Liste d'exercices 2 : les nombres entiers signés

Notions à considérer pour pouvoir faire les exercices

La matière du cours se trouve dans le chapitre 3 dans les notes de cours.

Contraintes imposées par le processeur :

- Les opérandes d'une instruction à 2 opérandes doivent avoir la même taille, à l'exception de l'instruction movsx.
- L'opérande destination doit pouvoir stocker le résultat de l'exécution de l'instruction. Il ne peut donc être une valeur numérique.
- Une instruction à 2 opérandes ne peut utiliser qu'au plus une variable.

Contraintes imposées par le langage C :

- Les opérations sont réalisées sur des entiers de 32 bits.
- On ne modifie pas les variables figurant à droite de l'opérateur d'affectation.
- Les opérateurs n'ont pas tous la même priorité.

Priorités	*Opérateurs en C*	*Instructions en assembleur*
+++++	`() (parenthèses)`	
+++	`*, /, % (multiplication, division, modulo)`	`imul, idiv, idiv`
++	`+, - (addition, soustraction)`	`add, sub`
+	`= (affectation)`	`mov`

Si plusieurs opérateurs avec la même priorité se trouvent dans une instruction d'affectation en C et que des parenthèses ne sont pas utilisées, alors l'évaluation se fait simplement de gauche à droite.

Exemple :

```
int i, j;
```

Langage C	*Assembleur*

`i = j - 2 * i + 5;`	`mov  ebx, 2`
	`imul ebx, i`
	`mov  eax, j`
	`sub  eax, ebx`
	`add  eax, 5`

	`mov  i, eax`

On a fait d'abord la multiplication. Ensuite, en allant de gauche à droite, on a fait la soustraction, puis l'addition. Enfin, on a affecté le résultat à la variable i.

Instructions à utiliser dans les exercices :

- *mov opd, ops* : opd = ops
- *add opd, ops* : opd = opd + ops
- *sub opd, ops* : opd = opd - ops
- *imul opd, ops* : opd = opd * ops
- *movsx opd, ops* : elle étend la valeur de l'opérande ops à la taille de l'opérande opd, puis elle copie cette valeur étendue dans opd. On utilise cette instruction pour étendre à 32 bits la valeur de variables ayant une taille de 8 ou de 16 bits.

  Exemple :

```
char  b = 25;       // b est une variable en mémoire d'1 octet
int   i = -110;     // i est une variable en mémoire de 4 octets
```

Langage C	*Assembleur*
`i = b;`	`movsx eax, b` `mov   i, eax`

On a étendu la valeur de la variable b à 4 octets, puis on a assigné cette valeur à la variable i.

- *cdq* : cette instruction étend la valeur de EAX à la paire EDX, EAX. Elle précède toujours l'instruction idiv.
- *idiv diviseur* : cette instruction sert à diviser un entier signé par un autre. Elle fonctionne ainsi : le nombre entier signé de 64 bits formé par la paire EDX, EAX est divisé par un nombre entier signé de 32 bits. Après l'opération, EAX stocke le quotient et EDX stocke le reste (modulo) de la division.

Exemple pour le modulo :

```
int  i, j;
```

Langage C	*Assembleur*

`i = i % j;`	`mov  eax, i`
	`cdq`
	`idiv j`
	`mov  i, edx`

On a préparé dans edx, eax le dividende sur 64 bits en utilisant cdq. Puis on a divisé ce dividende par la valeur de la variable j. Enfin, on a copié dans la variable i le reste de la division figurant dans le registre edx.

Exemple pour la division :

```
int  i, j;
```

Langage C	*Assembleur*

`i = i / j;`	`mov  eax, i`
	`cdq`
	`idiv j`
	`mov  i, eax`

On a préparé dans edx, eax le dividende sur 64 bits en utilisant cdq. Puis on a divisé ce dividende par la valeur de la variable j. Enfin, on a copié dans la variable i le quotient figurant dans le registre eax.

Registres de 32 bits disponibles pour stocker les valeurs intermédiaires : eax, ebx, ecx, edx.

Exercices

```
char  b = 25;    // b est une variable en mémoire d'1 octet avec 25 pour valeur
short s = 40;    // s est une variable en mémoire de 2 octets avec 40 pour valeur
int   i = -110;  // i est une variable en mémoire de 4 octets avec -110 pour valeur
```

- `i = s;` `// i = 40`
- `i = b;` `// i = 25`
- `i = i + b + s;` `// i = -45`
- `i = i * b + s * i;` `// i = -7150`
- `i = i * (b + s) * i;` `// i = 786500`
- `i = (i - s) * 3;` `// i = -450`
- `i = (i + b) * (s + i);` `// i = 5950`
- `i = i % b;` `// i = -10`
- `i = i / s;` `// i = -2`
- `i = i / 3 * i / 2;` `// i = 1980`
- `i = i / 3 + i / 2;` `// i = -91`