Corrigé Examen Décembre 06 - Architectures Avancées

3H – Tous documents autorisés

PIPELINES

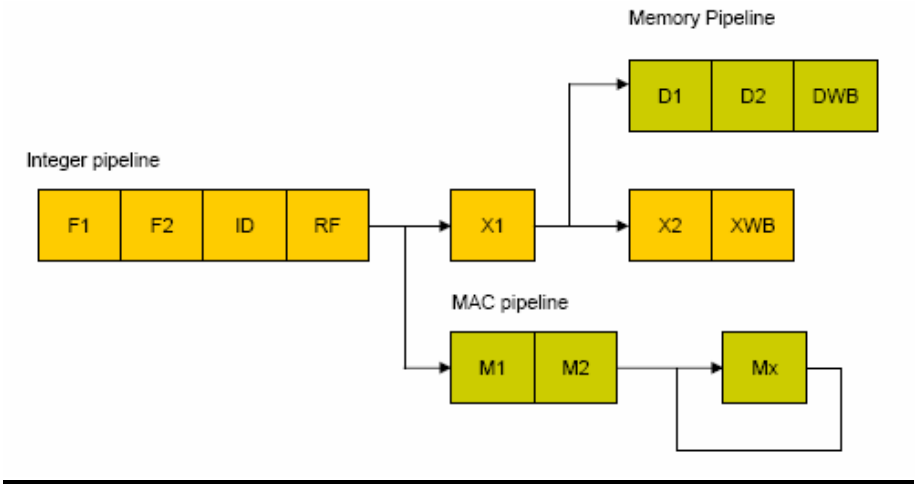


Figure 1 : pipelines entiers du processeur Xscale

La figure 1 donne les pipelines entiers du processeur Xscale pour les instructions entières, les instructions mémoires et l'instruction MAC (multiplication accumulation)
La signification des différents étages est

- F1 et F2 : phases d'acquisition (lecture) de l'instruction
- ID : décodage de l'instruction
- RF : lecture des registres
- X1 : calcul adresse mémoire, adresse de branchement, début exécution (EX1).
- X2 : fin exécution (EX2)
- XWB : écriture du résultat dans un registre
- D1 et D2 : phases accès cache données
- DWB : écrire de la donnée dans un registre
- M1 et M2 : phase de multiplication
- Mx : accumulateur ← accumulateur + résultat multiplication

Question 1 :

a) Donner les latences entre instruction producteur et instruction consommateur en nombre de cycles en supposant que tous les bypass nécessaires soient implémentés. (NB : la valeur n signifie que deux instructions dépendantes peuvent se suivre aux cycles i et i+n)

	Instruction producteur	Instruction consommateur	Latence scalaire
a	UAL Ri, -, -	UAL -, Ri, -	2
b	UAL Ri, -, -	ST Ri, ()	2
c	LW Ri, ()	UAL -, Ri, -	3

b) Quelle est la pénalité pour un branchement mal prédit ?
4 cycles

OPTIMISATION DE BOUCLES

On utilise le processeur superscalaire défini dans l'annexe 1
Soit la boucle suivante :

```
float X[128], Y[128], S, tmp ;
for (i = 0 ; i<128 ; i++){
      tmp = X[i]*Y[i]
      if (tmp >0.0) S+=tmp ; }
```

On supposera que l'adresse de X[0] est initialement dans R1, que l'adresse de Y[0] est initialement dans R2. R3 contient initialement le nombre d'itérations de la boucle. On utilisera F0 pour contenir la valeur 0.0.

Question 2

Quel est en nombre de cycles, le temps d'exécution par itération de la boucle originale sans et avec utilisation des instructions SIMD. ?

Sans SIMD : 10 cycles/itération
Avec SIMD : 11/4 = 2,75 cycles/itération

Sans déroulage, sans SIMD

	E0	E1	FA	FM
1 Boucle	LF F1, 0(R1)	LF F2, 0(R2)		
2	ADDI R1,R1,4	ADDI R2,R2,4		
3	ADDI R3,R3,-1			FMUL F1,F1,F2
4				
5				
6				
7			FMAX F1,F1,F0	
8				
9				
10		BNE R3, Boucle	FADD F9,F9,F1	

Sans déroulage, avec SIMD

	E0	E1	FA	FM
1 Boucle	PLF S1, 0(R1)	ADDI R3,R3,-4		
2	PLF F2, 0(R2)	ADDI R1,R1,16		
3	ADDI R2,R2,16			
4				PFMUL S1,S1,S2
5				
6				
7				
8			PFMAX S1,S1,S0	
9				
10				
11		BNE R3, Boucle	PFADD S9,S9,S1	

Question 3

Donner par itération de la boucle initiale
- le nombre de cycles sans instructions SIMD avec un déroulage d'ordre 4
- le nombre de cycles avec instructions SIMD avec un déroulage d'ordre 4

Avec déroulage, sans SIMD : 14/4 = 3,5 cycles/itération
Avec déroulage, avec SIMD : 17/ 16 = 1,0625 cycles/itération

Avec déroulage, sans SIMD

	E0	E1	FA	FM
1 Boucle	LF F1, 0(R1)	LF F2, 0(R2)		
2	LF F3, 4(R1)	LF F4, 4(R2)		
3	LF F5, 8(R1)	LF F6, 8(R2)		FMUL F1,F1,F2
4	LF F7, 12(R1)	LF F8, 12(R2)		FMUL F3,F3,F4
5	ADDI R1,R1,16	ADDI R2,R2,16		FMUL F5,F5,F6
6	ADDI R3,R3,-4			FMUL F7,F7,F8
7			FMAX F1,F1,F0	
8			FMAX F3,F3,F0	
9			FMAX F5,F5,F0	
10			FMAX F7,F7,F0	
11			FADD F9,F9,F1	
12			FADD F10,F10,F3	
13			FADD F11,F11,F5	
14		BNE R3, Boucle	FADD F12,F12,F7	

Avec déroulage, SIMD

	E0	E1	FA	FM
1 Boucle	PLF S1, 0(R1)	ADDI R3,R3,-16		
2	PLF S2, 0(R2)			
3	PLF S3, 16(R1)			
4	PLF S4, 16(R2)			PFMUL S1,S1,S2
5	PLF S5, 32(R1)			
6	PLF S6, 32(R2)			PFMUL S3,S3,S4
7	PLF S7, 48(R1)			
8	PLF S8, 48(R2)		PFMAX S1,S1,S0	PFMUL S5,S5,S6
9	ADDI R1,R1,64	ADDI R2,R2,64		
10			PFMAX S3,S3,S0	PFMUL S7,S7,S8
11			PFADD S9,S9,S1	
12			PFMAX S5,S5,S0	
13			PFADD S10,S10,S3	
14			PFMAX S7,S7,S0	
15			PFADD S11,S11,S3	
16				
17		BNE R3, Boucle	PFADD S12,S12,S5	

Résumé :
Sans déroulage sans SIMD : 10 cycles
Sans déroulage avec SIMD : 11/4 = 2,75 cycles
Avec déroulage sans SIMD : 14/4 = 3,75 cycles
Avec déroulage avec SIMD : 17/16 = 1,0625 cycles

Question 4

a) Pour la version sans SIMD avec déroulage de boucle d'ordre 4, donner le temps d'exécution total du programme pour les 128 itérations (prendre en compte les instructions avant l'entrée dans la boucle déroulée, et les instructions en sortie de la boucle déroulée)

b) Pour la version SIMD sans déroulage de boucle, on suppose qu'il n'existe pas d'instruction permettant de faire la somme des quatre floats contenus dans un registre SIMD. Ecrire un programme assembleur qui permette de faire cette somme et de ranger le résultat en mémoire (variable S). Donner le nombre de cycles pour exécuter ce programme.

a) Avant de rentrer dans la boucle, il faut initialiser à 0 le registre F0 et les registres F9, F10, F11 et F12 qui contiendront les sommes partielles.

FSUB F0, F0, F0
FSUB F9, F9, F9
FSUB F10, F10, F10
FSUB F11, F11, F11
FSUB F12, F12, F12

En sortie de boucle, il faut faire la somme des registres F9,F10,F11 et F12 et ranger dans la variable S
FADD F9,F9,F10
FADD F11,F11,F12

FADD F9,F9,F11

SF F9, adresse S

Total : 5 + 128 * 3,75 + 7 = 492 cycles

b) On range le registre 128 bits en mémoire, puis on fait la somme scalaire des 4 cases mémoires. Total : 12 cycles (en supposant des caches parfaits)

	E0	E1	FA	FM
1	PSF S9,TMP			
2				
3	LF F0, TMP(0)	LF F1, TMP(1)		
4	LF F2, TMP(2)	LF F1, TMP(3)		
5			FADD F0, F0,F1	
6			FADD F2, F3,F3	
7				
8				
9			FADD F0,F0,F2	
10				
11				
12	SF F0, S			

CACHES

La figure 2 donne le temps, exprimé en cycles par pixel, de la copie d'une image de N x N pixels avec un octet ou un float par pixel sur un processeur Pentium 4. Dans les deux cas, la copie est « vectorisée »

Question 5 : Expliquer les courbes obtenues.

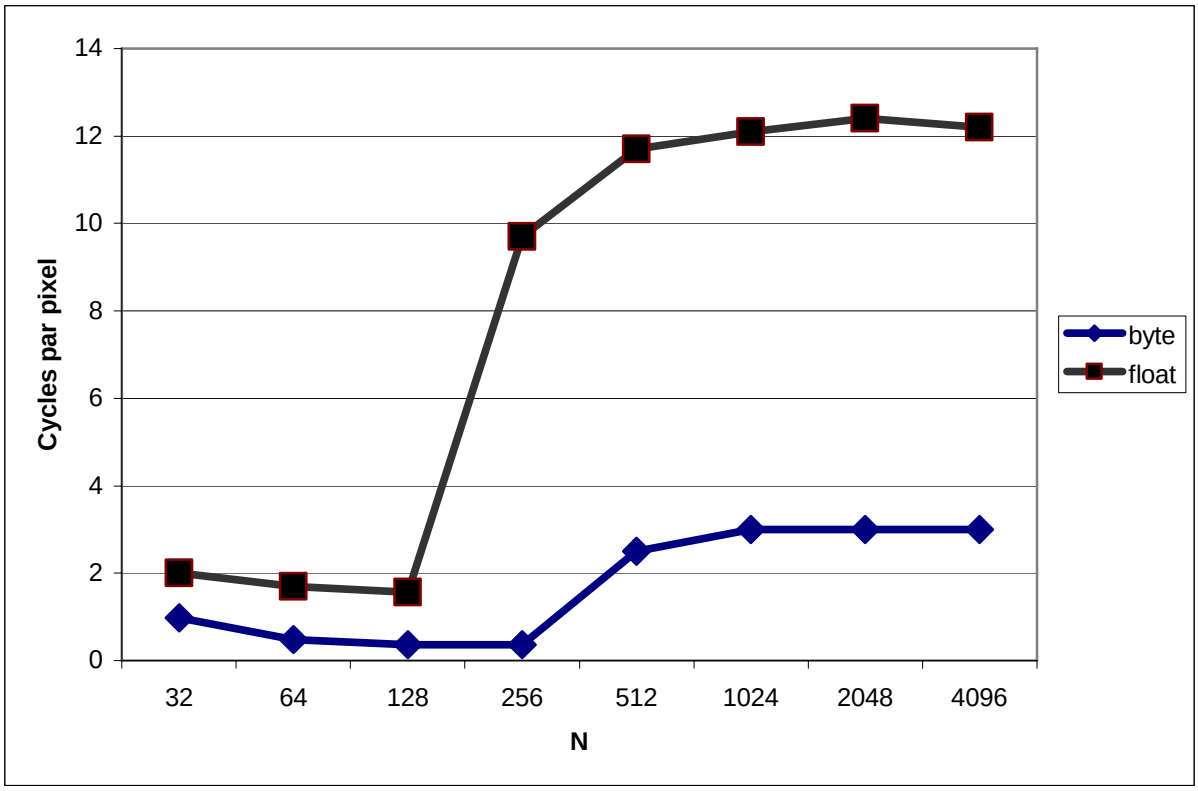


Figure 2 : Temps (en cycles par pixel) de la copie d'une image N x N pixels en fonction de N.

On constate que les valeurs de CPP pour les floats (32 bits) sont environ 4 fois celles pour les octets, ce qui est normal puisqu'il faudra transférer quatre fois plus d'octets.
La cassure dans les deux courbes commence à N=128 (floats) et N=256 (octets), ce qui correspond à nouveau à un rapport 4 (copie d'images N * N)
256 * 256 octets = 64 Ko.
512 * 512 octets = 256 Ko
Effet d'un cache L2 du Pentium 4 de 256 Ko

PIPELINE LOGICIEL AVEC TMS 320C62

Le code assembleur TMS320C62 ci-dessous donne l'itération du pipeline logiciel pour un programme C.

```
LOOP:
          LDW   .D1   *A4++,A2        ; load ai and ai+1
||        LDW   .D2   *B4++,B2        ; load bi and bi+1
||[A0]    ADD   .L1   A6,A7,A7        ; sum0+=(ai*bi)
||[B0]    ADD   .L2   B6,B7,B7        ; sum1+=(ai+1*bi+1)
||[A1]    B     .S2   LOOP            ; branch to loop

          MPY   .M1X  A2,B2,A6        ; ai*bi
||        MPYH  .M2X  A2,B2,B6        ; ai+1*bi+1
```

```
||        CMPLT .L1  0,A6,A0       ; A0 = 1 si A6 >0
||        CMPLT .L2 0,B6,B0        ; B0 = 1 si B6 >0
||[A1]    SUB  .S1  A1,1,A1   ; decrement loop counter

          ADD  .L1X A7,B7,A4       ; sum=sum0+sum1
```

Question 6 : Donner le code C correspondant au code assembleur.

```
short  X[N], Y[N], S,S1,S2, tmp1, tmp2 ;
S1=0; S2=0;
for (i = 0 ; i<N ; i+=2){
      tmp1 = X[i]*Y[i];
      tmp2 = X[i+1]*Y[i+1];
      if (tmp1 >0) S1+=tmp1 ;
      if (tmp2 >0) S1+=tmp1 ;
}
S=S1+S2
```

Question 7 : Quel est l'intervalle inter-itération (II) ? Justifier la valeur.

L'intervalle inter-itération est de 2. Il y a deux instructions utilisant L1 et deux instructions utilisant L2, d'où la valeur 2.

SIMD IA-32

Le jeu d'instruction IA-32 contient des instructions SIMD de conversion d'entiers 32 bits (int) en flottants 32 bits simple précision (float) : CVTDQ2PS .
On considère des variables 128 bits (_mm128i) v0, v1, v2, v3, v4, v5, v6, v7 pour les entiers et des variables 128 bits (_mm128) f0, f1, f2 et f3 pour les flottants simple précision.

Question 8 :En supposant que la variable v0 contient 16 octets (`unsigned char`) correspondant à des pixels (niveaux de gris), donner la suite des instructions SIMD nécessaires pour convertir les 16 données 8 bits en 16 données de type `float` dans les variables v4 à v7. Quelle opération faut-il alors effectuer pour avoir des données flottantes comprises entre 0.0 et 1.0 ? Quel est le nombre total d'instructions nécessaires pour faire la conversion ?

```
zero = _mm_xor_si128(zero,zero)
v1= _mm_unpacklo_epi8 (v0,zero) // Huit pixels bas de v0 étendus sur 16 bits
v2 = _mm_unpackhi_epi8(v0,zero) // Huit pixels hauts de v0 étendus sur 16 bits
v4 = _mm_unpacklo_epi16(v1, zero) // Quatre pixels bas de v1 étendus sur 32 bits
v5 = _mm_unpackhi_epi16(v1, zero) // Quatre pixels haut de v1 étendus sur 32 bits
v6 = _mm_unpacklo_epi16(v2, zero) // Quatre pixels bas de v2 étendus sur 32 bits
v7 = _mm_unpackhi_epi16(v2, zero) // Quatre pixels haut de v2 étendus sur 32 bits
f0= _mm_cvtepi32_ps(v4)           // int converti en float
f1= _mm_cvtepi32_ps(v5)           // int converti en float
f2= _mm_cvtepi32_ps(v6)           // int converti en float
f3= _mm_cvtepi32_ps(v7)           // int converti en float
w = 1/255.0                       // constante 1/255.0
c = _mm_set_ps1(w)                // constante 4 * (1/255.0)
f0 = _mm_mul_ps (f0, c)           // valeurs des pixels entre 0.0 et 1.0
f1 = _mm_mul_ps (f1, c)           // valeurs des pixels entre 0.0 et 1.0
f2 = _mm_mul_ps (f2, c)           // valeurs des pixels entre 0.0 et 1.0
f3 = _mm_mul_ps (f3, c)           // valeurs des pixels entre 0.0 et 1.0
```

Annexe 1

Soit un processeur superscalaire à ordonnancement statique qui a les caractéristiques suivantes :
- les instructions sont de longueur fixe (32 bits)
- Il a 32 registres entiers (dont R0=0) de 32 bits et 32 registres flottants (de F0 à F31) de 32 bits.
- Il peut lire et exécuter 4 instructions par cycle.
- L'unité entière contient deux pipelines d'exécution entière sur 32 bits, soit deux additionneurs, deux décaleurs. Tous les bypass possibles sont implantés.
- L'unité flottante contient un pipeline flottant pour l'addition et un pipeline flottant pour la multiplication. - L'unité Load/Store peut exécuter jusqu'à deux chargements par cycle, mais ne peut effectuer qu'un load et un store simultanément. Elle ne peut effectuer qu'un seul store par cycle.
- Il dispose d'un mécanisme de prédiction de branchement qui permet de "brancher" en 1 cycle si la prédiction est correcte. Les sauts et branchements ne sont pas retardés.

La Table 1 donne
- les instructions disponibles
- le pipeline qu'elles utilisent : E0 et E1 sont les deux pipelines entiers, FA est le pipeline flottant de l'addition et FM le pipeline flottant de la multiplication. Les instructions peuvent être exécutées simultanément si elles utilisent chacune un pipeline séparé.

L'addition et la multiplication flottante sont pipelinées. La division flottante n'est pas pipelinée (une division ne peut commencer que lorsque la division précédente est terminée).
L'ordonnancement est statique. Les chargements ne peuvent pas passer devant les rangements en attente.

JEU D'INSTRUCTIONS (extrait)

LF	LF Fi, dép.(Ra)	E0 ou E1	Fi ← M (Ra + dépl.16 bits avec ES)
SF	SF Fi, dép.(Ra)	E0	Fi -> M (Ra + dépl.16 bits avec ES)
ADD	ADD Rd,Ra, Rb	E0 ou E1	Rd ←Ra + Rb
ADDI	ADDI Rd, Ra, IMM	E0 ou E1	Rd ←Ra + IMM-16 bits avec ES
SUB	SUB Rd,Ra, Rb	E0 ou E1	Rd ←Ra - Rb
FADD	FADD Fd, Fa, Fb	FA	Fd ←Fa + Fb
FSUB	FSUB Fd, Fa, Fb	FA	Fd ← Fa - Fb
FMAX	FMAX Fd, Fa, Fb	FA	Fd ← Max(Fa, Fb)
FMIN	FMIN Fd, Fa, Fb	FA	Fd ← Min(Fa, Fb)
FMUL	FMUL Fd, Fa, Fb	FM	Fd ←Fa x Fb
FDIV	FDIV Fd, Fa, Fb	FA	Fd ← Fa/Fb
BEQ	BEQ Ri, dépl	E1	si Ri=0 alors CP ← NCP + depl
BNE	BNE Ri, dépl	E1	si Ri≠0 alors CP ← NCP + depl

Table 1 : instructions disponibles

La Table 2 donne la latence entre une instruction source et une instruction destination, dans le cas de dépendances de données. La valeur 1 est le cas où les deux instructions peuvent se succéder normalement, d'un cycle i au cycle i+1.

Latences ***Source*** / ***Destination***		UAL	LF/SF (données)	FADD/ FMAX/ FMIN	FMUL	FDIV	FSQRT
UAL		1	2				
LF/ST (adresses)		1	3				
SF (données)		1	2	3	4	20 (NP)	20 (NP)
Opération flottante			2	3	4	20 (NP)	20 (NP)

Table 2 : latences

Le processeur a également 32 registres de 128 bits S0 à S7 pouvant contenir chacun 4 flottants simple précision et les instructions SIMD données dans la Table 3. Cette table donne également les latences des instructions SIMD et le pipeline utilisé dans le cas superscalaire.

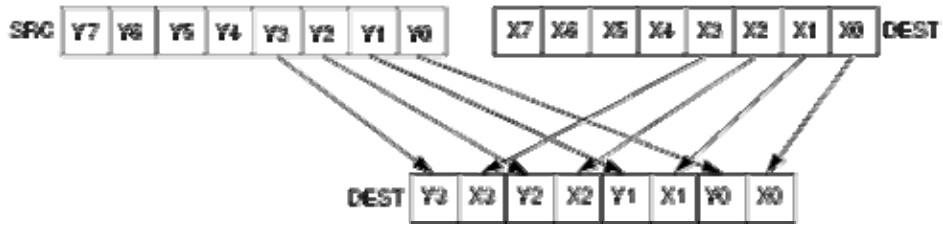
PLF	PLF Si, dép.(Ra)	2	E0	Charge quatre flottants simple précision à partir de l'adresse (Ra + dépl.16 bits avec ES).
PSF	PSF Si, dép.(Ra)	2	E0	Range en mémoire à partir de l'adresse (Ra + dépl.16 bits avec ES) quatre flottants simple précision
PFADD	PFADD Sd,Sa, Sb	3	FA	Sd ← Sa + Sb sur quatre « floats »
PFSUB	PFSUB Sd,Sa, Sb	3	FA	Sd ← Sa – Sb sur quatre « floats »
PFMUL	PFMUL Sd, Sa, Sb	4	FM	Sd ← Sa x Sb sur quatre « floats »
PFMULS	PMULS Sd, Sa, Fb	4	FM	SF ← Sa x Fb (chaque élément de Sa est multiplié par Fb et rangé dans l'élément correspondant de SF) sur quatre « floats »
PLB	PLB Si, dép.(Ra)	2	E0	Charge seize octets à partir de l'adresse (Ra + dépl.16 bits avec ES).
PSB	PSB Si, dép.(Ra)	2	E0	Range en mémoire à partir de l'adresse (Ra + dépl.16 bits avec ES) seize octets
PFMAX	PMAX Sd,Sa, Sb	1	FA	Sd ← Sa max Sb : maximum sur 4 floats
PFMIN	PMIN Sd,Sa, Sb	1	FA	Sd ← Sa min Sb : minimum sur 4 floats

Table 3 : Instructions SIMD

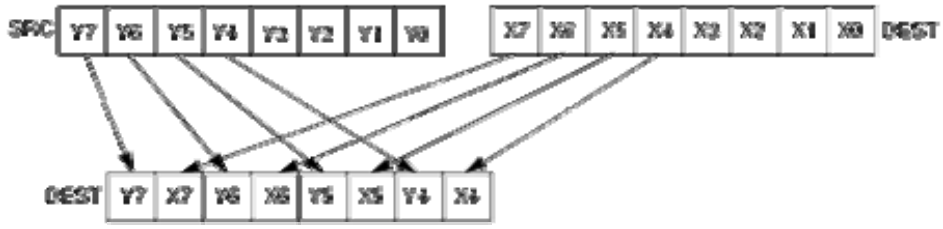
ANNEXE 2 : Instructions SIMD IA-32 utilisables

CVTDQ2PS	_mm_cvtepi32_ps (a)	Convertit 4 entiers 32 bits signés en 32 bits flottants
MULPS	_mm_mul_ps(a,b)	Quatre multiplications flottantes 32 bits
DIVPS	_mm_div_ps (a,b)	Quatre divisions flottantes 32 bits
MOVDQA	_mm_load_si128 (*p)	Chargement aligné de 128 bits
MOVDQA	_mm_store_si128 (*p,a)	Rangement aligné de 128 bits
MULPS	_mm_mul_ps (a, b)	Quatre multiplications flottantes 32 bits
PACKUSWB	_mm_packs_epu16 (m1, m2)	Compacte avec saturation shorts en octets non signés
PMAXUB	_mm_max_epu8 (a, b)	Max des octets non signés source et destination

PMINUB	_mm_min_epu8 (a, b)	Min des octets non signés source et destination
POR	_mm_or_si128 (a, b)	Ou logique parallèle
PSRLDQ	_mm_srli_si128 (a, imm)	Décalage logique gauche de « imm » octets
PSSLDQ	_mm_slli_si128 (a, imm)	Décalage logique droite de « imm » octets
PUNPCKHBW	_mm_unpacklo_epi8 (m1, m2)	Entrelace les octets (haut) de la destination et la source
PUNPCKHWD	_mm_unpacklo_epi16(m1, m2)	Entrelace les shorts (haut) de la destination et la source
PUNPCKLBW	_mm_unpacklo_epi8 (m1, m2)	Entrelace les octets (bas) de la destination et la source
PUNPCKLWD	_mm_unpacklo_epi16 (m1, m2)	Entrelace les shorts (bas) de la destination et la source
PXOR	_mm_xor_si128 (a, b)	Ou exclusif parallèle
	_mm_set_ps1 (float w)	Quatre fois le flottant 32 bits w dans un mot de 128 bits



PUNPCKLBW



PUNPCKHBW