Chapitre 16

Structures de contrôle

Nous avons vu, lors de l'étude de la modélisation des ordinateurs, que ce qui fait toute la puissance d'un ordinateur, et la différence avec une calculatrice, est ce qu'on appelle les *structures de contrôle*.

Nous avons vu également, en étudiant le principe de réalisation des ordinateurs, qu'il y a trois types de structures de contrôle sur un processeur :

– Le **séquencement** est la structure de contrôle par défaut et donc n'exige rien de particulier du point de vue du programmeur. Les instructions sont exécutées l'une après l'autre, dans l'ordre de la programmation, sauf dans le cas d'une **instruction de saut**.

– Un **saut inconditionnel** indique d'aller à telle instruction. Il faut repérer cette instruction, ce qui est fait bien entendu par son numéro (c'est-à-dire son adresse). L'utilisation d'un saut inconditionnel seul a peu d'intérêt puisque soit on saute une instruction une fois pour toute, soit on génère une boucle infinie.

– Un **saut conditionnel** indique d'aller à telle instruction lorsque telle situation est réalisée.

16.1 Saut inconditionnel

Le format d'un **saut inconditionnel** est :

`JMP adresse`

Le mnémonyme `jp` provient de l'anglais *to JumP*, c'est-à-dire *sauter*. Mais il faut distinguer plusieurs cas suivant la façon de spécifier l'adresse.

On distingue traditionnellement les **sauts absolus**, en spécifiant l'adresse, des **sauts relatifs**, en spécifiant le nombre d'octets à sauter. Les sauts absolus sont certainement les plus naturels mais présentent une difficulté lorsqu'on veut reloger une partie de code à une adresse différente : il faut alors recalculer toutes les adresses alors qu'on n'a pas besoin de le faire avec les sauts relatifs.

Le 8086/8088 introduit une notion nouvelle, non nécessaire pour les autres microprocesseurs. On distingue les **sauts intrasegmentaires**, c'est-à-dire à une adresse se trouvant dans le même segment de code, pour lesquels il suffit de spécifier le contenu IP sans changer le contenu de CS, des **sauts intersegmentaires**, pour lesquels il faut spécifier à la fois le contenu de IP et celui de CS.

Enfin, et même si cela ne nous intéresse pas dans une première étape, comme toujours avec le 8086/8088, on peut spécifier ce qu'il faut par une donnée, **saut direct**, soit par le contenu d'un registre, **saut indirect**.

16.1.1 Saut inconditionnel intrasegmentaire, direct et relatif

Langage symbolique.- On distingue les **sauts courts**, sur une étendue de 256 octets (saut relatif de - 128 à + 127) :

`JMP SHORT deplacement`

des **sauts longs**, sur la totalité du segment de - 32 768 à + 32 767) :

`JMP deplacement`

L'intérêt n'est pas immédiat pour le programmeur mais on aura compris, qu'au niveau du code machine, on utilise un ou deux octets pour spécifier le déplacement.

Langage machine.- Le saut court est codé sur deux octets :

`| 1110 1011 | deplacement |`

soit &HEB `deplacement`, alors que le saut long est codé sur trois octets :

`| 1110 1001 | deplacement bas | deplacement haut |`

soit &E9 `deplacement`.

La représentation des entiers relatifs s'effectue en complément à deux, notion que nous étudierons plus tard.

Exemple.- Écrivons un sous-programme qui additionne 1, 2 et 3 et place le résultat à l'emplacement mémoire `&H0000` mais en effectuant un saut inconditionnel (court) avant l'addition de 3, ce qui a pour conséquence que la dernière addition n'est pas prise en compte.

Le sous-programme s'écrit en langage symbolique :

```
MOV AX, 1
ADD AX, 2
JMP saut
ADD AX, 3
saut : MOV CS :[0000], AX
```

```
RETF
```

On remarque la difficulté pour l'écriture en langage machine car il faut déterminer soit l'adresse adéquate, soit le déplacement adéquat. On a ajouté une notion d'**étiquette** dans notre langage symbolique pour pallier à ce problème.

Traduisons-le maintenant en langage machine :

```
&HB8 &H01 &H00
&H05 &H02 &H00
&HEB &H??
&H05 &H03 &H00
&H2E &HA3 &H00 &H00
&HCB
```

Nous avons laissé indéterminé le déplacement pour l'instant. Maintenant que notre programme est écrit, on s'aperçoit qu'il s'agit de 3 puisque l'instruction d'addition que l'on veut sauter est codée sur trois octets. On peut donc réécrire la troisième instruction de la façon suivante :

```
&HEB &H03
```

Nous pouvons utiliser le programme QBasic suivant pour tester notre sous-programme, sachant que le résultat occupera deux octets et le code 16 octets :

```
CLS
DIM PM%(7)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'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 12
  PRINT PEEK(I%),
NEXT I%
'Execution du programme machine
CALL ABSOLUTE(OffPM% + 2)
'Visualisation du contenu du debut de ce segment apres execution
FOR I% = 0 TO 12
  PRINT PEEK(I%),
NEXT I%
'Retour au segment de depart
DEF SEG
'Affichage du resultat
PRINT PM%(0)
END
DATA B8,01,00
DATA 05,02,00
DATA EB,03
DATA 05,03,00
DATA 2E,A3,00,00
```

```
DATA CB
```

L'exécution se comporte comme prévu en affichant 3 :

```
0    0    B8    1    0
5    2    0     EB   3
5    3    0     2E   A3

3    0    B8    1    0
5    2    0     EB   3
5    3    0     2E   A3

3
```

Incidence sur les indicateurs.- Aucun indicateur n'est affecté.

16.1.2 Autres sauts inconditionnels

Saut inconditionnel intrasegmentaire indirect.- Dans ce cas, la nouvelle valeur de IP est le contenu d'une case mémoire ou d'un registre. Cette instruction est codé en langage machine sur deux ou quatre octets :

```
| 1111 1111 | mod 100 r/m |         |         |
```

Saut inconditionnel intersegmentaire.- Dans le cas direct, il faut spécifier la nouvelle valeur de IP et la nouvelle de CS. Cette instruction est codé en langage machine sur cinq octets :

```
| 1110 1010 | IPB | IPH | CSB | CSH |
```

Dans le cas indirect, on spécifie une adresse dans une case mémoire, adresse occupant quatre octets, les deux premiers spécifiant IP et les deux derniers CS. On ne peut donc pas, contrairement à ce qui se fait d'habitude, utiliser le contenu d'un registre. Cette instruction est codé en langage machine sur deux ou quatre octets :

```
| 1111 1111 | mod 101 r/m |         |         |
```

avec $r/m \neq 11$.

16.2 Sauts conditionnels

16.2.1 Les différents cas

Langage symbolique.- Le format d'un **saut conditionnel**, toujours relatif, est :

```
Jcondition deplacement
```

où le `deplacement` est codé sur un octet, avec une valeur comprise entre - 128 et + 127.

Sémantique.- On saute à l'instruction dont l'adresse est indiquée si la condition est remplie. Si la condition n'est pas réalisée, c'est l'instruction suivante qui est exécutée.

un nombre en hexadécimal et `condition` est l'une des huit abréviations suivantes :

Langage machine.- Ces sauts sont codés sur deux octets :

```
opcode deplacement
```

Différents cas.- Les différents cas sont résumés dans le tableau suivant :

Mnémonymes	Opcode		Conditions	
	Binaire	Hexa		
Tests non signés (supérieur/inférieur)				
JA/JNBE	0111 0111	77	(CF + ZF) = 0	$>$
JAE/JNB	0111 0011	73	CF = 0	$\geq$
JB/JNAE	0111 0010	72	CF = 1	$<$
JBE/JNA	0111 0110	76	(CF + ZF) = 1	$\leq$
JC	0111 0010	72	CF = 1	
JE/JZ	0111 0100	74	ZF = 1	$=$
JNC	0111 0011	73	CF = 0	
JNE/JNZ	0111 0101	75	ZF = 0	$\neq$
JNP/JPO	0111 1011	7B	PF = 0	
JP/JPE	0111 1010	7A	PF = 1	
Tests signés (plus grand que/plus petit que)				
JG/JNLE	0111 1111	7F	((SF $\oplus$ OF) + ZF) = 0	$>$
JGE/JNL	0111 1101	7D	(SF $\oplus$ OF) = 0	$\geq$
JL/JNGE	0111 1100	7C	(SF $\oplus$ OF) = 1	$<$
JLE/JNG	0111 1110	7E	((SF $\oplus$ OF) + ZF) = 1	$\leq$
JNO	0111 0001	71	OF = 0	
JNS	0111 1001	79	SF = 0	> 0
JO	0111 0000	70	OF = 1	
JS	0111 1000	78	SF = 1	< 0

dans lequel $+$ désigne la disjonction et $\oplus$ la disjonction exclusive. Les mnémonymes sont clairs en anglais :

- `JA` pour *Jump if Above*, saut si strictement supérieur à.
- `JNBE` pour *Jump if Not Below or Equal*, saut si non inférieur, ni égal à.
- `JAE` pour *Jump if Above or Equal*, saut si supérieur ou égal à.
- `JNB` pour *Jump if Not Below*, saut si non strictement inférieur à.
- `JB` pour *Jump if Below*, saut si strictement inférieur à.
- `JNAE` pour *Jump if Not Above or Equal*, saut si non supérieur, ni égal à.
- `JBE` pour *Jump if Below or Equal*, saut si inférieur ou égal à.
- `JNA` pour *Jump if Not Above*, saut si non strictement supérieur à.
- `JC` pour *Jump if Carry*, saut si retenue.
- `JE` pour *Jump if Equal*, saut si égal à.
- `JZ` pour *Jump if Zero*, saut si nul.
- `JNC` pour *Jump if Not Carry*, saut s'il n'y a pas de retenue.
- `JNE` pour *Jump if Not Equal*, saut si non égaux.
- `JNZ` pour *Jump if Not Zero*, saut si non nul.
- `JNP` pour *Jump if Not Parity*, saut si pas parité.
- `JPO` pour *Jump if Parity Odd*, saut si parité impaire.
- `JP` pour *Jump if Parity*, saut s'il y a parité.
- `JPE` pour *Jump if Parity Even*, saut si parité paire.
- `JG` pour *Jump if Greater than*, saut si strictement plus grand que.
- `JNLE` pour *Jump if Not Less than or Equal*, saut si non plus petit ou égal à.
- `JGE` pour *Jump if Greater than or Equal*, saut si supérieur ou égal à.
- `JNL` pour *Jump if Not Less than*, saut si non strictement inférieur à.
- `JL` pour *Jump if Less than*, saut si strictement plus petit que.
- `JNGE` pour *Jump if Not Greater or Equal*, saut si non plus grand ou égal à.

- `JLE` pour *Jump if Less or Equal*, saut si plus petit ou égal à.
- `JNG` pour *Jump if Not Greater than*, saut si non strictement plus grand que.
- `JNO` pour *Jump if Not Overflow*, saut si pas de dépassement de capacité.
- `JNS` pour *Jump if Not Signed*, saut si strictement positif.
- `JO` pour *Jump if Overflow*, saut s'il y a dépassement de capacité.
- `JS` pour *Jump if Signed*, saut si strictement négatif.

Affectation des indicateurs.- Les conditions prises en compte lors des sauts conditionnels sont les valeurs du registre des indicateurs. Il faut donc pouvoir initialiser celui-ci ou, tout au moins, savoir ce qui fait changer son contenu. Comme nous l'avons déjà dit, on ne peut pas changer directement son contenu ; celui-ci est revu après le déroulement de chaque instruction et, en particulier pour ce qui nous intéresse, après chaque instruction arithmétique. On doit donc décrire soigneusement comment chaque instruction modifie le registre des indicateurs. La plupart de ces changements sont naturels vu le choix du nom des indicateurs ; on peut hésiter dans certains cas.

Incidence sur les indicateurs.- Aucun indicateur n'est affecté.

16.2.2 Exemple : l'exponentiation

L'exponentiation $(a, b) \mapsto a^b$ sur les entiers naturels est implémentée en QBasic mais pas sur le 8088, comme nous l'avons vu. Programmons l'exponentiation. Un algorithme naïf pour cela (pas très efficace mais suffisant pour notre test) consiste à multiplier 1 par a, plus exactement b fois a.

Supposons que a, de capacité un mot, se trouve à l'adresse &H0000, que b, de capacité un mot également, se trouve à l'adresse &H0002 et que l'on veuille que le résultat, un double mot, se trouve, après exécution du sous-programme en langage machine, à l'adresse &H004.

Il y a rapidement dépassement de capacité mais nous ne nous en occuperons pas (tout au moins pour l'instant). Notre résultat ne sera valable que si le résultat tient sur un mot.

L'idée du sous-programme est la suivante, en supposant que b est non nul. On initialise le registre CX avec b. On initialise le registre BX avec a. On place 1 dans l'accumulateur. On multiplie l'accumulateur par BX, c'est-à-dire a. On décrémente CX. Si CX est non nul, on recommence à partir de la multiplication de l'accumulateur par BX. Si CX est nul, on a terminé, c'est-à-dire que le résultat se trouve dans AX, DX ; on place alors le contenu de AX dans la case mémoire &H0004.

Ce sous-programme s'écrit en langage symbolique :

```
MOV CX, CS :[02h]
MOV BX, CS :[00h]
MOV AX, 1
DEBUT : MUL BX
DEC CX
JNZ DEBUT
MOV CS :[04], AX
RETF
```

Traduisons maintenant ce sous-programme en langage machine. La première instruction, en oubliant le préfixe, est codée par :

```
| 1000 1011 | 00 001 110 | 0000 0020 | 0000 0000 |
```

soit &H8B &H0E &H02 &H00. La cinquième instruction est codée par :

```
| 0100 1 001 |
```

soit &H49. La sixième instruction est codée par &H75 &H ? ?, nous déterminerons le déplacement après avoir écrit une première esquisse du programme.

```
&H2E &H8B &H0E &H02 &H00
&H2E &H8B &H1E &H00 &H00
&HB8 &H01 &H00
&HF7 &HE3
&H49
&H75 &H??
&H2E &HA3 &H04 &H00
&HCB
```

Le déplacement est - 5. Nous avons besoin de savoir comment représenter - 5, ce que nous avions réservé à plus tard. Il s'agit du complément à deux, c'est-à-dire que 5 se représente en binaire sur un octet par : 0000 0101 ; il suffit de remplacer 1 par 0 et 0 par 1, ce qui donne 1111 1010, ce qui donne le complément à 1, puis on lui ajoute 1, ce qui donne 1111 1011, soit &HFB. Le code du sous-programme est donc :

```
&H2E &H8B &H0E &H02 &H00
&H2E &H8B &H1E &H00 &H00
&HB8 &H01 &H00
&HF7 &HE3
&H49
&H75 &HFB
&H2E &HA3 &H04 &H00
&HCB
```

Nous pouvons utiliser le programme QBasic suivant pour tester notre sous-programme, sachant que les données et le résultat occupent 8 octets et le code 23 octets :

```
CLS
DIM PM%(15)
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 22
  READ Octet$
  POKE OffPM% + 8 + 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% + 8)
'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%(2)
END
'Code machine
DATA 2E,8B,0E,02,00
DATA 2E,8B,1E,00,00
DATA B8,01,00
DATA F7,E3
DATA 49
DATA 75,FB
DATA 2E,A3,04,00
DATA CB
```

L'exécution se comporte comme prévu :

```
A = 2
B = 10
2     0    A      0    0
0     0    0      2E   8B
E     2    0      2E   8B

2     0    A      0    0
4     0    0      2E   8B
E     2    0      2E   8B

 1024
```

ce qui donne le bon résultat, à savoir 1024, au vu des données choisies, à savoir 2 et 10.

Commentaire.- Nous avons vu au passage une première façon de passer des paramètres au sous-programme écrit en code machine.

16.2.3 La comparaison

Dans notre exemple, nous avons eu de la chance : la dernière instruction avant la comparaison, à savoir la décrémentation, affecte les indicateurs. Ceci n'est pas toujours le cas. Pour appliquer les sauts conditionnels, il faut évaluer une condition. Pour cela, on se sert souvent de l'instruction de comparaison.

Langage symbolique.- L'instruction :

```
CMP destination, source
```

compare le contenu des deux opérandes en effectuant une soustraction (`destination - source`). Le résultat n'est placé nulle part mais les indicateurs sont affectés, de la façon spécifiée dans le tableau ci-dessous pour $a - b$:

		$OF \oplus SF$	ZF	CF
a et b positifs	$a > b$	0	0	0
	$a = b$	0	1	0
	$a < b$	1	0	1
a et b négatifs	$a > b$	0	0	0
	$a = b$	0	1	0
	$a < b$	1	0	1
a négatif, b positif	$a < b$	1	0	0
a positif, b négatif	$a > b$	0	0	1

Indicateurs affectés.- Les indicateurs affectés après une instruction de comparaison sont AF, CF, OF, PF, SF et ZF.

Langage machine.- Nous avons trois possibilités selon la nature des opérandes :
– Dans le cas d'une comparaison entre mémoire ou registre avec registre, on code sur deux ou quatre octets :

`| 0011 10 dw | mod reg r/m |        |        |`

– Dans le cas d'une comparaison immédiate avec l'accumulateur (AL ou AX), on code sur deux ou trois octets :

`| 0011 110 w | donnée | donnée si` $w = 1$ `|`

– Dans le cas d'une comparaison immédiate avec un registre ou une case mémoire, on code sur trois ou quatre octets :

`| 1000 00sw | mod 111 r/m | donnée | donnée si` $s = 0$ `et` $w = 1$ `|`

16.3 Historique

Le concept de 'structure de contrôle' apparaît dès les premiers modèles de calcul (voir chapitre un). En ce qui concerne la réalisation des calculateurs, comme nous l'avons vu, les structures de contrôle ne sont pas mises en place dans les machines de Zuse. C'est le rapport [Von-45] sur l'EDVAC de von Neumann qui va à nouveau attiré l'attention sur elles.

Le premier ordinateur (EDSAC, 1949 ; voir chapitre 14) n'utilise pas de saut inconditionnel mais deux types de sauts conditionnels. En 1952, une instruction de saut inconditionnel fut ajoutée, mais cela conduisit à réécrire de nombreuses sous-routines.

16.4 Bibliographie

[Ran-82] Randell, Brian, **The origins of Digital Computers**, Springer, 1984.

[Ste-81] Stern, Nancy, **From ENIAC to UNIVAC : An appraisal of the Eckert-Mauchly Computers**, Digital Presse, Bedford, Mass., 1981.

[Von-45] von Neumann, John, **First Draft of a Report on the EDVAC**, contract N^o W-670-ORD-492, Moore School of Electrical Engineering, University of Pennsylvania, 30 juin 1945, 101 p. Extraits dans [Ran-82], pp. 383–392. Reed. in [Ste-81], pp. 177–246 et avec des commentaires dans **Annals of the History of Computing**, vol. 15, 1993, n^o 4, pp. 27–75. Version électronique disponible à l'adresse :

`http://www.virtualtravelog.net/entries/2003-08-TheFirstDraft.pdf`