Chapitre 10

Programmation système en BASIC

Nous venons de voir comment effectuer de la programmation généraliste en BASIC, la seule indispensable. Nous avons déjà dit que nous n'aborderons ni la programmation graphique, ni la programmation sonore ici. 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.

10.1 Lecture de la RAM

10.1.1 La fonction `PEEK()`

Cas général.- Le BASIC permet de lire dans la mémoire. Le plus petit élément de mémoire accessible est l'octet, constitué de huit bits. Les octets de la mémoire sont numérotés de 0 à N, où N+1 est la capacité de la mémoire, en octets. La fonction :

```
PEEK(entier)
```

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

Cas du CPC.- L'Amstrad CPC utilise un microprocesseur Z80 avec 16 broches pour indiquer l'adresse en mémoire. Sa capacité maximale est donc de $2^{16} = 65\,536$ octets, soit 64 KiO. Tous les CPC comportent au moins 64 Ko de RAM (64 Ko pour le CPC 664, 128 Ko pour le 6128). On a donc N = 65 535.

Sur certains ordinateurs, par exemple les IBM PC, les adresses de la ROM et de la RAM sont disjointes, ce qui est naturel. Ce n'est pas le cas du CPC : la ROM occupe 16 KiO dont les adresses sont communes avec les 16 premiers KiO de la RAM ; on utilise une astuce pour savoir si on veut adresser la ROM ou la RAM.

Amstrad a implémenté la fonction `PEEK()` de façon à ce qu'elle lise dans la RAM (et non dans la ROM).

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

```
10 PRINT PEEK(1)
RUN
 137
Ready
```

Ceci ne nous dit pas grand chose pour l'instant.

10.1.2 Carte de la RAM

Nous savons donc maintenant comment lire dans la RAM mais il faut bien avouer que cela ne nous sert pas à grand chose tant que l'on ne sait pas comment le système est organisé, autrement dit tant que l'on ne connaît pas la **carte de la RAM**.

10.1.2.1 Cas du CPC

La RAM des CPC est organisée de la façon suivante :

Octets	Utilisation
0 à 367	Réservé au système
368 à 43 903	Pour les programmes BASIC
43 904 à 49 141	Réservé au système
49 142 à 65 535	Mémoire écran

L'utilisateur ne dispose donc pas de 64 KiO mais de 43 535 octets, ce qui semblait suffisant à l'époque de la conception du CPC. Nous reviendrons petit à petit sur ces différentes **zones** de mémoire.

10.1.3 La zone utilisateur

La zone utilisateur est réservée au programme BASIC. Au fur et à mesure que l'on entre des lignes de programme, celles-ci sont stockées à partir de l'adresse 43 904 mais pas telles quelles. Le programme suivant :

```
10 PRINT "A = ";10
20 FOR I=368 to 390
30 PRINT PEEK(I); " ";
40 NEXT
RUN
A = 10
 16 0 10 0 191 32 34 65
 32 61 32 34 59 25 10 0
 21 0 20 0 158 32 13
Ready
```

nous montre le contenu des 23 premiers octets de la zone utilisateur après avoir écrit le programme ci-dessus.

Il faut savoir comment Amstrad a décidé de conserver les programmes pour pouvoir interpréter le résultat. Nous allons analyser cette ligne en indiquant comment elle est sauvegardée :

- Les deux premiers octets indiquent la longueur de la ligne BASIC, l'octet le plus faible d'abord. Ici on a `16` et `0` donc une longueur de $16 + 0 \times 256 = 16$. Le numéro de ligne n'est pas compté dans la longueur, comme on peut s'en assurer.
- Les deux octets suivants indiquent le numéro de ligne BASIC, l'octet le plus faible d'abord. Ici on a `10` et `0`, ce qui confirme bien que notre programme commence par la ligne 10.
- Le programme commence par un mot clé du BASIC, à savoir 'PRINT'. Tout mot clé est remplacé par un **token**, c'est-à-dire un code. Le token de 'PRINT' est 191.
- L'octet suivant, `32`, est tout simplement le code ASCII de l'espace qui suit le mot clé 'PRINT'.
- Les cinq octets suivants spécifient la chaîne de caractères : `34` est le code ASCII de l'apostrophe verticale ", `65` le code de 'A', `32` le code de l'espace, `61` le code du signe égal et '32' le code de l'espace.
- L'octet suivant, `59`, est le code ASCII du point-virgule qui suit la chaîne de caractères.
- L'octet suivant, `25`, est un code pour indiquer que ce qui suit est une constante occupant un octet.
- La valeur de cette constante est donnée par l'octet suivant, `10`. On retrouve bien la valeur de la constante que nous avons donnée mais codée sur un octet au lieu de deux (un par caractère).
- L'octet qui suit est `0`, code qui marque la fin de la ligne.
- Les deux octets suivants indiquent la longueur de la deuxième ligne ; les deux suivants le numéro de ligne puis un token.

10.1.4 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 naturellement en base deux. Un nombre en base deux prend beaucoup de place et devient vite illisible. On a donc choisi une base intermédiaire dans 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 habituels '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 (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*).

Amstrad a décidé de les faire précéder d'une **esperluette**, le symbole & (*ampersand* en anglais).

Carte de la RAM.- La carte de la RAM Amstrad sera donc plutôt représentée de la façon suivante :

Octets	Utilisation
&0000 - &016F	Réservé au système
&0170 - &AB7F	Pour les programmes BASIC
&AB80 - &ABFF	Réservé au système
&C000 - &FFFF	Mémoire écran

On remarquera l'habitude de placer des zéros initiaux pour avoir le même nombre de chiffres dans certaines circonstances.

Réécriture du programme.- Le programme ci-dessus s'écrira donc plutôt de la façon suivante :

```
10 PRINT "A = ";10
20 FOR I=&170 to &187
30 PRINT PEEK(I); " ";
40 NEXT
RUN
A = 10
 16 0 10 0 191 32 34 65
 32 61 32 34 59 25 10 0
 21 0 20 0 158 32 13
Ready
```

10.1.5 Désignation des octets en hexadécimal

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

Fonction en BASIC.- La fonction :

```
HEX$(entier)
```

permet d'afficher l'entier en hexadécimal.

Testons-la avec le programme suivant :

```
10 PRINT HEX$(125)
RUN
7D
```

Amélioration.- Comment savoir si le nombre affiché est en décimal ou en hexadécimal ? On écrira plutôt le programme ci-dessus de la façon suivante :

```
10 PRINT "&";HEX$(125)
RUN
&7D
```

Réécriture du programme.- On réécrira donc notre premier programme système de la façon suivante :

```
10 PRINT "A = ";10
20 FOR I=&170 to &187
30 PRINT "&";HEX$(PEEK(I));" ";
40 NEXT
RUN
A = 10
 &10 &0 &A &0 &BF &20 &22 &41 &20 &3D &20
 &22 &3B &19 &A &0 &15 &0 &14 &0 &9E &20
 &D &5
Ready
```

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.*

10.2 Écriture dans la RAM

10.2.1 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 compris entre 0 et N ($N+1$ étant la taille de la mémoire) et `a` un entier compris entre 0 et 255, écrit à l'emplacement mémoire `I` l'octet de valeur `a`.

Cas du CPC.- On écrit évidemment uniquement dans la RAM. Comme pour la lecture, dans le cas de l'Amstrad CPC, on a N = 65 535.

10.2.2 Exemple

Il peut être dangereux d'écrire n'importe où. Notre premier programme a placé, entre autre, la valeur &A ou 10 à l'emplacement mémoire &180. Remplaçons-la par 78 ou &4E, faisons tourner le programme puis listons-le :

```
10 PRINT "A = ";10
20 POKE &17E,11
RUN
A = 10
READY
LIST
10 PRINT "A = ";11
20 POKE &17E,11
Ready
```

La ligne 10 a été changée : la constante est maintenant 11 au lieu de 10.