Chapitre 22

Les chaînes de caractères

La plupart des ordinateurs modernes ne sont plus orientés calculs mais orientés mots. Il est donc important de savoir comment traiter ceux-ci.

On peut toujours coder les mots et simuler les opérations sur ceux-ci. Mais, devant l'importance de la manipulation des mots par ordinateurs de nos jours, les concepteurs des microprocesseurs s'adaptent et créent des primitives pour celles-ci.

22.1 Définition des chaînes de caractères

Nous avons déjà vu ce qu'est, mathématiquement, un *mot* : il s'agit d'une suite finie sur un *alphabet*, ensemble fini non vide quelconque. En informatique on préfère parler de *chaînes de caractères* (*string* en anglais) pour insister sur le fait qu'on peut, en particulier, compter l'espace comme un caractère.

22.1.1 Représentation des caractères

Une chaîne de caractères étant une suite de caractères, il faut d'abord s'interroger sur la façon de représenter les caractères.

Le microprocesseur ne connaît pas la notion de caractères. Il ne connaît que les **codes** des caractères, qui peuvent être considérés comme des entiers.
Combien faut-il d'entiers pour représenter les caractères ? Il y a vingt-six lettres en français, mais sans distinguer les minuscules des majuscules, et sans tenir compte des signes diacritiques : 'é' n'est pas la même chose que 'e' ou 'è'. De plus on a besoin des chiffres, des signes de ponctuation et de quelques signes spéciaux. Nous avons vu que l'on utilise de nos jours Unicode avec la possibilité de 65 536 caractères mais à l'époque de la conception des premiers microprocesseurs, on se contentait de 256 caractères.

Pour le 8086/8088, un caractère est codé sur un octet. Qu'importe le code utilisé. C'est à l'utilisateur (dans la pratique au concepteur du système d'exploitation) de déterminer celui-ci.

22.1.2 Représentation des chaînes de caractères

Définition.- *Une* **chaîne de caractères** *est une suite d'octets d'adresses consécutives.*

Une chaîne de caractères est entièrement déterminée par son *adresse* et sa *longueur*.

Adresse d'une chaîne de caractères.- Soit la suite d'octets $w = o_1 o_2 ... o_n$. Si l'adresse, en mémoire centrale, du premier octet o_1 est p alors celle du second octet o_2 est $p + 1$, celle du troisième octet o_3 est $p + 2$, ... , celle du n-ième octet o_n est $p + n - 1$. Il suffit donc de connaître l'adresse du premier octet, p, qui est appelée l'**adresse de la chaîne de caractères**.
On remarquera que les octets sont numérotés naturellement de gauche à droite, contrairement aux bits d'un octet ou d'un mot, qui sont eux numérotés de droite à gauche.

Longueur d'une chaîne de caractères.- La détermination d'une chaîne de caractères en mémoire centrale se fait par son adresse mais aussi par sa **longueur** n, ce qui permet de déterminer la fin de celle-ci.

Exemple.- La chaîne de caractères "Bonjour" de longueur 7 à l'adresse 1024 sera représentée de la façon suivante en mémoire :

```
1024 'B'
1025 'o'
1026 'n'
1027 'j'
1028 'o'
1029 'u'
1030 'r'
```

22.1.3 Jeu de caractères utilisé

Les caractères sont des octets, ils sont donc limités à 256. Mais quels sont ces caractères ? Le microprocesseur ne se prononce pas : ils sont repérés par des numéros de 0 à 255, il appartiendra ensuite au concepteur du système d'exploitation de les interpréter, en particulier grâce à la *ROM caractère*.

Dans le cas de l'IBM-PC, il s'agit de l'alphabet ASCII étendu de deux façons : les caractères 0 à 31 sont remplacés par des caractères (dits **semi-graphiques**) propres à IBM et il y a le choix d'un ASCII national pour les caractères de 128 à 255.

22.2 Adressage indirect par registre et tableaux

Vous avez vu la notion de *tableau* dans votre cours d'initiation à la programmation et les services qu'elle rend au programmeur. Cette notion, très utile dans les langages évolués, n'existe pas en tant que telle en langage machine. Voyons ici la notion d'*adressage indirect*, plus particulièrement l'*adressage indirect par registre*, et comment celui-ci permet de mettre en place facilement les tableaux d'octets, autrement dit les chaînes de caractères.

22.2.1 Notion

Notion d'adressage indirect.- Jusqu'à maintenant nous avons manipulé des constantes (*adressage immédiat*), des valeurs se trouvant dans un registre (*adressage registre*) et des valeurs se trouvant à un emplacement donné de la mémoire vive (*adressage direct*).

Une autre façon de faire est de désigner un emplacement mémoire (de deux octets) qui contient non pas la valeur désirée mais l'adresse (en mémoire vive, c'est la raison pour laquelle on a besoin de deux octets) contenant la valeur désirée. On parle alors d'**adressage indirect**.

Syntaxe de l'adressage indirect par registre en langage symbolique.- Comme nous l'avons déjà vu, en langage symbolique, pour désigner une constante on écrit celle-ci en hexadécimal, pour une valeur se trouvant dans un registre on écrit le nom du registre, pour une valeur se trouvant à un emplacement de la mémoire centrale on indique l'adresse (une constante hexadécimale), ou plus exactement son déplacement, de cet emplacement entre crochets.

Pour une valeur se trouvant à l'adresse (de mémoire vive) désignée par le contenu d'un registre (de seize bits, puisque c'est la taille du déplacement), on indique le nom de ce registre entre crochets, par exemple `[BX]`. On parle alors d'**adressage indirect par registre**. On a donc :

Registres concernés.- On ne peut pas utiliser n'importe quel registre pour l'adressage indirect par registre. Les seuls registres utilisables pour le 8086/8088 sont `BX`, `BP`, `DI` et `SI`.

Les registres permis ne proviennent pas d'une restriction logique mais d'une question d'économie de câblage du microprocesseur : on gagne sur le nombre de transistors en ne donnant pas toutes les fonctionnalités à chaque registre mais en les spécialisant quelque peu.

Indication de la taille de l'adressage indirect en langage symbolique.- L'adressage indirect peut se faire pour des octets mais aussi pour des mots (deux octets) ou des mots doubles (quatre octets).

Lorsque cela n'est pas clair on utilise les **préfixes** suivants dans le cas du langage symbolique (il s'agit de codes d'opération différents pour le langage machine) :

- `byte ptr` pour un octet ;

– `word ptr` pour un mot ;
– `dword ptr` pour un mot double.

Par exemple :

```
MOV AL, [BX]
```

est visiblement uns instruction de copie d'un octet mais :

```
MOV [BX], 10
```

est ambigu. On doit donc préciser, par exemple :

```
MOV BYTE PTR [BX], 10
```

Exemple.- Avec le mot 'Bonjour' écrit à l'adresse indiquée ci-dessus, la portion de programme suivant :

```
MOV BX, 1024
MOV CL, [BX]
CMP BYTE PTR [BX], 0
```

placera 'B' dans le registre CL et le pointeur ZF ne sera pas nul puisque [BX] ne contient pas le caractère nul lors de la dernière ligne.

Adresse physique.- Lors de l'adressage indirect on indique l'adresse logique et, en fait, que le décalage de celle-ci. Comme d'habitude, si ceci n'est pas suffisant, l'adresse logique complète, et donc l'adresse physique, est obtenue en utilisant l'adresse de segment indiquée dans le registre DS pour les registres BX et SI, dans le registre ES pour DI.

Langage machine.- Nous avons déjà vu, sans nécessiarement en comprendre toutes les subtibilités, comment on code ceci en langage machine. Pour un octet :

```
| mod reg r/m |
```

on utilise `mod = 00`, le registre dépendant de la valeur de `r/m` :

r/m	AE
100	[SI]
101	[DI]
111	[BX]

22.2.2 Un exemple

Problème.- Écrivons un sous-programme qui commence par placer 'Bonjour' à partir de l'adresse `0002h`, suivi de la valeur sentinelle '0', puis qui compte le nombre de caractères de ce mot, c'est-à-dire le nombre de caractères depuis le début jusqu'à '0' non inclus, qui place la longueur dans le registre DX puis la reporte à l'emplacement mémoire `0000h`.

Langage symbolique.- Le sous-programme s'écrit de la façon suivante :

```
MOV BYTE PTR CS:[02], 42
MOV BYTE PTR CS:[03], 6F
MOV BYTE PTR CS:[04], 6E
MOV BYTE PTR CS:[05], 6A
MOV BYTE PTR CS:[06], 6F
MOV BYTE PTR CS:[07], 75
```

```
MOV BYTE PTR CS:[08], 72
MOV BYTE PTR CS:[09], 0
MOV DX, 0
MOV BX, 02
debut: MOV AL, CS:[BX]
CMP AL, 0
JE FIN
INC BX
INC DX
JMP debut
FIN: MOV CS:[00], DX
RETF
```

Langage machine.- La traduction de ce sous-programme en langage machine donne :

```
2E C6 06 02 00 42
2E C6 06 03 00 6F
2E C6 06 04 00 6E
2E C6 06 05 00 6A
2E C6 06 06 00 6F
2E C6 06 07 00 75
2E C6 06 08 00 72
2E C6 06 09 00 00
BA 00 00
BB 02 00
2E 8A 07
3C 00
74 04
43
42
EB F5
2E 89 16 00 00
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 10 octets et le code 71 octets :

```
CLS
DIM PM%(40)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 70
  READ Octet$
  POKE OffPM% + 10 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
```

```
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 10)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
PRINT "Longueur = "; PM%(0)
END
'Code machine
DATA 2E,C6,06,02,00,42
DATA 2E,C6,06,03,00,6F
DATA 2E,C6,06,04,00,6E
DATA 2E,C6,06,05,00,6A
DATA 2E,C6,06,06,00,6F
DATA 2E,C6,06,07,00,75
DATA 2E,C6,06,08,00,72
DATA 2E,C6,06,09,00,00
DATA BA,00,00
DATA BB,02,00
DATA 2E,BA,07
DATA 3C,00
DATA 74,04
DATA 43
DATA 42
DATA EB,F5
DATA 2E,89,16,00,00
DATA CB
```

L'exécution :

```
0     0     0     0     0
0     0     0     0     0
2E    C6    6     2     0

7     0     42    6F    6E
6A    6F    75    72    0
2E    C6    6     2     0

Longueur = 7
```

donne le résultat attendu.

22.3 Primitives de manipulation des mots

22.3.1 Copie

L'une des manipulations les plus courantes concernant les chaînes de caractères est la copie d'une telle chaîne d'un emplacement à un autre de la mémoire vive. On peut évidemment réaliser une telle copie avec les instructions que nous connaissons déjà. On peut également utiliser les deux nouvelles instructions MOVS et REP pour cela, ce qui aura pour conséquence de donner un code plus compact.

Exercice.- *Écrire un programme en langage machine permettant de copier la chaîne de caractères d'adresse contenue dans* AX *et de longueur contenue dans* CX *à l'adresse contenue dans* BX*.*

22.3.1.1 Copie d'un élément

On sait copier un élément (octet ou mot) d'un emplacement à un autre. Il existe cependant une instruction spécialisée pour ce faire, qui est surtout utile en conjonction avec une instruction de répétition spécialisée que nous verrons ensuite.

Syntaxe.- L'instruction `movs` (pour *MOVe a String*) se décline sous les deux formes :

```
MOVSB
```

et :

```
MOVSW
```

pour copier un octet ou un mot.

Paramètres.- Bien entendu il manque des informations dans l'instructions ci-dessus. Celles-ci sont apportées par les registres DS, SI, ES et DI.

Le couple de registres DS :SI (avec SI pour *Source Index*) doit contenir l'adresse de l'élément à copier. Le couple de registres ES :DI (avec DI pour *Destination Index*) doit contenir l'adresse à laquelle il faut copier l'élément.

Langage machine.- Ces instructions sont codées sur un octet par :

```
| 1010 010w |
```

avec `w = 0` pour la première et `w = 1` pour la seconde, soit A4h et A5h respectivement.

Après le transfert, SI et DI sont incrémentés si DF est nul ou décrémenytés si DF vaut un. Ceci n'a pas vraiment d'importance pour l'instant, mais en aura lorsqu'on utilisera l'instruction en conjonction avec REP.

Exemple.- Écrivons un programme machine qui place 'A' à l'emplacement mémoire 00h du tableau réservé pour le code et qui copie cette valeur à l'emplacement 01h.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV BYTE PTR CS:[00], 41
MOV AX, 0
MOV SI,AX
MOV AX,CS
MOV DS,AX
```

```
MOV ES,AX
MOV AX, 01
MOV DI,AX
MOVSB
POP ES
POP DS
RETF
```

sans oublier de sauvegarder la valeur de DS (et, dans une moindre mesure, de ES) sur la pile puisque celle-ci est changée dans le programme, puis de la récupérer à la fin.

La traduction de ce sous-programme en langage machine donne :

```
1E
06
2E C6 06 00 00 41
B8 00 00
89 C6
8C C8
8E D8
8E C0
B8 01 00
89 C7
A4
07
1F
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 2 octets et le code 28 octets :

```
CLS
DIM PM%(14)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 27
  READ Octet$
  POKE OffPM% + 2 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 2)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
```

```
NEXT I%
'Retour au segment de depart
DEF SEG
END
'Code machine
DATA 1E
DATA 06
DATA 2E,C6,06,00,00,41
DATA B8,00,00
DATA 89,C6
DATA 8C,C8
DATA 8E,D8
DATA 8E,C0
DATA B8,01,00
DATA 89,C7
DATA A4
DATA 07
DATA 1F
DATA CB
```

L'exécution :

```
0     0     1E    6     2E
C6    6     0     0     41
B8    0     0     89    C6

41    41    1E    6     2E
C6    6     0     0     41
B8    0     0     89    C6
```

donne le résultat attendu.

22.3.1.2 Copie répétée

Langage symbolique.- Le préfixe REP (pour *REPete*) permet de répéter un certain nombre de fois l'instruction qui suit, ce nombre de fois étant la valeur du registre CX (pour l'anglais *Count eXtended register*).

Elle peut être employée en conjonction avec certaines opérations concernant les chaînes de caractères (MOVS, CMPS, SCAS, STOS) et donc, en particulier, avec des deux instructions définies précédemment :

```
REP MOVSB
```

ou :

```
REP MOVSW
```

Langage machine.- Ce préfixe est codé sur un octet par :

```
| 1111 001z |
```

la valeur de `z` n'ayant de sens que pour CMPS et SCAS, ce qui ne nous intéresse pas pour l'instant, soit F2h ou F3h.

Exemple.- Écrivons un programme en langage machine qui permette d'écrire '`bonjour`' à l'adresse 00h du tableau réservé pour le code, y compris son terminateur 0, puis de copier le contenu de cette chaîne de caractères à l'adresse 08h.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV BYTE PTR CS:[00], 42
MOV BYTE PTR CS:[01], 6F
MOV BYTE PTR CS:[02], 6E
MOV BYTE PTR CS:[03], 6A
MOV BYTE PTR CS:[04], 6F
MOV BYTE PTR CS:[05], 75
MOV BYTE PTR CS:[06], 72
MOV BYTE PTR CS:[07], 0
MOV AX, 00
MOV SI, AX
MOV AX, CS
MOV DS, AX
MOV ES, AX
MOV AX, 08
MOV DI, AX
MOV CX, 08
REP MOVSB
POP ES
POP DS
RETF
```

sans oublier de sauvegarder la valeur de DS (et, dans une moindre mesure, de ES) sur la pile puisque celle-ci est changée dans le programme, puis de la récupérer à la fin.

La traduction de ce sous-programme en langage machine donne :

```
1E
06
2E C6 06 00 00 42
2E C6 06 01 00 6F
2E C6 06 02 00 6E
2E C6 06 03 00 6A
2E C6 06 04 00 6F
2E C6 06 05 00 75
2E C6 06 06 00 72
2E C6 06 07 00 00
B8 00 00
89 C6
8C C8
8E D8
8E C0
B8 08 00
89 C7
B9,08,00
```

```
F3 A4
07
1F
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 16 octets et le code 74 octets :

```
CLS
DIM PM%(45)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 73
  READ Octet$
  POKE OffPM% + 16 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 16)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
END
'Code machine
DATA 1E
DATA 06
DATA 2E,C6,06,00,00,42
DATA 2E,C6,06,01,00,6F
DATA 2E,C6,06,02,00,6E
DATA 2E,C6,06,03,00,6A
DATA 2E,C6,06,04,00,6F
DATA 2E,C6,06,05,00,75
DATA 2E,C6,06,06,00,72
DATA 2E,C6,06,07,00,00
DATA B8,00,00
DATA 89,C6
DATA 8C,C8
DATA 8E,D8
DATA 8E,C0
DATA B8,01,00
DATA 89,C7
```

```
DATA B9,08,00
DATA F3,A4
DATA 07
DATA 1F
DATA CB
```

L'exécution :

```
0       0       0       0       0
0       0       0       0       0
0       0       0       0       0

42      6F      6E      6A      6F
75      72      0       42      6F
6E      6A      6F      75      72
```

donne le résultat attendu.

22.3.2 Remplissage

On a souvent à remplir une partie de la mémoire avec le même octet, par exemple avec des '0' ou des espaces (dans le cas de l'effacement de l'écran, par exemple). Les concepteurs du 8086 ont donc implémenté une telle primitive, sous le nom de STOS pour *STOre a String*.

22.3.2.1 Stockage d'un élément

Syntaxe.- Les deux instructions :

`STOSB`

et :

`STOSW`

(pour *STOre a String*) permettent de stocker, respectivement, l'octet contenu dans le registre AL ou le mot contenu dans le registre AX à l'adresse ES :DI.

Langage machine.- Ces instructions sont codées sur un octet par :

`| 1010 101w |`

avec `w = 0` pour la première et `w = 1` pour la seconde, soit A4h et A5h respectivement.

Après le transfert, SI est incrémenté de 1 pour AL, de 2 pour AX, si l'indicateur de direction DF est nul ; sinon il y a décrémentation si DF vaut un. Ceci n'a pas vraiment d'importance pour l'instant, mais en aura lorsqu'on utilisera l'instruction en conjonction avec le préfixe REP.

Exemple.- Écrivons un programme machine qui place 'A' à l'emplacement mémoire 00h du tableau réservé pour le code.

Le sous-programme en langage symbolique s'écrit :

```
PUSH ES
MOV AX, 0
MOV SI,AX
MOV AX,CS
MOV ES,AX
MOV AL, 41
```

```
STOSB
POP ES
RETF
```

La traduction de ce sous-programme en langage machine donne :

```
06
B8 00 00
89 C7
8C C8
8E C0
B0 41
AA
07
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 1 octet (étendu à 2) et le code 15 octets :

```
CLS
DIM PM%(8)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 14
  READ Octet$
  POKE OffPM% + 2 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 2)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
END
'Code machine
DATA 06
DATA B8,00,00
DATA 89,C7
DATA 8C,C8
DATA 8E,C0
DATA B0,41
```

```
DATA AA
DATA 07
DATA CB
```

L'exécution :

```
0       0       6       B8      0
0       89      C7      8C      C8
8E      C0      B0      41      AA

41      0       6       B8      0
0       89      C7      8C      C8
8E      C0      B0      41      AA
```

donne le résultat attendu.

22.3.2.2 Remplissage d'une zone

Syntaxe.- Les deux instructions :

```
REP STOSB
```

et :

```
REP STOSW
```

permettent de remplir la zone d'adresse ES :DI et de longueur celle contenue dans le registre CX avec, respectivement, l'octet se trouvant dans le registre AL ou le mot dans le registre AX.

Exemple.- Écrivons un programme machine qui place 'A' dans les huit premières cases à partir de l'emplacement mémoire 00h du tableau réservé pour le code.

Le sous-programme en langage symbolique s'écrit :

```
PUSH ES
MOV AX, 0
MOV DI,AX
MOV AX,CS
MOV ES,AX
MOV AL, 41
MOV CX, 08
REP STOSB
POP ES
RETF
```

La traduction de ce sous-programme en langage machine donne :

```
06
B8 00 00
89 C7
8C C8
8E C0
B0 41
B9 08 00
F3 AA
07
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 8 octets et le code 19 octets :

```
CLS
DIM PM%(13)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 18
  READ Octet$
  POKE OffPM% + 2 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 8)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
END
'Code machine
DATA 06
DATA B8,00,00
DATA 89,C7
DATA 8C,C8
DATA 8E,C0
DATA B0,41
DATA B9,08,00
DATA F3,AA
DATA 07
DATA CB
```

L'exécution :

```
0      0      0      0      0
0      0      0      6      B8
0      0      89     C7     8C

41     41     41     41     41
41     41     41     6      B8
0      0      89     C7     8C
```

donne le résultat attendu.

22.3.3 Chargement

Nous avons vu comment copier le contenu de l'accumulateur vers un emplacement mémoire, et même un groupe d'emplacements mémoire. On peut vouloir le contraire : copier d'un emplacement mémoire vers l'accumulateur.

On peut évidemment faire ceci avec l'instruction `MOV`. Une instruction nouvelle a été introduite par les concepteurs du 8086 qui permet en plus d'incrémenter (ou décrémenter) la valeur du registre SI. Il s'agit de **LODS** pour l'anglais *LOaD a String*.

22.3.3.1 Chargement d'un élément

Langage symbolique.- On utilise les deux formes :

```
LODSB
```

et :

```
LODSW
```

pour charger, respectivement, l'octet ou le mot d'adresse DS :SI dans le registre AL ou AX, suivant le cas.

Langage machine.- Ces instructions sont codées sur un octet par :

```
| 1010 110w |
```

avec `w = 0` pour la première et `w = 1` pour la seconde.

Après le transfert, SI est incrémenté de 1 pour AL, de 2 pour AX, si l'indicateur de direction DF est nul ; sinon il y a décrémentation si DF vaut un. Ceci n'a pas vraiment d'importance pour l'instant, mais en aura lorsqu'on utilisera l'instruction en conjonction avec LOOP.

Exemple.- Écrivons un programme QBasic machine qui place 'A' dans le premier élément du tableau réservé pour le code puis fait appel à un sous-programme en langage machine qui récupère ce caractère, l'incrémente (donc le transforme en 'B') et le place en 01h du tableau.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX, 0
MOV SI,AX
MOV AX,CS
MOV ES,AX
MOV DS,AX
MOV AX, 1
MOV DI,AX
LODSB
INC AL
STOSB
POP ES
POP DS
RETF
```

La traduction de ce sous-programme en langage machine donne :

```
1E
06
```

```
B8 00 00
89 C6
8C C8
8E C0
8E D8
B8 01 00
89 C7
AC
FE C0
AA
07
1F
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 2 octets et le code 25 octets :

```
CLS
DIM PM%(13)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
PM%(0) = &H41
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 24
  READ Octet$
  POKE OffPM% + 2 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 2)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
END
'Code machine
DATA 1E
DATA 06
DATA B8,00,00
DATA 89,C6
DATA 8C,C8
DATA 8E,C0
DATA 8E,D8
```

```
DATA B8,01,00
DATA 89,C7
DATA AC
DATA FE,C0
DATA AA
DATA 07
DATA 1F
DATA CB
```

L'exécution :

```
41    0     1E    6     B8
0     0     89    C6    8C
C8    8E    C0    8E    D8

41    42    1E    6     B8
0     0     89    C6    8C
C8    8E    C0    8E    D8
```

donne le résultat attendu, puisque le deuxième octet est égal à 42h.

22.3.3.2 Chargement d'une zone avec transformation

Contexte.- On n'utilise pas ces instructions avec le préfixe REP, puisqu'on surchargerait l'accumulateur. On les utilise, en liaison avec STOS pour une copie avec changement durant le transfert (comme nous l'avons vu sur l'exemple précédent). Dans ce cas le nombre de répétitions est contrôlé par l'une des instructions de boucle suivantes.

Langage symbolique.- 1^o) L'instruction :

```
LOOP décalage
```

décrémente le contenu du registre CX et provoque un saut relatif court si le contenu de CX n'est pas nul, `décalage` étant compris entre - 128 et 127.

- 2^o) L'instruction :

```
LOOPE décalage
```

ou :

```
LOOPZ décalage
```

décrémente le contenu du registre CX et provoque un saut relatif court si l'indicateur ZF vaut 1 et si le contenu de CX n'est pas nul, autrement dit on sort de la boucle si (CX) = 0 ou si ZF = 0.

Cette instruction permet, par exemple, de rechercher le premier terme non nul dans une suite de n octets.

- 3^o) L'instruction :

```
LOOPNE décalage
```

ou :

```
LOOPNZ décalage
```

décrémente le contenu du registre CX et provoque un saut relatif court si l'indicateur ZF vaut 0 et si le contenu de CX n'est pas nul, autrement dit on sort de la boucle si (CX) = 0 ou si ZF = 1.

Cette instruction permet, par exemple, de rechercher le premier terme nul dans une suite de n octets.

<u>Langage machine</u>.- 1^o) LOOP est codée sur deux octets par :

```
| 1110 0010 | décalage |
```

soit E2h suivi du décalage.

- 2^o) LOOPE/LOOPZ est codée sur deux octets par :

```
| 1110 0001 | décalage |
```

soit E1h suivi du décalage.

- 3^o) LOOPNE/LOOPNZ est codée sur deux octets par :

```
| 1110 0000 | décalage |
```

soit E0h suivi du décalage.

<u>Exemple</u>.- Dans le cas du code ASCII, les minuscules 'a', ... , 'z' ont les mêmes numéros que les majuscules 'A', ... , 'Z' correspondantes avec `20h` de plus.

Plaçons une chaîne de caractères écrite exclusivement en majuscule, formée que de 'A' pour simplifier, à l'adresse de décalage 00h du tableau réservé au code du programme machine de longueur 8h puis transférons-la à l'adresse de décalage 08h en transformant les majuscules en minuscules.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,CS
MOV ES,AX
MOV DS,AX
MOV AX, 0
MOV DI,AX
MOV CX, 8
MOV AL, 41
REP STOBS
MOV AX, 0
MOV SI,AX
MOV AX, 8
MOV DI,AX
MOV CX, 8
DEBUT: LODSB
ADD AL, 20
STOSB
LOOP DEBUT
POP ES
POP DS
RETF
```

La traduction de ce sous-programme en langage machine donne :

```
1E
06
```

```
8C C8
8E C0
8E D8
B8 00 00
89 C7
B9 08 00
B0 41
F3 AA
B8 00 00
89 C6
B8 08 00
89 C7
B9 00 00
AC
04 20
AA
E2 FA
07
1F
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 16 octets et le code 42 octets :

```
CLS
DIM PM%(28)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 41
  READ Octet$
  POKE OffPM% + 16 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 16)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
END
'Code machine
DATA 1E
```

```
DATA 06
DATA 8C,C8
DATA 8E,C0
DATA 8E,D8
DATA B8,00,00
DATA 89,C7
DATA B9,08,00
DATA B0,41
DATA F3,AA
DATA B8,00,00
DATA 89,C6
DATA B8,08,00
DATA 89,C7
DATA B9,08,00
DATA AC
DATA 04,20
DATA AA
DATA E2,FA
DATA 07
DATA 1F
DATA CB
```

L'exécution :

```
0       0       0       0       0
0       0       0       0       0
0       0       0       0       0

41      41      41      41      41
41      41      41      61      61
61      61      61      61      61
```

donne le résultat attendu, avec huit 41 mais seulement sept 61 visibles.

22.3.4 Comparaison

22.3.4.1 Comparaison de deux octets ou de deux mots

Introduction.- Nous avons souvent besoin de *comparer* deux chaînes de caractères, c'est-à-dire à déterminer si elles sont égales ou si la première vient avant la seconde dans l'ordre lexicographique. Là encore, les concepteurs du 8086 ont implémenté cette opération comme primitive avec l'instruction CMPS pour l'anglais *CoMPare Strings*.

Langage symbolique.- On utilise les deux formes :

```
CMPSB
```

et :

```
CMPSW
```

pour la comparaison.

Sémantique.- Cette instruction soustrait, respectivement, l'octet ou le mot se trouvant à l'emplacement ES :DI à l'octet ou au mot se trouvant à l'emplacement DS :SI et positionne les indicateurs en conséquence. Le contenu de la mémoire n'est pas affecté par cette instruction. Les registres SI et DI sont incrémentés ou décrémentés suivant la valeur de l'indicateur de direction.

Langage machine.- Ces instructions sont codées sur un octet par :

`| 1010 011w |`

avec `w = 0` pour la première et `w = 1` pour la seconde, c'est-à-dire A6h ou A7h.

22.3.4.2 Comparaisons itérées

Les deux instruction sprécédentes peuvent être répétées si elles sont précédées des préfixes REP, REPNZ ou REPZ. Nous avons déjà rencontré la répétition inconditionnelle, voyons maintenant les répétitions conditionnelles.

Répétitions conditionnelles.- Pour déterminer si deux chaînes de caractères sont égales, on va les comparer caractère par caractère. L'instruction de répétition REP n'est pas très adaptée à ce cas car on va décrire les deux chaînes en entier alors qu'on peut s'arrêter dès qu'on trouve deux emplacements qui divergent.

- 1^o) Le préfixe REPNE ou REPNZ (pour *REPetition if No Equal* et *REPetition if No Zero*) permet la répétition de CMPS jusqu'à ce que le contenu de CX soit nul, à moins que les deux termes soient égaux.

- 2^o) Le préfixe REPE ou REPZ (pour *REPetition if Equal* et *REPetition if Zero*) permet la répétition de CMPS jusqu'à ce que le contenu de CX soit nul, à moins que les deux termes soient différents. On a répétition tant que CX est non nul et que l'indicateur ZF est égal à 1.

Langage machine.- Ce préfixe est codé sur un octet, avec le même code que REP :

`| 1111 001z |`

soit F2h pour REPNZ et F3h pour REPE.

Exemple.- Écrivons un programme qui place la chaîne de caractères '`Bon`' à l'emplacement 00h du tableau réservé pour le code machine, la chaîne de caractères '`BOn`' à l'emplacement 03h de ce même tableau puis qui compare les deux chaînes de caractères placées aux décalages 00h et 03h sur trois caractères, plaçant 0 à l'emplacement 06h si ces chaînes sont différentes et 1 sinon.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,CS
MOV ES,AX
MOV DS,AX
MOV BYTE PTR [00],42
MOV BYTE PTR [01],6F
MOV BYTE PTR [02],6E
MOV BYTE PTR [03],42
MOV BYTE PTR [04],4F
MOV BYTE PTR [05],6E
MOV AX, 0
MOV SI,AX
MOV AX, 3
MOV DI,AX
```

```
MOV CX, 3
REPZ CMPSB
JNZ NZ
MOV BYTE PTR [06], 1
JMP FIN
NZ: MOV BYTE PTR [06], 0
FIN: POP ES
POP DS
RETF
```

Remarquons que nous n'avons pas besoin d'utiliser le préfixe `CS :` puisque DS est égal à CS. La traduction de ce sous-programme en langage machine donne :

```
1E
06
8C C8
8E C0
8E D8
C6 06 00 00 42
C6 06 01 00 6F
C6 06 02 00 6E
C6 06 03 00 42
C6 06 04 00 4F
C6 06 05 00 6E
B8 00 00
89 C6
B8 03 00
89 C7
B9 03 00
F3 A6
75 07
C6 06 06 00 01
EB 05
C6 06 06 00 00
07
1F
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 8 octets et le code 70 octets :

```
CLS
DIM PM%(38)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 69
  READ Octet$
```

```
  POKE OffPM% + 8 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 8)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
END
'Code machine
DATA 1E
DATA 06
DATA 8C,C8
DATA 8E,C0
DATA 8E,D8
DATA C6,06,00,00,42
DATA C6,06,01,00,6F
DATA C6,06,02,00,6E
DATA C6,06,03,00,42
DATA C6,06,04,00,4F
DATA C6,06,05,00,6E
DATA B8,00,00
DATA 89,C6
DATA B8,03,00
DATA 89,C7
DATA B9,03,00
DATA F3,A6
DATA 75,07
DATA C6,06,06,00,01
DATA EB,05
DATA C6,06,06,00,00
DATA 07
DATA 1F
DATA CB
```

L'exécution :

```
0     0     0     0     0
0     0     0     1E    6
8C    C8    8E    C0    8E

42    6F    6E    42    4F
6E    0     0     1E    6
8C    C8    8E    C0    8E
```

donne le résultat attendu. Réessayer en changeant 4Fh par 6Fh à l'emplacement 4.

22.3.5 Recherche

Introduction.- On a également souvent à rechercher si une chaîne de caractères contient tel ou tel caractère. Ceci donne lieu à la dernière primitive implémentée sur le 8086 pour les chaînes de caractères, à savoir SCAS pour l'anglais *SCAn a String*.

Langage symbolique.- On utilise les deux formes :

```
SCASB
```

et :

```
SCASW
```

pour la comparaison.

Sémantique.- Cette instruction soustrait, respectivement, l'octet ou le mot se trouvant à l'emplacement ES :DI à l'octet ou au mot se trouvant dans le registre AL ou AX et positionne les indicateurs en conséquence. Le contenu de la mémoire n'est pas affecté par cette instruction. Le registre DI est incrémenté ou décrémenté suivant la valeur de l'indicateur de direction, d'une unité ou deux suivant la taille des termes comparés.

Langage machine.- Ces instructions sont codées sur un octet :

```
| 1010 111w |
```

avec `w = 0` pour la première et `w = 1` pour la seconde.

Exemple.- Écrivons un programme qui place la chaîne de caractères '`Bon`' à l'emplacement 00h du tableau réservé pour le code machine puis qui recherche si le caractère '`o`' apparaît dans celle-ci, plaçant 2 ou 1 à l'emplacement 04h suivant qu'il apparaît ou non.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,CS
MOV ES,AX
MOV DS,AX
MOV BYTE PTR [00],42
MOV BYTE PTR [01],6F
MOV BYTE PTR [02],6E
MOV AX, 0
MOV DI,AX
MOV CX, 3
MOV AL,6F
REPNZ SCASB
JNZ NZ
MOV BYTE PTR [04], 2
JMP FIN
NZ: MOV BYTE PTR [04], 1
FIN: POP ES
POP DS
RETF
```

La traduction de ce sous-programme en langage machine donne :

```
1E
06
8C C8
8E C0
8E D8
C6 06 00 00 42
C6 06 01 00 6F
C6 06 02 00 6E
B8 00 00
89 C7
B9 03 00
B0 6F
F2 AE
75 07
C6 06 04 00 02
EB 05
C6 06 04 00 01
07
1F
CB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 6 octets et le code 52 octets :

```
CLS
DIM PM%(28)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 51
  READ Octet$
  POKE OffPM% + 6 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 6)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
END
```

```
'Code machine
DATA 1E
DATA 06
DATA 8C,C8
DATA 8E,C0
DATA 8E,D8
DATA C6,06,00,00,42
DATA C6,06,01,00,6F
DATA C6,06,02,00,6E
DATA B8,00,00
DATA 89,C7
DATA B9,03,00
DATA B0,6F
DATA F2,AE
DATA 75,07
DATA C6,06,04,00,02
DATA EB,05
DATA C6,06,04,00,01
DATA 07
DATA 1F
DATA CB
```

L'exécution :

```
0       0       0       0       0
0       1E      6       8C      C8
8E      C0      8E      D8      C6

42      6F      6E      0       2
0       1E      6       8C      C8
8E      C0      8E      D8      C6
```

donne le résultat attendu. Réessayer en changeant la recherche de 6Fh par celle de 61Fh.

22.3.6 Choix de la direction

Dans tous les exemples ci-dessus, nous avons choisi la direction croissante. Considérons un exemple pour lequel le choix de la direction ne peut pas être arbitraire.

Exemple.- Considérons le problème suivant : déplacer la chaîne de caractère de l'emplacement `100h-120h` à l'emplacement `102h-122h`. À cause de l'empiètement, on ne peut pas effectuer le déplacement dans l'ordre croissant : les octets `100h` et `101h` pourraient être copiés dans les octets `102h` et `103h` ; mais au moment de déplacer l'octet `102h`, la valeur originelle de celui-ci aurait disparue.

On doit donc nécessairement copier dans l'ordre inverse : d'abord l'octet de l'emplacement `120h` en `122h`.

Exercice.- *Écrire un programme complet correspondant au problème ci-dessus.*

Langage symbolique.- On utilise l'indicateur DF (pour *Direction Flag*) pour indiquer le sens de la copie : s'il vaut 0, on copie de façon naturelle dans l'ordre croissant ; s'il vaut 1, on copie dans l'ordre décroissant.

Il existe deux instructions :

```
CLD
```

et :

```
STD
```

pour spécifier ce sens. L'instruction `CLD` (pour *CLear Direction flag*) met l'indicateur de direction à zéro. L'instruction `STD` (pour *SeT Direction flag*) met l'indicateur de direction à 1.

Langage machine.- Ces instructions sont codées sur un octet respectivement par :

```
| 1111 1100 |
```

soit FCh et par :

```
| 1111 1101 |
```

soit FDh.

22.4 Historique

Le premier ordinateur (EDSAC, 1949 ; voir chapitre 14) n'utilise aucune instruction de manipulation des caractères car il est orienté calcul. Il n'y a pas non plus de registre d'index ; pour effectuer des opérations sur les éléments d'un tableau, il est nécessaire au programme de modifier les adresses de ses propres instructions.

Le registre d'index est inventé en 1950.