Chapitre 18

Programmation modulaire

Nous avons vu, lors de l'initiation à la programmation, la notion de *sous-programme* (éventuellement sous des noms divers de *fonction*, *procédure*...) et en quoi la *programmation modulaire* est intéressante. Nous allons voir comment mettre en place les sous-programmes en langage machine du microprocesseur 8086/8088.

18.1 Notions générales

Voyons quels sont les problèmes posés par un programmeur pour la mise en place d'un sous-programme en langage machine.

Repérage d'un sous-programme.- Un sous-programme est repéré par un nom dans le cas d'un langage évolué. Dans le cas d'un langage machine il sera repéré par une *adresse de début*. Il s'agit en général de l'adresse de sa première instruction mais pas toujours : il peut lui-même contenir des sous-programmes qui viennent avant lui ; le sous-programme proprement dit peut-être précédé de (déclaration de) constantes.

Arguments d'un sous-programme.- Contrairement au cas des langages évolués, un sous-programme en langage machine n'a pas d'arguments. Autrement dit tous les paramètres sont passés comme variables globales, soit à l'aide des registres, soit comme emplacement mémoire, soit à l'aide de la pile que nous étudierons dans le chapitre suivant.

Fin d'un sous-programme.- Il faut évidemment indiquer la dernière instruction d'un sous-programme, sinon toutes les instructions qui suivent cette instruction (considérées par nous comme la dernière) seront effectuées par le microprocesseur (en allant en particulier dans le sous-programme suivant). Un sous-programme se termine par une instruction spéciale indiquant qu'il s'agit de la dernière instruction.

Adresse de retour.- Lorsqu'on a terminé un sous-programme, le microprocesseur doit savoir où retourner dans le programme principal, autrement dit quelle est l'**adresse de retour**. Pour faciliter la vie du programmeur, cette adresse est en général conservée par le microprocesseur, au moment de l'appel du sous-programme, sur une *pile* (et qu'importe ce que c'est pour l'instant), puis récupérée, également par le microprocesseur de façon transparente pour le programmeur, au moment du retour ; le programmeur n'a pas à s'en préoccuper.

Place d'un sous-programme.- On peut placer le code d'un sous-programme où l'on veut. Si on ne commence pas par le *programme principal*, il faut le faire précéder d'un saut inconditionnel, ce qui permet de passer par-dessus le code du sous-programme.

Appel d'un sous-programme.- On pourrait appeler un sous-programme par un saut mais il est plus astucieux d'utiliser une instruction spéciale (qui s'occupe entre autre de placer l'adresse de retour sur la pile), dont l'opérande est l'adresse du sous-programme.

18.2 Cas du microprocesseur 8086/8088

18.2.1 Appel du sous-programme

Langage symbolique.- Un sous-programme peut être appelé grâce à l'instruction `CALL` (*appel* en anglais) dont l'opérande est l'adresse du sous-programme :

```
CALL adresse
```

Langage machine.- Uun sous-programme peut se trouver le même segment que le programme qui l'appelle : la valeur du registre IP est alors changée mais il est inutile de changer celle du registre CS. Il peut aussi se trouver dans un autre segment : les valeurs des registres IP et CS nécessitent alors d'être changées. L'appel peut s'effectuer de façon directe (en spécifiant l'adresse par une constante) ou indirecte (l'adresse se trouve dans un registre ou à un emplacement mémoire) :

– Dans le cas d'un **appel intra-segmentaire** direct, on précise l'adresse par un déplacement (entier relatif) par rapport à la valeur en cours de IP :

```
| 1110 1000 | déplacement bas | déplacement haut |
```

L'appel a lieu pour un programme commençant en [IP] + `déplacement`, où [IP] est l'adresse de l'instruction sui suit l'appel.

– Dans le cas d'un **appel inter-segmentaire** direct, on précise la valeur de IP et également celle de CS :

```
| 1001 1010 | IP bas | IP haut | CS bas | CS haut |
```

– Dans le cas d'un appel intra-segmentaire indirect, on précise la valeur de IP :

```
| 1111 1111 | mod 010 r/m |       |       |
```

– Dans le cas d'un appel inter-segmentaire indirect, on précise la valeur de IP et de CS, contenus dans un emplacement mémoire (un registre n'est pas suffisant) :

```
| 1111 1111 | mod 011 r/m |       |       |
```

avec $mod \neq 11$. Dans ce cas [IP] et [CS] sont remplacés par les contenus des cases mémoire (16 bits) définies par l'adresse effective et l'adresse effective + 2.

18.2.2 Retour d'un sous-programme

Langage symbolique.- Un sous-programme doit contenir au moins une instruction `RET` (pour *RETurn* ou *RETour*), indiquant la fin du sous-programme (et donc le retour au programme principal) de façon inconditionnelle :

```
RET
```

pour le retour d'un sous-programme intra-segmentaire et :

```
RETF
```

(avec F pour *Far*, lointain en anglais) pour le retour d'un sous-programme inter-segmentaire.

Langage machine.- Il existe deux instructions de retour :

– le retour court est codé sur un octet par `1100 0011`, soit C3h ;

– le retour long est également codé sur un octet par `1100 1011`, soit CBh.

18.2.3 Un exemple

Écrivons un programme en langage machine qui va chercher un entier naturel n, compris entre 0 et 255, à l'adresse 3000 :402 et qui place la valeur de l'octet faible de 2^n+n à l'adresse 3000 :404. Le calcul de cette fonction effectué grâce à un sous-programme écrit en langage machine.

Nous avons déjà écrit un programme en langage machine permettant de calculer l'exponentiation x^n. Nous retrouverons ainsi souvent des parties de programme déjà écrites. Autant les placer dans des sous-programmes et les réutiliser.

De façon à réutiliser au mieux notre sous-programme d'exponentiation vu précédemment, le programme principal placera 2 à l'adresse 3000 :400, fera appel au sous-programme de calcul de l'exponentielle, qui place le résultat à l'adresse &H0004, puis ajoutera le contenu de la case mémoire 3000 :402 à celui de celle d'adresse 3000 :404.

Le programme s'écrit de la façon suivante en langage symbolique :

```
       MOV AX,3000
       MOV DS,AX
       MOV WORD PTR [0400],2
       CALL EXP
       MOV AX,[0402]
       ADD [0404],AX
       RET
EXP:   MOV CX,[0402]
       MOV BX,[0400]
       MOV AX,1
DEBUT: MUL BX
       DEC CX
       JNZ DEBUT
       MOV [0404],AX
       RET
```

L'appel, et donc le retour, du sous-programme du programme en langage machine peut être court.

Traduisons maintenant ce programme écrit en langage symbolique en langage machine, en recopiant pour le sous-programme le programme écrit pour l'exponentiation :

```
B8 00 30
8E D8
C7 06 00 04 02 00
E8 08 00
A1 02 04
01 06 04 04
C3
8B 0E 02 04
8B 1E 00 04
B8 01 00
F7 E3
49
75 FB
A3 04 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 C7 06 00 04 02 00
-E CS:10C E8 08 00
-E CS:10E A1 02 04
-E CS:111 01 06 04 04
-E CS:115 C3
-E CS:116 8B 0E 02 04
-E CS:11A 8B 1E 00 04
-E CS:11E B8 01 00
-E CS:121 F7 E3
-E CS:123 49
-E CS:124 75 FB
-E CS:126 A3 04 04
-E CS:129 C3
-R BX
BX 0000
:
-R CX
CX 0000
:02A
-N EXPADD.COM
-W
Ecriture de 0002A octets
-Q
```

Si on place 3 comme donnée :

```
C:\COM>debug
-E 3000:402 03
-q
C:\COM> EXPADD.COM
```

Après exécution du programme `expadd.com`, on vérifie que l'on a bien $2^3 + 3 = 11 = \text{Bh}$ à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 02 00 03 00 0B 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
```

18.3 Historique

Les modèles de calcul, dont la machine de Turing, utilisent la notion de sous-programme
Comme nous l'avons vu au chapitre 15, le premier langage symbolique (EDSAC, 1949) ne prévoit pas d'instructions particulière pour manipuler les sous-programmes. Un sous-programme est utilisé dès le premier programme proposé, celui sur le calcul des carrés. Le mécanisme de sous-programme est alors entièrement géré par l'utilisateur.