Chapitre 2

Accès à la mémoire vive

Jusqu'à maintenant nous n'avons fait appel qu'à des registres comme éléments de mémoire. Cependant il y a un nombre limité très petit de registres. Pour un programme faisant appel à beaucoup de mémoire, nous aurons besoin d'utiliser la *mémoire vive* et même la *mémoire de masse*. Intéressons-nous pour l'instant à la mémoire vive.

Le microprocesseur `i8086` se différencie, de ce point de vue, de la plupart des autres microprocesseurs. Pour des raisons de compatibilité avec le microprocesseur phare antérieur de Intel, 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. Nous étudierons la mémoire segmentée au chapitre 7.

2.1 Modèle mémoire plate du i8085

2.1.1 La mémoire du i8085

Capacité mémoire.- Pour le i8085, la capacité maximale adressée de mémoire vive est 64 Ko, quantité qui paraissait largement suffisante à l'époque de la conception de ce microprocesseur.

Bien entendu un système donné ne disposait pas nécessairement de cette capacité maximale. On ajoutait 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**).

2.1.2 Chargement depuis la mémoire

Syntaxe.- Pour placer le contenu de l'emplacement mémoire d'adresse `adresse` dans un registre d'un octet, on utilise l'instruction :

```
mov registre, [adresse]
```

en entourant l'adresse de crochets et où `adresse` comprend au plus quatre chiffres hexadécimaux.

Adressages immédiat et direct.- On remarquera que l'adresse est entre crochets pour faire la distinction entre l'**adressage immédiat** (on place une constante dans un registre) et ce nouveau type d'adressage, appelé **adressage direct** (on place le contenu d'un emplacement mémoire dans un registre).

Exemple.- Pour obtenir la valeur de l'emplacement mémoire `1004h` on peut utiliser le programme suivant :

```
C:>debug
-a
249C:0100 mov al,[1004]
249C:0103 int 3
249C:0104
-g
AX=0006 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=0103 NV UP EI PL NZ NA PO NC
249C:0103 CC INT 3
-q
```

La valeur est donc apparemment `06h`, sans évidemment moyen de vérifier le résultat pour l'instant.

2.1.3 Stockage en mémoire

Syntaxe.- Pour placer le contenu d'un registre (de un octet) dans l'emplacement mémoire d'adresse `adresse` on utilise l'instruction :

```
mov [adresse], registre
```

Exemple.- Pour placer la valeur 80h dans l'emplacement mémoire 1004h on peut utiliser le programme suivant :

```
C:>debug
-a
249C:0100 mov al, 80
249C:0102 mov [1004],al
249C:0105 mov bl, [1004]
249C:0109 int 3
249C:010A
-g
AX=0080 BX=0080 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=0109 NV UP EI PL NZ NA PO NC
249C:0109 CC INT 3
-q
```

On a vérifié de plus que l'opération a bien été effectuée en plaçant le contenu de cet élément mémoire dans le registre BL.

Remarque.- Faites bien attention en effectuant une telle opération. Il ne s'agit plus démulation. On est bien en train de changer le contenu de la mémoire vive.

2.2 Manipulation des mots sur le i8086

2.2.1 Nouvelles caractéristiques du i8086

Les différences entre le i8085 et le i8086 sont de deux ordres :

- Les registres ont une capacité de 16 bits au lieu de 8 bits.
- La capacité mémoire (maximum) du i8086 a été portée de 64 Ko à 1 Mo, c'est-à-dire qu'il y a vingt broches de sortie au lieu de 16. Une **adresse physique** est donc maintenant comprise entre 00000h et FFFFFh.

Nous étudierons comment tenir compte de la nouvelle capacité mémoire au chapitre 7. Intéressons-nous ici à l'adressage des mots.

2.2.2 Adressage des mots

Nous avons vu que la plus petite quantité adressable avec un microprocesseur i8086 est l'octet. Cependant le bus des données est de seize bits. On doit donc pouvoir adresser directement un mot de seize bits, soit deux octets, sinon on aurait à utiliser deux instructions de copie alors qu'une seule est suffisante.
Syntaxe.- Il suffit pour cela d'utiliser un registre de deux octets (qui n'existaient pas sur le i8085) au lieu d'un registre d'un seul octet.

Exemple.- Réalisons un exemple de stockage et de chargement dans le même programme :

```
C:>debug
-a
15BD:0100 mov ax,180
15BD:0103 mov [1004], ax
15BD:0106 mov bx, [1004]
```

```
15BD:010A int3
15BD:010B
-g

AX=0180  BX=0180  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=15BD  ES=15BD  SS=15BD  CS=15BD  IP=010A   NV UP EI PL NZ NA PO NC
15BD:010A CC            INT     3
-q
```

2.2.3 Initialisation d'un élément mémoire

Introduction.- Nous avons vu comment transférer le contenu d'un registre dans un élément mémoire. On peut aussi vouloir initialiser directement un élément de mémoire, disons celui dont l'adresse est `11Eh`, avec la valeur `25h`, par exemple.

Comment faire ?

On peut, bien sûr, penser à l'instruction :

```
mov [11E],25
```

Cependant le microprocesseur n'acceptera pas cette instruction et réagira par une erreur (essayez avec `debug` !).

Pourquoi ?

L'instruction n'est pas suffisamment précise : s'agit-il de l'initialisation de l'octet d'adresse `11Eh` ou du mot d'adresse `11Eh`. Le résultat n'est pas le même.

Il faut donc un moyen de préciser cette instruction.

Syntaxe.- On a les instructions :

```
mov word ptr mem,val
mov byte ptr mem,val
```

pour initialiser l'emplacement mémoire d'adresse `mem` avec la valeur `val`. Dans le premier cas il s'agit d'un emplacement d'un mot, dans le second d'un emplacement d'un octet.

Exemples.- On utilise :

```
mov word ptr [11E],25
mov byte ptr [122],30
```

2.3 Affichage et écriture mémoire avec `debug`

Nous venons de voir les instructions du microprocesseur `i8086` permettant d'accéder à la mémoire vive. L'utilitaire `debug` nous permet d'analyser une zone de mémoire ou de remplir une telle zone en une seule commande. Ceci est utile pour notre étude des fonctionnalités du microprocesseur, bien que cela ne corresponde pas à de nouvelles instructions de celui-ci.

2.3.1 Analyse d'une zone de mémoire avec `debug`

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

où l'adresse est soit un décalage, soit une adresse de segment et un décalage. 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.

2.3.2 Principe du stockage des mots chez Intel

Introduction.- Les exemples précédents utilisent des données d'un octet. Dans ce cas, les octets sont stockés les uns après les autres en mémoire. Qu'arrive-t-il lorsqu'on manipule des mots de deux octets ?

Exemple.- Plaçons un mot à un emplacement déterminé à l'aide d'un programme, puis vérifions ce qui est placé grâce à la commande D de debug.

```
C:>debug
-a
167F:0100 mov ax,1100
167F:0103 mov [1000],ax
167F:0106 int 3
167F:0107
-g

AX=1100  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=167F  ES=167F  SS=167F  CS=167F  IP=0106   NV UP EI PL NZ NA PO NC
167F:0106 CC            INT     3
-d 1000
167F:1000  00 11 56 9C 8B FA 8B F2-E8 6B F4 A2 D1 E2 BF 8E   ..V......k......
167F:1010  DE 0A C0 74 03 BF E9 E5-B9 13 00 E8 1E F5 72 65   ...t..........re
167F:1020  C7 06 C0 DE 00 00 C6 06-C2 DE 00 80 3E D1 E2 00   ............>...
```

```
167F:1030  74 07 81 C6 04 01 E8 A8-F5 E8 52 00 3B 06 C0 DE   t.........R.;...
167F:1040  7E 15 A3 C0 DE BF C2 DE-B9 0D 00 FC F3 A4 3C 08   ~.............<.
167F:1050  75 05 E8 71 F5 EB 15 BF-8E DE 80 3E D1 E2 00 74   u..q.......>...t
167F:1060  03 BF E9 E5 B9 13 00 E8-FE F4 73 BF A1 C0 DE 80   ..........s.....
167F:1070  3E 6E DB 01 74 12 A0 6E-DB A3 C0 DE EB 0A 8B 16   >n..t..n........
-q
```

On remarque que les deux octets du mot commençant à l'adresse 1000 sont `00h` et `11h` et non le contraire comme on pourrait s'y attendre. Ceci est dû à la façon de stocker les mots.

Principe de stockage des mots.- Il y a deux façons de stocker un mot de deux octets, disons `AB`, à l'adresse n.

La *technique petitboutienne* (en anglais *little endian* d'après une histoire des *Voyages de Gulliver* sur la façon dont les œufs doivent être ouverts : par le petit bout ou par le grand bout) place `B` à l'adresse n et `A` à l'adresse $n+1$. La *technique grandboutienne* (en anglais *big endian*) place `A` à l'adresse n et `B` à l'adresse $n+1$.

Tous les microprocesseurs Intel et plusieurs mini-ordinateurs, tels que les VAX de Digital, utilisent la technique petitboutienne. Les microprocesseurs Motorola (utilisés dans les Macintosh) et les grands systèmes utilisent la technique grandboutienne.

Remarquons que cette différence dans le codage des mots pose un problème pour traduire un logiciel d'un système à l'autre.

2.3.3 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 ou des guillemets (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
```

Mise en garde.- 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 ; au pire il faudra redémarrer l'ordinateur.

2.4 Programmes avec données

Jusqu'ici nous avons seulement vu des programmes qui n'exigent pas de données. Ce n'est évidemment pas le cas général. Les données peuvent être demandées à l'utilisateur au moment opportun et saisies au clavier ; nous verrons comment traiter ce cas plus tard. Les données peuvent également être placées en mémoire au même moment que le chargement du code. C'est ce cas que nous allons traiter ici.

2.4.1 Un exemple

Le problème.- Écrivons un programme qui place une valeur dans le registre `ax`, lui ajoute une autre valeur et place le résultat en mémoire. Les deux valeurs proviendront également de la mémoire.

Le programme.- Ceci nous conduit, par exemple, au programme suivant :

```
C:\>debug
-a
17A4:0100 mov ax,[10B]
17A4:0103 add ax,[10D]
17A4:0107 mov [10F],ax
17A4:010A nop
```

```
17A4:010B db 14 23
17A4:010D db 05 00
17A4:010F db 00 00
17A4:0111
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0100   NV UP EI PL NZ NA PO NC
17A4:0100 A10B01        MOV     AX,[010B]                       DS:010B=2314
-t

AX=2314  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0103   NV UP EI PL NZ NA PO NC
17A4:0103 03060D01      ADD     AX,[010D]                       DS:010D=0005
-t

AX=2319  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0107   NV UP EI PL NZ NA PO NC
17A4:0107 A30F01        MOV     [010F],AX                       DS:010F=0000
-t

AX=2319  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=010A   NV UP EI PL NZ NA PO NC
17A4:010A 90            NOP
-
-d 100
17A4:0100  A1 0B 01 03 06 0D 01 A3-0F 01 90 14 23 05 00 19   ............#...
17A4:0110  23 8A D0 4E AD 8B C8 46-8A C2 24 FE 34 00 93 17   #..N...F..$.4...
17A4:0120  AC F3 AA EB 06 3C B2 75-6D F3 A4 8A C2 A8 01 74   .....<.um......t
17A4:0130  B1 BE 32 01 0E 1F 8B 1E-04 00 FC 33 D2 AD 8B C8   ..2........3....
17A4:0140  E3 13 8B C2 03 C3 8E C0-AD 8B F8 83 FF FF 74 11   ..............t.
17A4:0150  26 01 1D E2 F3 81 FA 00-F0 74 16 81 C2 00 10 EB   &........t......
17A4:0160  DC 8C C0 40 8E C0 83 EF-10 26 01 1D 48 8E C0 EB   ...@.....&..H...
17A4:0170  E2 8B C3 8B 3E 08 00 8B-36 0A 00 03 F0 01 06 02   ....>...6.......
-q
```

On vérifie bien que l'emplacement `10Fh` contient la somme `19h`.

2.4.2 Les directives de définition

Le programme ci-dessus contient une nouvelle instruction et des directives de définition :

- L'instruction `nop` (pour *No OPeration*) indique que l'opération à effectuer ne fait rien. Dans notre exemple elle sert artificiellement à séparer la fin du programme proprement dit des données. Elle sert en général pour effectuer une temporisation (cette instruction ne fait rien mais occupe quelques cycles).
- Les données suivent le programme. On ne peut pas se contenter d'y placer les valeurs car `debug` ne saurait pas s'il s'agit d'un octet ou d'un mot de deux octets. La valeur proprement dite est donc précédée de l'une des **directive de définition** `db` (pour *Define Byte*) ou `dw` (pour *Define Word*) pour indiquer s'il s'agit d'un octet ou d'un mot.

2.5 Programmer en code machine

Jusqu'à maintenant, nous avons utilisé des programmes en langage symbolique. Nous allons voir maintenant comment entrer directement un programme en code machine (le seul qui est compréhensible par le microprocesseur) en mémoire centrale, en utilisant une fois de plus `debug`.

Nous n'allons pas voir comment concevoir un tel programme (nous le verrons plas tard), mais seulement comment l'introduire en mémoire centrale et l'exécuter.

2.5.1 Cas d'un programme sans données

Commençons par un programme n'utilisant pas de données.

Le programme.- Considérons le programme suivant, écrit en langage symbolique :

```
mov ax,0123
add ax,0025
mov bx,ax
add bx,ax
mov cx,bx
sub cx,ax
sub ax,ax
nop
```

La traduction de ce programme en code machine est :

```
B82301
052500
8BD8
03D8
8BCB
2BC8
2BC0
90
```

Nous sommes passé à la ligne pour plus de lisibilité mais, bien entendu le programme est une suite de 0 et de 1 ou, de façon plus lisible pour les êtres humains une suite de nombres hexdécimaux.

Nous nous sommes ici servi de la commande d'assemblage de `debug` pour écrire le programme en langage machine, à partir du programme écrit en langage symbolique. Nous verrons plus tard les principes de la traduction et comment le faire à la main (bien que ce ne soit pas une mince affaire).

Introduction du programme avec `debug`.- On entre le programme comme des données, comme nous l'avons vu ci-dessus :

```
C:\>debug
-E CS:100 B8 23 01 05 25 00
-E CS:106 8B D8 03 D8 8B CB
-E CS:10C 2B C8 2B C0 90
-
```

Nous avons choisi d'entrer six octets à chaque fois mais nous pouvons en entrer le nombre que nous voulons. Si on se trompe (et cela risque d'arriver souvent), il suffit de recommencer pour la ligne en question, il suffit même de le faire pour le seul octet erroné.

Exécution du programme.- Nous pouvons alors exécuter le programme comme d'habitude à l'aide d'une des commandes `T` ou `G` de `debug`. Dans le cas de notre programme nous ne pouvons pas vraiment utiliser la commande `G` puisqu'il n'y a pas de point d'arrêt. Utilisons donc la commande `T`, en commençant par une commande `R` de façon à voir le contenu initial des registres et la première instruction qui sera exécutée :

```
C:\>debug
-E CS:100 B8 23 01 05 25 00
-E CS:106 8B D8 03 D8 8B CB
-E CS:10C 2B C8 2B C0 90
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0100   NV UP EI PL NZ NA PO NC
17A4:0100 B82301        MOV     AX,0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0103   NV UP EI PL NZ NA PO NC
17A4:0103 052500        ADD     AX,0025
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0106   NV UP EI PL NZ NA PE NC
17A4:0106 8BD8          MOV     BX,AX
-T

AX=0148  BX=0148  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0108   NV UP EI PL NZ NA PE NC
17A4:0108 03D8          ADD     BX,AX
-T

AX=0148  BX=0290  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=010A   NV UP EI PL NZ AC PE NC
17A4:010A 8BCB          MOV     CX,BX
-T

AX=0148  BX=0290  CX=0290  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=010C   NV UP EI PL NZ AC PE NC
17A4:010C 2BC8          SUB     CX,AX
-T

AX=0148  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=010E   NV UP EI PL NZ AC PE NC
17A4:010E 2BC0          SUB     AX,AX
-T

AX=0000  BX=0290  CX=0148  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0110   NV UP EI PL ZR NA PE NC
17A4:0110 90            NOP
-q
```

On vérifie sans problème que le programme est correctement exécuté.

2.5.2 Programme avec données

Introduction.- Nous venons de voir comment entrer directement du code machine. Voyons maintenant comment entrer des données et du code machine.

Le problème.- Utilisons un programme analogue sauf que, au lieu d'initialiser le registre `AX`, on va chercher une valeur en mémoire centrale (ce qui correspond à

une donnée). Plus exactement, on va aller chercher une valeur à un emplacement mémoire pour le placer dans le registre `AX`, lui ajouter une valeur placée dans un autre emplacement mémoire et placer le résultat ainsi obtenu à un troisième emplacement mémoire.

Où placer les données ?.- On peut placer les données où l'on veut en mémoire centrale, du moment qu'il n'y a pas interpénétration entre données et code. En général on les place dans le segment des données dont le segment est repéré par le registre `DS` et le décalage est indiqué directement.

Le programme.- Plaçons les valeurs `0123h`, `0025h` et `0000h` aux emplacements `200h`, `202h` et `204h` du segment des données. On ne commence ni à 0 ni à `100h` puisque nous avons vu que, par défaut, le segment des données est le même que celui du code ; on évite ainsi les interpénétrations (notre code est suffisamment court pour ne pas atteindre `200h`). Les adresses des emplacements sont incrémentées de deux en deux, et non de un en un, puisqu'on place des mots (de deux octets) et non des octets. N'oublions pas d'entrer dans l'ordre inverse.

Le programme s'écrit sous forme symbolique :

```
mov ax,[200]
add ax,[202]
mov [204],ax
nop
```

soit en code machine :

```
A10002
03060202
A30402
90
```

On obtient, par exemple, ce code en utilisant `debug` :

```
C:\>debug
-a
17A4:0100 mov ax,[200]
17A4:0103 add ax,[202]
17A4:0107 mov [204],ax
17A4:010A nop
17A4:010B
-u
17A4:0100 A10002        MOV     AX,[0200]
17A4:0103 03060202      ADD     AX,[0202]
17A4:0107 A30402        MOV     [0204],AX
17A4:010A 90            NOP
```

Introduction des données et du programme.- Commençons par entrer le programme :

```
C:\>debug
-E CS:100 A1 00 02 03 06 02 02
-E CS:107 A3 04 02 90
```

```
-
```

puis entrons des données :

```
-E DS:0200 23 01 25 00 00 00
-
```

Exécution du programme.- Nous pouvons alors exécuter le programme comme dans le cas de notre premier exemple :

```
C:\>debug
-E CS:100 A1 00 02 03 06 02 02
-E CS:107 A3 04 02 90
-E DS:0200 23 01 25 00 00 00
-R
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0100   NV UP EI PL NZ NA PO NC
17A4:0100 A10002        MOV     AX,[0200]                       DS:0200=0123
-T

AX=0123  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0103   NV UP EI PL NZ NA PO NC
17A4:0103 03060202      ADD     AX,[0202]                       DS:0202=0025
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=0107   NV UP EI PL NZ NA PE NC
17A4:0107 A30402        MOV     [0204],AX                       DS:0204=0000
-T

AX=0148  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17A4  ES=17A4  SS=17A4  CS=17A4  IP=010A   NV UP EI PL NZ NA PE NC
17A4:010A 90            NOP
-q
```