Université d'Angers	Département Informatique
Faculté des Sciences	L3 Informatique, 09/10
Module : *Architecture des Ordinateurs*	Jean-Michel Richer

Examen 2h
Architecture des Ordinateurs
Aucun document autorisé, aucun appareil électronique autorisé

Exercice 1 -(10 pts) - On désire traduire en assembleur, la méthode suivante qui agit sur des tableaux de **float** dont la taille n est un multiple de 4 :

```
float compute(float *x, float *y, float *z, int n) {
  int i;
  float sum1=0.0, sum2=1.0;

  for (i=0; i<n; ++i) {
    x[i] = x[i] + y[i] * z[i];
    sum1 += x[i];
    sum2 *= y[i];
  }
  return sum1 + sum2;
}
```

1. traduire ce sous-programme en assembleur x86 32 bits (on ne donnera que la section de code) en utilisant les instructions du coprocesseur (fld, fldz, fld1, fadd, fmul, fstp). On utilisera :
   - `esi` pour stocker l'adresse de x
   - `edi` pour stocker l'adresse de y
   - `ebx` pour stocker l'adresse de z
   - `ecx` pour la variable de boucle i
2. donner une seconde version utilisant les instructions SSE (mulps, addps, haddps, movss, pshufd, ...). On utilisera :
   - `xmm0` pour stocker $x[i:i+3]$
   - `xmm1` pour stocker $y[i:i+3]$
   - `xmm2` pour stocker $z[i:i+3]$
   - `xmm7` pour stocker $sum1$
   - sommet de la pile du coprocesseur pour stocker $sum2$

   Attention, on ne peut pas réaliser le calcul de $sum2 * y[i]$ avec `mulps` par exemple car le calcul serait faussé. Il faut donc lors du calcul de $som2$ utiliser le coprocesseur. On peut par exemple multiplier $sum2$ successivement par $y[i]$, $y[i+1]$, $y[i+2]$, $y[i+3]$.

Dans la liste suivante qui indique le comportement de quelques instructions SSE (toutes ne sont pas à utiliser), on notera que :

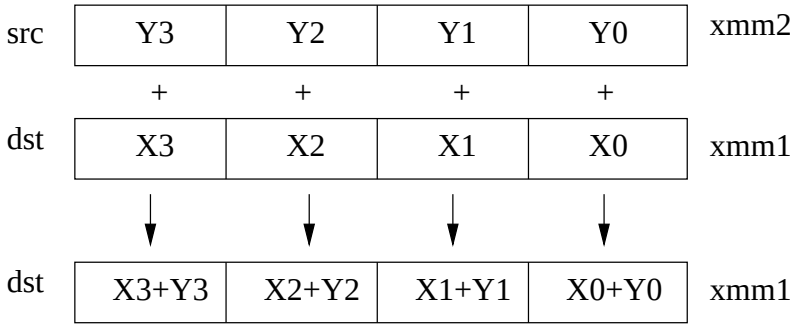
- **xmmDst** et **xmmSrc** représentent l'un des 8 registres $xmm0$ à $xmm7$
- **m32** représente un emplacement mémoire sur 32 bits
- **r32** représente un registre 32 bits
- **imm8** représente une constante entière sur 8 bits

Liste d'instructions FPU

- **FLD m32** : empile dans la pile du coprocesseur la valeur située à l'adresse donnée
- **FLDZ** : empile la valeur 0.0
- **FLD1** : empile la valeur 1.0
- **FST m32** : stocke la valeur au sommet de la pile à l'adresse donnée
- **FSTP m32** : stocke la valeur au sommet de la pile à l'adresse donnée, puis la dépile
- **FADD m32** : additionne la valeur à l'adresse donnée à celle du sommet de pile
- **FADD ST0,STi** : additionne à la valeur en sommet de pile, le registre du coprocesseur spécifié par i
- **FMUL m32** : multiplie le sommet de pile par la valeur réelle située à l'adresse donnée
- **FMUL ST0,STi** multiplie la valeur en sommet de pile par celle du registre du coprocesseur spécifié par i

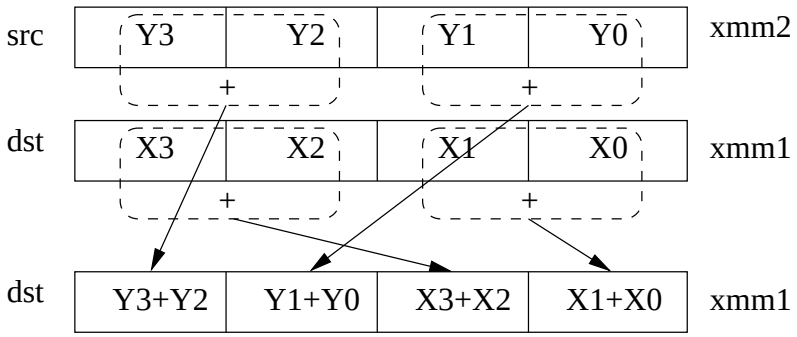
Liste d'instructions SSE

- **MOVAPS xmmDst/m32, xmmSrc** : charge les données réelles alignées contenues dans un registre SSE (xmmSrc) vers soit un emplacement mémoire (adresse multiple de 16) ou un registre SSE (xmmDst)
- **MOVAPS xmmDst, xmmSrc/m32** : charge les données réelles alignées contenues soit dans un registre SSE (xmmSrc) ou un emplacement mémoire (adresse multiple de 16) vers un registre SSE (xmmDst)
- **MOVUPS xmmDst/m32, xmmSrc** : charge les données réelles non alignées contenues dans un registre SSE (xmmSrc) vers soit un emplacement mémoire ou un registre SSE (xmmDst)
- **MOVUPS xmmDst, xmmSrc/m32** : charge les données réelles non alignées contenues soit dans un registre SSE (xmmSrc) ou un emplacement mémoire vers un registre SSE (xmmDst)
- **ADDPS xmmDst, xmmSrc** : réalise l'addition en parallèle de deux registres SSE en considérant que l'on traite 4 réels



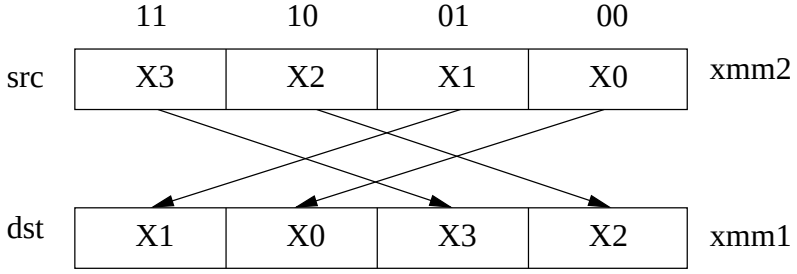
addps xmm1,xmm2

- **MULPS xmmDst, xmmSrc** : réalise la multiplication en parallèle de deux registres SSE en considérant que l'on traite 4 réels
- **HADDPS xmmDst, xmmSrc** : addition *horizontale* sur des réels



haddps xmm1,xmm2

- **MOVSS m32, xmmSrc** : copie les 32 bits de poids faible du registre xmmSrc vers une adresse mémoire
- **MOVSS xmmDst, m32** : copie les 32 bits situés à l'adresse donnée dans le registre xmmDst, les bits 33 à 127 du registre SSE sont mis à 0
- **PSHUFD (ou SHUFPS) xmmDst, xmmSrc, imm8** : copie les valeurs sur 32 bits du registre source dans le registre destination en indiquant lesquels choisir



pshufd xmm1,xmm2,01001110b

Exercice 2 -(5 pts) - Questions de cours - Expliquez brièvement en une dizaine de lignes (qu'est ce que c'est, à quoi ça sert, comment ça fonctionne) vous pouvez vous aider d'un schéma :

1. qu'est ce qu'un processeur superscalaire ?
2. qu'est ce que la technologie du pipeline et à quoi sert-elle ?

Exercice 3 -(5 pts) - On considère le circuit qui réalise la soustraction de deux nombres binaire A et B représentés respectivement sur 3 et 2 bits :

$$\begin{array}{cccc} & A_2 & A_1 & A_0 \\ & - & B_1 & B_0 \\ \hline C_3 & C_2 & C_1 & C_0 \end{array}$$

On rappelle que la table de vérité d'un soustracteur est :

R_e	X	Y	$S(R_e, X, Y)$	$R_s(R_e, X, Y)$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

où R_e est la retenue en entrée et R_s la retenue en sortie.

1. donnez l'expression algébrique de $S(R_e, X, Y)$ et $R_s(R_e, X, Y)$ en fonction de R_e, X, Y en simplifiant au maximum (remarque : on utilisera des portes XOR quand cela est possible)
2. donnez les expressions de C_0 à C_3 en fonction de A_2 à A_0 : par exmple, $C_0 = S(A_0, B_0, 0) = \ldots$
3. donnez les valeurs de C_3 à C_0 dans le cas où $A = 001$ et $B = 10$