
```

```
FIN SI
SI source(111..96) < MIN ALORS
  INDEX ← 6
  MIN ← source(111..96)
FIN SI
SI source(127..112) < MIN ALORS
  INDEX ← 7
  MIN ← source(127..112)
FIN SI
dest(15..0) ← MIN
dest(18..16) ← INDEX
dest(127..19) ← 00000000000000000000000000000h
```

Mnémonique

Instruction	Opcode	Description
PHMINPOSUW *xmm1,xmm2/m128*	66h 0Fh 38h 41h /r	Cette instruction permet de demander le minimum des valeurs de mot de l'opérande source et place dans la partie basse de l'opérande le mot.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 129 à 131.

Assembleur 80x86 PHSUBD

SSSE3+

Packed Horizontal Subtract Doubleword

Syntaxe

PHSUBD *dest*,*source*

Description

Cette instruction permet d'effectuer la soustraction de double mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Algorithme

SI taille d'opérande = 64 bits **ALORS**
  dest(31..0) ← *dest*(31..0) - *dest*(63..32)
  dest(63..32) ← *source*(31..0) - *source*(63..32)
SINON SI taille d'opérande = 128 bits **ALORS**
  dest(31..0) ← *dest*(31..0) - *dest*(63..32)
  dest(63..32) ← *dest*(95..64) - *dest*(127..96)
  dest(95..64) ← *source*(31..0) - *source*(63..32)
  dest(127..96) ← *source*(95..64) - *source*(127..96)
FIN SI

Mnémonique

Instruction	Opcode	Description
PHSUBD *mm1*,*mm2/m64*	0Fh 38h 06h /r	Cette instruction permet d'effectuer la soustraction de double mot entier horizontallement adjacent d'un opérande source et opérande

		destination et met le résultat dans l'opérande destination.
PHSUBD *xmm1,xmm2/m128*	66h 0Fh 38h 06h /r	Cette instruction permet d'effectuer la soustraction de double mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 132 à 135.

Assembleur 80x86 PHSUBSW

SSSE3+

Packed Horizontal Subtract and Saturate

Syntaxe

```
PHSUBSW dest,source
```

Description

Cette instruction permet d'effectuer la soustraction de mot entier horizontallement adjacent avec saturation d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Algorithme

```
SI taille d'opérande = 64 bits ALORS
  dest(15..0) ← SaturateToSignedWord(dest(15..0) - dest(31..16))
  dest(31..16) ← SaturateToSignedWord(dest(47..32) - dest(63..48))
  dest(47..32) ← SaturateToSignedWord(source(15..0) - source(31..16))
  dest(63..48) ← SaturateToSignedWord(source(47..32) - source(63..48))
SINON SI taille d'opérande = 128 bits ALORS
  dest(15..0) ← SaturateToSignedWord(dest(15..0) - dest(31..16))
  dest(31..16) ← SaturateToSignedWord(dest(47..32) - dest(63..48))
  dest(47..32) ← SaturateToSignedWord(dest(79..64) - dest(95..80))
  dest(63..48) ← SaturateToSignedWord(dest(111..96) - dest(127..112))
  dest(79..64) ← SaturateToSignedWord(source(15..0) - source(31..16))
  dest(95..80) ← SaturateToSignedWord(source(47..32) - source(63..48))
  dest(111..96) ← SaturateToSignedWord(source(79..64) - source(95..80))
  dest(127..112) ← SaturateToSignedWord(source(111..96) - source(127..112))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PHSUBSW *mm1,mm2/m64*	0Fh 38h 07h /r	Cette instruction permet d'effectuer la soustraction de mot entier horizontallement adjacent avec saturation d'un opérande source et opérande destination et met le résultat dans l'opérande destination.
PHSUBSW *xmm1,xmm2/m128*	66h 0Fh 38h 07h /r	Cette instruction permet d'effectuer la soustraction de mot entier horizontallement adjacent avec saturation d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 136 à 138.

Assembleur 80x86 PHSUBW

SSSE3+ *Packed Horizontal Subtract Word*

Syntaxe

PHSUBW *dest,source*

Description

Cette instruction permet d'effectuer la soustraction de mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Algorithme

SI taille d'opérande = 64 bits **ALORS**
  dest(15..0) ← *dest*(15..0) - *dest*(31..16)
  dest(31..16) ← *dest*(47..32) - *dest*(63..48)
  dest(47..32) ← *source*(15..0) - *source*(31..16)
  dest(63..48) ← *source*(47..32) - *source*(63..48)
SINON SI taille d'opérande = 128 bits **ALORS**
  dest(15..0) ← *dest*(15..0) - *dest*(31..16)
  dest(31..16) ← *dest*(47..32) - *dest*(63..48)
  dest(47..32) ← *dest*(79..64) - *dest*(95..80)
  dest(63..48) ← *dest*(111..96) - *dest*(127..112)
  dest(79..64) ← *source*(15..0) - *source*(31..16)
  dest(95..80) ← *source*(47..32) - *source*(63..48)
  dest(111..96) ← *source*(79..64) - *source*(95..80)
  dest(127..112) ← *source*(111..96) - *source*(127..112)
FIN SI

Mnémonique

Instruction	Opcode	Description
PHSUBW *mm1,mm2/m64*	0Fh 38h 05h /r	Cette instruction permet d'effectuer la soustraction de mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.
PHSUBW *xmm1,xmm2/m128*	66h 0Fh 38h 05h /r	Cette instruction permet d'effectuer la soustraction de mot entier horizontallement adjacent d'un opérande source et opérande destination et met le résultat dans l'opérande destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 132 à 135.

Assembleur 80x86 — PI2FD

AMD 3DNow!+ — *Integer To Float Point*

Syntaxe

PI2FD *dest,source*

Description

Cette instruction permet de convertir un paquet de format d'entier 32 bits à double mots d'un format réel.

Algorithme

dest(31..0) ← (réel) *source*(31..0)
dest(63..32) ← (réel) *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PI2FD *mm,mm/m64*	0Fh 0Fh 0Dh /r	Cette instruction permet de convertir un paquet de format d'entier 32 bits à double mots d'un format réel.

Assembleur 80x86 — PI2FW

AMD 3DNow!+ — *Integer To Float Point Word*

Syntaxe

PI2FW *dest,source*

Description

Cette instruction permet de convertir un paquet de format d'entier 16 bits à double mots d'un format réel.

Algorithme

dest(31..0) ← (réel) *source*(15..0)
dest(63..32) ← (réel) *source*(47..32)

Mnémonique

Instruction	Opcode	Description
PI2FW *mm,mm/m64*	0Fh 0Fh 0Ch /r	Cette instruction permet de convertir un paquet de format d'entier 16 bits à double mots d'un format réel.

Assembleur 80x86 PINSRB

SSE4.1+ *Insert Byte*

Syntaxe

PINSRW *dest,source,compteur*

Description

Cette instruction permet d'effectuer une copie d'un octet d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.

Algorithme

SEL ← *compteur*(3..0) MASK ← (0FFh << (SEL x 8)) TEMP ← (((*source*(7..0) << (SEL x 8)) ⋂ MASK) *dest* ← ((*dest* ⋂ ¬ MASK) U TEMP)

Mnémonique

Instruction	Opcode	Description
PINSRB *xmm1,r32/m8, imm8*	66h 0Fh 3Ah 20h /r *ib*	Cette instruction permet d'effectuer une copie d'un octet d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 139 à 141.*

Assembleur 80x86 PINSRD

SSE4.1+ *Insert Doubleword*

Syntaxe

PINSRD *dest*,*source*,*compteur*

Description

Cette instruction permet d'effectuer une copie d'un double mot d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.

Algorithme

SEL ← *compteur* (1..0)
MASK ← (0FFFFFFFFh << (SEL x 32))
TEMP ← (((*source* << (SEL x 32)) AND MASK)
dest ← ((*dest* ⋂ ¬ MASK) U TEMP)

Mnémonique

Instruction	Opcode	Description
PINSRD *xmm1,r/m32, imm8*	66h 0Fh 3Ah 22h /r *ib*	Cette instruction permet d'effectuer une copie d'un double mot d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 139 à 141.

Assembleur 80x86 PINSRQ

SSE4.1+ *Insert Quadword*

Syntaxe

PINSRQ *dest,source,compteur*

Description

Cette instruction permet d'effectuer une copie d'un quadruple mot d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.

Algorithme

```
SEL ← compteur(0)
MASK ← (0FFFFFFFFFFFFFFFFh << (SEL x 64))
TEMP ← (((source << (SEL x 64)) ⋂ MASK)
dest ← ((dest ⋂ ¬ MASK) U TEMP)
```

Mnémonique

Instruction	Opcode	Description
PINSRQ *xmm1,r/m64, imm8*	66h *REX.W* 0Fh 3Ah 22h */r ib*	Cette instruction permet d'effectuer une copie d'un quadruple mot d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 139 à 141.

Assembleur 80x86 — PINSRW

INTEL Pentium III (KNI/MMX2)+

Insert Word

Syntaxe

PINSRW *dest,source,compteur*

Description

Cette instruction permet d'effectuer une copie d'un mot d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.

Algorithme

```
SI opérande dest est 128 bits ALORS
  compteur ← compteur ⋂ 07h
  EVALUER CAS compteur
    CAS 0: masque ← 0000000000000000000000000000FFFFh
    CAS 1: masque ← 00000000000000000000000FFFF0000h
    CAS 2: masque ← 000000000000000000FFFF00000000h
    CAS 3: masque ← 0000000000000000FFFF000000000000h
    CAS 4: masque ← 00000000000FFFF0000000000000000h
    CAS 5: masque ← 00000000FFFF00000000000000000000h
    CAS 6: masque ← 0000FFFF000000000000000000000000h
    CAS 7: masque ← FFFF0000000000000000000000000000h
  FIN EVALUE CAS
SINON
  compteur ← compteur ⋂ 03h
  EVALUER CAS compteur
    CAS 0: masque ← 000000000000FFFFh
    CAS 1: masque ← 00000000FFFF0000h
    CAS 2: masque ← 0000FFFF00000000h
```

```
    CAS 3: masque ← FFFF000000000000h
  FIN EVALUE CAS
FIN SI dest ← (dest ∩ ¬ (masque)) U ((source << (compteur x 16)) ∩ masque)
```

Mnémonique

Instruction	Opcode	Description
PINSRW *mm, r32/m16, imm8*	0Fh C4h /r *ib*	Cette instruction permet d'effectuer une copie d'un mot d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.
PINSRW *xmm, r32/m16, imm8*	66h 0Fh C4h /r *ib*	Cette instruction permet d'effectuer une copie d'un mot d'un opérande source et de l'insérer dans l'opérande de destination de l'emplacement spécifié par un compteur d'opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 142 à 144.*

Assembleur 80x86 — PMACHRIW

Cyrix 6x86MX (EMMX)+ — *Packed Multiply and Accumulate with Rounding*

Syntaxe

PMACHRIW *destination, source*

Description

Cette instruction permet d'effectuer une multiplication de 2 opérandes sources en utilisant la méthode décrite par *PMULHRW* et accumule le résultat avec la valeur dans un registre de destination en utilisant un arrondissement arithmétique.

Algorithme

destination(15..0) ← *destination*(15..0) + (*destination*(15..0) x *source*(15..0) + 00004000h)(30..15)
destination(31..16) ← *destination*(31..16) + (*destination*(31..16) x *source*(31..16) + 00004000h)(30..15)
destination(47..32) ← *destination*(47..32) + (*destination*(47..32) x *source*(47..32) + 00004000h)(30..15)
destination(63..48) ← *destination*(63..48) + (*destination*(63..48) x *source*(63..48) + 00004000h)(30..15)

Mnémonique

Instruction	Opcode	Description
PMACHRIW *mm,m64*	0Fh 5Eh /r	Cette instruction permet d'effectuer une multiplication de 2 opérandes sources en utilisant la méthode décrite par *PMULHRW* et accumule le résultat avec la valeur dans un registre de destination en utilisant un arrondissement arithmétique.

Assembleur 80x86 PMADDUBSW

SSSE3+

Multiply and Add Packed Signed and Unsigned Bytes

Syntaxe

PMADDUBSW *dest*,*source*

Description

Cette instruction permet d'effectuer une multiplication vertical de chacun des octets d'un opérande de destination avec les octets (entier) d'un opérande source et produit des entiers 16 bits intermédiaire.

Algorithme

SI opérande *dest* est 128 bits **ALORS**
  dest(15..0) ← SaturateToSignedWord(*source*(15..8) x *dest*(15..8) + *source*(7..0) x *dest*(7..0))
  dest(127..112) ← SaturateToSignedWord(*source*(127..120) x *dest*(127..120) + *source*(119..112) x *dest*(119..112))
SINON
  dest(15..0) ← SaturateToSignedWord(*source*(15..8) x *dest*(15..8) + *source*(7..0) x *dest*(7..0))
  dest(31..16) ← SaturateToSignedWord(*source*(31..24) x *dest*(31..24) + *source*(23..16) x *dest*(23..16))
  dest(47..32) ← SaturateToSignedWord(*source*(47..40) x *dest*(47..40) + *source*(39..32) x *dest*(39..32))
  dest(63..48) ← SaturateToSignedWord(*source*(63..56) x *dest*(63..56) + *source*(55..48) x *dest*(55..48))
FIN SI *dest* ← (*dest* ⋂ ¬ (*masque*)) U ((*source* << (*compteur* x 16)) ⋂ *masque*)

Mnémonique

Instruction	Opcode	Description

PMADDUBSW *mm1, mm2/m64*	0Fh 38h 04h /r	Cette instruction permet d'effectuer une multiplication vertical de chacun des octets d'un opérande de destination avec les octets (entier) d'un opérande source et produit des entiers 16 bits intermédiaire.
PMADDUBSW *xmm1,xmm2/m128*	66h 0Fh 38h 04h /r	Cette instruction permet d'effectuer une multiplication vertical de chacun des octets d'un opérande de destination avec les octets (entier) d'un opérande source et produit des entiers 16 bits intermédiaire.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 145 à 147.

Assembleur 80x86 — PMADDWD

INTEL Pentium Pro+ — *Packed Multiply and Add Dwords*

Syntaxe

PMADDWD *destination*, *source*

Description

Cette instruction permet d'effectuer une multiplication de 2 registres MMX contenant dans un paquet de mots d'un emplacement/mémoire.

Algorithme

destination(31..0) ← *destination*(15..0) x *source*(15..0) + *destination*(31..16) x *source*(31..16)
destination(63..32) ← *destination*(47..32) x *source*(47..32) + *destination*(63..48) x *source*(63..48)

Mnémonique

Instruction	Opcode	Description
PMADDWD *mm,mm/m64*	0Fh F5h /r	Cette instruction permet d'effectuer une multiplication de 2 registres MMX contenant dans un paquet de mots d'un emplacement/mémoire.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 148 à 151.

Assembleur 80x86 — PMAGW

Cyrix 6x86MX (EMMX)+ — *Packed Magnitude*

Syntaxe

PMAGW *destination, source*

Description

Cette instruction permet d'effectuer la comparaison de la valeur absolue d'un paquet de mots de l'opérande source et de l'ensemble des mots de destination et fixe l'opérande de destination avec la valeur de la plus large magnétude.

Algorithme

```
SI ABS(source(15..0)) > ABS(source(15..0)) ALORS
  destination(15..0) ← source(15..0)
FIN SI
SI ABS(source(31..16)) > ABS(source(31..16)) ALORS
  destination(31..16) ← source(31..16)
FIN SI
SI ABS(source(47..32)) > ABS(source(47..32)) ALORS
  destination(47..32) ← source(47..32)
FIN SI
SI ABS(source(63..56)) > ABS(source(63..56)) ALORS
  destination(63..56) ← source(63..56)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMAGW *mm,mm/m64*	0Fh 52h /r	Cette instruction permet d'effectuer la comparaison de la valeur absolue d'un paquet de mots de l'opérande source et de l'ensemble des mots de destination et fixe l'opérande de destination avec la valeur de la plus large

		magnétude.

Assembleur 80x86 PMAXSB

SSE4.1+ *Maximum of Packed Signed Byte Integers*

Syntaxe

PMAXSB *destination*, *source*

Description

Cette instruction permet de retourner la valeur maximal de chacun des octets (entier) des paquets contenu dans deux opérandes.

Algorithme

SI *destination*(7..0) > *source*(7..0) **ALORS**
  destination(7..0) ← *destination*(7..0)
SINON
  destination(7..0) ← *source*(7..0)
FIN SI
SI *destination*(15..8) > *source*(15..8) **ALORS**
  destination(15..8) ← *destination*(15..8)
SINON
  destination(15..8) ← *source*(15..8)
FIN SI
SI *destination*(23..16) > *source*(23..16) **ALORS**
  destination(23..16) ← *destination*(23..16)
SINON
  destination(23..16) ← SRC(23..16)
FIN SI
SI *destination*(31..24) > *source*(31..24) **ALORS**
  destination(31..24) ← *destination*(31..24)
SINON
  destination(31..24) ← SRC(31..24)
FIN SI
SI *destination*(39..32) > *source*(39..32) **ALORS**

```
  destination(39..32) ← destination(39..32)
SINON
  destination(39..32) ← source(39..32)
FIN SI
SI destination(47..40) > source(47..40) ALORS
  destination(47..40) ← destination(47..40)
SINON
  destination(47..40) ← source(47..40)
FIN SI
SI destination(55..48) > source(55..48) ALORS
  destination(55..48) ← destination(55..48)
SINON
  destination(55..48) ← source(55..48)
FIN SI
SI destination(63..56) > source(63..56) ALORS
  destination(63..56) ← destination(63..56)
SINON
  destination(63..56) ← source(63..56)
FIN SI
SI destination(71..64) > source(71..64) ALORS
  destination(71..64) ← destination(71..64)
SINON
  destination(71..64) ← source(71..64)
FIN SI
SI destination(79..72) > source(79..72) ALORS
  destination(79..72) ← destination(79..72)
SINON
  destination(79..72) ← source(79..72)
FIN SI
SI destination(87..80) > source(87..80) ALORS
  destination(87..80) ← destination(87..80)
SINON
  destination(87..80) ← source(87..80)
FIN SI
SI destination(95..88) > source(95..88) ALORS
  destination(95..88) ← destination(95..88)
SINON
  destination(95..88) ← source(95..88)
FIN SI
SI destination(103..96) > source(103..96) ALORS
  destination(103..96) ← destination(103..96)
```

```
SINON
  destination(103..96) ← source(103..96)
FIN SI
SI destination(111..104) > source(111..104) ALORS
  destination(111..104) ← destination(111..104)
SINON
  destination(111..104) ← source(111..104)
FIN SI
SI destination(119..112) > source(119..112) ALORS
  destination(119..112) ← destination(119..112)
SINON
  destination(119..112) ← source(119..112)
FIN SI
SI destination(127..120) > source(127..120) ALORS
  destination(127..120) ← destination(127..120)
SINON
  destination(127..120) ← source(127..120)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMAXSB *xmm1,xmm2/m128*	66h 0Fh 38h 3Ch /r	Cette instruction permet de retourner la valeur maximal de chacun des octets (entier) des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 152 à 154.

Assembleur 80x86 PMAXSD

SSE4.1+

Maximum of Packed Signed Dword Integers

Syntaxe

PMAXSD *destination*, *source*

Description

Cette instruction permet de retourner la valeur maximal de chacun des doubles mots des paquets contenu dans deux opérandes.

Algorithme

SI *destination*(31..0) > *source*(31..0) **ALORS**
  destination(31..0) ← *destination*(31..0)
SINON
  destination(31..0) ← *source*(31..0)
FIN SI
SI *destination*(63..32) > *source*(63..32) **ALORS**
  destination(63..32) ← *destination*(63..32)
SINON
  destination(63..32) ← *source*(63..32)
FIN SI
SI *destination*(95..64) > SRC(95..64) **ALORS**
  destination(95..64) ← *destination*(95..64)
SINON
  destination(95..64) ← *source*(95..64)
FIN SI
SI *destination*(127..96) > SRC(127..96) **ALORS**
  destination(127..96) ← *destination*(127..96)
SINON
  destination(127..96) ← *source*(127..96)
FIN SI

Mnémonique

Instruction	Opcode	Description
PMAXSD *xmm1,xmm2/m128*	66h 0Fh 38h 3Dh /r	Cette instruction permet de retourner la valeur maximal de chacun des doubles mots des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 155 à 157.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

PMAXSW

Packed Signed Integer Word Maximum

Syntaxe

PMAXSW *destination*, *source*

Description

Cette instruction permet de retourner la valeur maximal de chacun des mots (entier) des paquets contenu dans deux opérandes.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  SI destination(0..15) <= source(0..15) ALORS
    destination(0..15) ← source(0..15)
  FIN SI
  SI destination(16..31) <= source[16..31]) ALORS
    destination(16..31) ← source(16..31)
  FIN SI
  SI destination(32..47) <= source(32..47) ALORS
    destination(32..47) ← source(32..47)
  FIN SI
  SI destination(48..63) <= source(48..63) ALORS
    destination(48..63) ← source(48..63)
  FIN SI
SINON
  SI destination(0..15) <= source(0..15) ALORS
    destination(0..15) ← source(0..15)
  FIN SI
  SI destination(16..31) <= source(16..31) ALORS
    destination(16..31) ← source(16..31)
```

```
  FIN SI
  SI destination(32..47) <= source(32..47) ALORS
    destination(32..47) ← source(32..47)
  FIN SI
  SI destination(48..63) <= source(48..63) ALORS
    destination(48..63) ← source(48..63)
  FIN SI
  SI destination(64..79) <= source(64..79) ALORS
    destination(64..79) ← source(64..79)
  FIN SI
  SI destination(80..95) <= source(80..95) ALORS
    destination(80..95) ← source(80..95)
  FIN SI
  SI destination(96..111) <= source(96..111) ALORS
    destination(96..111) ← source(96..111)
  FIN SI
  SI destination(112..127) <= source(112..127) ALORS
    destination(112..127) ← source(112..127)
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMAXSW *mm1,mm2/m64*	0Fh EEh /r	Cette instruction permet de retourner la valeur maximal de chacun des mots (entier) des paquets contenu dans deux opérandes.
PMAXSW *xmm1,xmm2/m128*	66h 0Fh EEh /r	Cette instruction permet de retourner la valeur maximal de chacun des mots (entier) des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 158 à 160.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

PMAXUB

Packed Unsigned Integer Byte Maximum

Syntaxe

PMAXUB *destination*, *source*

Description

Cette instruction permet de retourner la valeur maximal de chacun des octets des paquets contenu dans deux opérandes.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  SI destination(7..0) <= source(7..0) ALORS
    destination(7..0) ← source(7..0)
  FIN SI
  SI destination(8..15) <= source(8..15) ALORS
    destination(8..15) ← source(8..15)
  FIN SI
  SI destination(16..23) <= source(16..23) ALORS
    destination(16..23) ← source(16..23)
  FIN SI
  SI destination(24..31) <= source(24..31) ALORS
    Destination(24..31) ← Source(24..31)
  FIN SI
  SI destination(32..39) <= source(32..39) ALORS
    Destination(32..39) ← Source(32..39)
  FIN SI

  SI destination(40..47) <= source(40..47) ALORS
    Destination(40..47) ← Source(40..47)
```

```
  FIN SI
  SI destination(48..55) <= source(48..55) ALORS
    Destination(48..55) ← Source(48..55)
  FIN SI
  SI destination(63..56) <= source(63..56) ALORS
    destination(63..56) ← source(63..56)
  FIN SI
SINON
  SI destination(7..0) <= source(7..0) ALORS
    destination(7..0) ← source(7..0)
  FIN SI
  SI destination(8..15) <= source(8..15) ALORS
    destination(8..15) ← source(8..15)
  FIN SI
  SI destination(16..23) <= source(16..23) ALORS
    destination(16..23) ← source(16..23)
  FIN SI
  SI destination(24..31) <= source(24..31) ALORS
    destination(24..31) ← source(24..31)
  FIN SI
  SI destination(32..39) <= source(32..39) ALORS
    destination(32..39) ← source(32..39)
  FIN SI
  SI destination(40..47) <= source(40..47) ALORS
    destination(40..47) ← source(40..47)
  FIN SI
  SI destination(48..55) <= source(48..55) ALORS
    destination(48..55) ← source(48..55)
  FIN SI
  SI destination(56..63) <= source(56..63) ALORS
    destination(56..63) ← source(56..63)
  FIN SI
  SI destination(64..71) <= source(64..71) ALORS
    destination(64..71) ← source(64..71)
  FIN SI
  SI destination(72..79) <= source(72..79) ALORS
    destination(72..79) ← source(72..79)
  FIN SI
  SI destination(80..87) <= source(80..87) ALORS
    destination(80..87) ← source(80..87)
  FIN SI
```

```
  SI destination(88..95) <= source(88..95) ALORS
    destination(88..95) ← source(88..95)
  FIN SI
  SI destination(96..103) <= source(96..103) ALORS
    destination(96..103) ← source(96..103)
  FIN SI
  SI destination(104..111) <= source(104..111) ALORS
    destination(104..111) ← source(104..111)
  FIN SI
  SI destination(112..119) <= source(112..119) ALORS
    destination(112..119) ← source(112..119)
  FIN SI
  SI destination(127..120) <= source(127..120) ALORS
    (destination(127..120) ← source(127..120)
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMAXUB *mm1,mm2/m64*	0Fh DEh /r	Cette instruction permet de retourner la valeur maximal de chacun des octets des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 161 à 163.

Assembleur 80x86 PMAXUD

SSE4.1+

Maximum of Packed Unsigned Dword Integers

Syntaxe

PMAXUD *destination*, *source*

Description

Cette instruction permet de retourner la valeur maximal de chacun des doubles mots (naturel) des paquets contenu dans deux opérandes.

Algorithme

SI *destination*(31..0) > *source*(31..0) **ALORS**
  destination(31..0) ← *destination*(31..0)
SINON
  destination(31..0) ← *source*(31..0)
FIN SI
SI *destination*(63..32) > *source*(63..32) **ALORS**
  destination(63..32) ← *destination*(63..32)
SINON
  destination(63..32) ← *source*(63..32)
FIN SI
SI *destination*(95..64) > *source*(95..64) **ALORS**
  destination(95..64) ← *destination*(95..64)
SINON
  destination(95..64) ← *source*(95..64)
FIN SI
SI *destination*(127..96) > *source*(127..96) **ALORS**
  destination(127..96) ← *destination*(127..96)
SINON
  destination(127..96) ← *source*(127..96)
FIN SI

Mnémonique

Instruction	Opcode	Description
PMAXUD *xmm1,xmm2/m128*	66h 0Fh 38h 3Fh /r	Cette instruction permet de retourner la valeur maximal de chacun des doubles mots (naturel) des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 164 à 166.

Assembleur 80x86 PMAXUW

SSE4.1+

Maximum of Packed Unsigned Word

Syntaxe

PMAXUW *destination*, *source*

Description

Cette instruction permet de retourner la valeur maximal de chacun des mots (naturel) des paquets contenu dans deux opérandes.

Algorithme

```
SI destination(15..0) > source(15..0) ALORS
  destination(15..0) ← destination(15..0)
SINON
  destination(15..0) ← source(15..0)
FIN SI
SI destination(31..16) > source(31..16) ALORS
  destination(31..16) ← destination(31..16)
SINON
  destination(31..16) ← source(31..16)
FIN SI
SI destination(47..32) > source(47..32) ALORS
  destination(47..32) ← destination(47..32)
SINON
  destination(47..32) ← source(47..32)
FIN SI
SI destination(63..48) > source(63..48) ALORS
  destination(63..48) ← destination(63..48)
SINON
  destination(63..48) ← source(63..48)
FIN SI
SI destination(79..64) > source(79..64) ALORS
```

```
  destination(79..64) ← destination(79..64)
SINON
  destination(79..64) ← source(79..64)
FIN SI
SI destination(95..80) > source(95..80) ALORS
  destination(95..80) ← destination(95..80)
SINON
  destination(95..80) ← source(95..80)
FIN SI
SI destination(111..96) > source(111..96) ALORS
  destination(111..96) ← destination(111..96)
SINON
  destination(111..96) ← source(111..96)
FIN SI
SI destination(127..112) > source(127..112) ALORS
  destination(127..112) ← destination(127..112)
SINON
  destination(127..112) ← source(127..112)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMAXUW *xmm1,xmm2/m128*	66h 0Fh 38h 3Eh /r	Cette instruction permet de retourner la valeur maximal de chacun des mots (naturel) des paquets contenus dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 167 à 169.

Assembleur 80x86 PMINSB

SSE4.1+ *Minimum of Packed Signed Byte Integers*

Syntaxe

PMINSB *destination*, *source*

Description

Cette instruction permet de retourner la valeur minimal de chacun des octets (entier) des paquets contenu dans deux opérandes.

Algorithme

SI *destination*(7..0) < *source*(7..0) **ALORS**
  destination(7..0) ← *destination*(7..0)
SINON
  destination(7..0) ← *source*(7..0)
FIN SI
SI *destination*(15..8) < *source*(15..8) **ALORS**
  destination(15..8) ← *destination*(15..8)
SINON
  destination(15..8) ← *source*(15..8)
FIN SI
SI *destination*(23..16) < *source*(23..16) **ALORS**
  destination(23..16) ← *destination*(23..16)
SINON
  destination(23..16) ← *source*(23..16)
FIN SI
SI *destination*(31..24) < *source*(31..24) **ALORS**
  destination(31..24) ← *destination*(31..24)
SINON
  destination(31..24) ← *source*(31..24)
FIN SI
SI *destination*(39..32) < *source*(39..32) **ALORS**

```
  destination(39..32) ← destination(39..32)
SINON
  destination(39..32) ← source(39..32)
FIN SI
SI destination(47..40) < source(47..40) ALORS
  destination(47..40) ← destination(47..40)
SINON
  destination(47..40) ← source(47..40)
FIN SI
SI destination(55..48) < source(55..48) ALORS
  destination(55..48) ← destination(55..48)
SINON
  destination(55..48) ← source(55..48)
FIN SI
SI destination(63..56) < source(63..56) ALORS
  destination(63..56) ← destination(63..56)
SINON
  destination(63..56) ← source(63..56)
FIN SI
SI destination(71..64) < source(71..64) ALORS
  destination(71..64) ← destination(71..64)
SINON
  destination(71..64) ← source(71..64)
FIN SI
SI destination(79..72) < source(79..72) ALORS
  destination(79..72) ← destination(79..72)
SINON
  destination(79..72) ← source(79..72)
FIN SI
SI destination(87..80) < source(87..80) ALORS
  destination(87..80) ← destination(87..80)
SINON
  destination(87..80) ← source(87..80)
FIN SI
SI destination(95..88) < source(95..88) ALORS
  destination(95..88) ← destination(95..88)
SINON
  destination(95..88) ← source(95..88)
FIN SI
SI destination(103..96) < source(103..96) ALORS
  destination(103..96) ← destination(103..96)
```

```
SINON
  destination(103..96) ← source(103..96)
FIN SI
SI destination(111..104) < source(111..104) ALORS
  destination(111..104) ← destination(111..104)
SINON
  destination(111..104) ← source(111..104)
FIN SI
SI destination(119..112) < source(119..112) ALORS
  destination(119..112) ← destination(119..112)
SINON
  destination(119..112) ← source(119..112)
FIN SI
SI destination(127..120) < source(127..120) ALORS
  destination(127..120) ← destination(127..120)
SINON
  destination(127..120) ← source(127..120)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMINSB *xmm1,xmm2/m128*	66h 0Fh 38h 38h /r	Cette instruction permet de retourner la valeur minimal de chacun des octets (entier) des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 170 à 172.

Assembleur 80x86 PMINSD

SSE4.1+

Minimum of Packed Signed Double word

Syntaxe

PMINSD *destination*, *source*

Description

Cette instruction permet de retourner la valeur minimal de chacun des doubles mots des paquets contenu dans deux opérandes.

Algorithme

```
SI destination(31..0) < source(31..0) ALORS
  destination(31..0) ← destination(31..0)
SINON
  destination(31..0) ← source(31..0)
FIN SI
SI destination(63..32) < source(63..32) ALORS
  destination(63..32) ← destination(63..32)
SINON
  destination(63..32) ← source(63..32)
FIN SI
SI destination(95..64) < source(95..64) ALORS
  destination(95..64) ← destination(95..64)
SINON
  destination(95..64) ← source(95..64)
FIN SI
SI destination(127..96) < source(127..96) ALORS
  destination(127..96) ← destination(127..96)
SINON
  destination(127..96) ← source(127..96)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMINSD *xmm1,xmm2/m128*	66h 0Fh 38h 39h /r	Cette instruction permet de retourner la valeur minimal de chacun des doubles mots des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 173 à 175.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

PMINSW

Packed Signed Integer Word Minimum

Syntaxe

PMINSW *destination*, *source*

Description

Cette instruction permet de retourner la valeur minimal de chacun des mots des paquets contenu dans deux opérandes.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  SI destination(0..15) >= source(0..15) ALORS
    destination(0..15) ← source(0..15)
  FIN SI
  SI destination(16..31) >= source[16..31]) ALORS
    destination(16..31) ← source(16..31)
  FIN SI
  SI destination(32..47) >= source(32..47) ALORS
    destination(32..47) ← source(32..47)
  FIN SI
  SI destination(48..63) >= source(48..63) ALORS
    destination(48..63) ← source(48..63)
  FIN SI
SINON
  SI destination(0..15) >= source(0..15) ALORS
    destination(0..15) ← source(0..15)
  FIN SI
  SI destination(16..31) >= source(16..31) ALORS
    destination(16..31) ← source(16..31)
```

```
FIN SI
SI destination(32..47) >= source(32..47) ALORS
  destination(32..47) ← source(32..47)
FIN SI
SI destination(48..63) >= source(48..63) ALORS
  destination(48..63) ← source(48..63)
FIN SI
SI destination(64..79) >= source(64..79) ALORS
  destination(64..79) ← source(64..79)
FIN SI
SI destination(80..95) >= source(80..95) ALORS
  destination(80..95) ← source(80..95)
FIN SI
SI destination(96..111) >= source(96..111) ALORS
  destination(96..111) ← source(96..111)
FIN SI
SI destination(112..127) >= source(112..127) ALORS
  destination(112..127) ← source(112..127)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMINSW *mm1,mm2/m64*	0Fh EAh /r	Cette instruction permet de retourner la valeur minimal de chacun des mots des paquets contenu dans deux opérandes.
PMINSW *xmm1,xmm2/m128*	66h 0Fh EAh /r	Cette instruction permet de retourner la valeur minimal de chacun des mots des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 176 à 178.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

PMINUB

Packed Unsigned Integer Byte Minimum

Syntaxe

PMINUB *destination*, *source*

Description

Cette instruction permet de retourner la valeur minimal de chacun des octets des paquets contenu dans deux opérandes.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  SI destination(7..0) >= source(7..0) ALORS
    destination(7..0) ← source(7..0)
  FIN SI
  SI destination(8..15) >= source(8..15) ALORS
    destination(8..15) ← source(8..15)
  FIN SI
  SI destination(16..23) >= source(16..23) ALORS
    destination(16..23) ← source(16..23)
  FIN SI
  SI destination(24..31) >= source(24..31) ALORS
    Destination(24..31) ← Source(24..31)
  FIN SI
  SI destination(32..39) >= source(32..39) ALORS
    Destination(32..39) ← Source(32..39)
  FIN SI

  SI destination(40..47) >= source(40..47) ALORS
    Destination(40..47) ← Source(40..47)
```

```
  FIN SI
  SI destination(48..55) >= source(48..55) ALORS
    Destination(48..55) ← Source(48..55)
  FIN SI
  SI destination(63..56) >= source(63..56) ALORS
    destination(63..56) ← source(63..56)
  FIN SI
SINON
  SI destination(7..0) >= source(7..0) ALORS
    destination(7..0) ← source(7..0)
  FIN SI
  SI destination(8..15) >= source(8..15) ALORS
    destination(8..15) ← Source(8..15)
  FIN SI
  SI destination(16..23) >= source(16..23) ALORS
    destination(16..23) ← source(16..23)
  FIN SI
  SI destination(24..31) >= source(24..31) ALORS
    destination(24..31) ← source(24..31)
  FIN SI
  SI destination(32..39) >= source(32..39) ALORS
    destination(32..39) ← source(32..39)
  FIN SI
  SI destination(40..47) >= source(40..47) ALORS
    destination(40..47) ← source(40..47)
  FIN SI
  SI destination(48..55) >= source(48..55) ALORS
    destination(48..55) ← source(48..55)
  FIN SI
  SI destination(56..63) >= source(56..63) ALORS
    destination(56..63) ← source(56..63)
  FIN SI
  SI destination(64..71) >= source(64..71) ALORS
    destination(64..71) ← source(64..71)
  FIN SI
  SI destination(72..79) >= source(72..79) ALORS
    destination(72..79) ← source(72..79)
  FIN SI
  SI destination(80..87) >= source(80..87) ALORS
    destination(80..87) ← source(80..87)
  FIN SI
```

```
  SI destination(88..95) >= source(88..95) ALORS
    destination(88..95) ← source(88..95)
  FIN SI
  SI destination(96..103) >= source(96..103) ALORS
    destination(96..103) ← source(96..103)
  FIN SI
  SI destination(104..111) >= source(104..111) ALORS
    destination(104..111) ← source(104..111)
  FIN SI
  SI destination(112..119) >= source(112..119) ALORS
    destination(112..119) ← source(112..119)
  FIN SI
  SI destination(127..120) >= source(127..120) ALORS
    (destination(127..120) ← source(127..120)
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMINUB *mm1,mm2/m64*	0Fh DAh /r	Cette instruction permet de retourner la valeur minimal de chacun des octets des paquets contenu dans deux opérandes.
PMINUB *xmm1,xmm2/m128*	66h 0Fh DAh /r	Cette instruction permet de retourner la valeur minimal de chacun des octets des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 179 à 181.

Assembleur 80x86 PMINUD

SSE4.1+

Minimum of Packed Unsigned Doubleword

Syntaxe

PMINUD *destination*, *source*

Description

Cette instruction permet de retourner la valeur minimal de chacun des doubles mots (naturel) des paquets contenu dans deux opérandes.

Algorithme

SI *destination*(31..0) < *source*(31..0) **ALORS**
  destination(31..0) ← *destination*(31..0)
SINON
  destination(31..0) ← *source*(31..0)
FIN SI
SI *destination*(63..32) < *source*(63..32) **ALORS**
  destination(63..32) ← *destination*(63..32)
SINON
  destination(63..32) ← *source*(63..32)
FIN SI
SI *destination*(95..64) < *source*(95..64) **ALORS**
  destination(95..64) ← *destination*(95..64)
SINON
  destination(95..64) ← *source*(95..64)
FIN SI
SI *destination*(127..96) < *source*(127..96) **ALORS**
  destination(127..96) ← *destination*(127..96)
SINON
  destination(127..96) ← *source*(127..96)
FIN SI

Mnémonique

Instruction	Opcode	Description
PMINUD *xmm1,xmm2/m128*	66h 0Fh 38h 3Bh /r	Cette instruction permet de retourner la valeur minimal de chacun des doubles mots (naturel) des paquets contenu dans deux opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 182 à 184.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

PMOVMSKB

Move Byte Mask

Syntaxe

PMOVMSKB *destination*, *source*

Description

Cette instruction permet de copiés les masques de chacun des octets d'un paquet contenu dans une opérande.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  destination(0) ← *source*(7)
  destination(1) ← *source*(15)
  destination(2) ← *source*(23)
  destination(3) ← *source*(31)
  destination(4) ← *source*(39)
  destination(5) ← *source*(47)
  destination(6) ← *source*(55)
  destination(7) ← *source*(63)
  destination(8..31) ← 0
SINON
  destination(0) ← *source*(7)
  destination(1) ← *source*(15)
  destination(2) ← *source*(23)
  destination(3) ← *source*(31)
  destination(4) ← *source*(39)
  destination(5) ← *source*(47)
  destination(6) ← *source*(55)
  destination(7) ← *source*(63)

destination(8) ← *source*(71)
destination(9) ← *source*(79)
destination(10) ← *source*(87)
destination(11) ← *source*(95)
destination(12) ← *source*(103)
destination(13) ← *source*(111)
destination(14) ← *source*(119)
destination(15) ← *source*(127)
destination(16..31) ← 0
FIN SI

Mnémonique

Instruction	Opcode	Description
PMOVMSKB *r32,mm*	0Fh D7h /r	Cette instruction permet de copiés les masques de chacun des octets d'un paquet contenu dans une opérande.
PMOVMSKB *r64,mm*	(REX.W) 0Fh D7h /r	Cette instruction permet de copiés les masques de chacun des octets d'un paquet contenu dans une opérande.
PMOVMSKB *reg,xmm*	66h 0Fh D7h /r	Cette instruction permet de copiés les masques de chacun des octets d'un paquet contenu dans une opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 188 à 190.

Assembleur 80x86 PMOVSXBD

SSE4.1+ *Packed Move with Sign Extend*

Syntaxe

PMOVSXBD *destination*, *source*

Description

Cette instruction permet d'étendre les signes du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 32 bits dans l'opérande de destination.

Algorithme

```
destination(31..0) ← SignExtend(source(7..0))
destination(63..32) ← SignExtend(source(15..8))
destination(95..64) ← SignExtend(source(23..16))
destination(127..96) ← SignExtend(source(31..24))
```

Mnémonique

Instruction	Opcode	Description
PMOVSXBD *xmm1,xmm2/m32*	66h 0Fh 38h 21h /r	Cette instruction permet d'étendre les signes du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 32 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 191 à 194.

Assembleur 80x86 PMOVSXBQ

SSE4.1+ *Packed Move with Sign Extend*

Syntaxe

PMOVSXBQ *destination*, *source*

Description

Cette instruction permet d'étendre les signes du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Algorithme

destination(63..0) ← SignExtend(*source*(7..0)) *destination*(127..64) ← SignExtend(*source*(15..8))

Mnémonique

Instruction	Opcode	Description
PMOVSXBQ *xmm1,xmm2/m16*	66h 0Fh 38h 22h /r	Cette instruction permet d'étendre les signes du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 191 à 194.

Assembleur 80x86 PMOVSXBW

SSE4.1+ *Packed Move with Sign Extend*

Syntaxe

PMOVSXBW *destination, source*

Description

Cette instruction permet d'étendre les signes du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 16 bits dans l'opérande de destination.

Algorithme

destination(15..0) ← SignExtend(*source*(7..0))
destination(31..16) ← SignExtend(*source*(15..8))
destination(47..32) ← SignExtend(*source*(23..16))
destination(63..48) ← SignExtend(*source*(31..24))
destination(79..64) ← SignExtend(*source*(39..32))
destination(95..80) ← SignExtend(*source*(47..40))
destination(111..96) ← SignExtend(*source*(55..48))
destination(127..112) ← SignExtend(*source*(63..56))

Mnémonique

Instruction	Opcode	Description
PMOVSXBW *xmm1,xmm2/m64*	66h 0Fh 38h 20h /r	Cette instruction permet d'étendre les signes du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 16 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 191 à 194.

Assembleur 80x86 PMOVSXDQ

SSE4.1+ *Packed Move with Sign Extend*

Syntaxe

PMOVSXDQ *destination, source*

Description

Cette instruction permet d'étendre les signes du paquet des entiers de 32 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Algorithme

```
DEST(63..0) ← SignExtend(SRC(31..0))
DEST(127..64) ← SignExtend(SRC(63..32))
```

Mnémonique

Instruction	Opcode	Description
PMOVSXDQ *xmm1,xmm2/m64*	66h 0Fh 38h 25h /r	Cette instruction permet d'étendre les signes du paquet des entiers de 32 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 148 à 151.

Assembleur 80x86 PMOVSXWD

SSE4.1+ *Packed Move with Sign Extend*

Syntaxe

PMOVSXWD *destination*, *source*

Description

Cette instruction permet d'étendre les signes du paquet des entiers de 16 bits de la partie basse de l'opérande source en entiers 32 bits dans l'opérande de destination.

Algorithme

destination(31..0) ← SignExtend(*source*(15..0))
destination(63..32) ← SignExtend(*source*(31..16))
destination(95..64) ← SignExtend(*source*(47..32))
destination(127..96) ← SignExtend(*source*(63..48))

Mnémonique

Instruction	Opcode	Description
PMOVSXWD *xmm1,xmm2/m64*	66 0f 38 23 /r	Cette instruction permet d'étendre les signes du paquet des entiers de 16 bits de la partie basse de l'opérande source en entiers 32 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 191 à 194.

Assembleur 80x86 PMOVSXWQ

SSE4.1+ *Packed Move with Sign Extend*

Syntaxe

PMOVSXWQ *destination*, *source*

Description

Cette instruction permet d'étendre les signes du paquet des entiers de 16 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Algorithme

destination(63..0) ← SignExtend(*source*(15..0)) *destination*(127..64) ← SignExtend(*source*(31..16))

Mnémonique

Instruction	Opcode	Description
PMOVSXWQ *xmm1,xmm2/m32*	66h 0Fh 38h 24h /r	Cette instruction permet d'étendre les signes du paquet des entiers de 16 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 191 à 194.

Assembleur 80x86 PMOVZXBD

SSE4.1+ *Packed Move with Zero Extend*

Syntaxe

PMOVZXBD *destination*, *source*

Description

Cette instruction permet d'étendre les zéros du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 32 bits dans l'opérande de destination.

Algorithme

destination(31..0) ← ZeroExtend(*source*(7..0))
destination(63..32) ← ZeroExtend(*source*(15..8))
destination(95..64) ← ZeroExtend(*source*(23..16))
destination(127..96) ← ZeroExtend(*source*(31..24))

Mnémonique

Instruction	Opcode	Description
PMOVZXBD *xmm1,xmm2/m32*	66h 0Fh 38h 31h /r	Cette instruction permet d'étendre les zéros du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 32 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 195 à 197.

Assembleur 80x86 PMOVZXBQ

SSE4.1+ *Packed Move with Zero Extend*

Syntaxe

PMOVZXBQ *destination*, *source*

Description

Cette instruction permet d'étendre les zéros du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Algorithme

destination(63..0) ← ZeroExtend(*source*(7..0)) *destination*(127..64) ← ZeroExtend(*source*(15..8))

Mnémonique

Instruction	Opcode	Description
PMOVZXBQ *xmm1,xmm2/m16*	66h 0Fh 38h 32h /r	Cette instruction permet d'étendre les zéros du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 195 à 197.

Assembleur 80x86 PMOVZXBW

SSE4.1+ *Packed Move with Zero Extend*

Syntaxe

PMOVZXBW *destination, source*

Description

Cette instruction permet d'étendre les zéros du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 16 bits dans l'opérande de destination.

Algorithme

destination(15..0) ← ZeroExtend(*source*(7..0))
destination(31..16) ← ZeroExtend(*source*(15..8))
destination(47..32) ← ZeroExtend(*source*(23..16))
destination(63..48) ← ZeroExtend(*source*(31..24))
destination(79..64) ← ZeroExtend(*source*(39..32))
destination(95..80) ← ZeroExtend(*source*(47..40))
destination(111..96) ← ZeroExtend(*source*(55..48))
destination(127..112) ← ZeroExtend(*source*(63..56))

Mnémonique

Instruction	Opcode	Description
PMOVZXBW *xmm1,xmm2/m64*	66h 0Fh 38h 30h /r	Cette instruction permet d'étendre les zéros du paquet des entiers de 8 bits de la partie basse de l'opérande source en entiers 16 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 195 à 197.

Assembleur 80x86 PMOVZXDQ

SSE4.1+ *Packed Move with Zero Extend*

Syntaxe

PMOVZXDQ *destination*, *source*

Description

Cette instruction permet d'étendre les zéros du paquet des entiers de 32 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Algorithme

destination(63..0) ← ZeroExtend(*source*(31..0)) *destination*(127..64) ← ZeroExtend(*source*(63..32))

Mnémonique

Instruction	Opcode	Description
PMOVZXDQ *xmm1,xmm2/m64*	66h 0Fh 38h 35h /r	Cette instruction permet d'étendre les zéros du paquet des entiers de 32 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 195 à 197.

Assembleur 80x86 PMOVZXWD

SSE4.1+ *Packed Move with Zero Extend*

Syntaxe

PMOVZXWD *destination*, *source*

Description

Cette instruction permet d'étendre les zéros du paquet des entiers de 16 bits de la partie basse de l'opérande source en entiers 32 bits dans l'opérande de destination.

Algorithme

destination(31..0) ← ZeroExtend(*source*(15..0))
destination(63..32) ← ZeroExtend(*source*(31..16))
destination(95..64) ← ZeroExtend(*source*(47..32))
destination(127..96) ← ZeroExtend(*source*(63..48))

Mnémonique

Instruction	Opcode	Description
PMOVZXWD *xmm1,xmm2/m64*	66h 0Fh 38h 33h /r	Cette instruction permet d'étendre les zéros du paquet des entiers de 16 bits de la partie basse de l'opérande source en entiers 32 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 195 à 197.

Assembleur 80x86 PMOVZXWQ

SSE4.1+ *Packed Move with Zero Extend*

Syntaxe

PMOVZXWQ *destination, source*

Description

Cette instruction permet d'étendre les zéros du paquet des entiers de 16 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Algorithme

destination(63..0) ← ZeroExtend(*source*(15..0)) *destination*(127..64) ← ZeroExtend(*source*(31..16))

Mnémonique

Instruction	Opcode	Description
PMOVZXWQ *xmm1,xmm2/m32*	66h 0Fh 38h 34h /r	Cette instruction permet d'étendre les zéros du paquet des entiers de 16 bits de la partie basse de l'opérande source en entiers 64 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 195 à 197.

Assembleur 80x86 PMULDQ

SSE4.1+ *Multiply Packed Signed Dword Integers*

Syntaxe

PMULDQ *destination*, *source*

Description

Cette instruction permet d'effectuer la multiplication de deux valeurs signés de deux paires de double mots entier et entrepose les résultats dans un opérande en paquet de 64 bits.

Algorithme

destination(63..0) ← *destination*(31..0) x *source*(31..0)
destination(127..64) ← *destination*(95..64) x *source*(95..64)

Mnémonique

Instruction	Opcode	Description
PMULDQ *xmm1,xmm2/m128*	66h 0Fh 38h 28h /r	Cette instruction permet d'effectuer la multiplication de deux valeurs signés de deux paires de double mots entier et entrepose les résultats dans un opérande en paquet de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 199 à 201.

Assembleur 80x86 — PMULHRIW

Cyrix 6x86MX (EMMX)+

Packed Multiply High with Rounding, result to Implied Register

Syntaxe

PMULHRIW *mmi, source*

Description

Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination vaut 0 et effectue se traitement pour chacun des 8 octets du paquet.

Algorithme

mmi(15..0) ← (*mmi*(15..0) x *source*(15..0) + 00004000h)(30..15)
mmi(31..16) ← (*mmi*(31..16) x *source*(31..16) + 00004000h)(30..15)
mmi(47..32) ← (*mmi*(47..32) x *source*(47..32) + 00004000h)(30..15)
mmi(63..48) ← (*mmi*(63..48) x *source*(63..48) + 00004000h)(30..15)

Mnémonique

Instruction	Opcode	Description
PMULHRIW *mm,mm/m64*	0Fh 5Dh /r	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination vaut 0 et effectue se traitement pour chacun des 8 octets du paquet.

Assembleur 80x86 PMULHRSW

SSSE3+

Packed Multiply High with Round and Scale

Syntaxe

PMULHRSW *destination*, *source*

Description

Cette instruction permet d'effectuer multiplication vertical de chaque entier de 16 bits de l'opérande de destination avec l'opérande source et produit un résultat d'entiers de 32 bits dans l'opérande de destination.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  temp0(31..0) ← INT32 ((*destination*(15..0) x *source*(15..0)) >> 14) + 1
  temp1(31..0) ← INT32 ((*destination*(31..16) x *source*(31..16)) >> 14) + 1
  temp2(31..0) ← INT32 ((*destination*(47..32) x *source*(47..32)) >> 14) + 1
  temp3(31..0) ← INT32 ((*destination*(63..48) x *source*(63..48)) >> 14) + 1
  destination(15..0) ← temp0(16..1)
  destination(31..16) ← temp1(16..1)
  destination(47..32) ← temp2(16..1)
  destination(63..48) ← temp3(16..1)
SINON SI taille de l'opérande = 128 bits **ALORS**
  temp0(31..0) ← INT32 ((*destination*(15..0) x *source*(15..0)) >> 14) + 1
  temp1(31..0) ← INT32 ((*destination*(31..16) x *source*(31..16)) >> 14) + 1
  temp2(31..0) ← INT32 ((*destination*(47..32) x *source*(47..32)) >> 14) + 1
  temp3(31..0) ← INT32 ((*destination*(63..48) x *source*(63..48)) >> 14) + 1
  temp4(31..0) ← INT32 ((*destination*(79..64) x *source*(79..64)) >> 14) + 1
  temp5(31..0) ← INT32 ((*destination*(95..80) x *source*(95..80)) >> 14) + 1
  temp6(31..0) ← INT32 ((*destination*(111..96) x *source*(111..96)) >> 14) + 1
  temp7(31..0) ← INT32 ((*destination*(127..112) x *source*(127..112)) >> 14) + 1
  destination(15..0) ← temp0(16..1)
  destination(31..16) ← temp1(16..1)

```
  destination(47..32) ← temp2(16..1)
  destination(63..48) ← temp3(16..1)
  destination(79..64) ← temp4(16..1)
  destination(95..80) ← temp5(16..1)
  destination(111..96) ← temp6(16..1)
  destination(127..112) ← temp7(16..1)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMULHRSW *mm1,mm2/m64*	0Fh 38h 0Bh /r	Cette instruction permet d'effectuer multiplication vertical de chaque entier de 16 bits de l'opérande de destination avec l'opérande source et produit un résultat d'entiers de 32 bits dans l'opérande de destination.
PMULHRSW *xmm1,xmm2/m128*	66h 0Fh 38h 0Bh /r	Cette instruction permet d'effectuer multiplication vertical de chaque entier de 16 bits de l'opérande de destination avec l'opérande source et produit un résultat d'entiers de 32 bits dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 201 à 203.

Assembleur 80x86 — PMULHRW

Cyrix 6x86MX (EMMX)+ — *Packed Multiply High with Rounding*

Syntaxe

PMULHRW *destination, source*

Description

Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination vaut 0 et effectue se traitement pour chacun des 8 octets du paquet.

Algorithme

destination(15..0) ← (*destination*(15..0) x *source*(15..0) + 00004000h)(30..15)
destination(31..16) ← (*destination*(31..16) x *source*(31..16) + 00004000h)(30..15)
destination(47..32) ← (*destination*(47..32) x *source*(47..32) + 00004000h)(30..15)
destination(63..48) ← (*destination*(63..48) x *source*(63..48) + 00004000h)(30..15)

Mnémonique

Instruction	Opcode	Description
PMULHRW *mm,mm/m64*	0Fh 59h /r	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination vaut 0 et effectue se traitement pour chacun des 8 octets du paquet.

Assembleur 80x86 — PMULHUW

Pentium III (KNI/MMX2)+ — *Packed Multiply high unsigned word*

Syntaxe

PMULHUW *destination, source*

Description

Cette instruction permet d'effectuer la multiplication de la partie haute de chacun des entiers des 2 paquets d'opérandes.

Algorithme

temp1 ← *destination*(15..0) x *source*(15..0)
temp2 ← *destination*(31..16) x *source*(31..16)
temp3 ← *destination*(47..32) x *source*(47..32)
temp4 ← *destination*(63..48) x *source*(63..48)
destination(15..0) ← temp1(31..16)
destination(31..16) ← temp2(31..16)
destination(47..32) ← temp3(31..16)
destination(63..48) ← temp4(31..16)

Mnémonique

Instruction	Opcode	Description
PMULHUW *mm,m64*	0Fh E4h /r	Cette instruction permet d'effectuer la multiplication de la partie haute de chacun des entiers des 2 paquets d'opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 204 à 207.

Assembleur 80x86 PMULHW

INTEL Pentium Pro+

Packed Multiply High by Words

Syntaxe

PMULHW *destination*, *source*

Description

Cette instruction permet d'effectuer la multiplication de la partie haute de chacun des mots des 2 paquets d'opérandes.

Algorithme

destination(15..0) ← (*destination*(15..0) x *source*(15..0)) (31..16)
destination(31..16) ← (*destination*(31..16) x *source*(31..16)) (31..16)
destination(47..32) ← (*destination*(47..32) x *source*(47..32)) (31..16)
destination(63..48) ← (*destination*(63..48) x *source*(63..48)) (31..16)

Mnémonique

Instruction	Opcode	Description
PMULHW *mm,m64*	0Fh E5h /r	Cette instruction permet d'effectuer la multiplication de la partie haute de chacun des mots des 2 paquets d'opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 208 à 211.

Assembleur 80x86 — PMULLD

SSE4.1+ — *Multiply Packed Signed Dword Integers and Store Low*

Syntaxe

PMULLD *destination*, *source*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de chacun des double mots des 2 paquets d'opérandes.

Algorithme

Temp0(63..0) ← *destination*(31..0) x *source*(31..0)
Temp1(63..0) ← *destination*(63..32) x *source*(63..32)
Temp2(63..0) ← *destination*(95..64) x *source*(95..64)
Temp3(63..0) ← *destination*(127..96) x *source*(127..96)
destination(31..0) ← Temp0(31..0)
destination(63..32) ← Temp1(31..0)
destination(95..64) ← Temp2(31..0)
destination(127..96) ← Temp3(31..0)

Mnémonique

Instruction	Opcode	Description
PMULLD *xmm1,xmm2/m128*	66h 0Fh 38h 40h /r	Cette instruction permet d'effectuer la multiplication de la partie basse de chacun des double mots des 2 paquets d'opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 212 à 214.

Assembleur 80x86 — PMULLW

INTEL Pentium Pro+ — *Packed Multiply Low by Words*

Syntaxe

PMULLW *destination*, *source*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de chacun des mots des 2 paquets d'opérandes.

Algorithme

destination(15..0) ← (*destination*(15..0) x *source*(15..0)) (15..0)
destination(31..16) ← (*destination*(31..16) x *source*(31..16)) (15..0)
destination(47..32) ← (*destination*(47..32) x *source*(47..32)) (15..0)
destination(63..48) ← (*destination*(63..48) x *source*(63..48)) (15..0)

Mnémonique

Instruction	Opcode	Description
PMULLW *mm,m64*	0Fh D5h /r	Cette instruction permet d'effectuer la multiplication de la partie basse de chacun des mots des 2 paquets d'opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 215 à 218.

Assembleur 80x86 — PMULUDQ

INTEL Pentium 4 (SSE2)+ — *Multiply Packed Unsigned Doubleword Integers*

Syntaxe

PMULUDQ *destination*, *source*

Description

Cette instruction permet d'effectuer la multiplication de paquets d'opérandes de double mots.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  destination(0..63) ← *destination*(0..31) x *source*(0..31)
SINON
  destination(0..63) ← *destination*(0..31) x *source*(0..31)
  destination(64..127) ←*destination*(64..95) x *source*(64..95)
FIN SI

Mnémonique

Instruction	Opcode	Description
PMULUDQ *mm1*, *mm2/m64*	0Fh F4h /r	Cette instruction permet d'effectuer la multiplication de paquets d'opérandes de double mots.
PMULUDQ *xmm1*, *xmm2/m128*	66h 0Fh F4h /r	Cette instruction permet d'effectuer la multiplication de paquets d'opérandes de double mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 219 à 221.

Assembleur 80x86 — PMVGEZB

Cyrix 6x86MX (EMMX)+ — *Packed Conditional Move Great Zero Byte*

Syntaxe

PMVGEZB *destination, source*

Description

Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination est supérieur à 0 et effectue se traitement pour chacun des 8 octets du paquet.

Algorithme

```
SI destination(7..0) >= 0 ALORS
  destination(7..0) ← source(7..0)
FIN SI
SI destination(15..8) >= 0 ALORS
  destination(15..8) ← source(15..8)
FIN SI
SI destination(23..16) >= 0 ALORS
  destination(23..16) ← source(23..16)
FIN SI
SI destination(31..24) >= 0 ALORS
  destination(31..24) ← source(31..24)
FIN SI
SI destination(39..32) >= 0 ALORS
  destination(39..32) ← source(39..32)
FIN SI
SI destination(47..40) >= 0 ALORS
  destination(47..40) ← source(47..40)
FIN SI
SI destination(55..48) >= 0 ALORS
  destination(55..48) ← source(55..48)
FIN SI
SI destination(63..56) >= 0 ALORS
  destination(63..56) ← source(63..56)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMVGEZB *mm,m64*	0Fh 5Ch /r	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination est supérieur à 0 et effectue se traitement pour chacun des 8 octets du paquet.

Assembleur 80x86 — PMVLZB

Cyrix 6x86MX (EMMX)+ — *Packed Conditional Move Less Zero Byte*

Syntaxe

PMVLZB *destination*, *source*

Description

Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination est inférieur à 0 et effectue se traitement pour chacun des 8 octets du paquet.

Algorithme

```
SI destination(7..0) < 0 ALORS
  destination(7..0) ← source(7..0)
FIN SI
SI destination(15..8) < 0 ALORS
  destination(15..8) ← source(15..8)
FIN SI
SI destination(23..16) < 0 ALORS
  destination(23..16) ← source(23..16)
FIN SI
SI destination(31..24) < 0 ALORS
  destination(31..24) ← source(31..24)
FIN SI
SI destination(39..32) < 0 ALORS
  destination(39..32) ← source(39..32)
FIN SI
SI destination(47..40) < 0 ALORS
  destination(47..40) ← source(47..40)
FIN SI
SI destination(55..48) < 0 ALORS
  destination(55..48) ← source(55..48)
FIN SI
SI destination(63..56) < 0 ALORS
  destination(63..56) ← source(63..56)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMVLZB *mm,m64*	0Fh 5Bh /r	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination est inférieur à 0 et effectue se traitement pour chacun des 8 octets du paquet.

Assembleur 80x86 — PMVNZB

Cyrix 6x86MX (EMMX)+ — *Packed Conditional Move Not Zero Byte*

Syntaxe

PMVNZB *destination*, *source*

Description

Cette instruction permet de copier l'octet de paquet correspondant de l'opérnade source dans l'opérande de destination si l'octet de l'opérande destination ne vaut pas 0 et effectue se traitement pour chacun des 8 octets du paquet.

Algorithme

```
SI destination(7..0) <> 0 ALORS
  destination(7..0) ← source(7..0)
FIN SI
SI destination(15..8) <> 0 ALORS
  destination(15..8) ← source(15..8)
FIN SI
SI destination(23..16) <> 0 ALORS
  destination(23..16) ← source(23..16)
FIN SI
SI destination(31..24) <> 0 ALORS
  destination(31..24) ← source(31..24)
FIN SI
SI destination(39..32) <> 0 ALORS
  destination(39..32) ← source(39..32)
FIN SI
SI destination(47..40) <> 0 ALORS
  destination(47..40) ← source(47..40)
FIN SI
SI destination(55..48) <> 0 ALORS
  destination(55..48) ← source(55..48)
FIN SI
SI destination(63..56) <> 0 ALORS
  destination(63..56) ← source(63..56)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PMVNZB *mm,mm/m64*	0Fh 5Ah /r	Cette instruction permet de copier l'octet de paquet correspondant de l'opérnade source dans l'opérande de destination si l'octet de l'opérande destination ne vaut pas 0 et effectue se traitement pour chacun des 8 octets du paquet.

Assembleur 80x86 — PMVZB

Cyrix 6x86MX (EMMX)+ — *Packed Conditional Move Zero Byte*

Syntaxe

PMVZB *mmi, source*

Description

Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination vaut 0 et effectue se traitement pour chacun des 8 octets du paquet.

Algorithme

SI *mmi*(7..0) = 0 **ALORS**
  mmi(7..0) ← *source*(7..0)
FIN SI
SI *mmi*(15..8) = 0 **ALORS**
  mmi(15..8) ← *source*(15..8)
FIN SI
SI *mmi*(23..16) = 0 **ALORS**
  mmi(23..16) ← *source*(23..16)
FIN SI
SI *mmi*(31..24) = 0 **ALORS**
  mmi(31..24) ← *source*(31..24)
FIN SI
SI *mmi*(39..32) = 0 **ALORS**
  mmi(39..32) ← *source*(39..32)
FIN SI
SI *mmi*(47..40) = 0 **ALORS**
  mmi(47..40) ← *source*(47..40)
FIN SI
SI *mmi*(55..48) = 0 **ALORS**
  mmi[55..48] ← *source*(55..48)
FIN SI
SI *mmi*(63..56) = 0 **ALORS**
  mmi(63..56) ← *source*(63..56)
FIN SI

Mnémonique

Instruction	Opcode	Description
PMVZB *mm,m64*	0Fh 58h /r	Cette instruction permet de copier l'octet de paquet correspondant de l'opérande source dans l'opérande de destination si l'octet de l'opérande destination vaut 0 et effectue se traitement pour chacun des 8 octets du paquet.

Assembleur 80x86	**POP**
INTEL 8088+	*Pop Value Of Stack*

Syntaxe

POP *opérande*

Description

Cette instruction permet de désempiler de la pile une mot ou un double mot et la met dans une opérande.

Algorithme

```
SI taille de l'adresse de pile = 32 bits ALORS
  SI taille de opérande = 32 bits ALORS
    opérande ← SS:ESP
    ESP ← ESP + 4
  SINON
    opérande ← SS:ESP
    ESP ← ESP + 2
  FIN SI
SINON
  SI taille de l'opérande = 16 bits ALORS
    opérande ← SS:SP
    SP ← SP + 2
  SINON
    opérande ← SS:SP
    SP ← SP + 4
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
POP *reg/mem16*	8Fh /0	Désempile du sommet de la pile une valeur et la met dans un emplacement mémoire ou registre 16 bits.
POP *reg/mem32*	8Fh /0	Désempile du sommet de la pile une valeur et la met dans un emplacement mémoire ou registre 32 bits. Il n'y a pas de préfix en mode 64 bits.
POP *reg/mem64*	8Fh /0	Désempile du sommet de la pile une valeur et la met dans un emplacement mémoire ou registre 64 bits.
POP *reg16*	58h +rw	Désempile du sommet de la pile une valeur et la met dans le registre 16 bits.
POP *reg32*	58h +rd	Désempile du sommet de la pile une valeur et la met dans un registre 32 bits. Il n'y a pas de préfix en mode 64 bits.
POP *reg64*	58h +rq	Désempile du sommet de la pile une valeur et la met dans un registre 64 bits.
POP DS	1Fh	Désempile du sommet de la pile une valeur et la met dans le registre DS. Invalide en mode 64 bits.
POP ES	07h	Désempile du sommet de la pile une valeur et la met dans le registre ES. Invalide en mode 64 bits.

POP SS	17h	Désempile du sommet de la pile une valeur et la met dans le registre SS. Invalide en mode 64 bits.
POP FS	0Fh A1h	Désempile du sommet de la pile une valeur et la met dans le registre FS.
POP GS	0Fh A9h	Désempile du sommet de la pile une valeur et la met dans le registre GS.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Les instructions «POP DS», «POP ES» ou «POP SS» sont exécutés en mode 64 bits.
#NP(Sélecteur)			X	Les registres DS, ES, FS ou GS sont chargés dans un sélecteur de segment non-nulle et le segment est marqué non présent.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de

				pile ou n'est pas canonique
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur non-nulle et le segment est marqué non présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#GP(Sélecteur)			X	Un registre de segment est chargé, mais le descripteur de segment dépasse la

				limite de la table du descripteur.
			X	Un registre de segment est chargé et le bit *TI* de sélecteur de segment est fixé, mais le sélecteur *LDT* est un sélecteur nulle.
			X	Le registre *SS* est chargé avec un sélecteur de segment nulle dans un mode non 64 bits ou avec CPL = 3.
			X	Le registre *SS* est chargé et le sélecteur de segment *RPL* et le descripteur de segment *DPL* n'est pas égale au *CPL*.
			X	Le registre *SS* est chargé et le segment pointe dans un segment

				de données non écrivable.
			X	Le registre DS, ES, FS ou GS est chargé et le segment pointe sur des données ou un segment de code non-conforme, mais le *RPL* ou *CPL* est supérieur au *DPL*.
			X	Le registre DS, ES, FS ou GS est chargé et le segment ne pointe pas sur un segment de données ou un segment de code en lecture.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est

				effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction PUSH

Références

Le livre d'Or PC, *Martin Althaus*, *1992*, *ISBN: 2-7361-0934-1*, *page 825*
Assembleur Facile, *Philippe Mercier*, *1990*, *ISBN: 2-501-01176-7*, *page 412*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 222 à 228.*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 185 à 186.*

Assembleur 80x86	POPA
INTEL 80286+	*Pop All General Registers*

Syntaxe

```
POPA
```

Description

Cette instruction permet de désempiler de la pile respectivement les registres DI, SI, BP, SP, BX, DX, CX et AX.

Algorithme

```
DI ← SS:SP
SP ← SP + 2
SI ← SS:SP
SP ← SP + 2
BP ← SS:SP
SP ← SP + 2
* Ne depile jamais le SP
SP ← SP + 2
BX ← SS:SP
SP ← SP + 2
DX ← SS:SP
SP ← SP + 2
CX ← SS:SP
SP ← SP + 2
AX ← SS:SP
SP ← SP + 2
```

Mnémonique

Instruction	Opcode	Description

POPA	61h	Désempile les registres DI, SI, BP, SP, BX, DX, CX et AX. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instructions est exécuté en mode 64 bits.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction PUSHA
Instruction assembleur 80x86 - Instruction PUSHAD

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 826*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 229 à 231.*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 187.*

Assembleur 80x86	POPAD
INTEL 80386+	*Pop All General Registers Double*

Syntaxe

```
POPAD
```

Description

Cette instruction permet de désempiler de la pile respectivement les registres EDI, ESI, EBP, ESP, EBX, EDX, ECX et EAX.

Algorithme

```
EDI ← SS:(E)SP
(E)SP ← (E)SP + 2
ESI ← SS:(E)SP
(E)SP ← (E)SP + 2
EBP ← SS:(E)SP
(E)SP ← (E)SP + 2
* Ne depile jamais le ESP
(E)SP ← (E)SP + 2
EBX ← SS:(E)SP
(E)SP ← (E)SP + 2
EDX ← SS:(E)SP
(E)SP ← (E)SP + 2
ECX ← SS:(E)SP
(E)SP ← (E)SP + 2
EAX ← SS:(E)SP
(E)SP ← (E)SP + 2
```

Mnémonique

Instruction	Opcode	Description

POPAD	61h	Désempile les registres EDI, ESI, EBP, ESP, EBX, EDX, ECX et EAX. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instructions est exécuté en mode 64 bits.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction PUSHA
Instruction assembleur 80x86 - Instruction PUSHAD

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 826*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 229 à 231.*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 187.*

Assembleur 80x86 — POPCNT

AMD K10 (SSE4a)+ — *Bit Population Count*

Syntaxe

POPCNT *regdest*, *source*

Description

Cette instruction permet de compter le nombre de bits à 1 que possède une opérande source et place le résultat dans un registre destinataire.

Algorithme

```
Count ← 0
BOUCLE POUR i ← 0 JUSQU'A taille de l'opérande SAUT 1
  SI source(i) = 1 ALORS
    Count ← Count + 1
  FIN SI
FIN BOUCLE
DEST ← Count
```

Mnémonique

Instruction	Opcode	Description
POPCNT *reg16*, *reg/mem16*	F3h 0Fh B8h /r	Compte les 1 dans les registres ou mémoire 16 bits.
POPCNT *reg32*, *reg/mem32*	F3h 0Fh B8h /r	Compte les 1 dans les registres ou mémoire 32 bits.

POPCNT *reg64*, *reg/mem64*	F3h 0Fh B8h /r	Compte les 1 dans les registres ou mémoire 32 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Cette instruction n'est pas supporté, comme indiqué par le bit 23 du registre ECX de la fonction 0000_00001h de l'instruction *CPUID*.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique

			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BSF
Instruction assembleur 80x86 - Instruction BSR
Instruction assembleur 80x86 - Instruction LZCNT

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 232 à 232.

Assembleur 80x86	POPF
INTEL 8088+	*Pop Flags*

Syntaxe

```
POPF
```

Description

Cette instruction permet de désempiler de la pile le registre 16 bits de drapeau contenant les indicateurs d'état.

Algorithme

```
Registre de drapeaux ← SS:(E)SP
(E)SP ← (E)SP + 2
```

Mnémonique

Instruction	Opcode	Description
POPF	9Dh	Désempile le mot de la pile et la met dans le registres des drapeaux (FLAGS).

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de pile ou n'est pas canonique
#GP(Protection général)		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et CR4.VME vaut 0.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et une opérande est de taille de 32 bits.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et les bits EFLAGS.VIP et le nouveau EFLAGS.IF valent 1.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et le nouveau EFLAGS.TF vaut 1.

#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction PUSHF
Instruction assembleur 80x86 - Instruction PUSHFD
Instruction assembleur 80x86 - Instruction PUSHFQ

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 826
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 412
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 234 à 237.
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 190.

Assembleur 80x86	POPFD
INTEL 80386+	*Pop Flags Double*

Syntaxe

```
POPFD
```

Description

Cette instruction permet de désempiler de la pile le registre 32 bits de drapeau contenant les indicateurs d'état.

Algorithme

```
Registre de drapeaux ← SS:(E)SP
(E)SP ← (E)SP + 4
```

Mnémonique

Instruction	Opcode	Description
POPFD	9Dh	Désempile le double mot de la pile et la met dans le registres 32 bits des drapeaux (EFLAGS). Il n'y a pas de préfix en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-	X	X	X	Une adresse mémoire

canonique)				dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et CR4.VME vaut 0.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et une opérande est de taille de 32 bits.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et les bits EFLAGS.VIP et le nouveau EFLAGS.IF valent 1.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et le nouveau EFLAGS.TF vaut 1.

#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction PUSHF
Instruction assembleur 80x86 - Instruction PUSHFD
Instruction assembleur 80x86 - Instruction PUSHFQ

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 234 à 237.

Assembleur 80x86 POPFQ

x86-64+

POP to rFLAGS Quadword

Syntaxe

```
POPFQ
```

Description

Cette instruction permet de désempiler de la pile le registre 64 bits de drapeau (RFLAGS) contenant les indicateurs d'état.

Algorithme

```
Registre de drapeaux ← SS:(E)SP
(E)SP ← (E)SP + 8
```

Mnémonique

Instruction	Opcode	Description
POPFQ	9Dh	Désempile le quadruple mot de la pile et la met dans le registres 64 bits des drapeaux (*RFLAGS*).

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-	X	X	X	Une adresse mémoire

canonique)				dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et CR4.VME vaut 0.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et une opérande est de taille de 32 bits.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et les bits EFLAGS.VIP et le nouveau EFLAGS.IF valent 1.
		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et le nouveau EFLAGS.TF vaut 1.

#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction PUSHF
Instruction assembleur 80x86 - Instruction PUSHFD
Instruction assembleur 80x86 - Instruction PUSHFQ

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 234 à 237.

Assembleur 80x86 — POR

INTEL Pentium MMX+ — *Bitwise Logical OR*

Syntaxe

POR *destination, source*

Description

Cette instruction permet d'effectuer un «*OU BINAIRE*» d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.

Algorithme

destination ← *destination* U *source*

Mnémonique

Instruction	Opcode	Description
POR *mm,m64*	0Fh EBh /r	Cette instruction permet d'effectuer un «*OU BINAIRE*» d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 238 à 240.

Assembleur 80x86 PREFETCH

AMD K6-2+

Prefetch L1 Data-Cache Line

Syntaxe

PREFETCH *mem8*

Description

Cette instruction permet d'effectuer le chargement de l'entrée d'une séquence d'alignement de mémoire de 64 bits de l'adresse mémoire spécifié dans le cache de données *L1* du microprocesseur.

Mnémonique

Instruction	Opcode	Description
PREFETCH *mem8*	0Fh 0Dh /0	Charge la ligne de cache dans les données de cache de la ligne L1

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Cette instruction n'est pas supporté, comme indiqué par le bit 8 du registre ECX de la fonction

				8000_00001h de l'instruction *CPUID*.
	X	X	X	Le mode long n'est pas supporté, comme indiqué par le bit 29 du registre EDX de la fonction 8000_00001h de l'instruction *CPUID*.
	X	X	X	Les instructions «3DNow!» ne sont pas supportés, comme indiqué par le bit 31 du registre EDX de la fonction 8000_00001h de l'instruction *CPUID*.
	X	X	X	L'opérande est un registre.

Voir également

Instruction assembleur 80x86 - Instruction PREFETCH0
Instruction assembleur 80x86 - Instruction PREFETCH1
Instruction assembleur 80x86 - Instruction PREFETCH2

Assembleur 80x86 PREFETCH0

INTEL Pentium III (SSE)+ *Prefetch Data to Cache Level T0*

Syntaxe

PREFETCH0 *mem8*

Description

Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T0* du microprocesseur. Alias de *PREFETCHT0*.

Mnémonique

Instruction	Opcode	Description
PREFETCH0 *mem8*	0Fh 18h /1	Copie les données du processeur en utilisant la référence *T0*.

Exception

Aucune

Voir également

Instruction assembleur 80x86 - Instruction PREFETCH
Instruction assembleur 80x86 - Instruction PREFETCHW

Assembleur 80x86 — PREFETCH1

INTEL Pentium III (SSE)+ — *Prefetch Data to Cache Level T1*

Syntaxe

PREFETCH1 *mem8*

Description

Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T1* du microprocesseur. Alias de *PREFETCHT1*.

Mnémonique

Instruction	Opcode	Description
PREFETCH1 *mem8*	0Fh 18h /2	Copie les données du processeur en utilisant la référence *T1*.

Exception

Aucune

Voir également

Instruction assembleur 80x86 - Instruction PREFETCH
Instruction assembleur 80x86 - Instruction PREFETCHW

Assembleur 80x86 — PREFETCH2

INTEL Pentium III (SSE)+ — *Prefetch Data to Cache Level T2*

Syntaxe

PREFETCH2 *mem8*

Description

Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T2* du microprocesseur. Alias de *PREFETCHT2*.

Mnémonique

Instruction	Opcode	Description
PREFETCH2 *mem8*	0Fh 18h /3	Copie les données du processeur en utilisant la référence *T2*.

Exception

Aucune

Voir également

Instruction assembleur 80x86 - Instruction PREFETCH
Instruction assembleur 80x86 - Instruction PREFETCHW

Assembleur 80x86 — PREFETCHNTA

INTEL Pentium III (SSE)+ — *Prefetch Data to Cache Level NTA*

Syntaxe

PREFETCHNTA *mem8*

Description

Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *NTA* du microprocesseur.

Mnémonique

Instruction	Opcode	Description
PREFETCHNTA *mem8*	0Fh 18h /0	Copie les données du processeur en utilisant la référence *NTA*.

Exception

Aucune

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 241 à 243.

Assembleur 80x86 PREFETCHT0

INTEL Pentium III (SSE)+ *Prefetch Data to Cache Level T0*

Syntaxe

PREFETCHT0 *mem8*

Description

Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T0* du microprocesseur. Alias de *PREFETCH0*.

Mnémonique

Instruction	Opcode	Description
PREFETCHT0 *mem8*	0Fh 18h /1	Copie les données du processeur en utilisant la référence *T0*.

Exception

Aucune

Voir également

Instruction assembleur 80x86 - Instruction PREFETCH
Instruction assembleur 80x86 - Instruction PREFETCHW

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 241 à 243.

Assembleur 80x86 PREFETCHT1

INTEL Pentium III (SSE)+

Prefetch Data to Cache Level T1

Syntaxe

PREFETCHT1 *mem8*

Description

Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T1* du microprocesseur. Alias de *PREFETCH1*.

Mnémonique

Instruction	Opcode	Description
PREFETCHT1 *mem8*	0Fh 18h 2	Copie les données du processeur en utilisant la référence *T1*.

Exception

Aucune

Voir également

Instruction assembleur 80x86 - Instruction PREFETCH
Instruction assembleur 80x86 - Instruction PREFETCHW

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 241 à 243.

Assembleur 80x86 PREFETCHT2

INTEL Pentium III (SSE)+ *Prefetch Data to Cache Level T2*

Syntaxe

PREFETCHT2 *mem8*

Description

Cette instruction permet d'effectuer le chargement de la ligne de cache de l'adresse mémoire spécifié dans le cache de données *T2* du microprocesseur. Alias de *PREFETCH2*.

Mnémonique

Instruction	Opcode	Description
PREFETCHT2 *mem8*	0Fh 18h /3	Copie les données du processeur en utilisant la référence *T2*.

Exception

Aucune

Voir également

Instruction assembleur 80x86 - Instruction PREFETCH
Instruction assembleur 80x86 - Instruction PREFETCHW

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 241 à 243.

Assembleur 80x86 PREFETCHW

AMD K6-2+

Prefetch L1 Data-Cache Line Write

Syntaxe

PREFETCHW *mem8*

Description

Cette instruction permet d'effectuer le chargement de l'entrée d'une séquence d'alignement de mémoire de 64 bits de l'adresse mémoire spécifié dans le cache de données *L1* du microprocesseur dans un état de modification.

Mnémonique

Instruction	Opcode	Description
PREFETCHW *mem8*	0Fh 0Dh /1	Charge la ligne de cache dans les données de cache de la ligne L1 et marque qu'il est modifié.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)	X	X	X	Cette instruction n'est pas supporté, comme indiqué par le bit 8 du registre ECX

				de la fonction 8000_00001h de l'instruction *CPUID*.
	X	X	X	Le mode long n'est pas supporté, comme indiqué par le bit 29 du registre EDX de la fonction 8000_00001h de l'instruction *CPUID*.
	X	X	X	Les instructions «3DNow!» ne sont pas supportés, comme indiqué par le bit 31 du registre EDX de la fonction 8000_00001h de l'instruction *CPUID*.
	X	X	X	L'opérande est un registre.

Voir également

Instruction assembleur 80x86 - Instruction PREFETCH0
Instruction assembleur 80x86 - Instruction PREFETCH1
Instruction assembleur 80x86 - Instruction PREFETCH2

Assembleur 80x86 PSADBW

INTEL Pentium III (SSE)+ *Compute Sum of Absolute Differences*

Syntaxe

PSADBW *destination*, *source*

Description

Cette instruction permet d'effectuer le calcul la valeur de absolue de la différence des octets de paquet contenu dans une opérande source et destination.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  Temporaire0 ← | *destination*(0..7) - *source*(0..7) |
  Temporaire1 ← | *destination*(8..15) - *source*(8..15) |
  Temporaire2 ← | *destination*(16..23) - *source*(16..23) |
  Temporaire3 ← | *destination*(24..31) - *source*(24..31) |
  Temporaire4 ← | *destination*(32..39) - *source*(32..39) |
  Temporaire5 ← | *destination*(40..47) - *source*(40..47) |
  Temporaire6 ← | *destination*(48..55) - *source*(48..55) |
  Temporaire7 ← | *destination*(56..63) - *source*(56..63) |
  destination(0..15) ← SOMME(Temporaire0..Temporaire7)
  destination(16..63) ← 0
SINON
  Temporaire0 ← | *destination*(0..7) - *source*(0..7) |
  Temporaire1 ← | *destination*(8..15) - *source*(8..15) |
  Temporaire2 ← | *destination*(16..23) - *source*(16..23) |
  Temporaire3 ← | *destination*(24..31) - *source*(24..31) |
  Temporaire4 ← | *destination*(32..39) - *source*(32..39) |
  Temporaire5 ← | *destination*(40..47) - *source*(40..47) |
  Temporaire6 ← | *destination*(48..55) - *source*(48..55) |
  Temporaire7 ← | *destination*(56..63) - *source*(56..63) |
  Temporaire8 ← | *destination*(64..71) - *source*(64..71) |

```
  Temporaire9 ← | destination(72..79) - source(72..79) |
  Temporaire10 ← | destination(80..87) - source(80..87) |
  Temporaire11 ← | destination(88..95) - source(88..95) |
  Temporaire12 ← | destination(96..103) - source(96..103) |
  Temporaire13 ← | destination(104..111) - source(104..111) |
  Temporaire14 ← | destination(112..119) - source(112..119) |
  Temporaire15 ← | destination(120..127) - source(120..127) |
  Destination(0..15) ← SOMME(Temporaire0...Temporaire7)
  Destination(16..63) ← 0
  Destination(64..79) ← SOMME(Temporaire8...Temporaire15)
  Destination(80..127) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PSADBW *mm1, mm2/m64*	0F F6 /r	Cette instruction permet d'effectuer le calcul la valeur de absolue de la différence des octets de paquet contenu dans une opérande source et destination.
PSADBW *xmm1, xmm2/m128*	66 0F F6 /r	Cette instruction permet d'effectuer le calcul la valeur de absolue de la différence des octets de paquet contenu dans une opérande source et destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 244 à 247.

Assembleur 80x86 PSHUFB

SSSE3+ *Packed Shuffle Bytes*

Syntaxe

PSHUFB *destination*, *source*

Description

Cette instruction permet d'effectuer un mélange des octets en place dans l'opérande de destination avec un masque de contrôle dans l'opérande source.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  POUR POUR i ← 0 JUSQU'A 7
    SI source((i x 8) + 7) = 1 ALORS
      destination((i x 8) + 7 ..(i x 8) + 0) ← 0
    SINON
      index(2..0) ← source((i x 8) + 2 .. (i x 8) + 0)
      destination((i x 8) + 7 ..(i x 8) + 0) ← destination((index x 8 + 7)..(index x 8 + 0))
    FIN SI
  FIN BOUCLE POUR
SINON SI taille de l'opérande = 128 bits ALORS
  BOUCLE POUR i ← 0 JUSQU'A 15
    SI source((i x 8) + 7) = 1 ALORS
      destination((i x 8) + 7 .. (i x 8) + 0) ← 0
    SINON
      index(3..0) ← source((i x 8) + 3 .. (i x 8) + 0)
      destination((i x 8) + 7 .. (i x 8) + 0) ← destination((index x 8 + 7)..(index x 8 + 0))
    FIN SI
  FIN BOUCLE POUR
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PSHUFB *mm1,mm2/m64*	0Fh 38h 00h /r	Cette instruction permet d'effectuer un mélange des octets en place dans l'opérande de destination avec un masque de contrôle dans l'opérande source.
PSHUFB *xmm1,xmm2/m128*	66h 0Fh 38h 00h /r	Cette instruction permet d'effectuer un mélange des octets en place dans l'opérande de destination avec un masque de contrôle dans l'opérande source.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 247 à 251.

Assembleur 80x86 PSHUFD

INTEL Pentium 4 (SSE2)+ *Shuffle Packed Doublewords*

Syntaxe

PSHUFD *destination*, *source*, *ordre*

Description

Cette instruction permet de copier un double mots d'un opérande source avec emplacement spécifié par un opérande immédiate dans un opérande de destination.

Algorithme

destination(0..31) ← (*source* >> (*ordre*(0..1) x 32))(0..31)
destination(32..63) ← (*source* >> (*ordre*(2..3) x 32))(0..31)
destination(64..95) ← (*source* >> (*ordre*(4..5) x 32))(0..31)
destination(96..127) ← (*source* >> (*ordre*(6..7) x 32))(0..31)

Mnémonique

Instruction	Opcode	Description
PSHUFD *xmm1*, *xmm2/m128*, *imm8*	66h 0Fh 70h /r *ib*	Cette instruction permet de copier un double mots d'un opérande source avec emplacement spécifié par un opérande immédiate dans un opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 252 à 254.*

Assembleur 80x86 — PSHUFHW

INTEL Pentium 4 (SSE2)+ — *Shuffle Packed High Words*

Syntaxe

PSHUFHW *destination*, *source*, *ordre*

Description

Cette instruction permet de copier un mot d'une opérande source avec emplacement spécifié par un opérande immédiate dans la partie haute d'un opérande de destination.

Algorithme

destination(0..63) ← *source*(0..63)
destination(64..79) ← (*source* >> (*ordre*(0..1) x 16))(64..79)
destination(80..95) ← (*source* >> (*ordre*(2..3) x 16))(64..79)
destination(96..111) ← (*source* >> (*ordre*(4..5) x 16))(64..79)
destination(112..127) ← (*source* >> (*ordre*(6..7) x 16))(64..79)

Mnémonique

Instruction	Opcode	Description
PSHUFHW *xmm1*, *xmm2/m128*, *imm8*	F3h 0Fh 70h /r *ib*	Cette instruction permet de copier un mot d'une opérande source avec emplacement spécifié par un opérande immédiate dans la partie haute d'un opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 255 à 257.

Assembleur 80x86 — PSHUFLW

INTEL Pentium 4 (SSE2)+ — *Shuffle Packed Low Words*

Syntaxe

PSHUFLW *destination*, *source*, *ordre*

Description

Cette instruction permet de copier un mot d'une opérande source avec emplacement spécifié par un opérande immédiate dans la partie basse d'un opérande de destination.

Algorithme

destination(0..15) ← (*source* >> (*ordre*(0..1) x 16))(0..15)
destination(16..31) ← (*source* >> (*ordre*(2..3) x 16))(0..15)
destination(32..47) ← (*source* >> (*ordre*(4..5) x 16))(0..15)
destination(48..63) ← (*source* >> (*ordre*(6..7) x 16))(0..15)
destination(64..127) ← *source*(64..127)

Mnémonique

Instruction	Opcode	Description
PSHUFLW *xmm1*, *xmm2/m128*, *imm8*	F2h 0Fh 70h /r *ib*	Cette instruction permet de copier un mot d'une opérande source avec emplacement spécifié par un opérande immédiate dans la partie basse d'un opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 258 à 260.

Assembleur 80x86 — PSHUFW

INTEL Pentium III (SSE)+ — *Shuffle Packed Words*

Syntaxe

PSHUFW *destination*, *source*, *ordre*

Description

Cette instruction permet de copier une double mots d'un opérande source avec emplacement spécifié par un opérande immédiate dans un opérande de destination.

Algorithme

destination(0..15) ← (*source* >> (*ordre*(0..1) x 16))(0..15)
destination(16..31) ← (*source* >> (*ordre*(2..3) x 16))(0..15)
destination(32..47) ← (*source* >> (*ordre*(4..5) x 16))(0..15)
destination(48..63) ← (*source* >> (*ordre*(6..7) x 16))(0..15)

Mnémonique

Instruction	Opcode	Description
PSHUFW *mm1*, *mm2/m64*, *imm8*	0Fh 70h /r *ib*	Cette instruction permet de copier une double mots d'un opérande source avec emplacement spécifié par un opérande immédiate dans un opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 261 à 263.*

Assembleur 80x86 — PSIGNB

SSSE3+ — *Packed Sign Byte*

Syntaxe

PSIGNB *destination*, *source*

Description

Cette instruction permet d'effectuer la négation de chaque octets de l'opérande de destination si la valeur du signe de l'entier des octets correspondant à l'opérande source est inférieur à 0.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  SI source(7..0) < 0 ALORS
    destination(7..0) ← Neg(destination(7..0))
  SINON SI source(7..0) = 0 ALORS
    DEST(7..0) ← 0
  SINON SI source(7..0) > 0 ALORS
    destination(7..0) ← destination(7..0)
  FIN SI
  SI source(15..8) < 0 ALORS
    destination(15..8) ← Neg(destination(15..8))
  SINON SI source(15..8) = 0 ALORS
    destination(15..8) ← 0
  SINON SI source(15..8) > 0 ALORS
    destination(15..8) ← destination(15..8)
  FIN SI
  SI source(23..16) < 0 ALORS
    destination(23..16) ← Neg(destination(23..16))
  SINON SI source(23..16) = 0 ALORS
```

```
    destination(23..16) ← 0
  SINON SI source(23..16) > 0 ALORS
    destination(23..16) ← destination(23..16)
  FIN SI
  SI source(31..24) < 0 ALORS
    destination(31..24) ← Neg(destination(31..24))
  SINON SI source(31..24) = 0 ALORS
    destination(31..24) ← 0
  SINON SI source(31..24) > 0 ALORS
    destination(31..24) ← destination(31..24)
  FIN SI
  SI source(39..32) < 0 ALORS
    destination(39..32) ← Neg(destination(39..32))
  SINON SI source(39..32) = 0 ALORS
    destination(39..32) ← 0
  SINON SI source(39..32) > 0 ALORS
    destination(39..32) ← destination(39..32)
  FIN SI
  SI source(47..40) < 0 ALORS
    destination(47..40) ← Neg(destination(47..40))
  SINON SI source(47..40) = 0 ALORS
    destination(47..40) ← 0
  SINON SI source(47..40) > 0 ALORS
    destination(47..40) ← destination(47..40)
  FIN SI
  SI source(55..48) < 0 ALORS
    destination(55..48) ← Neg(destination(55..48))
  SINON SI source(55..48) = 0 ALORS
    destination(55..48) ← 0
  SINON SI source(55..48) > 0 ALORS
    destination(55..48) ← destination(55..48)
  FIN SI
  SI source(63..56) < 0 ALORS
    destination(63..56) ← Neg(destination(63..56))
  SINON SI source(63..56) = 0 ALORS
    destination(63..56) ← 0
  SINON SI source(63..56) > 0 ALORS
    destination(63..56) ← destination(63..56)
  FIN SI
SINON SI taille de l'opérande = 128 bits ALORS
  SI source(7..0) < 0 ALORS
```

```
  destination(7..0) ← Neg(destination(7..0))
SINON SI source(7..0) = 0 ALORS
  destination(7..0) ← 0
SINON SI source(7..0) > 0 ALORS
  destination(7..0) ← destination(7..0)
FIN SI
SI source(15..8) < 0 ALORS
  destination(15..8) ← Neg(destination(15..8))
SINON SI source(15..8) = 0 ALORS
  destination(15..8) ← 0
SINON SI source(15..8) > 0 ALORS
  destination(15..8) ← destination(15..8)
FIN SI
SI source(23..16) < 0 ALORS
  destination(23..16) ← Neg(destination(23..16))
SINON SI source(23..16) = 0 ALORS
  destination(23..16) ← 0
SINON SI source(23..16) > 0 ALORS
  destination(23..16) ← destination(23..16)
FIN SI
SI source(31..24) < 0 ALORS
  destination(31..24) ← Neg(destination(31..24))
SINON SI source(31..24) = 0 ALORS
  destination(31..24) ← 0
SINON SI source(31..24) > 0 ALORS
  destination(31..24) ← destination(31..24)
FIN SI
SI source(39..32) < 0 ALORS
  destination(39..32) ← Neg(destination(39..32))
SINON SI source(39..32) = 0 ALORS
  destination(39..32) ← 0
SINON SI source(39..32) > 0 ALORS
  destination(39..32) ← destination(39..32)
FIN SI
SI source(47..40) < 0 ALORS
  destination(47..40) ← Neg(destination(47..40))
SINON SI source(47..40) = 0 ALORS
  destination(47..40) ← 0
SINON SI source(47..40) > 0 ALORS
  destination(47..40) ← destination(47..40)
FIN SI
```

```
SI source(55..48) < 0 ALORS
  destination(55..48) ← Neg(destination(55..48))
SINON SI source(55..48) = 0 ALORS
  destination(55..48) ← 0
SINON SI source(55..48) > 0 ALORS
  destination(55..48) ← destination(55..48)
FIN SI
SI source(63..56) < 0 ALORS
  destination(63..56) ← Neg(destination(63..56))
SINON SI source(63..56) = 0 ALORS
  destination(63..56) ← 0
SINON SI source(63..56) > 0 ALORS
  destination(63..56) ← destination(63..56)
FIN SI
SI source(71..64) < 0 ALORS
  destination(71..64) ← Neg(destination(71..64))
SINON SI source(71..64) = 0 ALORS
  destination(71..64) ← 0
SINON SI source(71..64) > 0 ALORS
  destination(71..64) ← destination(71..64)
FIN SI
SI source(79..72) < 0 ALORS
  destination(79..72) ← Neg(destination(79..72))
SINON SI source(79..72) = 0 ALORS
  destination(79..72) ← 0
SINON SI source(79..72) > 0 ALORS
  destination(79..72) ← destination(79..72)
FIN SI
SI source(87..80) < 0 ALORS
  destination(87..80) ← Neg(destination(87..80))
SINON SI source(87..80) = 0 ALORS
  destination(87..80) ← 0
SINON SI source(87..80) > 0 ALORS
  destination(87..80) ← destination(87..80)
FIN SI
SI source(95..88) < 0 ALORS
  destination(95..88) ← Neg(destination(95..88))
SINON SI source(95..88) = 0 ALORS
  destination(95..88) ← 0
SINON SI source(95..88) > 0 ALORS
  destination(95..88) ← destination(95..88)
```

```
  FIN SI
  SI source(103..96) < 0 ALORS
    destination(103..96) ← Neg(destination(103..96))
  SINON SI source(103..96) = 0 ALORS
    destination(103..96) ← 0
  SINON SI source(103..96) > 0 ALORS
    destination(103..96) ← destination(103..96)
  FIN SI
  SI source(111..104) < 0 ALORS
    destination(111..104) ← Neg(destination(111..104))
  SINON SI source(111..104) = 0 ALORS
    destination(111..104) ← 0
  SINON SI source(111..104) > 0 ALORS
    destination(111..104) ← destination(111..104)
  FIN SI
  SI source(119..112) < 0 ALORS
    destination(119..112) ← Neg(destination(119..112))
  SINON SI source(119..112) = 0 ALORS
    destination(119..112) ← 0
  SINON SI source(119..112) > 0 ALORS
    destination(119..112) ← destination(119..112)
  FIN SI
  SI source(127..120) < 0 ALORS
    destination(127..120) ← Neg(destination(127..120))
  SINON SI source(127..120) = 0 ALORS
    destination(127..120) ← 0
  SINON SI SRC(127..120) > 0 ALORS
    destination(127..120) ← destination(127..120)
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PSIGNB *mm1,mm2/m64*	0Fh 38h 08h /r	Cette instruction permet d'effectuer la négation de chaque octets de l'opérande de destination si la valeur du signe de l'entier des octets

		correspondant à l'opérande source est inférieur à 0.
PSIGNB *xmm1,xmm2/m128*	66h 0Fh 38h 08h /r	Cette instruction permet d'effectuer la négation de chaque octets de l'opérande de destination si la valeur du signe de l'entier des octets correspondant à l'opérande source est inférieur à 0.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 264 à 269.

Assembleur 80x86 — PSIGND

SSSE3+ — *Packed Sign Doubleword*

Syntaxe

PSIGND *destination*, *source*

Description

Cette instruction permet d'effectuer la négation de chaque double mots de l'opérande de destination si la valeur du signe de l'entier des double mots correspondant à l'opérande source est inférieur à 0.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  SI source(31..0) < 0 ALORS
    destination(31..0) ← Neg(destination(31..0))
  SINON SI source(31..0) = 0 ALORS
    destination(31..0) ← 0
  SINON SI source(31..0) > 0 ALORS
    destination(31..0) ← destination(31..0)
  FIN SI
  SI source(63..32) < 0 ALORS
    destination(63..32) ← Neg(destination(63..32))
  SINON SI source(63..32) = 0 ALORS
    destination(63..32) ← 0
  SINON SI source(63..32) > 0 ALORS
    destination(63..32) ← destination(63..32)
  FIN SI
SINON SI taille de l'opérande = 128 bits ALORS
  SI source(31..0) < 0 ALORS
    destination(31..0) ← Neg(destination(31..0))
  SINON SI source(31..0) = 0 ALORS
    destination(31..0) ← 0
```

```
  SINON SI source(31..0) > 0 ALORS
    destination(31..0) ← destination(31..0)
  FIN SI
  SI source(63..32) < 0 ALORS
    destination(63..32) ← Neg(destination(63..32))
  SINON SI source(63..32) = 0 ALORS
    destination(63..32) ← 0
  SINON SI source(63..32) > 0 ALORS
    destination(63..32) ← destination(63..32)
  FIN SI
  SI source(95..64) < 0 ALORS
    destination(95..64) ← Neg(destination(95..64))
  SINON SI source(95..64) = 0 ALORS
    destination(95..64) ← 0
  SINON SI source(95..64) > 0 ALORS
    destination(95..64) ← destination(95..64)
  FIN SI
  SI source(127..96) < 0 ALORS
    destination(127..96) ← Neg(destination(127..96))
  SINON SI source(127..96) = 0 ALORS
    destination(127..96) ← 0
  SINON SI source(127..96) > 0 ALORS
    destination(127..96) ← destination(127..96)
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PSIGND *mm1,mm2/m64*	0Fh 38h 0Ah /r	Cette instruction permet d'effectuer la négation de chaque double mots de l'opérande de destination si la valeur du signe de l'entier des double mots correspondant à l'opérande source est inférieur à 0.
PSIGND *xmm1,xmm2/m128*	66h 0Fh 38h 0Ah /r	Cette instruction permet d'effectuer la négation de chaque double mots

		de l'opérande de destination si la valeur du signe de l'entier des double mots correspondant à l'opérande source est inférieur à 0.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 264 à 269.

Assembleur 80x86 PSIGNW

SSSE3+

Packed Sign Word

Syntaxe

PSIGNW *destination*, *source*

Description

Cette instruction permet d'effectuer la négation de chaque mots de l'opérande de destination si la valeur du signe de l'entier des mots correspondant à l'opérande source est inférieur à 0.

Algorithme

```
SI taille de l'opérande = 64 bits ALORS
  SI source(15..0) < 0 ALORS
    destination(15..0) ← Neg(destination(15..0))
  SINON SI source(15..0) = 0 ALORS
    destination(15..0) ← 0
  SINON SI source(15..0) > 0 ALORS
    destination(15..0) ← destination(15..0)
  FIN SI    SI source(31..16) < 0 ALORS
    destination(31..16) ← Neg(destination(31..16))
  SINON SI source(31..16) = 0 ALORS
    destination(31..16) ← 0
  SINON SI source(31..16) > 0 ALORS
    destination(31..16) ← destination(31..16)
  FIN SI    SI source(47..32) < 0 ALORS
    destination(47..32) ← Neg(destination(47..32))
  SINON SI source(47..32) = 0 ALORS
    destination(47..32) ← 0
  SINON SI source(47..32) > 0 ALORS
    destination(47..32) ← destination(47..32)
  FIN SI    SI source(63..48) < 0 ALORS
```

```
  destination(63..48) ← Neg(destination(63..48))
SINON SI source(63..48) = 0 ALORS
  destination(63..48) ← 0
SINON SI source(63..48) > 0 ALORS
  destination(63..48) ← destination(63..48)
FIN SI SINON SI taille de l'opérande = 128 bits ALORS
SI source(15..0) < 0 ALORS
  destination(15..0) ← Neg(destination(15..0))
SINON SI source(15..0) = 0 ALORS
  destination(15..0) ← 0
SINON SI source(15..0) > 0 ALORS
  destination(15..0) ← destination(15..0)
FIN SI
SI source(31..16) < 0 ALORS
  destination(31..16) ← Neg(destination(31..16))
SINON SI source(31..16) = 0 ALORS
  destination(31..16) ← 0
SINON SI source(31..16) > 0 ALORS
  destination(31..16) ← destination(31..16)
FIN SI
SI source(47..32) < 0 ALORS
  destination(47..32) ← Neg(destination(47..32))
SINON SI source(47..32) = 0 ALORS
  destination(47..32) ← 0
SINON SI source(47..32) > 0 ALORS
  destination(47..32) ← destination(47..32)
FIN SI
SI source(63..48) < 0 ALORS
  destination(63..48) ← Neg(destination(63..48))
SINON SI source(63..48) = 0 ALORS
  destination(63..48) ← 0
SINON SI source(63..48) > 0 ALORS
  destination(63..48) ← destination(63..48)
FIN SI
SI source(79..64) < 0 ALORS
  destination(79..64) ← Neg(destination(79..64))
SINON SI source(79..64) = 0 ALORS
  destination(79..64) ← 0
SINON SI source(79..64) > 0 ALORS
  destination(79..64) ← destination(79..64)
FIN SI
```

```
  SI source(95..80) < 0 ALORS
    destination(95..80) ← Neg(destination(95..80))
  SINON SI source(95..80) = 0 ALORS
    destination(95..80) ← 0
  SINON SI source(95..80) > 0 ALORS
    destination(95..80) ← destination(95..80)
  FIN SI
  SI source(111..96) < 0 ALORS
    destination(111..96) ← Neg(destination(111..96))
  SINON SI source(111..96) = 0 ALORS
    destination(111..96) ← 0
  SINON SI source(111..96) > 0 ALORS
    destination(111..96) ← destination(15..0)
  FIN SI
  SI source(127..112) < 0 ALORS
    destination(127..112) ← Neg(destination(127..112))
  SINON SI source(127..112) = 0 ALORS
    destination(127..112) ← 0
  SINON SI source(127..112) > 0 ALORS
    destination(127..112) ← destination(127..112)
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PSIGNW *mm1,mm2/m64*	0Fh 38h 09h /r	Cette instruction permet d'effectuer la négation de chaque mots de l'opérande de destination si la valeur du signe de l'entier des mots correspondant à l'opérande source est inférieur à 0.
PSIGNW *xmm1,xmm2/m128*	66h 0Fh 38h 09h /r	Cette instruction permet d'effectuer la négation de chaque mots de l'opérande de destination si la valeur du signe de l'entier des mots correspondant à l'opérande source

		est inférieur à 0.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 264 à 269.

Assembleur 80x86 — PSLLD

INTEL Pentium Pro+ — *Packed Shift Left Logical Dwords*

Syntaxe

PSLLD *destination*, *source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de double mots.

Algorithme

destination(31..0) ← *destination*(31..0) << *source*
destination(63..32) ← *destination*(63..32) << *source*

Mnémonique

Instruction	Opcode	Description
PSLLD *mm,mm/m64*	0Fh F2h /r	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de double mots.
PSLLD *mm,Imm8*	0Fh 72h /6 *Imm8*	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de double mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 272 à 277.

Assembleur 80x86 — PSLLDQ

INTEL Pentium 4 (SSE2)+ — *Packed Shift Double Quadword Left Logical*

Syntaxe

PSLLDQ *destination*, *source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de double quadruple mots.

Algorithme

```
Temporaire ← source
SI Temporaire > 15 ALORS
  Temporaire ← 16
FIN SI
destination ← destination << (Temporaire x 8)
```

Mnémonique

Instruction	Opcode	Description
PSLLDQ *xmm1*, *imm8*	66h 0Fh 73h /7 ib	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de double quadruple mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 270 à 271.*

Assembleur 80x86 — PSLLQ

INTEL Pentium Pro+ — *Packed Shift Left Logical Qwords*

Syntaxe

PSLLQ *destination*, *source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un quadruple mot.

Algorithme

destination(63..0) ← *destination*(63..0) << *source*

Mnémonique

Instruction	Opcode	Description
PSLLQ *mm,mm/m64*	0Fh F3h /r	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un quadruple mot.
PSLLQ *mm,Imm8*	0Fh 73h /6 *Imm8*	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un quadruple mot.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 270 à 271.

Assembleur 80x86 — PSLLW

INTEL Pentium Pro+ — *Packed Shift Left Logical Words*

Syntaxe

PSLLW *destination, source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de mots.

Algorithme

destination(15..0) ← *destination*(15..0) << *source*
destination(31..16) ← *destination*(31..16) << *source*
destination(47..32) ← *destination*(47..32) << *source*
destination(63..48) ← *destination*(63..48) << *source*

Mnémonique

Instruction	Opcode	Description
PSLLW *mm,mm/m64*	0Fh F1h /r	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de mots.
PSLLW *mm,Imm8*	0Fh 71h /6 *Imm8*	Cette instruction permet d'effectuer le décalage de bits vers la gauche d'un paquet de mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 270 à 271.

Assembleur 80x86 — PSRAD

INTEL Pentium Pro+ — *Packed Shift Right Arithmetic Dwords*

Syntaxe

PSRAD *destination, source*

Description

Cette instruction permet d'effectuer le décalage arithmétique de bits vers la droite d'un paquet de double mots.

Algorithme

destination(31..0) ← SignExtend(*destination*(31..0)) >> *source*
destination(63..32) ← SignExtend(*destination*(63..32)) >> *source*

Mnémonique

Instruction	Opcode	Description
PSRAD *mm,mm/m64*	0Fh E2h /r	Cette instruction permet d'effectuer le décalage arithmétique de bits vers la droite d'un paquet de double mots.
PSRAD *mm,Imm8*	0Fh 72h /4 *Imm8*	Cette instruction permet d'effectuer le décalage arithmétique de bits vers la droite d'un paquet de double mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 278 à 282.

Assembleur 80x86 — PSRAW

INTEL Pentium Pro+ — *Packed Shift Right Arithmetic Words*

Syntaxe

PSRAW *destination*, *source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de mots.

Algorithme

destination(15..0) ← SignExtend(*destination*(15..0)) >> *source*
destination(31..16) ← SignExtend(*destination*(31..16)) >> *source*
destination(47..32) ← SignExtend(*destination*(47..32)) >> *source*
destination(63..48) ← SignExtend(*destination*(63..48)) >> *source*

Mnémonique

Instruction	Opcode	Description
PSRAW *mm,mm/m64*	0Fh E1h /r	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de mots.
PSRAW *mm,Imm8*	0Fh 71h /4 *Imm8*	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 278 à 282.*

Assembleur 80x86 — PSRLD

INTEL Pentium Pro+ — *Packed Shift Right Logical Dwords*

Syntaxe

PSRLD *destination*, *source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de double mots.

Algorithme

destination(31..0) ← *destination*(31..0) >> *source*
destination(63..32) ← *destination*(63..32) >> *source*

Mnémonique

Instruction	Opcode	Description
PSRLD *mm,mm/m64*	0Fh D2h /r	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de double mots./td>
PSRLD *mm,Imm8*	0Fh 72h /2 *Imm8*	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de double mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 285 à 289.

Assembleur 80x86 — PSRLDQ

INTEL Pentium 4 (SSE2)+ — *Shift Double Quadword Right Logical*

Syntaxe

PSRLDQ *destination*, *source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de double quadruple mots.

Algorithme

Temporaire ← *source*
SI Temporaire > 15 **ALORS**
  Temporaire ← 16
FIN SI
destination = *destination* >> (Temporaire x 8)

Mnémonique

Instruction	Opcode	Description
PSRLDQ *xmm1*, *imm8*	66h 0Fh 73h /3 *ib*	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de double quadruple mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 283 à 284.

Assembleur 80x86 — PSRLQ

INTEL Pentium Pro+ — *Packed Shift Right Logical Qwords*

Syntaxe

PSRLQ *destination*, *source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la droite d'un quadruple mot.

Algorithme

destination(63..0) ← *destination*(63..0) >> *source*

Mnémonique

Instruction	Opcode	Description
PSRLQ *mm,mm/m64*	0Fh D3h /r	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un quadruple mot.
PSRLQ *mm,Imm8*	0Fh 73h /2 *Imm8*	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un quadruple mot.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 285 à 289.

Assembleur 80x86 — PSRLW

INTEL Pentium Pro+ — *Packed Shift Right Logical Words*

Syntaxe

PSRLW *destination*, *source*

Description

Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de mots.

Algorithme

destination(15..0) ← *destination*(15..0) >> *source*
destination(31..16) ← *destination*(31..16) >> *source*
destination(47..32) ← *destination*(47..32) >> *source*
destination(63..48) ← *destination*(63..48) >> *source*

Mnémonique

Instruction	Opcode	Description
PSRLW *mm,mm/m64*	0Fh D1h /r	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de mots.
PSRLW *mm,Imm8*	0Fh 71h /2 *Imm8*	Cette instruction permet d'effectuer le décalage de bits vers la droite d'un paquet de mots.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 285 à 289.*

Assembleur 80x86 — PSUBB

INTEL Pentium Pro+ — *Packed Subtract Bytes*

Syntaxe

PSUBB *destination, source*

Description

Cette instruction permet d'effectuer la soustraction de la valeur de chacun des octets d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

destination(7..0) ← *destination*(7..0) - *source*(7..0)
destination(15..8) ← *destination*(15..8) - *source*(15..8)
destination(23..16) ← *destination*(23..16) - *source*(23..16)
destination(31..24) ← *destination*(31..24) - *source*(31..24)
destination(39..32) ← *destination*(39..32) - *source*(39..32)
destination(47..40) ← *destination*(47..40) - *source*(47..40)
destination(55..48) ← *destination*(55..48) - *source*(55..48)
destination(63..56) ← *destination*(63..56) - *source*(63..56)

Mnémonique

Instruction	Opcode	Description
PSUBB mm,mm/m64	0Fh F8h /r	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des octets d'un paquet contenu dans des opérandes de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 290 à 293.

Assembleur 80x86 — PSUBD

INTEL Pentium Pro+ — *Packed Subtract Dwords*

Syntaxe

PSUBD *destination, source*

Description

Cette instruction permet d'effectuer la soustraction de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

destination(31..0) ← *destination*(31..0) - *source*(31..0) *destination*(63..32) ← *destination*(63..48) - *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PSUBD *mm,mm/m64*	0Fh FAh /r	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des doubles mots d'un paquet contenu dans des opérandes de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 290 à 293.

Assembleur 80x86 PSUBQ

INTEL Pentium 4 (SSE2)+ *Subtract Packed Quadword*

Syntaxe

PSUBQ *destination*, *source*

Description

Cette instruction permet d'effectuer la soustraction de la valeur de chacun des quadruples mots d'un paquet contenu dans des opérandes.

Algorithme

SI taille de l'opérande = 64 bits **ALORS**
  destination(0..63) ← *destination*(0..63) - *source*(0..63)
SINON
  destination(0..63) ← *destination*(0..63) - *source*(0..63)
  destination(64..127) ← *destination*(64..127) - *source*(64..127)
FIN SI

Mnémonique

Instruction	Opcode	Description
PSUBQ *mm1*, *mm2/m64*	0Fh FBh /r	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des quadruples mots d'un paquet contenu dans des opérandes.
PSUBQ *xmm1*, *xmm2/m128*	66h 0Fh FBh /r	Cette instruction permet d'effectuer la soustraction de la valeur de chacun

		des quadruples mots d'un paquet contenu dans des opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 294 à 296.

Assembleur 80x86 — PSUBSB

INTEL Pentium Pro+ — *Packed Subtract with Saturation Bytes*

Syntaxe

PSUBSB *destination*, *source*

Description

Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des octets signés d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

destination(7..0) ← SaturateToSignedByte(*destination*(7..0) - *source*(7..0))
destination(15..8) ← SaturateToSignedByte(*destination*(15..8) - *source*(15..8))
destination(23..16) ← SaturateToSignedByte(*destination*(23..16) - *source*(23..16))
destination(31..24) ← SaturateToSignedByte(*destination*(31..24) - *source*(31..24))
destination(39..32) ← SaturateToSignedByte(*destination*(39..32) - *source*(39..32))
destination(47..40) ← SaturateToSignedByte(*destination*(47..40) - *source*(47..40))
destination(55..48) ← SaturateToSignedByte(*destination*(55..48) - *source*(55..48))
destination(63..56) ← SaturateToSignedByte(*destination*(63..56) - *source*(63..56))

Mnémonique

Instruction	Opcode	Description
PSUBSB *mm,mm/m64*	0Fh E8h /r	Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des octets signés d'un paquet contenu dans des opérandes de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 297 à 300.

Assembleur 80x86 — PSUBSIW

Cyrix 6x86MX (EMMX)+

Packed Subtract with Saturation, using Implied Destination

Syntaxe

PSUBSIW *dest, source*

Description

Cette instruction permet d'effectuer la soustraction du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans le registre *MMX*.

Algorithme

mmi(15..0) ← SaturateToSignedWord(*dest*(15..0) - *source*(15..0))
mmi(31..16) ← SaturateToSignedWord(*dest*(31..16) - *source*(31..16))
mmi(47..32) ← SaturateToSignedWord(*dest*(47..32) - *source*(47..32))
mmi(63..48) ← SaturateToSignedWord(*dest*(63..48) - *source*(63..48))

Mnémonique

Instruction	Opcode	Description
PSUBSIW *mm,mm/m64*	0Fh 55h /r	Cette instruction permet d'effectuer la soustraction du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans le registre *MMX*.

Assembleur 80x86 — PSUBSW

INTEL Pentium Pro+ — *Packed Subtract with Saturation*

Syntaxe

PSUBSW *dest*, *source*

Description

Cette instruction permet d'effectuer la soustraction du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans l'opérande destinataire.

Algorithme

dest(15..0) ← SaturateToSignedWord(*dest*(15..0) - *source*(15..0))
dest(31..16) ← SaturateToSignedWord(*dest*(31..16) - *source*(31..16))
dest(47..32) ← SaturateToSignedWord(*dest*(47..32) - *source*(47..32))
dest(63..48) ← SaturateToSignedWord(*dest*(63..48) - *source*(63..48))

Mnémonique

Instruction	Opcode	Description
PSUBSW *mm,mm/m64*	0Fh E9h /r	Cette instruction permet d'effectuer la soustraction du mot de format entier de l'opérande source au mot de format entier de l'opérande de destination et écrit le résultat dans l'opérande destinataire.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 297 à 300.

Assembleur 80x86

PSUBUSB

INTEL Pentium Pro+

Packed Subtract Unsigned with Saturation Bytes

Syntaxe

PSUBUSB *destination*, *source*

Description

Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des octets d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

destination(7..0) ← SaturateToUnsignedByte(*destination*(7..0) - *source*(7..0))
destination(15..8) ← SaturateToUnsignedByte(*destination*(15..8) - *source*(15..8))
destination(23..16) ← SaturateToUnsignedByte(*destination*(23..16) - *source*(23..16))
destination(31..24) ← SaturateToUnsignedByte(*destination*(31..24) - *source*(31..24))
destination(39..32) ← SaturateToUnsignedByte(*destination*(39..32) - *source*(39..32))
destination(47..40) ← SaturateToUnsignedByte(*destination*(47..40) - *source*(47..40))
destination(55..48) ← SaturateToUnsignedByte(*destination*(55..48) - *source*(55..48))
destination(63..56) ← SaturateToUnsignedByte(*destination*(63..56) - *source*(63..56))

Mnémonique

Instruction	Opcode	Description
PSUBUSB *mm,mm/m64*	0Fh D8h /r	Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des octets d'un paquet contenu dans des opérandes de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 301 à 305.

Assembleur 80x86 — PSUBUSW

INTEL Pentium Pro+ — *Packed Subtract Unsigned with Saturation Word*

Syntaxe

PSUBUSW *dest*, *source*

Description

Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des mots d'un paquet contenu dans des opérandes de destinations.

Algorithme

dest(15..0) ← SaturateToUnsignedWord(*dest*(15..0) - *source*(15..0))
dest(31..16) ← SaturateToUnsignedWord(*dest*(31..16) - *source*(31..16))
dest(47..32) ← SaturateToUnsignedWord(*dest*(47..32) - *source*(47..32))
dest(63..48) ← SaturateToUnsignedWord(*dest*(63..48) - *source*(63..48))

Mnémonique

Instruction	Opcode	Description
PSUBUSW *mm,mm/m64*	0Fh D9h /r	Cette instruction permet d'effectuer la soustraction avec saturation de la valeur de chacun des mots d'un paquet contenu dans des opérandes de destinations.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 301 à 305.

Assembleur 80x86 — PSUBW

INTEL Pentium Pro+ — *Packed Subtract Words*

Syntaxe

PSUBW *dest*, *source*

Description

Cette instruction permet d'effectuer la soustraction de la valeur de chacun des mots d'un paquet contenu dans des opérandes de 64 bits.

Algorithme

dest(15..0) ← *dest*(15..0) - *source*(15..0)
dest(31..16) ← *dest*(31..16) - *source*(31..16)
dest(47..32) ← *dest*(47..32) - *source*(47..32)
dest(63..48) ← *dest*(63..48) - *source*(63..48)

Mnémonique

Instruction	Opcode	Description
PSUBW *mm,mm/m64*	0Fh F9h /r	Cette instruction permet d'effectuer la soustraction de la valeur de chacun des mots d'un paquet contenu dans des opérandes de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 290 à 293.

Assembleur 80x86 — PSWAPD

3DNow! — *Packed Swap Doubleword*

Syntaxe

PSWAPD *destination, source*

Description

Cette instruction permet d'effectuer l'échange de la partie basse de 32 bits avec la partie haute de 32 bits dans des opérandes source de 64 bits et met le résultat dans une opérande 64 bits destinataire.

Algorithme

destination(63..32) ← *source*(31..0)
destination(31..0) ← *source*(63..32)

Mnémonique

Instruction	Opcode	Description
PSWAPD *mm,mm/m64*	0Fh 0Fh BBh /r	Cette instruction permet d'effectuer l'échange de la partie basse de 32 bits avec la partie haute de 32 bits dans des opérandes source de 64 bits et met le résultat dans une opérande 64 bits destinataire.

Assembleur 80x86 PTEST

SSE4.1+ *Packed Test*

Syntaxe

PTEST *destination*, *source*

Description

Cette instruction permet d'effectuer une comparaison de paquet avec un «*Et binaire*» sans modifier les opérandes.

Algorithme

```
SI source(127..0) ∩ destination(127..0) = 0 ALORS
  ZF ← 1
SINON
  ZF ← 0
FIN SI
SI source(127..0) ∩ ¬ destination(127..0) = 0 ALORS
  CF ← 1
SINON
  CF ← 0
FIN SI
AF ← 0
OF ← 0
PF ← 0
SF ← 0
```

Mnémonique

Instruction	Opcode	Description

PTEST *xmm1,xmm2/m128*	66h 0Fh 38h 17h /r	Cette instruction permet d'effectuer une comparaison de paquet avec un «*Et binaire*» sans modifier les opérandes.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 306 à 308.

Assembleur 80x86 — PUNPCKHBW

INTEL Pentium Pro+ — *Unpack High Bytes to Words*

Syntaxe

PUNPCKHBW *destination*, *source*

Description

Cette instruction permet de décompacter des octets en mots dans le haut d'un paquets de 64 bits.

Algorithme

destination(63..56) ← *source*(63..56)
destination(55..48) ← *destination*(63..56)
destination(47..40) ← *source*(55..48)
destination(39..32) ← *destination*(55..48)
destination(31..24) ← *source*(47..40)
destination(23..16) ← *destination*(47..40)
destination(15..8) ← *source*(39..32)
*destination*7..0) ← *destination*(39..32)

Mnémonique

Instruction	Opcode	Description
PUNPCKHBW *mm,mm/m64*	0Fh 68h /r	Cette instruction permet de décompacter des octets en mots dans le haut d'un paquets de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 309 à 314.

Assembleur 80x86 — PUNPCKHDQ

INTEL Pentium Pro+ — *Unpack High Dwords to Qwords*

Syntaxe

PUNPCKHDQ *destination*, *source*

Description

Cette instruction permet de décompacter des doubles mots en quadruples mots dans le haut d'un paquets de 64 bits.

Algorithme

destination(63..32) ← *source*(63..32)
destination(31..0) ← *destination*(63..32)

Mnémonique

Instruction	Opcode	Description
PUNPCKHDQ *mm,mm/m64*	0Fh 6Ah /r	Cette instruction permet de décompacter des doubles mots en quadruples mots dans le haut d'un paquets de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 309 à 314.

Assembleur 80x86 PUNPCKHQDQ

INTEL Pentium 4 (SSE2)+ *Unpack High Data Quadword*

Syntaxe

PUNPCKHQDQ *destination, source*

Description

Cette instruction permet de décompacter des doubles mots en quadruples mots dans le haut d'un paquets de 128 bits.

Algorithme

destination(0..63) ← *destination*(64..127) *destination*(64..127) ← *source*(64..127)

Mnémonique

Instruction	Opcode	Description
PUNPCKHQDQ *xmm1, xmm2/m128*	66h 0Fh 6Dh /r	Cette instruction permet de décompacter des doubles mots en quadruples mots dans le haut d'un paquets de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 309 à 314.

Assembleur 80x86 — PUNPCKHWD

INTEL Pentium Pro+ — *Unpack High Words to Dwords*

Syntaxe

PUNPCKHWD *destination*, *source*

Description

Cette instruction permet de décompacter des mots en double mots dans le haut d'un paquets de 64 bits.

Algorithme

destination(63..56) ← *source*(63..48)
destination(47..32) ← *destination*(63..48)
destination(31..16) ← *source*(47..32)
destination(15..0) ← *destination*(47..32)

Mnémonique

Instruction	Opcode	Description
PUNPCKHWD *mm,mm/m64*	0Fh 69h /r	Cette instruction permet de décompacter des mots en double mots dans le haut d'un paquet de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 309 à 314.

Assembleur 80x86 — PUNPCKLBW

INTEL Pentium Pro+ — *Unpack Low Bytes to Words*

Syntaxe

PUNPCKLBW *destination*, *source*

Description

Cette instruction permet de décompacter des octets en mots dans le bas d'un paquets de 64 bits.

Algorithme

destination(63..56) ← *source*(31..24)
destination(55..48) ← *destination*(31..24)
destination(47..40) ← *source*(23..16)
destination(39..32) ← *destination*(23..16)
destination(31..24) ← *source*(15..8)
destination(23..16) ← *destination*(15..8)
destination(15..8) ← *source*(7..0)
destination(7..0) ← *destination*(7..0)

Mnémonique

Instruction	Opcode	Description
PUNPCKLBW *mm,mm/m32*	0Fh 60h /r	Cette instruction permet de décompacter des octets en mots dans le bas d'un paquets de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 315 à 320.

Assembleur 80x86 — PUNPCKLDQ

INTEL Pentium Pro+ — *Unpack Low Dwords to Qwords*

Syntaxe

PUNPCKLDQ *destination, source*

Description

Cette instruction permet de décompacter des doubles mots en quadruples mots dans le bas d'un paquets de 64 bits.

Algorithme

destination(63..32) ← *source*(31..0)
destination(31..0) ← *destination*(31..0)

Mnémonique

Instruction	Opcode	Description
PUNPCKLDQ *mm,mm/m32*	0Fh 62h /r	Cette instruction permet de décompacter des doubles mots en quadruples mots dans le bas d'un paquets de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 315 à 320.

Assembleur 80x86 PUNPCKLQDQ

INTEL Pentium 4 (SSE2)+ *Unpack Low Data Quadword*

Syntaxe

PUNPCKLQDQ *destination, source*

Description

Cette instruction permet de décompacter des doubles mots en quadruples mots dans le bas d'un paquets de 128 bits.

Algorithme

destination(0..63) ← *destination*(0..63) *destination*(64..127) ← *source*(0..63)

Mnémonique

Instruction	Opcode	Description
PUNPCKLQDQ *xmm1, xmm2/m128*	66h 0Fh 6Ch /r	Cette instruction permet de décompacter des doubles mots en quadruples mots dans le bas d'un paquets de 128 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 315 à 320.

Assembleur 80x86 — PUNPCKLWD

INTEL Pentium Pro+ — *Unpack Low Words to Dwords*

Syntaxe

PUNPCKLWD *destination*, *source*

Description

Cette instruction permet de décompacter des mots en double mots dans le bas d'un paquets de 64 bits.

Algorithme

destination(63..48) ← *source*(31..16)
destination(47..32) ← *destination*(31..16)
destination(31..16) ← *source*(15..0)
destination(15..0) ← *destination*(15..0)

Mnémonique

Instruction	Opcode	Description
PUNPCKLWD *mm,mm/m32*	0Fh 61h /r	Cette instruction permet de décompacter des mots en double mots dans le bas d'un paquets de 64 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 315 à 320.

Assembleur 80x86	**PUSH**
INTEL 8088+	*Push Value Onto Stack*

Syntaxe

PUSH *opérande*

Description

Cette instruction permet d'empiler une mot ou un double mot dans la pile.

Algorithme

```
SI taille de l'adresse = 32 bits ALORS
  SI taille de l'opérande = 32 bits ALORS
    ESP ← ESP - 4
    SS:ESP ← opérande
  SINON
    ESP ← ESP - 2
    SS:ESP ← opérande
  FIN SI
SINON
  SI taille de l'opérande = 16 bits ALORS
    SP ← SP - 2
    SS:SP ← opérande
  SINON
    SP ← SP - 4
    SS:SP ← opérande
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
PUSH *reg/mem16*	FFh /6	Empile le contenu d'une opérande registre ou mémoire 16 bits dans la pile.
PUSH *reg/mem32*	FFh /6	Empile le contenu d'une opérande registre ou mémoire 32 bits dans la pile. Il n'y a pas de préfixe en mode 64 bits.
PUSH *reg/mem64*	FFh /6	Empile le contenu d'une opérande registre ou mémoire 64 bits dans la pile.
PUSH *reg16*	50h +rw	Empile le contenu d'un registre 16 bits dans la pile.
PUSH *reg32*	50h +rd	Empile le contenu d'un registre 32 bits dans la pile. Il n'y a pas de préfixe en mode 64 bits.
PUSH *reg64*	50h +rq	Empile le contenu d'un registre 64 bits dans la pile.
PUSH *imm8*	6Ah ib	Empile le contenu d'une valeur immédiate 8 bits dans la pile.
PUSH *imm16*	68h iw	Empile le contenu d'une valeur immédiate 16 bits dans la pile.
PUSH *imm32*	68h id	Empile le contenu d'une valeur immédiate 32 bits dans la pile. Il n'y a pas de préfixe en mode 64 bits.
PUSH *imm64*	68h id	Empile le contenu d'une valeur immédiate 32 bits dans la pile. Il n'y a pas de préfixe en mode 64 bits.

PUSH CS	0Eh	Empile le contenu du registre CS dans la pile. Invalide en mode 64 bits.
PUSH DS	1Eh	Empile le contenu du registre DS dans la pile. Invalide en mode 64 bits.
PUSH ES	06h	Empile le contenu du registre ES dans la pile. Invalide en mode 64 bits.
PUSH FS	0Fh A0h	Empile le contenu du registre FS dans la pile.
PUSH GS	0Fh A8h	Empile le contenu du registre GS dans la pile.
PUSH SS	16h	Empile le contenu du registre SS dans la pile. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Les instructions «PUSH CS», «PUSH DS», «PUSH ES» ou «PUSH SS» sont exécutés en mode 64 bits.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est

				pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction POP

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 826
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 413
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 197.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 321 à 325.

Assembleur 80x86	**PUSHA**
INTEL 80286+	*Push All General Registers*

Syntaxe

```
PUSHA
```

Description

Cette instruction permet d'empiler respectivement les registres DI, SI, BP, SP, BX, DX, CX et AX dans la pile.

Algorithme

```
temp ← ESP
(E)SP ← (E)SP - 2
SS:(E)SP ← AX
(E)SP ← (E)SP - 2
SS:(E)SP ← CX
(E)SP ← (E)SP - 2
SS:(E)SP ← DX
(E)SP ← (E)SP - 2
SS:(E)SP ← BX
(E)SP ← (E)SP - 2
SS:(E)SP ← temp
(E)SP ← (E)SP - 2
SS:(E)SP ← BP
(E)SP ← (E)SP - 2
SS:(E)SP ← SI
(E)SP ← (E)SP - 2
SS:(E)SP ← DI
```

Mnémonique

Instruction	Opcode	Description
PUSHA	60h	Empile le contenu des registres AX, CX, DX, BX, SP avant l'appel de cette instruction, BP, SI et DI dans la pile. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction est exécuté en mode 64 bits.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification

				d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction POPA
Instruction assembleur 80x86 - Instruction POPAD

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 827
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 199.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 326 à 327.

Assembleur 80x86	**PUSHAD**
INTEL 80386+	*Push All General Registers Double*

Syntaxe

```
PUSHAD
```

Description

Cette instruction permet d'empiler respectivement les registres EDI, ESI, EBP, ESP, EBX, EDX, ECX et EAX dans la pile.

Algorithme

```
temp ← ESP
(E)SP ← (E)SP - 2
SS:(E)SP ← EAX
(E)SP ← (E)SP - 2
SS:(E)SP ← ECX
(E)SP ← (E)SP - 2
SS:(E)SP ← EDX
(E)SP ← (E)SP - 2
SS:(E)SP ← EBX
(E)SP ← (E)SP - 2
SS:(E)SP ← temp
(E)SP ← (E)SP - 2
SS:(E)SP ← EBP
(E)SP ← (E)SP - 2
SS:(E)SP ← ESI
(E)SP ← (E)SP - 2
SS:(E)SP ← EDI
```

Mnémonique

Instruction	Opcode	Description
PUSHAD	60h	Empile le contenu des registres EAX, ECX, EDX, EBX, ESP avant l'appel de cette instruction, EBP, ESI et EDI dans la pile. Invalide en mode 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#UD(Opcode invalide)			X	Cette instruction est exécuté en mode 64 bits.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification

				d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction POPA
Instruction assembleur 80x86 - Instruction POPAD

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 827
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 199.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 326 à 327.

Assembleur 80x86	**PUSHF**
INTEL 8088+	*Push Flags*

Syntaxe

```
PUSHF
```

Description

Cette instruction permet d'empiler respectivement le registre 16 bits de drapeau des indicateurs d'état dans la pile.

Algorithme

```
(E)SP ← (E)SP - 2
SS:(E)SP ← Registre de drapeaux (FLAGS)
```

Mnémonique

Instruction	Opcode	Description
PUSHF	9Ch	Empile le mot du registre de drapeaux 16 bits (FLAGS) dans la pile.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du

				segment de pile ou n'est pas canonique
#GP(Protection générale)		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et le VME n'est pas actif ou la taille de l'opérande n'est pas 16 bits.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction POPF
Instruction assembleur 80x86 - Instruction POPFD
Instruction assembleur 80x86 - Instruction POPFQ

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 827

Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 413
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 200.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 328 à 330.

Assembleur 80x86	PUSHFD
INTEL 80386+	*Push Flags Double*

Syntaxe

```
PUSHFD
```

Description

Cette instruction permet d'empiler respectivement le registre 32 bits de drapeau des indicateurs d'état dans la pile.

Algorithme

```
(E)SP ← (E)SP - 4
SS:(E)SP ← Registre de drapeaux (EFLAGS)
```

Mnémonique

Instruction	Opcode	Description
PUSHFD	9Ch	Empile le double mot du registre de drapeaux 32 bits (EFLAGS) dans la pile.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de pile ou n'est pas canonique
#GP(Protection générale)		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et le VME n'est pas actif ou la taille de l'opérande n'est pas 16 bits.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction POPF
Instruction assembleur 80x86 - Instruction POPFD
Instruction assembleur 80x86 - Instruction POPFQ

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 328 à 330.

Assembleur 80x86 PUSHFQ

x86-64+

Push rFLAGS onto Stack

Syntaxe

```
PUSHFQ
```

Description

Cette instruction permet d'empiler respectivement le registre 64 bits de drapeau (RFLAGS) des indicateurs d'état dans la pile.

Algorithme

```
(E)SP ← (E)SP - 8
SS:(E)SP ← Registre de drapeaux (EFLAGS)
```

Mnémonique

Instruction	Opcode	Description
PUSHFQ	9Ch	Empile le quadruple mot du registre de drapeaux 64 bits (RFLAGS) dans la pile.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire

				dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection générale)		X		Le niveau de prévilège d'entrée/sortie est inférieur à 3 et le VME n'est pas actif ou la taille de l'opérande n'est pas 16 bits.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction POPF
Instruction assembleur 80x86 - Instruction POPFD
Instruction assembleur 80x86 - Instruction POPFQ

Assembleur 80x86 — PXOR

INTEL Pentium MMX+ — *Bitwise Logical Exclusive OR*

Syntaxe

PXOR *destination, source*

Description

Cette instruction permet d'effectuer un ou exclusif binaire d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.

Algorithme

destination ← *destination* XOR *source*

Mnémonique

Instruction	Opcode	Description
PXOR *mm,mm/m64*	0Fh EFh /r	Cette instruction permet d'effectuer un ou exclusif binaire d'un quadruple mot d'une opérande source avec une opérande destination dans le cas des registres *XMM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 331 à 333.

Assembleur 80x86	**RCL**
INTEL 8088+	*Rotate Through Carry Left*

Syntaxe

RCL *opérandecible*,*nombredebits*

Description

Cette instruction permet d'effectuer une rotation des bits vers la gauche en réinsérant le bit dans l'indicateur de retenue (*CF*) ainsi que dans le bit le plus à droite libéré.

Algorithme

```
temp ← max (nombredebits, 31)
SI temp = 1 ALORS
  OF ← highbit(opérandecible) <> CF
SINON
  OF ← ?
FIN SI
Valeur ← concaténation ( CF, opérandecible )
FAIRE TANT QUE temp <> 0
  X ← highbit ( valeur )
  valeur ← ( valeur décalage à gauche 1 ) + X
  temp ← temp - 1
FIN DE FAIRE
CF ← highbit ( valeur )
opérandecible ← valeur
```

Mnémonique

Instruction	Opcode	Description

RCL *reg/mem8*,1	D0h /2	Cette instruction effectue une rotation de 9 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 8 bits) d'un seul bit vers la gauche.
RCL *reg/mem8*, CL	D2h /2	Cette instruction effectue une rotation de 9 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 8 bits) avec le nombre de bits spécifié par le registre CL vers la gauche.
RCL *reg/mem8*, *imm8*	C0h /2 ib	Cette instruction effectue une rotation de 9 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 8 bits) avec le nombre de bits spécifié par la valeur immédiate vers la gauche.
RCL *reg/mem16*, 1	D1h /2	Cette instruction effectue une rotation de 17 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 16 bits) d'un seul bit vers la gauche.
RCL *reg/mem16*, CL	D3h /2	Cette instruction effectue une rotation de 17 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 16 bits) avec le nombre de bits spécifié par le registre CL vers la gauche.
RCL *reg/mem16*, *imm8*	C1h /2 ib	Cette instruction effectue une rotation de 17 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 16 bits) avec le nombre de bits spécifié par la valeur immédiate vers la gauche.

RCL *reg/mem32*, 1	D1h /2	Cette instruction effectue une rotation de 33 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 32 bits) d'un seul bit vers la gauche.
RCL *reg/mem32*, CL	D3h /2	Cette instruction effectue une rotation de 33 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 32 bits) avec le nombre de bits spécifié par le registre CL vers la gauche.
RCL *reg/mem32*, *imm8*	C1h /2 ib	Cette instruction effectue une rotation de 33 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 32 bits) avec le nombre de bits spécifié par la valeur immédiate vers la gauche.
RCL *reg/mem64*, 1	D1h /2	Cette instruction effectue une rotation de 65 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 64 bits) d'un seul bit vers la gauche.
RCL *reg/mem64*, CL	D3h /2	Cette instruction effectue une rotation de 65 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 64 bits) avec le nombre de bits spécifié par le registre CL vers la gauche.
RCL *reg/mem64*, *imm8*	C1h /2 ib	Cette instruction effectue une rotation de 65 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 64 bits) avec le nombre de bits spécifié par la valeur immédiate vers la gauche.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction RCR
Instruction assembleur 80x86 - Instruction ROL
Instruction assembleur 80x86 - Instruction ROR

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 827
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 413
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 202.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 334 à 341.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

RCPPS

Reciprocals of Packed Single-Precision

Syntaxe

RCPPS *destination*, *source*

Description

Cette instruction permet d'effectuer le calcul de la réciproque d'un paquet de valeurs de réel de simple précision.

Algorithme

destination(31..0) ← APPROX(1.0 / *source*(31..0)) *destination*(63..32) ← APPROX(1.0 / *source*(63..32)) *destination*(95..64) ← APPROX(1.0 / *source*(95..64)) *destination*(127..96) ← APPROX(1.0 / *source*(127..96))

Mnémonique

Instruction	Opcode	Description
RCPPS *xmm1,xmm2/m128*	0Fh 53h /r	Cette instruction permet d'effectuer le calcul de la réciproque d'un paquet de valeurs de réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 342 à 345.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

RCPSS

Reciprocals of Packed Scalar Single-Precision

Syntaxe

RCPSS *destination*, *source*

Description

Cette instruction permet d'effectuer le calcul de la réciproque d'une valeur de réel de simple précision.

Algorithme

destination(31..0) ← APPROX(1.0 / *source*(31..0))

Mnémonique

Instruction	Opcode	Description
RCPSS *xmm1,xmm2/m32*	F3h 0Fh 53h /r	Cette instruction permet d'effectuer le calcul de la réciproque d'une valeur de réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 346 à 348.

Assembleur 80x86	**RCR**
INTEL 8088+	*Rotate Through Carry Right*

Syntaxe

RCR *opérandecible*,*nombredebits*

Description

Cette instruction permet d'effectuer une rotation des bits vers la droite en réinsérant le bit dans l'indicateur de retenue (*CF*) ainsi que dans le bit le plus à gauche libéré.

Algorithme

```
temp ← max (nombredebits, 31)
SI temp = 1 ALORS
  OF ← highbit ( opérandecible ) <> highbit ( opérandecible ) décalage à droite ) )
SINON
  OF ← ?
FIN SI
Valeur ← concaténation ( opérandecible, CF )
FAIRE TANT QUE temp <> 0
  X ← valeur ∩ 1
  valeur ← valeur décalage à droite de 1
  highbit ( valeur ) ← x
  temp ← temp - 1
FIN DE FAIRE
CF ← highbit ( valeur )
opérandecible ← valeur
```

Mnémonique

Instruction	**Opcode**	**Description**

RCR *reg/mem8*, 1	D0h /3	Cette instruction effectue une rotation de 9 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 8 bits) d'un seul bit vers la droite.
RCR *reg/mem8*,CL	D2h /3	Cette instruction effectue une rotation de 9 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 8 bits) avec le nombre de bits spécifié par le registre CL vers la droite.
RCR *reg/mem8*,*imm8*	C0h /3 ib	Cette instruction effectue une rotation de 9 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 8 bits) avec le nombre de bits spécifié par la valeur immédiate vers la droite.
RCR *reg/mem16*,1	D1h /3	Cette instruction effectue une rotation de 17 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 16 bits) d'un seul bit vers la droite.
RCR *reg/mem16*,CL	D3h /3	Cette instruction effectue une rotation de 17 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 16 bits) avec le nombre de bits spécifié par le registre CL vers la droite.
RCR *reg/mem16*, *imm8*	C1h /3 ib	Cette instruction effectue une rotation de 17 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 16 bits) avec le nombre de bits spécifié par la valeur immédiate vers la droite.

RCR *reg/mem32*,1	D1h /3	Cette instruction effectue une rotation de 33 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 32 bits) d'un seul bit vers la droite.
RCR *reg/mem32*,CL	D3h /3	Cette instruction effectue une rotation de 33 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 32 bits) avec le nombre de bits spécifié par le registre CL vers la droite.
RCR *reg/mem32*, *imm8*	C1h /3 ib	Cette instruction effectue une rotation de 33 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 32 bits) avec le nombre de bits spécifié par la valeur immédiate vers la droite.
RCR *reg/mem64*,1	D1h /3	Cette instruction effectue une rotation de 65 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 64 bits) d'un seul bit vers la droite.
RCR *reg/mem64*,CL	D3h /3	Cette instruction effectue une rotation de 65 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 64 bits) d'un seul bit vers la droite.
RCR *reg/mem64*, *imm8*	C1h /3 ib	Cette instruction effectue une rotation de 65 bits (drapeau de retenue 1 bit + emplacement registre ou mémoire 64 bits) avec le nombre de bits spécifié par la valeur immédiate vers la droite.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction RCL
Instruction assembleur 80x86 - Instruction ROR
Instruction assembleur 80x86 - Instruction ROL

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 828
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 413
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 204.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 334 à 341.

Assembleur 80x86 RDMSR

INTEL Pentium+

Read from Model Specific Register

Syntaxe

```
RDMSR
```

Description

Cette instruction permet de charger le contenu du modèle de registre 64-bits (*MSR*) indiquer par le registre ECX dans le couple de registre *EDX*:*EAX*.

Algorithme

```
EDX:EAX ← MSR(ECX)
```

Mnémonique

Instruction	Opcode	Description
RDMSR	0Fh 32h	Cette instruction permet de charger le contenu du modèle de registre 64-bits (*MSR*) indiquer par le registre ECX dans le couple de registre *EDX*:*EAX*.

Voir également

Instruction assembleur 80x86 - Instruction WRMSR
Instruction assembleur 80x86 - Instruction RDTSC
Instruction assembleur 80x86 - Instruction RDPMC

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 349 à 350.

Assembleur 80x86 — RDPMC

INTEL Pentium Pro+ — *Read Performance-Monitoring Counters*

Syntaxe

```
RDPMC
```

Description

Cette instruction permet d'effectuer la lecture du compteur du moniteur de performance.

Algorithme

```
SI (ECX = 0 OU ECX = 1) ET ((CR4.PCE = 1) OU ((CR4.PCE = 0) ET (CPL = 0))) ALORS
  EDX:EAX ← PMC(ECX)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
RDPMC	0Fh 33h	Cette instruction permet d'effectuer la lecture du compteur du moniteur de performance.

Voir également

Instruction assembleur 80x86 - Instruction RDMSR
Instruction assembleur 80x86 - Instruction WRMSR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 351 à 356.

Assembleur 80x86 — RDSHR

Cyrix CX6x86MX+ — *Read SMM Header Pointer Register*

Syntaxe

RDSHR *destination*

Description

Cette instruction permet d'effectuer la lecture du registre de pointeur vers l'entête *SMM*.

Algorithme

destination ← SMHR

Mnémonique

Instruction	Opcode	Description
RDMSR	0Fh 36h /r	Cette instruction permet d'effectuer la lecture du registre de pointeur vers l'entête *SMM*.

Assembleur 80x86 — RDTSC

INTEL Pentium+ — *Read Time-Stamp Counter*

Syntaxe

```
RDTSC
```

Description

Cette instruction permet de charger la valeur courante du compteur de temps du microprocesseur dans le couple de registres *EDX*:*EAX*.

Algorithme

```
SI ((CR4.TSD = 0) OU (CR4.TDS = 1)) ET (CPL = 0) ALORS
  EDX:EAX ← Compteur de temps
SINON
  EXCEPTION #GP(0) ← Compteur de temps
FIN SI
```

Mnémonique

Instruction	Opcode	Description
RDTSC	0Fh 31h	Cette instruction permet de charger la valeur courante du compteur de temps du microprocesseur dans le couple de registres *EDX*:*EAX*.

Voir également

Instruction assembleur 80x86 - Instruction RDTSCP
Instruction assembleur 80x86 - Instruction RDMSR
Instruction assembleur 80x86 - Instruction WRMSR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 357 à 358.

Assembleur 80x86 RDTSCP

x86-64+

Read Time-Stamp Counter and Processor ID

Syntaxe

```
RDTSCP
```

Description

Cette instruction permet de charger la valeur courante du compteur de temps du microprocesseur dans le couple de registres *EDX*:*EAX* et la valeur du *TSC_AUX* dans le registre *ECX*.

Mnémonique

Instruction	Opcode	Description
RDTSCP	0Fh 01h F9h	Cette instruction permet de charger la valeur courante du compteur de temps du microprocesseur dans le couple de registres *EDX*:*EAX* et la valeur du *TSC_AUX* dans le registre *ECX*.

Voir également

Instruction assembleur 80x86 - Instruction RDTSC

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 359 à 360.

Assembleur 80x86	**REP**
INTEL 8088+	*Repeat String Prefix*

Syntaxe

REP *chaine_d'instruction_paramètre*

Description

Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions tant que CX ne vaut pas 0.

Algorithme

```
SI OpCode est [ INS, INSB, INSD, INSW, LODS, LODSB, LODSD, LODSQ, LODSW, MOVS, MOVSB,
MOVSD, MOVSQ, MOVSW, OUTS, OUTSB, OUTSD, OUTSW, STOS, STOSB, STOSD, STOSQ, STOSW
] ALORS
  FAIRE TANT QUE CX <> 0
    Exécute instruction_paramètre
    CX ← CX - 1
  FIN DE FAIRE
FIN SI
SI OpCode est [ CMPS, CMPSB, CMPSD, CMPSQ, CMPSW, SCAS, SCASB, SCASD, SCASW ] ALORS
  FAIRE TANT QUE CX <> 0
    Exécute instruction_paramètre
    CX ← CX - 1
  SI ZF = 0 ALORS Fin de boucle
  FIN DE FAIRE
FIN SI
```

Mnémonique

Instruction	Opcode	Description

REP *autreinstruction*	F3h *autreinstruction*	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions tant que CX ne vaut pas 0.

Références

Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 413
Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 828
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 361 à 365.

INTEL 8088+

Syntaxe

REPE *chaine_d'instruction_paramètre*

Description

Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 0.

Algorithme

```
SI OpCode est [ INS, INSB, INSD, INSW, LODS, LODSB, LODSD, LODSQ, LODSW, MOVS, MOVSB,
MOVSD, MOVSQ, MOVSW, OUTS, OUTSB, OUTSD, OUTSW, STOS, STOSB, STOSD, STOSQ, STOSW
] ALORS
  FAIRE TANT QUE CX <> 0 OU ZF <> 0
    Exécute instruction_paramètre
    CX ← CX - 1
  FIN DE FAIRE
FIN SI
SI OpCode est [ CMPS, CMPSB, CMPSD, CMPSQ, CMPSW, SCAS, SCASB, SCASD, SCASW ] ALORS
  FAIRE TANT QUE CX <> 0 OU ZF <> 0
    Exécute instruction_paramètre
    CX ← CX - 1
  SI ZF = 0 ALORS Fin de boucle
  FIN DE FAIRE
FIN SI
```

Mnémonique

Instruction	Opcode	Description

REPE *autreinstruction*	F3h *autreinstruction*	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 0.

Références

Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 414
Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 828
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 361 à 365.

INTEL 8088+

Syntaxe

REPNE *chaine_d'instruction_paramètre*

Description

Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 1.

Algorithme

```
SI OpCode est [ INS, INSB, INSD, INSW, LODS, LODSB, LODSD, LODSQ, LODSW, MOVS, MOVSB,
MOVSD, MOVSQ, MOVSW, OUTS, OUTSB, OUTSD, OUTSW, STOS, STOSB, STOSD, STOSQ, STOSW
] ALORS
  FAIRE TANT QUE CX <> 0 OU ZF <> 1
    Exécute instruction_paramètre
    CX ← CX - 1
  FIN DE FAIRE
FIN SI
SI OpCode est [ CMPS, CMPSB, CMPSD, CMPSQ, CMPSW, SCAS, SCASB, SCASD, SCASW ] ALORS
  FAIRE TANT QUE CX <> 0 OU ZF <> 1
    Exécute instruction_paramètre
    CX ← CX - 1
  SI ZF = 0 ALORS Fin de boucle
  FIN DE FAIRE
FIN SI
```

Mnémonique

Instruction	Opcode	Description

REPNE *autreinstruction*	F2h *autreinstruction*	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 1.

Références

Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 414
Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 829
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 361 à 365.

INTEL 8088+

Syntaxe

REPNZ *chaine_d'instruction_paramètre*

Description

Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 1.

Algorithme

```
SI OpCode est [ INS, INSB, INSD, INSW, LODS, LODSB, LODSD, LODSQ, LODSW, MOVS, MOVSB,
MOVSD, MOVSQ, MOVSW, OUTS, OUTSB, OUTSD, OUTSW, STOS, STOSB, STOSD, STOSQ, STOSW
] ALORS
  FAIRE TANT QUE CX <> 0 OU ZF <> 1
    Exécute instruction_paramètre
    CX ← CX - 1
  FIN DE FAIRE
FIN SI
SI OpCode est [ CMPS, CMPSB, CMPSD, CMPSQ, CMPSW, SCAS, SCASB, SCASD, SCASW ] ALORS
  FAIRE TANT QUE CX <> 0 OU ZF <> 1
    Exécute instruction_paramètre
    CX ← CX - 1
  SI ZF = 0 ALORS Fin de boucle
  FIN DE FAIRE
FIN SI
```

Mnémonique

Instruction	Opcode	Description

REPNZ *autreinstruction*	F2h *autreinstruction*	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 1.

Références

Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 414
Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 829
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 361 à 365.

INTEL 8088+

Syntaxe

REPZ *chaine_d'instruction_paramètre*

Description

Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 0.

Algorithme

```
SI OpCode est [ INS, INSB, INSD, INSW, LODS, LODSB, LODSD, LODSQ, LODSW, MOVS, MOVSB,
MOVSD, MOVSQ, MOVSW, OUTS, OUTSB, OUTSD, OUTSW, STOS, STOSB, STOSD, STOSQ, STOSW
] ALORS
  FAIRE TANT QUE CX <> 0 OU ZF <> 0
    Exécute instruction_paramètre
    CX ← CX - 1
  FIN DE FAIRE
FIN SI
SI OpCode est [ CMPS, CMPSB, CMPSD, CMPSQ, CMPSW, SCAS, SCASB, SCASD, SCASW ] ALORS
  FAIRE TANT QUE CX <> 0 OU ZF <> 0
    Exécute instruction_paramètre
    CX ← CX - 1
  SI ZF = 0 ALORS Fin de boucle
  FIN DE FAIRE
FIN SI
```

Mnémonique

Instruction	Opcode	Description

REPZ *autreinstruction*	F3h *autreinstruction*	Cette instruction est utilisé comme préfixe avec d'autres instructions pour effectuer des répétitions d'instructions jusqu'à ce que CX = 0 ou tant que l'indicateur ZF = 0.

Références

Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 414
Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 829
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 361 à 365.

Assembleur 80x86

AMD Am386SXLV, Am386DXLV

RES3

Restore All CPU Registers 386

Syntaxe

RES3

Description

Cette instruction permet d'effectuer le chargement de tous les registres de descripteur de cache.

Algorithme

```
(ES:EDI+0000h)(4) ← CR0
(ES:EDI+0004h)(4) ← EFLAGS
(ES:EDI+0008h)(4) ← EIP
(ES:EDI+000Ch)(4) ← EDI
(ES:EDI+0010h)(4) ← ESI
(ES:EDI+0014h)(4) ← EBP
(ES:EDI+0018h)(4) ← ESP
(ES:EDI+001Ch)(4) ← EBX
(ES:EDI+0020h)(4) ← EDX
(ES:EDI+0024h)(4) ← ESX
(ES:EDI+0028h)(4) ← EAX
(ES:EDI+002Ch)(4) ← DR6
(ES:EDI+0030h)(4) ← DR7
(ES:EDI+0034h)(4) ← TR (16 bits complété par des 0)
(ES:EDI+0038h)(4) ← LDT
(ES:EDI+003Ch)(4) ← GS (16 bits complété par des 0)
(ES:EDI+0040h)(4) ← FS (16 bits complété par des 0)
(ES:EDI+0044h)(4) ← DS (16 bits complété par des 0)
(ES:EDI+0048h)(4) ← SS (16 bits complété par des 0)
(ES:EDI+004Ch)(4) ← CS (16 bits complété par des 0)
(ES:EDI+0050h)(4) ← ES (16 bits complété par des 0)
(ES:EDI+0054h)(4) ← Attribut de TSS
(ES:EDI+0058h)(4) ← Base de TSS
(ES:EDI+005Ch)(4) ← Limite de TSS
(ES:EDI+0060h)(4) ← 0s
```

(ES:EDI+0064h)(4) ← Base de IDT
(ES:EDI+0068h)(4) ← Limite de IDT
(ES:EDI+006Ch)(4) ← 0s
(ES:EDI+0070h)(4) ← Base de GDT
(ES:EDI+0074h)(4) ← Limite de GDT
(ES:EDI+0078h)(4) ← Attribut de LDT
(ES:EDI+007Ch)(4) ← Base de LDT
(ES:EDI+0080h)(4) ← Limite de LDT
(ES:EDI+0084h)(4) ← Attribut de GS
(ES:EDI+0088h)(4) ← Base de GS
(ES:EDI+008Ch)(4) ← Limite de GS
(ES:EDI+0090h)(4) ← Attribut de FS
(ES:EDI+0094h)(4) ← Base de FS
(ES:EDI+0098h)(4) ← Limite de FS
(ES:EDI+009Ch)(4) ← Attribut de DS
(ES:EDI+00A0h)(4) ← Base de DS
(ES:EDI+00A4h)(4) ← Limite de DS
(ES:EDI+00A8h)(4) ← Attribut de SS
(ES:EDI+00ACh)(4) ← Base de SS
(ES:EDI+00B0h)(4) ← Limite de SS
(ES:EDI+00B4h)(4) ← Attribut de CS
(ES:EDI+00B8h)(4) ← Base de CS
(ES:EDI+00BCh)(4) ← Limite de CS
(ES:EDI+00C0h)(4) ← Attribut de ES
(ES:EDI+00C4h)(4) ← Base de ES
(ES:EDI+00C8h)(4) ← Limite de ES
(ES:EDI+00CCh)(34h) ← Non utilisé
(ES:EDI+0100h)(4) ← Registre temporaire
(ES:EDI+0104h)(4) ← Registre temporaire
(ES:EDI+0108h)(4) ← Registre temporaire
(ES:EDI+010Ch)(4) ← Registre temporaire
(ES:EDI+0110h)(4) ← Registre temporaire
(ES:EDI+0114h)(4) ← Registre temporaire
(ES:EDI+0118h)(4) ← Registre temporaire
(ES:EDI+011Ch)(4) ← Registre temporaire
(ES:EDI+0120h)(4) ← Registre temporaire
(ES:EDI+0124h)(4) ← Dernier instruction EIP pour le redémarrage

Mnémonique

Instruction	Opcode	Description
RES3	0Fh 07h	Cette instruction permet d'effectuer la lecture des données à l'adresse ES:EDI.

Assembleur 80x86

AMD Am386SXLV, Am386DXLV

RES4

Restore All CPU Registers 486

Syntaxe

RES4

Description

Cette instruction permet d'effectuer le chargement de tous les registres de descripteur de cache.

Algorithme

```
(ES:EDI+0000h)(4) ← CR0
(ES:EDI+0004h)(4) ← EFLAGS
(ES:EDI+0008h)(4) ← EIP
(ES:EDI+000Ch)(4) ← EDI
(ES:EDI+0010h)(4) ← ESI
(ES:EDI+0014h)(4) ← EBP
(ES:EDI+0018h)(4) ← ESP
(ES:EDI+001Ch)(4) ← EBX
(ES:EDI+0020h)(4) ← EDX
(ES:EDI+0024h)(4) ← ESX
(ES:EDI+0028h)(4) ← EAX
(ES:EDI+002Ch)(4) ← DR6
(ES:EDI+0030h)(4) ← DR7
(ES:EDI+0034h)(4) ← TR (16 bits complété par des 0)
(ES:EDI+0038h)(4) ← LDT
(ES:EDI+003Ch)(4) ← GS (16 bits complété par des 0)
(ES:EDI+0040h)(4) ← FS (16 bits complété par des 0)
(ES:EDI+0044h)(4) ← DS (16 bits complété par des 0)
(ES:EDI+0048h)(4) ← SS (16 bits complété par des 0)
(ES:EDI+004Ch)(4) ← CS (16 bits complété par des 0)
(ES:EDI+0050h)(4) ← ES (16 bits complété par des 0)
(ES:EDI+0054h)(4) ← Attribut de TSS
(ES:EDI+0058h)(4) ← Base de TSS
(ES:EDI+005Ch)(4) ← Limite de TSS
(ES:EDI+0060h)(4) ← 0s
```

(ES:EDI+0064h)(4) ← Base de IDT
(ES:EDI+0068h)(4) ← Limite de IDT
(ES:EDI+006Ch)(4) ← 0s
(ES:EDI+0070h)(4) ← Base de GDT
(ES:EDI+0074h)(4) ← Limite de GDT
(ES:EDI+0078h)(4) ← Attribut de LDT
(ES:EDI+007Ch)(4) ← Base de LDT
(ES:EDI+0080h)(4) ← Limite de LDT
(ES:EDI+0084h)(4) ← Attribut de GS
(ES:EDI+0088h)(4) ← Base de GS
(ES:EDI+008Ch)(4) ← Limite de GS
(ES:EDI+0090h)(4) ← Attribut de FS
(ES:EDI+0094h)(4) ← Base de FS
(ES:EDI+0098h)(4) ← Limite de FS
(ES:EDI+009Ch)(4) ← Attribut de DS
(ES:EDI+00A0h)(4) ← Base de DS
(ES:EDI+00A4h)(4) ← Limite de DS
(ES:EDI+00A8h)(4) ← Attribut de SS
(ES:EDI+00ACh)(4) ← Base de SS
(ES:EDI+00B0h)(4) ← Limite de SS
(ES:EDI+00B4h)(4) ← Attribut de CS
(ES:EDI+00B8h)(4) ← Base de CS
(ES:EDI+00BCh)(4) ← Limite de CS
(ES:EDI+00C0h)(4) ← Attribut de ES
(ES:EDI+00C4h)(4) ← Base de ES
(ES:EDI+00C8h)(4) ← Limite de ES
(ES:EDI+00CCh)(34h) ← Non utilisé
(ES:EDI+0100h)(4) ← Registre temporaire
(ES:EDI+0104h)(4) ← Registre temporaire
(ES:EDI+0108h)(4) ← Registre temporaire
(ES:EDI+010Ch)(4) ← Registre temporaire
(ES:EDI+0110h)(4) ← Registre temporaire
(ES:EDI+0114h)(4) ← Registre temporaire
(ES:EDI+0118h)(4) ← Registre temporaire
(ES:EDI+011Ch)(4) ← Registre temporaire
(ES:EDI+0120h)(4) ← Registre temporaire
(ES:EDI+0124h)(4) ← Dernier instruction EIP pour le redémarrage
(ES:EDI+0128h)(4) ← PEIP: Pointeur d'instruction sur l'espace SRAM précédent
(ES:EDI+012Eh)(36) ← Non utilisé
(ES:EDI+0150h)(4) ← Registres de pointeur interne de nombre réel (virgule flottante)

Mnémonique

Instruction	Opcode	Description

RES4	0Fh 07h	Cette instruction permet d'effectuer la lecture des données à l'adresse ES:EDI.

Assembleur 80x86	**RET**
INTEL 8088+	*Return*

Syntaxe

RET
RET *immédiat*

Description

Cette instruction permet de quitter une procédure. Ainsi, elle indique au microprocesseur qu'il doit désempiler la valeur du pointeur d'instructions contenu dans la pile et la copier dans les registres CS:IP pour une procédure de type *FAR* ou dans le registre IP pour une procédure de type *NEAR*. Ensuite, l'exécution du code se poursuit à l'instruction suivant l'instruction «*CALL*».

Algorithme

```
SI instruction = RETN ALORS
  SI opérande de taille = 32 bits ALORS
    SI les 12 octets du haut de la pile sont en dehors de la limite de la pile ALORS
      EXCEPTION #SS(0)
    FIN SI
    EIP ← Pop()
  SINON
    SI les 6 octets du haut de la pile sont en dehors de la limite de la pile ALORS
      EXCEPTION #SS(0)
    FIN SI
    tempEIP ← Pop()
    tempEIP ← tempEIP ∩ 0000FFFFh
    SI tempEIP n'est pas dans limite du segment de code ALORS
      EXCEPTION #GP(0)
    FIN SI
    EIP ← tempEIP
  FIN SI
```

```
  SI instruction a une opérande immédiate ALORS
    SI taille de l'adresse de la pile = 32 bits ALORS
      ESP ← ESP + SRC
    SINON
      SP ← SP + SRC
    FIN SI
  FIN SI
FIN SI

SI ((PE = 0) OU (PE = 1 ET VM = 1)) ET instruction RETF ALORS
  SI taille de l'opérande = 32 bits ALORS
    SI les 12 octets du haut de la pile sont en dehors de la limite de la pile ALORS
      EXCEPTION #SS(0)
    FIN SI
    EIP ← Pop()
    CS ← Pop()
  SINON
    SI les 6 octets du haut de la pile sont en dehors de la limite de la pile ALORS
      EXCEPTION #SS(0)
    FIN SI
    tempEIP ← Pop()
    tempEIP ← tempEIP ∩ 0000FFFFh
    SI tempEIP n'est pas dans limite du segment de code ALORS
      EXCEPTION #GP(0)
    FIN SI
    EIP ← tempEIP
    CS ← Pop()
  FIN SI
  SI instruction a une opérande immédiate ALORS
    SP ← SP + (SRC ∩ FFFFh)
  FIN SI
FIN SI

SI (PE = 1 ET VM = 0) ET instruction RETF ALORS
  SI taille de l'opérande = 32 bits ALORS
    SI le deuxième double mot de la pile est en dehors de la limite de la pile ALORS
      EXCEPTION #SS(0)
    FIN SI
  FIN SI
  SI sélecteur de code de retour est nulle ALORS
    EXCEPTION GP(0)
```

```
  FIN SI
  SI adresse du sélecteur de retour de code segment est en dehors des limites de la table de descripteur ALORS
    EXCEPTION GP(selecteur)
  FIN SI
  Demande le descripteur avec le point du sélecteur de segment de code de retour du descripteur de table
  SI descripteur de retour de code segment n'est pas un code segment ALORS
    EXCEPTION #GP(selecteur)
  FIN SI
  SI selecteur de retour de segment de code RPL < CPL ALORS
    EXCEPTION #GP(Sélecteur)
  FIN SI
  SI descripteur retour de code de segment retour est conforme ET retour de segment de code DPL > sélecteur de retour de segment de code RPL ALORS
    EXCEPTION #GP(Sélecteur)
  FIN SI
  SI descripteur de retour segment de code n'est pas présent ALORS
    EXCEPTION #NP(Sélecteur)
  FIN SI
  SI sélecteur de retour de segment de code RPL > CPL ALORS
    ALLER A RETURN-OUTER-PRIVILEGE-LEVEL
  SINON
    ALLER A RETURN-TO-SAME-PRIVILEGE-LEVEL
  FIN SI
FIN SI

RETURN-SAME-PRIVILEGE-LEVEL:
  SI l'instruction de pointeur n'est pas à l'intérieur de la limite de segment de code ALORS
    EXCEPTION #GP(0)
  FIN SI
  SI taille de l'opérande = 32 bits ALORS
    EIP ← Pop()
    CS ← Pop()
    ESP ← ESP + SRC
  SINON
    EIP ← Pop()
    EIP ← EIP ∩ 0000FFFFh
    CS ← Pop()
    ESP ← ESP + SRC
  FIN SI
```

RETURN-OUTER-PRIVILEGE-LEVEL:
  SI (16 + SRC) octets du haut de la pile sont en dehors de la limite de la pile ET taille de l'opérande = 32 bits **OU** (8 + SRC) octets du haut de la pile sont en dehors de la limite de la pile ET taille de l'opérande = 16 **ALORS**
    EXCEPTION #SS(0)
  FIN SI
  Lecture du sélecteur de retour de segment
  SI sélecteur de segment de pile n'est pas nulle **ALORS**
    EXCEPTION #GP(0)
  FIN SI
  SI index de retour du sélecteur de segment n'est pas dans les limites du descripteur de table **ALORS**
    EXCEPTION #GP(selector)
  FIN SI
  Lecture du descripteur de segment pointant sur le retour de sélecteur de segment
  SI sélecteur de segment de pile RPL <> RPL du retour de sélecteur de segment de code **OU** segment de pile n'est pas dans une segment de données écrivable **OU** descripteur de segment de pile DPL <> RPL de retour de sélecteur de segment de code **ALORS**
    EXCEPTION #GP(Séelecteur)
  FIN SI
  SI segment de pile n'est pas présent **ALORS**
    EXCEPTION #SS(Sélecteur de segment de pile)
  FIN SI
  SI pointeur de retourn d'instruction n'est pas dans la limite de segment de code **ALORS**
    EXCEPTION #GP(0)
  FIN SI
  CPL ← Retour du sélecteur de segment de code du RPL
  SI taille de l'opérande = 32 bits **ALORS**
    EIP ← Pop()
    CS ← Pop()
    CS(RPL) ← CPL
    ESP ← ESP + SRC
    tempESP ← Pop()
    tempSS ← Pop()
    ESP ← tempESP
    SS ← tempSS
  SINON
    EIP ← Pop()
    EIP ← EIP ∩ 0000FFFFh
    CS ← Pop()

```
    CS(RPL) ← CPL
    ESP ← ESP + SRC
    tempESP ← Pop()
    tempSS ← Pop()
    ESP ← tempESP
    SS ← tempSS
  FIN SI
  BOUCLE POUR chaque registre de segment (ES, FS, GS et DS) FAIRE
    SI registre de segment pointe sur des données ou segment de code est non-conforme ET CPL
> descripteur de segment DPL ALORS
      Sélecteur de segment ← 0
    FIN SI
  FIN BOUCLE POUR
  BOUCLE POUR chaque registre de segment ES, FS, GS et DS FAIRE
    SI index de sélecteur de segment est en dehors de la limite du descripteur de table OU
descripteur de segment indique que le segment n'est pas dans des données OU lecteur de
segment de code si le segment de données OU segment de code non-conforme ET descripteur
de segment DPL < CPL OU RPL du sélecteur de segment de code ALORS
      Sélecteur de egistre de segment ← Sélecteur nulle
    FIN SI
  FIN BOUCLE POUR
  ESP ← ESP + SRC
```

Mnémonique

Instruction	Opcode	Description
RET	C3h	Retour d'un appel de procédure court.
RET *imm16*	C2h iw	Retour d'un appel de procédure court et spécifie le nombre d'octets à retirer de la pile.
RET	CBh	Retour d'un appel de procédure long.
RET *imm16*	CAh iw	Retour d'un appel de procédure long et spécifie le nombre d'octets à

		retirer de la pile.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
#NP(Sélecteur)			X	Le retour de segment de code est marqué non présent.
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur non-nulle et le segment est marqué non présent.

#GP(Sélecteur)			X	Le sélecteur de code de retour est un sélecteur nulle.
			X	Le sélecteur de retour de pile et de mode de retour n'est pas un mode 64 bits ou CPL vaut 3.
			X	Le code de retour ou le descripteur de pile dépasse la limite du descripteur de table.
			X	Le code de retour ou le bit TI du sélecteur de pile est fixé mais le sélecteur LDT n'est pas un sélecteur nulle.
			X	Le descripteur de segment pour le code retour n'est pas un segment de

				code.
			X	Le *RPL* du sélecteur du retour de segment de code est inférieur au *CPL*.
			X	Le retour du segment de code n'est pas conforme et le sélecteur de segment *DPL* n'est pas égale au *RPL* du sélecteur de segment de code.
			X	Le retour du segment de code est conforme et le sélecteur de segment *DPL* est supérieur au *RPL* du sélecteur de segment de code.
			X	Le descripteur de segment pour le retour de pile n'est pas dans segment de

				données écrivable.
			X	Le descripteur de segment de pile *DPL* n'est pas égale au RPL du sélecteur de retour de segment de code.
			X	Le sélecteur de segment de pile *RPL* n'est pas égale au *RPL* du retour de sélecteur de segment de code.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 830
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 414
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 207.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 366 à 376.

Assembleur 80x86	**RETF**
INTEL 8088+	*Return Far*

Syntaxe

RETF
RETF *immédiat*

Description

Cette instruction permet de quitter une procédure ayant lieu avec un appel long (FAR CALL). Ainsi, elle indique au microprocesseur qu'il doit désempiler la valeur du pointeur d'instructions contenu dans la pile et la copier dans les registres CS:IP. Ensuite, l'exécution du code se poursuit à l'instruction «*CALL*».

Algorithme

```
SI ((PE = 0) OU (PE = 1 ET VM = 1)) ALORS
  SI taille de l'opérande = 32 bits ALORS
    SI les 12 octets du haut de la pile sont en dehors de la limite de la pile ALORS
      EXCEPTION #SS(0)
    FIN SI
    EIP ← Pop()
    CS ← Pop()
  SINON
    SI les 6 octets du haut de la pile sont en dehors de la limite de la pile ALORS
      EXCEPTION #SS(0)
    FIN SI
    tempEIP ← Pop()
    tempEIP ← tempEIP ∩ 0000FFFFh
    SI tempEIP n'est pas dans limite du segment de code ALORS
      EXCEPTION #GP(0)
    FIN SI
    EIP ← tempEIP
    CS ← Pop()
```

```
  FIN SI
  SI instruction a une opérande immédiate ALORS
    SP ← SP + (SRC ∩ FFFFh)
  FIN SI
FIN SI

SI (PE = 1 ET VM = 0) ALORS
  SI taille de l'opérande = 32 bits ALORS
    SI le deuxième double mot de la pile est en dehors de la limite de la pile ALORS
      EXCEPTION #SS(0)
    FIN SI
  FIN SI
  SI sélecteur de code de retour est nulle ALORS
    EXCEPTION GP(0)
  FIN SI
  SI adresse du sélecteur de retour de code segment est en dehors des limites de la table de
descripteur ALORS
    EXCEPTION GP(selecteur)
  FIN SI
  Demande le descripteur avec le point du sélecteur de segment de code de retour du
descripteur de table
  SI descripteur de retour de code segment n'est pas un code segment ALORS
    EXCEPTION #GP(selecteur)
  FIN SI
  SI selecteur de retour de segment de code RPL < CPL ALORS
    EXCEPTION #GP(Sélecteur)
  FIN SI
  SI descripteur retour de code de segment retour est conforme ET retour de segment de code
DPL > sélecteur de retour de segment de code RPL ALORS
    EXCEPTION #GP(Sélecteur)
  FIN SI
  SI descripteur de retour segment de code n'est pas présent ALORS
    EXCEPTION #NP(Sélecteur)
  FIN SI
  SI sélecteur de retour de segment de code RPL > CPL ALORS
    ALLER A RETURN-OUTER-PRIVILEGE-LEVEL
  SINON
    ALLER A RETURN-TO-SAME-PRIVILEGE-LEVEL
  FIN SI
FIN SI
```

RETURN-SAME-PRIVILEGE-LEVEL:
  SI l'instruction de pointeur n'est pas à l'intérieur de la limite de segment de code **ALORS**
    EXCEPTION #GP(0)
  FIN SI
  SI taille de l'opérande = 32 bits **ALORS**
    EIP ← Pop()
    CS ← Pop()
    ESP ← ESP + SRC
  SINON
    EIP ← Pop()
    EIP ← EIP ⋂ 0000FFFFh
    CS ← Pop()
    ESP ← ESP + SRC
  FIN SI

RETURN-OUTER-PRIVILEGE-LEVEL:
  SI (16 + SRC) octets du haut de la pile sont en dehors de la limite de la pile ET taille de l'opérande = 32 bits **OU** (8 + SRC) octets du haut de la pile sont en dehors de la limite de la pile ET taille de l'opérande = 16 **ALORS**
    EXCEPTION #SS(0)
  FIN SI
  Lecture du sélecteur de retour de segment
  SI sélecteur de segment de pile n'est pas nulle **ALORS**
    EXCEPTION #GP(0)
  FIN SI
  SI index de retour du sélecteur de segment n'est pas dans les limites du descripteur de table **ALORS**
    EXCEPTION #GP(selector)
  FIN SI
  Lecture du descripteur de segment pointant sur le retour de sélecteur de segment
  SI sélecteur de segment de pile RPL <> RPL du retour de sélecteur de segment de code **OU** segment de pile n'est pas dans une segment de données écrivable **OU** descripteur de segment de pile DPL <> RPL de retour de sélecteur de segment de code **ALORS**
    EXCEPTION #GP(Séelecteur)
  FIN SI
  SI segment de pile n'est pas présent **ALORS**
    EXCEPTION #SS(Sélecteur de segment de pile)
  FIN SI
  SI pointeur de retourn d'instruction n'est pas dans la limite de segment de code **ALORS**
    EXCEPTION #GP(0)
  FIN SI

```
CPL ← Retour du sélecteur de segment de code du RPL
SI taille de l'opérande = 32 bits ALORS
  EIP ← Pop()
  CS ← Pop()
  CS(RPL) ← CPL
  ESP ← ESP + SRC
  tempESP ← Pop()
  tempSS ← Pop()
  ESP ← tempESP
  SS ← tempSS
SINON
  EIP ← Pop()
  EIP ← EIP ∩ 0000FFFFh
  CS ← Pop()
  CS(RPL) ← CPL
  ESP ← ESP + SRC
  tempESP ← Pop()
  tempSS ← Pop()
  ESP ← tempESP
  SS ← tempSS
FIN SI
BOUCLE POUR chaque registre de segment (ES, FS, GS et DS) FAIRE
  SI registre de segment pointe sur des données ou segment de code est non-conforme ET CPL
> descripteur de segment DPL ALORS
    Sélecteur de segment ← 0
  FIN SI
FIN BOUCLE POUR
BOUCLE POUR chaque registre de segment ES, FS, GS et DS FAIRE
  SI index de sélecteur de segment est en dehors de la limite du descripteur de table OU
descripteur de segment indique que le segment n'est pas dans des données OU lecteur de
segment de code si le segment de données OU segment de code non-conforme ET descripteur
de segment DPL < CPL OU RPL du sélecteur de segment de code ALORS
    Sélecteur de egistre de segment ← Sélecteur nulle
  FIN SI
FIN BOUCLE POUR
ESP ← ESP + SRC
```

Mnémonique

Instruction	Opcode	Description
RETF	CBh	Retour d'un appel de procédure long.
RETF *imm16*	CAh iw	Retour d'un appel de procédure long et spécifie le nombre d'octets à retirer de la pile.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#NP(Sélecteur)			X	Le retour de segment de code est marqué non présent.
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#SS(Sélecteur)	X	X	X	Le registre SS est chargé avec un sélecteur non-nulle et le segment est marqué non présent.
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de données ou n'est pas canonique
#GP(Sélecteur)			X	Le sélecteur de code de retour est un sélecteur nulle.
			X	Le sélecteur de retour de pile et de mode de retour n'est pas un mode 64 bits ou CPL vaut 3.
			X	Le code de retour ou le descripteur de pile dépasse la limite du descripteur de table.
			X	Le code de retour ou le bit TI du sélecteur de pile est fixé mais le sélecteur LDT n'est pas un sélecteur nulle.

			X	Le descripteur de segment pour le code retour n'est pas un segment de code.
			X	Le *RPL* du sélecteur du retour de segment de code est inférieur au *CPL*.
			X	Le retour du segment de code n'est pas conforme et le sélecteur de segment *DPL* n'est pas égale au *RPL* du sélecteur de segment de code.
			X	Le retour du segment de code est conforme et le sélecteur de segment *DPL* est supérieur au *RPL* du sélecteur de segment de code.

			X	Le descripteur de segment pour le retour de pile n'est pas dans segment de données écrivable.
			X	Le descripteur de segment de pile *DPL* n'est pas égale au RPL du sélecteur de retour de segment de code.
			X	Le sélecteur de segment de pile *RPL* n'est pas égale au *RPL* du retour de sélecteur de segment de code.
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est

				effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CALL
Instruction assembleur 80x86 - Instruction RET

Assembleur 80x86	**RETN**
INTEL 8088+	*Return Near*

Syntaxe

RETN
RETN *immédiat*

Description

Cette instruction permet de quitter une procédure. Ainsi, elle indique au microprocesseur qu'il doit désempiler la valeur du pointeur d'instructions contenu dans la pile et la copie dans le registre IP. Ensuite, l'exécution du code se poursuit à l'instruction «*CALL*».

Algorithme

```
SI opérande de taille = 32 bits ALORS
  SI les 12 octets du haut de la pile sont en dehors de la limite de la pile ALORS
    EXCEPTION #SS(0)
  FIN SI
  EIP ← Pop()
SINON
  SI les 6 octets du haut de la pile sont en dehors de la limite de la pile ALORS
    EXCEPTION #SS(0)
  FIN SI
  tempEIP ← Pop()
  tempEIP ← tempEIP ∩ 0000FFFFh
  SI tempEIP n'est pas dans limite du segment de code ALORS
    EXCEPTION #GP(0)
  FIN SI
  EIP ← tempEIP
FIN SI
SI instruction a une opérande immédiate ALORS
  SI taille de l'adresse de la pile = 32 bits ALORS
```

```
    ESP ← ESP + SRC
  SINON
    SP ← SP + SRC
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
RETN	C3h	Retour d'un appel de procédure court.
RETN *imm16*	C2h iw	Retour d'un appel de procédure court et spécifie le nombre d'octets à retirer de la pile.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas

				canonique
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CALL
Instruction assembleur 80x86 - Instruction RET

Assembleur 80x86	**ROL**
INTEL 8088+	*Rotate Left*

Syntaxe

ROL *opérandecible,nombredebits*

Description

Cette instruction permet d'effectuer une rotation des bits vers la gauche en réinsérant le bit dans le bit le plus à droite libéré.

Algorithme

```
temp ← max ( compteur, 31)
SI temp = 1 ALORS
  OF ← highbit ( destination ) <> CF
SINON
  OF ← ?
FIN SI
BOUCLE FAIRE TANT QUE temp <> 0
  X ← highbit ( destination )
  destination ← (destination décalage à gauche de 1 ) + x
  temp ← temp - 1
FIN BOUCLE DE TANT QUE
CF ← highbit ( destination )
```

Mnémonique

Instruction	Opcode	Description
ROL *reg/mem8*, 1	D0h /0	Cette instruction effectue une rotation de 8 bits d'un opérande registre ou mémoire 8 bits d'un seul

		bit vers la gauche.
ROL *reg/mem8*, CL	D2h /0	Cette instruction effectue une rotation de 8 bits d'un opérande registre ou mémoire 8 bits avec le nombre de bits spécifié par le registre CL vers la gauche.
ROL *reg/mem8*, *imm8*	C0h /0 ib	Cette instruction effectue une rotation de 8 bits d'un opérande registre ou mémoire 8 bits avec le nombre de bits spécifié par la valeur immédiate vers la gauche.
ROL *reg/mem16*, 1	D1h /0	Cette instruction effectue une rotation de 16 bits d'un opérande registre ou mémoire 16 bits d'un seul bit vers la gauche.
ROL *reg/mem16*, CL	D3h /0	Cette instruction effectue une rotation de 16 bits d'un opérande registre ou mémoire 16 bits avec le nombre de bits spécifié par le registre CL vers la gauche.
ROL *reg/mem16*, *imm8*	C1h /0 ib	Cette instruction effectue une rotation de 16 bits d'un opérande registre ou mémoire 16 bits avec le nombre de bits spécifié par la valeur immédiate vers la gauche.
ROL *reg/mem32*, 1	D1h /0	Cette instruction effectue une rotation de 32 bits d'un opérande registre ou mémoire 32 bits d'un seul bit vers la gauche.
ROL *reg/mem32*, CL	D3h /0	Cette instruction effectue une rotation de 32 bits d'un opérande registre ou mémoire 32 bits avec le nombre de bits spécifié par le registre

		CL vers la gauche.
ROL *reg/mem32*, *imm8*	C1h /0 ib	Cette instruction effectue une rotation de 32 bits d'un opérande registre ou mémoire 32 bits avec le nombre de bits spécifié par la valeur immédiate vers la gauche.
ROL *reg/mem64*, 1	D1h /0	Cette instruction effectue une rotation de 64 bits d'un opérande registre ou mémoire 64 bits d'un seul bit vers la gauche.
ROL *reg/mem64*, CL	D3h /0	Cette instruction effectue une rotation de 64 bits d'un opérande registre ou mémoire 64 bits avec le nombre de bits spécifié par le registre CL vers la gauche.
ROL *reg/mem64*, *imm8*	C1h /0 ib	Cette instruction effectue une rotation de 64 bits d'un opérande registre ou mémoire 64 bits avec le nombre de bits spécifié par la valeur immédiate vers la gauche.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection	X	X	X	Une adresse

général)				mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction RCL
Instruction assembleur 80x86 - Instruction RCR
Instruction assembleur 80x86 - Instruction ROR

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 830*
Assembleur Facile, *Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 415*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 211.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 334 à 341.*

Assembleur 80x86	**ROR**
INTEL 8088+	*Rotate Right*

Syntaxe

ROR *opérandecible*,*nombredebits*

Description

Cette instruction permet d'effectuer une rotation des bits vers la droite en réinsérant le bit dans le bit le plus à gauche libéré.

Algorithme

```
temp ← max ( compteur, 31 )
SI temp = 1 ALORS
  OF ← highbit ( destination ) <> highbit ( destination décalage à droite )
SINON
  OF ← ?
FIN SI
BOUCLE FAIRE TANT QUE temp <> 0
  x ← valeur ∩ 1
  valeur ← valeur décalage à droite de 1
  highbit ( valeur ) ← x
  temp ← temp - 1
BOUCLE FIN DE TANT QUE
CF ← highbit ( valeur )
destination ← valeur
```

Mnémonique

Instruction	Opcode	Description

ROR *reg/mem8*, 1	D0h /1	Cette instruction effectue une rotation de 8 bits d'un opérande registre ou mémoire 8 bits d'un seul bit vers la droite.
ROR *reg/mem8*, CL	D2h /1	Cette instruction effectue une rotation de 8 bits d'un opérande registre ou mémoire 8 bits avec le nombre de bits spécifié par le registre CL vers la droite.
ROR *reg/mem8*, *imm8*	C0h /1 ib	Cette instruction effectue une rotation de 8 bits d'un opérande registre ou mémoire 8 bits avec le nombre de bits spécifié par la valeur immédiate vers la droite.
ROR *reg/mem16*, 1	D1h /1	Cette instruction effectue une rotation de 16 bits d'un opérande registre ou mémoire 16 bits d'un seul bit vers la droite.
ROR *reg/mem16*, CL	D3h /1	Cette instruction effectue une rotation de 16 bits d'un opérande registre ou mémoire 16 bits avec le nombre de bits spécifié par le registre CL vers la droite.
ROR *reg/mem16*, *imm8*	C1h /1 ib	Cette instruction effectue une rotation de 16 bits d'un opérande registre ou mémoire 16 bits avec le nombre de bits spécifié par la valeur immédiate vers la droite.
ROR *reg/mem32*, 1	D1h /1	Cette instruction effectue une rotation de 32 bits d'un opérande registre ou mémoire 32 bits d'un seul bit vers la droite.

ROR *reg/mem32*, CL	D3h /1	Cette instruction effectue une rotation de 32 bits d'un opérande registre ou mémoire 32 bits avec le nombre de bits spécifié par le registre CL vers la droite.
ROR *reg/mem32*, *imm8*	C1h /1 ib	Cette instruction effectue une rotation de 32 bits d'un opérande registre ou mémoire 32 bits avec le nombre de bits spécifié par la valeur immédiate vers la droite.
ROR *reg/mem64*, 1	D1h /1	Cette instruction effectue une rotation de 64 bits d'un opérande registre ou mémoire 64 bits d'un seul bit vers la droite.
ROR *reg/mem64*, CL	D3h /1	Cette instruction effectue une rotation de 64 bits d'un opérande registre ou mémoire 64 bits avec le nombre de bits spécifié par le registre CL vers la droite.
ROR *reg/mem64*, *imm8*	C1h /1 ib	Cette instruction effectue une rotation de 64 bits d'un opérande registre ou mémoire 64 bits avec le nombre de bits spécifié par la valeur immédiate vers la droite.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de

				pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement

				est activé

Voir également

Instruction assembleur 80x86 - Instruction RCL
Instruction assembleur 80x86 - Instruction RCR
Instruction assembleur 80x86 - Instruction ROL

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 830
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 415
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 213.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 334 à 341.

Assembleur 80x86 — ROUNDPD

SSE4.1+

Round Packed Double Precision Floating-Point Values

Syntaxe

ROUNDPD *destination*, *source*, *immediate*

Description

Cette instruction permet d'effectuer l'arrondissement de 2 valeurs réel de double précision dans l'opérande source en utilisant le mode spécifié par l'opérande immédiate et place le résultat dans l'opérande de destination.

Algorithme

```
SI immediate(2) = 1 ALORS
  destination(63..0) ← ConvertDPFPToInteger_M(source(63..0))
  destination(127..64) ← ConvertDPFPToInteger_M(source(127..64))
SINON
  destination(63..0) ← ConvertDPFPToInteger_Imm(source(63..0))
  destination(127..64) ← ConvertDPFPToInteger_Imm(source(127..64))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
ROUNDPD *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 09h /r *ib*	Cette instruction permet d'effectuer l'arrondissement de 2 valeurs réel de double précision dans l'opérande source en utilisant le mode

		spécifié par l'opérande immédiate et place le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 378 à 381.

Assembleur 80x86 — ROUNDPS

SSE4.1+ — *Round Packed Single Precision Floating-Point Values*

Syntaxe

ROUNDPS *destination*, *source*, *immediate*

Description

Cette instruction permet d'effectuer l'arrondissement de 4 valeurs réel de simple précision dans l'opérande source en utilisant le mode spécifié par l'opérande immédiate et place le résultat dans l'opérande de destination.

Algorithme

```
SI immediate(2) = 1 ALORS
  destination(31..0) ← ConvertSPFPToInteger_M(source(31..0))
  destination(63..32) ← ConvertSPFPToInteger_M(source(63..32))
  destination(95..64) ← ConvertSPFPToInteger_M(source(95..64))
  destination(127..96) ← ConvertSPFPToInteger_M(source(127..96))
SINON
  destination(31..0) ← ConvertSPFPToInteger_Imm(source(31..0))
  destination(63..32) ← ConvertSPFPToInteger_Imm(source(63..32))
  destination(95..64) ← ConvertSPFPToInteger_Imm(source(95..64))
  destination(127..96) ← ConvertSPFPToInteger_Imm(source(127..96))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
ROUNDPS *xmm1,xmm2/m128,imm8*	66h 0Fh 3Ah 08h /r *ib*	Cette instruction permet

		d'effectuer l'arrondissement de 4 valeurs réel de simple précision dans l'opérande source en utilisant le mode spécifié par l'opérande immédiate et place le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 382 à 384.

Assembleur 80x86 — ROUNDSD

SSE4.1+

Round Scalar Double Precision Floating-Point Values

Syntaxe

ROUNDSD *destination*, *source*, *immediate*

Description

Cette instruction permet d'effectuer l'arrondissement scalaire de valeurs réel de double précision, dans sa partie basse du quadruple mot, dans l'opérande source en utilisant le mode spécifié par l'opérande immédiate et place le résultat dans l'opérande de destination.

Algorithme

```
SI immediate(2) = 1 ALORS
  destination(63..0) ← ConvertDPFPToInteger_M(source(63..0))
SINON
  destination(63..0) ← ConvertDPFPToInteger_Imm(source(63..0))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
ROUNDSD *xmm1,xmm2/m64*, *imm8*	66h 0Fh 3Ah 0Bh /r *ib*	Cette instruction permet d'effectuer l'arrondissement scalaire de valeurs réel de double précision, dans sa partie basse du quadruple mot, dans l'opérande source en utilisant le mode spécifié par l'opérande

		immédiate et place le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 385 à 386.

Assembleur 80x86 — ROUNDSS

SSE4.1+

Round Scalar Single Precision Floating-Point Values

Syntaxe

ROUNDSS *destination*, *source*, *immediate*

Description

Cette instruction permet d'effectuer l'arrondissement scalaire de valeurs réel de simple précision, dans sa partie basse du quadruple mot, dans l'opérande source en utilisant le mode spécifié par l'opérande immédiate et place le résultat dans l'opérande de destination.

Algorithme

```
SI immediate(2) = 1 ALORS
  destination(31..0) ← ConvertSPFPToInteger_M(source(31..0))
SINON
  destination(31..0) ← ConvertSPFPToInteger_Imm(source(31..0))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
ROUNDSS *xmm1,xmm2/m32,imm8*	66h 0Fh 3Ah 0Ah /r *ib*	Cette instruction permet d'effectuer l'arrondissement scalaire de valeurs réel de simple précision, dans sa partie basse du quadruple mot, dans l'opérande source en utilisant le mode spécifié par l'opérande immédiate

		et place le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 387 à 389.

Assembleur 80x86

Cyrix Cx486S/S2/D/D2/DX/DX2/DX4, IBM BL486DX/DX2, TI 486SLC/DLC/e, TI 486SXL/SXL2/SXLC, TI Potomac

RSDC

Restore Register and Descriptor

Syntaxe

RSDC *destination, source*

Description

Cette instruction permet d'effectuer le restauration de la structure des registres et des descripteurs.

Algorithme

destination[*sélecteur, descripteur*] ← *source*

Mnémonique

Instruction	Opcode	Description
RSDC ES,*mem80*	0Fh 79h (*mm* 000b *mmm*)	Cette instruction permet d'effectuer le restauration de la structure des registres et des descripteurs à partir du registre de segment ES.
RSDC CS,*mem80*	0Fh 79h (*mm* 001b *mmm*)	Cette instruction permet d'effectuer le restauration de la structure des registres et des descripteurs à partir du registre de segment CS.

RSDC SS,*mem80*	0Fh 79h (*mm* 010b *mmm*)	Cette instruction permet d'effectuer le restauration de la structure des registres et des descripteurs à partir du registre de segment SS.
RSDC DS,*mem80*	0Fh 79h (*mm* 011b *mmm*)	Cette instruction permet d'effectuer le restauration de la structure des registres et des descripteurs à partir du registre de segment DS.
RSDC FS,*mem80*	0Fh 79h (*mm* 100b *mmm*)	Cette instruction permet d'effectuer le restauration de la structure des registres et des descripteurs à partir du registre de segment FS.
RSDC GS,*mem80*	0Fh 79h (*mm* 101b *mmm*)	Cette instruction permet d'effectuer le restauration de la structure des registres et des descripteurs à partir du registre de segment GS.

Assembleur 80x86

Cyrix Cx486S/S2/D/D2/DX/DX2/DX4, IBM BL486DX/DX2, TI 486SLC/DLC/e, TI 486SXL/SXL2/SXLC, TI Potomac

RSLDT

Restore LDTR and Descriptor

Syntaxe

RSLDT *source*

Description

Cette instruction permet d'effectuer la restauration des LDTR et des descripteurs.

Algorithme

LDTR[*sélecteur*, *descripteur*] ← *source*

Mnémonique

Instruction	Opcode	Description
RSLDT *mem80*	0Fh 7Bh (mm 000b mmm)	Cette instruction permet d'effectuer la restauration des LDTR et des descripteurs.

Assembleur 80x86 RSM

INTEL 80486+

Resume from System Management

Syntaxe

```
RSM
```

Description

Cette instruction permet de retourner le contrôle du programme du mode de gestion système (*SMM*) pour le programme d'application ou la procédure du système d'exploitation ayant été interrompu lorsque le microprocesseur à recu un signal *SSM*.

Algorithme

```
Retourne du SSM
ProcessorState ← Restaure le SSMDump
```

Mnémonique

Instruction	Opcode	Description
RSM	0Fh AAh	Cette instruction permet de retourner le contrôle du programme du mode de gestion système (*SMM*) pour le programme d'application ou la procédure du système d'exploitation ayant été interrompu lorsque le microprocesseur à recu un signal *SSM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 390 à 391.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

RSQRTPS

Packed Single-Precision FP Square Root Reciprocal

Syntaxe

RSQRTPS *destination*, *source*

Description

Cette instruction permet de calculer la réciproque approximative de la racine carré d'un paquet de simple précision.

Algorithme

destination(31..0) ← APPROX(1.0 / SQRT(*source*(31..0)))
destination(63..32) ← APPROX(1.0 / SQRT(*source*(63..32)))
destination(95..64) ← APPROX(1.0 / SQRT(*source*(95..64)))
destination(127..96) ← APPROX(1.0 / SQRT(*source*(127..96)))

Mnémonique

Instruction	Opcode	Description
RSQRTPS *xmm1,xmm2/m128*	0Fh 52h /r	Cette instruction permet de calculer la réciproque approximative de la racine carré d'un paquet de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 392 à 394.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

RSQRTSS

Scalar Single-Precision FP Square Root Reciprocal

Syntaxe

RSQRTSS *destination, source*

Description

Cette instruction permet de calculer la réciproque approximative de la racine carré d'un scalaire de simple précision.

Algorithme

destination(31..0) ← APPROX(1.0 / SQRT(*source*(31..0)))

Mnémonique

Instruction	Opcode	Description
RSQRTSS *xmm1,xmm2/m128*	F3h 0Fh 52h /r	Cette instruction permet de calculer la réciproque approximative de la racine carré d'un scalaire de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 395 à 397.

Assembleur 80x86

Cyrix Cx486S/S2/D/D2/DX/DX2/DX4, IBM BL486DX/DX2, TI 486SLC/DLC/e, TI 486SXL/SXL2/SXLC, TI Potomac

RSTS

Restore TR and Descriptor

Syntaxe

RSTS *source*

Description

Cette instruction permet d'effectuer le restauration des TR et des descripteurs.

Algorithme

TR[*sélecteur*, *descripteur*] ← *source*

Mnémonique

Instruction	Opcode	Description
RSTS *mem80*	0Fh 7Dh (mm 000b mmm)	Cette instruction permet d'effectuer le restauration des TR et des descripteurs.

Assembleur 80x86	**SAHF**
INTEL 8088+	*Store AH Into Flags*

Syntaxe

```
SAHF
```

Description

Cette instruction permet de copier les bits du registre AH dans l'octet de poids faible dans le registre des drapeaux (les indicateurs d'état).

Algorithme

```
Registre de drapeaux ← Registre de drapeaux ∩ ¬ 0D5h + ( AH ∩ D5h )
```

Mnémonique

Instruction	**Opcode**	**Description**
SAHF	9Eh	Copie le drapeau de signe, de zéro, auxiliaire, de parité et de retenue dans le registre AH.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#UD(Opcode invalide)			X	Cette instruction n'est pas supporté en

				mode 64 bits, comme indiqué par le bit 0 du registre ECX de la fonction 8000_00001h de l'instruction *CPUID*.

Voir également

Instruction assembleur 80x86 - Instruction LAHF

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 831*
Assembleur Facile, *Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 416*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 215.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 398 à 399.*

Assembleur 80x86	**SAL**
INTEL 8088+	*Shift Arithmetic Left*

Syntaxe

SAL *opérandecible,nombredebits*

Description

Cette instruction permet d'effectuer une rotation des bits vers la gauche en réinsérant le bit dans l'indicateur de retenue (*CF*).

Algorithme

```
temp ← compteur ∩ 001Fh
BOUCLE FAIRE TANT QUE temp1 = 0
  CF ← highorder ( destination )
  destination ← destination décalage à gauche de 1
  temp ← temp - 1
BOUCLE FIN DE FAIRE
SI count = 1 ALORS
  OF ← highorder ( destination ) <> CF
SINON
  OF ← ?
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SAL *reg/mem8*, 1	D0h /4	Cette instruction effectue un déclage d'un seul bit vers la gauche d'un opérande registre ou mémoire 8 bits.

SAL *reg/mem8*, CL	D2h /4	Cette instruction effectue un déclage du nombre de bits spécifié par le registre CL vers la gauche d'un opérande registre ou mémoire 8 bits.
SAL *reg/mem8*, *imm8*	C0h /4 ib	Cette instruction effectue un déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 8 bits.
SAL *reg/mem16*, 1	D1h /4	Cette instruction effectue un déclage d'un seul bit vers la gauche d'un opérande registre ou mémoire 16 bits.
SAL *reg/mem16*, CL	D3h /4	Cette instruction effectue un déclage du nombre de bits spécifié par le registre CL vers la gauche d'un opérande registre ou mémoire 16 bits.
SAL *reg/mem16*, *imm8*	C1h /4 ib	Cette instruction effectue un déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 16 bits.
SAL *reg/mem32*, 1	D1h /4	Cette instruction effectue un déclage d'un seul bit vers la gauche d'un opérande registre ou mémoire 32 bits.
SAL *reg/mem32*, CL	D3h /4	Cette instruction effectue un déclage du nombre de bits spécifié par le registre CL vers la gauche d'un opérande registre ou mémoire 32 bits.

SAL *reg/mem32*, *imm8*	C1h /4 ib	Cette instruction effectue un déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 32 bits.
SAL *reg/mem64*, 1	D1h /4	Cette instruction effectue un déclage d'un seul bit vers la gauche d'un opérande registre ou mémoire 64 bits.
SAL *reg/mem64*, CL	D3h /4	Cette instruction effectue un déclage du nombre de bits spécifié par le registre CL vers la gauche de 64 bits d'un opérande registre ou mémoire 64 bits.
SAL *reg/mem64*, *imm8*	C1h /4 ib	Cette instruction effectue un déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la

				limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction SAR

Instruction assembleur 80x86 - Instruction SHR
Instruction assembleur 80x86 - Instruction SHLD
Instruction assembleur 80x86 - Instruction SHRD

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 831
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 415
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 216.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 400 à 408.

Assembleur 80x86	**SAR**
INTEL 8088+	*Shift Arithmetic Right*

Syntaxe

SAR *opérandecible,nombredebits*

Description

Cette instruction permet d'effectuer une rotation des bits vers la droite en réinsérant le bit dans l'indicateur de retenue (*CF*).

Algorithme

```
temp ← compteur ∩ 001Fh
BOUCLE FAIRE TANT temp <> 0
  x ← highorder ( destination )
  CF ← destination ∩ 1
  destination ← destination décalage à droite de 1
  highorder ( destination ) ← x
  temp ← temp - 1
BOUCLE FIN DE FAIRE
SI compteur = 1 ALORS
  OF ← 0
SINON
  OF ← ?
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SAR *reg/mem8*, 1	D0h /7	Cette instruction effectue une déclage d'un seul bit vers la droite de

		8 bits d'un opérande registre ou mémoire 8 bits.
SAR *reg/mem8*, CL	D2h /7	Cette instruction effectue une déclage du nombre de bits spécifié par le registre CL vers la droite de 8 bits d'un opérande registre ou mémoire 8 bits.
SAR *reg/mem8*, *imm8*	C0h /7 ib	Cette instruction effectue une déclage du nombre de bits spécifié par la valeur immédiate vers la droite de 8 bits d'un opérande registre ou mémoire 8 bits.
SAR *reg/mem16*, 1	D1h /7	Cette instruction effectue une déclage d'un seul bit vers la droite de 16 bits d'un opérande registre ou mémoire 16 bits.
SAR *reg/mem16*, CL	D3h /7	Cette instruction effectue une déclage du nombre de bits spécifié par le registre CL vers la droite de 16 bits d'un opérande registre ou mémoire 16 bits.
SAR *reg/mem16*, *imm8*	C1h /7 ib	Cette instruction effectue une déclage du nombre de bits spécifié par la valeur immédiate vers la droite de 16 bits d'un opérande registre ou mémoire 16 bits.
SAR *reg/mem32*, 1	D1h /7	Cette instruction effectue une déclage d'un seul bit vers la droite de 32 bits d'un opérande registre ou mémoire 32 bits.
SAR *reg/mem32*, CL	D3h /7	Cette instruction effectue une déclage du nombre de bits spécifié par le registre CL vers la droite de 32

		bits d'un opérande registre ou mémoire 32 bits.
SAR *reg/mem32*, *imm8*	C1h /7 ib	Cette instruction effectue une déclage du nombre de bits spécifié par la valeur immédiate vers la droite de 32 bits d'un opérande registre ou mémoire 32 bits.
SAR *reg/mem64*, 1	D1h /7	Cette instruction effectue une déclage d'un seul bit vers la droite de 64 bits d'un opérande registre ou mémoire 64 bits.
SAR *reg/mem64*, CL	D3h /7	Cette instruction effectue une déclage du nombre de bits spécifié par le registre CL vers la droite de 64 bits d'un opérande registre ou mémoire 64 bits.
SAR *reg/mem64*, *imm8*	C1h /7 ib	Cette instruction effectue une déclage du nombre de bits spécifié par la valeur immédiate vers la droite de 64 bits d'un opérande registre ou mémoire 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique

#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction SAL
Instruction assembleur 80x86 - Instruction SHL
Instruction assembleur 80x86 - Instruction SHR
Instruction assembleur 80x86 - Instruction SHLD
Instruction assembleur 80x86 - Instruction SHRD

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 831
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 416
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 219.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 400 à 408.

Assembleur 80x86	**SBB**
INTEL 8088+	*Subtraction With Borrow*

Syntaxe

SBB *opérandecible*,*opérandesource*

Description

Cette instruction permet de soustraire avec l'indicateur de retenue (*CF*) une valeur à une opérande.

Algorithme

opérandecible ← *opérandecible* - (*opérandesource* + CF)

Mnémonique

Instruction	**Opcode**	**Description**
SBB AL, *imm8*	1Ch *ib*	Soustrait une valeur immédiate de 8 bits du registre AL avec la retenue.
SBB AX, *imm16*	1Dh *iw*	Soustrait une valeur immédiate de 16 bits du registre AX avec la retenue.
SBB EAX, *imm32*	1Dh *id*	Soustrait une valeur immédiate de 32 bits du registre EAX avec la retenue.
SBB RAX, *imm32*	1Dh *id*	Soustrait une valeur entière immédiate de 32 bits du registre RAX avec la retenue.

SBB *reg/mem8*, *imm8*	80h /3 *ib*	Soustrait une valeur immédiate de 8 bits d'un emplacement registre ou mémoire de 8 bits avec la retenue.
SBB *reg/mem16*, *imm16*	81h /3 *iw*	Soustrait une valeur immédiate de 16 bits d'un emplacement registre ou mémoire de 16 bits avec la retenue.
SBB *reg/mem32*, *imm32*	81h /3 *id*	Soustrait une valeur immédiate de 32 bits d'un emplacement registre ou mémoire de 32 bits avec la retenue.
SBB *reg/mem64*, *imm32*	81h /3 *id*	Soustrait une valeur immédiate de 32 bits d'un emplacement registre ou mémoire de 64 bits avec la retenue.
SBB *reg/mem16*, *imm8*	83h /3 *ib*	Soustrait une valeur immédiate de 8 bits d'un emplacement registre ou mémoire de 16 bits avec la retenue.
SBB *reg/mem32*, *imm8*	83h /3 *ib*	Soustrait une valeur entière immédiate de 8 bits d'un emplacement registre ou mémoire de 32 bits avec la retenue.
SBB *reg/mem64*, *imm8*	83h /3 *ib*	Soustrait une valeur entière immédiate de 8 bits d'un emplacement registre ou mémoire de 64 bits avec la retenue.
SBB *reg/mem8*, *reg8*	18h /r	Soustrait un registre de 8 bits d'un emplacement registre ou mémoire de 8 bits avec la retenue.
SBB *reg/mem16*, *reg16*	19h /r	Soustrait un registre de 16 bits d'un emplacement registre ou mémoire de 16 bits avec la retenue.
SBB *reg/mem32*, *reg32*	19h /r	Soustrait un registre de 32 bits d'un emplacement registre ou mémoire de

		32 bits avec la retenue.
SBB *reg/mem64*, *reg64*	19h /r	Soustrait un registre de 32 bits d'un emplacement registre ou mémoire de 32 bits avec la retenue.
SBB *reg8*, *reg/mem8*	1Ah /r	Soustrait un emplacement registre ou mémoire de 8 bits avec la retenue d'un registre de 8 bits.
SBB *reg16*, *reg/mem16*	1Bh /r	Soustrait un emplacement registre ou mémoire de 16 bits avec la retenue d'un registre de 16 bits.
SBB *reg32*, *reg/mem32*	1Bh /r	Soustrait un emplacement registre ou mémoire de 32 bits avec la retenue d'un registre de 32 bits.
SBB *reg64*, *reg/mem64*	1Bh /r	Soustrait un emplacement registre ou mémoire de 32 bits avec la retenue d'un registre de 32 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du

				segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction SUB
Instruction assembleur 80x86 - Instruction ADD

Instruction assembleur 80x86 - Instruction ADC

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 832*
Assembleur Facile, *Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 416*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 221.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 409 à 412.*

Assembleur 80x86	**SCAS**
INTEL 8088+	*Scan String*

Syntaxe

SCAS *chainecible*

Description

Cette instruction permet de comparer un octet, un mot ou un double mot avec la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI en fonction de la taille de l'opérande cible et de l'état du drapeau de direction.

Algorithme

```
SI chainecible est un octet ALORS
  temp ← AL - chainecible
  SetStatusFlags(temp)
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 1
  SINON
    (E)DI ← (E)DI - 1
  FIN SI
SINON SI chainecible est un mot ALORS
  temp ← AX - chainecible
  SetStatusFlags(temp)
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 2
  SINON
    (E)DI ← (E)DI - 2
  FIN SI
SINON SI chainecible est un double mot ALORS
  temp ← EAX - chainecible
  SetStatusFlags(temp)
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 4
```

```
  SINON
    (E)DI ← (E)DI - 4
  FIN SI
SINON
  temp ← EAX - chainecible
  SetStatusFlags(temp)
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 8
  SINON
    (E)DI ← (E)DI - 8
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SCAS *mem8*	AEh	Compare le contenu d'un registre AL avec l'octet ES:(R)DI et incrémente ou décrémente R(DI).
SCAS *mem16*	AFh	Compare le contenu d'un registre AX avec le mot ES:(R)DI et incrémente ou décrémente R(DI).
SCAS *mem32*	AFh	Compare le contenu d'un registre EAX avec le double mot ES:(R)DI et incrémente ou décrémente R(DI).
SCAS *mem64*	AFh	Compare le contenu d'un registre RAX avec le double mot ES:(R)DI et incrémente ou décrémente R(DI).

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description

#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment ES ou n'est pas canonique
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction CMP
Instruction assembleur 80x86 - Instruction CMPS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 832

AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 223.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 413 à 417.

Assembleur 80x86	**SCASB**
INTEL 8088+	*Scan String Byte*

Syntaxe

```
SCASB
```

Description

Cette instruction permet de comparer le registre AL avec la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 1 en fonction de l'état du drapeau de direction.

Algorithme

```
temp ← AL - chainecible
SetStatusFlags(temp)
SI DF = 0 ALORS
  (E)DI ← (E)DI + 1
SINON
  (E)DI ← (E)DI - 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SCASB	AEh	Compare le contenu d'un registre AL avec l'octet ES:(R)DI et incrémente ou décrémente R(DI).

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment ES ou n'est pas canonique
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 833
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 416
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007,

Publication No. 24594, page 223.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 413 à 417.

Assembleur 80x86	**SCASD**
INTEL 80386+	*Scan String Doubleword*

Syntaxe

```
SCASD
```

Description

Cette instruction permet de comparer le registre EAX avec la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 4 en fonction de l'état du drapeau de direction.

Algorithme

```
temp ← EAX - chainecible
SetStatusFlags(temp)
SI DF = 0 ALORS
  (E)DI ← (E)DI + 4
SINON
  (E)DI ← (E)DI - 4
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SCASD	AFh	Compare le contenu d'un registre EAX avec le double mot ES:(R)DI et incrémente ou décrémente R(DI).

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment ES ou n'est pas canonique
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 833
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 223.

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 413 à 417.

Assembleur 80x86	**SCASW**
INTEL 8088+	*Scan String Word*

Syntaxe

```
SCASW
```

Description

Cette instruction permet de comparer le registre AX avec la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 2 en fonction de l'état du drapeau de direction.

Algorithme

```
temp ← AX - chainecible
SetStatusFlags(temp)
SI DF = 0 ALORS
  (E)DI ← (E)DI + 2
SINON
  (E)DI ← (E)DI - 2
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SCASW	AFh	Compare le contenu d'un registre AX avec le mot ES:(R)DI et incrémente ou décrémente R(DI).

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment ES ou n'est pas canonique
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 834
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 417
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007,

Publication No. 24594, page 223.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 413 à 417.

INTEL 80386+

Syntaxe

SET *condition opérande*

Description

Ces instructions permettent de fixer la valeur d'une opérande 1 si la condition d'indicateur d'état du registre 32 bits de drapeau est vrai sinon fixe la valeur à 0.

Il est a noter que les combinaisons de lettres permettant de représenter la condition sont indiquées dans le tableau suivant:

Instruction	Condition	Description
SETA	A	Supérieur
SETAE	AE	Supérieur ou égal
SETB	B	Inférieur
SETBE	BE	Inférieur ou égal
SETC	C	Indicateur de retenue à égal à 1
SETCXZ	CXZ	Si CX = 0
SETE	E	Zéro (Egal)
SETECXZ	ECXZ	Si ECX = 0
SETG	G	Supérieur

SETGE	GE	Supérieur ou égal
SETL	L	Inférieur
SETLE	LE	Inférieur ou égal
SETNA	NA	Pas supérieur
SETNAE	NAE	Ni inférieur ni égal
SETNB	NB	Pas inférieur
SETNC	NC	Indicateur de retenue égal à 0
SETNE	NE	Pas égal (Zéro)
SETNG	NG	Pas supérieur
SETNGE	NGE	Ni supérieur ni égal
SETNL	NL	Pas inférieur
SETNLE	NLE	Ni inférieur ni égal
SETNO	NO	Pas de débordement (indicateur de débordement égal à 0)
SETNP	NP	Pas de parité
SETNS	NS	Pas de signe
SETNZ	NZ	Pas zéro
SETO	O	Débordement (indicateur de débordement égal à 1)
SETP	P	Parité
SETPE	PE	Parité paire

SETPO	PO	Parité impaire
SETS	S	Signe
SETZ	Z	Zéro

Algorithme

```
SI condition ALORS
  DEST ← 1
SINON
  DEST ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SETO *reg/mem8*	0Fh 90h /0	Fixe l'octet si le drapeau de débordement vaut 1 (OF = 1).
SETNO *reg/mem8*	0Fh 91h /0	Fixe l'octet si le drapeau de débordement vaut 0 (OF = 0).
SETB *reg/mem8*	0Fh 92h /0	Fixe l'octet si le drapeau de retenue vaut 1 (CF = 1).
SETC *reg/mem8*	0Fh 92h /0	Fixe l'octet si le drapeau de retenue vaut 1 (CF = 1).
SETNAE *reg/mem8*	0Fh 92h /0	Fixe l'octet si le drapeau de retenue vaut 1 (CF = 1).
SETNB *reg/mem8*	0Fh 93h /0	Fixe l'octet si le drapeau de retenue vaut 0 (CF = 0).

SETNC *reg/mem8*	0Fh 93h /0	Fixe l'octet si le drapeau de retenue vaut 0 (CF = 0).
SETAE *reg/mem8*	0Fh 93h /0	Fixe l'octet si le drapeau de retenue vaut 0 (CF = 0).
SETZ *reg/mem8*	0Fh 94h /0	Fixe l'octet si le drapeau de zéro vaut 1 (ZF = 1).
SETE *reg/mem8*	0Fh 94h /0	Fixe l'octet si le drapeau de zéro vaut 1 (ZF = 1).
SETNZ *reg/mem8*	0Fh 95h /0	Fixe l'octet si le drapeau de zéro vaut 1 (ZF = 1).
SETNE *reg/mem8*	0Fh 95h /0	Fixe l'octet si le drapeau de zéro vaut 1 (ZF = 1).
SETBE *reg/mem8*	0Fh 96h /0	Fixe l'octet si le drapeau de retenue et zéro vaut 1 (CF = 1 ou ZF = 1).
SETNA *reg/mem8*	0Fh 96h /0	Fixe l'octet si le drapeau de retenue et zéro vaut 1 (CF = 1 ou ZF = 1).
SETNBE *reg/mem8*	0Fh 97h /0	Fixe l'octet si le drapeau de retenue et zéro vaut 0 (CF = 0 ou ZF = 0).
SETA *reg/mem8*	0Fh 97h /0	Fixe l'octet si le drapeau de retenue et zéro vaut 0 (CF = 0 ou ZF = 0).
SETS *reg/mem8*	0Fh 98h /0	Fixe l'octet si le drapeau de signe vaut 1 (SF = 1).
SETNS *reg/mem8*	0Fh 99h /0	Fixe l'octet si le drapeau de signe vaut 0 (SF = 0).
SETP *reg/mem8*	0Fh 9Ah /0	Fixe l'octet si le drapeau de parité vaut 1 (PF = 1).

SETPE *reg/mem8*	0Fh 9Ah /0	Fixe l'octet si le drapeau de parité vaut 1 (PF = 1).
SETNP *reg/mem8*	0Fh 9Bh /0	Fixe l'octet si le drapeau de parité vaut 0 (PF = 0).
SETPO *reg/mem8*	0Fh 9Bh /0	Fixe l'octet si le drapeau de parité vaut 0 (PF = 0).
SETL *reg/mem8*	0Fh 9Ch /0	Fixe l'octet si SF <> OF.
SETNGE *reg/mem8*	0Fh 9Ch /0	Fixe l'octet si SF <> OF.
SETNL *reg/mem8*	0Fh 9Dh /0	Fixe l'octet si SF = OF.
SETGE *reg/mem8*	0Fh 9Dh /0	Fixe l'octet si SF = OF.
SETLE *reg/mem8*	0Fh 9Eh /0	Fixe l'octet si ZF = 1 ou SF <> OF.
SETNG *reg/mem8*	0Fh 9Eh /0	Fixe l'octet si ZF = 1 ou SF <> OF.
SETNLE *reg/mem8*	0Fh 9Fh /0	Fixe l'octet si ZF = 0 et SF = OF.
SETG *reg/mem8*	0Fh 9Fh /0	Fixe l'octet si ZF = 0 et SF = OF.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique

#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 834 à 836
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 225.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 418 à 422.

Assembleur 80x86	**SETALC**
INTEL 8088+	

Syntaxe

```
SETALC
```

Description

Cette instruction permet de copier la valeur du drapeau de retenue dans le registre AL en la multipliant par 0FFh.

Algorithme

```
AL ← 0FFh x Drapeau CF
```

Mnémonique

Instruction	**Opcode**	**Description**
SETALC	D6h	Cette instruction permet de copier la valeur du drapeau de retenue dans le registre AL en la multipliant par 0FFh.

Assembleur 80x86 — SFENCE

INTEL Pentium III (SSE)+ — *Store Fence*

Syntaxe

```
SFENCE
```

Description

Cette instruction permet d'agir comme une barrière pour forcer une priorité en mémoire (sérialisation) entre les instructions précédant emmagasiner du *SFENCE* et les instructions suivant le *SFENCE*.

Mnémonique

Instruction	Opcode	Description
SFENCE	0Fh AEh F8h	Force l'ordre de sérialisation des opérations entreposés.

Voir également

Instruction assembleur 80x86 - Instruction LFENCE
Instruction assembleur 80x86 - Instruction MFENCE

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 423.

Assembleur 80x86

INTEL 80386+

SGDT

Store Global Descriptor Table

Syntaxe

SGDT *operande*

Description

Cette instruction permet d'entreposer le sélecteur de segment dans le registre *GDTR* (registre de table global de descripteur) dans l'opérande de destination.

Algorithme

SI taille de l'*operande* = 16 **ALORS**
  operande (0..15) ← GDTR(Limite)
  operande (16..39) ← GDTR(Base)
  operande (40..47) ← 0
SINON
  operande (0..15) ← GDTR(Limite)
  operande (16..47) ← GDTR(Base)
FIN SI

Mnémonique

Instruction	Opcode	Description
SGDT *mem*	0Fh 01h /0	Cette instruction permet d'entreposer le sélecteur de segment dans le registre *GDTR* (registre de table global de descripteur) dans

		l'opérande de destination.

Voir également

Instruction assembleur 80x86 - Instruction SLDT
Instruction assembleur 80x86 - Instruction SIDT
Instruction assembleur 80x86 - Instruction STR
Instruction assembleur 80x86 - Instruction LIDT
Instruction assembleur 80x86 - Instruction LGDT
Instruction assembleur 80x86 - Instruction LLDT
Instruction assembleur 80x86 - Instruction LTR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 424 à 426.

Assembleur 80x86	**SHL**
INTEL 8088+	*Shift Left*

Syntaxe

SHL *opérandecible,nombredebits*

Description

Cette instruction permet d'effectuer une rotation des bits vers la gauche en réinsérant le bit dans l'indicateur de retenue (*CF*).

Algorithme

```
temp ← compteur ∩ 001Fh
POUR FAIRE TANT QUE temp1 = 0
  CF ← highorder ( destination )
  destination ← destination décalage à gauche de 1
  temp ← temp - 1
POUR FIN DE FAIRE
SI count = 1 ALORS
  OF ← highorder ( destination ) <> CF
SINON
  OF ← ?
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SHL *reg/mem8*, 1	D0h /4	Cette instruction effectue une déclage d'un seul bit vers la gauche de 8 bits d'un opérande registre ou

		mémoire 8 bits.
SHL *reg/mem8*, CL	D2h /4	Cette instruction effectue une déclage du nombre de bits spécifié par le registre CL vers la gauche de 8 bits d'un opérande registre ou mémoire 8 bits.
SHL *reg/mem8*, *imm8*	C0h /4 ib	Cette instruction effectue une déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 8 bits.
SHL *reg/mem16*, 1	D1h /4	Cette instruction effectue une déclage d'un seul bit vers la gauche de 16 bits d'un opérande registre ou mémoire 16 bits.
SHL *reg/mem16*, CL	D3h /4	Cette instruction effectue une déclage du nombre de bits spécifié par le registre CL vers la gauche de 16 bits d'un opérande registre ou mémoire 16 bits.
SHL *reg/mem16*, *imm8*	C1h /4 ib	Cette instruction effectue une déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 16 bits d'un opérande registre ou mémoire 16 bits.
SHL *reg/mem32*, 1	D1h /4	Cette instruction effectue une déclage d'un seul bit vers la gauche de 32 bits d'un opérande registre ou mémoire 32 bits.
SHL *reg/mem32*, CL	D3h /4	Cette instruction effectue une déclage du nombre de bits spécifié par le registre CL vers la gauche de 32 bits d'un opérande registre ou

		mémoire 32 bits.
SHL *reg/mem32*, *imm8*	C1h /4 ib	Cette instruction effectue une déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 32 bits d'un opérande registre ou mémoire 32 bits.
SHL *reg/mem64*, 1	D1h /4	Cette instruction effectue une déclage d'un seul bit vers la gauche de 64 bits d'un opérande registre ou mémoire 64 bits.
SHL *reg/mem64*, CL	D3h /4	Cette instruction effectue une déclage du nombre de bits spécifié par le registre CL vers la gauche de 64 bits d'un opérande registre ou mémoire 64 bits.
SHL *reg/mem64*, *imm8*	C1h /4 ib	Cette instruction effectue une déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 64 bits d'un opérande registre ou mémoire 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-canonique)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection	X	X	X	Une adresse

général)				mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction SAR
Instruction assembleur 80x86 - Instruction SHR
Instruction assembleur 80x86 - Instruction SHLD
Instruction assembleur 80x86 - Instruction SHRD

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 836*
Assembleur Facile, *Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 415*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 216.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 400 à 408.*

Assembleur 80x86 — SHLD

INTEL 80386+ — *Shift Left Double Precision*

Syntaxe

SHLD *operande16*, *operande16*, *immediat8*
SHLD *operande16*, *operande16*, CL
SHLD *operande32*, *operande32*, *immediat8*
SHLD *operande32*, *operande32*, CL

Description

Cette instruction permet d'effectuer une rotation des bits d'un double mot vers la gauche en réinsérant le bit dans l'indicateur de retenue (*CF*).

Algorithme

```
COUNT ← COUNT ⋂ 1Fh
SIZE ← taille de l'opérande
SI COUNT = 0 ALORS
  pas d'opération
SINON
  SI COUNT >= SIZE ALORS
    DEST est indéfinis
    CF, OF, SF, ZF, AF, PF sont indéfinis
  SINON
    CF ← BIT(DEST, SIZE - COUNT)
    BOUCLE POUR i ← SIZE - 1 JUSQU'A COUNT FAIRE
      BIT(DEST, i) ← BIT(DEST, i - COUNT)
    FIN BOUCLE POUR
    BOUCLE POUR i ← COUNT - 1 JUSQU'A 0 FAIRE
```

```
      BIT(DEST, i) ← BIT(inBits,i - COUNT + SIZE)
    FIN BOUCLE POUR
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SHLD *reg/mem16*, *reg16*, *imm8*	0Fh A4h /r *ib*	Effecute un décalage de bits vers la gauche de la combinaison d'une opérande registre ou mémoire 16 bits et d'un registre 16 bits du nombre de bits spécifié par la valeur immédiate de 8 bits.
SHLD *reg/mem16*, *reg16*, CL	0Fh A5h /r	Effecute un décalage de bits vers la gauche de la combinaison d'une opérande registre ou mémoire 16 bits et d'un registre 16 bits du nombre de bits spécifié par le registre CL.
SHLD *reg/mem32*, *reg32*, *imm8*	0Fh A4h /r *ib*	Effecute un décalage de bits vers la gauche de la combinaison d'une opérande registre ou mémoire 32 bits et d'un registre 32 bits du nombre de bits spécifié par la valeur immédiate de 8 bits.
SHLD *reg/mem32*, *reg32*, CL	0Fh A5h /r	Effecute un décalage de bits vers la gauche de la combinaison d'une opérande registre ou mémoire 32 bits et d'un registre 32 bits du nombre de bits spécifié par le registre CL.
SHLD *reg/mem64*, *reg64*, *imm8*	0Fh A4h /r *ib*	Effecute un décalage de bits vers la

		gauche de la combinaison d'une opérande registre ou mémoire 64 bits et d'un registre 64 bits du nombre de bits spécifié par la valeur immédiate de 8 bits.
SHLD *reg/mem64*, *reg64*, CL	0Fh A5h /r	Effecute un décalage de bits vers la gauche de la combinaison d'une opérande registre ou mémoire 64 bits et d'un registre 64 bits du nombre de bits spécifié par le registre CL.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment

				non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction SHRD
Instruction assembleur 80x86 - Instruction SAL
Instruction assembleur 80x86 - Instruction SAR
Instruction assembleur 80x86 - Instruction SHR
Instruction assembleur 80x86 - Instruction SHL

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 427 à 429.*

Assembleur 80x86	**SHR**
INTEL 8088+	*Shift Right*

Syntaxe

SHR *opérandecible,nombredebits*

Description

Cette instruction permet d'effectuer une rotation des bits vers la droite en réinsérant le bit dans l'indicateur de retenue (*CF*).

Algorithme

```
temp ← compteur ∩ 001Fh
BOUCLE FAIRE TANT temp <> 0
  x ← highorder ( destination )
  CF ← destination ∩ 1
  destination ← destination décalage à droite de 1
  highorder ( destination ) ← x
  temp ← temp - 1
BOUCLE FIN DE FAIRE
SI compteur = 1 ALORS
  OF ← 0
SINON
  OF ← ?
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SHR *reg/mem8*, 1	D0h /5	Cette instruction effectue un déclage d'un seul bit vers la droite d'un

		opérande registre ou mémoire 8 bits.
SHR *reg/mem8*, CL	D2h /5	Cette instruction effectue un déclage du nombre de bits spécifié par le registre CL vers la gauche d'un opérande registre ou mémoire 8 bits.
SHR *reg/mem8*, *imm8*	C0h /5 ib	Cette instruction effectue un déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 8 bits.
SHR *reg/mem16*, 1	D1h /5	Cette instruction effectue un déclage d'un seul bit vers la droite d'un opérande registre ou mémoire 16 bits.
SHR *reg/mem16*, CL	D3h /5	Cette instruction effectue un déclage du nombre de bits spécifié par le registre CL vers la gauche d'un opérande registre ou mémoire 16 bits.
SHR *reg/mem16*, *imm8*	C1h /5 *ib*	Cette instruction effectue un déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 16 bits.
SHR *reg/mem32*, 1	D1h /5	Cette instruction effectue un déclage d'un seul bit vers la droite d'un opérande registre ou mémoire 32 bits.
SHR *reg/mem32*, CL	D3h /5	Cette instruction effectue un déclage du nombre de bits spécifié par le registre CL vers la gauche d'un opérande registre ou mémoire 32

		bits.
SHR *reg/mem32*, *imm8*	C1h /5 *ib*	Cette instruction effectue un déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 32 bits.
SHR *reg/mem64*, 1	D1h /5	Cette instruction effectue un déclage d'un seul bit vers la droite d'un opérande registre ou mémoire 64 bits.
SHR *reg/mem64*, CL	D3h /5	Cette instruction effectue un déclage du nombre de bits spécifié par le registre CL vers la gauche d'un opérande registre ou mémoire 64 bits.
SHR *reg/mem64*, *imm8*	C1h /5 *ib*	Cette instruction effectue un déclage du nombre de bits spécifié par la valeur immédiate vers la gauche de 8 bits d'un opérande registre ou mémoire 64 bits.

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection	X	X	X	Une adresse

général)				mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction SHL
Instruction assembleur 80x86 - Instruction SAL
Instruction assembleur 80x86 - Instruction SAR
Instruction assembleur 80x86 - Instruction SHLD
Instruction assembleur 80x86 - Instruction SHRD

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 836
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 417
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 231.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 400 à 408.

Assembleur 80x86 — SHRD

INTEL 80386+ — *Shift Right Double Precision*

Syntaxe

SHRD *operande16*, *operande16*, *immediat8*
SHRD *operande16*, *operande16*, CL
SHRD *operande32*, *operande32*, *immediat8*
SHRD *operande32*, *operande32*, CL

Description

Cette instruction permet d'effectuer une rotation des bits d'un double mot vers la droite en réinsérant le bit dans l'indicateur de retenue (*CF*).

Algorithme

```
COUNT ← COUNT ∩ 1Fh
SIZE ← taille de l'opérande
SI COUNT = 0 ALORS
  pas d'opération
SINON
  SI COUNT >= SIZE ALORS
    DEST est indéfinis
    CF, OF, SF, ZF, AF, PF sont indéfinis
  SINON
    CF ← BIT(DEST, COUNT - 1)
    BOUCLE POUR i ← 0 JUSQU'A SIZE - 1 - COUNT FAIRE
      BIT(DEST, i) ← BIT(DEST, i - COUNT)
    FIN BOUCLE POUR
    BOUCLE POUR i ← SIZE - COUNT JUSQU'A SIZE - 1 FAIRE
```

```
      BIT(DEST, i) ← BIT(inBits,i + COUNT - SIZE)
    FIN BOUCLE POUR
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SHRD *reg/mem16*, *reg16*, *imm8*	0Fh ACh /r *ib*	Effecute un décalage de bits vers la droite de la combinaison d'une opérande registre ou mémoire 16 bits et d'un registre 16 bits du nombre de bits spécifié par la valeur immédiate de 8 bits.
SHRD *reg/mem16*, *reg16*, CL	0Fh ADh /r	Effecute un décalage de bits vers la droite de la combinaison d'une opérande registre ou mémoire 16 bits et d'un registre 16 bits du nombre de bits spécifié par le registre CL.
SHRD *reg/mem32*, *reg32*, *imm8*	0Fh ACh /r *ib*	Effecute un décalage de bits vers la droite de la combinaison d'une opérande registre ou mémoire 32 bits et d'un registre 32 bits du nombre de bits spécifié par la valeur immédiate de 8 bits.
SHRD *reg/mem32*, *reg32*, CL	0Fh ADh /r	Effecute un décalage de bits vers la droite de la combinaison d'une opérande registre ou mémoire 32 bits et d'un registre 32 bits du nombre de bits spécifié par le registre CL.
SHRD *reg/mem64*, *reg64*, *imm8*	0Fh ACh /r *ib*	Effecute un décalage de bits vers la

		droite de la combinaison d'une opérande registre ou mémoire 64 bits et d'un registre 64 bits du nombre de bits spécifié par la valeur immédiate de 8 bits.
SHRD *reg/mem64*, *reg64*, CL	0Fh ADh /r	Effecute un décalage de bits vers la droite de la combinaison d'une opérande registre ou mémoire 64 bits et d'un registre 64 bits du nombre de bits spécifié par le registre CL.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment

				non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction SHLD
Instruction assembleur 80x86 - Instruction SHR
Instruction assembleur 80x86 - Instruction SHL
Instruction assembleur 80x86 - Instruction SAR
Instruction assembleur 80x86 - Instruction SAL

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 430 à 433.

Assembleur 80x86 — SHUFPD

INTEL Pentium 4 (SSE2)+ — *Shuffle Packed Double-Precision Floating-Point Values*

Syntaxe

SHUFPD *destination*, *source*,*select*

Description

Cette instruction permet de copier 4 paquets de valeurs de format réel de double précision dans un opérande destinataire dans la partie basse d'un quadruple mot de celle-ci et copie 2 des 4 paquets de valeurs de format réel de simple précision dans l'opérande source dans la partie haute d'un quadruple mot de l'opérande destinataire.

Algorithme

```
SI select(0) = 0 ALORS
  destination(0..63) ← destination(0..63)
SINON
  destination(0..63) ← destination(64..127)
FIN SI
SI select(1) = 0 ALORS
  destination(64..127) ← source(0..63)
SINON
  destination(64..127) ← source(64..127)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SHUFPD *xmm1*, *xmm2/m128*, *imm8*	66h 0Fh C6h /r *ib*	Cette instruction permet de

		copier 4 paquets de valeurs de format réel de double précision dans un opérande destinataire dans la partie basse d'un quadruple mot de celle-ci et copie 2 des 4 paquets de valeurs de format réel de simple précision dans l'opérande source dans la partie haute d'un quadruple mot de l'opérande destinataire.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 433 à 436.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

SHUFPS

Shuffle Packed Single-Precision Floating-Point Values

Syntaxe

SHUFPS *destination*, *source*, *select*

Description

Cette instruction permet de copier 4 paquets de valeurs de format réel de simple précision dans une opérande destinataire dans la partie basse d'un quadruple mot de celle-ci et copie 2 des 4 paquets de valeurs de format réel de simple précision dans l'opérande source dans la partie haute d'un quadruple mot de l'opérande destinataire.

Algorithme

```
EVALUER CAS select(0..1)
  CAS 0:
    destination(0..31) ← destination(0..31)
  CAS 1:
    destination(0..31) ← destination(32..63)
  CAS 2:
    destination(0..31) ← destination(64..95)
  CAS 3:
    destination(0..31) ← destination(96..127)
FIN EVALUER CAS
EVALUER CAS select(2..3)
  CAS 0:
    destination(32..63) ← destination(0..31)
  CAS 1:
    destination(32..63) ← destination(32..63)
  CAS 2:
    destination(32..63) ← destination(64..95)
```

```
  CAS 3:
    destination(32..63) ← destination(96..127)
FIN EVALUER CAS
EVALUER CAS select(4..5)
  CAS 0:
    destination(64..95) ← source(0..31)
  CAS 1:
    destination(64..95) ← source(32..63)
  CAS 2:
    destination(64..95) ← source(64..95)
  CAS 3:
    destination(64..95) ← source(96..127)
FIN EVALUER CAS
EVALUER CAS select(6..7)
  CAS 0:
    destination(96..127) ← source(0..31)
  CAS 1:
    destination(96..127) ← source(32..63)
  CAS 2:
    destination(96..127) ← source(64..95)
  CAS 3:
    destination(96..127) ← source(96..127)
FIN EVALUER CAS
```

Mnémonique

Instruction	Opcode	Description
SHUFPS *xmm*, *xmm*, *imm8*	0Fh C6h /r *ib*	Cette instruction permet de copier 4 paquets de valeurs de format réel de simple précision dans une opérande destinataire dans la partie basse d'un quadruple mot de celle-ci et copie 2 des 4 paquets de valeurs de format réel de simple précision dans l'opérande source dans la partie haute d'un quadruple mot de l'opérande destinataire.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 437 à 440.

Assembleur 80x86 — SIDT

INTEL 80386+ — *Store Interrupt Descriptor Table Register*

Syntaxe

SIDT *operande*

Description

Cette instruction permet d'entreposer le registre de descripteur de table d'interruption (*IDTR*) dans l'opérande de destination.

Algorithme

SI taille de l'opérande = 16 bits **ALORS**
  operande(0..15) ← IDTR(Limit)
  operande(16..39) ← IDTR(Base)
  operande(40..47) ← 0
SINON
  operande(0..15) ← IDTR(Limit)
  operande(16..47) ← IDTR(Base)
FIN SI

Mnémonique

Instruction	Opcode	Description
SIDT *mem*	0Fh 01h /1	Cette instruction permet d'entreposer le registre de descripteur de table d'interruption (*IDTR*) dans l'opérande de destination.

Voir également

Instruction	assembleur	80x86	-	Instruction	SLDT
Instruction	assembleur	80x86	-	Instruction	SGDT
Instruction	assembleur	80x86	-	Instruction	STR
Instruction	assembleur	80x86	-	Instruction	LIDT
Instruction	assembleur	80x86	-	Instruction	LGDT
Instruction	assembleur	80x86	-	Instruction	LLDT
Instruction	assembleur	80x86	-	Instruction	LTR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 441 à 444.

Syntaxe

SKINIT EAX

Description

Cette instruction permet de réinitialiser de façon sécuritaire le microprocesseur et de démarrer un logiciel de confiance comme un *VMM*.

Algorithme

```
SI ((EFER.SVMEN = 0) ET PAS (CPUID 8000_0001.ECX(SKINIT)) OU (PAS
PROTECTED_MODE)) ALORS
   EXCEPTION #UD()
FIN SI
SI CPL <> 0 ALORS
   EXCEPTION #GP()
FIN SI
Initialise l'état du microproceseur pour un signal INIT
CR0.PE ← 1
CS.sel ← 0008h
CS.attr ← 32-bit code, lecture/exécution
CS.base ← 0
CS.limit ← FFFFFFFFh
SS.sel ← 0010h
SS.attr ← pile 32-bits, lecture/écriture, haute expansion
SS.base ← 0
SS.limit ← FFFFFFFFh
EAX ← EAX ∩ FFFF0000h
EDX ← famille/modèle/niveau
ESP ← EAX + 00010000h
Efface le GPR autre que EAX, EDX, ESP
EFER ← 0
VM_CR.DPD ← 1
VM_CR.R_INIT ← 1
VM_CR.DIS_A20M ← 1
Active SL_DEV, et protège 64 Kiloctets de mémoire physique de démarrage à l'adresse
```

physique du registre EAX GIF ← 0 Lecture de la longueur du SL de l'offset 0002h du SLB Copie l'image du SL dans le TPM pour attestation Lecture du point d'entrée SL de l'offset 0000h du SLB Saut au point d'entrée SL, à EIP = EAX + offset de point d'entrée

Mnémonique

Instruction	**Opcode**	**Description**
SKINIT EAX	0Fh 01h DEh	Cette instruction permet d'effectuer l'initialisation sécurisé et le saut avec l'attestation.

Assembleur 80x86 SLDT

INTEL 80386+

Store Local Descriptor Table Register

Syntaxe

SLDT *operande*

Description

Cette instruction permet d'entreposer le sélecteur de segment dans le registre *LDTR* (registre de table local de descripteur) dans l'opérande de destination.

Algorithme

operande ← LDTR(*Selecteur de segment*)

Mnémonique

Instruction	Opcode	Description
SLDT *r/m16*	0Fh 00h /0	Cette instruction permet d'entreposer le sélecteur de segment dans le registre *LDTR* (registre de table local de descripteur) dans l'opérande de destination.

Voir également

Instruction assembleur 80x86 - Instruction SIDT
Instruction assembleur 80x86 - Instruction SGDT
Instruction assembleur 80x86 - Instruction STR
Instruction assembleur 80x86 - Instruction LIDT

Instruction assembleur 80x86 - Instruction LGDT
Instruction assembleur 80x86 - Instruction LLDT
Instruction assembleur 80x86 - Instruction LTR

Assembleur 80x86 — SMI

AMD Am386SXLV,Am386DXLV+

System Managment Interrupt

Syntaxe

```
SMI
```

Description

Cette instruction permet de gérer les interruptions systèmes en mode de déboguage.

Algorithme

```
SI SMIE = 1 ALORS
   Sauve l'état d'exécution de la SMRAM
   ENTER SMM
   SMMS ← 1
SINON
   INT 01h
FIN SI
```

Mnémonique

Instruction	Opcode	Description
SMI	F1h	Cette instruction permet de gérer les interruptions systèmes en mode de déboguage.

Assembleur 80x86 — SMINT

P6, Cyrix — *Software SMM Interrupt*

Syntaxe

```
SMINT
```

Description

Cette instruction permet de faire entrée le microprocesseur en mode *SMM* (*System Management Mode*).

Algorithme

```
Sauvegarde l'état CPU de l'entête SMM au sommet de l'espace SMM
Entre en mode SMM
```

Mnémonique

Instruction	Opcode	Description
SMINT	0Fh 38h	Cette instruction permet de faire entrée le microprocesseur en mode *SMM* (*System Management Mode*).

Assembleur 80x86

Cyrix Cx486DX/DX2/DX4, IBM BL486DX/DX2

SMINTOLD

Software SMM Interrupt

Syntaxe

SMINTOLD

Description

Cette instruction permet de faire entrée le microprocesseur en mode *SMM* (*System Management Mode*).

Algorithme

Sauvegarde l'état CPU de l'entête SMM au sommet de l'espace SMM
Entre en mode SMM

Mnémonique

Instruction	Opcode	Description
SMINTOLD	0Fh 7Eh	Cette instruction permet de faire entrée le microprocesseur en mode *SMM* (*System Management Mode*).

Assembleur 80x86 — SMSW

INTEL 80386+ — *Store Machine Status Word*

Syntaxe

SMSW *operande*

Description

Cette instruction permet d'entreposer le mot des états (soit les bits de 0 à 15 du registre de contrôle *CR0*) à partir du registre de tâche (*TR*) dans l'opérande cible.

Algorithme

operande ← CR0(15..0)

Mnémonique

Instruction	Opcode	Description
SMSW *r/m16*	0Fh 01h /4	Cette instruction permet d'entreposer le mot des états (soit les bits de 0 à 15 du registre de contrôle *CR0*) à partir du registre de tâche (*TR*) dans l'opérande cible.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 447 à 349.

Assembleur 80x86 — SQRTPD

INTEL Pentium 4 (SSE2)+

Square Roots of Packed Double-Precision Floating-Point Values

Syntaxe

SQRTPD *destination*, *source*

Description

Cette instruction permet d'effectuer le calcul de la racine carré d'un paquet de valeur de double précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de double précision.

Algorithme

destination(0..63) ← SquareRoot(*source*(0..63))
destination(64..127) ← SquareRoot(*source*(64..127))

Mnémonique

Instruction	Opcode	Description
SQRTPD *xmm1*, *xmm2/m128*	66h 0Fh 51h /r	Cette instruction permet d'effectuer le calcul de la racine carré d'un paquet de valeur de double précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 450 à 453.*

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

SQRTPS

Packed Single-Precision Floating Square Root

Syntaxe

SQRTPS *dest,source*

Description

Cette instruction permet d'effectuer le calcul de la racine carré d'un paquet de valeur de simple précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de simple précision.

Algorithme

dest(31..0) ← SQRT(*source*(31..0))
dest(63..32) ← SQRT(*source*(63..32))
dest(95..64) ← SQRT(*source*(95..64))
dest(127..96) ← SQRT(*source*(127..96))

Mnémonique

Instruction	Opcode	Description
SQRTPS *xmmreg,r/m128*	0Fh 51h /r	Cette instruction permet d'effectuer le calcul de la racine carré d'un paquet de valeur de simple précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel

		de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 453 à 455.

Assembleur 80x86 — SQRTSD

INTEL Pentium 4 (SSE2)+ — *Square Root of Scalar Double-Precision Floating-Point Value*

Syntaxe

SQRTSD *destination*, *source*

Description

Cette instruction permet d'effectuer le calcul de la racine carré d'une valeur de double précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de double précision.

Algorithme

destination(0..63) ← SquareRoot(*source*(0..63))

Mnémonique

Instruction	Opcode	Description
SQRTSD *xmm1*, *xmm2/m64*	F2h 0Fh 51h /r	Cette instruction permet d'effectuer le calcul de la racine carré d'une valeur de double précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 456 à 458.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

SQRTSS

Scalar Single-Precision Floating-Point Square Root

Syntaxe

SQRTSS *dest*,*source*

Description

Cette instruction permet d'effectuer le calcul de la racine carré d'une valeur de simple précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de simple précision.

Algorithme

dest(31..0) ← SQRT(*source*(31..0))

Mnémonique

Instruction	Opcode	Description
SQRTSS *xmm1,xmm2/m32*	F3h 0Fh 51h /r	Cette instruction permet d'effectuer le calcul de la racine carré d'une valeur de simple précision réel de l'opérande source et de mettre son résultat dans l'opérande de destination sous forme d'un réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 459 à 460.*

Assembleur 80x86	**SS:**
INTEL 8088+	

Syntaxe

instruction **SS:**[*adresse*]

Description

Ce préfixe permet d'indiquer à une instruction d'utiliser le Segment de Pile (*Stack Segment*) pour adresse dans l'opérande.

Assembleur 80x86	STC
INTEL 8088+	*Set Carry Flag*

Syntaxe

```
STC
```

Description

Cette instruction permet de fixer l'indicateur de retenue (*CF*) à la valeur 1.

Algorithme

```
drapeau CF ← 1
```

Remarque

A titre purement informatif, sachez que ce registre d'état de retenue est toujours égal à 1 lorsqu'une opération génère une retenue arithmétique et qu'il est par conséquent impossible de représenter le résultat qu'avec l'aide des bits disponibles. Dans ce éventualité, l'indicateur de retenue est considéré comme $17^{ième}$ ou $9^{ième}$ bit du résultat. C'est surtout dans la perspective des instructions *ADC* et *SBB* que le positionnement à 0 de l'indicateur de retenue s'avère utile. Dans les autres cas, l'interruption 21h du système d'exploitation sans servira pour indiquer s'il y a eu une erreur d'exécution, toutefois ce n'est pas une situation relier directement au microprocesseur mais une programmation logiciel.

Mnémonique

Instruction	Opcode	Description
STC	F9h	Fixe le registre de drapeau de retenue à 1

Exception

Aucune

Voir également

Instruction assembleur 80x86 - Instruction CLC
Instruction assembleur 80x86 - Instruction CMC

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 836*
Assembleur Facile, *Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 417*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 235.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 461.*

Assembleur 80x86	**STD**
INTEL 8088+	*Set Direction Flag*

Syntaxe

```
STD
```

Description

Cette instruction permet de fixer l'indicateur de direction (*DF*) à la valeur 1. Les opérations sur les chaînes de caractères, tel *CMPSB*, *CMPSD*, *CMPSQ*, *CMPSW*, *LODSB*, *LODSD*, *LODSQ*, *LODSW*, *MOVSB*, *MOVSD*, *MOVSQ*, *MOVSW*, *OUTSB*, *OUTSD*, *OUTSW*, *STOSB*, *STOSD*, *STOSQ* et *STOSW* par exemple, incrémenteront maintenant les registres *SI* et *DI*.

Algorithme

```
drapeau DF ← 1
```

Mnémonique

Instruction	Opcode	Description
STD	F9h	Fixe le registre de drapeau de direction à 1

Exception

Aucune

Voir également

Instruction assembleur 80x86 - Instruction CLD

Instruction assembleur 80x86 - Instruction INS
Instruction assembleur 80x86 - Instruction LODS
Instruction assembleur 80x86 - Instruction MOVS
Instruction assembleur 80x86 - Instruction OUTS
Instruction assembleur 80x86 - Instruction SCAS
Instruction assembleur 80x86 - Instruction STOS
Instruction assembleur 80x86 - Instruction CMPS

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 836
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 417
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 236.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 462.

Assembleur 80x86 STGI

AMD-V

Set Global Interrupt Flag

Syntaxe

```
STGI
```

Description

Cette instruction permet de fixer les drapeaux global d'interruption (*GIF*). Quand le *GIF* est à 1, les interruptions externes sont réactivés.

Algorithme

```
drapeau GIF ← 1
```

Mnémonique

Instruction	Opcode	Description
STGI	0Fh 01h DCh	Cette instruction permet de fixer les drapeaux global d'interruption (*GIF*).

Voir également

Instruction assembleur 80x86 - Instruction CLGI

Assembleur 80x86	**STI**
INTEL 8088+	*Set Interrupt Flag*

Syntaxe

```
STI
```

Description

Cette instruction permet de fixer l'indicateur d'interruption (*IF*) à la valeur 1. Après avoir exécuter cette instruction, aucune interruption ne sera admise tant que l'instruction *STI* n'est pas rencontrée.

Algorithme

```
SI CPL <= IOPL ALORS
  drapeau IF ← 1
SINON SI (((VIRTUAL_MODE) ET (CR4.VME = 1)) OU ((PROTECTED_MODE) ET (CR4.PVI = 1) ET
(CPL = 3))) ALORS
  SI RFLAGS.VIP = 1 ALORS
    EXCEPTION #GP(0)
  FIN SI
  RFLAGS.VIF ← 1
SINON
  EXCEPTION #GP(0)
FIN SI
```

Remarque

- Afin que les choses soit bien claire, sachez que même si ce registre d'état d'interruption est mit à 0, le microprocesseur ne masque pas les interruptions de type *NMI* (*Non masquable Interruption*). La commande *CLI* vise donc à faire en sorte que toutes les interruptions masquables ne soient plus exécutées. Cette interdiction peut être levée à l'aide de la commande *STI*.

Mnémonique

Instruction	Opcode	Description
STI	FBh	Cette instruction permet de fixer l'indicateur d'interruption (*IF*) à la valeur 1.

Voir également

Instruction assembleur 80x86 - Instruction CLI

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 836
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 417
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 305.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 463 à 465.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

STMXCSR

Store MXCSR

Syntaxe

STMXCSR *dest*

Description

Cette instruction permet de copier le *MXCSR* dans un emplacement mémoire de 32 bits.

Algorithme

dest ← MXCSR

Mnémonique

Instruction	Opcode	Description
STMXCSR *m32*	0Fh AEh /3	Cette instruction permet de copier le *MXCSR* dans un emplacement mémoire de 32 bits.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 466 à 467.

Assembleur 80x86	**STOS**
INTEL 8088+	*Store String Data*

Syntaxe

STOS *chainecible*

Description

Cette instruction permet de copier un octet, un mot ou un double mot dans la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI en fonction de la taille de l'opérande cible et de l'état du drapeau de direction.

Algorithme

```
SI chainecible est octet ALORS
  chainecible ← AL
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 1
  SINON
    (E)DI ← (E)DI - 1
  FIN SI
SINON SI chainecible est un mot ALORS
  chainecible ← AX
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 2
  SINON
    (E)DI ← (E)DI - 2
  FIN SI
SINON SI chainecible est un double mot ALORS
  chainecible ← EAX
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 4
  SINON
    (E)DI ← (E)DI - 4
  FIN SI
```

```
SINON
  chainecible ← RAX
  SI DF = 0 ALORS
    (E)DI ← (E)DI + 8
  SINON
    (E)DI ← (E)DI - 8
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
STOS *mem8*	AAh	Entrepose le contenu du registre AL dans ES:(R)DI et incrémente ou décrémente (R)DI.
STOS *mem16*	ABh	Entrepose le contenu du registre AX dans ES:(R)DI et incrémente ou décrémente (R)DI.
STOS *mem32*	ABh	Entrepose le contenu du registre EAX dans ES:(R)DI et incrémente ou décrémente (R)DI.
STOS *mem64*	ABh	Entrepose le contenu du registre RAX dans ES:(R)DI et incrémente ou décrémente (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du

				segment ES ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment ES non écrivable
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction LODS
Instruction assembleur 80x86 - Instruction MOVS

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 837*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 237.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 468 à 472.*

Assembleur 80x86 — STOSB

INTEL 8088+ — *Store String Byte*

Syntaxe

```
STOSB
```

Description

Cette instruction permet de copier le registre AL dans la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 1 en fonction de l'état du drapeau de direction.

Algorithme

```
chainecible ← AL
SI DF = 0 ALORS
  (E)DI ← (E)DI + 1
SINON
  (E)DI ← (E)DI - 1
FIN SI
```

Mnémonique

Instruction	Opcode	Description
STOSB	AAh	Entrepose le contenu du registre AL dans ES:(R)DI et incrémente ou décrémente (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment ES ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment ES non écrivable
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Exemples

Cet exemple suivant, destiné au 8088 mais peu performante, permet de remplir avec la valeur de la variable «*Valeur*» un tampon d'octets spécifié par la variable «*Tampon*» de la longueur spécifié par la variable «*Longueur*» avec une cadence de 8 bits à la fois :

```
1. CLD
2. LES DI,Tampon
3. MOV CX,Longueur
4. MOV AL,Valeur
5. REP STOSB
```

Cet exemple suivant, destiné au 8086 et très performante, permet de remplir avec la valeur de la variable «*Valeur*» un tampon d'octets spécifié par la variable «*Tampon*» de la longueur spécifié par la variable «*Longueur*» avec une cadence de 16 bits à la fois :

```
1. CLD
2. LES DI,Tampon
3. MOV CX,Longueur
4. MOV AL,Valeur
5. MOV AH,AL
6. SHR CX,1
7. REP STOSW
8. ADC CX,CX
9. REP STOSB
```

Cet exemple suivant, destiné au 80386 ou plus, permet de remplir avec la valeur de la variable «*Valeur*» un tampon d'octets spécifié par la variable «*Tampon*» de la longueur spécifié par la variable «*Longueur*» avec une cadence de 32 bits à la fois :

```
1. CLD
2. LES DI,Tampon
3. MOV CX,Longueur
4. MOV AL,Valeur
5. MOV AH,AL
6.  ; EAX := AX:AX
7. PUSH AX
8. PUSH AX
9. POP EAX
10. MOV BX,CX
11. AND BX,3
12. SHR CX,2
13. REP STOSD
```

```
14. MOV CX,BX
15. REP STOSB
```

Voir également

Instruction assembleur 80x86 - Instruction STOSD
Instruction assembleur 80x86 - Instruction STOSQ
Instruction assembleur 80x86 - Instruction STOSW

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 837
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 418
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 237.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 468 à 472.

Assembleur 80x86	**STOSD**
INTEL 80386+	*Store String Doubleword*

Syntaxe

```
STOSD
```

Description

Cette instruction permet de copier le registre EAX dans la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 4 en fonction de l'état du drapeau de direction.

Algorithme

```
chainecible ← EAX
SI DF = 0 ALORS
  (E)DI ← (E)DI + 4
SINON
  (E)DI ← (E)DI - 4
FIN SI
```

Mnémonique

Instruction	Opcode	Description
STOSD	ABh	Entrepose le contenu du registre EAX dans ES:(R)DI et incrémente ou décrémente (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment ES ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment ES non écrivable
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 837
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 237.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 468 à 472.

Assembleur 80x86 STOSQ

x86-64+

Store String Quadword

Syntaxe

```
STOSQ
```

Description

Cette instruction permet de copier le registre RAX dans la cellule mémoire à l'adresse ES:[(R)DI] et incrémente/décrémente le registre (R)DI de 8 en fonction de l'état du drapeau de direction.

Algorithme

```
chainecible ← RAX
SI DF = 0 ALORS
  (E)DI ← (E)DI + 8
SINON
  (E)DI ← (E)DI - 8
FIN SI
```

Mnémonique

Instruction	Opcode	Description
STOSQ	ABh	Entrepose le contenu du registre RAX dans ES:(R)DI et incrémente ou décrémente (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment ES ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment ES non écrivable
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 468 à 472.*

Assembleur 80x86 — STOSW

INTEL 8088+ — *Store String Word*

Syntaxe

```
STOSW
```

Description

Cette instruction permet de copier le registre AX dans la cellule mémoire à l'adresse ES:[DI] et incrémente/décrémente le registre DI de 2 en fonction de l'état du drapeau de direction.

Algorithme

```
chainecible ← AX
SI DF = 0 ALORS
  (E)DI ← (E)DI + 2
SINON
  (E)DI ← (E)DI - 2
FIN SI
```

Mnémonique

Instruction	Opcode	Description
STOSW	ABh	Entrepose le contenu du registre AX dans ES:(R)DI et incrémente ou décrémente (R)DI.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment ES ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment ES non écrivable
			X	Un segment ES nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Exemple

L'exemple suivant, s'adressant au mode réel sous un système d'exploitation *DOS*, permet d'entreproser dans des pointeurs de variables le contenu de la date :

```
1.  MOV AH,2Ah
2.  INT 21h
3.  CLD
4.  LES DI,DayOfWeek
5.  STOSB
6.  MOV AL,DL
7.  LES DI,Day
8.  STOSB
9.  MOV AL,DH
10. LES DI,Month
11. STOSB
12. XCHG AX,CX
13. LES DI,Year
14. STOSW
```

Références

Le livre d'Or PC, *Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 838*
Assembleur Facile, *Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 418*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 237.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 468 à 472.*

Assembleur 80x86 STR

INTEL 80286+ *Store Task Register*

Syntaxe

STR *operande*

Description

Cette instruction permet d'entreposer le sélecteur de segment à partir du registre de tâche (*TR*) à l'opérande cible.

Fonction

operande ← TR(Sélecteur de segment)

Mnémonique

Instruction	Opcode	Description
STR *r/m16*	0Fh 00h /1	>Cette instruction permet d'entreposer le sélecteur de segment à partir du registre de tâche (*TR*) à l'opérande cible.

Voir également

Instruction assembleur 80x86 - Instruction LGDT
Instruction assembleur 80x86 - Instruction LIDT
Instruction assembleur 80x86 - Instruction LLDT
Instruction assembleur 80x86 - Instruction LTR
Instruction assembleur 80x86 - Instruction SIDT

Instruction assembleur 80x86 - Instruction SGDT
Instruction assembleur 80x86 - Instruction SLDT

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 474 à 475.

Assembleur 80x86	**SUB**
INTEL 8088+	*Arithmetic Substraction Unsigned*

Syntaxe

SUB *opérandecible,opérandesource*

Description

Cette instruction permet de soustraire une valeur à une opérande.

Algorithme

opérandecible ← *opérandecible - opérandesource*

Mnémonique

Instruction	Opcode	Description
SUB AL, *imm8*	2Ch *ib*	Soustrait une valeur immédiate de 8 bits du registre AL et entrepose le résultat dans le registre AL.
SUB AX, *imm16*	2Dh *iw*	Soustrait une valeur immédiate de 16 bits du registre AX et entrepose le résultat dans le registre AX.
SUB EAX, *imm32*	2Dh *id*	Soustrait une valeur immédiate de 32 bits du registre EAX et entrepose le résultat dans le registre EAX.
SUB RAX, *imm32*	2Dh *id*	Soustrait une valeur entière immédiate de 32 bits du registre RAX et entrepose le résultat dans le

		registre RAX.
SUB *reg/mem8*, *imm8*	80h /5 *ib*	Soustrait une valeur entière immédiate de 8 bits de l'emplacement registre ou mémoire de 8 bits.
SUB *reg/mem16*, *imm16*	81h /5 *iw*	Soustrait une valeur entière immédiate de 16 bits de l'emplacement registre ou mémoire de 16 bits.
SUB *reg/mem32*, *imm32*	81h /5 *id*	Soustrait une valeur entière immédiate de 32 bits de l'emplacement registre ou mémoire de 32 bits.
SUB *reg/mem64*, *imm32*	81h /5 *id*	Soustrait une valeur entière immédiate de 64 bits de l'emplacement registre ou mémoire de 32 bits.
SUB *reg/mem16*, *imm8*	83h /5 *ib*	Soustrait une valeur entière immédiate de 16 bits de l'emplacement registre ou mémoire de 8 bits.
SUB *reg/mem32*, *imm8*	83h /5 *ib*	Soustrait une valeur entière immédiate de 32 bits de l'emplacement registre ou mémoire de 8 bits.
SUB *reg/mem64*, *imm8*	83h /5 *ib*	Soustrait une valeur entière immédiate de 64 bits de l'emplacement registre ou mémoire de 8 bits.
SUB *reg/mem8*, *reg8*	28h /r	Soustrait un registre de 8 bits de l'emplacement registre ou mémoire

		de 8 bits.
SUB *reg/mem16*, *reg16*	29h /r	Soustrait un registre de 16 bits de l'emplacement registre ou mémoire de 16 bits.
SUB *reg/mem32*, *reg32*	29h /r	Soustrait un registre de 32 bits de l'emplacement registre ou mémoire de 32 bits.
SUB *reg/mem64*, *reg64*	29h /r	Soustrait un registre de 64 bits de l'emplacement registre ou mémoire de 64 bits.
SUB *reg8*, *reg/mem8*	2Ah /r	Soustrait de l'opérande d'emplacement registre ou mémoire de 8 bits d'un registre de 8 bits.
SUB *reg16*, *reg/mem16*	2Bh /r	Soustrait de l'opérande d'emplacement registre ou mémoire de 16 bits d'un registre de 16 bits.
SUB *reg32*, *reg/mem32*	2Bh /r	Soustrait de l'opérande d'emplacement registre ou mémoire de 32 bits d'un registre de 32 bits.
SUB *reg64*, *reg/mem64*	2Bh /r	Soustrait de l'opérande d'emplacement registre ou mémoire de 64 bits d'un registre de 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du

				segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification

				d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction ADC
Instruction assembleur 80x86 - Instruction ADD
Instruction assembleur 80x86 - Instruction SBB

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 838
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 418
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 239.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 476 à 478.

Assembleur 80x86 — SUBPD

INTEL Pentium 4 (SSE2)+ — *Subtract Packed Double-Precision Floating-Point Values*

Syntaxe

SUBPD *destination*, *source*

Description

Cette instruction permet d'effectuer une soustraction de 2 paquets de valeurs réels de double précision d'un opérande source et d'un opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de double précision.

Algorithme

destination(0..63) ← *destination*(0..63) - *source*(0..63)
destination(64..127) ← *destination*(64..127) - *source*(64..127)

Mnémonique

Instruction	Opcode	Description
SUBPD *xmm1*, *xmm2/m128*	66h 0Fh 5Ch /r	Cette instruction permet d'effectuer une soustraction de 2 paquets de valeurs réels de double précision d'un opérande source et d'un opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 479 à 480.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

SUBPS

Packed Single-Float-Point Subtract

Syntaxe

SUBPS *destination,source*

Description

Cette instruction permet d'effectuer une soustraction de 4 paquets de valeurs réels de simple précision d'une opérande source et d'une opérande destination et entrepose le résultat dans l'opérande de destination sous le format d'un paquet de valeurs réels de simple précision.

Algorithme

destination(31..0) ← *destination*(31..0) - *source*(31..0)
destination(63..32) ← *destination*(63..32) - *source*(63..32)
destination(95..64) ← *destination*(95..64) - *source*(95..64)
destination(127..96) ← *destination*(127..96) - *source*(127..96)

Mnémonique

Instruction	Opcode	Description
SUBPS *xmm1,xmm2/m128*	0Fh 5Ch /r	Cette instruction permet d'effectuer une soustraction de 4 paquets de valeurs réels de simple précision d'une opérande source et d'une opérande destination et entrepose le résultat dans l'opérande de

		destination sous le format d'un paquet de valeurs réels de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 481 à 483.

Assembleur 80x86 — SUBSD

INTEL Pentium 4 (SSE2)+ — *Subtract Scalar Double-Precision Floating-Point Values*

Syntaxe

SUBSD *destination*, *source*

Description

Cette instruction permet d'effectuer une soustraction de la partie basse d'une valeur réel de double précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de double précision.

Algorithme

destination(0..63) ← *destination*(0..63) - *source*(0..63)

Mnémonique

Instruction	Opcode	Description
SUBSD *xmm1*, *xmm2/m64*	F2h 0Fh 5Ch /r	Cette instruction permet d'effectuer une soustraction de la partie basse d'une valeur réel de double précision d'un opérande source et destination et entrepose le résultat dans un opérande de destination de valeur réel de double précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 484 à 486.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

SUBSS

Scalar Single-Float-Point Subtract

Syntaxe

SUBSS *destination,source*

Description

Cette instruction permet d'effectuer une soustraction de la partie basse d'une valeur réel de simple précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de simple précision.

Algorithme

destination(31..0) ← *destination*(31..0) - *source*(31..0)

Mnémonique

Instruction	Opcode	Description
SUBSS *xmm1,xmm2/m32*	F3h 0Fh 5Ch /r	Cette instruction permet d'effectuer une soustraction de la partie basse d'une valeur réel de simple précision d'une opérande source et destination et entrepose le résultat dans une opérande de destination de valeur réel de simple précision.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 487 à 489.*

Assembleur 80x86

Cyrix Cx486S/S2/D/D2/DX/DX2/DX4, IBM BL486DX/DX2, TI 486SLC/DLC/e, TI 486SXL/SXL2/SXLC, TI Potomac

SVDC

Save Register and Descriptor

Syntaxe

SVDC *destination*, *source*

Description

Cette instruction permet de sauvegarder le registre et le descripteur.

Algorithme

destination ← *source*[*sélecteur*,*shadow_descriptor*]

Mnémonique

Instruction	Opcode	Description
SVDC *mem80,sreg*	0Fh 78h (mm *sreg* mmm)	Cette instruction permet de sauvegarder le registre et le descripteur.

Assembleur 80x86

Cyrix Cx486S/S2/D/D2/DX/DX2/DX4, IBM BL486DX/DX2, TI 486SLC/DLC/e, TI 486SXL/SXL2/SXLC, TI Potomac

SVLDT

Save Register and Descriptor

Syntaxe

SVLDT *destination*

Description

Cette instruction permet de sauvegarder le LDTR et le descripteur.

Algorithme

destination ← LDTR[*sélecteur*,*shadow_descriptor*]

Mnémonique

Instruction	**Opcode**	**Description**
SVLDT *mem80*	0Fh 7Ah (mm 000b mmm)	Cette instruction permet de sauvegarder le LDTR et le descripteur.

Assembleur 80x86

Cyrix Cx486S/S2/D/D2/DX/DX2/DX4, IBM BL486DX/DX2, TI 486SLC/DLC/e, TI 486SXL/SXL2/SXLC, TI Potomac

SVTS

Save TR and Descriptor

Syntaxe

SVTS *destination*

Description

Cette instruction permet de sauvegarder le TR et le descripteur.

Algorithme

destination ← TR[*sélecteur*,*shadow_descriptor*]

Mnémonique

Instruction	**Opcode**	**Description**
SVTS *mem80*	0Fh 7Ch (mm 000b mmm)	Cette instruction permet de sauvegarder le TR et le descripteur.

Assembleur 80x86 SWAPGS

x86-64+

Swap GS Register with KernelGSbase MSR

Syntaxe

```
SWAPGS
```

Description

Cette instruction permet de fournir une méthode à un logiciel système pour charger un pointeur sur une structure de données système.

Mnémonique

Instruction	Opcode	Description
SWAPGS	0Fh 01h F8h	Cette instruction permet de fournir une méthode à un logiciel système pour charger un pointeur sur une structure de données système.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 490 à 491.

Assembleur 80x86 — SYSCALL

AMD K6-2+ — *Fast System Call*

Syntaxe

```
SYSCALL
```

Description

Cette instruction permet d'effectuer le transfert du contrôle d'un point d'entrée fixe au système d'exploitation.

Algorithme

```
SYSCALL_START:
  SI MSR_EFER.SCE = 0 ALORS
    EXCEPTION #UD()
  FIN SI
  SI LONG_MODE ALORS
    SYSCALL_LONG_MODE
  SINON
    SYSCALL_LEGACY_MODE
  FIN SI

SYSCALL_LONG_MODE:
  RCX.q ← next_RIP
  R11.q ← RFLAGS
  SI mode 64 bits ALORS
    temp_RIP.q ← MSR_LSTAR
  SINON
    temp_RIP.q ← MSR_CSTAR
    CS.sel ← MSR_STAR.SYSCALL_CS ∩ FFFCh
    CS.attr ← 64-bit code,dpl0
    CS.base ← 00000000h
    CS.limit ← FFFFFFFFh
```

```
    SS.sel ← MSR_STAR.SYSCALL_CS + 8
    SS.attr ← pile 64 bits,dpl0
    SS.base ← 00000000h
    SS.limit ← FFFFFFFFh
    RFLAGS ← RFLAGS ∩ ¬ MSR_SFMASK
    RFLAGS.RF ← 0
    CPL ← 0
    RIP ← temp_RIP
  FIN SI
  FIN

SYSCALL_LEGACY_MODE:
  RCX.d ← prochain RIP
  temp_RIP.d ← MSR_STAR.EIP
  CS.sel ← MSR_STAR.SYSCALL_CS ∩ FFFCh
  CS.attr ← code 32 bits,dpl0
  CS.base ← 00000000h
  CS.limit ← FFFFFFFFh
  SS.sel ← MSR_STAR.SYSCALL_CS + 8
  SS.attr ← pile 32 bits,dpl0
  SS.base ← 00000000h
  SS.limit ← FFFFFFFFh
  RFLAGS.VM,IF,RF ← 0
  CPL ← 0
  RIP ← temp_RIP
  FIN
```

Mnémonique

Instruction	Opcode	Description
SYSCALL	0Fh 05h	Cette instruction permet d'effectuer le transfert du contrôle d'un point d'entrée fixe au système d'exploitation.

Voir également

Instruction assembleur 80x86 - Instruction SYSENTER
Instruction assembleur 80x86 - Instruction SYSEXIT
Instruction assembleur 80x86 - Instruction SYSRET

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 492 à 493.

Assembleur 80x86 SYSENTER

INTEL Pentium Pro+ *System Enter*

Syntaxe

SYSENTER

Description

Cette instruction permet d'effectuer le transférer du contrôle d'un point d'entrée au système d'exploitation.

Mnémonique

Instruction	Opcode	Description
SYSENTER	0Fh 34h	Cette instruction permet d'effectuer le transférer du contrôle d'un point d'entrée au système d'exploitation.

Voir également

Instruction assembleur 80x86 - Instruction SYSCALL
Instruction assembleur 80x86 - Instruction SYSEXIT
Instruction assembleur 80x86 - Instruction SYSRET

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 494 à 498.

Assembleur 80x86 SYSEXIT

INTEL Pentium Pro+ *System Return*

Syntaxe

```
SYSEXIT
```

Description

Cette instruction permet de retourner du système d'exploitation à une application.

Mnémonique

Instruction	Opcode	Description
SYSEXIT	0Fh 35h	Cette instruction permet de retourner du système d'exploitation à une application.

Voir également

Instruction assembleur 80x86 - Instruction SYSCALL
Instruction assembleur 80x86 - Instruction SYSENTER
Instruction assembleur 80x86 - Instruction SYSRET

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 499 à 502.

Assembleur 80x86 — SYSRET

AMD K6-2+ — *Fast System Return*

Syntaxe

```
SYSRET
```

Description

Cette instruction permet de retourner du système d'exploitation à une application.

Algorithme

```
SYSRET_START:
  SI MSR_EFER.SCE = 0 ALORS
    EXCEPTION #UD()
  FIN SI
  SI ((PAS PROTECTED_MODE) OU (CPL != 0)) ALORS
    EXCEPTION #GP(0)
  FIN SI
  SI mode 64 bits ALORS
    ALLER A SYSRET_64BIT_MODE
  SINON
    ALLER A SYSRET_NON_64BIT_MODE
  FIN SI

SYSRET_64BIT_MODE:
  SI taille de l'opérande = 64 bits ALORS
    CS.sel ← (MSR_STAR.SYSRET_CS + 16) U 3
    CS.base ← 00000000h
    CS.limit ← FFFFFFFFh
    CS.attr ← 64-bit code,dpl3
    temp_RIP.q ← RCX
  SINON
```

```
    CS.sel ← MSR_STAR.SYSRET_CS U 3
    CS.base ← 00000000h
    CS.limit ← FFFFFFFFh
    CS.attr ← 32-bit code,dpl3
    temp_RIP.d = RCX
  FIN SI
  SS.sel ← MSR_STAR.SYSRET_CS + 8
  RFLAGS.q ← R11
  CPL ← 3
  RIP ← temp_RIP
  FIN

SYSRET_NON_64BIT_MODE:
  CS.sel ← MSR_STAR.SYSRET_CS U 3
  CS.base ← 00000000h
  CS.limit ← FFFFFFFFh
  CS.attr ← 32-bit code,dpl3
  temp_RIP.d ← RCX
  SS.sel ← MSR_STAR.SYSRET_CS + 8
  RFLAGS.IF ← 1
  CPL ← 3
  RIP ← temp_RIP
  FIN
```

Mnémonique

Instruction	Opcode	Description
SYSRET	0Fh 07h	Cette instruction permet de retourner du système d'exploitation à une application.

Voir également

Instruction assembleur 80x86 - Instruction SYSCALL
Instruction assembleur 80x86 - Instruction SYSENTER
Instruction assembleur 80x86 - Instruction SYSEXIT

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 503 à 504.

INTEL 8088+

Syntaxe

TEST *opérandecible,opérandesource*

Description

Cette instruction permet d'effectuer un «Et binaire» sur une opérande cible sans modifier sa valeur.

Algorithme

TEMP ← *opérandecible* ∩ *opérandesource*
SF ← MSB(TEMP)
SI TEMP = 0 **ALORS**
  ZF ← 1
SINON
  ZF ← 0
FIN SI
PF ← BitwiseXNOR(TEMP(0 à 7))
CF ← 0
OF ← 0

Mnémonique

Instruction	Opcode	Description
TEST AL, *imm8*	A8h *ib*	Effectue un test de «Et binaire» avec une valeur immédiate de 8 bits sur le registre AL et indique le résultat de la comparaison sur le registre des

		drapeaux.
TEST AX, *imm16*	A9h *iw*	Effectue un test de «Et binaire» avec une valeur immédiate de 16 bits sur le registre AX et indique le résultat de la comparaison sur le registre des drapeaux.
TEST EAX, *imm32*	A9h *id*	Effectue un test de «Et binaire» avec une valeur immédiate de 32 bits sur le registre EAX et indique le résultat de la comparaison sur le registre des drapeaux.
TEST RAX, *imm32*	A9h *id*	Effectue un test de «Et binaire» avec une valeur immédiate de 32 bits sur le registre RAX et indique le résultat de la comparaison sur le registre des drapeaux.
TEST *reg/mem8*, *imm8*	F6h /0 *ib*	Effectue un test de «Et binaire» avec une valeur immédiate de 8 bits sur l'opérande registre ou mémoire 8 bits et indique le résultat de la comparaison sur le registre des drapeaux.
TEST *reg/mem16*, *imm16*	F7h /0 *iw*	Effectue un test de «Et binaire» avec une valeur immédiate de 16 bits sur l'opérande registre ou mémoire 16 bits et indique le résultat de la comparaison sur le registre des drapeaux.
TEST *reg/mem32*, *imm32*	F7h /0 *id*	Effectue un test de «Et binaire» avec une valeur immédiate de 32 bits sur l'opérande registre ou mémoire 32 bits et indique le résultat de la comparaison sur le registre des

		drapeaux.
TEST *reg/mem64*, *imm32*	F7h /0 *id*	Effectue un test de «Et binaire» avec une valeur immédiate de 32 bits sur l'opérande registre ou mémoire 64 bits et indique le résultat de la comparaison sur le registre des drapeaux.
TEST *reg/mem8*, *reg8*	84h /r	Effectue un test de «Et binaire» avec un registre de 8 bits sur l'opérande registre ou mémoire 8 bits et indique le résultat de la comparaison sur le registre des drapeaux.
TEST *reg/mem16*, *reg16*	85h /r	Effectue un test de «Et binaire» avec un registre de 16 bits sur l'opérande registre ou mémoire 16 bits et indique le résultat de la comparaison sur le registre des drapeaux.
TEST *reg/mem32*, *reg32*	85h /r	Effectue un test de «Et binaire» avec un registre de 32 bits sur l'opérande registre ou mémoire 32 bits et indique le résultat de la comparaison sur le registre des drapeaux.
TEST *reg/mem64*, *reg64*	85h /r	Effectue un test de «Et binaire» avec un registre de 64 bits sur l'opérande registre ou mémoire 64 bits et indique le résultat de la comparaison sur le registre des drapeaux.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile non-	X	X	X	Une adresse

canonique)				mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est

				effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction AND
Instruction assembleur 80x86 - Instruction CMP

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 838
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 418
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 241.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 505 à 507.

Assembleur 80x86 — UCOMISD

INTEL Pentium III+

Unordered Compare Scalar Double-Precision Floating-Point Values and Set EFLAGS

Syntaxe

UCOMISD *dest*,*source*

Description

Cette instruction permet d'effectuer une comparaison désordonnée de valeurs réels de double précision dans la partie basse d'un double mot du premier opérande et du deuxième opérande, et fixe les drapeaux ZF, PF et FC dans le registre EFLAGS selon le résultat (non-ordonnée, supérieur à, inférieur ou égal).

Algorithme

```
SI (dest(63..0) UNORDERED source(63..0)) ALORS
  ZF ← 1
  PF ← 1
  CF ← 1
SINON SI (dest(63..0) > source(63..0)) ALORS
  ZF ← 0
  PF ← 0
  CF ← 0
SINON SI (dest(63..0) < source(63..0)) ALORS
  ZF ← 0
  PF ← 0
  CF ← 1
SINON
  ZF ← 1
  PF ← 0
  CF ← 0
FIN SI
```

OF ← 0 SF ← 0 AF ← 0

Mnémonique

Instruction	Opcode	Description
UCOMISD *xmm1, xmm2/m64*	66h 0Fh 2Eh /r	Cette instruction permet d'effectuer une comparaison désordonnée de valeurs réels de double précision dans la partie basse d'un double mot du premier opérande et du deuxième opérande, et fixe les drapeaux ZF, PF et FC dans le registre EFLAGS selon le résultat (non-ordonnée, supérieur à, inférieur ou égal).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 508 à 510.

Assembleur 80x86 — UCOMISS

INTEL Pentium III+

Unordered Compare Scalar Single-Precision Floating-Point Values and Set EFLAGS

Syntaxe

UCOMISS *dest, source*

Description

Cette instruction permet d'effectuer une comparaison désordonnée de valeurs réels de simple précision dans la partie basse d'un double mot du premier opérande et du deuxième opérande, et fixe les drapeaux ZF, PF et FC dans le registre EFLAGS selon le résultat (non-ordonnée, supérieur à, inférieur ou égal).

Algorithme

```
OF ← 0
SF ← 0
AF ← 0

SI (dest(31..0) UNORDERED source(31..0)) ALORS
  ZF ← 1
  PF ← 1
  CF ← 1
SINON SI (dest(31..0) > source(31..0)) ALORS
  ZF ← 0
  PF ← 0
  CF ← 0
SINON SI (dest(31..0) < source(31..0)) ALORS
  ZF ← 0
  PF ← 0
  CF ← 1
SINON
  ZF ← 1
```

```
  PF ← 0
  CF ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
UCOMISS *xmm1,xmm2/m32*	0Fh 2Eh /r	Cette instruction permet d'effectuer une comparaison désordonnée de valeurs réels de simple précision dans la partie basse d'un double mot du premier opérande et du deuxième opérande, et fixe les drapeaux ZF, PF et FC dans le registre EFLAGS selon le résultat (non-ordonnée, supérieur à, inférieur ou égal).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 511 à 514.

Assembleur 80x86

AMD Am5k86 (SSA/5, K5)+

UD

Undefined

Syntaxe

```
UD
```

Description

Cette instruction permet de provoquer l'exécution d'un code indéfinie.

Fonction

```
EXCEPTION #UD
```

Mnémonique

Instruction	Opcode	Description
UD	0Fh FFh	Cette instruction permet de provoquer l'exécution d'un code indéfinie.

Assembleur 80x86 UD2

INTEL Pentium Pro+ *Undefined To*

Syntaxe

```
UD2
```

Description

Cette instruction permet de provoquer l'exécution d'un code indéfinie.

Fonction

```
EXCEPTION #UD
```

Mnémonique

Instruction	Opcode	Description
UD2	0Fh 0Bh	Cette instruction permet de provoquer l'exécution d'un code indéfinie.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 514.

Assembleur 80x86

AMD Am386SXLV, Am386DXLV, AMD 486s, IBM 486SLC2

UMOV

User Move Data

Syntaxe

UMOV *destination, source*

Description

Cette instruction permet de copier un opérande source dans une emplacement mémoire principal.

Algorithme

destination ← source

Mnémonique

Instruction	Opcode	Description
UMOV *r/m8,reg8*	0Fh 10h /r	Cette instruction permet de copier un opérande source dans une emplacement mémoire principal.
UMOV *r/m16,reg16*	0Fh 11h /r	Cette instruction permet de copier un opérande source dans une emplacement mémoire principal.
UMOV *r/m32,reg32*	66h 0Fh 11h /r	Cette instruction permet de copier un opérande source dans une emplacement mémoire principal.
UMOV *reg8,r/m8*	0Fh 12h /r	Cette instruction permet de copier un opérande source dans une emplacement mémoire principal.
UMOV *reg16,r/m16*	0Fh 13h /r	Cette instruction permet de copier un opérande source dans une

		emplacement mémoire principal.
UMOV *reg32,r/m32*	66h 0Fh 13h /r	Cette instruction permet de copier un opérande source dans une emplacement mémoire principal.

Assembleur 80x86 — UNPCKHPD

INTEL Pentium 4 (SSE2)+

Unpack and Interleave High Packed Double-Precision Floating-Point Values

Syntaxe

UNPCKHPD *destination, source*

Description

Cette instruction permet d'effectuer un dépaquetage de la partie haute d'un réel de double précision d'un opérande source et destination et met le résultat dans l'opérande de destination.

Algorithme

destination(0..63) ← *destination*(64..127)
destination(64..127) ← *source*(64..127)

Mnémonique

Instruction	Opcode	Description
UNPCKHPD *xmm1, xmm2/m128*	66h 0Fh 15h /r	Cette instruction permet d'effectuer un dépaquetage de la partie haute d'un réel de double précision d'un opérande source et destination et met le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 515 à 517.

Assembleur 80x86

INTEL Pentium III (KNI/MMX2)+

UNPCKHPS

Unpack and Interleave High Packed Single-Precision Floating-Point Values

Syntaxe

UNPCKHPS *destination*, *source*

Description

Cette instruction permet d'effectuer un dépaquetage de la partie haute d'un réel de simple précision d'un opérande source et destination et met le résultat dans l'opérande de destination.

Algorithme

destination(0..31) ← *destination*(64..95) *destination*(32..63) ← *source*(64..95) *destination*(64..95) ← *destination*(96..127) *destination*(96..127) ← *source*(96..127)

Mnémonique

Instruction	Opcode	Description
UNPCKHPS *xmm1*, *xmm2/m128*	0Fh 15h /r	Cette instruction permet d'effectuer un dépaquetage de la partie haute d'un réel de simple précision d'un opérande source et destination et met le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 518 à 520.

Assembleur 80x86 — UNPCKLPD

INTEL Pentium 4 (SSE2)+

Unpack and Interleave Low Packed Double-Precision Floating-Point Values

Syntaxe

UNPCKLPD *destination*, *source*

Description

Cette instruction permet d'effectuer un dépaquetage de la partie basse d'un réel de double précision d'un opérande source et destination et met le résultat dans l'opérande de destination.

Algorithme

destination(0..63) ← *destination*(0..63)
destination(64..127) ← *source*(0..63)

Mnémonique

Instruction	Opcode	Description
UNPCKLPD *xmm1*, *xmm2/m128*	66h 0Fh 14h /r	Cette instruction permet d'effectuer un dépaquetage de la partie basse d'un réel de double précision d'un opérande source et destination et met le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 521 à 522.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

UNPCKLPS

Unpack Low Packed Single-Precision Floating-Point Values

Syntaxe

UNPCKLPS *dest,source*

Description

Cette instruction permet d'effectuer un dépaquetage de la partie basse d'un réel de simple précision d'une opérande source et destination et met le résultat dans l'opérande de destination.

Algorithme

dest(31..0) ← *dest*(31..0)
dest(63..32) ← *source*(31..0)
dest(95..64) ← *dest*(63..32)
dest(127..96) ← *source*(63..32)

Mnémonique

Instruction	Opcode	Description
UNPCKLPS *xmm1,xmm2/m128*	0Fh 14h /r	Cette instruction permet d'effectuer un dépaquetage de la partie basse d'un réel de simple précision d'une opérande source et destination et met le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 523 à 526.

Assembleur 80x86 VBROADCASTF128

AVX (AMD ou INTEL)

Vector Load with Broadcast Float 128 bits

Syntaxe

VBROADCASTF128 *destination, source*

Description

Cette instruction permet d'effectuer le chargement de valeurs réels de 128 bits d'une opérande source et de se diffuser dans tous les éléments de l'opérande de destination.

Algorithme

```
temp ← source(127..0)
destination(127..0) ← temp
destination(255..128) ← temp
```

Mnémonique

Instruction	Opcode	Description
VBROADCASTF128 *ymm1,m128*	(VEX.256) 66h 0Fh 38h 1Ah /r	Cette instruction permet d'effectuer le chargement de valeurs réels de 128 bits d'une opérande source et de se diffuser dans tous les éléments de l'opérande de destination.

Assembleur 80x86 VBROADCASTSD

AVX (AMD ou INTEL)

Vector Load with Broadcast Double-Precision

Syntaxe

VBROADCASTSD *destination, source*

Description

Cette instruction permet d'effectuer le chargement de valeurs réels de double précision d'une opérande source et de se diffuser dans tous les éléments de l'opérande de destination.

Algorithme

```
temp ← SRC(63..0)
destination(63..0) ← temp
destination(127..64) ← temp
destination(191..128) ← temp
destination(255..192) ← temp
```

Mnémonique

Instruction	Opcode	Description
VBROADCASTSD *ymm1,m64*	(VEX.256) 66h 0Fh 38h 19h /r	Cette instruction permet d'effectuer le chargement de valeurs réels de double précision d'une opérande source et de se diffuser dans tous les éléments de l'opérande de destination.

Assembleur 80x86 — VBROADCASTSS

AVX (AMD ou INTEL)

Vector Load with Broadcast Single-Precision

Syntaxe

VBROADCASTSS *destination, source*

Description

Cette instruction permet d'effectuer le chargement de valeurs réels de simple précision d'une opérande source et de se diffuser dans tous les éléments de l'opérande de destination.

Algorithme

```
SI version 128 bits ALORS
   temp ← source(31..0)
   destination(31..0) ← temp
   destination(63..32) ← temp
   destination(95..64) ← temp
   destination(127..96) ← temp
   destination(255..128) ← 0
SINON
   temp ← source(31..0)
   destination(31..0) ← temp
   destination(63..32) ← temp
   destination(95..64) ← temp
   destination(127..96) ← temp
   destination(159..128) ← temp
   destination(191..160) ← temp
   destination(223..192) ← temp
   destination(255..224) ← temp
FIN SI
```

Mnémonique

Instruction	Opcode	Description

VBROADCASTSS *xmm1,m32*	(VEX.128) 66h 0Fh 38h 18h /r	Cette instruction permet d'effectuer le chargement de valeurs réels de simple précision d'une opérande source et de se diffuser dans tous les éléments de l'opérande de destination.
VBROADCASTSS *ymm1,m32*	(VEX.256) 66h 0Fh 38h 18h /r	Cette instruction permet d'effectuer le chargement de valeurs réels de simple précision d'une opérande source et de se diffuser dans tous les éléments de l'opérande de destination.

Assembleur 80x86 VERR

INTEL 80286+

Verify a Segment for Reading

Syntaxe

VERR *operande*

Description

Cette instruction permet de vérifier si le code ou le segment de données spécifié est en mode lecture à partir du niveau de privilège courant (*CPL*).

Algorithme

```
SI operande(Offset) > (GDTR(Limit) OU (LDTR(Limit)) ALORS
  ZF ← 0
FIN SI
Lecture du descripteur de segment
SI Descripteur de segment(Type de descripteur) = 0 OU (Descripteur de segment(Type)
conforme au code segment) ET (CPL > DPL) OU (RPL > DPL) ALORS
  ZF ← 0
SINON
  SI (segment est lisible)) ALORS
    ZF ← 1
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VERR *r/m16*	0Fh 00h /4	Cette instruction permet de vérifier si le code ou le segment de données

		spécifié est en mode lecture à partir du niveau de privilège courant (*CPL*).

Voir également

Instruction assembleur 80x86 - Instruction ARPL
Instruction assembleur 80x86 - Instruction LAR
Instruction assembleur 80x86 - Instruction LSL
Instruction assembleur 80x86 - Instruction VERW

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, *Edition Intel, Mars 2010, Publication No. 253667-034US, page 526 à 528.*

Assembleur 80x86 — VERW

INTEL 80286+ — *Verify a Segment for Writing*

Syntaxe

VERW *operande*

Description

Cette instruction permet de vérifier si le code ou le segment de données spécifié est en mode écriture à partir du niveau de privilège courant (*CPL*).

Algorithme

```
SI operande(Offset) > (GDTR(Limit) OU (LDTR(Limit)) ALORS
  ZF ← 0
FIN SI
Lecture du descripteur de segment
SI Descripteur de segment(Type de descripteur) = 0 OU (Descripteur de segment(Type)
conforme au code segment) ET (CPL > DPL) OU (RPL > DPL) ALORS
  ZF ← 0
SINON
  SI (segment est écrivable)) ALORS
    ZF ← 1
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VERW *r/m16*	0Fh 00h /5	Cette instruction permet de vérifier si le code ou le segment de données

		spécifié est en mode écriture à partir du niveau de privilège courant (*CPL*).

Voir également

Instruction assembleur 80x86 - Instruction ARPL
Instruction assembleur 80x86 - Instruction LAR
Instruction assembleur 80x86 - Instruction LSL
Instruction assembleur 80x86 - Instruction VERR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 526 à 528.

Assembleur 80x86 — VEXTRACTF128

AVX (AMD ou INTEL) — *Vector Extract packed floating-point values*

Syntaxe

VEXTRACTF128 *destination, source, immediat8*

Description

Cette instruction permet d'effectuer l'extraction de paquet de valeurs réel de 128 bits d'une opérande source dans l'opérande de destination en tenant compte de la position d'extraction avec une opérande de destination.

Algorithme

```
EVALUER CAS immediat8(0)
   CAS 0: destination(127..0) ← source(127..0)
   CAS 1: destination(127..0) ← source(255..128)
FIN EVALUER CAS
destination(255..128) ← 0
```

Mnémonique

Instruction	Opcode	Description
VEXTRACTF128 *xmm1/m128, ymm2, imm8*	(VEX.256) 66h 0Fh 3Ah 19h /r *ib*	Cette instruction permet d'effectuer l'extraction de paquet de valeurs réel de 128 bits d'une opérande source dans l'opérande de destination

		en tenant compte de la position d'extraction avec une opérande de destination.

Assembleur 80x86 — VFMADD132PD

FMA (INTEL)

Fused Multiply-Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFMADD132PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(destination(n+63..n) x
source3(n+63..n) + source2(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  DEST(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADD132PD *xmm0, xmm1,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm2/m128	W1 98h /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFMADD132PD *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 98h /r	Cette instruction permet d'effectuer la

		multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMADD132PS

FMA (INTEL)

Fused Multiply-Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFMADD132PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(destination(n+31..n) x
source3(n+31..n) + source2(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADD132PS *xmm0, xmm1,xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 98h /r	Cette instruction

		permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFMADD132PS *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 98h /r	Cette instruction permet d'effectuer la multiplication de quatre ou

		huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMADD132SD

FMA (INTEL)

Fused Multiply-Add of Scalar Double-Precision Floating-Point Values

Syntaxe

VFMADD132SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(*destination*(63..0) x *source3*(63..0) + *source2*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMADD132SD *xmm0, xmm1,xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 99h /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un

		premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMADD132SS

FMA (INTEL)

Fused Multiply-Add of Scalar Single-Precision Floating-Point Values

Syntaxe

VFMADD132SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(*destination*(31..0) x *source3*(31..0) + *source2*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMADD132SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 99h /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un

		premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMADD213PD

FMA (INTEL)

Fused Multiply-Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFMADD213PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un second opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(source2(n+63..n) x
destination(n+63..n) + source3(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADD213PD *xmm0, xmm1,xmm2/m128*	(VEX.DDS.128.66) 0Fh 38h W1 A8h /r	Cette instruction

		permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un second opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFMADD213PD *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 A8h /r	Cette instruction permet d'effectuer la multiplication

		de deux ou quatre paquets de valeur réel de double précision d'un second opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMADD213PS

FMA (INTEL)

Fused Multiply-Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFMADD213PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un second opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(source2(n+31..n) x
destination(n+31..n) + source3(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  DEST(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADD213PS *xmm0, xmm1,xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 A8h /r	Cette instruction

		permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un second opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFMADD213PS *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 A8h /r	Cette instruction permet d'effectuer la multiplication de quatre ou

		huit paquets de valeur réel de simple précision d'un second opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMADD213SD

FMA (INTEL)

Fused Multiply-Add of Scalar Double-Precision Floating-Point Values

Syntaxe

VFMADD213SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un second opérande source avec la partie basse de paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(*source2*(63..0) x *destination*(63..0) + *source3*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMADD213SD *xmm0, xmm1,xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 A9h /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un

		second opérande source avec la partie basse de paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMADD213SS

Fused Multiply-Add of Scalar Single-Precision Floating-Point Values

Syntaxe

VFMADD213SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(*source2*(31..0) x *destination*(31..0) + *source3*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMADD213SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 A9h /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMADD231PD

FMA (INTEL)

Fused Multiply-Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFMADD231PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un second opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(source2(n+63..n) x
source3(n+63..n) + destination(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADD231PD *xmm0, xmm1,xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 B8h /r	Cette instruction

		permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un second opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFMADD231PD *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 B8h /r	Cette instruction permet d'effectuer la multiplication

		de deux ou quatre paquets de valeur réel de double précision d'un second opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMADD231PS

FMA (INTEL)

Fused Multiply-Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFMADD231PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un second opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
   MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(source2(n+31..n) x
source3(n+31..n) + destination(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  DEST(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADD231PS *xmm0, xmm1,xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 B8h /r	Cette instruction

		permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un second opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFMADD231PS *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 B8h /r	Cette instruction permet d'effectuer la multiplication de quatre ou

		huit paquets de valeur réel de simple précision d'un second opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMADD231SD

FMA (INTEL)

Fused Multiply-Add of Scalar Double-Precision Floating-Point Values

Syntaxe

VFMADD231SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un second opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(*source2*(63..0) x *source3*(63..0) + *destination*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMADD231SD *xmm0, xmm1,xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 B9h /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un

		second opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMADD231SS

FMA (INTEL)

Fused Multiply-Add of Scalar Single-Precision Floating-Point Values

Syntaxe

VFMADD231SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(*source2*(31..0) x *source3*(63..0) + *destination*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMADD231SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 B9h /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un

		deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMADDSUB132PD

FMA (INTEL)

Fused Multiply-Alternating Add/Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFMADDSUB132PD *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

SI VEX.128 **ALORS**
  destination(63..0) ← RoundFPControl_MXCSR(*destination*(63..0) x *source3*(63..0) - *source2*(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(*destination*(127..64) x *source3*(127..64) + *source2*(127..64))
  destination(255..128) ← 0
SINON SI VEX.256 **ALORS**
  destination(63..0) ← RoundFPControl_MXCSR(*destination*(63..0) x *source3*(63..0) - *source2*(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(*destination*(127..64) x *source3*(127..64) + *source2*(127..64))
  destination(191..128) ← RoundFPControl_MXCSR(*destination*(191..128) x *source3*(191..128) - *source2*(191..128))
  destination(255..192) ← RoundFPControl_MXCSR(*destination*(255..192) x *source3*(255..192) + *source2*(255..192))
FIN SI

Mnémonique

Instruction	Opcode	Description
VFMADDSUB132PD *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 96h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination

		(premier opérande source).
VFMADDSUB132PD *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 96h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans

		un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMADDSUB132PS

Multiply-Alternating Add/Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFMADDSUB132PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL - 1
  n ← 64 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(destination(n+31..n) x
source3(n+31..n) - source2(n+31..n))
  destination(n+63..n+32) ← RoundFPControl_MXCSR(destination(n+63..n+32) x
source3(n+63..n+32) + source2(n+63..n+32))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADDSUB132PS *xmm0,xmm1*, xmm2/m128	(VEX.DDS.128) 66h 0Fh 38h W0 96h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier

		opérande source).
VFMADDSUB132PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38 W0 96h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination

		(premier opérande source).

Assembleur 80x86 VFMADDSUB213PD

FMA (INTEL)

Fused Multiply-Alternating Add/Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFMADDSUB213PD *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

SI VEX.128 **ALORS**
  destination(63..0) ← RoundFPControl_MXCSR(*source2*(63..0) x *destination*(63..0) - *source3*(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(*source2*(127..64) x *destination*(127..64) + *source3*(127..64))
  destination(255..128) ← 0
SINON SI VEX.256 **ALORS**
  destination(63..0) ← RoundFPControl_MXCSR(*source2*(63..0) x *destination*(63..0) - *source3*(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(*source2*(127..64) x *destination*(127..64) + *source3*(127..64))
  destination(191..128) ← RoundFPControl_MXCSR(*source2*(191..128) x *destination*(191..128) - *source3*(191..128))
  destination(255..192) ← RoundFPControl_MXCSR(*source2*(255..192) x *destination*(255..192) + *source3*(255..192)
FIN SI

Mnémonique

Instruction	Opcode	Description
VFMADDSUB213PD *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 A6h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination

		(premier opérande source).
VFMADDSUB213PD *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 A6h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans

		un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMADDSUB213PS

Multiply-Alternating Add/Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFMADDSUB213PS *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxème opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL -1
  n ← 64 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(source2(n+31..n) x
destination(n+31..n) - source3(n+31..n))
  destination(n+63..n+32) ← RoundFPControl_MXCSR(source2(n+63..n+32) x
destination(n+63..n+32) + source3(n+63..n+32))
FIN BOUCLE POUR
SI VEX.128 ALORS
  DEST(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADDSUB213PS *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 A6h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxème opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier

		opérande source).
VFMADDSUB213PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 A6h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxème opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination

		(premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMADDSUB231PD

Fused Multiply-Alternating Add/Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFMADDSUB231PD *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

SI VEX.128 **ALORS**
  destination(63..0) ← RoundFPControl_MXCSR(*source2*(63..0) x *source3*(63..0) - *destination*(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(*source2*(127..64) x *source3*(127..64) + *destination*(127..64))
  destination(255..128) ← 0
SINON SI VEX.256 **ALORS**
  destination(63..0) ← RoundFPControl_MXCSR(*source2*(63..0) x *source3*(63..0) - *destination*(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(*source2*(127..64) x *source3*(127..64) + *destination*(127..64))
  destination(191..128) ← RoundFPControl_MXCSR(*source2*(191..128) x *source3*(191..128) - *destination*(191..128))
  destination(255..192) ← RoundFPControl_MXCSR(*source2*(255..192) x *source3*(255..192) + *destination*(255..192))
FIN SI

Mnémonique

Instruction	Opcode	Description
VFMADDSUB231PD *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 B6h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier

		opérande source).
VFMADDSUB231PD *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 B6h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination

		(premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMADDSUB231PS

Multiply-Alternating Add/Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFMADDSUB231PS *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
POUR BOUCLE i ← 0 JUSQU'A MAXVL -1
  n ← 64 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(source2(n+31..n) x
source3(n+31..n) - destination(n+31..n))
  destination(n+63..n+32) ← RoundFPControl_MXCSR(source2(n+63..n+32) x
source3(n+63..n+32) + destination(n+63..n+32))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMADDSUB231PS *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 B6h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande

		source).
VFMADDSUB231PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 B6h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire impaire et soustrait la précision intermédiaire paire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande

		source).

Assembleur 80x86 VFMSUBADD132PD

FMA (INTEL)

Fused Multiply-Alternating Subtract/Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFMSUBADD132PD *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

SI VEX.128 **ALORS**
  destination(63..0) ← RoundFPControl_MXCSR(*destination*(63..0) x *source3*(63..0) + *source2*(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(*destination*(127..64) x *source3*(127..64) - *source2*(127..64))
  destination(255..128) ← 0
SINON SI VEX.256 **ALORS**
  destination(63..0) ← RoundFPControl_MXCSR(*destination*(63..0) x *source3*(63..0) + *source2*(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(*destination*(127..64) x *source3*(127..64) - *source2*(127..64))
  destination(191..128) ← RoundFPControl_MXCSR(*destination*(191..128) x *source3*(191..128) + *source2*(191..128))
  destination(255..192) ← RoundFPControl_MXCSR(*destination*(255..192) x *source3*(255..192) - *source2*(255..192))
FIN SI

Mnémonique

Instruction	Opcode	Description
VFMSUBADD132PD *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 97h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de

		destination (premier opérande source).
VFMSUBADD132PD *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 97h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de

		double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMSUBADD132PS

Fused Multiply-Alternating Subtract/Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFMSUBADD132PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL -1
  n ← 64 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(destination(n+31..n) x
source3(n+31..n) + source2(n+31..n))
  destination(n+63..n+32) ← RoundFPControl_MXCSR(destination(n+63..n+32) x
source3(n+63..n+32) - source2(n+63..n+32))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUBADD132PS *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 97h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination

		(premier opérande source).
VFMSUBADD132PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 97h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans

		un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMSUBADD213PD

Fused Multiply-Alternating Subtract/Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFMSUBADD213PD *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  destination(63..0) ← RoundFPControl_MXCSR(source2(63:0) x destination(63..0) + source3(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(source2(127..64) x destination(127..64) - source3(127..64))
  destination(255..128) ← 0
SINON SI VEX.256 ALORS
  destination(63..0) ← RoundFPControl_MXCSR(source2(63..0) x destination(63..0) + source3(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(source2(127..64) x destination(127..64) - source3(127..64))
  destination(191..128) ← RoundFPControl_MXCSR(source2(191..128) x destination(191..128) + source3(191..128))
  destination(255..192) ← RoundFPControl_MXCSR(source2(255..192) x destination(255..192) - source3(255..192))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUBADD213PD *xmm0,xmm1, xmm2/m128*	VEX.DDS.128.66.0F38.W1 A7 /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de

		destination (premier opérande source).
VFMSUBADD213PD *ymm0,ymm1, ymm2/m256*	VEX.DDS.256.66.0F38.W1 A7 /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de

		double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMSUBADD213PS

Fused Multiply-Alternating Subtract/Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFMSUBADD213PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL -1
  n ← 64 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(source2(n+31..n) x
destination(n+31..n) + source3(n+31..n))
  destination(n+63..n+32) ← RoundFPControl_MXCSR(source2(n+63..n+32) x
destination(n+63..n+32) - source3(n+63..n+32))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUBADD213PS *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 A7h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination

		(premier opérande source).
VFMSUBADD213PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 A7h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans

		un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMSUBADD231PD

Fused Multiply-Alternating Subtract/Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFMSUBADD231PD *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  destination(63..0) ← RoundFPControl_MXCSR(source2(63..0) x source3(63..0) + destination(63..0))
  destination(127..64) ← RoundFPControl_MXCSR(source2(127..64) x source3(127..64) - destination(127..64))
  destination(255..128) ← 0
SINON SI VEX.256 ALORS
  destination(63..0) ← RoundFPControl_MXCSR(source2(63..0) x source3(63..0) + destination(63..0)])
  destination(127..64) ← RoundFPControl_MXCSR(source2(127..64) x source3(127..64) - destination(127..64))
  destination(191..128) ← RoundFPControl_MXCSR(source2(191..128)x source3(191..128) + destination(191..128))
  destination(255..192) ← RoundFPControl_MXCSR(source2(255..192) x source3(255..192) - destination(255..192))
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUBADD231PD *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 B7h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination

		(premier opérande source).
VFMSUBADD231PD *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 B7h /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans

		un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFMSUBADD231PS

Fused Multiply-Alternating Subtract/Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFMSUBADD231PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
POUR BOUCLE i ← 0 JUSQU'A MAXVL -1
  n ← 64 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(SRC2(n+31..n) x SRC3(n+31..n) + destination(n+31..n))
  destination(n+63..n+32) ← RoundFPControl_MXCSR(SRC2(n+63..n+32) x SRC3(n+63..n+32) - destination(n+63..n+32))
FIN SI
SI VEX.128 ALORS
  DEST(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUBADD231PS *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 B7h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier

		opérande source).
VFMSUBADD231PS *ymm0,ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 B7h /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire impaire et ajoute la précision intermédiaire paire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination

		(premier opérande source).

Assembleur 80x86 — VFMSUB132PD

FMA (INTEL)

Fused Multiply-Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFMSUB132PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(destination(n+63..n) x
source3(n+63..n) - source2(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUB132PD *xmm0,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm1,xmm2/m128	W1 9Ah /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFMSUB132PD *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 9Ah /r	Cette instruction permet

		d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMSUB132PS

FMA (INTEL)

Fused Multiply-Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFMSUB132PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(destination(n+31..n) x
source3(n+31..n) - source2(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUB132PS *xmm0,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm1,xmm2/m128	W0 9Ah /r	instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFMSUB132PS *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 9Ah /r	Cette instruction permet d'effectuer la

		multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMSUB132SD

FMA (INTEL)

Fused Multiply-Subtract of Scalar Double-Precision Floating-Point Values

Syntaxe

VFMSUB132SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(*destination*(63..0) x *source3*(63..0) - *source2*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMSUB132SD *xmm0, xmm1,xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 9Bh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un

		premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMSUB132SS

FMA (INTEL)

Fused Multiply-Subtract of Scalar Single-Precision Floating-Point Values

Syntaxe

VFMSUB132SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(*destination*(31..0) x *source3*(31..0) - *source2*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMSUB132SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 9Bh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un

		premier opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMSUB213PD

FMA (INTEL)

Fused Multiply-Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFMSUB213PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(source2(n+63..n) x
destination(n+63..n) - source3(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUB213PD *xmm0,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm1,xmm2/m128	W1 AAh /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFMSUB213PD *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 AAh /r	Cette instruction permet

		d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMSUB213PS

FMA (INTEL)

Fused Multiply-Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFMSUB213PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(source2(n+31..n) x
destination(n+31..n) - source3(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUB213PS *xmm0,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm1,xmm2/m128	W0 AAh /r	instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFMSUB213PS *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 AAh /r	Cette instruction permet d'effectuer la

		multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMSUB213SD

FMA (INTEL)

Fused Multiply-Subtract of Scalar Double-Precision Floating-Point Values

Syntaxe

VFMSUB213SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(*source2*(63..0) x *destination*(63..0) - *source3*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMSUB213SD *xmm0, xmm1,xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 ABh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un

		deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMSUB213SS

FMA (INTEL)

Fused Multiply-Subtract of Scalar Single-Precision Floating-Point Values

Syntaxe

VFMSUB213SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(*source2*(31..0) x *destination*(31..0) - *source3*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMSUB213SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 ABh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un

		deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMSUB231PD

FMA (INTEL)

Fused Multiply-Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFMSUB231PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(SRC2(n+63..n) x SRC3(n+63..n) - destination(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUB231PD *xmm0,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm1,xmm2/m128	W1 BAh /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFMSUB231PD *ymm0, ymm1,ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 BAh /r	Cette instruction permet

		d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMSUB231PS

FMA (INTEL)

Fused Multiply-Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFMSUB231PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(source2(n+31..n) x
source3(n+31..n) - destination(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFMSUB231PS *xmm0,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm1,xmm2/m128	W0 BAh /r	instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFMSUB231PS *ymm0, ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 BAh /r	Cette instruction permet d'effectuer la

		multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFMSUB231SD

FMA (INTEL)

Fused Multiply-Subtract of Scalar Double-Precision Floating-Point Values

Syntaxe

VFMSUB231SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(*source2*(63..0) x *source3*(63..0) - *destination*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMSUB231SD *xmm0, xmm1,xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 BBh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un

		deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFMSUB231SS

FMA (INTEL)

Fused Multiply-Subtract of Scalar Single-Precision Floating-Point Values

Syntaxe

VFMSUB231SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(*source2*(31..0) x *source3*(63..0) - *destination*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFMSUB231SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 BBh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un

		deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD132PD

FMA (INTEL)

Fused Negative Multiply-Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFNMADD132PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(-(destination(n+63..n) x
source3(n+63..n)) + source2(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFNMADD132PD *xmm0,xmm1,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm2/m128	W1 9Ch /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFNMADD132PD *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 9Ch /r	Cette instruction permet

		d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD132PS

FMA (INTEL)

Fused Negative Multiply-Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFNMADD132PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(- (destination(n+31..n) x
source3(n+31..n)) + source2(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFNMADD132PS *xmm0,xmm1,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm2/m128	W0 9Ch /r	instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFNMADD132PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 9Ch /r	Cette instruction permet d'effectuer la

		multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMADD132SD

FMA (INTEL)

Fused Negative Multiply-Add of Scalar Double-Precision Floating-Point Values

Syntaxe

VFNMADD132SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(- (*destination*(63..0) x *source3*(63..0)) + *source2*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMADD132SD *xmm0,xmm1, xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 9Dh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double

		précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD132SS

FMA (INTEL)

Fused Negative Multiply-Add of Scalar Single-Precision Floating-Point Values

Syntaxe

VFNMADD132SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(- (*destination*(31..0) x *source3*(31..0)) + *source2*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMADD132SS *xmm0,xmm1, xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 9Dh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple

		précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD213PD

FMA (INTEL)

Fused Negative Multiply-Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFNMADD213PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(-(source2(n+63..n) x
destination(n+63..n)) + source3(n+63..n))
FIN SI
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFNMADD213PD *xmm0,xmm1,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm2/m128	W1 ACh /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFNMADD213PD *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 ACh /r	Cette instruction permet

		d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD213PS

FMA (INTEL)

Fused Negative Multiply-Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFNMADD213PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(- (source2(n+31..n) x
destination(n+31..n)) + source3(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  DEST(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFNMADD213PS *xmm0,xmm1,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm2/m128	W0 ACh /r	instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFNMADD213PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 ACh /r	Cette instruction permet d'effectuer la

		multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMADD213SD

FMA (INTEL)

Fused Negative Multiply-Add of Scalar Double-Precision Floating-Point Values

Syntaxe

VFNMADD213SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(- (*source2*(63..0) x *destination*(63..0)) + *source3*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMADD213SD *xmm0,xmm1, xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 ADh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD213SS

FMA (INTEL)

Fused Negative Multiply-Add of Scalar Single-Precision Floating-Point Values

Syntaxe

VFNMADD213SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(- (*source2*(31..0) x *destination*(31..0)) + *source3*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMADD213SS *xmm0,xmm1, xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 ADh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du premier opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD231PD

FMA (INTEL)

Fused Negative Multiply-Add of Packed Double-Precision Floating-Point Values

Syntaxe

VFNMADD231PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR(-(source2(n+63..n) x
source3(n+63..n)) + destination(n+63..n))
FIN SI
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFNMADD231PD *xmm0,xmm1,*	VEX.DDS.128.66.0F38.W1 BC /r	Cette

xmm2/m128		instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFNMADD231PD *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W1 BCh /r	Cette instruction permet

		d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD231PS

FMA (INTEL)

Fused Negative Multiply-Add of Packed Single-Precision Floating-Point Values

Syntaxe

VFNMADD231PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR(- (source2(n+31..n) x
source3(n+31..n)) + destination(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VFNMADD231PS *xmm0,xmm1,*	(VEX.DDS.128) 66h 0Fh 38h	Cette

xmm2/m128	W0 BCh /r	instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFNMADD231PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 BCh /r	Cette instruction permet d'effectuer la

		multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMADD231SD

FMA (INTEL)

Fused Negative Multiply-Add of Scalar Double-Precision Floating-Point Values

Syntaxe

VFNMADD231SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(- (*source2*(63..0) x *destination*(63..0)) + *source3*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMADD231SD *xmm0,xmm1, xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 BDh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 — VFNMADD231SS

FMA (INTEL)

Fused Negative Multiply-Add of Scalar Single-Precision Floating-Point Values

Syntaxe

VFNMADD231SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(- (*source2*(31..0) x *source3*(63..0)) + *destination*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMADD231SS *xmm0,xmm1, xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 9Dh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, ajoute la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFNMSUB132PD

Fused Negative Multiply-Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFNMSUB132PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR( - (destination(n+63..n) x
source3(n+63..n)) - source2(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description

VFNMSUB132PD *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 9Eh /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFNMSUB132PD *ymm0,ymm1,*	(VEX.DDS.256) 66h 0Fh 38h	Cette

ymm2/m256	W1 9Eh /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un premier opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFNMSUB132PS

Fused Negative Multiply-Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFNMSUB132PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR( - (destination(n+31..n) x
source3(n+31..n)) - source2(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description

VFNMSUB132PS *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 9Eh /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFNMSUB132PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 9Eh /r	Cette instruction

		permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un premier opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMSUB132SD

FMA (INTEL)

Fused Negative Multiply-Subtract of Scalar Double-Precision Floating-Point Values

Syntaxe

VFNMSUB132SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(- (*destination*(63..0) x *source3*(63..0)) - *source2*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMSUB132SD *xmm0,xmm1, xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 9Fh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double

		précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMSUB132SS

FMA (INTEL)

Fused Negative Multiply-Subtract of Scalar Single-Precision Floating-Point Values

Syntaxe

VFNMSUB132SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(- (*destination*(31..0) x *source3*(31..0)) - *source2*(31..0))
destination(127..32) ← *destination*(127..32)
destination(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMSUB132SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 9Fh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple

		précision d'un premier opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le deuxième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFNMSUB213PD

Fused Negative Multiply-Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFNMSUB213PD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR( - (source2(n+63..n) x
destination(n+63..n)) - source3(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description

VFNMSUB213PD *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 AEh /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFNMSUB213PD *ymm0,ymm1,*	(VEX.DDS.256) 66h 0Fh 38h	Cette

ymm2/m256	W1 AEh /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFNMSUB213PS

Fused Negative Multiply-Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFNMSUB213PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR( - (source2(n+31..n) x
destination(n+31..n)) - source3(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description

VFNMSUB213PS *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 AEh /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFNMSUB213PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 AEh /r	Cette instruction

		permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMSUB213SD

FMA (INTEL)

Fused Negative Multiply-Subtract of Scalar Double-Precision Floating-Point Values

Syntaxe

VFNMSUB213SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(- (*source2*(63..0) x *destination*(63..0)) - *source3*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMSUB132SD *xmm0,xmm1, xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 9Fh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMSUB213SS

FMA (INTEL)

Fused Negative Multiply-Subtract of Scalar Single-Precision Floating-Point Values

Syntaxe

VFNMSUB213SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(- (*source2*(31..0) x *destination*(31..0)) - *source3*(31..0)) *destination*(127..32) ← *destination*(127..32) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMSUB213SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 AFh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du premier opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le troisième opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFNMSUB231PD

Fused Negative Multiply-Subtract of Packed Double-Precision Floating-Point Values

Syntaxe

VFNMSUB231PD *destination*, *source2*, *source3*

Description

Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 2
SINON SI VEX.256 ALORS
  MAXVL ← 4
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 64 x i
  destination(n+63..n) ← RoundFPControl_MXCSR( - (source2(n+63..n) x
source3(n+63..n)) - destination(n+63..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description

VFNMSUB231PD *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W1 BEh /r	Cette instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).
VFNMSUB231PD *ymm0,ymm1,*	(VEX.DDS.256) 66h 0Fh 38h	Cette

ymm2/m256	W1 BEh /r	instruction permet d'effectuer la multiplication de deux ou quatre paquets de valeur réel de double précision d'un deuxième opérande source avec les deux ou quatres paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de deux ou quatres paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de deux ou quatre paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86

FMA (INTEL)

VFNMSUB231PS

Fused Negative Multiply-Subtract of Packed Single-Precision Floating-Point Values

Syntaxe

VFNMSUB231PS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

```
SI VEX.128 ALORS
  MAXVL ← 4
SINON SI VEX.256 ALORS
  MAXVL ← 8
FIN SI
BOUCLE POUR i ← 0 JUSQU'A MAXVL-1
  n ← 32 x i
  destination(n+31..n) ← RoundFPControl_MXCSR( - (source2(n+31..n) x
source3(n+31..n)) - destination(n+31..n))
FIN BOUCLE POUR
SI VEX.128 ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description

VFNMSUB231PS *xmm0,xmm1, xmm2/m128*	(VEX.DDS.128) 66h 0Fh 38h W0 BEh /r	Cette instruction permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).
VFNMSUB231PS *ymm0,ymm1, ymm2/m256*	(VEX.DDS.256) 66h 0Fh 38h W0 BEh /r	Cette instruction

		permet d'effectuer la multiplication de quatre ou huit paquets de valeur réel de simple précision d'un deuxième opérande source avec les quatre ou huit paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de quatre ou huit paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de quatre ou huit paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMSUB231SD

FMA (INTEL)

Fused Negative Multiply-Subtract of Scalar Double-Precision Floating-Point Values

Syntaxe

VFNMSUB231SD *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Algorithme

destination(63..0) ← RoundFPControl_MXCSR(- (*source2*(63..0) x *source3*(63..0)) - *destination*(63..0)) *destination*(127..64) ← *destination*(127..64) *destination*(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMSUB213SD *xmm0,xmm1, xmm2/m64*	(VEX.DDS.128) 66h 0Fh 38h W1 AFh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de double

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de double précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de double précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VFNMSUB231SS

FMA (INTEL)

Fused Negative Multiply-Subtract of Scalar Single-Precision Floating-Point Values

Syntaxe

VFNMSUB231SS *destination, source2, source3*

Description

Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Algorithme

destination(31..0) ← RoundFPControl_MXCSR(- (*source2*(31..0) x *source3*(63..0)) - *destination*(31..0))
destination(127..32) ← *destination*(127..32)
destination(255..128) ← 0

Mnémonique

Instruction	Opcode	Description
VFNMSUB231SS *xmm0, xmm1,xmm2/m32*	(VEX.DDS.128) 66h 0Fh 38h W0 BFh /r	Cette instruction permet d'effectuer la multiplication de la partie basse de paquets de valeur réel de simple

		précision d'un deuxième opérande source avec la partie basse de paquets de valeurs réels de simple précision du troisième opérande source, soustrait la précision infinie intermédiaire négative au résultat de la partie basse de paquets dans le premier opérande source, effectue l'arrondissement et entrepose le résultat de paquets de valeurs réel de simple précision dans un opérande de destination (premier opérande source).

Assembleur 80x86 VINSERTF128

AVX (AMD ou INTEL)

Vector Insert packed floating-point values

Syntaxe

VINSERTF128 *destination, source1, source2, immediat8*

Description

Cette instruction permet d'effectuer l'extraction de paquet de valeurs réel de 128 bits d'une opérande source dans l'opérande de destination en tenant compte de la position d'extraction avec une opérande de destination.

Algorithme

```
TEMP(255..0) ← source1(255..0)
EVALUER CAS immediat8(0)
   CAS 0: TEMP(127..0) ← source2(127..0)
   CAS 1: TEMP(255..128) ← source2(127..0)
FIN EVALUER CAS
destination ← TEMP
```

Mnémonique

Instruction	Opcode	Description
VINSERTF128 *ymm1, ymm2,xmm3/m128, imm8*	(VEX.NDS.256) 66h 0Fh 3Ah 18h /r *ib*	Cette instruction permet d'effectuer l'extraction de paquet de valeurs réel de 128 bits d'une opérande source dans l'opérande de

		destination en tenant compte de la position d'extraction avec une opérande de destination.

Assembleur 80x86 — VLDMXCSR

AVX (AMD ou INTEL) — *Vector Load MXCSR Register*

Syntaxe

VLDMXCSR *source*

Description

Cette instruction permet d'effectuer le chargement du mot de contrôle (MXCSR) d'une opérande mémoire 32 bits pour l'*AVX*.

Algorithme

MXCSR ← *source*

Mnémonique

Instruction	Opcode	Description
VLDMXCSR *m32*	(VEX.128) 0Fh AEh /2	Cette instruction permet d'effectuer le chargement du mot de contrôle (MXCSR) d'une opérande mémoire 32 bits pour l'*AVX*.

Assembleur 80x86 — VMASKMOVPD

AVX (AMD ou INTEL)

Vector Mask Move Packed Double precision

Syntaxe

VMASKMOVPD *destination, source1, source2*

Description

Cette instruction permet d'effectuer un chargement conditionnel des valeurs réels de double précision de paquets d'une opérande de 128 ou 256 bits avec le masque de destination et entrepose le résultat dans l'opérande de destination.

Algorithme

```
SI charge 256 bits ALORS
  SI source1(63) ALORS
    destination(63..0) ← Load_64(mem)
  SINON
    destination(63..0) ← 0
  FIN SI
  SI source1(127) ALORS
    destination(127..64) ← Load_64(mem + 8)
  SINON
    destination(127..64) ← 0
  FIN SI
  SI source1(191) ALORS
    destination(195..128) ← Load_64(mem + 16)
  SINON
    destination(195..128) ← 0
  FIN SI
  SI source1(255) ALORS
    destination(255..196) ← Load_64(mem + 24)
  SINON
    destination(255..196) ← 0
  FIN SI
SINON SI charge 128 bits ALORS
  SI source1(63) ALORS
    destination(63..0) ← Load_64(mem)
  SINON
```

```
    destination(63..0) ← 0
  FIN SI
  SI source1(127) ALORS
    destination(127..64) ← Load_64(mem + 16)
  SINON
    destination(127..64) ← 0
  FIN SI
  destination(255..128) ← 0
FIN SI
SI entrepose 256 bits ALORS
  SI source1(63) ALORS
    DEST(63..0) ← source2(63..0)
  FIN SI
  SIsource1(127) ALORS
    DEST(127..64) ← source2(127..64)
  FIN SI
  SIsource1(191) ALORS
    DEST(191..128) ← source2(191..128)
  FIN SI
  SIsource1(255) ALORS
    DEST(255..192) ← source2(255..192)
  FIN SI
SINON SI entrepose 128 bits ALORS
  SI source1(63) ALORS
    DEST(63..0) ← source2(63..0)
  FIN SI
  SI source1(127) ALORS
    DEST(127..64) ← source2(127..64)
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMASKMOVPD *xmm1,xmm2, m128*	(VEX.NDS.128) 66h 0Fh 38h 2Dh /r	Cette instruction permet d'effectuer un chargement conditionnel des valeurs réels de double

		précision de paquets d'une opérande de 128 ou 256 bits avec le masque de destination et entrepose le résultat dans l'opérande de destination.
VMASKMOVPD *xmm1,xmm2, m128*	(VEX.NDS.128) 66h 0Fh 38h 2Dh /r	Cette instruction permet d'effectuer un chargement conditionnel des valeurs réels de double précision de paquets d'une opérande de 128 ou 256 bits avec le masque de destination et entrepose le résultat dans l'opérande de destination.
VMASKMOVPD *m128,xmm1, xmm2*	(VEX.NDS.128) 66h 0Fh 38h 2Fh /r	Cette instruction permet d'effectuer un chargement conditionnel

		des valeurs réels de double précision de paquets d'une opérande de 128 ou 256 bits avec le masque de destination et entrepose le résultat dans l'opérande de destination.
VMASKMOVPD *m256,ymm1, ymm2*	(VEX.NDS.256) 66h 0Fh 38h 2Fh /r	Cette instruction permet d'effectuer un chargement conditionnel des valeurs réels de double précision de paquets d'une opérande de 128 ou 256 bits avec le masque de destination et entrepose le résultat dans l'opérande de destination.

Assembleur 80x86 — VMASKMOVPS

AVX (AMD ou INTEL) — *Vector Mask Move Packed Single precision*

Syntaxe

VMASKMOVPS *destination, source1, source2*

Description

Cette instruction permet d'effectuer un chargement conditionnel des valeurs réels de simple précision de paquets d'une opérande de 128 ou 256 bits avec le masque d'opérande et entrepose le résultat dans l'opérande de destination.

Algorithme

```
SI charge 256 bits ALORS
  SI source1(31) ALORS
     destination(31..0) ← Load_32(mem)
  SINON
     destination(31..0) ← 0
  FIN SI
  SI source1(63) ALORS
    destination(63..32) ← Load_32(mem + 4)
  SINON
    destination(63..32) ← 0
  FIN SI
  SI source1(95) ALORS
    destination(95..64) ← Load_32(mem + 8)
  SINON
    destination(95..64) ← 0
  FIN SI
  SI source1(127) ALORS
    destination(127..96) ← Load_32(mem + 12)
  SINON
    destination(127..96) ← 0
  FIN SI
  SI source1(159) ALORS
    destination(159..128) ← Load_32(mem + 16)
  SINON
    destination(159..128) ← 0
```

```
  FIN SI
  SI source1(191) ALORS
    destination(191..160) ← Load_32(mem + 20)
  SINON
    destination(191..160) ← 0
  FIN SI
  SI source1(223) ALORS
    destination(223..192) ← Load_32(mem + 24)
  SINON
    destination(223..192) ← 0
  FIN SI
  SI source1(255) ALORS
    destination(255..224) ← Load_32(mem + 28)
  SINON
    destination(255..224) ← 0
  FIN SI
  SINON SI charge 128 bits ALORS
  SI source1(31) ALORS
    destination(31..0) ← Load_32(mem)
  SINON
    destination(31..0) ← 0
  FIN SI
  SI source1(63) ALORS
    destination(63..32) ← Load_32(mem + 4)
  SINON
    destination(63..32) ← 0
  FIN SI
  SI source1(95) ALORS
    destination(95..64) ← Load_32(mem + 8)
  SINON
    destination(95..64) ← 0
  FIN SI
    destination(127..97) ←
  SI source1(127) ALORS
    destination(95..64) ← Load_32(mem + 12)
  SINON
    destination(95..64) ← 0
  FIN SI
    destination(255..128) ← 0
  FIN SI
SI entrepose 256 bits ALORS
  SI source1(31) ALORS
    destination(31..0) ← source2(31..0)
  FIN SI
  SI source1(63) ALORS
    destination(63..32) ← source2(63..32)
```

```
  FIN SI
  SI source1(95) ALORS
    destination(95..64) ← source2(95..64)
  FIN SI
  SI source1(127) ALORS
    destination(127..96) ← source2(127..96)
  FIN SI
  SI source1(159) ALORS
    destination(159..128) ← source2(159..128)
  FIN SI
  SI source1(191) ALORS
    destination(191..160) ← source2(191..160)
  FIN SI
  SI source1(223) ALORS
    destination(223..192) ← source2(223..192)
  FIN SI
  SI source1(255) ALORS
    destination(255..224) ← source2(255..224)
  FIN SI
SINON SI entrepose 128 bits ALORS
  SI source1(31) ALORS
    destination(31..0) ← source2(31..0)
  FIN SI
  SI source1(63) ALORS
    destination(63..32) ← source2(63..32)
  FIN SI
  SI source1(95) ALORS
    destination(95..64) ← source2(95..64)
  FIN SI
  SI source1(127) ALORS
    destination(127..96) ← source2(127..96)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMASKMOVPS *xmm1,xmm2, m128*	(VEX.NDS.128) 66h 0Fh 38h 2Ch /r	Cette instruction permet d'effectuer un chargement conditionnel des valeurs

		réels de simple précision de paquets d'une opérande de 128 ou 256 bits avec le masque d'opérande et entrepose le résultat dans l'opérande de destination.
VMASKMOVPS *ymm1,ymm2, m256*	(VEX.NDS.256) 66h 0Fh 38h 2Ch /r	Cette instruction permet d'effectuer un chargement conditionnel des valeurs réels de simple précision de paquets d'une opérande de 128 ou 256 bits avec le masque d'opérande et entrepose le résultat dans l'opérande de destination.
VMASKMOVPS *m128,xmm1, xmm2*	(VEX.NDS.128) 66h 0Fh 38h 2Eh /r	Cette instruction permet d'effectuer un

		chargement conditionnel des valeurs réels de simple précision de paquets d'une opérande de 128 ou 256 bits avec le masque d'opérande et entrepose le résultat dans l'opérande de destination.
VMASKMOVPS *m256,ymm1, ymm2*	(VEX.NDS.256) 66h 0Fh 38h 2Eh /r	Cette instruction permet d'effectuer un chargement conditionnel des valeurs réels de simple précision de paquets d'une opérande de 128 ou 256 bits avec le masque d'opérande et entrepose le résultat dans l'opérande de destination.

Assembleur 80x86 VMCALL

INTEL VMX (Virtualisation)

Call to VM Monitor

Syntaxe

```
VMCALL
```

Description

Cette instruction permet de fournir un mécanisme invité logiciel pour appeler un service dans un moniteur *VM*.

Algorithme

```
SI pas dans opération VMX ALORS
  EXCEPTION #UD
SINON mais pas l'opération racine ALORS
  quitte VM
SINON SI (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON dans SMM ou le processeur logique ne support pas le traitement multiple moniteur de
SMI et SMM ou le bit valide dans le IA32_SMM_MONITOR_CTL MSR vaut 0 ALORS
  VMfail (VMCALL exécuté dans une opération racine VMX
SINON SI traitement multiple moniteur de SMI et SMM est actif ALORS
  quitte le SMM VM
SINON SI pointeur courant VMCS n'est pas valide ALORS
  VMfailInvalid
SINON SI lance l'état du VMCS courant est fixé à 1 ALORS
  VMfailValid(VMCALL avec VMCS vaut 1)
SINON SI champs de contrôle de sortie VM n'est pas valide ALORS
  VMfailValid (VMCALL avec champs de contôle de sortie VM)
SINON
  entrer dans SMM
```

```
  lecture de l'identificateur de révision dans MSEG
  SI identificateur de révision ne correspondant pas au processeur supporté ALORS
    laisse SMM
    VMfailValid(VMCALL avec identificateur de révision MSEG incorrect)
  SINON
    lecture des champs de fournitures du moniteur SMM dans MSEG
    SI champs de fournitures est invalide ALORS
      laisse SMM
      VMfailValid(VMCALL avec fourniture de moniteur SMM invalide)
    SINON
      active le traitement multiple moniteur du SMI et SMM
    FIN SI
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMCALL	0Fh 01h C1h	Cette instruction permet de fournir un mécanisme invité logiciel pour appeler un service dans un moniteur *VM*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 577 à 579.

Assembleur 80x86 — VMCLEAR

INTEL VMX (Virtualisation) — *Clear Virtual-Machine Control Structure*

Syntaxe

VMCLEAR *opérande*

Description

Cette instruction permet d'appliquer un *VMCS* à une région *VMCS* à l'adresse physique contenu dans l'opération spécifié.

Algorithme

```
SI (opérande registre) ou (pas dans une opération VMX) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON SI pas dans une opération racine VMX ALORS
  quitte VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON
  addr ← contient une opérande d'adressage mémoire de 64 bits
  SI (addr n'est pas dans l'alignement 4 Ko) ou (processeur supportes l'architecture Intel 64 et addr à tous les bits fixé 1 dans la largeur de l'adresse phyisque) ou (processeur ne support pas l'architecture Intel 64, addr à tous les bits fixé dans le rang de bits de 63 à 32) ALORS
    VMfail(VMCLEAR avec une adresse physique invalide)
  SINON SI addr = pointeur VMXON ALORS
    VMfail(VMCLEAR avec pointeur VMXON)
  SINON
    Fait en sort que les données VMCS sont référencé par l'opérande en mémoire
    initialise les spécification de données dans les régions VMCS
    lance l'état de référencement VMCS par l'opérande ← 0
    SI opérande addr = pointeur courant VMCS ALORS
      pointeur courant VMCS ← FFFFFFFFFFFFFFFFh
```

```
    FIN SI
    succès VM
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMCLEAR *m64*	66h 0Fh C7h /6	Cette instruction permet de copier les données *VMCS* dans une région mémoire *VMCS*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 579 à 581.

Assembleur 80x86 VMLAUNCH

INTEL VMX (Virtualisation)

Launch Virtual Machine

Syntaxe

```
VMLAUNCH
```

Description

Cette instruction permet de gérer l'entrée VM par le VMCS courant, de façon à échoué si l'état de lancement VMCS courant n'est pas effacé. Si l'instruction est un succès, il définit l'état de lancement à relancer.

Algorithme

```
SI (pas dans l'opération VMX operation) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0)
ALORS
  EXCEPTION #UD
SINON SI pas dans une opération racine VMX ALORS
  quitte VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON SI pointeur du VMCS courant n'est pas valide ALORS
  VMfailInvalid
SINON SI événement est verrouillé par MOV SS ALORS
  VMfailValid(entrée VM avec événements bloqué par MOV SS)
SINON SI (état de lancement du VMCS courant n'est pas effacé) ALORS
  VMfailValid(VMLAUNCH avec VMCS non effacé)
SINON
  Vérifie les ajustements des contrôles VMX et la région d'état de l'hôte
  SI ajustement invalide ALORS
    VMfailValid(entrée VM avec champs de contrôle VMX invalide) ou
    VMfailValid(entrée VM avec champs d'état de l'hôte invalide) ou
    VMfailValid(entrée VM avec pointeur d'exécution VMCS invalide)) ou
    VMfailValid(entrée VM avec non-lancement exécutif du VMCS) ou
```

```
    VMfailValid(entrée VM avec pointeur exécutif VMCS pas pointeur VMXON) ou
    VMfailValid(entrée VM avec champ de contrôle d'exécution VM dans l'exécutif VMCS)
    sont appropriés
  SINON
    Attente de chargement de l'état d'invité et PDPTRs sont approprié
    Efface les rangs d'adresse du moniteur
    SI échec de vérification de l'état ou PDPTR ALORS
      échec d'entrée VM
    SINON
      Attente de chargement MSR de l'entrée VM chargé en région MSR
      SI échec ALORS
        échec d'entrée VM
      SINON
        fixe l'état du VMCS à lancé
        SI dans SMM et contrôle d'entrée SMM de l'entrée VM est 0 ALORS
          SI contrôle d'entrée VM a désactiver le traitement de multiple moniteur est à 0 ALORS
            pointeur VMCS de transfèrer SMM ← pointeur VMCS courant
          FIN SI
          SI executive-VMCS pointer is VMX pointer ALORS
            pointeur de VMCS courant ← pointeur de liaison VMCS
          SINON
            pointeur VMCS courant ← pointeur exécutif VMCS
          FIN SI
          laisse SMM
        FIN SI
        succès d'entrée VM
      FIN SI
    FIN SI
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMLAUNCH	0Fh 01h C2h	Cette instruction permet de lancer la gestion de la machine virtuel du *VMCS* courant.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 582 à 583.

Assembleur 80x86 VMLOAD

AMD-V

Load State from VMCB

Syntaxe

```
VMLOAD RAX
```

Description

Cette instruction permet d'effectuer le chargement d'un sous-ensemble d'état de microprocesseur dans un *VMCB* spécifié par une adresse physique contenu dans le registre *RAX*.

Algorithme

```
SI ((MSR_EFER.SVME = 0) OU (PAS PROTECTED_MODE)) ALORS
  EXCEPTION #UD()
FIN SI
SI CPL <> 0 ALORS
  EXCEPTION #GP()
FIN SI
SI RAX contient une adresse physique non supporté ALORS
  EXCEPTION #GP()
FIN SI
Charge un VMCB à l'adresse physique du registre RAX: FS, GS, TR, LDTR (inclus tous les états caché), KernelGsBase, STAR, LSTAR, CSTAR, SFMASK, SYSENTER_CS, SYSENTER_ESP, SYSENTER_EIP
```

Mnémonique

Instruction	Opcode	Description

VMLOAD RAX	0Fh 01h DAh	>Cette instruction permet d'effectuer le chargement additionnel dans le *VMCB*.

Voir également

Instruction assembleur 80x86 - Instruction VMSAVE

Assembleur 80x86 VMMCALL

AMD-V *Call VMM*

Syntaxe

```
VMMCALL
```

Description

Cette instruction permet de fournir un mécanisme invité pour communiquer explicitement avec le *VMM* en générant un *#VMEXIT*.

Mnémonique

Instruction	Opcode	Description
VMMCALL	0Fh 01h D9h	Cette instruction permet de fournir un mécanisme invité pour communiquer explicitement avec le *VMM* en générant un *#VMEXIT*.

Assembleur 80x86 — VMPTRLD

INTEL VMX (Virtualisation)

Load Pointer to Virtual-Machine Control Structure

Syntaxe

VMPTRLD *opérande*

Description

Cette instruction permet de marquer le pointeur VMCS courant valide et charge celui-ci avec une adresse physique dans l'opérande d'instruction.

Algorithme

```
SI (opérande de registre) ou (pas dans une opération VMX) ou (RFLAGS.VM = 1) ou
(IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON SI pas dans la racine d'opération VMX ALORS
  quitte VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON
  addr ← contenu de l'opérande source de 64 bits en mémoire
  SI addr n'est pas dans alignement de 4 Ko ou (processeur supportes l'architecture Intel 64 et
addr a tous les bits de fixé dans la largueur d'adresse phyisque du processeur) ou (processeur ne
support pas l'architecture Intel 64 et addr à tous les bits dans les rangs 63 à 32 de fixé) ALORS
    VMfail(VMPTRLD avec une adresse physique invalide)
  SINON SI addr = pointeur VMXON ALORS
    VMfail(VMPTRLD avec pointeur VMXON)
  SINON
    rev ← emplacement 32 bits de l'adresse physique de addr
    SI rev = identificateur de révision VMCS supporté par le processeur ALORS
      VMfail(VMPTRLD avec un identificateur de révision incorrect VMCS)
    SINON
```

```
        pointeur VMCS courant ← addr
        succès VM
    FIN SI
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMPTRLD *m64*	66h 0Fh C7h /6	Cette instruction permet de marquer le pointeur VMCS courant valide et charge celui-ci avec une adresse physique dans l'opérande d'instruction.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 585 à 587.

Assembleur 80x86 — VMPTRST

INTEL VMX (Virtualisation)

Store Pointer to Virtual-Machine Control Structure

Syntaxe

VMPTRST *opérande*

Description

Cette instruction permet d'entreposer le pointeur VMCS courant dans l'adresse mémoire spécifié.

Algorithme

```
SI (opérande de registre) ou (pas dans opération VMX) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON SI pas dans opération racine VMX ALORS
  quitte VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON
  opérande de destination 64 bits en memory ← pointeur du VMCS courant
  succès VM
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMPTRST *m64*	0Fh C7h /7	Cette instruction permet d'entreposer le pointeur VMCS

		courant dans l'adresse mémoire spécifié.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 588 à 589.

Assembleur 80x86 — VMREAD

INTEL VMX (Virtualisation)

Read Field from Virtual-Machine Control Structure

Syntaxe

VMREAD *destination,source*

Description

Cette instruction permet d'effectuer la lecture du champ spécifié dans le VMCS et l'entrepose dans l'opérande de destination spécifié.

Algorithme

SI (pas dans l'opération VMX) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0) **ALORS**
  EXCEPTION #UD
SINON SI pas dans l'opération racine VMX **ALORS**
  sortir VM
SINON SI CPL > 0 **ALORS**
  EXCEPTION #GP(0)
SINON SI pointeur VMCS courant n'est pas valide **ALORS**
  VMfailInvalid
SINON SI registre d'opérande source ne correspond pas au champ VMCS **ALORS**
  VMfailValid(VMREAD/VMWRITE de/à non-supporté par le composant VMCS)
SINON
  destination ← contenu du champ VMCS indexé par le registre de l'opérande *source*
  succès VM
FIN SI

Mnémonique

Instruction	Opcode	Description

VMREAD *r/m64, r64*	0Fh 78h	Cette instruction permet d'effectuer la lecture du champ spécifié dans le VMCS et l'entrepose dans l'opérande de destination spécifié.
VMREAD *r/m32, r32*	0Fh 78h	Cette instruction permet d'effectuer la lecture du champ spécifié dans le VMCS et l'entrepose dans l'opérande de destination spécifié.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 590 à 592.

Assembleur 80x86 VMRESUME

INTEL VMX (Virtualisation)

Resume Virtual Machine

Syntaxe

```
VMRESUME
```

Description

Cette instruction permet de gérer l'entrée VM par le VMCS courant, de façon à échoué si l'état le lancement VMCS courant n'est pas lancé.

Algorithme

```
SI (pas dans l'opération VMX operation) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0)
ALORS
  EXCEPTION #UD
SINON SI pas dans une opération racine VMX ALORS
  quitte VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON SI pointeur du VMCS courant n'est pas valide ALORS
  VMfailInvalid
SINON SI événement est verrouillé par MOV SS ALORS
  VMfailValid(entrée VM avec événements bloqué par MOV SS)
SINON SI (état de lancement du VMCS courant n'est pas lancé) ALORS
  VMfailValid(VMRESUME avec VMCS non lancé)
SINON
  Vérifie les ajustements des contrôles VMX et la région d'état de l'hôte
  SI ajustement invalide ALORS
    VMfailValid(entrée VM avec champs de contrôle VMX invalide) ou
    VMfailValid(entrée VM avec champs d'état de l'hôte invalide) ou
    VMfailValid(entrée VM avec pointeur d'exécution VMCS invalide)) ou
    VMfailValid(entrée VM avec non-lancement exécutif du VMCS) ou
    VMfailValid(entrée VM avec pointeur exécutif VMCS pas pointeur VMXON) ou
```

```
    VMfailValid(entrée VM avec champ de contrôle d'exécution VM dans l'exécutif VMCS)
    sont appropriés
  SINON
    Attente de chargement de l'état d'invité et PDPTRs sont approprié
    Efface les rangs d'adresse du moniteur
    SI échec de vérification de l'état ou PDPTR ALORS
      échec d'entrée VM
    SINON
      Attente de chargement MSR de l'entrée VM chargé en région MSR
      SI échec ALORS
        échec d'entrée VM
      SINON
        SI dans SMM et contrôle d'entrée SMM de l'entrée VM est 0 ALORS
          SI contrôle d'entrée VM a désactiver le traitement de multiple moniteur est à 0 ALORS
            pointeur VMCS de transfèrer SMM ← pointeur VMCS courant
          FIN SI
          SI executive-VMCS pointer is VMX pointer ALORS
            pointeur de VMCS courant ← pointeur de liaison VMCS
          SINON
            pointeur VMCS courant ← pointeur exécutif VMCS
          FIN SI
          laisse SMM
        FIN SI
        succès d'entrée VM
      FIN SI
    FIN SI
  FIN SI
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMRESUME	0Fh 01h C3h	Cette instruction permet de gérer l'entrée VM par le VMCS courant, de façon à échoué si l'état le lancement VMCS courant n'est pas lancé.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 582 à 583.

Assembleur 80x86 VMRUN

AMD-V

Run Virtual Machine

Syntaxe

```
VMRUN RAX
```

Description

Cette instruction permet de lancer l'exécution d'un flux d'instructions invité.

Algorithme

```
SI ((MSR_EFER.SVME = 0) OU (!PROTECTED_MODE)) ALORS
  EXCEPTION #UD()
FIN SI
SI CPL <> 0 ALORS
  EXCEPTION #GP()
FIN SI
SI RAX contient un adresse physique non supporté ALORS
  EXCEPTION #GP()
FIN SI
SI interception (VMRUN) ALORS
  #VMEXIT (VMRUN)
FIN SI
Mémorise l'adresse VMCB fournit dans RAX pour le prochain #VMEXIT
Sauvegarde l'état de l'hôte à l'adresse mémoire physique indiquer dans le VM_HSAVE_PA MSR:
ES.sel, CS.sel, SS.sel, DS.sel, GDTR.{base,limit}, IDTR.{base,limit}, EFER, CR0, CR4, CR3, RFLAGS,
VMRUN, RIP, RSP, RAX
Charge les informations de contrôle VMCB de l'adresse physique contient dans le registre RAX:
vecteur d'interception, TSC_OFFSET contrôle d'interruption (v_irq, v_intr_*, v_tpr), EVENTINJ
champ, ASID
SI page supporté ALORS
  NP_ENABLE
```

```
  SI NP_ENABLE = 1 ALORS
    nCR3
  FIN SI
FIN SI

Charge l'état d'invité VMCB de l'adresse physique RAX: ES.{base,limit,attr,sel},
CS.{base,limit,attr,sel}, SS.{base,limit,attr,sel}, DS.{base,limit,attr,sel}, GDTR.{base,limit},
IDTR.{base,limit}, EFER, CR0, CR4, CR3, CR2
  SI NP_ENABLE = 1 ALORS
    Charge également gPAT, RFLAGS, RIP, RSP, RAX, DR7, DR6, CPL
  FIN SI
  INTERRUPT_SHADOW

SI LBR virtualisation supporté ALORS
  LBR_VIRTUALIZATION_ENABLE
SI LBR_VIRTUALIZATION_ENABLE = 1 ALORS
  Sauvegarde l'état LBR dans la région de l'hôte sauvegardé: DBGCTL, BR_FROM, BR_TO,
LASTEXCP_FROM, LASTEXCP_TO
  Charge l'état LBR du VMCB: DBGCTL, BR_FROM, BR_TO, LASTEXCP_FROM, LASTEXCP_TO
FIN SI
SI état invité consiste en un échec de vérification ALORS
  #VMEXIT(INVALID)
FIN SI
Exécute une commande entreprosé dans TLB_CONTROL
GIF ← 1
SI EVENTINJ.V ALORS
  EXCEPTION Interruption dans l'invité
SINON
  Saut à la prochaine instruction invité
FIN SI

#VMEXIT:
  GIF ← 0
  Sauvaregarde l'état invité du VMCB: ES.{base,limit,attr,sel}, CS.{base,limit,attr,sel},
SS.{base,limit,attr,sel}, DS.{base,limit,attr,sel}, GDTR.{base,limit}, IDTR.{base,limit}, EFER, CR4,
CR3, CR2, CR0
    SI pagination activé ALORS
      Sauvegarde également gPAT, RFLAGS, RIP, RSP, RAX, DR7, DR6, CPL, INTERRUPT_SHADOW
    FIN SI
  Sauvegarde l'état additionnel et interception les informations: V_IRQ, V_TPR, EXITCODE,
EXITINFO1, EXITINFO2, EXITINTINFO
```

```
  Efface le champ EVENTINJ dans le VMCB
  Prépare pour le mode hôte par effacement interne des bits d'état du microprocesseur: efface l'interception, v_irq, v_intr_masking et tsc_offset, désactive la pagination, ASID ← 0
  Recharge l'état de l'hôte: GDTR.{base,limit}, IDTR.{base,limit}, EFER, CR0, CR0.PE ← 1, CR4, CR3
    SI hôte dams le mode de pagination PAE ALORS
      Recharge l'hôte PDPEs
    FIN SI
    Charge également RFLAGS, RIP, RSP, RAX, DR7 ← tous désactivé, CPL ← 0
    Recharge ES.sel du descripteur de segment GDT
    Recharge CS.sel du descripteur de segment GDT
    Recharge SS.sel du descripteur de segment GDT
    Recharge DS.sel du descripteur de segment GDT
  SI virtualisation LBR supporté ALORS
    LBR_VIRTUALIZATION_ENABLE
    SI LBR_VIRTUALIZATION_ENABLE = 1 ALORS
      Sauvegarde l'état LBR du VMCB: DBGCTL, BR_FROM, BR_TO, LASTEXCP_FROM, LASTEXCP_TO
      Charge l'état LBR state de l'état de la région de sauvegarde: DBGCTL, BR_FROM, BR_TO, LASTEXCP_FROM, LASTEXCP_TO
  SI chargement illégale de l'état de l'hôte OU exception du l'état de l'hôte ALORS
    Redémarrage du microprocesseur
  SINON
    Execute la première instruction de l'hôte suivant l'instruction VMRUN
  FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMRUN RAX	0Fh 01h D8h	Cette instruction permet de lancer l'exécution d'un flux d'instructions invité.

Voir également

Instruction assembleur 80x86 - Instruction VMLOAD
Instruction assembleur 80x86 - Instruction VMSAVE

Assembleur 80x86 VMSAVE

AMD-V

Save State to VMCB

Syntaxe

```
VMSAVE RAX
```

Description

Cette instruction permet d'entreposer un sous-ensemble d'état du microprocesseur dans un *VMCB* spécifié par une adresse phyisque contenu dans le registre *RAX*.

Algorithme

```
SI ((MSR_EFER.SVME = 0) OU (PAS PROTECTED_MODE)) ALORS
  EXCEPTION #UD()
FIN SI
SI CPL <> 0 ALORS
  EXCEPTION #GP
SI RAX contient une adresse physique non supporté ALORS
  EXCEPTION #GP
Charge du VMCB l'adresse physique du registre RAX: FS, GS, TR, LDTR (include dans tous les états), KernelGsBase, STAR, LSTAR, CSTAR, SFMASK, SYSENTER_CS, SYSENTER_ESP, SYSENTER_EIP
```

Mnémonique

Instruction	Opcode	Description
VMSAVE RAX	0Fh 01h DBh	>Cette instruction permet d'entreposer un sous-ensemble d'état du microprocesseur dans un *VMCB* spécifié par une adresse

		phyisque contenu dans le registre *RAX*.

Voir également

Instruction assembleur 80x86 - Instruction SYSRET
Instruction assembleur 80x86 - Instruction SYSENTER
Instruction assembleur 80x86 - Instruction SYSEXIT

Assembleur 80x86 — VMWRITE

INTEL VMX (Virtualisation)

Write Field to Virtual-Machine Control Structure

Syntaxe

VMWRITE *destination,source*

Description

Cette instruction permet d'effectuer l'écriture du champ spécifié du VMCS spécifié par l'opérande source secondaire (registre seulement) en utilisant le contenu de l'opérande source primaire (registre ou mémoire).

Algorithme

```
SI (pas dans l'opération VMX) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON SI pas dans l'opération racine VMX ALORS
  sortir VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON SI pointeur VMCS courant n'est pas valide ALORS
  VMfailInvalid
SINON SI registre d'opérande de destination ne correspond pas à aucun champ VMCS ALORS
  VMfailValid(VMREAD/VMWRITE de/à non-supporté par le composante VMCS)
SINON SI champ VMCS indexé par l'opérande de registre destination est en lecture seulement
ALORS
  VMfailValid(VMWRITE est en lecture seulement dans le composante VMCS)
SINON
  champ VMCS indexé par le registre de l'opérande destination ← source
  succès VM
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMWRITE *r64, r/m64*	0Fh 79h	Cette instruction permet d'effectuer l'écriture du champ spécifié du VMCS spécifié par l'opérande source secondaire (registre seulement) en utilisant le contenu de l'opérande source primaire (registre ou mémoire).
VMWRITE *r32, r/m32*	0Fh 79h	Cette instruction permet d'effectuer l'écriture du champ spécifié du VMCS spécifié par l'opérande source secondaire (registre seulement) en utilisant le contenu de l'opérande source primaire (registre ou mémoire).

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 593 à 594.

Assembleur 80x86 — VMXOFF

INTEL VMX (Virtualisation)

Leave VMX Operation

Syntaxe

```
VMXOFF
```

Description

Cette instruction permet d'utiliser le processeur logique en dehors de l'opération VMX, de débloquer le signal *INIT*, de conditionnellement réactiver le *A20M* et d'effacer tous les rangs d'adresse du moniteur.

Algorithme

```
SI (pas dans l'opération VMX) ou (RFLAGS.VM = 1) ou (IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON SI pas dans l'opération racine VMX ALORS
  sortir VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON SI traitement de multiple moniteur du SMI et SMM est actif ALORS
  VMfail(VMXOFF dans le traitement de multiple moniteur du SMI et SMM)
SINON
  laisse l'opération VMX
  débloque INIT
  SI en dehors des limites de l'opération 2 SMX ALORS
    débloque et active A20M
  FIN SI
  rang d'adresse du moniteur est effacé
  succès VM
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMXOFF	0Fh 01h C4h	Cette instruction permet d'utiliser le processeur logique en dehors de l'opération VMX, de débloquer le signal *INIT*, de conditionnellement réactiver le *A20M* et d'effacer tous les rangs d'adresse du moniteur.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 595 à 596.

Assembleur 80x86 VMXON

INTEL VMX (Virtualisation)

Enter VMX Operation

Syntaxe

VMXON *operande*

Description

Cette instruction permet d'utiliser le processeur logique dans une opération VMX avec le VMCS non courant, le signal INIT bloqué, la désactivation A20M, et en effaçant n'importe quel rang d'adresse du moniteur établit avec l'instruction *MONITOR*.

Algorithme

```
SI (opérande de registre) ou (CR4.VMXE = 0) ou (CR0.PE = 0) ou (RFLAGS.VM = 1) ou
(IA32_EFER.LMA = 1 et CS.L = 0) ALORS
  EXCEPTION #UD
SINON SI pas dans opération VMX ALORS
  SI (CPL > 0) ou (dans mode A20M) ou (les valeurs de CR0 et CR4 ne sont pas supporté dans
l'opération 2 VMX) ou
  (bit 0 (vérouille le bit) de IA32_FEATURE_CONTROL MSR est effacé) ou
  (dans opération 3 SMX et bit 1 de IA32_FEATURE_CONTROL MSR est effacé) ou
  (opération SMX en dehors des limites et bit 2 de IA32_FEATURE_CONTROL MSR est effacé)
ALORS
    EXCEPTION #GP(0)
  SINON
    addr ← contenus de l'opérande source de 64 bits en mémoire
    SI addr n'est pas dans le 4 Ko aligné ou (processeur support l'architecture Intel 64 et
      addr fixe n'importe quel bits dans la largeur de l'adresse phyisuqe VMX) ou
      (processeur ne support pas l'architecture Intel 64 et addr a tous les bits des range 63 à 32
fixé) ALORS
      VMfailInvalid
    SINON
      rev ← emplacement 32 bits à l'adresse physique addr
```

```
      SI rev = identificateur de révision VMCS est supporté par le processeur ALORS
        VMfailInvalid
      SINON
        pointeur VMCS courant ← FFFFFFFFFFFFFFFFh
        entre dans l'opération VMX
        bloque le signal INIT
        bloque et désactive le A20M
        efface le rang d'addresse du moniteur
        succès VM
      FIN SI
    FIN SI
  FIN SI
SINON SI pas dans opération VMX ALORS
  sortir VM
SINON SI CPL > 0 ALORS
  EXCEPTION #GP(0)
SINON
  VMfail(VMXON exécuté dans une opération racine VMX)
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VMXON *m64*	F3h 0Fh C7h /6	Cette instruction permet d'utiliser le processeur logique dans une opération VMX avec le VMCS non courant, le signal INIT bloqué, la désactivation A20M, et en effaçant n'importe quel rang d'adresse du moniteur établit avec l'instruction *MONITOR*.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 597 à 599.

Assembleur 80x86 — VPERM2F128

AVX (AMD ou INTEL)

Vector Permute 128 bits Floating-Point Values

Syntaxe

VPERM2F128 *destination*, *source1*, *source2*

Description

Cette instruction permet d'effectuer la permutation de valeurs réel de 128 bits d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.

Algorithme

```
EVALUER CAS IMM8(1..0)
  CAS 0: destination(127..0) ← source11(127..0)
  CAS 1: destination(127..0) ← source1(255..128)
  CAS 2: destination(127..0) ← source2(127..0)
  CAS 3: destination(127..0) ← source2(255..128)
FIN EVALUER CAS
EVALUER CAS IMM8(5..4)
  CAS 0: destination(255..128) ← source1(127..0)
  CAS 1: destination(255..128) ← source1(255..128)
  CAS 2: destination(255..128) ← source2(127..0)
  CAS 3: destination(255..128) ← source2(255..128)
FIN EVALUER CAS
SI imm8(3) ALORS
  destination(127..0) ← 0
FIN SI
SI imm8(7) ALORS
  destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description

VPERMILPS *xmm1, xmm2,xmm3/m128*	(VEX.NDS.256) 66h 0Fh 3Ah 06h /r *ib*	Cette instruction permet d'effectuer la permutation de valeurs réel de 128 bits d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.

Assembleur 80x86 — VPERMILPD

AVX (AMD ou INTEL)

Vector Permute Double-Precision Floating-Point Values

Syntaxe

VPERMILPD *destination, source1, source2*

Description

Cette instruction permet d'effectuer la permutation de valeurs réels de double précision d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.

Algorithme

SI version 256 bits immédiate **ALORS**
  SI imm8(0) = 0 **ALORS**
    destination(63..0) ← *source1*(63..0)
  FIN SI
  SI imm8(0) = 1 **ALORS**
    destination(63..0) ← *source1*(127..64)
  FIN SI
  SI imm8(1) = 0 **ALORS**
    destination(127..64) ← *source1*(63..0)
  FIN SI
  SI imm8(1) = 1 **ALORS**
    destination(127..64) ← *source1*(127..64)
  FIN SI
  SI imm8(2) = 0 **ALORS**
    destination(191..128) ← *source1*(191..128)
  FIN SI
  SI imm8(2) = 1 **ALORS**
    destination(191..128) ← *source1*(255..192)
  FIN SI
  SI imm8(3) = 0 **ALORS**
    destination(255..192) ← *source1*(191..128)
  FIN SI
  SI imm8(3) = 1 **ALORS**
    destination(255..192) ← *source1*(255..192)
  FIN SI

SINON SI version 128 bits variable **ALORS**
  SI imm8(0) = 0 **ALORS**
    destination(63..0) ← *source1*(63..0)
  FIN SI
  SI imm8(0) = 1 **ALORS**
    destination(63..0) ← *source1*(127..64)
  FIN SI
  SI imm8(1) = 0 **ALORS**
    destination(127..64) ← *source1*(63..0)
  FIN SI
  SI imm8(1) = 1 **ALORS**
    destination(127..64) ← *source1*(127..64)
  FIN SI
  destination(255..128) ← 0
SINON SI version 256 bits variable **ALORS**
  SI *source2*(1) = 0 **ALORS**
    destination(63..0) ← *source1*(63..0)
  FIN SI
  SI *source2*(1) = 1 **ALORS**
    destination(63..0) ← *source1*(127..64)
  FIN SI
  SI *source2*(65) = 0 **ALORS**
    destination(127..64) ← *source1*(63..0)
  FIN SI
  SI *source2*(65) = 1 **ALORS**
    destination(127..64) ← *source1*(127..64)
  FIN SI
  SI *source2*(129) = 0 **ALORS**
    destination(191..128) ← *source1*(191..128)
  FIN SI
  SI *source2*(129) = 1 **ALORS**
    destination(191..128) ← *source1*(255..192)
  FIN SI
  SI *source2*(193) = 0 **ALORS**
    destination(255..192) ← *source1*(191..128)
  FIN SI
  SI *source2*(193) = 1 **ALORS**
    destination(255..192) ← *source1*(255..192)
  FIN SI
SINON SI version 128 bits immédiate **ALORS**
  SI *source2*(1) = 0 **ALORS**
    destination(63..0) ← *source1*(63..0)
  FIN SI
  SI *source2*(1) = 1 **ALORS**
    destination(63..0) ← *source1*(127..64)
  FIN SI

```
   SI source2(65) = 0 ALORS
     destination(127..64) ← source1(63..0)
   FIN SI
   SI source2(65) = 1 ALORS
     destination(127..64) ← source1(127..64)
   FIN SI
   destination(255..128) ← 0
FIN SI
```

Mnémonique

Instruction	Opcode	Description
VPERMILPD *xmm1, xmm2,xmm3/m128*	(VEX.NDS.128) 66h 0Fh 38h 0Dh /r	Cette instruction permet d'effectuer la permutation de valeurs réels de double précision d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.
VPERMILPD *ymm1, ymm2,ymm3/m256*	(VEX.NDS.256) 66h 0Fh 38h 0Dh /r	Cette instruction

		permet d'effectuer la permutation de valeurs réels de double précision d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.
VPERMILPD *xmm1,xmm2/m128, imm8*	(VEX.128) 66h 0Fh 3Ah 05h /r *ib*	Cette instruction permet d'effectuer la permutation de valeurs réels de double précision d'un premier opérande source utilisant un champ de 8 bits de contrôle

		dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.
VPERMILPD *ymm1,ymm2/m256, imm8*	(VEX.256) 66h 0Fh 3Ah 05h /r *ib*	Cette instruction permet d'effectuer la permutation de valeurs réels de double précision d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.

Assembleur 80x86 — VPERMILPS

AVX (AMD ou INTEL)

Vector Permute Single-Precision Floating-Point Values

Syntaxe

VPERMILPS *destination*, *source1*, *source2*

Description

Cette instruction permet d'effectuer la permutation de valeurs réels de simple précision d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.

Algorithme

```
MODULE Select4(SRC, control)
  EVALUER CAS control(1..0)
    CAS 0: TMP ← SRC(31..0)
    CAS 1: TMP ← SRC(63..32)
    CAS 2: TMP ← SRC(95..64)
    CAS 3: TMP ← SRC(127..96)
  FIN EVALUER CAS
  RETOURNER TMP

SI version immediate 256 bits ALORS
  destination(31..0) ← Select4(source1(127..0), imm8(1..0))
  destination(63..32) ← Select4(source1(127..0), imm8(3..2))
  destination(95..64) ← Select4(source1(127..0), imm8(5..4))
  destination(127..96) ← Select4(source1(127..0), imm8(7..6))
  destination(159..128) ← Select4(source1(255..128), imm8(1..0))
  destination(191..160) ← Select4(source1(255..128), imm8(3..2))
  destination(223..192) ← Select4(source1(255..128), imm8(5..4))
  destination(255..224) ← Select4(source1(255..128), imm8(7..6))
SINON SI version immediate 128 bits ALORS
  destination(31..0) ← Select4(source1(127..0), imm8(1..0))
  destination(63..32) ← Select4(source1(127..0), imm8(3..2))
  destination(95..64) ← Select4(source1(127..0), imm8(5..4))
  destination(127..96) ← Select4(source1(127..0), imm8(7..6))
  destination(255..128) ← 0
SINON SI version variable 256 bits ALORS
```

destination(31..0) ← Select4(*source1*(127..0), *source2*(1..0))
destination(63..32) ← Select4(*source1*(127..0), *source2*(33..32))
destination(95..64) ← Select4(*source1*(127..0), *source2*(65..64))
destination(127..96) ← Select4(*source1*(127..0), *source2*(97..96))
destination(159..128) ← Select4(*source1*(255..128), *source2*(129..128))
destination(191..160) ← Select4(*source1*(255..128), *source2*(161..160))
destination(223..192) ← Select4(*source1*(255..128), *source2*(193..192))
destination(255..224) ← Select4(*source1*(255..128), *source2*(225..224))
SINON SI version variable 128 bits **ALORS**
destination(31..0) ← Select4(*source1*(127..0), *source2*(1..0))
destination(63..32) ← Select4(*source1*(127..0), *source2*(33..32))
destination(95..64) ← Select4(*source1*(127..0), *source2*(65..64))
destination(127..96) ← Select4(*source1*(127..0), *source2*(97..96))
destination(255..128) ← 0
FIN SI

Mnémonique

Instruction	**Opcode**	**Description**
VPERMILPS *xmm1, xmm2,xmm3/m128*	(VEX.NDS.128) 66h 0Fh 38h 0Ch /r	Cette instruction permet d'effectuer la permutation de valeurs réels de simple précision d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats

		dans l'opérande de destination.
VPERMILPS *xmm1,xmm2/m128, imm8*	(VEX.128) 66h 0Fh 3Ah 04h /r *ib*	Cette instruction permet d'effectuer la permutation de valeurs réels de simple précision d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.
VPERMILPS *ymm1, ymm2,ymm3/m256*	(VEX.NDS.256) 66h 0Fh 38h 0Ch /r	Cette instruction permet d'effectuer la permutation de valeurs réels de simple précision d'un premier

		opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de destination.
VPERMILPS *ymm1,ymm2/m256, imm8*	(VEX.256) 66h 0Fh 3Ah 04h /r *ib*	Cette instruction permet d'effectuer la permutation de valeurs réels de simple précision d'un premier opérande source utilisant un champ de 8 bits de contrôle dans les octets bas d'un second opérande source et entreprose les résultats dans l'opérande de

		destination.

Assembleur 80x86 — VZEROALL

AVX (AMD ou INTEL) — *Vector Zero All YMM registers*

Syntaxe

VZEROALL

Description

Cette instruction permet de mettre la valeur 0 dans tous les registres *XMM* ou *YMM*.

Algorithme

SI mode 64 bits **ALORS**
  YMM0(255..0) ← 0
  YMM1(255..0) ← 0
  YMM2(255..0) ← 0
  YMM3(255..0) ← 0
  YMM4(255..0) ← 0
  YMM5(255..0) ← 0
  YMM6(255..0) ← 0
  YMM7(255..0) ← 0
  YMM8(255..0) ← 0
  YMM9(255..0) ← 0
  YMM10(255..0) ← 0
  YMM11(255..0) ← 0
  YMM12(255..0) ← 0
  YMM13(255..0) ← 0
  YMM14(255..0) ← 0
  YMM15(255..0) ← 0
SINON
  YMM0(255..0) ← 0
  YMM1(255..0) ← 0
  YMM2(255..0) ← 0
  YMM3(255..0) ← 0
  YMM4(255..0) ← 0
  YMM5(255..0) ← 0
  YMM6(255..0) ← 0
  YMM7(255..0) ← 0
FIN SI

Mnémonique

Instruction	Opcode	Description
VZEROALL	(VEX.256) 0Fh 77h	Cette instruction permet de mettre la valeur 0 dans tous les registres *XMM* ou *YMM*.

Assembleur 80x86 — VZEROUPPER

AVX (AMD ou INTEL)

Zero Upper bits of YMM registers

Syntaxe

VZEROUPPER

Description

Cette instruction permet de mettre la valeur 0 dans chacun des 128 bits de la partie haute de tous les registres *XMM* ou *YMM*.

Algorithme

```
SI mode 64 bits ALORS
   YMM0(255..128) ← 0
   YMM1(255..128) ← 0
   YMM2(255..128) ← 0
   YMM3(255..128) ← 0
   YMM4(255..128) ← 0
   YMM5(255..128) ← 0
   YMM6(255..128) ← 0
   YMM7(255..128) ← 0
   YMM8(255..128) ← 0
   YMM9(255..128) ← 0
   YMM10(255..128) ← 0
   YMM11(255..128) ← 0
   YMM12(255..128) ← 0
   YMM13(255..128) ← 0
   YMM14(255..128) ← 0
   YMM15(255..128) ← 0
SINON
   YMM0(255..128) ← 0
   YMM1(255..128) ← 0
   YMM2(255..128) ← 0
   YMM3(255..128) ← 0
   YMM4(255..128) ← 0
   YMM5(255..128) ← 0
   YMM6(255..128) ← 0
   YMM7(255..128) ← 0
```

FIN SI

Mnémonique

Instruction	**Opcode**	**Description**
VZEROUPPER	(VEX.128) 0Fh 77h	Cette instruction permet de mettre la valeur 0 dans chacun des 128 bits de la partie haute de tous les registres *XMM* ou *YMM*.

Assembleur 80x86	**WAIT**
INTEL 8088+	*Wait Until Not Busy*

Syntaxe

```
WAIT
```

Description

Cette instruction permet de faire passer le microprocesseur en mode d'attente jusqu'à ce que la ligne de teste sur la carte mère s'active.

Mnémonique

Instruction	Opcode	Description
WAIT	9Bh	Cette instruction permet de faire passer le microprocesseur en mode d'attente jusqu'à ce que la ligne de teste sur la carte mère s'active.

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 839

Assembleur 80x86	**WBINVD**
INTEL 80486+	*Write Back and Invalid Cache*

Syntaxe

```
WBINVD
```

Description

Cette instruction permet de désactiver et de vider le tampon interne du micro-processeur.

Algorithme

```
SI (cache interne est WB et dans mode WB) ALORS
   Retour d'écriture du cache interne
FIN SI
Vide le cache interne
Envoi un signal externe de cache au retour d'écriture
Envoi un signal externe de cache à vider
```

Mnémonique

Instruction	Opcode	Description
WBINVD	0Fh 09h	Cette instruction permet de désactiver et de vider le tampon interne du micro-processeur.

Voir également

Instruction assembleur 80x86 - Instruction CLFLUSH
Instruction assembleur 80x86 - Instruction INVD

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 531 à 532.

Assembleur 80x86	**WRMSR**
INTEL Pentium	*Write Model-Specific Register*

Syntaxe

```
WRMSR
```

Description

Cette instruction écrit les valeurs contenu dans le *MSR* (*Model-Specific Register*) en fonction du registre d'index ECX dans la paire des registres EDX:EAX.

Algorithme

```
SI ECX est un nombre valide de MSR ET CPL = 0 ALORS
  MSR(ECX) ← EDX:EAX
SINON
  Faute de protection général: INT 0Dh
FIN SI
```

Mnémonique

Instruction	Opcode	Description
WRMSR	0Fh 30h	Cette instruction écrit les valeurs contenu dans le *MSR* (*Model-Specific Register*) en fonction du registre d'index ECX dans la paire des registres EDX:EAX.

Voir également

Instruction assembleur 80x86 - Instruction RDMSR

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 533 à 534.

Assembleur 80x86 — WRSHR

Cyrix 6x86MX (EMMX)+ — *Write SMM Header Pointer Register*

Syntaxe

WRSHR *source*

Description

Cette instruction écrit les valeurs contenu dans l'opérande source dans l'entête *SMM*.

Algorithme

SMHR ← *source*

Mnémonique

Instruction	Opcode	Description
RDSHR *reg/mem32*	0Fh 37h /r	Cette instruction écrit les valeurs contenu dans l'opérande source dans l'entête *SMM*.

Assembleur 80x86 XADD

INTEL 80486+

eXchange ADDition

Syntaxe

XADD *dest*, *source*

Description

Cette instruction permet d'échanger le premier opérande avec le deuxième opérande, et ensuite effectue la somme des valeurs dans le première opérande.

Algorithme

Temporary ← *dest*
dest ← *dest* + *source*
source ← *Temporary*

Mnémonique

Instruction	Opcode	Description
XADD *reg/mem8*, *reg8*	0Fh C0h /r	Echange le contenu d'un registre 8 bits avec le contenu d'une opérande mémoire ou registre 8 bits et charge la somme dans la destination.
XADD *reg/mem16*, *reg16*	0Fh C1h /r	Echange le contenu d'un registre 16 bits avec le contenu d'une opérande mémoire ou registre 16 bits et charge la somme dans la destination.

XADD *reg/mem32*, *reg32*	0Fh C1h /r	Echange le contenu d'un registre 32 bits avec le contenu d'une opérande mémoire ou registre 32 bits et charge la somme dans la destination.
XADD *reg/mem64*, *reg64*	0Fh C1h /r	Echange le contenu d'un registre 64 bits avec le contenu d'une opérande mémoire ou registre 64 bits et charge la somme dans la destination.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment

				de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BSWAP
Instruction assembleur 80x86 - Instruction XCHG

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 535 à 537.

Assembleur 80x86 XBTS

INTEN 80386 niveau A0-B0 seulement

Extract Bit String

Syntaxe

XBTS *destination, source*

Description

Cette instruction permet de prendre des bits d'une opérande source et de les mettres dans une opérande destinataire.

Mnémonique

Instruction	Opcode	Description
XBTS *reg16,r/m16*	0Fh A6h /r	Cette instruction permet de prendre des bits d'une opérande source et de les mettres dans une opérande destinataire.
XBTS *reg32,r/m32*	66h 0Fh A6h /r	Cette instruction permet de prendre des bits d'une opérande source et de les mettres dans une opérande destinataire.

Assembleur 80x86	**XCHG**
INTEL 8088+	*Exchange*

Syntaxe

XCHG *opérandecible1*,*opérandecible2*

Description

Cette instruction permet d'échanger la valeur de deux opérandes.

Algorithme

```
temp ← opérandecible1
opérandecible1 ← opérandecible2
opérandecible2 ← temp
```

Mnémonique

Instruction	Opcode	Description
XCHG AX, *reg16*	90h +rw	Echange le contenu du registre AX avec le contenu d'un registre 16 bits.
XCHG *reg16*, AX	90h +rw	Echange le contenu du registre 16 bits avec le contenu du registre AX.
XCHG EAX, *reg32*	90h +rd	Echange le contenu du registre EAX avec le contenu d'un registre 32 bits.
XCHG *reg32*, EAX	90h +rd	Echange le contenu du registre 32 bits avec le contenu du registre EAX.

XCHG RAX, *reg64*	90h +rq	Echange le contenu du registre RAX avec le contenu d'un registre 64 bits.
XCHG *reg/mem8*, *reg8*	86h /r	Echange le contenu d'une opérande registre ou mémoire 8 bits avec le contenu d'un registre 8 bits.
XCHG *reg8*, *reg/mem8*	86h /r	Echange le contenu d'un registre 8 bits avec le contenu d'une opérande registre ou mémoire 8 bits.
XCHG *reg/mem16*, *reg16*	87h /r	Echange le contenu d'une opérande registre ou mémoire 16 bits avec le contenu d'un registre 16 bits.
XCHG *reg16*, *reg/mem16*	87h /r	Echange le contenu d'un registre 16 bits avec le contenu d'une opérande registre ou mémoire 16 bits.
XCHG *reg/mem32*, *reg32*	87h /r	Echange le contenu d'une opérande registre ou mémoire 32 bits avec le contenu d'un registre 32 bits.
XCHG *reg32*, *reg/mem32*	87h /r	Echange le contenu d'un registre 32 bits avec le contenu d'une opérande registre ou mémoire 32 bits.
XCHG *reg/mem64*, *reg64*	87h /r	Echange le contenu d'une opérande registre ou mémoire 64 bits avec le contenu d'un registre 64 bits.
XCHG *reg64*, *reg/mem64*	87h /r	Echange le contenu d'un registre 64 bits avec le contenu d'une opérande registre ou mémoire 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction
#AC(Vérifie		X	X	Un désalignement

l'alignement)				de la référence mémoire est effectué quand une vérification d'alignement est activé

Voir également

Instruction assembleur 80x86 - Instruction BSWAP
Instruction assembleur 80x86 - Instruction XADD

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 839
Assembleur Facile, Philippe Mercier, 1990, ISBN: 2-501-01176-7, page 418
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 245.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 538 à 541.

Assembleur 80x86 — XGETBV

INTEL SDM (Core i7)+ — *Get Value of Extended Control Register*

Syntaxe

XGETBV

Description

Cette instruction permet d'effectuer la lecture du registre de contrôle étendue (XCR) spécifié par le registre ECX dans les registres EDX:EAX.

Algorithme

EDX:EAX ← XCR(ECX)

Mnémonique

Instruction	Opcode	Description
XGETBV	0Fh 01h D0h	Cette instruction permet d'effectuer la lecture du registre de contrôle étendue (XCR) spécifié par le registre ECX dans les registres EDX:EAX.

Exemple

Cet exemple permet de tester la présence de cette instruction avant d'effectuer la lecture du registre de contrôle étendue :

```
1.  PUSH EBX
2.   MOV EAX,1
```

```
3.    CPUID
4.    BT ECX, 27  ; CPUID.1:ECX.OSXSAVE[bit 27] = 1: Est-
      ce que le XGETBV est disponible et actif pour l'utilisation d'application ?
5.    JNC @not_supported
6.     ; Mettre quelques instructions avant ici
7.     ; ...
8.    XGETBV
9.     ; Mettre quelques instructions après ici
10.    ; ...
11. @not_supported:
12. POP EBX
```

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 542 à 543.

Assembleur 80x86	**XLAT**
INTEL 8088+	*Translate By Table Lookup*

Syntaxe

XLAT *tablesource*

Description

Cette instruction permet de remplacer le contenu du registre AL par un octet de la «*tablesource*».

Algorithme

AL ← DS:BX+AL

Mnémonique

Instruction	**Opcode**	**Description**
XLAT *mem8*	D7h	Fixe le contenu avec le DS:[(R)BX+AL]

Exceptions

Message	**Mode réel**	**Virtuel 8086**	**Mode protégé**	**Description**
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de

				pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

Références

Le livre d'Or PC, Martin Althaus, 1992, ISBN: 2-7361-0934-1, page 839
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, Edition Advanced Micro Devices, Revision 3.14, September 2007, Publication No. 24594, page 245.
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 544 à 546.

Assembleur 80x86 XLATB

INTEL 8088+

Table Look-up Translation Byte

Syntaxe

```
XLATB
```

Description

Cette instruction permet de remplacer le contenu du registre AL par un octet de la «*tablesource*» sans opérande.

Algorithme

```
AL ← DS:BX+AL
```

Mnémonique

Instruction	Opcode	Description
XLATB	D7h	Fixe le contenu avec le DS:[(R)BX+AL]

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du

				segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat de l'exécution de l'instruction

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 544 à 546.

Assembleur 80x86 — XOR

INTEL 8088+ — *binary eXclusive OR*

Syntaxe

XOR *OpérandeCible*, *OpérandeSource*

Description

L'instruction *XOR* effectue un *OU exclusf BINAIRE* sur les 2 opérandes spécifiés, le calcul est placé dans la première opérande, c'est-à-dire l'*Opérande Cible*. Rappelons qu'un *OU exclusif BINAIRE* donne le résultat 1 si une des 2 opérandes vaut 1 et donne 0 si les deux bits valent 0 ou 1.

Algorithme

OpérandeCible ← *OpérandeCible* XOR *OpérandeSource*

Mnémonique

Instruction	Opcode	Description
XOR AL, *imm8*	34h *ib*	Effectue un «Ou exclusif binaire» du contenu du registre AL avec une valeur immédiate de 8 bits.
XOR AX, *imm16*	35h *iw*	Effectue un «Ou exclusif binaire» du contenu du registre AX avec une valeur immédiate de 16 bits.
XOR EAX, *imm32*	35h *id*	Effectue un «Ou exclusif binaire» du contenu du registre EAX avec une

		valeur immédiate de 32 bits.
XOR RAX, *imm32*	35h *id*	Effectue un «Ou exclusif binaire» du contenu du registre RAX avec une valeur immédiate de 32 bits.
XOR *reg/mem8*, *imm8*	80h /6 *ib*	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 8 bits avec une valeur immédiate de 8 bits.
XOR *reg/mem16*, *imm16*	81h /6 *iw*	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 16 bits avec une valeur immédiate de 16 bits.
XOR *reg/mem32*, *imm32*	81h /6 *id*	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 32 bits avec une valeur immédiate de 32 bits.
XOR *reg/mem64*, *imm32*	81h /6 *id*	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 64 bits avec une valeur immédiate de 32 bits.
XOR *reg/mem16*, *imm8*	83h /6 *ib*	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 16 bits avec une valeur immédiate de 8 bits.
XOR *reg/mem32*, *imm8*	83h /6 *ib*	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 32 bits avec une valeur immédiate de 8 bits.
XOR *reg/mem64*, *imm8*	83h /6 *ib*	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 64 bits avec une valeur

		immédiate de 8 bits.
XOR *reg/mem8*, *reg8*	30h /r	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 8 bits avec le contenu d'un registre de 8 bits.
XOR *reg/mem16*, *reg16*	31h /r	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 16 bits avec le contenu d'un registre de 16 bits.
XOR *reg/mem32*, *reg32*	31h /r	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 32 bits avec le contenu d'un registre de 32 bits.
XOR *reg/mem64*, *reg64*	31h /r	Effectue un «Ou exclusif binaire» du contenu d'une opérande mémoire ou registre 64 bits avec le contenu d'un registre de 64 bits.
XOR *reg8*, *reg/mem8*	32h /r	Effectue un «Ou exclusif binaire» du contenu d'un registre de 8 bits avec le contenu d'une opérande mémoire ou registre 8 bits.
XOR *reg16*, *reg/mem16*	33h /r	Effectue un «Ou exclusif binaire» du contenu d'un registre de 16 bits avec le contenu d'une opérande mémoire ou registre 16 bits.
XOR *reg32*, *reg/mem32*	33h /r	Effectue un «Ou exclusif binaire» du contenu d'un registre de 32 bits avec le contenu d'une opérande mémoire ou registre 32 bits.
XOR *reg64*, *reg/mem64*	33h /r	Effectue un «Ou exclusif binaire» du contenu d'un registre de 64 bits avec le contenu d'une opérande mémoire

		ou registre 64 bits.

Exceptions

Message	Mode réel	Virtuel 8086	Mode protégé	Description
#SS(Pile)	X	X	X	Une adresse mémoire dépasse la limite du segment de pile ou n'est pas canonique
#GP(Protection général)	X	X	X	Une adresse mémoire dépasse la limite du segment de données ou n'est pas canonique
			X	L'opérande de destination n'est pas dans un segment non écrivable
			X	Un segment de données nulle est utilisé comme référence mémoire
#PF(Faute de page)		X	X	Une faute de page résultat

				de l'exécution de l'instruction
#AC(Vérifie l'alignement)		X	X	Un désalignement de la référence mémoire est effectué quand une vérification d'alignement est activé

Exemple

Cet exemple permet de mettre la valeur 0 dans le registre EAX (EAX = 0). Cette technique utilise moins d'octets que l'instruction «*MOV EAX,0*», cependant il modifie l'état du registre des drapeaux, il faut donc l'utiliser avec astuce lors d'optimisation de code :

```
1. XOR EAX,EAX
```

Voir également

Instruction assembleur 80x86 - Instruction OR
Instruction assembleur 80x86 - Instruction AND
Instruction assembleur 80x86 - Instruction NOT
Instruction assembleur 80x86 - Instruction NEG

Références

Le livre d'Or PC, *Martin Althaus*, *1992*, *ISBN: 2-7361-0934-1*, *page 839*
Assembleur Facile, *Philippe Mercier*, *1990*, *ISBN: 2-501-01176-7*, *page 419*
AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions, *Edition Advanced Micro Devices*, *Revision 3.14*, *September 2007*, *Publication No. 24594*, *page 248.*
Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 547 à 549.

Assembleur 80x86 — XORPD

INTEL Pentium 4 (SSE2)+

Bitwise Logical XOR for Double-Precision Floating-Point Values

Syntaxe

XORPD *destination*, *source*

Description

Cette instruction permet d'effectuer un ou exclusif binaire de 2 paquets de valeurs réel de double précision dans un opérande source et destination et de mettre le résultat dans l'opérande de destination.

Algorithme

destination(127..0) ← *destination*(127..0) XOR *source*(127..0)

Mnémonique

Instruction	Opcode	Description
XORPD *xmm1*, *xmm2/m128*	66h 0Fh 57h /r	Cette instruction permet d'effectuer un ou exclusif binaire de 2 paquets de valeurs réel de double précision dans un opérande source et destination et de mettre le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction

Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 550 à 551.

Assembleur 80x86

INTEL Pentium III+ (KNI/MMX2)

XORPS

Bitwise Logical XOR for Single-Precision Floating-Point Values

Syntaxe

XORPS *dest,source*

Description

Cette instruction permet d'effectuer un ou exclusif binaire de 4 paquets de valeurs réel de simple précision dans une opérande source et destination et de mettre le résultat dans l'opérande de destination.

Algorithme

dest ← *dest* XOR *source*

Mnémonique

Instruction	Opcode	Description
XORPS *xmm1,xmm2/m128*	0Fh 57h /r	Cette instruction permet d'effectuer un ou exclusif binaire de 4 paquets de valeurs réel de simple précision dans une opérande source et destination et de mettre le résultat dans l'opérande de destination.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 552 à 553.

Assembleur 80x86 — XRSTOR

INTEL SDM (Core i7)+ — *Restore Processor Extended States*

Syntaxe

XRSTOR *operande*

Description

Cette instruction permet d'effectuer une restauration complète ou partiel des états actifs du processeur en utilisant les informations entreposé dans les adresses de mémoire spécifié par l'opérande.

Algorithme

```
RS_TMP_MASK(62..0) ← (EDX(30..0) << 32 ) U EAX(31..0)
ST_TMP_MASK(62..0) ← SRCMEM.HEADER.XSTATE_BV(62..0)
SI ( (XCR0(62..0) XOR 7FFFFFFFFFFFFFFFh ) ∩ ST_TMP_MASK(62..0) ) ALORS
  EXCEPTION #GP(0)
SINON
  BOUCLE POUR i ←0 JUSQU'A 62 SAUT 1
    SI RS_TMP_MASK(i) ∩ XCR0(i) ALORS
      SI ST_TMP_MASK(i) ALORS
      EVALUER CAS i
        CAS 0: Processor state(x87 FPU) ← SRCMEM. FPUSSESave_Area(FPU)
        CAS 1: Processor state(SSE) ← SRCMEM. FPUSSESave_Area(SSE)
        AUTREMENT:
          Processor state(i) ← SRCMEM. Ext_Save_Area(i)
      FIN EVALUER CAS
      SINON
        Processor extended state(i) ← Valeurs que le Processor fournit
        EVALUER CAS i
          CAS 1: MXCSR ← SRCMEM. FPUSSESave_Area(SSE)
        FIN EVALUER CAS
      FIN SI
```

```
    FIN SI
  FIN BOUCLE POUR
FIN SI
```

Mnémonique

Instruction	Opcode	Description
XRSTOR *mem*	0Fh AEh /5	Cette instruction permet d'effectuer une restauration complète ou partiel des états actifs du processeur en utilisant les informations entreposé dans les adresses de mémoire spécifié par l'opérande.
XRSTOR64 *mem*	REX.W+ 0Fh AEh /5	Cette instruction permet d'effectuer une restauration complète ou partiel des états actifs du processeur en utilisant les informations entreposé dans les adresses de mémoire spécifié par l'opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 554 à 559.

Assembleur 80x86 — XSAVE

INTEL SDM (Core i7)+ — *Save Processor Extended States*

Syntaxe

XSAVE *operande*

Description

Cette instruction permet d'effectuer une sauvegarde complète ou partiel des états actifs du processeur en utilisant les informations entreposé dans les adresses de mémoire spécifié par l'opérande.

Algorithme

```
TMP_MASK(62..0) ← ( (EDX(30..0) << 32 ) U EAX(31..0) ) ∩ XFEATURE_ENABLED_MASK(62..0)
BOUCLE POUR i ← 0 JUSQU'A 62 SAUT 1
  SI TMP_MASK(i) = 1 ALORS
    EVALUER CAS i
      CAS 0: DEST.FPUSSESAVE_Area(x87 FPU) ← processor state(x87 FPU)
      CAS 1: DEST.FPUSSESAVE_Area(SSE) ← processor state(SSE)
      AUTREMENT:
      DEST.Ext_Save_Area(i) ← processor state(i)
    FIN EVALUER CAS
    DEST.HEADER.XSTATE_BV(i) ← INIT_FUNCTION(i)
  FIN SI
FIN BOUCLE POUR
```

Mnémonique

Instruction	Opcode	Description

XSAVE *mem*	0Fh AEh /4	Cette instruction permet d'effectuer une sauvegarde complète ou partiel des états actifs du processeur en utilisant les informations entreposé dans les adresses de mémoire spécifié par l'opérande.
XSAVE64 *mem*	REX.W+ 0Fh AEh /4	Cette instruction permet d'effectuer une sauvegarde complète ou partiel des états actifs du processeur en utilisant les informations entreposé dans les adresses de mémoire spécifié par l'opérande.

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 560 à 563.

Assembleur 80x86 — XSETBV

INTEL SDM (Core i7)+ — *Set Extended Control Register*

Syntaxe

```
XSETBV
```

Description

Cette instruction permet d'effectuer l'écriture du registre de contrôle étendue (XCR) spécifié par les registres EDX:EAX à l'aide du registre ECX.

Algorithme

```
XCR(ECX) ← EDX:EAX
```

Mnémonique

Instruction	Opcode	Description
XSETBV	0Fh 01h D1h	Cette instruction permet d'effectuer l'écriture du registre de contrôle étendue (XCR) spécifié par les registres EDX:EAX à l'aide du registre ECX.

Exemple

Cet exemple permet de tester la présence de cette instruction avant d'effectuer l'écriture du registre de contrôle étendue :

```
1.  PUSH EBX
2.   MOV EAX,1
3.   CPUID
4.   BT ECX, 27  ; CPUID.1:ECX.OSXSAVE[bit 27] = 1: Est-
    ce que le XSETBV est disponible et actif pour l'utilisation d'application ?
5.   JNC @not_supported
6.    ; Mettre quelques instructions avant ici
7.    ; ...
8.   XSETBV
9.    ; Mettre quelques instructions après ici
10.   ; ...
11. @not_supported:
12.  POP EBX
```

Références

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2B: Instruction Set Reference, N-Z, Edition Intel, Mars 2010, Publication No. 253667-034US, page 564 à 565.