7. Déclaration des données et aperçu sur quelques opérations de base

Les déclarations sont réalisées grâce à l'utilisation d'un certain nombre de directives (pseudo-instructions). La différence entre une instruction et une pseudo-instruction est que l'instruction est destinée pour être exécuté par le processeur. Par contre, une pseudo-instruction est destinée à être examiné par l'outil de traduction (l'outil assembleur inclut dans QtSpim). En général une directive est précédée par un point ".".
La déclaration des constantes et des variables du programme se font au niveau du segment de données. La déclaration des données est présentée dans les sections 7.1 et 7.2. Les sections restantes présentent quelques opérations élémentaires : opérations de transfère de données, opérations arithmétiques et les opérations d'E/S.

7.1. La déclaration des constantes

Pour déclarer une constante, on utilise le symbole de l'égalité "=" comme suit :

Nom constante = valeur_ou_expression_constante

Exemple :

Soit Const1 une constante déclarée par :

Const1 = 9

on peut la déclarer d'une manière équivalente par :

Const1 = 2*2+5

La base par défaut de la valeur constante est la base décimale (10). Pour exprimer la valeur constante en hexadécimales (base 16) on débute la valeur par 0x (par exemple la valeur hexadécimale 0x0F)

7.2. La déclaration des variables

Déclarer une variable conduit à lui réserver de l'espace dans la mémoire centrale (pour stocker des valeurs ou des donnés). Grossièrement il n'ya que deux types de valeurs : les valeurs "numériques" (entiers, nombres réels) et les valeurs caractères/chaines de caractères. Chaque caractère est codé dans l'ordinateur à l'aide d'un codage tel que le codage ASCII par exemple.

Pour la déclaration des variables numériques, on utilise plusieurs directives pour définir le type (directives de types) telles que :

- Directive " .byte" : pour les entiers dont on réserve 8 bits dans la M.C

- Directive " .half" : pour les entiers dont on réserve 16 bits dans la M.C
- Directive " .word" : pour les entiers dont on réserve 32 bits dans la M.C

Pour la déclaration des variables caractères/chaines de caractères, on utilise plusieurs directives pour définir le type telles que :

- Directive " .ascii" : pour les chaines de caractères
- Directive " .asciiz" : pour les chaines de caractères terminés avec le caractère ASCII non affichable (de contrôle) **NULL** (code ascii de NULL= 0).
  NULL est nécessaire pour le service d'affichage des chaines de caractères (Voir la section concernant les E/S pour plus de détails).

Pour réserver **n bytes** non initialisés dans la M.C, on utilise la directive: **".space n"** .

La déclaration d'une variable est réalisée comme suit :

<nom_de_variable> : . <directive_de_type> <valeur_initiale>

Dans le bas niveau de l'ordinateur, une variable peut être soit une ***variable simple*** (contenant un seul élément) ou bien une ***variable complexe*** sous forme d'un tableau ou vecteur de valeurs (contenant plusieurs valeurs).

En assembleur, pour déclarer une variable on utilise une ***directive de type***. Une directive de type ***précise la taille à allouer à un élément*** parmi les éléments associés à la variable. Si une variable contient **plusieurs éléments (variable complexe)**, alors ces éléments ont la **même taille**.

Dans ce qui suit, on va essayer de comprendre ces directives (et par conséquent la déclaration des variables) à travers des exemples.

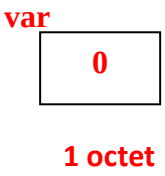
7.2.1. La directive de type .byte

La première directive de type qu'on va illustrer est la directive .**byte**. Cette dernière réserve la taille d'***un seul octet*** (byte) pour l'élément.

Exemple 1 : var: .byte 0

Dans la déclaration d'une variable, on commence toujours par le **nom de la variable** *("var" dans notre exemple), suivie par* **un espace** *puis de la* **directive de type** *(dans notre exemple il s'agit de .byte), suivie par* **un espace** *et enfin on précise* **la valeur initiale** *(dans l'exemple, on a la valeur initiale 0). La nature de la valeur initiale nous aide à* **déterminer s'il s'agit d'une variable simple ou complexe**. *Par exemple, pour ce 1er exemple : la valeur initiale est constitué d'un seul élément (le 0) donc la variable "var" est une variable simple (1 seul élément).*

Pour cet exemple, on va réserver 1 octet (directive .byte) pour l'élément 0.



Donc:

la taille de l'espace réservé à la variable "var"=

taille de l'espace définie par la directive de type (.byte) × nombre d'éléments=

1 octet× 1 élement = 1 octet (initialisé par 0)

Exemple 2 : var2: .byte 4,7,5

On commence par déterminer si la variable "var2" est une variable simple ou complexe? La valeur initiale de var2 est composée de plusieurs éléments (vecteur): le 1er élément= 4, 2ème élément=7 et 3ème élément=5. Donc var2 est une variable complexe. Dans cet exemple chacun de ces trois éléments va occuper 1 octet puisqu'on a la directive byte. Il est important de noter que dans la déclaration de la variable, ***les éléments sont séparés par une virgule***.

Donc:

la taille de l'espace réservé à la variable "var2"=

taille de l'espace définie par .byte × nombre d'éléments= 1 octet× 3éléments

= *3 octets (initialisés respectivement par 4, 7 et 5)*

7.2.2. Les directives de type .half et .word

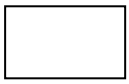
Ces deux directives réservent respectivement la taille d'***un demi-mot*** (16 bits) et d'***un seul mot*** (32 bits) pour l'élément. ***La seule différence entre ces deux directives et la directive .byte réside dans la taille de l'espace réservé pour l'élément***.

7.2.3. La directive de type .space

Exemple: var3: .space 1

La variable var3 est une *variable simple* car elle contient un seul élément (on a un seul élément byte car on a un 1 après la directive .space). L'utilisation de la directive space implique que l'élément ou la case n'a ***pas de valeur initiale***.

var3



Case ou élément non initialisé

1 octet

Donc:

la taille de l'espace réservé à la variable "var3"=

taille de l'espace définie par space × nombre d'éléments= 1 octet× 1 élement

= *1 octets (non initialisé)*

7.2.4. La directive de type .ascii

Exemple1 : var4: .ascii "A"

La variable var4 est une variable simple car sa valeur initiale est constituée d'un seul élément (***caractère*** **"A"**).

var4

1 octet

Donc:

la taille de l'espace réservé à la variable "var4"=

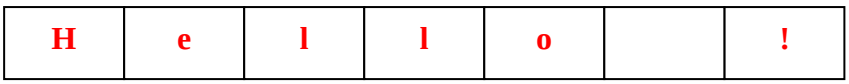
taille de l'espace définie par .ascii × nombre d'éléments= 1 octet× 1 élément

= *1 octets (initialisé par le caractère A)*

Exemple2 : var5: .ascii "Hello !"

La variable var5 est une variable complexe car sa valeur initiale est constituée de ***plusieurs éléments*** (les éléments ou caractères constituant le message "***Hello !***").

var5



1 octet 1 octet 1 octet 1 octet 1 octet 1 octet 1 octet

Donc:

la taille de l'espace réservé à la variable "var5"=

taille de l'espace définie par .ascii × nombre d'éléments= 1 octet× 7éléments

= *7 octets (initialisés respectivement par H, e, l, l, o, un espace, !)*

La variable var5 peut être aussi déclarée de manière équivalente par :

```
var5: .ascii "H"
      .ascii "e"
      .ascii "llo !"
```

7.2.5. La déclaration des matrices

```
var6:  .byte 1, 2, 3
       .byte 4, 5, 6
```

Cette variable représente la matrice :

var6

1	**2**	**3**
4	**5**	**6**

7.3. Les opérations de déplacement des données

Les instructions en assembleur sont simples et basiques et chacune effectue une seule opération. Dans les langages de haut niveau, une instruction est en générale traduite en un ensemble d'instructions en assembleur.

Une instruction en assembleur est grosso modo constituée d'un **code opération** (opération à réaliser) et un ensemble d'**opérandes**. Un opérande fait référence à une donnée (***opérande source***) ou à l'endroit où on va mettre le résultat (***opérande de destination***).

7.3.1. L'instruction Load

Cette instruction, dans son format de base, permet de charger un registre (de destination) avec le contenu d'une variable (espace en M.C) :

l<type> registre_destination, variable

avec type = b dans le cas d'une donnée de taille byte
type = h dans le cas d'une donnée de taille half
type = w dans le cas d'une donnée de taille word

Exemple :

```
var7:  .word 100
var8:  .half 100
var9:  .byte 100
.
.
.
lw $t0, var7   #$t0 reçoit le contenu de var7
lh $t1, var8   #$t1 reçoit le contenu de var8
lb $t2, var9   #$t2 reçoit le contenu de var9
```

Il existe deux autres formats de load :

- **li registre_destination, valeur_constante :** qui permet de charger un registre de destination avec une valeur immédiate (constante).

Exemple : li $t0, 12

le registre $t0 va contenir la constante 12

- **la registre_destination, variable :** qui permet de charger un registre de destination avec l'adresse d'une variable ou espace mémoire.

Exemple :

```
var10: .asciiz "B"
.
.
.
la $a0, var10
```

le registre $a0 va contenir l'adresse de l'espace mémoire de var10

7.3.2. L'instruction Store

Cette instruction permet de charger une variable (espace en M.C) avec le contenu d'un registre. Son format général est :

s<type> registre_source, variable
avec type = b dans le cas d'une donnée de taille byte
type = h dans le cas d'une donnée de taille half
type = w dans le cas d'une donnée de taille word

Exemple:

```
var7:  .word 100
var8:  .half 100
var9:  .byte 100
.
.
.
sw $t0, var7  #charger var7 avec le contenu de $t0
sh $t1, var8   #charger var8 avec le contenu de $t1
sb $t2, var9    #charger var9 avec le contenu de $t2
```

7.3.3. L'instruction move

Cette instruction permet de déplacer les données entres registres. Il n'y a aucune instruction qui permet de déplacer une donnée entre deux emplacements mémoires c'est-à-dire entre variables. Il ya plusieurs formats de cette instruction comme le montre le tableau suivant :

Instruction	**Signification**
move reg_{dest}, reg_{sr}	**$reg_{dest} \leftarrow reg_{sr}$**
mfhi reg_{dest}	**$reg_{dest} \leftarrow$ \$hi**
mflo reg_{dest}	**$reg_{dest} \leftarrow$ \$lo**
mthi reg_{sr}	**$Reg_{sr} \rightarrow$ \$hi**
mtlo reg_{sr}	**$Reg_{sr} \rightarrow$ \$lo**

Tableau 3. variantes de l'instruction move