Chapitre 8

Programmation modulaire

Nous avons vu, dans les chapitres précédents, les instructions fondamentales d'un microprocesseur, à savoir celles qui permettent d'émuler une machine de Turing. Nous allons commencer à voir, dans ce chapitre, des notions "avancées", c'est-à-dire non indispensables en théorie. Commençons par la mise en place des sous-programmes.

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* était intéressante. Nous allons voir comment mettre en place les sous-programmes en langage symbolique de `debug`, très proche du langage machine du microprocesseur `i8086`.

En fait, dans le cas du microprocesseur `i8086`, il y a deux types de sous-programmes, selon que ceux-ci sont placés dans le même segment que le programme appelant ou dans un autre segment. Il s'agit évidemment, une fois de plus, d'une question de compatibilité ascendante avec le `i8085`.

8.1 Notions générales

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

Repérage d'un sous-programme.- Un sous-programme est repéré par un nom dans le cas d'un langage évolué. Dans le cas du langage machine et du langage symbolique de `debug`, il sera repéré par une *adresse de début*, disons l'adresse de sa première instruction.

Arguments d'un sous-programme.- Contrairement au cas des langages évolués, un sous-programme en langage symbolique 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 verrons plus loin.

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) seraient 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.

Place d'un sous-programme.- On peut placer la définition 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.- Pour appeler un sous-programme, c'est-à-dire faire exécuter la portion de code qui en constitue le corps, il faut une instruction spéciale dont l'opérande est l'adresse du sous-programme.

8.2 Sous-programmes intra-segmentaire

Un **sous-programme** est **intra-segmentaire** si son code se trouve dans le même segment que le programme appelant.

8.2.1 Mise en place

Appel du sous-programme.- Un sous-programme est appelé grâce à l'instruction `call` (*appel* en anglais) dont l'opérande est l'adresse du sous-programme (plus exactement son décalage dans le segment).

Fin du sous-programme.- Le sous-programme se termine par l'instruction `ret` (pour *return* ou *retour*).

8.2.2 Un exemple

Problème.- Écrivons un programme qui calcule la somme des n premiers carrés $1^2 + 2^2 + ... + 3^2$.

Implémentation.- Écrivons un programme principal faisant appel à un sous-programme pour calculer le carré d'un entier. L'intérêt de cette façon de faire est qu'on peut alors facilement remplacer ce sous-programme par un qui calcule le cube, la puissance quatrième... d'un entier.

Le sous-programme utilisera `AL` pour l'entrée et le résultat sera dans `AX`. Le choix de ce registre est dû au fait qu'il est utilisé pour la multiplication. Pour le programme principal, on place l'entier n dans `CL` et on utilise `BX` pour le résultat.

Le programme.- Ceci nous conduit au programme suivant :

```
C:>debug
-a
249C:0100 mov cl, 3
249C:0102 mov bx,0
249C:0105 mov al,cl
249C:0107 call 114
249C:010A add bx,ax
249C:010C dec cl
249C:010E cmp cl,0
249C:0111 jnz 105
249C:0113 int 3
249C:0114 mov ah, 0
249C:0116 mul al
249C:0118 ret
249C:0119
-g
AX=0001 BX=000E CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=249C ES=249C SS=249C CS=249C IP=0113 NV UP EI PL ZR NA PE NC
249C:0113 CC INT 3
-q
```

dont on vérifie bien le résultat, à savoir `Eh` ou 14 pour la somme des trois premiers carrés.

8.3 Sous-programme inter-segmentaire

Un **sous-programme** est **inter-segmentaire** si son code se trouve dans un segment différent du segment dans lequel se trouve le programme appelant.

Syntaxe.- Lors de l'appel, comme dans le cas des sauts, on précise l'adresse en donnant l'adresse de segment et le déclage, c'est-à-dire qu'on utilise la forme :

```
call SSSS:DDDD
```

où `SSSS` est l'adresse de segment et `DDDD` le décalage.