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 celle-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

Nous avons vu la notion de *tableau* dans le 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

<u>Notion d'adressage indirect</u>.- 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**.

<u>Syntaxe de l'adressage indirect par registre en langage symbolique</u>.- 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**.

<u>Registres concernés</u>.- 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.

<u>Indication de la taille de l'adressage indirect en langage symbolique</u>.- L'adressage indirect peut se faire pour des octets mais aussi pour des mots (deux 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.

Par exemple :

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

est visiblement une 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 en fait déjà vu, sans nécessairement 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 programme qui commence par placer 'Bonjour' à partir de l'adresse 3000 :402, 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 3000 :400.

Langage symbolique.- Le programme s'écrit :

```
MOV AX,3000
MOV DS,AX
MOV BYTE PTR [402],42
MOV BYTE PTR [403],6F
MOV BYTE PTR [404],6E
MOV BYTE PTR [405],6A
MOV BYTE PTR [406],6F
MOV BYTE PTR [407],75
```

```
       MOV BYTE PTR [408],72
       MOV BYTE PTR [409],0
       MOV DX,0
       MOV BX,402
debut: MOV AL,[BX]
       CMP AL,0
       JE fin
       INC BX
       INC DX
       JMP debut
fin:   MOV [400],DX
       RET
```

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

```
B8 00 30
8E D8
C6 06 02 04 42
C6 06 03 04 6F
C6 06 04 04 6E
C6 06 05 04 6A
C6 06 06 04 6F
C6 06 07 04 75
C6 06 08 04 72
C6 06 09 04 00
BA 00 00
BB 02 04
8A 07
3C 00
74 04
43
42
EB F5
89 16 00 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 C6 06 02 04 42
-E CS:10A C6 06 03 04 6F
-E CS:10F C6 06 04 04 6E
-E CS:114 C6 06 05 04 6A
-E CS:119 C6 06 06 04 6F
-E CS:11E C6 06 07 04 75
-E CS:123 C6 06 08 04 72
-E CS:128 C6 06 09 04 00
-E CS:12D BA 00 00
-E CS:130 BB 02 04
-E CS:133 8A 07
```

```
-E CS:135 3C 00
-E CS:137 74 04
-E CS:139 43
-E CS:13A 42
-E CS:13B EB 75
-E CS:13D 89 16 00 04
-E CS:141 C3
-R BX
BX 0000
:
-R CX
CX 0000
:42
-N COUNT.COM
-W
Ecriture de 00042 octets
-Q
```

Après exécution du programme `count.com`, on vérifie que l'on trouve bien une longueur de 7 à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 07 00 42 6F 6E 6A 6F 75-72 00 00 00 00 00 00 00 ..Bonjour.......
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

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, en langage symbolique :

```
MOVSB
```

et :

```
MOVSW
```

pour copier un octet ou un mot.

Paramètres.- Il semble manquer des informations dans l'instructions ci-dessus. En fait, 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 décrémentés 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 3000 :400 et qui copie cette valeur à l'emplacement 3000 :401.

Le programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,3000
MOV DS,AX
MOV ES,AX
MOV BYTE PTR [400],41
MOV AX,400
MOV SI,AX
MOV AX,401
MOV DI,AX
MOVSB
POP ES
POP DS
RET
```

sans oublier maintenant (ce que nous n'avons pas fait jusqu'à maintenant) 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 programme en langage machine donne :

```
1E
06
B8 00 30
8E D8
8E C0
C6 06 00 00 41
```

```
B8 00 04
89 C6
B8 01 04
89 C7
A4
07
1F
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 1E
-E CS:101 06
-E CS:102 B8 00 30
-E CS:105 8E D8
-E CS:107 8E C0
-E CS:109 C6 06 00 04 41
-E CS:10E B8 00 04
-E CS:111 89 C6
-E CS:113 B8 01 04
-E CS:116 89 C7
-E CS:118 A4
-E CS:119 07
-E CS:11A 1F
-E CS:11B C3
-R BX
BX 0000
:
-R CX
CX 0000
:1C
-N MOVSB.COM
-W
Ecriture de 0001C octets
-Q
```

Après exécution du programme `movsb.com`, on vérifie que la copie a bien eu lieu :

```
C:\COM>debug
-D 3000:400
3000:0400 41 41 42 6F 6E 6A 6F 75-72 00 00 00 00 00 00 00 AABonjour.......
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

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 contenu dans le 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 pour obtenir :

```
REP MOVSB
```

et :

```
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'emplacement mémoire 3000 :400, y compris son terminateur 0, puis de copier le contenu de cette chaîne de caractères à l'adresse 3000 :408.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,3000
MOV DS,AX
MOV ES,AX
MOV BYTE PTR [400],42
MOV BYTE PTR [401],6F
MOV BYTE PTR [402],6E
MOV BYTE PTR [403],6A
MOV BYTE PTR [404],6F
MOV BYTE PTR [405],75
MOV BYTE PTR [406],72
MOV BYTE PTR [407],0
MOV AX,400
MOV SI,AX
MOV AX,408
MOV DI,AX
MOV CX,8
REP MOVSB
POP ES
POP DS
RET
```

La traduction de ce programme en langage machine donne :

```
1E
06
B8 00 30
```

```
8E D8
8E C0
C6 06 00 04 42
C6 06 01 04 6F
C6 06 02 04 6E
C6 06 03 04 6A
C6 06 04 04 6F
C6 06 05 04 75
C6 06 06 04 72
C6 06 07 04 00
B8 00 04
89 C6
B8 08 04
89 C7
F3 A4
07
1F
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 1E
-E CS:101 06
-E CS:102 B8 00 30
-E CS:105 8E D8
-E CS:107 8E C0
-E CS:109 C6 06 00 04 42
-E CS:10E C6 06 01 04 6F
-E CS:113 C6 06 02 04 6E
-E CS:118 C6 06 03 04 6A
-E CS:11D C6 06 04 04 6F
-E CS:122 C6 06 05 04 75
-E CS:127 C6 06 06 04 72
-E CS:12C C6 06 07 04 00
-E CS:131 B8 00 04
-E CS:134 89 C6
-E CS:136 B8 08 04
-E CS:139 8C C7
-E CS:13B B9 08 00
-E CS:13E F3
-E CS:13F A4
-E CS:140 07
-E CS:141 1F
-E CS:142 C3
-R BX
BX 0000
:
-R CX
CX 0000
:43
-N MOVSB.COM
-W
```

```
Ecriture de 00043 octets
-Q
```

Après exécution du programme `movsb.com`, on vérifie que l'on trouve bien la copie de la chaîne de caractères à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 42 6F 6E 6A 6F 75 72 00-42 6F 6E 6A 6F 75 72 00 Bonjour.Bonjour.
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

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.- En langage symbolique, les deux instructions :

```
STOSB
```

et :

```
STOSW
```

(pour *STOre a String*) permettent de stocker l'octet contenu dans le registre AL, respectivement 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 AAh et ABh 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 3000 :400.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,3000
MOV DS,AX
MOV ES,AX
MOV AX,400
MOV DI,AX
MOV AL, 41
STOSB
```

```
POP ES
POP DS
RET
```

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

```
1F
06
B8 00 30
8E D8
8E C0
B8 00 04
89 C7
B0 41
AA
07
1F
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 1E
-E CS:101 06
-E CS:102 B8 00 30
-E CS:105 8E D8
-E CS:107 8E C0
-E CS:109 B8 00 04
-E CS:10C 89 C7
-E CS:10E B0 41
-E CS:110 AA
-E CS:111 07
-E CS:112 1F
-E CS:113 C3
-R BX
BX 0000
:
-R CX
CX 0000
:14
-N STOSB.COM
-W
Ecriture de 00014 octets
-Q
```

Après exécution du programme `stosb.com`, on vérifie que l'on a bien placé 'A' à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 41 6F 6E 6A 6F 75 72 00-42 6F 6E 6A 6F 75 72 00 Aonjour.Bonjour.
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
```

```
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

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 (de longueur double dans le cas d'un mot) 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 3000 :400.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,3000
MOV DS,AX
MOV ES,AX
MOV AX,400
MOV DI,AX
MOV AL,41
MOV CX,08
REP STOSB
POP ES
POP DS
RET
```

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

```
1E
06
B8 00 30
8E D8
8E C0
B8 00 04
89 C7
B0 41
B9 08 00
F3
AA
07
1F
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 1E
-E CS:101 06
-E CS:102 B8 00 30
-E CS:105 8E D8
-E CS:107 8E C0
-E CS:109 B8 00 04
-E CS:10C 89 C7
-E CS:10E B0 41
-E CS:110 B9 08 00
-E CS:113 F3
-E CS:114 AA
-E CS:115 07
-E CS:116 1F
-E CS:117 C3
-R BX
BX 0000
:
-R CX
CX 0000
:18
-N RSTOSB.COM
-W
Ecriture de 00018 octets
-Q
```

Après exécution du programme `rstosb.com`, on vérifie que l'on a bien placé 'A' aux emplacements mémoire choisis :

```
C:\COM>debug
-D 3000:400
3000:0400 41 41 41 41 41 41 41 41-42 6F 6E 6A 6F 75 72 00 AAAAAAAABonjour.
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

22.3.3 Chargement

Nous avons vu comment copier le contenu de l'accumulateur vers un emplacement mémoire, et même une zone 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, respectivement AX.

<u>Langage machine</u>.- 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 $w = 0$, de 2 pour $w = 1$, 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.

<u>Exemple</u>.- Écrivons un programme machine qui place 'A' à l'emplacement mémoire 3000 :400 puis récupère ce caractère, l'incrémente (donc le transforme en 'B') et le place en 3000 :401.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,3000
MOV DS,AX
MOV ES,DX
MOV BYTE PTR [400],41
MOV AX,400
MOV SI,AX
MOV AX,401
MOV DI,AX
LODSB
INC AL
STOSB
POP ES
POP DS
RET
```

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

```
1E
06
B8 00 30
8E D8
8E C0
C6 06 00 04 41
B8 00 04
89 C6
B8 01 04
89 C7
AC
FE C0
```

```
AA
07
1F
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 1E
-E CS:101 06
-E CS:102 B8 00 30
-E CS:105 8E D8
-E CS:107 8E C0
-E CS:109 C6 06 00 04 41
-E CS:10E B8 00 04
-E CS:111 89 C6
-E CS:113 B8 01 04
-E CS:116 89 C7
-E CS:118 AC
-E CS:119 FE C0
-E CS:11B AA
-E CS:11C 07
-E CS:11D 1F
-E CS:11E C3
-R BX
BX 0000
:
-R CX
CX 0000
:1F
-N LODSB.COM
-W
Ecriture de 0001F octets
-Q
```

Après exécution du programme `lodsb.com`, on vérifie que l'on a bien placé 'A' et 'B' aux emplacements mémoire choisis :

```
C:\COM>debug
-D 3000:400
3000:0400 41 42 00 00 00 00 00 00-00 00 00 00 00 00 00 00 AB..............
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

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.

Langage machine.- 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.

Exemple.- 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 3000 :400 de longueur 8 puis transférons-la à l'adresse de décalage 408 en transformant les majuscules en minuscules.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,3000
MOV DS,AX
MOV ES,AX
```

```
       MOV AX,400
       MOV DI,AX
       MOV CX,8
       MOV AL,41
       REP STOSB
       MOV AX,400
       MOV SI,AX
       MOV AX,408
       MOV DI,AX
       MOV CX,8
DEBUT: LODSB
       ADD AL,20
       STOSB
       LOOP DEBUT
       POP ES
       POP DS
       RET
```

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

```
1E
06
B8 00 30
8E D8
8E C0
B8 00 04
89 C7
B8 00 04
89 C7
B9 08 00
B0 41
F3
AA
B8 00 04
89 C6
B8 08 04
89 C7
B9 08 00
AC
04 20
AA
E2 FA
07
1F
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 1E
-E CS:101 06
```

```
-E CS:102 B8 00 30
-E CS:105 8E D8
-E CS:107 8E C0
-E CS:109 B8 00 04
-E CS:10C 89 C7
-E CS:10E B9 08 00
-E CS:111 B0 41
-E CS:113 F3
-E CS:114 AA
-E CS:115 B8 00 04
-E CS:118 89 C6
-E CS:11A B8 08 04
-E CS:11D 89 C7
-E CS:11F B9 08 00
-E CS:122 AC
-E CS:123 04 20
-E CS:125 AA
-E CS:126 E2 FA
-E CS:128 07
-E CS:129 1F
-E CS:12A C3
-R BX
BX 0000
:
-R CX
CX 0000
:2B
-N LOOP.COM
-W
Ecriture de 0002B octets
-Q
```

Après exécution du programme `loop.com`, on vérifie que l'on a bien placé huit 'A' et reporté huit 'a' aux emplacements mémoire choisis :

```
C:\COM>debug
-D 3000:400
3000:0400 41 41 41 41 41 41 41 41-61 61 61 61 61 61 61 61 AAAAAAAAaaaaaaaa
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

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 mémoire 3000 :400, la chaîne de caractères '`BOn`' à l'emplacement mémoire 3000 :403 puis qui compare les deux chaînes de caractères placées aux décalages 400h et 403h sur trois caractères, plaçant 0 à l'emplacement 406h si ces chaînes sont différentes et 1 sinon.

Le sous-programme en langage symbolique s'écrit :

```
PUSH DS
PUSH ES
MOV AX,3000
MOV DS,AX
MOV ES,AX
MOV BYTE PTR [400],42
```

```
     MOV BYTE PTR [401],6F
     MOV BYTE PTR [402],6E
     MOV BYTE PTR [403],42
     MOV BYTE PTR [404],4F
     MOV BYTE PTR [405],6E
     MOV AX,400
     MOV SI,AX
     MOV AX,403
     MOV DI,AX
     MOV CX,3
     REPZ CMPSB
     JNZ NZ
     MOV BYTE PTR [406],1
     JMP FIN
NZ:  MOV BYTE PTR [406],0
FIN: POP ES
     POP DS
     RET
```

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

```
1E
06
B8 00 30
8E D8
8E C0
C6 06 00 04 42
C6 06 01 04 6F
C6 06 02 04 6E
C6 06 03 04 42
C6 06 04 04 4F
C6 06 05 04 6E
B8 00 04
89 C6
B8 03 04
89 C7
B9 03 00
F3
A6
75 07
C6 06 06 04 01
EB 05
C6 06 06 04 00
07
1F
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:0100 1E
```

```
-E CS:0101 06
-E CS:0102 B8 00 30
-E CS:0105 8E D8
-E CS:0107 8E C0
-E CS:0109 C6 06 00 04 42
-E CS:010E C6 06 01 04 6F
-E CS:0113 C6 06 02 04 6E
-E CS:0118 C6 06 03 04 42
-E CS:011D C6 06 04 04 4F
-E CS:0122 C6 06 05 04 6E
-E CS:0127 B8 00 04
-E CS:012A 89 C6
-E CS:012C B8 03 04
-E CS:012F 89 C7
-E CS:0131 B9 03 00
-E CS:0134 F3
-E CS:0135 A6
-E CS:0136 75 07
-E CS:0138 C6 06 06 04 01
-E CS:013D EB 05
-E CS:013F C6 06 06 04 00
-E CS:0144 07
-E CS:0145 1F
-E CS:0146 C3
-R BX
BX 0000
:
-R CX
CX 0000
:47
-N CMPSB.COM
-W
Ecriture de 00047 octets
-Q
```

Après exécution du programme `cmpsb.com`, on vérifie que l'on a bien placé 0 à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 42 6F 6E 42 4F 6E 00 41-61 61 61 61 61 61 61 61 BonBOn.Aaaaaaaaa
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

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 mémoire 3000 :400 puis qui recherche si le caractère '`o`' apparaît dans celle-ci, plaçant 2 ou 1 à l'emplacement 3000 :404 suivant qu'il apparaît ou non.

Le sous-programme en langage symbolique s'écrit :

```
      PUSH DS
      PUSH ES
      MOV AX,CS
      MOV DS,AX
      MOV ES,AX
      MOV BYTE PTR [400],42
      MOV BYTE PTR [401],6F
      MOV BYTE PTR [402],6E
      MOV AX,400
      MOV DI,AX
      MOV CX,3
      MOV AL,6F
      REPNZ SCASB
      JNZ NZ
      MOV BYTE PTR [404],2
      JMP FIN
NZ:   MOV BYTE PTR [404],1
FIN:  POP ES
      POP DS
      RET
```

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

```
1E
06
B8 00 30
8E D8
8E C0
```

```
C6 06 00 04 42
C6 06 01 04 6F
C6 06 02 04 6E
B8 00 04
89 C7
B0 6F
F2
AE
75 07
C6 06 04 04 02
EB 05
C6 06 04 04 01
07
1F
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:0100 1E
-E CS:0101 06
-E CS:0102 B8 00 30
-E CS:0105 8E D8
-E CS:0107 8E C0
-E CS:0109 C6 06 00 04 42
-E CS:010E C6 06 01 04 6F
-E CS:0113 C6 06 02 04 6E
-E CS:0118 B8 00 04
-E CS:011B 89 C7
-E CS:011D B9 03 00
-E CS:0120 B0 6F
-E CS:0122 F2
-E CS:0123 AE
-E CS:0124 75 07
-E CS:0126 C6 06 04 04 02
-E CS:012B EB 05
-E CS:012D C6 06 04 04 01
-E CS:0132 07
-E CS:0133 1F
-E CS:0134 C3
-R BX
BX 0000
:
-R CX
CX 0000
:35
-N SCASB.COM
-W
Ecriture de 00035 octets
-Q
```

Après exécution du programme `scasb.com`, on vérifie que l'on a bien placé 2 à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 42 6F 6E 42 02 6E 00 41-61 61 61 61 61 61 61 61 BonB.n.Aaaaaaaaa
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
-q
```

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 `400h-420h` à l'emplacement `402h-422h` du segment d'adresse 3000h. À cause de l'empiètement, on ne peut pas effectuer le déplacement dans l'ordre croissant : les octets `400h` et `401h` pourraient être copiés dans les octets `402h` et `403h` ; mais, au moment de déplacer l'octet `402h`, la valeur originelle de celui-ci aurait disparue.

On doit donc nécessairement copier dans l'ordre inverse : d'abord l'octet de l'emplacement `420h` en `422h`.

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 ? ?) 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.