MMX & XMM

Arithmétique

		Byte(b)	Word(w)	Double(d)	Quad(q)
Somme +	$M1_B[i] = M1_B[i] + M2_B[i]$	pADDb	pADDw	pADDd	pADDq
	$M1_B[i] = limit(-128, M1_B[i] + M2_B[i], 127)$	pADDsb	pADDsw		
	$M1_B[i] = limit(0, M1_B[i] + M2_B[i], 255)$	pADDusb	pADDusw		
Soustraction -	$M1_B[i] = M1b[i] - M2b[i]$	pSUBb	pSUBw	pSUBd	pSUBq
	$M1_B[i] = limit(-128, M1_B[i] - M2_B[i], 127)$	pSUBsb	pSUBsw		
	$M1_B[i] = limit(0, M1_B[i] - M2_B[i], 255)$	pSUBusb	pSUBusw		
Multiplication X	$M1_{UW}[i] = M1_{UW}[i]*M2_{UW}[i]$ & 0xFFFF		pMULlw	pMULdq (i)	
	$M1_{UW}[i] = M1_{UW}[i]*M2_{UW}[i] >> 16$		pMULhw		
			pMULhuw		
Multiplication puis Somme	$M1_D[i] = M1_W[2i]*M2_W[2i] + M1_W[2i+1]*M2_W[2i+1]$	pMADDubsw	pMADDwd		
Somme des différences absolues	$M1_W[0] = \sum\|M1_B[i]-M2_B[i]\|, M1_W[i<>0] = 0$		pSADbw		
			MPSADbw (i)		
Valeur absolue	$M1_W[i] = \|M2_W[i]\|$	pABSb	pABSw	pABSd	pABSq
Maximum	$M1_{UB}[i] = Max(M1_{UB}[i], M2_{UB}[i])$	pMAXsb (i)	pMAXsw	pMAXsd (i)	
		pMAXub (i)	pMAXuw (i)	pMAXud (i)	
Minimum	$M1_{UB}[i] = Min(M1_{UB}[i], M2_{UB}[i])$	pMINsb (i)	pMINsw	pMINsd (i)	
		pMINub (i)	pMINuw (i)	pMINud (i)	
Moyenne	$M1_B[i] = (M1_B[i] + M2_B[i])/2$		pAVGb	pAVGw	
Égalité ?	$M1_B[i] = M1_B[i] == M2_B[i]? 0xFF : 0$	pCmpEQb	pCmpEQw	pCmpEQd	pCmpEQq (i)
Supérieur ?	$M1_B[i] = M1_B[i] > M2_B[i]? 0xFF : 0$	pCmpGTb	pCmpGTw	pCmpGTd	pCmpGTq (i)
Signes	$reg_{32}[0..7] = M_B[0..7] >> 8$	pMovMskb reg_{32},mem_{64}			
Signe conditionnel	$M1_W[i] = M1_W[i]*sign(M2_W[i])$	pSIGNb	pSIGNw	pSIGNd	

Logique

		Word(w)	Double(d)	Quad(q)	DQuad(dq)
And	$M1_Q = M1_Q$ & $M2_Q$				pAND
And not	$M1_Q = \sim M1_Q$ & $M2_Q$				pANDN
Or	$M1_Q = M1_Q$ | $M2_Q$				pOR
Xor	$M1_Q = M1_Q$ ^ $M2_Q$				pXOR
Shl	$M1_W[i] = M1_W[i] << imm8$	pSLLw	pSLLd	pSLLq	pSLLdq
Shr	$M1_W[i] = M1_W[i] >> imm8$	pSRLw	pSRLd	pSRLq	pSRLdq
Sar	$M1_W[i] = M1_W[i] / 2^{imm8}$	pSRAw	pSRAd	pSRAq	

Saturation : si le résultat dépasse une limite, il est ramené à la limite.

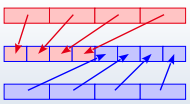
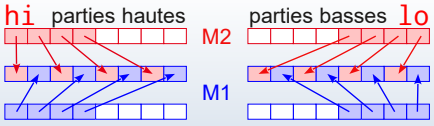
Unsigned : résultat non signé

Si le mode Saturation n'est pas actif, une troncature sanctionne les dépassements.

MMX XMM (i) intel seulement

MMX & XMM

Mouvements

		Byte(b)	Word(w)	Double(d)	Quad(q) DQuad(dq)
Comprimer			PACKsswb PACKuswb	PACKssdw	
Décomprimer		pUNPCKhbw pUNPCKlbw	pUNPCKhwd pUNPCKlwd	pUNPCKhdq pUNPCKldq	pUNPCKhqdq pUNPCKlqdq
Extension de signe	$M1_W[i] = int16(M2_B[i])$		pMOVSXbw	pMOVSXbd pMOVSXwd	pMOVSXbq pMOVSXwq pMOVSXdq
Extension à zéro	$M1_W[i] = unsigned\_int16(M2_{UB}[i])$		pMOVZXbw	pMOVZXbd pMOVZXwd	pMOVZXbq pMOVZXwq pMOVZXdq
Insérer	$M_W[imm8 \& 3] = reg_{32}[0..15]$ $M_W[autre] = idem$	pINSb	pINSw	pINSd	pINSq
Extraire	$reg_{32}[0..15] = M_W[imm8 \& 3]$ $reg_{32}[16..31] = 0$	pEXTRb	pEXTRw	pEXTRd	pEXTRq
Répartir	$M1_W[i] = M2_W[(imm8 >> 2i) \& 3]$ $X1_W[i] = X2_W[(imm8 >> 2i) \& 3 + (h ? 4:0)]$ $X1_B[i] = X2_B[(X2 >> 4i) \& 15]$	pSHUFb	pSHUFw pSHUFlw pSHUFhw	pSHUFd	
Échanger	$M_Q \leftrightarrow partie\ basse(X_{DQ})$				MOVdq2q MOVq2dq
Registre ↔ mémoire	$reg_{32}/mem_{32} = M_D[0]$ $M_D[0] = reg_{32}/mem_{32}$ $M_D[1] = 0$			MOVd MOVd	MOVq MOVdqu MOVq MOVdqu
Mêler	$M1_W[i] = imm8 >> i \& 1 ? M2_W[i]: M1_W[i]$ $M1_W[i] = XMM0_W[i] < 0 ? M2_W[i]: M1_W[i]$		pBLENDw pBLENDVw		
Quitter le mode MMX *(indispensable en MMX, inutile en XMM)*		EMMS			

Saturation : si le résultat dépasse une limite, il est ramené à la limite.

Si le mode Saturation n'est pas actif, une troncature sanctionne les dépassements.

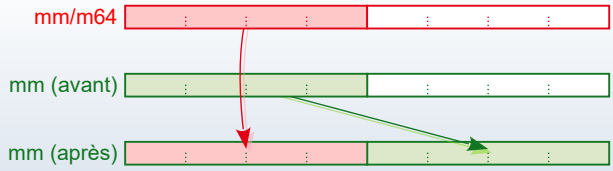
Unsigned : résultat non signé

MMX XMM intel seulement

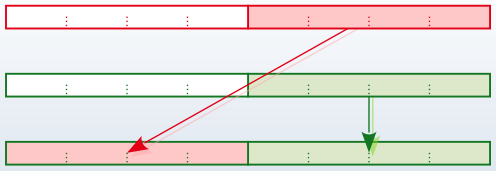
MMX & XMM

Entrelacer

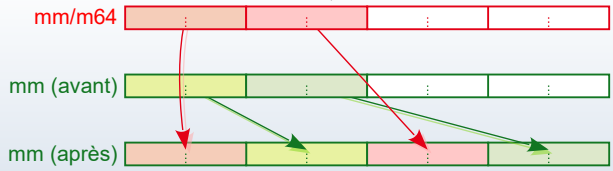
PUNPCKHDQ mm, mm/mem



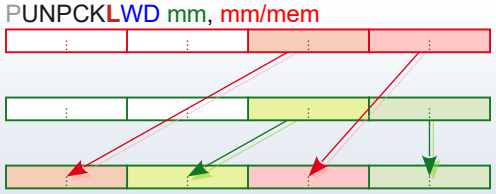
PUNPCKLDQ mm, mm/mem



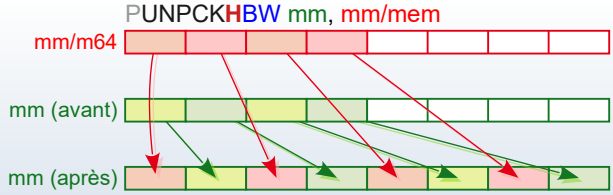
PUNPCKHWD mm, mm/mem



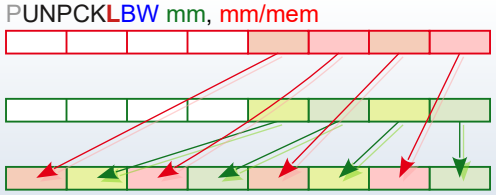
PUNPCKLWD mm, mm/mem



PUNPCKHBW mm, mm/mem

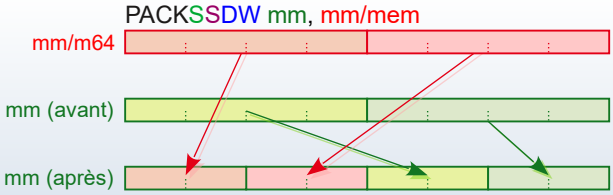


PUNPCKLBW mm, mm/mem

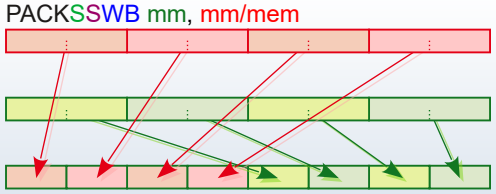


Compacter

PACKSSDW mm, mm/mem

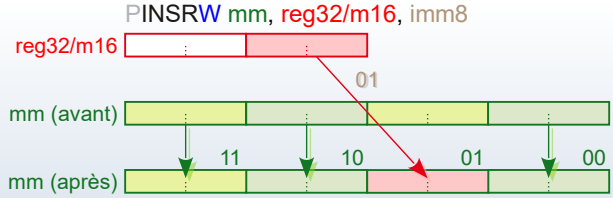


PACKUSWB mm, mm/mem
PACKSSWB mm, mm/mem

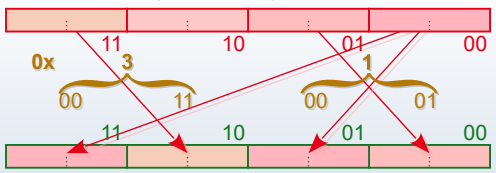


Répartir

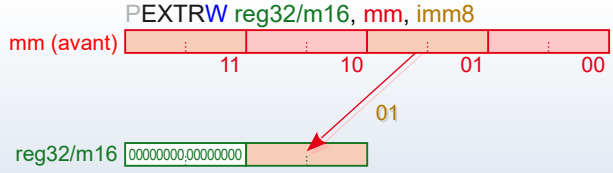
PINSRW mm, reg32/m16, imm8



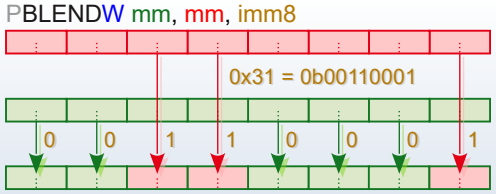
PSHUFW mm, mm/mem, imm8



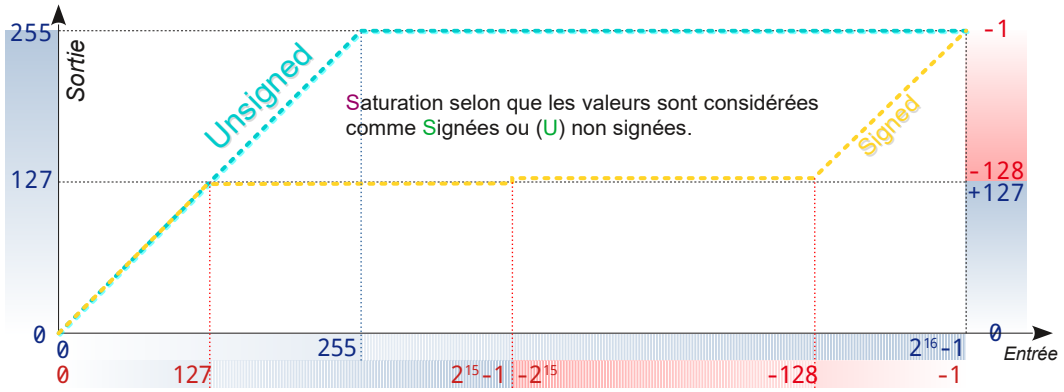
PEXTRW reg32/m16, mm, imm8



PBLENDW mm, mm, imm8



Saturation



Compactage entrelacé avec saturation

Les instructions PACK compressent 2 paquets en un seul dans l'ordre des arguments :

```
Entrées : MM0 = D0₁,D0₀  MM1 = D1₁,D1₀ PACKSSDW MM0, MM1 ; Wi = min(max(-2¹⁵, d), 2¹⁵-1)
```

Il peut être intéressant d'avoir la même opération avec un entrelacement des résultats :

```
Entrées : MM0 = D0₁,D0₀  MM1 = D1₁,D1₀ PACKSSDW  MM0, MM0 ; W0ᵢ = int16(D0ᵢ)
          Valeurs signées               PACKSSDW  MM1, MM1 ; W1ᵢ = int16(D1ᵢ)
                                        PUNPKLWD  MM0, MM1 ; D0ᵢ = int16(D1ᵢ) << 16 + int16(D0ᵢ)
```

Remarque : la compression présuppose les opérandes signés en entrée et c'est l'instruction qui définit le type du résultat : PACKSSDW prend 2x4 entiers signés de 32 bits (**D**) qu'il convertit, les **S**aturant si nécessaire, en 8 entiers de 16 bits **S**ignés . PACKUSWB prend 2x8 entiers signés de 16 bits (**W**) qu'il convertit, les **S**aturant si nécessaire, en 16 entiers non signés (**U**) de 8 bits (**B**).

Compactage entrelacé sans saturation

Similaire au cas précédent sans saturation laquelle est remplacée par un modulo au format cible (i.e. W_0 des D).

```
Entrées : MM0 = D0₁, D0₀          PSLLD MM1, 16           ; MM1 = W1₁ << 16, W1₀ << 16
      et MM1 = D1₁, D1₀           PAND  MM0, {0,-1,0,-1}  ; MM0 =      W0₂,       W0₀
                                  POR   MM0, MM1          ; MM0 =  W1₁, W0₂,  W1₀,  W0₀
```

Décompactage non-entrelacé

```
Entrées : MM0 = W0₃, W0₂, W0₁, W0₀   MOVQ      MM2, MM0 ; W2ᵢ = W0ᵢ
      et MM1 = W1₃, W1₂, W1₁, W1₀    PUNPCKLDQ MM0, MM1 ; MM0 = W1₁, W1₀, W0₁, W0₀
                                     PUNPCKHDQ MM2, MM1 ; MM2 = W1₃, W1₂, W0₃, W0₂
```

Décompactage signé (exemple : 8 entiers 16 bits → 2 x 4 entiers 32 bits)

```
Entrée : MM0 = W0₃, W0₂, W0₁, W0₀   PXOR      XMM0, XMM0 ; W0ᵢ = 0
                                    PCMPGTW   XMM0, XMM1 ; W0ᵢ = (0 > W1) ? $FFFF : 0
                                    MOVDQU    XMM2, XMM1 ; W2ᵢ = W1ᵢ
                                    PUNPCKLWD XMM1, XMM0 ; D1ᵢ = int32(W0ᵢ)
                                    PUNPCKHWD XMM2, XMM0 ; D2ᵢ = int32(W0ᵢ₊₄)
```

Différence absolue de nombres non signés

```
Entrées : MM0 = D0₃, … , D0₀   MOVQ    MM2, MM0 ; B2ᵢ = B0ᵢ
      et MM1 = D1₃, … , D1₀    PSUBUSB MM0, MM1 ; B0ᵢ = B0ᵢ < B1ᵢ ? 0 : B0ᵢ - B1ᵢ
                               PSUBUSB MM1, MM2 ; B1ᵢ = B0ᵢ < B1ᵢ ? B1ᵢ - B0ᵢ : 0
                               POR     MM0, MM1 ; B1ᵢ = B0ᵢ < B1ᵢ ? B1ᵢ - B0ᵢ : B0ᵢ - B1ᵢ
```

Différence absolue de nombres signés

```
Entrées : MM0 = D0₃, D0₂, D0₁, D0₀   PSUBD   MM1, MM0 ; D1ᵢ = D0ᵢ-D1ᵢ
      et MM1 = D1₃, D1₂, D1₁, D1₀    PABSD   MM0, MM1 ; D0i = |D0ᵢ-D1ᵢ|
```

MMX & XMM

Limitations d'un nombre non signé à une gamme [LO, HI]

```
PADDUSW MM0, ~P_HI           ; W0_i = 0xFFFF + min(X_i - HI, 0)
PSUBUSW MM0, ~(P_HI - P_LO)  ; W0_i = max(min(X_i, HI) - LO, 0)
PADDW   MM0, LO              ; W0_i = min(max(X_i, HI),  LO)
```

Mettre à 1

```
PXOR   MM0, MM0
PCMPEQ MM1, MM1
PSUBB  MM0, MM1      PSUBW MM0, MM1      PSUBD MM0, MM1
```

Mettre à 0

```
PXOR MM0, MM0
```

Mettre à -1

```
PCMPEQ MM1, MM1
```

Mettre à 2^n-1

```
PCMPEQ MM1, MM1
PSRLW  MM1, 16-n     PSRLD MM1, 32-n
```

Mettre à 2^n

```
PCMPEQW MM1, MM1
PCMPEQW MM2, MM2
PSLLW   MM1, n
PADDW   MM1, MM2
PANDN   MM1, MM2
```

Complémenter à 1 (NOT)

```
PCMPEQW MM2, MM2
PANDN   MM1, MM2
```

Limitation d'un nombre signé à une gamme [LO, HI]

Utilise les instructions d'addition et de soustraction avec saturation non signée. Nous utiliserons les constantes suivantes :

```
P_LO  =  LO,   LO,   LO,   LO        P_HI = HI, HI, HI, HI
P_MIN =  2^15, 2^15, 2^15, 2^15      WMIN = 0x8000
P_HX  = WHX3, WHX2, WHX1, WHX0       WHX_i = 0x7FFF - HI
P_DX  = WDX3, WDX2, WDX1, WDX0       WDX_i = 0xFFFF + LO - HI
```

Entrée: MM0 = X_3, X_2, X_1, X_0

```
PADDW   MM0, P_MIN        ; W0_i = X_i + 0x8000
PADDUSW MM0, P_HX         ; W0_i = 0xFFFF + max(X_i - HI, 0)
PSUBUSW MM0, P_DX         ; W0_i = min(max(X_i, HI) - LO, 0)
PADDW   MM0, P_LO         ; W0_i = min(max(X_i, HI),  LO)
```

Si HI-LO > 0x8000 alors l'algorithme peut être simplifié :

Ce code économise 1 cycle mais si HI – LO ≥ 0x8000, il ne fonctionne plus : 0xFFFF – (X < 0x8000) ≥ 0x8000 donc négatif.

```
PADDSSW MM0, ~P_HI        ; W0_i = 0xFFFF + min(X_i - HI, 0)
PSUBSSW MM0, ~(P_HI-P_LO) ; W0_i = max(min(X_i, HI) - LO, 0)
PADDW   MM0, P_LO         ; W0_i = min(max(X_i, HI),  LO)
```