# Chapitre 19

# La pile

Nous avons rencontré un problème à propos des données intermédiaires. Lorsqu'elles sont en petit nombre, on peut les placer dans des registres. Lorsqu'elles sont en nombre plus important, on les place dans des cases mémoire, mais ceci exige une gestion minutieuse de la mémoire. La *pile* permet d'entreposer plus facilement les données temporaires.

## 19.1 La pile

### 19.1.1 Étude générale de la pile

Notion de pile.- Lorsqu'on effectue un calcul, on a souvent des résultats intermédiaires. Lorsqu'on effectue les calculs à la main, on place ces résultats intermédiaires sur un coin de la feuille. Mais comment faire lorsqu'il s'agit d'un ordinateur ?

Lorsque ces résultats intermédiaires ne sont pas en nombre très important, on peut utiliser des registres s'il y en a de libre. Mais on arrive très vite à la saturation du nombre très limité de registres.

On peut alors, comme nous l'avons vu, placer ces résultats intermédiaires en mémoire vive. Reste alors à gérer la mémoire vive pour que ces résultats ne viennent pas détruire d'autres données ou le programme. Jusqu'à maintenant nous avons effectué cette gestion à la main.

Une façon de gérer la mémoire vive consiste à réserver une certaine zone de celle-ci, appelée la **pile** (*stack* en anglais), qui sert spécifiquement à stocker les données temporaires. Son accès suit des règles spécifiques qui expliquent sa dénomination 'pile'.

Intérêt et inconvénient de la pile.- La pile sert surtout, comme nous venons de le dire, à stocker des données temporaires et, plus particulièrement, les valeurs des registres lorsqu'on est à court de registres. Une autre façon de faire serait d'augmenter le nombre de registres mais reste à savoir combien et, de toute façon, cela revient très cher. Bien entendu le principal inconvénient concernant l'utilisation de la pile est qu'il s'agit de mémoire vive et donc que les temps d'accès sont importants.

Structure de la pile.- La pile est une zone connexe de la mémoire vive. Son emplacement est donc entièrement déterminé par l'adresse de début et l'adresse de fin de la pile. On pourrait prévoir un index se déplaçant entre ces deux limites pour désigner l'élément qui nous intéresse.

En fait, comme son nom de pile l'indique, on accède toujours à la pile par le haut. On place un élément au sommet de la pile (on **empile**, *to push* en anglais) et on récupère l'élément du sommet de la pile (on **dépile**, *to pop* en anglais).

L'adresse importante, à un instant donné, est donc celle du **sommet de la pile** (*top* en anglais).

Instructions d'accès à la pile.- Il y a donc deux instructions pour accéder à la pile : placer un nouvel élément (au sommet de la pile), c'est-à-dire *empiler*, et récupérer l'élément du sommet de la pile, c'est-à-dire *dépiler*.

Remarque sur la pile pleine.- Il devrait y avoir une troisième instruction pour savoir si la pile est pleine : en effet si on empile un élément alors qu'elle est pleine, on risque de détruire des données essentielles par ailleurs. Cependant cette instruction est rarement implémentée, considérant que cela a peu de chances d'arriver.

### 19.1.2 Cas du microprocesseur 8086/8088

#### 19.1.2.1 Philosophie de la mise en place

Emplacement de la pile.- Pour éviter que le programme et la pile n'interfèrent pas, le microprocesseur 8080 a été conçu en suggérant que le code et les données soient placés avec des adresses de numéro le plus bas possible dans la mémoire vive alors que la pile doit avoir des adresses de numéro le plus haut possible. Bien entendu, l'utilisateur (ou le concepteur du système d'exploitation) peut toujours faire ce qu'il veut, mais il a intérêt à suivre cette suggestion.

De ce fait la pile est inversée : le premier élément est placé à la dernière position de la mémoire, l'élément suivant avant et ainsi de suite.

Puisque ce sont essentiellement des adresses qui sont placées sur la pile (celles de retour d'un sous-programme), chaque élément de la pile occupe deux octets. Il n'y a pas de possibilité d'y placer un octet isolé.

Cela a pour conséquence que, pour le 8080, l'adresse du sommet de la pile décroît automatiquement (de deux unités) lorsqu'on ajoute un élément à la pile et qu'elle croît automatiquement (de deux unités également) lorsqu'on lui enlève un élément.

Pour le 8086/8088, puisqu'il est prévu que le code, les données et la pile soient placés dans des segments différents, on aurait pu ne pas conserver ces conventions. Cependant, pour des raisons de compatibilité, elles ont été conservées.

Éléments de la pile.- Comme nous l'avons déjà dit, tous les éléments de la pile sont des mots (de deux octets).

Adresse du sommet de la pile.- Cette adresse est contenue dans le **registre de pile** `SP` (pour l'anglais *Stack Pointer*). L'élément mémoire d'adresse le contenu de `SP` contient l'octet de poids faible du sommet de la pile et l'élément mémoire d'adresse le contenu de `SP` plus un contient l'octet de poids fort du sommet de la pile.

#### 19.1.2.2 Empilement

Langage symbolique.- Pour placer le contenu d'un registre (nécessairement de 16 bits) ou d'une case mémoire (également nécessairement de 16 bits) au sommet de la pile, on utilise en langage symbolique l'instruction :

```
PUSH source
```

qui place le contenu à l'adresse indiquée par `SP` et décrémente `SP` de deux unités.

Langage machine.- On a plusieurs instructions suivant le type de source :

– La sauvegarde d'un registre de données est codée sur un octet :

```
0101 0 reg
```

– La sauvegarde d'un registre de segment est également codée sur un octet :

```
000 reg 110
```

– La sauvegarde du contenu d'une case mémoire est codée sur deux ou quatre octets :

```
| 1111 1111 | mod 110 r/m |        |        |
```

Exercice corrigé.- *Traduire l'instruction* :

```
PUSH AX
```

*en langage machine.*

D'après ce que nous venons de dire, cette instruction se traduit par l'octet :

```
0101 0 000
```

en binaire, soit 50h.

Sauvegarde du registre des indicateurs.- Le registre des indicateurs F peut également être sauvegardé sur la pile grâce à l'instruction en langage symbolique :

```
PUSHF
```

codée sur un octet par :

```
1001 1100
```

en langage machine, soit 9Ch.

#### 19.1.2.3 Dépilement

Langage symbolique.- Pour récupérer le mot (de 16 bits) contenu au sommet de la pile et le placer dans un registre (nécessairement de 16 bits) ou une case mémoire, on utilise l'instruction en langage symbolique :

```
POP destination
```

qui effectue l'opération indiquée et incrémente `SP` de deux unités.

Langage machine.- On a plusieurs instructions suivant le type de source :

– La récupération et le placement dans un registre de données est codée sur un octet :

```
0101 1 reg
```

– La récupération et le placement dans un registre de segment est également codée sur un octet :

```
000 reg 111
```

mais POP CS est interdit.

– La récupération et le placement dans une case mémoire est codée sur deux ou quatre octets :

```
| 1000 1111 | mod 000 r/m |        |        |
```

Exercice corrigé.- *Traduire l'instruction* :

```
POP AX
```

*en langage machine.*

D'après ce que nous venons de dire, cette instruction se traduit par l'octet :

```
0101 1 000
```

en binaire, soit 58h.

Récupération dans le registre des indicateurs.- Le registre des indicateurs F peut également recevoir le contenu du sommet de la pile grâce à l'instruction en langage symbolique :

```
POPF
```

codée sur un octet par :

```
1001 1101
```

en langage machine, soit 9Dh.

Remarque.- Si on veut conserver deux registres et les récupérer plus tard, il faut bien faire attention à les dépiler dans l'ordre inverse de leur empilement :

```
PUSH AX
PUSH BX
[...]
POP BX
POP AX
```

Remarque.- (**Initialisation du registre des indicateurs**)

Nous avons dit que nous ne pouvions pas initialiser le registre des indicateurs, contrairement aux autres registres. En fait on peut le faire, de façon détournée. Il suffit de placer la valeur que l'on veut dans un registre, d'empiler ce registre et de dépiler cette valeur dans le registre des indicateurs.

Exemple.- Le sous-programme suivant permet d'échanger les contenus des registres AX et BX en utilisant la pile :

```
PUSH AX
PUSH BX
POP AX
POP BX
RET
```

### 19.1.3 Un exemple

Écrivons un programme en langage machine qui échange le contenu des emplacements mémoire, de capacité un mot, 3000 :400 et 300 :402. L'échange des contenus sera facilité en utilisant la pile.

Le programme en langage symbolique s'écrit :

```
MOV  AX,3000
MOV  DS,AX
PUSH [400]
PUSH [402]
POP  [400]
POP  [402]
RET
```

Traduisons ce programme en langage machine :

```
B8 00 30
8E D8
FF 36 00 04
FF 36 02 04
8F 06 00 04
8F 06 02 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 FF 36 00 04
-E CS:109 FF 36 02 04
-E CS:10D 8F 06 00 04
-E CS:111 8F 06 02 04
-E CS:115 C3
-R BX
BX 0000
:
-R CX
CX 0000
:016
-N PILE.COM
-W
Ecriture de 00016 octets
-Q
```

Si on place 3 et 2 comme données :

```
C:\COM>debug
-E 3000:400 03 00 02 00
-q

C:\COM> PILE.COM
```

Après exécution du programme `pile.com`, on vérifie que l'on a bien 2 et 3 aux emplacements mémoire choisis :

```
C:\COM>debug
-D 3000:400
3000:0400 02 00 03 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
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
```

## 19.2 Le passage des paramètres à un sous-programme

Pour passer un paramètre à un sous-programme écrit en langage machine, on utilise tout d'abord les registres. Lorsque ceux-ci ne sont pas en nombre suffisant, on place ces paramètres dans un coin de la mémoire, qui sera partagé par le programme principal et le sous-programme. La gestion de ce coin de mémoire pouvant être délicat, on utilise le plus souvent la pile.

Intéressons-nous à un même problème pour tester ces trois façons de faire : on met un entier à l'emplacement mémoire 3000 :400 ; on fait ensuite appel à un programme machine qui récupère cette valeur et la passe en paramètre à un sous-programme dont le rôle est de mettre la valeur passée à l'emplacement mémoire 3000 :400.

### 19.2.1 Utilisation des registres de données

La valeur récupérée par le programme machine est placée dans le registre AX. On fait appel au sous-programme machine qui utilise le registre AX pour manipuler le paramètre.

Le programme en langage symbolique est :

```
      MOV AX,3000
      MOV DS,AX
      MOV AX,[400]
      CALL SOUS
      RET
SOUS: MOV [402],AX
      RET
```

On peut évidemment se demander l'intérêt d'utiliser un sous-programme pour un tel programme. Là n'est pas la question : rappelons qu'il s'agit de tester le passage des paramètres par registres.

Traduisons ce programme en langage machine :

```
B8 00 30
8E D8
A1 00 04
E8 01 00
C3
A3 02 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 A1 00 04
-E CS:108 E8 01 00
-E CS:10B C3
-E CS:10C A3 02 04
-E CS:10F C3
-R BX
BX 0000
:
-R CX
CX 0000
:010
-N PASREG.COM
-W
Ecriture de 00010 octets
-Q
```

Si on place 2 comme donnée :

```
C:\COM>debug
-E 3000:400 02 00
-q

C:\COM> PASREG.COM
```

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

```
C:\COM>debug
-D 3000:400
3000:0400 02 00 02 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
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
```

### 19.2.2 Utilisation de la mémoire

Nous l'avons déjà utilisé plusieurs fois.

### 19.2.3 Utilisation de la pile

Principe.- Rappelons que chaque programme (en langage machine) du 80886/8088 possède quatre segments : un segment de code, un segment des données, un segment de pile et un segment supplémentaire. Lorsqu'on fait appel à un sous-programme en langage machine, le segment de code (en faisant un appel long) peut changer mais pas le segment de pile, de façon à partager un espace mémoire commun, permettant entre autres la transmission des paramètres.

Avant de faire appel au sous-programme, on place les paramètres sur la pile. En faisant appel au sous-programme, le microprocesseur ajoute un ou deux éléments sur la pile, de façon à pouvoir passer à l'instruction suivante du programme principal lors du retour du sous-programme : le contenu du pointeur d'instruction IP et, dans le cas d'un appel intersegmentaire, le contenu du segment de code CS.

Récupération des paramètres par le sous-programme.- Dans le programme principal, on place un paramètre sur la pile par une instruction PUSH. Dans le sous-programme, on ne peut pas se contenter d'effectuer un POP, puisque l'appel au sous-programme a placé une ou deux valeurs (qu'il ne faut pas perdre) sur la pile.

Une astuce serait de récupérer les paramètres par une instruction :

```
MOV recup, [sp + 4]
```

en calculant bien l'emplacement, c'est-à-dire le déplacement à utiliser.

Si on regarde les instructions MOV, ceci n'est pas permis.

On va donc utiliser :

```
MOV BP, SP
MOV recup, [BP + 4]
```

d'où l'intérêt du registre BP dont nous ne nous sommes pas servis jusqu'à maintenant.

Le problème du dépilement.- Dans cette façon de faire, on n'a pas retiré de la pile les paramètres qui y ont été placés. En général les paramètres passés au sous-programme n'intéresseront plus le programme principal, qui peut, par contre, utiliser la pile pour autre chose. Il faut donc mettre à jour le pointeur de pile. Il existe deux nouvelles instructions en langage machine qui permettent d'effectuer ceci de façon automatique.

Langage symbolique.- 1$^o$) Au lieu de terminer un sous-programme court (c'est-à-dire intra-segmentaire) par un :

```
RET
```

on le termine par :

```
RET constante
```

où `constante` est le nombre d'octets utilisés pour les paramètres. Dans ces conditions, `RET` change la valeur de IP mais également celle de SP, en lui ajoutant la `constante`.

- 2$^o$) Au lieu de terminer un sous-programme long (c'est-à-dire inter-segmentaire) par un :

```
RETF
```

on le termine par :

```
RETF constante
```

où `constante` est le nombre d'octets utilisés pour les paramètres. Dans ces conditions, `RET` change les valeurs de IP et de CS mais également celle de SP, en lui ajoutant la `constante`.

Langage machine.- 1$^o$) La première instruction est codée sur trois octets par :

```
| 1100 0010 | constante basse | constante haute |
```

soit C2h suivi de la valeur de la constante sur seize bits.

- 2$^o$) La seconde instruction est codée sur trois octets par :

```
| 1100 1010 | constante basse | constante haute |
```

soit CAh suivi de la valeur de la constante sur seize bits.

Exemple.- Reprenons notre exemple que nous allons traiter en utilisant la pile. La valeur récupérée par le programme machine est placée sur la pile. On fait appel au sous-programme machine qui récupère cette valeur sur la pile, non pas au sommet, comme nous l'avons fait remarquer, mais à l'emplacement adéquat, et la place comme deuxième élément du tableau.

Le programme en langage symbolique est :

```
      MOV AX,3000
      MOV DS,AX
      MOV AX,[400]
      PUSH AX
      CALL SOUS
      RET
SOUS: PUSH BP
      MOV BP,SP
      MOV BX,[BP + 4]
      MOV [402],BX
      POP BP
      RET 2
```

avec BP + 4 puisque la pile contient, en partant du sommet, l'ancienne valeur de BP (au déplacement 0), la valeur de IP de retour (au déplacement 2) et enfin le paramètre (au déplacement 4). Lors du retour, on demande d'incrémenter SP de 2 (taille du paramètre).

Traduisons ce programme en langage machine :

```
B8 00 30
8E D8
A1 00 04
50
E8 01 00
C3
55
89 E5
8B 5E 04
89 1E 02 04
5D
C2 02 00
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 A1 00 04
-E CS:108 50
-E CS:109 E8 01 00
-E CS:10C C3
-E CS:10D 55
-E CS:10E 89 E5
-E CS:110 8B 5E 04
-E CS:113 89 1E 02 04
```

```
-E CS:117 5D
-E CS:118 C2 02 00
-R BX
BX 0000
:
-R CX
CX 0000
:01B
-N PASPIL.COM
-W
Ecriture de 0001B octets
-Q
```

Si on place 2 comme donnée :

```
C:\COM>debug
-E 3000:400 02 00 00 00
-q

C:\COM> PASPIL.COM
```

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

```
C:\COM>debug
-D 3000:400
3000:0400 02 00 02 00 00 00 00 00-00 00 00 00 00 00 00 00 ................
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
```

Commentaire.- Lorsqu'on surcharge un registre dans un sous-programme, on risque de détruire la valeur qui est peut-être utilisée par le programme principal. On place donc les valeurs des registres surchargés dans le sous-programme sur la pile en début de sous-programme et on les récupère en fin de sous-programme.

Dans notre cas, nous pouvons prévoir qu'il n'y a pas de conséquence. Pour donner un exemple, nous avons placé BP sur la pile en début de sous-programme et nous avons récupéré l'ancienne valeur en fin de sous-programme. En toute rigueur, nous aurions dû également le faire pour le registre AX.