Chapitre 11

Accès à la mémoire vive du 8088

Le microprocesseur `i8086` se différencie, du point de vue de l'accès à la mémoire vive, de la plupart des autres microprocesseurs sur un point : pour des raisons de compatibilité avec le microprocesseur phare précédent d'*Intel*, à savoir le `i8085`, la mémoire est *segmentée*. Commençons donc par étudier l'accès à la mémoire sur le `i8085`, que l'on peut explorer avec le `i8086` ou l'un de ses descendants grâce à la compatibilité ascendante

11.1 Modèles de mémoire

11.1.1 La mémoire plate du i8085

Capacité mémoire.- Pour le i8085, la capacité maximale adressée de mémoire vive est de 64 KiO, quantité qui paraît largement suffisante au moment de la conception de ce microprocesseur.

Bien entendu un système donné ne dispose pas nécessairement de cette capacité maximale. On ajoute des barettes de mémoire au fur et à mesure de ses besoins (et de ses moyens).

Adressage.- L'unité de mémoire minimum est l'octet. Les octets sont numérotés de 0 à 65 535. On parle de l'**adresse** de l'octet, tenant sur seize bits, soit sur deux octets (on dit aussi un **mot**).

11.1.2 Segmentation de la mémoire du 8086/8088

Lors de la conception du microprocesseur i8086, les autres microprocesseurs pouvaient accéder à au plus un KiO (ce qui correspond à 2^{16} octets, chaque octet étant repéré par une adresse de 16 bits, exigeant donc 16 broches d'adresse sur le microprocesseur). Cette capacité mémoire semble largement suffisante à l'époque. Les concepteurs du i8086 veulent 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.

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 i8086 (et donc du i8088) n'a une capacité de 20 bits ; ils ont tous une capacité de 16 bits. Les concepteurs du i8086 imaginent d'associer deux entiers de 16 bits chacun pour spécifier une adresse.

La mémoire 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 11.1). Pour spécifier une adresse, il faut donc 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 grâce à la formule suivante :

$$16 \times s + d.$$

Pour compliquer la vie du programmeur, et puisque tout est en 16 bits, les concepteurs du i8086 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).

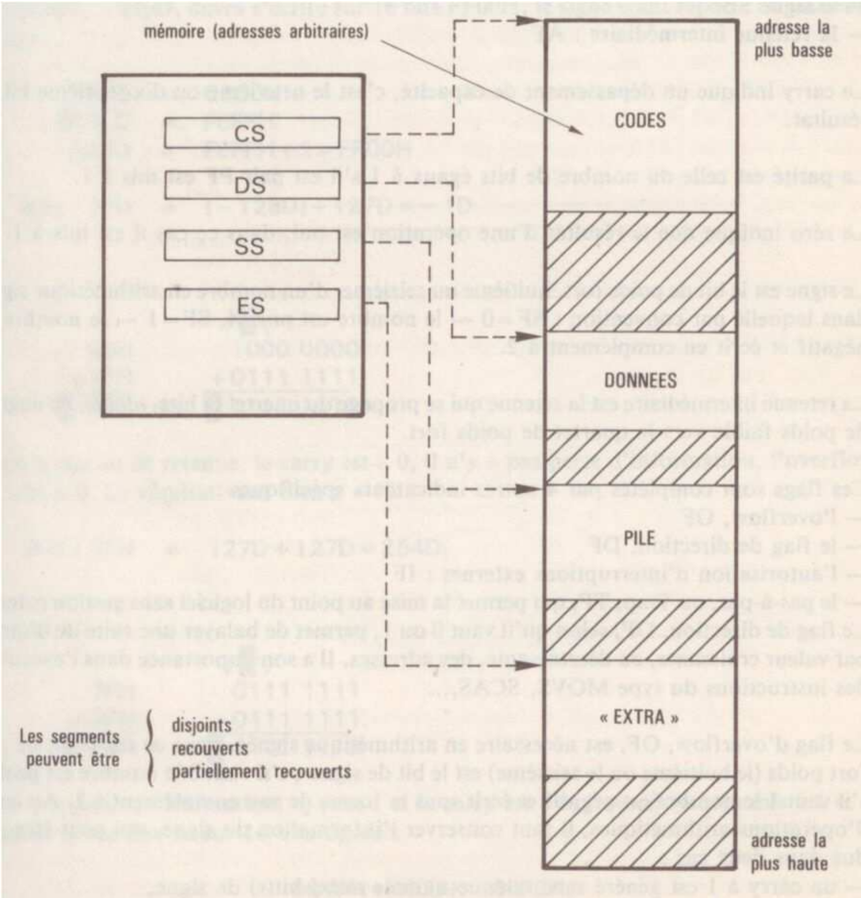

FIGURE 11.1 – Segmentation de la mémoire

11.2 Écriture et lecture en mémoire avec debug

11.2.1 Le logiciel debug

L'utilitaire `debug` du système d'exploitation MS-DOS [1] permet d'examiner et de changer le contenu de la mémoire, d'entrer un programme (en langage d'assemblage ou en langage machine) et de faire exécuter un tel programme. Commençons par voir comment activer `debug` et comment en sortir. Nous verrons au cours des sections suivantes des options de cet utilitaire.

Accès.- Pour accéder à `debug`, il suffit tout simplement d'écrire son nom après le prompteur :

```
C:> debug <retour>
```

Remarques.- Comme toujours avec MS-DOS on peut écrire indifféremment en minuscule ou en majuscule. Si la variable d'environnement `path` indique dans `autoexec.bat` où se trouve les commandes du MS-DOS, on peut accéder à `debug` depuis n'importe quel répertoire.

Réaction.- Le prompteur de `debug` apparaît alors, à savoir un tiret '-' :

```
C:> debug <retour>
-
```

Sortie.- La première commande de `debug` à connaître est évidemment de savoir comment en sortir. Il suffit d'écrire 'q' (pour l'anglais *Quit*) :

```
C:> debug <retour>
-q <retour>
C:>
```

11.2.2 Contenu en hexadécimal

Avec l'utilitaire `debug`, on utilise les entiers exprimés non pas en base dix (numération décimale) mais dans une base liée à la base deux (numération binaire), à savoir la base seize (numération hexadécimale).

On écrit un entier en base seize en utilisant les chiffres successifs '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' (comme en base dix puis) '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 'f' ou 'F' (pour quinze). Par exemple vingt s'écrira 14 (puisque vingt égal seize plus quatre).

11.2.3 Analyse d'une zone de mémoire avec debug

L'utilitaire `debug` nous permet d'analyser, c'est-à-dire lire, une zone de mémoire vive ou d'en changer le contenu. Commençons par la lecture.

Syntaxe.- On utilise pour cela la commande `D` (pour *to Dump*, littéralement *vider* (la mémoire) ; ce sens spécifique provient de l'utilisation des mémoires à tores de ferrite car on perdait alors le contenu de la mémoire en l'analysant), sous l'une des trois formes suivantes :

```
D
D <adresse de départ> <adresse de fin>
```

1. On retrouve cet utilitaire, exécutable dans une fenêtre d'émulation de terminal, sous *Windows* (32 bits) mais avec les restrictions dues au mode protégé, ce qui ne convient pas pour ce que nous voulons faire. Cet utilitaire n'est plus disponible sous *Windows* 64 bits.

```
D <adresse de départ> L <nombre d'octets>
```

avec `L` pour *Length* (longueur), où l'adresse est soit un décalage, soit une adresse de segment et un décalage séparés par ' :'. Les adresses et le nombre d'octets doivent être donnés en hexadécimal. Si l'adresse est un décalage, l'adresse de segment est celle contenue dans `DS`.

Dans le premier cas, au premier appel `debug` affiche à l'écran 128 octets consécutifs en commençant à l'adresse `DS:100`, sous la forme de 8 lignes de 16 octets, à la fois sous forme hexadécimale et sous forme ASCII (un point étant substitué à un caractère non affichable). Aux appels suivants on a les 128 octets suivants.

Exemple.- Pour afficher le contenu des 128 premiers octets de la mémoire, on utilise la commande suivante :

```
C:>debug
-D 0000:0000
0000:0000 9E 0F CA 00 65 04 70 00-16 00 AF 0F 65 04 70 00 ....e.p.....e.p.
0000:0010 65 04 70 00 54 FF 00 F0-4C E1 00 F0 6F EF 00 F0 e.p.T...L...o...
0000:0020 00 00 00 C8 D2 08 86 10-6F EF 00 F0 6F EF 00 F0 ........o...o...
0000:0030 6F EF 00 F0 6F EF 00 F0-9A 00 AF 0F 65 04 70 00 o...o.......e.p.
0000:0040 07 00 70 C8 4D F8 00 F0-41 F8 00 F0 67 25 5B FD ..p.M...A...g%[.
0000:0050 39 E7 00 F0 40 02 D2 03-2D 04 70 00 28 0A 67 0D 9...@...-.p.{.g.
0000:0060 A4 E7 00 F0 2F 00 71 10-6E FE 00 F0 04 06 67 0D ..../.q.n.....g.
0000:0070 1D 00 00 C8 A4 F0 00 F0-22 05 00 00 40 42 00 C0 ........"...@B..
-q
```

On remarque qu'une ligne commence par l'adresse du premier octet qui est affiché sur la ligne, suivie de seize octets (en hexadécimal) avec un tiret entre le huitième et le neuvième. La ligne se termine par l'affichage des codes ASCII correspondants dans le cas de caractères affichables et par un point sinon.

11.2.4 Changer une zone de mémoire avec `debug`

Syntaxe.- Pour changer le contenu de la mémoire vive avec `debug`, on utilise la commande `E` (pour l'anglais *Enter*) dont la syntaxe est la suivante :

```
E <adresse> <donnees>
```

ou :

```
E <adresse>
```

Dans le deuxième cas, l'ancienne valeur est affichée et on peut la changer.

Exemples.- 1^o) Entrons quelques valeurs (en hexadécimal) à partir de l'adresse 100 et vérifions que cela a bien été effectué en utilisant la commande `D` :

```
C>debug
-E 100 AE
-D 100
17A4:0100  AE 74 26 3C 22 74 22 3C-5C 74 1E 3C 3A 74 1A 3C   .t&<"t"<\t.<:t.<
-q
```

On vérifie bien que 'AE' est placé à l'adresse 100. On peut entrer plusieurs octets à la fois en listant ces octets.

- 2^o) On peut entrer les caractères ou les chaînes de caractères sans passer par l'hexadécimal en les encadrant par des apostrophes verticales ou des guillemets verticaux (au choix) :

```
C>debug
-E 100 'Paul Dubois'
-D 100
17A4:0100  50 61 75 6C 20 44 75 62-6F 69 73 3C 3A 74 1A 3C   Paul Dubois<:t.<
-q
```

On peut vérifier que la chaîne a bien été prise en compte, soit grâce à l'affichage hexadécimal (si on sait faire la correspondance), mais surtout grâce à l'affichage ASCII.

- 3^o) On peut directement vérifier la donnée et la changer ensuite. Si la commande est entrée avec une adresse et sans liste de données, `debug` suppose que l'on veut examiner cet octet de mémoire et le changer éventuellement. Il commence par afficher l'octet en question. Ensuite on a quatre possibilités :

1. On peut entrer une nouvelle valeur pour cet octet. `Debug` remplacera l'ancienne valeur par cette nouvelle valeur.
2. On peut appuyer sur la touche 'retour', ce qui indique que l'on ne désire pas changer la valeur.
3. On peut appuyer sur la barre d'espacement, ce qui laisse l'octet affiché inchangé, l'octet suivant est alors affiché, ce qui permet de le changer éventuellement.
4. On peut entrer le signe moins, '-', ce qui laisse l'octet affiché inchangé, l'octet précédent étant alors affiché, ce qui permet de le changer éventuellement.

Montrons une utilisation de la première possibilité :

```
C:>debug
-E 100
17A4:0100  41.42
-D 100
17A4:0100  42 72 74 68 75 72 75 62-6F 69 73 3C 3A 74 1A 3C   Brthurubois<:t.<
-q
```

<u>Mise en garde</u>.- Il faut être particulièrement vigilant lorsqu'on entre des valeurs en mémoire centrale avec `debug`. En effet placer des valeurs à un mauvais emplacement ou entrer de mauvaises valeurs peut causer des résultats imprévisibles. On ne causera pas de dommages importants cependant ; au pire il faudra redémarrer l'ordinateur.