Chapitre 12

Programmation système en BASIC

Nous venons de voir comment effectuer de la programmation généraliste en BASIC. On parle de **programmation système** lorsqu'on veut intervenir sur le système informatique sur lequel on se trouve. Si les principes de celle-ci sont indépendants de tout système, les programmes effectifs exigent une connaissance du système.

12.1 Lecture de la RAM

12.1.1 Segmentation de la mémoire du 8086/8088

Lors de la conception du microprocesseur 8086, les autres microprocesseurs pouvaient accéder au plus à 64 KiO (ce qui correspond à 2^{16} octets, chaque octet pouvant être repérer par une adresse de 16 bits). Cette capacité mémoire semblait largement suffisante à l'époque. Les concepteurs du 8086 ont voulu se démarquer en utilisant 20 broches pour spécifier une adresse en mémoire vive, ce qui conduit à une capacité maximale de mémoire vive de $2^{20} = 1\ 048\ 576$ octets, soit 1 Mio.

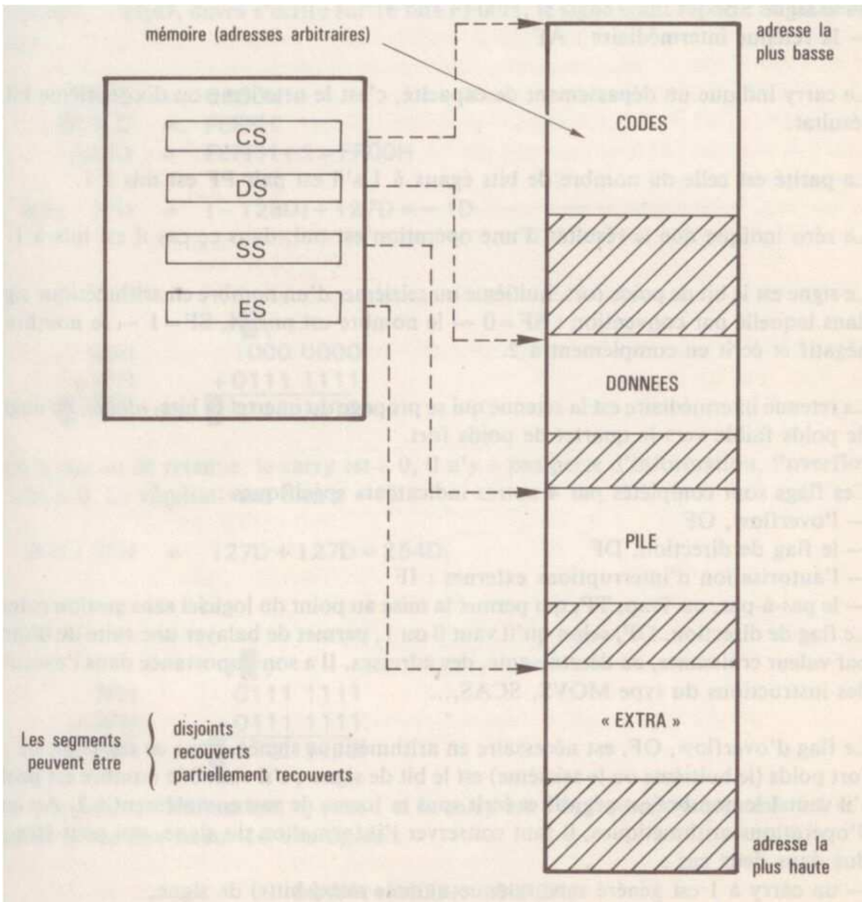

FIG. 12.1 – Segmentation de la mémoire

Ceci étant, cela conduit à une contorsion pour accéder à la mémoire qui aura des conséquences pendant quinze ans. En effet, aucun des registres du 8086 (et donc du 8088) n'a une capacité de 20 bits ; ils ont tous une capacité de 16 bits. Les concepteurs du 8086 ont donc imaginé d'associer deux entiers de 16 bits chacun pour spécifier une adresse.

La mémoire est de 1 Mio est **segmentée**, c'est-à-dire qu'elle est découpée en $2^{16} = 65536$ **segments** d'une capacité d'*a priori* $2^4 = 16$ octets (voir figure 12.1). Pour spécifier une adresse, il faut donc de connaître à la fois son numéro de segment s (appelé **adresse de segment**) et un entier de 4 bits, appelé **décalage** (*offset* en anglais), d, ce qui spécifie l'adresse physique $16 \times s + d$.

Pour compliquer la vie du programmeur, et puisque tout est en 16 bits, les concepteurs du 8086 n'ont pas choisi des décalages de 4 bits mais de 16 bits. La formule ci-dessus reste valable mais les segments s'enchevêtrent les uns les autres.

À un moment donné, on peut accéder à seulement quatre segments simultanément. Les numéros de chacun de ces segments sont les contenus des quatre *registres de segment*, dont nous avons déjà parlé sans en comprendre le rôle jusqu'ici :

– Le **registre de segment de code CS** (pour l'anglais *Code Segment*).
– Le **registre de segment de données DS** (pour l'anglais *Data Segment*).
– Le **registre de segment de pile SS** (pour l'anglais *Stack Segment*).
– Le **registre de segment supplémentaire ES** (pour l'anglais *Extra Segment*).

Le contenu de chacun de ces segments n'est pas quelconque. Comme leur nom l'indique, le segment de code contient le code (le programme), le segment de données les données et le segment de pile la pile (nous verrons plus tard de quoi il s'agit). Notre seule lattitude porte sur le segment supplémentaire, bien que nous verrons qu'il est également spécialisé, surtout lors du transfert des données d'un segment dans un autre.

Nous avons vu également que le microprocesseur possède une unité, la BIU (*Bus Interface Unit*) qui s'occupe de calculer l'adresse physique, c'est-à-dire d'envoyer les signaux nécessaires aux 20 broches d'adresse à partir du contenu du registre de segment adéquat et du déplacement spécifié (nous verrons comment).

12.1.2 La fonction `PEEK()`

Cas général.- Le BASIC permet de lire dans la mémoire vive. Le plus petit élément de mémoire accessible est l'octet, constitué de huit bits. Pour les premiers interpréteurs l'implémentant, la fonction :

```
PEEK(entier)
```

renvoie un entier, compris entre 0 et 255, qui est la valeur de l'octet de l'emplacement mémoire de numéro `entier`.

Cas du QBasic.- Puisque MS-DOS utilise un microprocesseur 8088, qui gère la mémoire par segments, dans ce cas `entier`, qui doit être un entier codé sur deux octets, soit seize bits, spécifie le déplacement. Reste à spécifier le segment. Par défaut il s'agit du segment de données, spécifié par le registre de segment de données DS.

Exemple.- Le programme suivant permet d'afficher le contenu du deuxième octet du segment de données, celui de numéro 1 :

```
PRINT PEEK(1)
```

Ce qui nous donne, par exemple, 117, ce qui ne nous dit pas grand chose pour l'instant.

12.1.3 Spécification du segment

Lorsqu'on déclare une variable dans un programme BASIC, l'interpréteur (en fait le système d'exploitation) cherche un emplacement libre dans la mémoire vive pour pouvoir y entreposer celle-ci de façon contigüe (ce qui est important dans le cas d'un tableau). Du coup, on ne sait pas vraiment où celle-ci se trouve. QBasic nous fournit cependant deux fonctions et une instruction pour s'y retrouver.

Syntaxe.- Les fonctions :

```
VARSEG(variable)
VARPTR(variable)
```

renvoient respectivement le numéro de segment et le décalage de la variable numérique `variable`.

L'instruction :

```
DEF SEG valeur
```

spécifie le segment, grâce à la valeur `valeur` (entier codé sur 16 bits) fournie, comme complément au décalage donné à un certain nombre de fonctions, dont `PEEK()`. Lorsqu'aucune valeur n'est donnée, il s'agit du segment dans lequel on se trouve.

Exemple.- Le programme suivant :

```
CLS
A% = 10
SegA = VARSEG(A%)
OffA = VARPTR(A%)
DEF SEG = SegA
PRINT PEEK(OffA)
DEF SEG
```

affiche 10, comme désiré.

Remarques.- 1^o) Ne pas spécifier le type de la variable `A` ne nous renverrait pas 10 car, par défaut, une variable est de type SINGLE.

- 2^o) Ne surtout pas oublier de revenir au segment dans lequel on se trouvait au départ, faute de quoi on ne maîtrise plus le système.

12.1.4 Désignation des adresses en hexadécimal

Tradition.- Il est traditionnel en informatique, particulièrement en programmation système, de désigner les adresses non pas en décimal mais en hexadécimal, c'est-à-dire en base seize. Ceci est dû au fait qu'on travaille en informatique en base deux. Un nombre en base deux prend beaucoup de place et devient vite illisible pour un être humain. On a donc choisi une base intermédiaire pour laquelle on peut effectuer des conversions très rapidement. Un chiffre en base seize exige quatre bits, soit un demi-octet (*nibble* en anglais).

Représentation des nombres entiers en hexadécimal.- On a besoin de seize chiffres en hexadécimal : on prend les dix chiffres de la numération décimale usuelle '0', '1', '2', '3', '4', '5', '6', '7', '8' et '9' et on continue par les premières lettres de l'alphabet : 'A' ou 'a' pour dix, 'B' ou 'b' pour onze, 'C' ou 'c' pour douze, 'D' ou 'd' pour treize, 'E' ou 'e' pour quatorze et enfin 'F' ou 'f' pour quinze.

En informatique on note de façon habituelle les nombres exprimés en base dix et de façons variées les nombres exprimés en base seize : il faut bien distinguer 11 en base deux (écrit 11_2 en mathématiques) de 11 en base 10 (11 ou 11_{10} s'il y a un doute, la base s'écrivant toujours en base dix) et de 11 en base seize (11_{16} en mathématiques). On écrit par exemple les nombres exprimés en base seize en les faisant suivre d'un 'h' (initiale de *hexadécimal*).

Cas du QBasic.- En QBasic on marque deux fois le fait qu'un entier soit exprimé en hexadécimal. On le fait précéder du préfixe '&H' : l'**esperluette** (le symbole &, *ampersand* en anglais) indique qu'il ne s'agit pas d'une variable, le 'h' qu'on est en base seize.

12.1.5 Désignation des octets en hexadécimal

Tradition.- Il est également traditionnel, en programmation système, de désigner le contenu des octets en hexadécimal, par deux chiffres.

Conversion grâce au BASIC.- La fonction :

`HEX$(entier)`

permet d'afficher l'entier en hexadécimal.

Testons-la. Le programme suivant :

```
PRINT HEX$(125)
```

affiche '`7D`' dont on pourra vérifier qu'il s'agit bien de la valeur hexadécimale de 125.

Amélioration.- Comment savoir alors si le nombre affiché est en décimal ou en hexadécimal ? Pour suivre les conventions QBasic, on écrira plutôt le programme ci-dessus de la façon suivante :

```
PRINT "&H";HEX$(125)
```

qui affiche '`&H7D`'.

Exercice.- 1^o) *Écrire un programme* BASIC *qui effectue l'analogue de la fonction* `HEX$()`.
- 2^o) *La fonction BASIC est-elle implémentée de cette façon ? Expliquez pourquoi.*

12.2 Écriture dans la RAM

12.2.1 Écriture d'un octet

Cas général.- Le BASIC permet d'écrire dans la mémoire. Comme pour la lecture, le plus petit élément de mémoire accessible est l'octet. La fonction :

`POKE I, a`

où `I` est un entier codé sur seize bits et `a` un entier compris entre 0 et 255, écrit au déplacement `I` l'octet de valeur `a`. On peut spécifier le segment comme dans le cas de la fonction PEEK().

12.2.2 Exemple

Il peut être dangereux d'écrire n'importe où. Servons-nous de notre programme précédent pour remplacer la valeur d'une variable et vérifions que ça marche :

```
CLS
A% = 10
SegA = VARSEG(A%)
OffA = VARPTR(A%)
DEF SEG = SegA
POKE OffA, 12
DEF SEG
PRINT A%
```

affiche 12, comme désiré.