Le calcul flottant avec le processeur intel 64 bits

©Sovanna Tan

Novembre 2013

L'unité de calcul flottant (FPU : Floating Point Unit)

- Au début, elle était un circuit séparé, le coprocesseur mathématique.
- Puis, elle a été intégrée au microprocesseur, mais se programmait toujours de façon distincte.
- Avec les architectures 64 bits, apparaît un nouveau jeu d'instructions SSE (Streaming SIMD (Single Instruction Multiple Data) Extensions) pour des registres de taille 128 ou 256 bits selon les processeurs.

Les types de nombres flottants

- Simple précision IEEE 754-2008 : 1 bit signe, 8 bits exposant, 23 bits mantisse
- Double précision IEEE 754-2008 : 1 bit signe, 11 bits exposant, 52 bits mantisse
- Precision étendue intel 80 bits : 1 bit signe, 15 bits exposant, 64 bits mantisse
  - Ce format est différent des formats IEEE 754-2008.
  - Dans le processeur Intel 32 bits, tous les calculs étaient effectués en précision étendue puis convertis en simple ou double précision IEEE 754-2008.
- Quadruple précision IEEE 754-2008 : 1 bit signe, 15 bits exposant, 112 bits mantisse
- 256 bits (extrapolation de IEEE 754-2008) : 1 bit signe, 19 bits exposant, 236 bits mantisse

Les registres

- 16 registres généraux **xmm0**, **xmm1**,... et **xmm15**.
- Ces registres peuvent être utilisés avec des instructions
  - pour une seule valeur
  - ou pour un vecteur de valeurs.

  Ce vecteur comporte alors soit 4 **float**s soit 2 **double**s.
- Les processeurs de la série `i-core` ont 16 registres 256 bits **ymm0**, **ymm1**,... et **ymm15**. Les registres **xmm0**, **xmm1**,... et **xmm15** correspondent aux premiers 128 bits des registres **ymm0**, **ymm1**,... et **ymm15**. Ces derniers peuvent être vu comme des vecteurs de 8 **float**s ou 4 **double**s.

Les instructions de déplacement de données

- Déplacement d'une seule valeur

movsd *dest,src*	déplacement d'un flottant **double**
movss *dest,src*	déplacement d'un flottant simple (**float**)

- Déplacement de paquets (plusieurs valeurs simultanément)

movapd *dest,src*	déplacement de 2 **double**s alignés [1]
movaps *dest,src*	déplacement de 4 **float**s alignés
movdqa *dest,src*	déplacement d'un **double qword** aligné
movupd *dest,src*	déplacement de 2 **double**s non alignés
movups *dest,src*	déplacement de 4 **float**s non alignés
movdqu *dest,src*	déplacement d'un **double qword** non aligné

- Les noms des instructions pour les registres 256 bits commencent par `v`, **vmovapd**, **vmovaps**,...

1. alignés sur 1 bloc de 16 octets

Calculs flottants

addsd *dest,src*	addition de 2 **double**s
addss *dest,src*	addition de 2 **float**s
addpd *dest,src*	2 **double**s + 2 **double**s
addps *dest,src*	4 **float**s + 4 **float**s
subsd *dest,src*	soustraction de 2 **double**s
subss *dest,src*	soustraction de 2 **float**s
subpd *dest,src*	2 **double**s - 2 **double**s
subps *dest,src*	4 **float**s - 4 **float**s
mulsd *dest,src*	multiplication de 2 **double**s
mulss *dest,src*	multiplication de 2 **float**s
mulpd *dest,src*	2 **double**s * 2 **double**s
mulps *dest,src*	4 **float**s * 4 **float**s
divsd *dest,src*	division de 2 **double**s
divss *dest,src*	division de 2 **float**s
divpd *dest,src*	2 **double**s / 2 **double**s
divps *dest,src*	4 **float**s / 4 **float**s

- *dest* doit être un registre flottant.

Les instructions de conversion

cvtss2sd *dest,src*	convertit un **float** en **double**
cvtps2pd *dest,src*	convertit un paquet de 2 **float**s en un paquet de 2 **double**s
cvtsd2ss *dest,src*	convertit un **double** en **float**
cvtpd2ps *dest,src*	convertit un paquet de 2 **double**s en un paquet de 2 **float**s
cvtss2si *dest,src*	convertit un **float** en **dword** ou **qword**
cvtsd2si *dest,src*	convertit un **double** en **dword** ou **qword**
cvtsi2ss *dest,src*	convertit un entier en **float**
cvtsi2sd *dest,src*	convertit un entier en **double**

- Lorsque l'opérande source est dans la mémoire, il faut préciser la taille de la donnée à lire.

Les instructions de comparaison

comisd *op1*,*op2*	comparaison ordonnée de deux **double**s
comiss *op1*,*op2*	comparaison ordonnée de deux **float**s
ucomisd *op1*,*op2*	comparaison non ordonnée de deux **double**s
ucomiss *op1*,*op2*	comparaison non ordonnée de deux **float**s

- *op1* doit être un registre, ,*op2* peut être un registre ou une valeur en mémoire.
- Les instructions mettent à jour les drapeaux ZF, PF et CF.
- Les comparaisons non ordonnées lèvent une exception si l'un des opérandes vaut SNaN (Signaling Not a Number i.e. que des 1 dans l'exposant et mantisse de la forme $0x$) et les comparaisons ordonnées lèvent une exception si l'un des deux opérandes vaut SNaN ou QNaN (Quiet Not a Number i.e. que des 1 dans l'exposant et mantisse de la forme $1x$).

Les branchements conditionnels pour les flottants

- Après une instruction de comparaison de flottants (**comisd**, **comiss**, **ucomisd** ou **ucomiss**) entre *op1* et *op2* :

jb *op*	branchement à *op* si $op1 < op2$	CF=1
jbe *op*	branchement à *op* si $op1 \leq op2$	CF=1 ou ZF=1
ja *op*	branchement à *op* si $op1 > op2$	CF=1
jae *op*	branchement à *op* si $op1 \geq op2$	CF=1 ou ZF=1

Arrondis

roundsd *dest,src,mode*	arrondi *src* de type **double** dans *dest*
roundss *dest,src,mode*	arrondi *src* de type **float** dans *dest*
roundpd *dest,src,mode*	arrondi des 2 **double**s de *src* dans *dest*
roundps *dest,src,mode*	arrondis des 4 **float**s de *src* dans *dest*

- *dest* doit être un registre flottant.
- *mode* est un opérande immédiat :

0	arrondi à la valeur entière paire la plus proche
1	arrondi à la valeur entière inférieure
2	arrondi à la valeur supérieure
3	arrondi à la valeur absolue inférieure (à la valeur entière la plus proche de 0)

Instructions diverses

minsd *dest,src*	minimun de 2 **double**s (*src* et *dest*) dans *dest*
minss *dest,src*	minimun de 2 **float**s (*src* et *dest*) dans *dest*
minpd *dest,src*	2 calculs de minimun de 2 **double**s dans *dest*
minps *dest,src*	4 calculs de minimun de 2 **float**s dans *dest*
maxsd *dest,src*	maximun de 2 **double**s (*src* et *dest*) dans *dest*
maxss *dest,src*	maximun de 2 **float**s (*src* et *dest*) dans *dest*
maxpd *dest,src*	2 calculs de maximun de 2 **double**s dans *dest*
maxps *dest,src*	4 calculs de maximun de 2 **float**s dans *dest*
sqrtsd *dest,src*	racine carrée de *src* de type **double** dans *dest*
sqrtss *dest,src*	racine carrée de *src* de type **float** dans *dest*
sqrtpd *dest,src*	2 racines carrées de *src*(2 **double**s) dans *dest*
sqrtps *dest,src*	4 racines carrées de *src*(4 **float**s) dans *dest*

- *dest* doit être un registre flottant.

Exemple : calcul de la valeur d'un polynôme (1)

```
extern printf, scanf
        segment .data
coeff   dq      1.0, 2.0, 3.0, 4.0
x       dq      2.0
promptDouble db "%lf",0
prompt1DoubleIn db "4x3+3x2+2x+1 (2) = %lf",10,0
prompt2DoubleIn db "4x3+3x2+2x+1 (%lf) = %lf",10,0
promptEntrer db "Entrez x : ",0
        segment .bss
y       resq 1
        segment .text
        global  asm_main, horner
asm_main:
        push    rbp
        mov     rbp, rsp
        lea     rdi, [coeff]
        movsd   xmm0, [x]
        mov     esi, 3
        call    horner
        mov rdi,prompt1DoubleIn
        mov rax,1
        call printf
        mov rdi,promptEntrer
        mov rax,0
        call printf
        mov rdi,promptDouble
        mov rsi,y
        mov rax,0
        call scanf
        lea     rdi, [coeff]
        movsd   xmm0, [y]
        mov     esi, 3
        call    horner
        mov rdi,prompt2DoubleIn
        movsd   xmm1,xmm0
        movsd   xmm0, [y]
        mov rax,2
        call printf
        xor     eax, eax
        leave
        ret
; extrait du livre de Ray Seyfarth
horner: movsd   xmm1, xmm0
; use xmm1 as x
        movsd   xmm0, [rdi+rsi*8]
; accumulator for b_k
        cmp     esi, 0
; is the degree 0?
        jz      done
        more:   sub     esi, 1
        mulsd   xmm0, xmm1
; b_k * x
        addsd   xmm0, [rdi+rsi*8]
; add p_k
        jnz     more
done:   ret
```

Exemple : calcul de la valeur d'un polynôme (2)

```
./horner
4x3+3x2+2x+1 (2) = 49.000000
Entrez x : 3
4x3+3x2+2x+1 (3.000000) = 142.000000

./horner
4x3+3x2+2x+1 (2) = 49.000000
Entrez x : 1.5
4x3+3x2+2x+1 (1.500000) = 24.250000
```

Algorithme :

- $P(x) = (...(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})...)x + a_0$

Exemple : calcul de la somme de deux vecteurs (1)

```
extern printf
        segment .data
a       dq 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0
b       dq 2.0, 0.5, 4.0, 0.2, 3.0, 1.5, 0.5, 0.4
floatFormat db "%lf ",0
promptRes db "a*b=",0
promptln db 10,0
        segment .bss
c       resq 8
        segment .text
        global asm_main, horner
asm_main:
        push    rbp
        mov     rbp, rsp
        mov rcx,0
        mov rdx,8
        jmp test1
next1:  movapd  xmm0,[a+8*rcx]
        movapd  xmm1,[b+8*rcx]
        addpd   xmm1,xmm0
        movapd  [c+8*rcx],xmm1
        add rcx,2
test1:  cmp rcx,rdx
        jl next1

        mov     rdi,a
        mov     rsi, 8
        call    affichage
        mov     rdi,b
        mov     rsi, 8
        call    affichage
        mov     rdi,c
        mov     rsi, 8
        call    affichage
        mov     rax,0
        leave
        ret

affichage:
        enter 0,0
        push r12
        push r13
        push r14
        push rbx

        mov rbx,rdi
        mov r13,rsi
        mov r12,0
        jmp test3
next3:  mov rdi,floatFormat
        movsd xmm0,[rbx+8*r12]
        mov rax,1
        call printf
        inc r12
test3:  cmp r12,r13
        jl next3
```

Exemple : calcul de la somme de deux vecteurs (2)

```
        mov rdi, promptln
        mov rax, 0
        call    printf

        pop rbx
        pop r14
        pop r13
        pop r12
        mov rax, 0
        leave
        ret
```

```
./addvect
1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000
2.000000 0.500000 4.000000 0.200000 3.000000 1.500000 0.500000 0.400000
3.000000 2.500000 7.000000 4.200000 8.000000 7.500000 7.500000 8.400000
```