Université d'Angers	Département Informatique
Faculté des Sciences	L3 Informatique, 2010-2011
Module : *Architecture des Ordinateurs*	Jean-Michel Richer

Examen 2h

Architecture des Ordinateurs

Aucun document autorisé, aucun appareil électronique autorisé

Exercice 1 -(2 pts) - Qu'est ce que la phrase "programmer un ordinateur" signifie pour vous ? Répondre en une dizaine de lignes maximum.

Exercice 2 -(8 pts) - On désire écrire une fonction qui compare des chaînes de caractères suivant leurs longueurs. La fonction devra retrourner un entier :

- égal à 0, si les deux chaînes sont de la même longueur
- inférieur à 0, si la longueur première chaîne est inférieure à la seconde
- supérieur à 0, si la longueur première chaîne est supérieure à la seconde

La traduction la plus simple de cette fonction est :

```
int strlencmp(unsigned char *s, unsigned char *t) {
  return strlen(s) - strlen(t);
}
```

Cependant, pour des raisons d'efficacité, on désire écrire une version *optimisée* comme suit :

```
int strlencmp_opt(unsigned char *s, unsigned char *t) {
  int i=0;
  while ((s[i]!='\0') && (t[i]!='\0')) ++i;
  return ((int)s[i]) - ((int)t[i]);
}
```

1. traduire `strlencmp_opt` en assembleur x86 32 bits (on ne donnera que la section de code), indiquez ce que stockent les registres, par exemple on pourra choisir `ecx` pour représenter la variable de boucle i,
2. donnez une version de `strlencmp_opt` utilisant les registres SSE, on supposera que `s` et `t` sont alignés sur une adresse multiple de 16 et qu'il n'y a pas de débordement mémoire, c'est à dire qu'une chaîne de 33 octets est contenue dans un bloc de mémoire alloué de 48 (3×16) octets. Le pseudo code de la fonction dans sa version SSE ressemble à celà :

```
int i=0;
for(;;) {
 j = la premiere valeur de [i:i+15] telle que s[j]!='\0'
 k = la premiere valeur de [i:i+15] telle que t[k]!='\0'
  if (j!=0) {
    if (k!=0) return j - k; else return -1;
  }
  if (k!=0) {
    if (j!=0) return j - k; else return +1;
  }
  i=i+16;
}
```

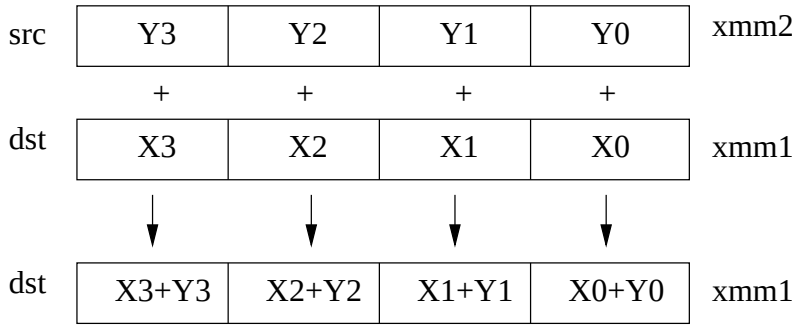
Dans la liste suivante qui indique le comportement de quelques instructions, on notera que :
- **xmmDst** et **xmmSrc** représentent l'un des 8 registres $xmm0$ à $xmm7$
- **m32** représente un emplacement mémoire sur 32 bits
- **r32**, **r32S**, **r32D** représentent un registre 32 bits
- **imm8** représente une constante entière sur 8 bits

Liste d'instructions classiques

- **MOV r32, [m32]** : met dans le regitre la valeur contenue à l'adresse indiquée
- **MOV [m32], r32** : met à l'adresse indiquée la valeur contenue dans le registre
- **CMP r32D, r32S** : compare 2 registres 32 bits et ne modifie pas le contenu des registres
- **JE/JNE/JG/JL m32** : saut à l'adresse si la dernière comparaison donne une égalité/ une inégalité/supériorité/ infériorité
- **PUSH r32** : mettre le contenu d'un registre en sommet de pile
- **POP r32** : mettre le sommet de pile dans un registre
- **BSF r32D, r32S** (Bit Scan Forward) : met dans le registre r32D la valeur du premier bit de r32S qui n'est pas 0. Par exemple si en binaire r32S = $...10110000_2$, alors r32D reçoit la valeur 4

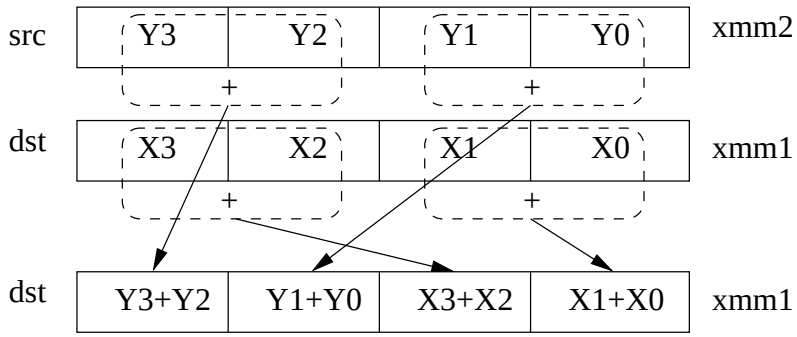
Liste d'instructions SSE

- **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



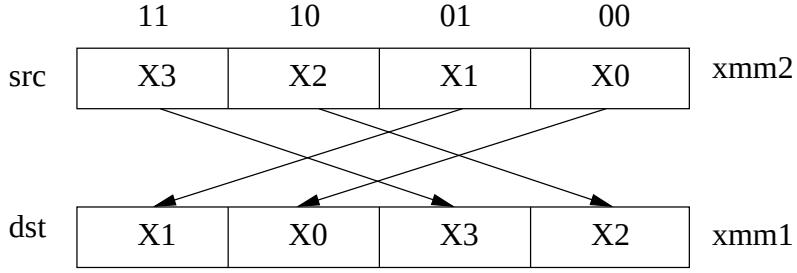
addps xmm1,xmm2

- **PCMPEQB xmmDst, xmmSrc** : compare 2 registres SSE octect par octet et fixe l'octet xmmDst[i] à FF_{16}, si initialement xmmDst[i] == xmmSrc[i], sinon on lui donne la valeur 00_{16}. i varie ici de 0 à 15.
- **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 en particulier **PSHUFD xmm0,xmm0,0** recopie xmm0[0] dans xmm0[1 :3].



pshufd xmm1,xmm2,01001110b

- **PMOVMSKB r32, xmmSrc** : copie les bits de poids fort des 16 octets de xmmSrc et les place dans les 16 premiers bits de r32, les 16 bits de poids fort de r32 sont mis à 0

Exercice 3 -(5 pts) - Questions de cours - Expliquez brièvement (en une dizaine de lignes) à quelqu'un qui ne connaîtrait pas ces notions (qu'est ce que c'est, à quoi ça sert, comment ça fonctionne, qu'est ce que cela apporte), ce que sont :

1. le dépliage de boucle
2. le pipeline

Vous pouvez vous aider d'un schéma.

Exercice 4 -(5 pts) - On considère la fonction booléenne de 4 variables $f(A, B, C, D) = (0 - 3, 8, 9, 12 - 15)$

1. donnez l'expression algébrique de $F(A, B, C, D)$ et la simplifier avec un tableau de Karnaugh
2. donner l'expression algébrique de la fonction booléenne $prem(A, B, C, D)$ qui vaut 1 si le nombre en représentation binaire $ABCD$ est un nombre premier (**Attention** : on considérera que 1 est premier).
3. donnez l'expression simplifiée du circuit sous forme algébrique (vous pouvez utiliser la fonction XOR)