Chapitre 19

La pile

Nous avons rencontré un problème à propos des sous-programmes : comment passer les paramètres ? Lorsqu'ils sont en petit nombre on peut utiliser les registres. Lorsqu'ils 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 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 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 être sûr que le programme et la pile n'interfèrent pas, le microprocesseur 8080 a été conçu en suggérant que le code et les données devraient être placés avec des adresses de numéro le plus bas possible dans la mémoire vive alors que la pile devrait 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.

On considère que ce sont des adresses qui seront placées sur la pile. Chaque élément de la pile occupe donc deux octets.

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 (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 au sommet de la pile, on utilise l'instruction en langage symbolique :

```
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 &H50 en hexadécimal.

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 &H9C.

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 &H58 en hexadécimal.

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 &H9D.

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 QBasic qui saisit deux entiers naturels (courts), les place dans les variables, échange le contenu des variables puis affiche le contenu de celles-ci pour vérifier que l'échange a bien eu lieu. L'écahnge des contenus des variables se fera en utilisant un sous-programme machine utilisant la pile.

Plaçons les contenus des variables en 0 et en 2 comme d'habitude. Nous avons vu ci-dessus l'idée du sous-programme machine :

```
PUSH CS:[00]
PUSH CS:[02]
POP CS:[02]
POP CS:[00]
RETF
```

Traduisons ce programme écrit en langage symbolique en langage machine :

```
&H2E &HFF &H36 &H00 &H00
&H2E &HFF &H36 &H02 &H00
&H2E &H8F &H06 &H00 &H00
&H2E &H8F &H06 &H02 &H00
&H2E &H01 &H06 &H04 &H00
&HCB
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 4 octets et le code 21 octets :

```
CLS
DIM PM%(12)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Saisie des donnees
INPUT "A = ", A%
PM%(0) = A%
INPUT "B = ", B%
PM%(1) = B%
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 20
  READ Octet$
  POKE OffPM% + 4 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT PEEK(I%),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 4)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
```

```
  PRINT PEEK(I%),
NEXT I%
'Retour au segment de depart
DEF SEG
'Affichage du resultat
PRINT "A = "; PM%(0)
PRINT "B = "; PM%(1)
END
'Code machine
DATA 2E,FF,36,00,00
DATA 2E,FF,36,02,00
DATA 2E,8F,06,00,00
DATA 2E,8F,06,02,00
DATA CB
```

L'exécution se comporte comme prévu :

```
A = 2
B = 3
2     0    3     0    2E
FF    36   0     0    2E
FF    36   2     0    2E

3     0    2     0    2E
FF    36   0     0    2E
FF    36   2     0    2E

A = 3
B = 2
```

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 peut être délicat, aussi utilise-t-on la pile.

Intéressons-nous à un même problème pour tester ces trois façons de faire : un programme QBasic demande un entier N et le place au début de la zone mémoire réservée pour le tableau ; il 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 placer la valeur passée comme deuxième élément du tableau.

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 machine est :

```
MOV AX, CS:[00]
CALL sous
```

```
RETF
sous : MOV CS:[02], AX
RET
```

pour lequel on peut se demander l'intérêt d'utiliser un sous-programme. Rappelons qu'il s'agit de tester le passage des paramètres par registres.

Traduisons ce programme écrit en langage symbolique en langage machine :

```
&H2E &HA1 &H00 &H00
&HE8 &H01 &H00
&HCB
&H2E &HA3 &H02 &H00
&HC3
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 4 octets et le code 13 octets :

```
CLS
DIM PM%(8)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Saisie de la donnee
INPUT "N = ", N%
PM%(0) = N%
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 12
  READ Octet$
  POKE OffPM% + 4 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 14
  PRINT PEEK(I%),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 4)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT PEEK(I%),
NEXT I%
'Retour au segment de depart
DEF SEG
'Affichage du resultat
PRINT PM%(1)
END
'Code machine
DATA 2E,A1,00,00
DATA E8,01,00
DATA CB
DATA 2E,A3,02,00
DATA C3
```

L'exécution se comporte comme prévu :

```
N = 5
5     0    0     0    2E
A1    0    0     E8   1
0     CB   2E    A3   2

5     0    0     0    2E
A1    0    0     E8   1
0     CB   2E    A3   2

 5
```

19.2.2 Utilisation de la mémoire

Nous l'avons déjà utilisé plusieurs fois, y compris dans le programme précédent pour passer un paramètre du programme QBasic au programme machine.

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 POP. 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 récupèrer 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]
```

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 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 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 &HC2 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 &HCA 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 machine est :

```
MOV AX, CS:[00]
PUSH AX
CALL sous
RETF
sous: PUSH BP
MOV BP, SP
MOV BX, [BP + 4]
MOV CS:[02], 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 écrit en langage symbolique en langage machine :

```
&H2E &HA1 &H00 &H00
&H50
&HE8 &H01 &H00
&HCB
&H55
```

```
&H89 &HE5
&H8B &H5E &H04
&H2E &H89 &H1E &H02 &H00
&H5D
&HC2 02 00
```

Utilisons alors le programme QBasic suivant, sachant que les données occupent 4 octets et le code 24 octets :

```
CLS
DIM PM%(13)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Saisie de la donnee
INPUT "N = ", N%
PM%(0) = N%
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 12
  READ Octet$
  POKE OffPM% + 4 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 23
  PRINT PEEK(I%),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 4)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT PEEK(I%),
NEXT I%
'Retour au segment de depart
DEF SEG
'Affichage du resultat
PRINT PM%(1)
END
'Code machine
DATA 2E,A1,00,00
DATA 50
DATA E8,01,00
DATA CB
DATA 55
DATA 89,E5
DATA 8B,5E,04
DATA 2E,89,1E,02,00
DATA 5D
DATA C2,02,00
```

L'exécution se comporte comme prévu :

```
N = 5
5     0    0     0    2E
A1    0    0     50   E8
1     0    CB    55   89

5     0    5     0    2E
A1    0    0     50   E8
1     0    CB    55   89

 5
```

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.

19.3 Cas du QBasic

D'après ce que l'on a dit, on comprend que l'instruction :

```
CALL ABSOLUTE(Offset)
```

est un appel à un sous-programme court et que les deux instructions :

```
DEF SEG Segment
CALL ABSOLUTE(Offset)
```

donnent lieu à un sous-programme long.

Pouvons-nous passer des paramètres à un tel programme machine ?

Nous avons utilisé une astuce pour cela jusqu'à maintenant : on plaçait les paramètres au début du tableau dans lequel on place le code. Cela a toujours bien marché d'ailleurs, mais c'est un coup de chance. En effet, pour ce faire, nous avons fait l'hypothèse implicite que le déplacement `OffPM%` est nul. Cette hypothèse n'est pas justifiée. Il vaut donc mieux utiliser la pile.

Rappelons que l'on distingue deux façons de passer les paramètres : *par valeur*, c'est-à-dire que l'on transmet la valeur d'une expression mais on ne peut pas changer le contenu de la variable dans le sous-programme si l'expression est une variable ; *par référence*, c'est-à-dire que le paramètre est nécessairement une variable, dont on peut changer le contenu dans le sous-programme.

QBasic permet à la fois le passage par valeur et par référence.

19.3.1 Passage des paramètres par valeur

Syntaxe.- On peut passer des paramètres par valeur :

```
CALL ABSOLUTE(BYVAL Par1, BYVAL Par2, Offset)
```

autant qu'on veut même si dans notre exemple il y en a deux. On a intérêt à les typer explicitement pour savoir combien on a mis d'octets sur la pile :

```
CALL ABSOLUTE(BYVAL Par1%, BYVAL Par2%, Offset)
```

Exemple.- Écrivons un programme QBasic qui demande un entier naturel (de deux octets) et fait appel à un sous-programme en langage machine qui passe la valeur en paramètre, récupère cette valeur, l'incrémente et la place comme premier élément du tableau dans lequel on écrit le code machine.

Le programme machine est :

```
PUSH BP
MOV BP, SP
MOV BX, [BP + 06]
INC BX
MOV CS:[00], BX
POP BP
RETF 2
```

avec BP + 6 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 écrit en langage symbolique en langage machine :

```
&H55
&H89 &HE5
&H8B &H5E &H06
&H43
&H2E &H89 &H1E &H00 &H00
&H5D
&HCA 02 00
```

Utilisons alors le programme QBasic suivant, sachant que les données auxiliaires occupent 2 octets et le code 16 octets :

```
CLS
DIM PM%(8)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Saisie de la donnee
INPUT "N = ", N%
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du tableau par le code machine
FOR I% = 0 TO 15
  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 PEEK(I%),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(BYVAL N%, OffPM% + 2)
```

```
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT PEEK(I%),
NEXT I%
'Retour au segment de depart
DEF SEG
'Affichage du resultat
PRINT N%
END
'Code machine
DATA 55
DATA 89,E5
DATA 8B,5E,06
DATA 43
DATA 2E,89,1E,00,00
DATA 5D
DATA CA,02,00
```

L'exécution se comporte comme prévu :

```
N = 5
0     0    55     89   E5
8B    5E   6      43   2E
89    1E   0      0    5D

6     0    55     89   E5
8B    5E   6      43   2E
89    1E   0      0    5D

 5
```

c'est-à-dire que 6 est placé à l'endroit prévu mais la valeur de N% n'a pas changée.

19.3.2 Passage des paramètres par référence

Syntaxe.- On peut passer des paramètres par référence :

`CALL ABSOLUTE(Par1, Par2, Offset)`

(sans le modificateur BYVAL) autant qu'on veut même si dans notre exemple il y en a deux. On n'a même pas besoin de les les typer explicitement car c'est l'adresse qui est placée sur la pile, plus exactement son déplacement (qui comporte toujours deux octets).

Exemple.- Écrivons un programme QBasic qui demande un entier naturel (de deux octets) et fait appel à un sous-programme en langage machine qui passe la variable contenant cet entier par référence, récupère la valeur de la variable, l'incrémente et la place comme premier élément du tableau dans lequel on écrit le code machine.

Le programme machine est :

```
PUSH BP
MOV BP, SP
MOV BX, [BP + 06]
```

```
MOV AX, [BX]
INC AX
MOV [BX], AX
MOV CS:[00], AX
POP BP
RETF 2
```

On récupère dans le registre BX l'adresse (plus exactement le déplacement de la variable N%), d'où l'intérêt de placer le contenu de la case mémoire de numéro BX dans le registre AX pour manipuler le contenu de N%. On replace la valeur manipulée dans la case mémoire adéquate, de numéro BX, si on veut que la manipulation soit visible par le programme appelant.

Traduisons ce programme écrit en langage symbolique en langage machine :

```
&H55
&H89 &HE5
&H8B &H5E &H06
&H8B &H07
&H40
&H89 &H07
&H2E &HA3 &H00 &H00
&H5D
&HCA 02 00
```

Utilisons alors le programme QBasic suivant, sachant que les données auxiliaires occupent 2 octets et le code 19 octets :

```
CLS
DIM PM%(10)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Saisie de la donnee
INPUT "N = ", N%
'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 PEEK(I%),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(N%, OffPM% + 2)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 14
  PRINT PEEK(I%),
NEXT I%
'Retour au segment de depart
DEF SEG
```

```
'Affichage du resultat
PRINT N%
END
'Code machine
DATA 55
DATA 89,E5
DATA 8B,5E,06
DATA 8B,07
DATA 40
DATA 89,07
DATA 2E,A3,00,00
DATA 5D
DATA CA,02,00
```

L'exécution se comporte comme prévu :

```
N = 5
0     0    55     89   E5
8B    5E   6      8B   7
40    89   7      2E   A3

0     0    55     89   E5
8B    5E   6      8B   7
40    89   7      2E   A3

 6
```

c'est-à-dire que 6 est placé à l'endroit prévu et la valeur de N% a également changée.