

Liste d'exercices 4 : les entiers et les flottants à simple précision (1 séance)

Notions à considérer pour pouvoir faire les exercices

Priorité des opérateurs :

<i>Priorités</i>	<i>Opérateurs en C</i>
+++++++	()
+++++++	-, ~, (changement de nature), (casting)
++++++	*, /, %
+++++	+, -
++++	&
+++	^
++	
+	=

Les opérateurs logiques ne peuvent être appliqués que sur des entiers.

Le type de données float :

<i>Types</i>	<i>Tailles</i>	<i>Plage de valeurs acceptée</i>
float	32 bits	$[1.175494351 * 10^{-38}, \dots, 3.402823466 * 10^{38}]$

Les instructions de traitement sur les flottants :

Intel a conçu, au cours du temps, 3 générations distinctes d'unités dans ses processeurs pouvant effectuer des traitements sur les flottants :

1. **FPU** (Floating-Point Unit) : apparue en 1981, une unité FPU a un fonctionnement basé sur une pile interne et non sur des registres. Par exemple, pour réaliser une addition, les 2 nombres figurant au sommet de la pile interne à la FPU sont dépilés, le calcul est réalisé, puis le résultat est empilé.
2. **SSE** (Streaming SIMD (Single Instruction, Multiple Data) Extensions) : apparue avec le Pentium III en 1999, une unité SSE a un fonctionnement classique basé sur des registres.
3. **AVX** (Advanced Vector Extensions) : apparue avec la microarchitecture Sandy Bridge en 2011, une unité AVX est une unité SSE améliorée permettant, notamment, l'usage d'instructions ayant plus de 2 opérandes.

Exemple: voici 3 façons d'écrire en assembleur le code qui correspond à une même instruction d'affectation en langage C :

`float f = 52.5, f2 = 2, f3 = 3;`

<i>Langage C</i>	<i>Code pour FPU</i>	<i>Code pour SSE</i>	<i>Code pour AVX</i>
<code>f = f + f / f2 + f / f3;</code>	<code>fld f</code> <code>fld f2</code> <code>fdiv</code> <code>fld f</code> <code>fld f3</code> <code>fdiv</code> <code>fld f</code> <code>fadd</code> <code>fadd</code> <code>fstp f</code>	<code>movss xmm0, f</code> <code>divss xmm0, f2</code> <code>movss xmm1, f</code> <code>divss xmm1, f3</code> <code>movss xmm2, f</code> <code>addss xmm2, xmm0</code> <code>addss xmm2, xmm1</code> <code>movss f, xmm2</code>	<code>vmovss xmm0, f</code> <code>vdivss xmm1, xmm0, f2</code> <code>vdivss xmm2, xmm0, f3</code> <code>vaddss xmm1, xmm0, xmm1</code> <code>vaddss xmm2, xmm1, xmm2</code> <code>vmovss f, xmm2</code>

Dans ce cours, on utilise les instructions SSE pour faire les traitements sur les flottants.

Registres et instructions pour les traitements sur les flottants à simple précision (type float):

- Registres pour le stockage temporaire de flottants: XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7.
- Instructions permettant des opérations sur les flottants à simple précision:
 - *MOVSS opd, ops* effectue $opd = ops$.
 - *ADDSS opd, ops* effectue $opd = opd + ops$.
 - *SUBSS opd, ops* effectue $opd = opd - ops$.
 - *MULSS opd, ops* effectue $opd = opd * ops$.
 - *DIVSS opd, ops* effectue $opd = opd / ops$.

L'opérande *opd* doit être un registre XMM. L'opérande *ops* peut être un registre XMM ou un emplacement en mémoire du type float.

- Instructions permettant de convertir des nombres d'un type de données vers un autre:
 - *CVTTSS2SI opd, ops* copie dans *opd* la partie entière du flottant à simple précision se trouvant dans *ops*. L'opérande *opd* doit être un registre général de 32 bits.
 - *CVTSI2SS opd, ops* convertit l'entier se trouvant dans *ops* en un flottant à simple précision et copie ce flottant dans *opd*. L'opérande *opd* doit être un registre XMM.

Changer le signe d'un flottant :

L'instruction *neg* n'existe pas avec les flottants. Il faut multiplier le nombre par -1 avec *mulss* pour changer son signe.

Exemple:

```
float f;  
float f1 = 10;  
const float fconst = -1;
```

Langage C	Assembleur
f = -f1;	movss xmm0, f1 mulss xmm0, fconst movss f, xmm0

Les instructions SSE n'acceptent pas une valeur immédiate, ici la valeur -1, comme opérande source. Ceci explique pourquoi la constante *fconst* contenant la valeur -1 est utilisée dans l'instruction *mulss*.

Nouvelle contrainte du langage C : quand on a un entier avec un flottant à simple précision dans une opération, l'entier est transformé en un flottant à simple précision et l'opération est réalisée sur des flottants à simple précision.

Exemple:

```
char bVar;  
int iVar;  
float fVar;  
const float f3 = 3;
```

Langage C	Assembleur
$fVar = iVar - bVar + bVar * fVar + f3;$ <p>The diagram illustrates the evaluation of the expression $fVar = iVar - bVar + bVar * fVar + f3;$ in C. It shows the flow of data and operations with numbered steps (1) through (9) and conversion annotations. Step (1) is the multiplication $bVar * fVar$. Step (2) is the conversion of $bVar$ to float. Step (3) is the multiplication of the converted $bVar$ with $fVar$. Step (4) is the extension of $bVar$ to signed integer. Step (5) is the subtraction of the extended $bVar$ from $iVar$. Step (6) is the conversion of $iVar$ to float. Step (7) is the addition of the result of step (3) to the result of step (5). Step (8) is the addition of $f3$ to the result of step (7). Step (9) is the final assignment to $fVar$.</p>	<pre>movsx eax, bVar // (1) et (4) cvtsi2ss xmm0, eax // (2) mulss xmm0, fVar // (3) mov ebx, iVar sub ebx, eax // (5) cvtsi2ss xmm1, ebx // (6) addss xmm1, xmm0 // (7) addss xmm1, f3 // (8) movss fVar, xmm1 // (9)</pre>

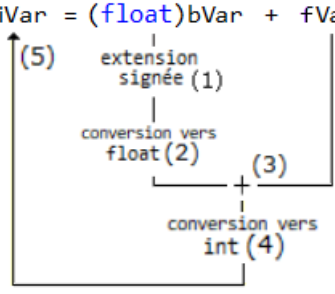
Dès la multiplication entre *bVar* et *fVar*, le type *float* s'impose et tout les traitements qui suivent sont réalisés sur des flottants à simple précision.

Les castings explicites :

- Casting d'entier vers flottant à simple précision:

Exemple:

```
char bVar;  
int iVar;  
float fVar;
```

<i>Langage C</i>	<i>Assembleur</i>
<pre>iVar = (float)bVar + fVar;</pre> 	<pre>movsx eax, bVar // (1) cvtsi2ss xmm0, eax // (2) addss xmm0, fVar // (3) cvtts2si ebx, xmm0 // (4) mov iVar, ebx // (5)</pre>

Tout d'abord, la valeur de bVar est étendue à 32 bits. Ensuite, l'application du casting (float) entraîne la conversion de la valeur du type int vers le type float. L'addition est alors réalisée sur 2 flottants à simple précision. Enfin, la somme est convertie vers le type int et est copiée dans iVar.

- Casting de flottant à simple précision vers entier:

Exemple:

```
int iVar;  
float fVar;
```

<i>Langage C</i>	<i>Assembleur</i>
<pre>iVar = (int)fVar + 3;</pre>	<pre>cvtts2si eax, fVar add eax, 3 mov iVar, eax</pre>

Exercices

Écrivez la séquence d'instructions en assembleur qui correspond à chaque instruction d'affectation en langage C suivante (en commentaire en vert figure la valeur à obtenir au final) :

```
char    b = 40;
int     i = 20;
int     j = -125;
float   f = 2.7;
const float f281 = 2.81, f35 = 3.5;
```

- `b = f;` // b = 2
- `f = (int)f281 + i;` // f = 22.0
- `i = j + ~(int)f;` // i = -128
- `b = j / 2 + -f + f35 / 2;` // b = -62
- `j = ~(j ^ 0xffff) + -(int)f + f35;` // j = 3972
- `i = (float)i * j;` // i = -2500
- `i = i * 2 + f - 5 * b;` // i = -157
- `f = (f * 100) / (b - 20 | 0xf0);` // f = 1.10655737
- `f = ((int)f * 100) / (~b - 20);` // f = -3.00000000
- `f = b * (i & 0x0f | 0x1f00) - f;` // f = 317597.313
- `f = (-i + f281 - -b * f + f35) / 2;` // f = 47.1549988
- `f = b / 2 + i / f + (b - 3);` // f = 64.4074097
- `j = f + (0xf2 | i) / (b - 32) / -f;` // j = -8