

LANGAGE D'ASSEMBLAGE ET EXERCICES (17)

Partie théorique : les instructions de division DIV et IDIV

L'instruction `DIV` permet de diviser des nombres non signés alors que l'instruction `IDIV` permet de diviser des nombres signés.

`DIV / IDIV OPERANDE`

L'instruction `DIV` (ou `IDIV`) permet de diviser les bytes contenus dans `AX` ou `DX-AX` par des bytes référencés par `OPERANDE` et de stocker les résultats (le quotient et le reste) dans les bytes contenus dans `AL` et `AH` ou dans `AX` et `DX`.

Pour savoir si le dividende est `AX` ou `DX-AX` et où sont stockés les résultats, il faut regarder la taille de l'opérande (le diviseur). Si le diviseur a une taille de un byte, le dividende sera le contenu de `AX` et le quotient sera stocké dans `AL` et le reste dans `AH` mais si le diviseur a une taille de deux bytes, le dividende sera le contenu du couple de registres `DX-AX` et le quotient sera stocké dans `AX` et le reste dans `DX`.

Les flags ne sont pas positionnés par les instructions de division `DIV` et `IDIV`.

L'opérande est soit un registre soit une zone mémoire.

Le format général de l'instruction est :

| | |
|------|----------|
| DIV | registre |
| ou | ou |
| IDIV | mémoire |

Exemples pour des nombres binaires non signés :

| | | |
|------------------|----------------------------|--|
| <code>DIV</code> | <code>BL</code> | <code>AL</code> ← quotient <code>AX / BL</code> et <code>AH</code> ← reste <code>AX / BL</code> |
| <code>DIV</code> | <code>byte ptr [BX]</code> | <code>AL</code> ← quotient <code>AX / byte [DS :BX]</code> et <code>AH</code> ← reste <code>AX / byte [DS :BX]</code> |
| <code>DIV</code> | <code>BX</code> | <code>AX</code> ← quotient <code>DX-AX / BX</code> et <code>DX</code> ← reste <code>DX-AX / BX</code> |
| <code>DIV</code> | <code>word ptr [SI]</code> | <code>AX</code> ← quotient <code>DX-AX / mot [DS :SI]</code> et <code>DX</code> ← reste <code>DX-AX / mot [DS :SI]</code> |

Exemples pour des nombres binaires signés :

| | | |
|-------------------|----------------------------|--|
| <code>IDIV</code> | <code>CH</code> | <code>AL</code> ← quotient <code>AX / CH</code> et <code>AH</code> ← reste <code>AX / CH</code> |
| <code>IDIV</code> | <code>byte ptr [SI]</code> | <code>AL</code> ← quotient <code>AX / byte [DS :SI]</code> et <code>AH</code> ← reste <code>AX / byte [DS :SI]</code> |
| <code>IDIV</code> | <code>DI</code> | <code>AX</code> ← quotient <code>DX-AX / DI</code> et <code>DX</code> ← reste <code>DX-AX / DI</code> |
| <code>IDIV</code> | <code>word ptr [BX]</code> | <code>AX</code> ← quotient <code>DX-AX / mot [DS :BX]</code> et <code>DX</code> ← reste <code>DX-AX / mot [DS :BX]</code> |

Partie théorique : affichage d'un nombre

En langage d'assemblage, l'affichage, comme la lecture, d'un nombre se fait en plusieurs étapes. La première consiste à convertir le nombre binaire en une chaîne de chiffres, la seconde à afficher cette chaîne de chiffres. Quelle que soit la base dans laquelle on désire afficher le nombre, le raisonnement reste identique.

affichage du contenu hexadécimal d'un registre :

Comme expliqué dans la première partie du cours le nombre hexadécimal 2A5C

$$= 2 * 10^3 + A * 10^2 + 5 * 10 + C \quad \text{en base 16}$$

Pour retrouver chaque chiffre hexadécimal contenu dans un nombre, il faut garder les restes des divisions successives par 10_{16} (soit 16_{10}). En effet :

| | |
|-------------------|---------|
| $2A5C / 10 = 2A5$ | reste C |
| $2A5 / 10 = 2A$ | reste 5 |
| $2A / 10 = 2$ | reste A |
| $2 / 10 = 0$ | reste 2 |

Pour convertir un nombre hexadécimal en une suite de chiffres hexadécimaux, il faut, donc, implémenter l'algorithme suivant:

```
i ← 4
répéter tant que le nombre est ≠ 0
    tab[i] ← reste (nombre / 1016)
    nombre ← quotient (nombre / 1016)
    i ← i-1
fin répéter
```

Plus précisément, en langage d'assemblage, pour afficher le contenu hexadécimal d'un registre de deux bytes, il faut implémenter l'algorithme suivant:

```
réserver un tableau résultat de 4 bytes
réserver et initialiser à la valeur 4 un compteur i
répéter 4 fois
    diviser le contenu du registre par 1610
    résultat[i] ← reste de la division
    le registre ← quotient de la division
    i ← i-1
fin répéter
ajuster le contenu du tableau résultat afin qu'il contienne les codes ASCII des chiffres hexadécimaux
au lieu de leurs valeurs
afficher le tableau résultat
```

Exemple:

si AX contient la valeur 3A 2C
résultat devra contenir 03 0A 02 0C
la chaîne devra contenir 33 41 32 43 afin d'afficher 3 A 2 C

Rappel des codes ASCII des chiffres hexadécimaux:

| chiffre | code |
|---------|------|
| 0 | 30 |
| 1 | 31 |
| 2 | 32 |
| 3 | 33 |
| 4 | 34 |
| 5 | 35 |
| 6 | 36 |
| 7 | 37 |

| chiffre | code |
|---------|------|
| 8 | 38 |
| 9 | 39 |
| A | 41 |
| B | 42 |
| C | 43 |
| D | 44 |
| E | 45 |
| F | 46 |

En effet, l'algorithme appliqué à l'exemple donne, sachant que AX contient la valeur 3A 2C :

↙
 $i \leftarrow 4$
tab[4] \leftarrow reste $(3A2C / 10_{16}) = C$
AX \leftarrow quotient $(3A2C / 10_{16}) = 3A2$
 $i \leftarrow 3$
tab[3] \leftarrow reste $(3A2 / 10_{16}) = 2$
AX \leftarrow quotient $(3A2 / 10_{16}) = 3A$
 $i \leftarrow 2$
tab[2] \leftarrow reste $(3A / 10_{16}) = A$
AX \leftarrow quotient $(3A / 10_{16}) = 3$
 $i \leftarrow 1$
tab[1] \leftarrow reste $(3 / 10_{16}) = 3$
AX \leftarrow quotient $(3 / 10_{16}) = 0$

tab[1] $\leftarrow 3 + 30 = 33$
tab[2] $\leftarrow A + 37 = 41$
tab[3] $\leftarrow 2 + 30 = 32$
tab[4] $\leftarrow C + 37 = 43$

afficher les par exemple

Exercice :

Ecrire un programme, en langage d'assemblage, qui affiche le contenu hexadécimal d'un registre de deux bytes.

Partie théorique : affichage en décimal de nombres non signés

Pour retrouver les chiffres décimaux contenus dans un nombre qui est stocké en hexadécimal, il faut garder les restes des divisions successives par A_{16} (soit 10_{10}). En effet :

| | |
|------------------|---------|
| $2A5C / A = 43C$ | reste 4 |
| $43C / A = 6C$ | reste 4 |
| $6C / A = A$ | reste 8 |
| $A / A = 1$ | reste 0 |
| $1 / A = 0$ | reste 1 |

Pour convertir un nombre hexadécimal (de 4 chiffres) en une chaîne de chiffres décimaux, il faut, donc, implémenter l'algorithme suivant:

```
i ← 5
répéter tant que le nombre est ≠ 0
    tab[i] ← reste (nombre / 0A16)
    nombre ← quotient (nombre / 0A16)
    i ← i-1
fin répéter
```

Plus précisément, en langage d'assemblage, pour afficher, en décimal, le contenu d'un registre de deux bytes qui contient un nombre non signé, il faut implémenter l'algorithme suivant:

```
réserver un tableau résultat de 5 bytes
initialiser chaque élément du tableau à la valeur 0
réserver et initialiser à la valeur 5 un compteur i
répéter
    diviser le contenu du registre par 1010
    résultat[i] ← reste de la division
    le registre ← quotient de la division
    i ← i - 1
jusqu'à ce que le quotient soit nul
ajuster le contenu du tableau résultat afin qu'il contienne les codes ASCII des chiffres hexadécimaux au lieu de leurs valeurs
afficher le tableau résultat
```

Exemple:

| | |
|-------------------------------------|----------------|
| si AX contient la valeur | 3A 2C |
| résultat devra contenir | 01 04 08 09 02 |
| la chaîne à afficher devra contenir | 31 34 38 39 32 |
| l'affichage sera | 14892 |

Exercice :

Écrire un programme, en langage d'assemblage, qui affiche, en décimal, le contenu d'un registre de deux bytes qui contient un nombre non signé.

Partie théorique : affichage en décimal de nombres signés

L'algorithme à implémenter est le suivant:

Si le registre contient une valeur négative

alors afficher -

prendre le complément à 2 du contenu du registre

sinon afficher +

fin si

convertir le nombre en une chaîne de chiffres suivant le même algorithme que pour l'exercice précédant

afficher la chaîne de chiffres

Exemple:

si AX contient la valeur 3A 2C

la chaîne à afficher devra contenir 2B 31 34 38 39 32

l'affichage sera +14892

si AX contient la valeur C5 D4

la chaîne à afficher devra contenir 2D 31 34 38 39 32

l'affichage sera -14892

Exercice :

Ecrire un programme, en langage d'assemblage, qui affiche, en décimal, le contenu d'un registre de deux bytes qui contient un nombre signé.