Liste d'exercices 6 : les entiers, les flottants à simple précision et les flottants à double précision (1 séance)

Notions à considérer pour pouvoir faire les exercices

Les opérateurs ++ et -- :

Ces 2 opérateurs ne se comportent pas comme les autres opérateurs :

- Ils sont appliqués sur la variable directement et non sur la copie dans un registre de la valeur de la variable, comme c'est le cas avec les autres opérateurs (sauf l'opérateur d'affectation =).
- Ils peuvent se trouver avant ou après une variable :
  - Pré-incrementation/pré-décrémentation : l'opérateur ++ / -- se trouve avant une variable et dans ce cas cette variable est modifiée avant l'évaluation de l'instruction d'affectation.
  - Post-incrementation/post-décrémentation : l'opérateur ++ / -- se trouve après une variable et dans ce cas cette variable est modifiée après l'évaluation de l'instruction d'affectation.

Les opérateurs ++ et -- avec les entiers :

Instructions :

- *INC op* augmente de 1 unité le contenu de l'opérande *op.*
- *DEC op* diminue de 1 unité le contenu de l'opérande *op.*

Exemple:

```
char b, b1;
int  i;
```

Langage C	*Assembleur*

`i = b1-- * ++b;`	`inc   b`
	`movsx eax, b1`
	`movsx ecx, b`
	`imul  eax, ecx`
	`mov   i, eax`
	`dec   b1`

L'opérateur ++ placé avant la variable b entraîne l'incrémentation du contenu de cette variable au tout début. L'opérateur -- placé après la variable b1 entraîne la décrémentation du contenu de cette variable tout à la fin.

Les opérateurs ++ et -- avec les flottants à simple précision :

Avec les flottants à simple précision, il n'y a pas d'instructions équivalentes à inc et dec. Pour incrémenter ou décrémenter, il faut utiliser addss ou subss.

Exemple:

```
float f, f1;
const float fconst1 = 1;
```

Langage C	*Assembleur*

`f = f1++;`	`movss xmm0, f1`
	`movss f, xmm0`
	`addss xmm0, fconst1`
	`movss f1, xmm0`

`f = ++f1;`	`movss xmm0, f1`
	`addss xmm0, fconst1`
	`movss f1, xmm0`
	`movss f, xmm0`

Pour rappel, les instructions sur flottants n'acceptent aucune valeur immédiate comme opérande source. Ceci explique pourquoi la constante fconst1 est utilisée.

Les opérateurs ++ et -- avec les flottants à double précision :

Exemple:

```
double d, d1;
const double dconst1 = 1, dconst55 = 5.5;
```

Langage C	*Assembleur*

`d = d1++ * 5.5;`	`movsd xmm0, d1`
	`mulsd xmm0, dconst55`
	`movsd d, xmm0`
	`movsd xmm0, d1`
	`addsd xmm0, dconst1`
	`movsd d1, xmm0`

`d = ++d1 * 5.5;`	`movsd xmm0, d1`
	`addsd xmm0, dconst1`
	`movsd d1, xmm0`
	`mulsd xmm0, dconst55`
	`movsd d, xmm0`

Exercices

Écrivez la séquence d'instructions en assembleur qui correspond à chaque instruction d'affectation en langage C suivante :

```
char   b = -25;
short  s = 40;
int    i = -110, j = -125;
float  f = 212.9;
double d = 6.48;
const float f251 = 2.51, f85 = -8.5;
```

- `i = ++s + 2 + ++s;` `// i = 86, s = 42`
- `i = s++ + 2 + s++;` `// i = 82, s = 42`
- `s = -i++ + -++b + s;` `// s = 174, i = -109, b = -24`
- `i = i++ * b * i + i * (b++ + s) * --i;` `// i = -123209, b = -24`
- `d = (++i + b) * -(--s + i);` `// d = -9380.0, i = -109, s = 39`
- `f = -f-- * - --f;` `// f = 44900.6055`
- `f = -(0x12 - i + ++b) * 5 / 2.4;` `// f = -216.666672, b = -24`
- `f = (int)++f + i;` `// f = 103.0`
- `f = i * (float)d-- * (int)--d;` `// f = -3014.0, d = 4.48000…`
- `i = (double)--j + ~(int)f--;` `// i = -339, j = -126, f = 211.899994`
- `i = -j / 2 + --f + (int)f85 / 2.3;` `// i = 270, f = 211.899994`
- `f = --f + (0xf2 | i++) * (double)--b * 1 / -f;` `// f = 210.182205, i = -109, b = -26`
- `d = ((int)f + 0.5 * i) / (float)b + (int)++f;` `// d = 206.6800..., f = 213.899994`

Pour simplifier la résolution de chaque exercice :

1. prendre en compte l'instruction d'affectation comme si elle ne contenait aucun des opérateurs `++` et `- -`, exactement comme avec les exercices des listes d'exercices précédentes.

2. Cela fait, prendre alors en compte les opérateurs `++` et `--`.

Rappel : les valeurs numériques dans les expressions qui ont une partie décimale, par exemple la valeur 5.2, sont du type double, tandis que les valeurs numériques qui n'ont aucune partie décimale, par exemple la valeur 3, sont du type int.