Liste d'exercices 5 : les entiers, les flottants à simple précision et les flottants à double précision (1 séance)

Notions à considérer pour pouvoir faire les exercices

Le type de données double :

Types	*Tailles*	*Plage de valeurs acceptée*
double	64 bits	[2.2250738585072014 * 10^{-308}, ..., 1.7976931348623158 * 10^{308}]

Registres et instructions liés aux traitements sur les flottants à double précision (type double):

- Registres pour le stockage temporaire de flottants: XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6, XMM7.
- Instructions permettant des opérations sur les flottants à double précision:
  - *MOVSD opd, ops* effectue opd = ops.
  - *ADDSD opd, ops* effectue opd = opd + ops.
  - *SUBSD opd, ops* effectue opd = opd – ops.
  - *MULSD opd, ops* effectue opd = opd * ops.
  - *DIVSD opd, ops* effectue opd = opd / ops.
- Instructions additionnelles permettant de convertir des nombres d'un type de données vers un autre :
  - *CVTTSD2SI opd, ops* copie dans *opd* la partie entière du flottant à double précision se trouvant dans *ops*. L'opérande *opd* doit être un registre général de 32 bits.
  - *CVTSI2SD opd, ops* convertit l'entier se trouvant dans *ops* en un flottant à double précision et copie ce flottant dans *opd*. L'opérande *opd* doit être un registre XMM.
  - *CVTSD2SS opd, ops* convertit le flottant à double précision se trouvant dans *ops* en un flottant à simple précision et copie celui-ci dans *opd*. L'opérande *opd* doit être un registre XMM.
  - *CVTSS2SD opd, ops* convertit le flottant à simple précision se trouvant dans *ops* en un flottant à double précision et copie celui-ci dans *opd*. L'opérande *opd* doit être un registre XMM.

Règles relatives aux castings implicites à appliquer lors de l'évaluation d'une instruction d'affectation en C :

Le **casting** est l'opération de conversion d'une valeur d'un type de données vers un autre.

Les valeurs qui interviennent dans une opération doivent être codées de la même façon. Pour ce faire, avant la réalisation de toute opération arithmétique, etc, les règles suivantes doivent être respectées :

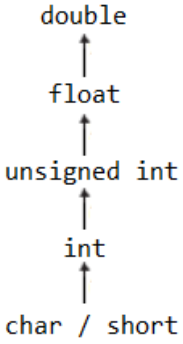
1. Avant une opération sur des entiers, les valeurs entières sur 8 bits (char) ou sur 16 bits (short) sont étendues au type de données int (on suppose que le type de données int a une taille de 32 bits) en tenant compte de leur nature signé (instruction movsx) ou non signé (instruction movzx), ainsi que d'éventuels conversions explicites.

2. Avant une opération dans laquelle intervient un entier signé de 32 bits et un entier non signé de 32 bits, la nature non signée l'emporte.

3. Avant une opération dans laquelle intervient un entier signé de 32 bits et un flottant à simple précision, l'entier est converti en un flottant à simple précision.

4. Avant une opération dans laquelle intervient un entier signé de 32 bits et un flottant à double précision, l'entier est converti en un flottant à double précision.

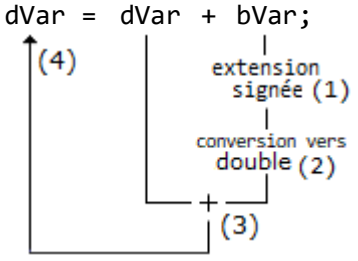
5. Avant une opération dans laquelle intervient un flottant à simple précision et un flottant à double précision, le flottant à simple précision est converti en un flottant à double précision.



Exemple:

```
char   bVar;
double dVar;
```

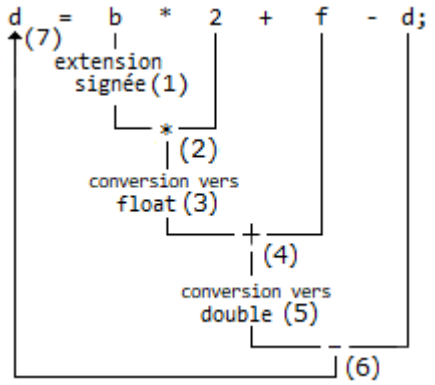
Langage C	*Assembleur*

`dVar = dVar + bVar;` 	`movsx    eax, bVar      // (1)`
	`cvtsi2sd xmm0, eax      // (2)`
	`addsd    xmm0, dVar     // (3)`
	`movsd    dVar, xmm0     // (4)`

Exemple:

```
char b;
float f;
double d;
```

Langage C	*Assembleur*

	`movsx eax, b // (1)`
	`imul eax, 2 // (2)`
	`cvtsi2ss xmm0, eax // (3)`
	`addss xmm0, f // (4)`
	`cvtss2sd xmm0, xmm0 // (5)`
	`subsd xmm0, d // (6)`
	`movsd d, xmm0 // (7)`

Changer le signe d'un double: pour changer le signe d'un nombre à virgule flottante à double précision, il faut multiplier ce nombre par -1.

Exemple:

```
double d;
double d1 = 10;
const double dconst = -1;
```

Langage C	*Assembleur*

`d = -d1;`	`movsd xmm0, d1`
	`mulsd xmm0, dconst`
	`movsd d, xmm0`

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

Les castings explicites : les changements de type ont un impact sur le choix des instructions.

Exemple:

```
int i = -8;
int i1 = 5;
double d = 3.2;
const double d20 = 2.0;
```

Langage C	*Assembleur*

`d = (double)i / i1;`	`cvtsi2sd xmm0, i`
	`cvtsi2sd xmm1, i1`
	`divsd xmm0, xmm1`
	`movsd d, xmm0`

`i = (int)d / 3;`	`cvttsd2si eax, d`
	`cdq`
	`mov ebx, 3`
	`idiv ebx`
	`mov i, eax`

`i = (float)d / 3;`	`cvtsd2ss xmm0, d`
	`mov ebx, 3`
	`cvtsi2ss xmm1, ebx`
	`divss xmm0, xmm1`
	`cvttss2si eax, xmm0`
	`mov i, eax`

`i = (unsigned int)d / 3;`	`cvttsd2si eax, d`
	`mov edx, 0`
	`mov ebx, 3`
	`div ebx`
	`mov i, eax`

`d = i1 / 2.0;`	`cvtsi2sd xmm0, i1`
	`divsd xmm0, d20`
	`movsd d, xmm0`

`d = (float)i / i1;`	`cvtsi2ss xmm0, i`
	`cvtsi2ss xmm1, i1`
	`divss xmm0, xmm1`
	`cvtss2sd xmm0, xmm0`
	`movsd d, xmm0`

Quand une valeur numérique n'a pas de partie décimale, par exemple la valeur 3, elle est considérée comme étant du type int.

Quand une valeur numérique a une partie décimale, par exemple la valeur 2.0, elle est considérée comme étant du type double.

Exercices

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

```
int    i = 20;
float  f = 52.5;
double d = 3.14;
char   b = 40;
```

- `d = d + b * 3.5;` `// d = 143.13999...`
- `f = i * (float)d;` `// f = 62.8000031`
- `i = (float)i * 2 + f - 5.2 * b;` `// i = -115`
- `f = (int)f / (b - 36);` `// f = 13.0`
- `i = (float)d * 2 + f - 5 * b;` `// i = -141`
- `d = ((int)d * 1000) / (f - 20);` `// d = 92.30769348...`
- `d = -((double)b + (int)d * i) / 2.3;` `// d = -43.478260...`
- `i = (int)(-d + 2.81 - -b * f + 3.5) / 3;` `// i = 701`
- `d = (int)d * 3 + (double)-i / f + (b + 2.1);` `// d = 50.7190476190...`
- `b = (f + 0.5 * i) / b + f;` `// b = 54`
- `f = d / 2 * i / 3 + b / 2.5;` `// f = 26.4666672`
- `f = (d * 100) / (b - 0xff & 0xf0);` `// f = 9.8125`

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.