Chapitre 15

Calculs sur les entiers naturels

Nous avons vu que les grandes fonctions d'un microprocesseur sont les accès à celui-ci et les calculs. Nous avons commencé à étudier les premières au chapitre précédent. Voyons maintenant comment effectuer des calculs, en commençant par les calculs sur les entiers naturels. Avant cela, continuons un peu les instruvtions de transfert.

15.1 Compléments sur les instructions de transfert

15.1.1 Transfert de mémoire vive à accumulateur

Langage symbolique.- L'instruction de transfert d'accumulateur à mémoire vive s'écrit :

```
MOV A, [decalage]
```

où `A` désigne l'un des registres `AL` ou `AX` et `decalage` le décalage de l'adresse, qui est un entier de seize bits. Le segment par défaut est le segment de données.

Sémantique.- La signification de cette instruction est la copie du contenu de l'élément de mémoire vive d'adresse spécifiée vers l'accumulateur.

Remarque.- Il s'agit de l'analogue de l'instruction PEEK du BASIC.

Langage machine.- Le transfert mémoire à accumulateur exige trois octets :

```
opcode faible fort
```

où `faible` est l'octet de poids faible du décalage de l'adresse, `fort` l'octet de poids fort et `opcode` est :

```
1010 000w
```

avec, évidemment, `w` égal à 0 pour `AL` et à 1 pour `AX`, soit &HA0 ou $HA1.

<u>Exercice corrigé</u>.- *Traduire l'instruction :*

```
MOV AX, [A900h]
```

en langage machine.
D'après ce que nous venons de dire, cette instruction se traduit par les trois octets :

```
1010 0001 0000 000 1010 1001
```

en binaire, soit &HA1, &H00, &HA9 en hexadécimal.

15.1.2 Transfert entre registres ou registre et case mémoire

<u>Langage symbolique</u>.- L'instruction de transfert entre registres ou registre et case mémoire s'écrit :

```
MOV destination, source
```

ce qui ne nous apprend pas grand chose.

r/m	AE
000	[BX] + [SI]
001	[BX] + [DI]
010	[BP] + [SI]
011	[BP] + [DI]
100	[SI]
101	[DI]
110	déplacement
111	[BX]

FIG. 15.1 – Modes d'adressage du 8088 lorsque `mod = 00`

<u>Langage machine</u>.- On a deux octets dans le cas de registres et deux ou quatre octets dans le cas d'une case mémoire :

```
1000 10dw | mod reg r/m |           |           |
```

- Dans le premier octet, le bit `d` (de destination) est égal à 1 si la destination est un registre, il s'agit alors du regsitre `reg` du deuxième octet. Le bit `w` est égal à 0 pour le transfert d'un octet et à 1 pour le transfert de deux octets (un mot).
- Le deuxième octet du code opération, que nous retrouverons souvent, spécifie le mode d'adressage et le ou les registres concernés :
  - Les trois bits de `reg` spécifient le registre dans le cas d'un transfert entre registre et case mémoire et le registre de destination dans le cas d'un transfert entre registres. Leur signification est déterminée par le tableau de la figure 13.1.
  - les deux bits de `mod` et les trois bits de `r/m` spécifient le mode d'adressage :

- Si `mod = 11`, on a un transfert entre registres et `r/m` spécifie le second registre suivant le tableau de la figure 13.1.
- Si `mod = 00`, on a un transfert entre registre et case mémoire (rappelons que le sens est déterminé par le bit `d` du premier octet), l'**adresse effective** AE de la case mémoire étant spécifiée par `r/m` suivant le tableau de la figure 15.1. Seul le cas `r/m = 110` nous intéressera pour l'instant. Dans ce cas, le déplacement (sur 16 bits) doit être spécifié par deux octets supplémentaires. Nous étudierons les autres cas plus tard ; contentons-nous de dire que, par exemple, `[BX]` signifie le contenu du registre `BX`.
- Les modes `mod = 01` et `mod = 10` ne nous intéresserons pas pour l'instant : le décalage est la somme de contenus de registres et d'un déplacement spécifié sur un ou deux octets.

<u>Exercice corrigé 1</u>.- *Traduire l'instruction* :

```
MOV BX, AX
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire :

```
1000 1011 | 11 011 000
```

soit &H8B &HD8 en hexadécimal.

<u>Vocabulaire</u>.- Il est traditionnel de parler d'**adressage implicite** ou d'**adressage de registre** dans le cas de copie entre registres. Le nom d'adressage *implicite* provient de ce que les opérandes (les deux registres concernés) ne sont pas indiqués par un octet distinct : c'est le code d'opération qui contient (qui implique) les noms des registres concernés.

<u>Exercice corrigé 2</u>.- *Traduire l'instruction* :

```
MOV BX, [10h]
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les quatre octets représentés en binaire :

```
1000 1011 | 00 011 110 | 0001 000 | 0000 0000
```

soit &H8B &H1D &H10 &H00 en hexadécimal.

15.2 Incrémentation et décrémentation

15.2.1 Les opérations arithmétiques

Les premières opérations auxquelles on s'attend sur un ordinateur concernent, bien entendu, les opérations sur les entiers naturels, à savoir essentiellement l'addition, la multiplication et la soustraction (cette dernière opération étant une opération partielle).

En fait on ne peut pas manipuler l'ensemble des entiers naturels mais seulement une partie d'entre eux, celle-ci dépendant de la taille des registres.

L'opération fondamentale est, en théorie, l'*incrémentation*, c'est-à-dire le passage au suivant, car elle permet, en utilisant les *structures de contrôle* que nous verrons après, d'obtenir toutes les opérations qui nous intéressent. Cependant l'addition est câblée sur tous les microprocesseurs, ce qui n'est pas toujours le cas de la multiplication.

On peut décomposer tout « calcul » en calculs élémentaires (il faut bien partir de quelque chose) et en combinant ceux-ci. Les instructions fondamentales d'un microprocesseur sont donc les *instructions de calcul élémentaire* et les *instructions de contrôle*.

Instructions de calcul élémentaire.- L'ordinateur est un descendant des machines à calculer perfectionnées, qui calculaient beaucoup plus vite que l'être humain : machines mécaniques puis électro-mécaniques puis électroniques. Il résulte de cet héritage que les opérations courantes, tout au moins les opérations les plus élémentaires, sont implémentées sur le microprocesseur. C'est le cas en général de l'addition et de la soustraction (éventuellement de la multiplication) sur les entiers naturels. La division euclidienne est aussi souvent implémentée. Les opérations logiques (conjonction, disjonction, négation globales et bits à bits et les rotations des bits) sont également implémentées, à la fois parce que c'est facile et parce qu'on les utilise très souvent en programmation système, alors qu'elles ne l'étaient pas sur les machines à calculer.

Instructions de contrôle.- Vous avez certainement vu, dans le cours d'initiation à la programmation (sous-entendu en langage évolué), qu'en langage évolué on dispose de beaucoup moins d'opérations que sur une calculette scientifique mais que ce qui fait toute la force d'un tel langage sont ses instructions de contrôle. On doit donc retrouver de telles instructions au niveau du microprocesseur, même s'il ne s'agit pas du même jeu d'instructions.

15.2.2 Représentation des entiers naturels

Amplitude des entiers naturels manipulés.- Puisque la taille des mots sur un 8088 est de 16 bits (deux octets), on peut manipuler les entiers naturels allant de `&H0` à `&HFFFF`, c'est-à-dire de 0 à 65 535 (= 2^{16} - 1).

Entiers plus grands.- On peut aussi coder un entier naturel sur plusieurs mots (sous-entendu machine). Nous verrons qu'alors il faut utiliser plus d'instructions pour pouvoir effectuer des opérations sur ces « grands entiers ».

15.2.3 Incrémentation

L'**incrémentation** fait passer d'un entier au suivant, autrement dit ajoute plus un à un entier.

15.2.3.1 Incrémentation d'un registre 16 bits

Langage symbolique.- Le passage au suivant pour le contenu d'un registre 16 bits s'écrit :

```
INC reg
```

où `reg` est un registre de 16 bits.

Sémantique.- Après cette instruction, le contenu du registre est augmenté de 1. L'**arithmétique** est **modulaire**, c'est-à-dire que l'on passe de `&HFFFF` à 0.

Langage machine.- Cette incrémentation s'effectue grâce à une instruction codée sur un octet :

```
0100 0 reg
```

où le registre est désigné de la façon habituelle (tableau de la figure 13.1).

Exercice corrigé.- *Traduire l'instruction* :

```
INC AX
```

en langage machine.

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

```
0100 0 000
```

soit &H40 en hexadécimal.

Exemple.- Écrivons un sous-programme en langage machine qui initialise l'accumulateur à 17, incrémente l'accumulateur puis place son contenu dans l'emplacement mémoire de déplacement &000A du segment de code en cours.

Le programme s'écrit en langage symbolique :

```
MOV AX, 11h
INC AX
MOV CS :[000A], AX
RET
```

Nous avons déjà traduit toutes ces instructions en langage machine. Le programme s'écrit donc en langage machine :

```
&HB8 &H11 &H00
&H40
&H2E &HA3 &H0A &H00
&HCB
```

Nous pouvons donc utiliser le programme QBasic suivant pour tester notre sous-programme :

```
CLS
DIM PM%(5)      'PM comme Programme Machine
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 8
  READ Octet$
  POKE OffPM% + i%, VAL("&H" + Octet$)
NEXT i%
'Visualisation du contenu du debut de ce segment avant execution
PRINT PEEK(0), PEEK(1), PEEK(2), PEEK(3), PEEK(4)
PRINT PEEK(5), PEEK(6), PEEK(7), PEEK(8), PEEK(9), PEEK(10), PEEK(11)
'Execution du programme machine
CALL ABSOLUTE(OffPM%)
'Visualisation du contenu du debut de ce segment apres execution
PRINT PEEK(0), PEEK(1), PEEK(2), PEEK(3), PEEK(4)
PRINT PEEK(5), PEEK(6), PEEK(7), PEEK(8), PEEK(9), PEEK(10), PEEK(11)
'Retour au segment de depart
DEF SEG
PRINT PM%(5)
```

```
END
DATA B8,11,00
DATA 40
DATA 2E,A3,0A,00
DATA CB
```

dont l'affichage de 18 :

```
184   17     0    64    46
163   10     0    203    0    0    0

184   17     0    64    46
163   10     0    203    0   18    0

18
```

de deux façons différentes, nous laisse penser que le sous-programme en langage machine s'est correctement déroulé.

Commentaires.- 1^o) Ne pas oublier de changer la dimension du tableau `PM%` pour tenir compte du nombre plus élevé d'instructions.

- 2^o) Ne pas oublier non plus de changer la limite supérieure de la boucle POUR, également pour tenir compte du nombre plus élevé d'instructions.

- 3^o) Nous avons ajouté de même quelques PEEK pour voir plus loin dans la mémoire.

- 4^o) Puisque nous avons déplacé l'emplacement où placer le contenu de l'accumulateur dans la mémoire vive, il faut en tenir compte lors de l'affichage (on affiche `PM%(5)` et non plus `PM%(4)`).

- 5^o) Pourquoi d'ailleurs avons-nous changé cet emplacement ? Que se passe-t-il si on ne le fait pas ? Essayez ! Pouvez-vous expliquer ce qui s'est passé ?

La troisième instruction place `&H08` *à l'emplacement de la quatrième instruction. Lorsque celle-ci est exécutée, au lieu de rencontrer le retour* `&HCB`*, on rencontre donc* `&H08`*, qui est exécuté. Qu'importe à quelle instruction cela correspond, il ne s'agit pas de RETF, on ne revient pas au programme principal. On a un écran vide duquel on ne peut pas d'en sortir. Il faut redémarrer l'ordinateur.*

Ceci explique pourquoi les systèmes d'exploitation « modernes » séparent la partie de la mémoire vive consacrée au programme de celle consacrée aux données. Cet incident ne serait pas survenu si on avait suivi cette politique.

Exercice.- *Vérifier que le suivant de* `FFFFh` *est* 0.

Exercice^L.- *Concevoir un circuit (combinatoire) qui permette d'effectuer l'incrémentation.*

15.2.3.2 Incrémentation d'un registre (8 ou 16 bits) ou d'une case mémoire (ou deux)

Langage symbolique.- Le passage au suivant s'écrit toujours :

```
INC var
```

et la sémantique est la même, c'est le langage machine qui change.

Langage machine.- Cette incrémentation s'effectue grâce à une instruction codée sur deux à quatre octets :

```
1111 111w | mod 000 r/m |        |
```

avec les interprétations, maintenant habituelles, de `w`, de `mod` et de `r/m`.

Exercice corrigé.- *Traduire l'instruction* :

```
INC AL
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représenté en binaire :

```
1111 1110 |11 000 00
```

soit &HFE &HC0 en hexadécimal.

15.2.4 Décrémentation

La **décrémentation** fait passer d'un entier au précédent, c'est-à-dire lui retranche un.

15.2.4.1 Décrémentation d'un registre 16 bits

Langage symbolique.- Le passage au précédent pour le contenu d'un registre 16 bits s'écrit :

```
DEC reg
```

où `reg` est un registre de 16 bits.

Sémantique.- Après cette instruction, le contenu du registre est diminué de 1. L'**arithmétique** est **modulaire**, c'est-à-dire que l'on passe de 0 à `&HFFFF`.

Langage machine.- Cette décrémentation s'effectue grâce à une instruction codée sur un octet :

```
0100 1 reg
```

où le registre est désigné de la façon maintenant habituelle (tableau de la figure 13.1).

Exercice corrigé.- *Traduire l'instruction* :

```
DEC AX
```

en langage machine.

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

```
0100 1 000
```

soit &H48 en hexadécimal.

Exercice.- *Écrire un sous-programme en langage machine qui initialise l'accumulateur à 17, décrémente l'accumulateur puis place son contenu dans l'emplacement mémoire de déplacement*

&000A du segment de code en cours puis utiliser un programme QBasic pour tester ce sous-programme.

Exercice.- *Vérifier que le précédent de* 0 *est* `FFFFh`.

Exercice[L].- *Concevoir un circuit (combinatoire) qui permette d'effectuer la décrémentation.*

15.2.4.2 Décrémentation d'un registre (8 ou16 bits) ou d'une case mémoire (ou deux)

Langage symbolique.- Le passage au précédent s'écrit toujours :

```
DEC var
```

et la sémantique est la même, c'est le langage machine qui change.

Langage machine.- Cette décrémentation s'effectue grâce à une instruction codée sur deux à quatre octets :

```
1111 111w | mod 001 r/m |        |
```

avec les interprétations, maintenant habituelles, de `w`, de `mod` et de `r/m`.

Exercice corrigé.- *Traduire l'instruction* :

```
DEC AL
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire par :

```
1111 1110 |11 001 00
```

soit &HFE &HC8 en hexadécimal.

15.3 Addition

L'addition peut s'effectuer tout naturellement sur un mot, de deux octets puisque le 8086 est un microprocesseur 16 bits. Par compatibilité avec les microprocesseurs 8 bits, l'addition sur un octet est également prévue. On peut également mettre en place des additions sur plusieurs mots (pour les « grands » entiers), mais pas en une seule fois.

15.3.1 Addition sur un octet ou un mot

15.3.1.1 Addition immédiate dans l'accumulateur

Langage symbolique.- L'instruction est :

```
ADD A, constante
```

où `A` est l'un des accumulateurs `AX` ou `AL` et `constante` un entier codé sur seize ou huit bits suivant le cas. On parle quelquefois d'**addition immédiate** pour cette instruction faisant intervenir une constante.

Sémantique.- Après l'exécution de cette instruction le contenu de l'accumulateur est la somme des contenus de l'accumulateur et de la constante indiquée.

Langage machine.- Cette addition s'effectue grâce à une instruction codée sur deux ou trois octets suivant le cas :

```
0000 010w | donnée | donnée si w = 1
```

concernant `AX` si $w = 1$ et `AL` si $w = 0$.

Exercice corrigé.- *Traduire l'instruction* :

```
ADD AL, 3h
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire par :

```
0000 0100 | 0000 0011
```

soit &H04 &H03 en hexadécimal.

Exercice corrigé.- *Traduire l'instruction* :

```
ADD AX, 1FC4h
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les trois octets représentés en binaire par :

```
0000 0101 |1100 01000 | 0001 1111
```

soit &H05 &HC4 &H1F en hexadécimal.

Test d'un programme.- Écrivons un sous-programme machine qui place 17 dans l'accumulateur `AX`, ajoute 5 à celui-ci puis place le résultat dans l'emplacement mémoire &H000C du segment de code. En langage symbolique nous avons :

```
MOV AX, 11h
ADD AX, 5h
MOV CS :[0Ch], AX
RETF
```

Traduisons-le en langage machine :

```
&HB8 &H11 &H00
&H05 &H05 &H00
&H2E &HA3 &H0C &H00
&HCB
```

dont la longueur est 11 octets, ce qui explique pourquoi nous plaçons le résultat à partir du douzième octet.

Écrivons enfin le programme QBasic suivant pour le tester :

```
CLS
DIM PM%(6)
```

```
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 10
  READ Octet$
  POKE OffPM% + 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%)
'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%(6)
END
DATA B8,11,00
DATA 05,05,00
DATA 2E,A3,08,00
DATA CB
```

dont l'affichage de 22 :

```
184   17     0     5     5
0     46   163    12     0
203    0     0     0     0

184   17     0     5     5
0     46   163    12     0
203    0    22     0     0

22
```

de deux façons différentes, nous laisse penser que le sous-programme en langage machine s'est correctement déroulé.

Commentaire.- Nous avons profité, puisque l'amplitude de la mémoire visualisée devient plus importante, pour remplacer la suite de PEEK par une boucle POUR.

Amélioration du programme BASIC.- Les programmes BASIC que nous avons écrit nous permettent de tester nos sous-programmes en code machine mais ils ont un inconvénient : il faut à chaque fois calculer la longueur du code machine pour placer le résultat à un endroit où il ne détruit pas la fin du sous-programme. Une astuce utilisée en langage machine depuis longtemps consiste à placer les données et les résultats *avant* le code machine. Ceci fonctionne tant qu'on connaît à l'avance la taille des données et des résultats, ce qui est souvent le cas.

Réécrivons donc notre programme de façon à placer le résultat (il n'y a pas de données dans notre exemple) aux octets 0 et 1 et le code à partir de l'octet 2 :

```
CLS
DIM PM%(6)
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 10
  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 B8,11,00
DATA 05,05,00
DATA 2E,A3,00,00
DATA CB
```

autrement dit nous avons changés quatre lignes : celle avec un POKE en ajoutant un décalage de 2, celle avec le CALL ABSOLUTE en ajoutant également ce décalage de 2, la dernière instruction PRINT puisque maintenant le résultat se trouve dans l'élément du tableau d'index 0 et, enfin, la troisième ligne de DATA puisqu'on place maintenant le résultat à l'adresse 0h.

L'exécution se comporte comme prévu en affichant 22 :

```
0      0     184     17      0
5      5     0       46    163
0      0     203      0      0

22     0     184     17      0
5      5     0       46    163
0      0     203      0      0

22
```

15.3.1.2 Autres formes

Code machine.- Il existe deux autres formes de l'addition :

- l'addition du contenu d'une case mémoire ou d'un registre avec le contenu d'un registre, codée en langage machine sur deux ou quatre octets suivant le cas :

```
0000 00dw | mod reg r/m |        |
```

- l'addition immédiate au contenu d'une case mémoire ou d'un registre, codée en langage machine sur trois ou quatre octets suivant le cas :

1000 00sw | mod 000 r/m | | donnée si $s = 0$ et $w = 1$

Exercice corrigé.- *Traduire l'instruction* :

```
ADD CH, BL
```

en langage machine.

Cette instruction se traduit par les deux octets représentés en binaire par :

```
0000 0010 | 11 101 011
```

soit &H02 &HEB en hexadécimal.

Exercice corrigé.- *Traduire l'instruction* :

```
ADD CX, 432h
```

en langage machine.

Cette instruction se traduit par les quatre octets représentés en binaire par :

```
1000 0001 | 00 001 110 | 0011 0010 | 0000 0100
```

soit &H81 &H0E &H32 &H04 en hexadécimal.

Exercice.- *À quelle opération correspond l'utilisation de l'accumulateur comme registre dans une addition implicite.*

15.3.1.3 Incidence sur les indicateurs

Chaque instruction arithmétique (et même, plus généralement, presque chaque instruction) effectue un rapport de statut (ou d'erreur) dans le registre des indicateurs.

Structure du registre des indicateurs.- Le registre des indicateurs est un registre seize bits. Chacun de ces bits est un **indicateur** (*flag*, c'est-à-dire drapeau, en anglais) à lui tout seul. En fait seuls neuf bits (sur les seize possibles) sont utilisés par le microprocesseur 8086/8088. Ces six bits sont divisés en deux groupes, les **indicateurs de contrôle** et les **indicateurs de statut**. Un indicateur est **levé** (en anglais *a flag is set*) s'il a la valeur 1.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	Tf	SF	ZF		AF		PF		CF

Chacun des six bits utilisés porte un nom en langage symbolique, rappelant sa fonction essentielle (que nous détaillerons au fur et à mesure de l'avancée du cours). Donnons ces noms en indiquant rapidement, pour l'instant, les rôles de chacun des indicateurs. On trouve dans l'octet de poids faible les 5 indicateurs du 8085 :

– Le bit 0 est l'**indicateur de retenue CF** (pour l'anglais *Carry Flag*). Il indique le dépassement de capacité. C'est le neuvième ou dix-septième bit d'une addition ou d'une soustraction de deux entiers de huit bits ou seize bits. Il sert aussi dans les opérations de rotation.
– Le bit 2 est levé lorsque le résultat d'une opération possède un nombre pair de bits égaux à 1, d'où son nom d'**indicateur de parité PF** (pour l'anglais *Parity Flag*).
– Le bit 6 est l'**indicateur de zéro ZF** (pour l'anglais *Zero Flag*). Il est levé lorsque le résultat d'une soustraction est nulle.
– Le bit 7 est l'**indicateur de signe SF** (pour l'anglais *Sign Flag*). Il est levé lorsque le résultat d'une opération est négatif, c'est-à-dire (comme nous le verrons) lorsque le bit de poids fort, huitième ou seizième, est égal à 1.
– Le bit 4 est l'**indicateur auxiliaire AF** (pour l'anglais *Auxiliary Flag*), ou de retenue intermédiaire. Il est levé lorsque la retenue se propage du quartet de poids faible vers le quartet de poids fort.

Ces 5 indicateurs sont complétés par 4 indicateurs nouveaux :

– Le bit 11 est l'**indicateur de dépassement OF** (pour l'anglais *Overflow Flag*) dont nous verrons son rôle dans l'arithmétique des entiers relatifs.
– Le bit 10 sert à choisir la direction lors de la copie d'un tableau d'octets (en ordre croissant ou décroissant), d'où son nom d'**indicateur de direction DF** (pour l'anglais *Direction Flag*).
– Le bit 9 est levé lorsque les interruptions externes sont permises, d'où son nom d'**indicateur d'interruption IF** (pour l'anglais *Interruption Flag*).
– Le bit 8 est levé pour l'exécution d'un programme pas-à-pas lors de la mise au point d'un logiciel, d'où son nom d'**indicateur de pas-à-pas TF** (pour l'anglais *Trap Flag*).

Cas d'une addition.- L'exécution d'une addition sur les entiers naturels a les incidences prévisibles suivantes sur les indicateurs `ZF`, `PF`, `AF` et `CF`.

15.3.2 Addition sur plusieurs mots

Principe.- Si au moins l'un des entiers à additionner est supérieur à 65 535, on peut utiliser plusieurs mots pour représenter ces nombres. Si on veut, par exemple, additionner `99FFEh` et `88003h`, il suffit d'additionner, d'une part, `9FFEh` et `8003h` et, d'autre part, `9h` et `8h`. Mais il y a une retenue pour l'addition des deux mots de faible poids dont il faut tenir compte dans l'addition des deux mots de poids fort.

Il existe une nouvelle instruction qui nous permet de manipuler la retenue éventuelle.

Addition avec retenue.- L'instruction :

```
ADC destination, source
```

(pour l'anglais *ADdition with Carry*) place dans `destination` la somme du contenu de `destination`, du contenu de `source` et de l'indicateur de retenue `CF` :

```
destination := destination + source + CF
```

Langage machine.- Cette instruction possède trois codages selon la nature des opérandes :

– Mémoire ou registre avec registre :

```
| 0001 00dw | mod reg r/m |        |        |
```

– Addition immédiate dans l'accumulateur (AX ou AL) :

`| 0001 010w | donnée | donnée si` $w = 1$ `|`

– Addition immédiate au contenu d'une case mémoire ou d'un registre :

`| 1000 00sw | mod 010 r/m | donnée | donnée si` $s = 0$ `et` $w = 1$ `|`

Exemple.- Écrivons un programme permettant de calculer la somme de 99FFEh et 88003h.

Reprenons l'astuce de notre dernière forme de programme en plaçant les données 99FFEh et 88003h au début du segment de mémoire réservé par le tableau puis le code machine, permettant d'additionner ces deux données et de placer le résultat à la place du premier opérande, après.

Nous pourrions utiliser le transfert immédiat vers une case mémoire pour placer les données en code machine. Pour simplifier, nous utiliserons BASIC : `PM%(0) = &H9FFE, PM%(1) = &H9, PM%(2) = &H8003, PM%(3) = &H8`. Le contenu du début du segment est alors :

0	1	2	3	4	5	6	7
FE	9F	09	00	03	80	08	00

Pour effectuer l'addition voulue, récupérons le mot commençant à l'adresse 0 dans AX. Ajoutons-lui le mot commençant à l'adresse 4 (l'indicateur CF devrait donc être levé). Plaçons le résultat dans le mot commençant à l'adresse 0. Heureusement les instructions de transfert ne changent pas le registre des indicateurs, donc la valeur de CF n'est pas perdue. Récupérons le mot commençant à l'adresse 2 dans AX. Ajoutons-lui, en tenant compte de la retenue éventuelle, le mot commençant à l'adresse 6. Plaçons le résultat dans le mot commençant à l'adresse 2. Le sous-programme machine est donc :

```
MOV AX, CS :[0h]
ADD AX, CS :[4h]
MOV CS :[0h], AX
MOV AX, CS :[2h]
ADC AX, CS :[6h]
MOV CS :[2h], AX
RETF
```

La première instruction, transfert d'une case mémoire vers l'accumulateur se code :

```
2E | 1010 0001 | 0 | 0 |
```

soit &H2E, &HA1, &H00, &H00. La seconde instruction, addition à l'accumulateur du contenu d'une case mémoire, se code :

```
2E | 0000 0011 | 00 000 110 | 04 | 00 |
```

soit &H2E, &H03, &H06, &H04, &H00.Nous avons l'habitude de coder la troisième instruction, transfert du contenu de l'accumulateur vers une case mémoire, ce qui donne &H2E, &HA3, &H00, &H00. La quatrième instruction est l'analogue de la première, ce qui donne &H2E, &HA1, &H02, &H00. Nous venons de voir ci-dessus, et c'est la seule instruction vraiment nouvelle, comment coder la cinquième instruction, une addition avec retenue :

```
2E | 0001 0011 | 00 000 110 | 06 | 00 |
```

soit &H2E, &H13, &H06, &H06, &H00. La sixième instruction se code come d'habitude &H2E, &HA3, &H02, &H00. La dernière instruction se code comme d'habitude &HCB.

Écrivons maintenant le programme QBasic permettant de tester ce sous-programme machine. Il y a 8 octets de données et 27 octets de code. Nous avons donc besoin de réserver de la place pour 35 octets, soit 18 mots :

```
CLS
```

```
DIM PM%(18)
SegPM% = VARSEG(PM%(0))
OffPM% = VARPTR(PM%(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du debut du tableau par les donnees
PM%(0) = &H9FFE
PM%(1) = &H9
PM%(2) = &H8003
PM%(3) = &H8
'Initialisation de la suite du tableau par le code machine
FOR I% = 0 TO 26
  READ Octet$
  POKE OffPM% + 8 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 12
  PRINT HEX$(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 HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
'Affichage du resultat
PRINT HEX$(PM%(0));HEX$(PM%(1))
END
DATA 2E,A1,00,00
DATA 2E,03,06,04,00
DATA 2E,A3,00,00
DATA 2E,A1,02,00
DATA 2E,13,06,06,00
DATA 2E,A3,02,00
DATA CB
```

Remarquons que nous affichons maintenant le contenu des cases mémoires et du résultat en hexadécimal grâce à la fonction HEX$(), ce qui facilite la vérification.

L'exécution se comporte comme prévu en affichant 122001h, ce qui est résultat comme on pourra le vérifier à la main (ou à l'aide d'une calculette hexadécimale) :

```
FE    9F   9     0    3
80    8    0     2E   A1
0     0    2E    3    6

1     20   12    0    3
80    8    0     2E   A1
0     0    2E    3    6
```

```
122001
```

Amélioration du programme BASIC.- On peut ici améliorer le programme BASIC en utilisant le type `long` au lieu du type `integer` :

```
CLS
DIM PM&(9)
SegPM% = VARSEG(PM&(0))
OffPM% = VARPTR(PM&(0))
'Passage au segment du code machine
DEF SEG = SegPM%
'Initialisation du debut du tableau par les donnees
PM&(0) = &H99FFE
PM&(1) = &H88003
'Initialisation de la suite du tableau par le code machine
FOR I% = 0 TO 26
  READ Octet$
  POKE OffPM% + 8 + I%, VAL("&H" + Octet$)
NEXT I%
'Visualisation du contenu du debut de ce segment avant execution
FOR I% = 0 TO 12
  PRINT HEX$(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 HEX$(PEEK(I%)),
NEXT I%
'Retour au segment de depart
DEF SEG
'Affichage du resultat
PRINT HEX$(PM&(0))
END
DATA 2E,A1,00,00
DATA 2E,03,06,04,00
DATA 2E,A3,00,00
DATA 2E,A1,02,00
DATA 2E,13,06,06,00
DATA 2E,A3,02,00
DATA CB
```

qui donne le même résultat.

15.4 Soustraction

Comme l'addition, la soustraction peut s'effectuer sur un octet ou un mot de deux octets et, également, sur plusieurs mots (pour les « grands » entiers), mais pas en une seule fois.

15.4.1 Soustraction sur un octet ou un mot

Langage symbolique.- L'instruction est :

```
SUB destination, source
```

Sémantique.- Il ne s'agit pas d'une soustraction complète dans le sens où le résultat serait placé dans un emplacement autre que l'un des deux opérandes :

```
resultat := destination - source
```

En effet, le résultat est placé dans le premier opérande :

```
destination := destination - source
```

Langage machine.- Comme pour l'addition, il existe trois formes de soustraction ; elle peut avoir lieu entre registres, registre et mémoire, registre ou mémoire et donnée immédiate :

– La soustraction immédiate avec l'accumulateur (AL ou AX) est codée sur deux ou trois octets suivant le cas :

```
| 0010 110w | donnée | donnée si w = 1 |
```

concernant `AX` si $w = 1$ et `AL` si $w = 0$.

– La soustraction du contenu d'une case mémoire ou d'un registre avec le contenu d'un registre, est codée en langage machine sur deux ou quatre octets suivant le cas :

```
| 0010 10dw | mod reg r/m |         |         |
```

– La soustraction immédiate au contenu d'une case mémoire ou d'un registre est codée en langage machine sur trois ou quatre octets suivant le cas :

| 1000 00sw | mod 101 r/m | | donnée si $s = 0$ et $w = 1$ |

Exercice corrigé.- *Traduire l'instruction :*

```
SUB AL, 4h
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire par :

```
0010 1100 | 0000 0100
```

soit &H2C &H04 en hexadécimal.

Exercice corrigé.- *Traduire l'instruction :*

```
SUB AX, 660h
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les trois octets représentés en binaire par :

`| 0010 1101 | 0110 00000 | 0000 0110 |`

soit &H2D &H60 &H06 en hexadécimal.

Exercice corrigé.- *Traduire l'instruction* :

`SUB CH, DL`

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire par :

`| 0010 1010 | 11 101 010 |`

soit &H2A &HEA en hexadécimal.

Exercice corrigé.- *Traduire l'instruction* :

`SUB SI, 312`

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les quatre octets représentés en binaire par :

`| 1000 0001 | 11 101 110 | 0011 1000 | 0000 0001 |`

soit &H81 &HEE &38 &01 en hexadécimal.

Test d'un programme.- Écrivons un sous-programme machine qui place 17 dans l'accumulateur `AX`, soustrait 5 à celui-ci puis place le résultat dans l'emplacement mémoire &H000C du segment de code. En langage symbolique nous avons :

```
MOV AX, 11h
SUB AX, 5h
MOV CS :[0Ch], AX
RETF
```

Traduisons-le en langage machine :

```
&HB8 &H11 &H00
&H2D &H05 &H00
&H2E &HA3 &H0C &H00
&HCB
```

Pour tester ce sous-programme en langage machine, il suffit, dans le programme QBasic analogue pour tester l'addition, de remplacer la deuxième ligne de DATA :

```
DATA 05,05,00
```

par :

```
DATA 2D,05,00
```

dont l'affichage de 12 :

```
184   17     0    45     5
0     46   163    12     0
203    0     0     0     0
```

```
184   17    0    5    5
0     46  163   12    0
203    0   12    0    0

12
```

de deux façons différentes, nous laisse penser que le sous-programme en langage machine s'est correctement déroulé.

Incidence sur les indicateurs.- Lors d'une soustraction :

- L'indicateur de retenue CF est levé lorsqu'il y a report, c'est-à-dire lorsque le premier opérande est plus petit que le second.
- L'indicateur de parité PF est levé si le résultat contient un nombre pair de bits égaux à 1.
- L'indicateur de zéro ZF est levé lorsque le résultat de la soustraction est nulle, autrement dit lorsque les deux opérandes sont égaux.
- L'indicateur de signe SF est affecté mais ne nous intéresse pas pour la soustraction sur les entiers naturels.
- L'indicateur auxiliaire AF est levé lorsqu'il y a report du quartet de poids faible vers le quartet de poids fort.
- L'indicateur de dépassement OF est affecté mais ne nous intéresse pas pour la soustraction sur les entiers naturels.

15.4.2 Cas d'une différence négative

Pour la sémantique de la soustraction, nous avons dit que le contenu du premier opérande est la différence des contenus du premier opérande et du deuxième opérande. Ceci n'a de sens que si le premier opérande est plus grand que le second opérande pour les entiers naturels. Sinon il y a un emprunt (*borrow* en anglais). Dans ce cas, la sémantique de la soustraction est : le contenu du premier opérande est égal à 2^8 dans le cas d'une soustraction d'octets, 2^{16} dans le cas d'une soustraction de mots, plus le contenu du premier opérande moins le contenu du deuxième opérande et l'indicateur de retenue `CF` du registre des indicateurs est égal à 1.

15.4.3 Soustraction sur plusieurs mots

Principe.- Comme pour l'addition, si au moins l'un des entiers à soustraire est supérieur à 65 535, on peut utiliser plusieurs mots pour représenter les opérandes. Si on veut, par exemple, soustraire `89003h` à `98FFEh`, il suffit de soustraire, d'une part, `9003h` à `8FFEh` et, d'autre part, `8h` à `9h`. Mais il y a un report lors de la soustraction des deux mots de faible poids dont il faut tenir compte dans la soustraction des deux mots de poids fort (on obtient 0 et non 1).

Comme pour l'addition, il existe une nouvelle instruction qui nous permet de manipuler le report éventuel.

Soustraction avec emprunt.- L'instruction :

```
SBB destination, source
```

(pour l'anglais *SuBstraction with Borrow*) place dans `destination` la différence du contenu de `destination`, d'avec la somme du contenu de `source` et de l'indicateur de retenue `CF` :

```
destination := destination - (source + CF)
```

Langage machine.- Cette instruction possède trois codages selon la nature des opérandes :

– Mémoire ou registre avec registre :

`| 0001 10dw | mod reg r/m |        |        |`

– Soustraction avec emprunt immédiate dans l'accumulateur (AX ou AL) :

`| 0001 110w | donnée | donnée si` $w = 1$ `|`

– Soustraction avec emprunt immédiate au contenu d'une case mémoire ou d'un registre :

`| 1000 00sw | mod 011 r/m | donnée | donnée si` $s = 0$ `et` $w = 1$ `|`

Exemple.- Écrivons un programme permettant de calculer la différence de 98FFEh et de 89003h.

Adaptons le programme écrit pour l'addition de 99FFEh et de 88003h.

Nous utilisons les instructions BASIC pour placer les deux opérandes en données au début du tableau : `PM&(0) = &H98FFE, PM&(1) = &H89003`. Le contenu du début du segment est alors :

0	1	2	3	4	5	6	7
FE	8F	09	00	03	90	08	00

Pour effectuer la soustraction voulue, récupérons le mot commençant à l'adresse 0 dans AX. Soustrayons-lui le mot commençant à l'adresse 4 (l'indicateur CF devrait donc être levé). Plaçons le résultat dans le mot commençant à l'adresse 0. Récupérons le mot commençant à l'adresse 2 dans AX. Soustrayons-lui, en tenant compte de l'emprunt éventuel, le mot commençant à l'adresse 6. Plaçons le résultat dans le mot commençant à l'adresse 2. Le sous-programme machine est donc :

```
MOV AX, CS :[0h]
SUB AX, CS :[4h]
MOV CS :[0h], AX
MOV AX, CS :[2h]
SBB AX, CS :[6h]
MOV CS :[2h], AX
RETF
```

Par rapport au programme que nous adaptons, seules les deuxièmes et cinquièmes instructions changent. Intéressons-nous à celles-ci. La seconde instruction, soustraction à l'accumulateur du contenu d'une case mémoire, se code :

```
2E | 0010 1011 | 00 000 110 | 04 | 00 |
```

soit &H2E, &H2B, &H06, &H04, &H00. La cinquième instruction, une soustraction avec emprunt, se code :

```
2E | 0001 1011 | 00 000 110 | 06 | 00 |
```

soit &H2E, &H1B, &H06, &H06, &H00.

Dans l'adaptation du programme QBasic permettant de tester ce sous-programme machine, il suffit donc de remplacer l'initialisation des deux premiers éléments du tableau ainsi que la seconde et la cinquième lignes de DATA.

L'exécution se comporte comme prévu en affichant FFFBh, ce qui est résultat comme on pourra le vérifier à la main (ou à l'aide d'une calculette hexadécimale) :

```
FE    8F   9     0    3
90    8    0     2E   A1
0     0    2E    2B   6
```

FB	FF	0	0	3
90	8	0	2E	A1
0	0	2E	2B	6

FFFB

15.5 Multiplication

Langage symbolique.- L'instruction est :

`MUL A, source`

où `A` est l'un des accumulateurs `AX` ou `AL` et `source` une constante, un registre ou une case mémoire.

Sémantique.- Cette instruction permet de multiplier le contenu de l'accumulateur 8 bits AL (resp. 16 bits AX) par le contenu d'un registre ou d'une case mémoire 8 bits (resp. 16 bits). Le résultat, qui a une taille double, est placé dans AX (resp. AX et DX, ce dernier pour le mot de poids fort).

Langage machine.- Cette multiplication s'effectue grâce à une instruction codée sur deux ou quatre octets suivant le cas :

```
| 1111 011w | mod 100 r/m |        |        |
```

concernant `AX` si $w = 1$ et `AL` si $w = 0$.

Exercice corrigé.- *Traduire l'instruction* :

`MUL BL` (aussi écrite `MUL AL, BL`)

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire par :

```
| 1111 0110 | 11 100 011 |
```

soit &HF6 &HE3 en hexadécimal.

Exercice corrigé.- *Traduire l'instruction* :

`MUL BX` (aussi écrite `MUL AX, BX`)

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire par :

```
| 1111 0111 | 11 100 011 |
```

soit &HF7 &HE3 en hexadécimal.

Exercice corrigé.- *Traduire l'instruction* :

`MUL AX, [1FC4h]`

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les quatre octets représentés en binaire par :

```
| 1111 0111 |00 100 110 |1100 01000 | 0001 1111 |
```

soit &HF7 &H26 &HC4 &H1F en hexadécimal.

Test d'un programme.- Écrivons un sous-programme machine qui place &H1234 dans le registre BX, qui place &H23 dans l'accumulateur `AX`, puis place le résultat dans l'emplacement mémoire (de quatre octets) &H0000 du segment de code. En langage symbolique nous avons :

```
MOV BX, 1234h
MOV AX, 23h
MUL AX, BX
MOV CS :[00h], AX
MOV CS :[02h], DX
RETF
```

Pour la traduction en langage machine, il n'y a guère que la première et la cinquième instructions que nous n'avons jamais rencontrées. Pour la première, on a, en binaire :

```
| 1011 1 011 | 0001 0010 | 0011 0100 |
```

soit &HBB, &H12, &H34. Pour la cinquième, on a, en binaire, si on oublie le préfixe :

```
| 1000 10 0 1 | 00 010 110 | 0000 0000 | 0000 0010 |
```

soit &H89 &H16 &H02 &H00. Le code machine du sous-programme est donc :

```
&HBB &H34 &H12
&HB8 &H23 &H00
&HF7 &HE3
&H2E &HA3 &H00 &H00
&H2E &H89 &H16 &H02 &H00
&HCB
```

dont la longueur est 18 octets.

Écrivons enfin le programme QBasic suivant pour le tester :

```
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 17
  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 12
  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 12
  PRINT PEEK(I%),
NEXT I%
'Retour au segment de depart
```

```
DEF SEG
'Affichage du resultat
PRINT PM&(0)
END
DATA BB,34,12
DATA B8,23,00
DATA F7,E3
DATA 2E,A3,00,00
DATA 2E,89,16,02,00
DATA CB
```

L'exécution se comporte comme prévu en affichant &H27D1C, dont on pourra vérifier le résultat en utilisant une calculette hexadécimale :

```
0    0    0     0    BB
34   12   B8    23   0
F7   E3   2E    A3   0

1C   7D   2     0    BB
34   12   B8    23   0
F7   E3   2E    A3   0

27D1C
```

<u>Incidence sur les indicateurs</u>.- Si après une multiplication, si l'octet (resp. le mot) de poids fort est nul, les indicateurs CF et OF sont mis à zéro, sinon ils sont mis à 1. Les indicateurs AF, PF, SF et ZF sont dans un état indéfini.

15.6 Division euclidienne

<u>Introduction</u>.- La division n'est pas une opération totale sur l'ensemble $\mathbb{N}$ des entiers naturels. Rappelons cependant qu'il existe une autre sorte de division, la *division euclidienne*, qui est une opération totale donnant lieu à deux résultats. Pour un entier a et un entier non nul b, il existe un unique couple (q, r) d'entiers tels que :

$$a = b.q + r \;\; et \;\; 0 \leq r < b.$$

L'entier q est appelé le *quotient* (*exact*) et l'entier r le *reste* dans la *division euclidienne* de a par b.

<u>Langage symbolique</u>.- L'instruction est :

```
DIV source
```

où le dividende de 16 bits (resp. 32 bits) est placé dans l'accumulateur `AX` (resp. `AX` et `DX`, ce dernier contenant le mot de poids fort) et le diviseur dans `source`, qui est un registre ou une case mémoire.

<u>Sémantique</u>.- Cette instruction permet d'effectuer la division euclidienne du dividende par le diviseur, le quotient exact est placé dans AL (ou AX) et le reste dans AH (ou DX).

Langage machine.- Cette division euclidienne s'effectue grâce à l'instruction codée sur deux ou quatre octets suivant le cas :

```
| 1111 011w | mod 110 r/m |         |         |
```

Si $w = 0$, on a dividende = [AX], quotient = [AL] et reste = [AH].
Si $w = 1$, on a dividende = [DX,AX], quotient = [AX] et reste = [DX].

Exercice corrigé.- *Traduire l'instruction :*

```
DIV BL
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire par :

```
| 1111 0110 | 11 110 011 |
```

soit &HF6 &HF3 en hexadécimal.

Exercice corrigé.- *Traduire l'instruction :*

```
DIV BX
```

en langage machine.

D'après ce que nous venons de dire, cette instruction se traduit par les deux octets représentés en binaire par :

```
| 1111 0111 | 11 110 011 |
```

soit &HF7 &HF3 en hexadécimal.

Exercice corrigé.- *Traduire l'instruction :*

```
DIV byte [1FC4h]
```

en langage machine, c'est-à-dire dans le cas où le quotient est un octet et non un mot.

D'après ce que nous venons de dire, cette instruction se traduit par les quatre octets représentés en binaire par :

```
| 1111 0110 |00 110 110 |1100 01000 | 0001 1111 |
```

soit &HF6 &H36 &HC4 &H1F en hexadécimal.

Test d'un programme.- Écrivons un sous-programme machine qui place &H34 dans le registre BL, qui place &H1234 dans l'accumulateur `AX`, effectue la division euclidienne de [AX] par [BL] puis place le quotient dans l'emplacement mémoire (d'un octet) &H0000 du segment de code et le reste dans l'emplacement mémoire &H0001. En langage symbolique nous avons :

```
MOV BL, 34h
MOV AX, 1234h
DIV BL
MOV CS :[00h], AX
RETF
```

Pour la traduction en langage machine, il n'y a guère que la première que nous n'avons jamais rencontrée. On a, en binaire :

```
| 1011 0 011 | 0011 0100
```

soit &HB3, &H34. Le code machine du sous-programme est donc :

```
&HB3 &H34
&HB8 &H34 &H12
```

```
   &HF6 &HF3
   &H2E &HA3 &H00 &H00
   &HCB
```

dont la longueur est 12 octets.

Écrivons enfin le programme QBasic suivant pour le tester :

```
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 11
  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 B3,34
DATA B8,34,12
DATA F6,F3
DATA 2E,A3,00,00
DATA CB
```

L'exécution se comporte comme prévu en affichant &H59 pour le quotient et &H20 pour le reste, dont on pourra vérifier le résultat en utilisant une calculette hexadécimale :

```
0     0     B3     34    B8
34    12    F6     F3    2E
A3    0     0      CB    0

59    20    B3     34    B8
34    12    F6     F3    2E
A3    0     0      CB    0

2059
```

Incidence sur les indicateurs.- Tous les indicateurs sont affectés mais sans signification particulière.

15.7 Historique

15.7.1 Les tous premiers programmes (en langage machine)

15.7.1.1 Première exécution d'un programme (21 juin 1948)

Le premier programme exécuté sur un calculateur à programme enregistré est écrit par Tom Kilburn et exécuté le lundi 21 juin 1948 sur le prototype d'ordinateur Mark I de l'université de Manchester en Angleterre. On dit que c'est le premier et dernier programme écrit par Kilburn. Le programme trouva le plus grand facteur premier d'un entier en moins d'une minute. La deuxième exécution, pour un entier différent, prit 2 minutes et 52 secondes. Williams décrit l'exécution de la façon suivante, la mémoire utilisée est composée de tubes cathodiques :

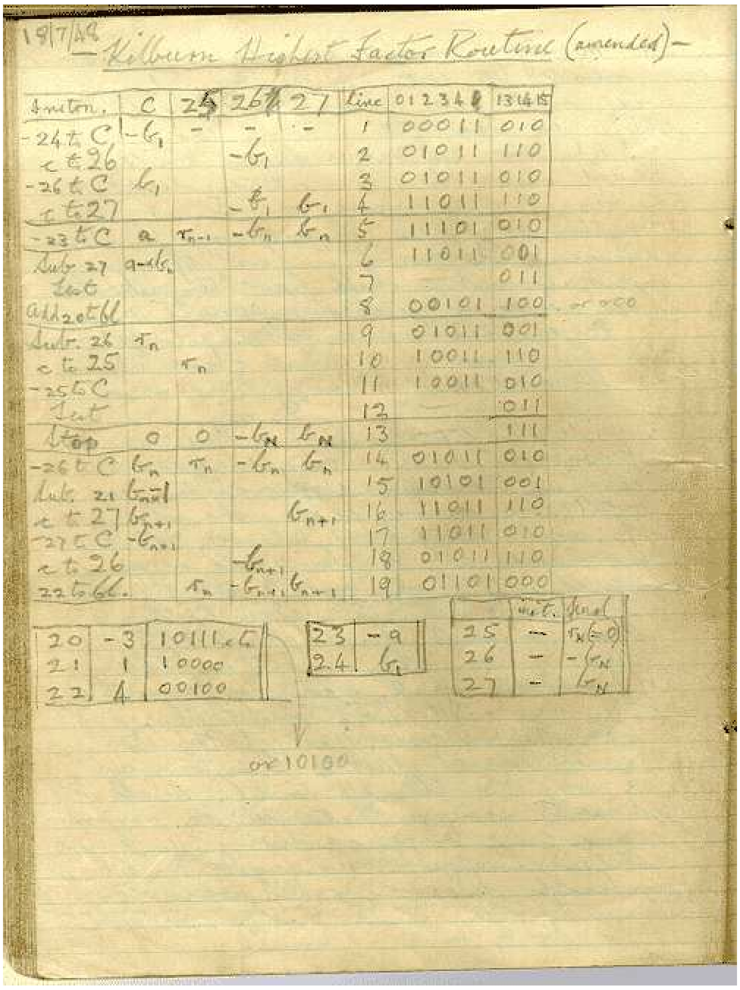

Fig. 15.2 – Le premier programme (18 juillet 1948)

> *Lorsque* [la machine fut] *construite, un programme fut laborieusement inséré et le bouton de départ enfoncé. Immédiatement les points sur le tube d'affichage entrèrent dans une danse folle. Dans les essais précédents c'était une danse macabre ne condui-*

sant à aucun résultat utile [...]. *Mais un jour cela s'arrêta et là, brillant à la place espérée, fut la réponse attendue.*

[Cam-80], pp. 134.

Malheureusement personne ne pense à conserver ce programme jusqu'à ce que Geoffrey C. Tootill en écrive une version corrigée dans son cahier de notes un mois plus tard, le 18 juillet 1948. L'original a été perdu mais il en reste une copie (figure 15.2).

Ce programme est commenté dans le chapitre 15 de [Lav-80], dans [Cam-80] et dans [SB-98].

Memory position.	Order.	
0	T 0 S	
1	H 2 S	Set multiplier.
2	T 0 S	Transfer control to 6.
3	E 6 S	
4	P 1 S	Pseudo-orders.
5	5 S	
6	T 0 S	
7	I 0 S	Input function digits; shift to correct position in 0.
8	A 0 S	$x_1 = 0$
9	R 16 S	
10	T 0 L	
1	I 2 S	
2	A 2 S	
3	S 5 S	Input next symbol. Test for digit or discriminant.
4	E 21 S	
5	T 3 S	
6	V 1 S	
7	L 8 S	$10\,x_1 + b$ to 1.
8	A 2 S	
9	T 1 S	
20	E 11 S	
1	R 4 S	
2	A 1 S	Length discriminant digit to acc.
3	L 0 L	
4	A 0 S	Form order and transfer to n.
5	(T 31) S	
6	A 25 S	
7	A 4 S	n+1 to n.
8	U 25 S	
9	S 31 S	
30	G 6 S	Test for start of programme.

Fig. 15.3 – Première routine (Worsley 1950)

15.7.1.2 Le premier programme publié (6 mai 1949)

Nous avons vu ci-dessus qu'une conférence s'est tenue en juin 1949 à l'université de Cambridge lors de l'inauguration de l'EDSAC. Nous avons vu comment Maurice Wilkes y décrit cet ordinateur ainsi que le langage symbolique associé au langage machine. Cet article est accompagné d'une description de l'exécution de programmes de démonstration (établissement d'une table de carrés et d'une table de nombres premiers), avec les listings du programme et les copies des sorties produites par la machine :

Démonstration de l'E.D.S.A.C.

(W. Renwick)
Rapport préparé par mademoiselle B. H. Worsley

31	T	123 S	As required by initial input.	
2	E	84 S	Jump to 84.	
3	(P	S)	Use to keep count of subtractions.	
4	(P	S)	Power of 10 being subtracted.	
5	P	10000 S	For use in the decimal binary conversion	Pseudo-orders
6	P	1000 S		
7	P	100 S		
8	P	10 S		
9	P	1 S		
40	Q	S		
1	π	S	Figures	
2	A	40 S		
3	φ	S	Space	
4	Δ	S	Line-feed	
5	ϑ	S	Car. return	
6	O	43 S		
7	O	33 S		
8	(P	S)	Becomes number to be printed	
→ 9	A	46 S	Put O 43 S in S(65)	
50	T	65 S		
1	T	129 S	Clear S(129).	
2	(A	35 S)	Put power of 10 in S(34).	
3	T	34 S		
4	E	61 S	Jump to 61	
→ 5	T	48 S		
6	A	47 S		
57	T	65 S	To control printing	
8	A	33 S		
9	A	40 S		
60	T	33 S		
→ 1	A	48 S	Print contents of S(48).	
2	S	34 S		
3	E	55 S		
4	A	34 S		
5	(P	S)		
6	T	48 S		
7	T	33 S		
8	A	52 S		
9	A	4 S		
70	U	52 S		
1	S	42 S		
2	G	51 S		
3	A	117 S		
4	T	52 S		

5	(P	S)	End print
6	(P	S)	Becomes x.
7	(P	S)	Becomes x^2.
8	(P	S)	Becomes x^2.
9	(P	S)	Becomes Δx^2.
80	E	110 S	
1	E	118 S	
2	P	100 S	
83	E	95 S	
4	O	41 S	Set on print figures
↗ 5	T	129 S	Clear 129.
6	O	44 S	
7	O	45 S	$x+1$ to S(76) and S(48)
8	A	76 S	
9	A	4 S	
90	U	76 S	
1	T	48 S	
2	A	83 S	Set switch Z.
3	T	75 S	
4	E	49 S	
5	O	43 S	Double space.
6	O	43 S	
7	H	76 S	$x^2 \cdot 2^{15}$ to S(77).
8	V	76 S	
9	L	64 S	
100	L	32 S	
1	U	77 S	
2	S	78 S	Δx^2 to S(79)
3	T	79 S	
4	A	77 S	
5	U	78 S	
6	T	48 S	x^2 to S(49) and print.
107	A	80 S	
8	T	75 S	
9	E	49 S	
110	O	43 S	Double space.
1	O	43 S	
2	A	79 S	
3	T	48 S	Δx^2 to S(48) and print.
4	A	81 S	
5	T	75 S	
6	E	49 S	
7	A	35 S	Test for finish.
8	A	76 S	
9	S	82 S	
120	G	85 S	
1	O	41 S	
2	Z	S	

FIG. 15.4 – Deuxième routine (Worsley 1950)

Lors du premier jour de la conférence, une démonstration du nouveau calculateur de Cambridge, l'E.D.S.A.C., fut effectuée. Lors de cette démonstration, des tables de carrés et de nombres premiers furent imprimées. Ces deux problèmes ont servi comme routines de test pour la machine ; ils sont détaillés ici à titre d'enregistrement et non en tant qu'exemples de programmation élégante. On n'essaiera pas d'expliquer tous les trucs et outils disponibles au programmeur. Cependant on espère que les notes qui *suivent, les organigrammes (*flow diagrams*) et les annotations rendront intelligibles les routines utilisées lors de la démonstration.*

Description de la démonstration

La séquence suivante d'événements a été observée :

1. Le ruban du téléscripteur, perforé suivant les ordres (ii) ou (iii) (voir ci-dessous), fut mis en place sur le lecteur de ruban perforé et on a appuyé sur le bouton de démarrage. Le reste de l'opération s'est alors effectué sans intervention de la part de l'opérateur.

```
   31 T  107 S
    2 O   92 S   Figures
→   3 O   93 S   Line feed
    4 O   94 S   Carriage
                 return
    5 S    5 S   Set p
→   6 T    6 S
    7 O   95 S   Double
    8 O   95 S   space
→   9 T    7 S
   40 A   96 S
    1 R    4 S
→   2 S   97 S
    3 E   42 S   Test
    4 L    4 S   whether
→   5 A   97 S   m a factor
    6 G   45 S   of n.
    7 S   98 S
    8 G  100 S
    9 T    7 S
   50 A   97 S
    1 A    4 S   m + 2 to m
    2 T   97 S
    3 H   97 S
    4 N   97 S   if m > √n
    5 L   64 S   stop testing
    6 L   64 S
```

```
   57 A   96 S
    8 E   39 S
    9 T    7 S
   60 A   96 S
    1 U    1 S
    2 A    4 S
    3 T   96 S
    4 A   99 S   go to next
    5 T   97 S   factor
    6 S   88 S
→   7 T    7 S
    8 H   91 S
    9 A    1 S
   70 E   72 S
→   1 V   91 S
→   2 S   89 S   If n Prime,
    3 E   71 S   transfer →
    4 A   89 S   to S(O).
    5 T    L
    6 O    S
    7 H   90 S
    8 V    1 S
    9 L    4 S   Print C(O)
   80 T    L
    1 A    7 S
    2 A   98 S
    3 G   67 S
```

```
   84 A    6 S
    5 A   98 S   Test p
    6 G   36 S   If the 5th no.
                 printed,
    7 E   33 S   linefeed,
                 carriage
                 return.
    8 P    2 S ≡ 4
    9 P  500 S
   90 J      S ≡ 8/10
    1 P   16 S
    2 π      S
    3 ϑ      S   Pseudo-
    4 Δ      S   orders
    5 φ      S
    6 (P   2 L)n
    7 (P   1 L)m
    8 P      L ≡ 1
    9 P    1 L ≡ 3
  100 T    7 S
    1 A   99 S
    2 T   97 S
    3 A    4 S   if n not a
    4 A   96 S   Prime.
    5 T   96 S
    6 E   39 S
```

FIG. 15.5 – Troisième routine (Worsley 1950)

2. Les ordres initiaux (i) (voir ci-dessous) ont été alors directement entrés dans les 31 premières positions de stockage court grâce à l'action des unisélecteurs. Ces ordres, utilisés pour chaque problème, sont pré-câblés sous leur forme binaire sur les unisélecteurs, à partir desquels ils sont transferrés automatiquement à leurs emplacements de stockage appropriés. Cette opération a pris à peu près 5 secondes et fut accompagnée d'une série de sons de clics à la cadence d'à peu près 6 par seconde.

3. Le lecteur de ruban perforé vint alors en action de façon à ce que les ordres (ii) ou (iii) soient synthétisés sous leur forme binaire et placés dans les positions de stockage commençant à la position 31. Cette action était sous le contrôle des ordres initiaux (i), utilisant ainsi la machine elle-même pour effectuer les conversions nécessaires. L'entrée se fit à la cadence d'un symbole par 150 millisecondes ou, en moyenne, d'un ordre et demi par seconde, un clic étant entendu pour chaque symbole.

4. Finalement on vit le téléscripteur commencer à opérer, le calcul et les sorties étant sous le contrôle des ordres stockés en interne (ii) ou (iii). Le temps entre la sortie de deux nombres premiers consécutifs devint de plus en plus appréciable à mesure que les nombres testés devinrent de plus en plus grands. L'impression des carrés fut à peu près aussi rapide que s'il ne s'agissait que d'écriture, un symbole toutes les 150 millisecondes.

Pour l'opération dans son intégralité, les contenus des conteneurs suivants furent affichés sur l'écran cathodique sous la forme binaire utilisée par la machine :

1. Les 16 mots (32 nombres courts ou ordres) pour les emplacements mémoire désirés.

2. L'accumulateur.

3. Le conteneur de comptage.

4. Le multiplicateur.

5. Le multiplicande.

6. Le conteneur de séquence d'ordre.

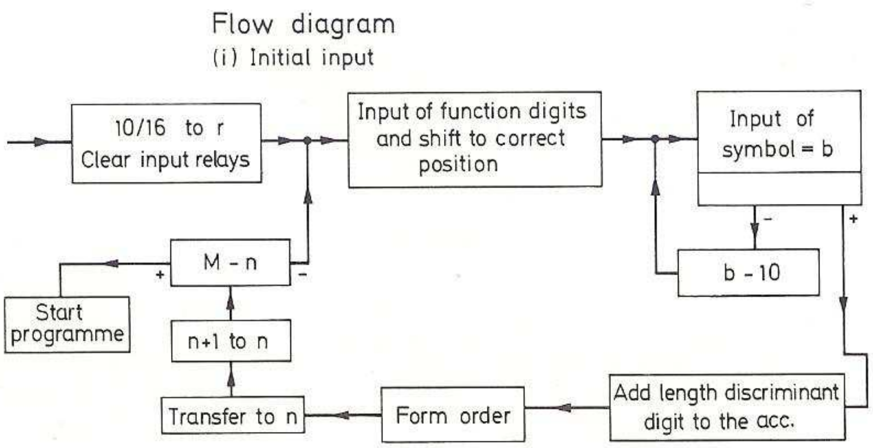

FIG. 15.6 – Premier organigramme (Worsley 1950)

Notes sur les routines

1. Le code des ordres de l'E.D.S.A.C. a été donné par monsieur M. V. Wilkes lors de sa description de la machine et n'a pas besoin d'être répété ici.

2. Les ordres d'entrée initiaux ont été conçus pour prendre les ordres sous la forme standard suivante :

(Lette de code de l'opération)(L'adresse sous forme décimale)

(lettre de code pour nombre long ou court)

Remarquons que ce n'est pas la seule possibilité mais qu'elle a été adoptée comme rendant le plus de service[1].

3. L'entrée initiale est seulement capable de synthétiser les symboles sous la forme standard des ordres. Cependant puisque, à l'intérieur de la machine, les ordres sont représentés comme des nombres, il est possible d'entrer certains nombres sous la forme d'ordres. Ceux-ci seront appelés des pseudo-ordres. Ainsi, par exemple, P5S représente le nombre 5×2^{-15} *dans la machine et peut aussi être interprété comme le nombre 5 multiplié par le facteur d'échelle* 2^{-15}. *Dans les problèmes de test il arrive que tous les nombres requis soient construits à partir des entiers simples de cette façon.*

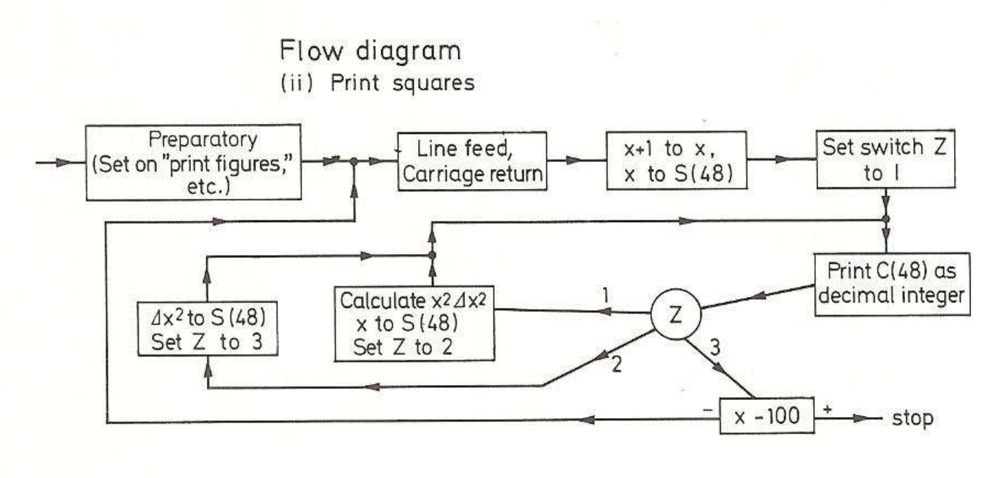

FIG. 15.7 – Second organigramme (Worsley 1950)

4. La disposition de l'impression est aussi sous le contrôle du programmeur. Ainsi, par exemple, en utilisant l'ordre « 0nS », où l'emplacement de stockage « n » contient « ΔS *», il peut faire effectuer un passage à la ligne au téléscripteur. Une attention toute particulière doit être apportée à la suppression des zéros sans signification si on le désire.*

5. Dans ces routines, l'emplacement de stockage est listé pour une référence extérieure. Seuls les ordres sont perforés sur le ruban d'entrée. Un ruban de téléscripteur à cinq trous standard est utilisé, avec un code légèrement modifié.

[1] Ceci est précisé dans la partie de l'article de WILKES que nous n'avons pas encore traduite :

> *Les ordres initiaux actuellement utilisés dans l'EDSAC mettent en place les ordres pris sur le ruban, perforés sous la forme suivante. Premièrement une lettre indiquant la fonction est perforée, ensuite la partie numérique de l'ordre, sous forme décimale, et enfin la lettre L ou S indiquant, respectivement, que l'ordre fait référence à un nombre long ou court. Sous le contrôle des ordres initiaux, la machine convertit la partie numérique de l'ordre sous forme binaire et assemble l'ordre avec les chiffres de la fonction et les chiffres numériques à leurs positions relatives correcte.*

[WR-50], pp. 420–421

6. Les flèches indiquent un point d'entrée à partir d'une autre position dans la routine. Normalement les ordres sont obéis consécutivement.

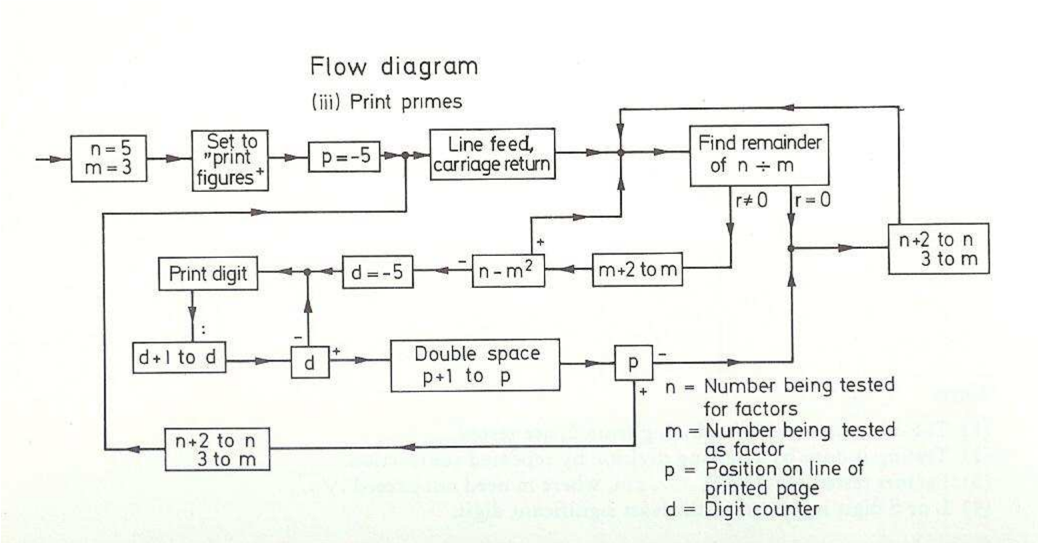

FIG. 15.8 – Troisième organigramme (Worsley 1950)

7. Les quantités entre parenthèses sont changées au cours de l'exécution des ordres.

Notation. [pour les commentaires]

S(n) désigne l'emplacement de stockage n.

C(n) désigne le contenu de l'emplacement de stockage n.

A désigne le contenu de l'accumulateur.

R désigne le contenu du registre multiplicateur.

n = 0. 1. 2. ....1 023, c'est-à-dire un emplacement de stockage court.

a to b signifie remplacer b par a.

Note : [relative à la routine (i)]

Le premier ordre perforé sur le ruban pour toute routine doit être T n S, le dernier ordre à entrer étant à l'emplacement n-1. Le contrôle est alors automatiquement transféré au début de la routine après que le dernier ordre ait été entré par la routine d'entrée initiale ci-dessus.

Note : [relative à la routine (iii)]

(1) Les nombres impairs, n, en commençant par 5, sont testés.

(2) Le test est effectué en effectuant la division par soustractions répétées.

(3) Les facteurs testés sont 3, 5, 7, ..., m, où m n'excède pas $\sqrt{n}$.

(4) Le chiffre L ou S est traité comme le chiffre de poids faible.

[Wor-50]

Le code utilisé est celui des téléscripteurs de l'époque, c'est-à-dire le code Baudot binaire à cinq bits, développé par Émile BAUDOT en 1874, appelé également code télégraphique ou alphabet international (AI) n^o 1.

1	1	1
2	4	3
3	9	5
4	16	7
5	25	9
6	36	11
7	49	13
8	64	15
9	81	17
10	100	19
11	121	21
12	144	23
13	169	25
14	196	27
15	225	29
16	256	31
17	289	33
18	324	35
19	361	37
20	400	39
21	441	41
22	484	43
23	529	45
24	576	47
25	625	49
26	676	51
27	729	53
28	784	55
29	841	57
30	900	59
31	961	61
32	1024	63

Left: Table of Squares of the natural numbers, together with the arguments and first differences.

FIG. 15.9 – Table des carrés (Worsley 1950)

Commentaires sur la première routine

Les trois programmes restent un peu difficiles à comprendre. Commentons-les ligne à ligne :

0. L'instruction numéro 0 a pour but d'initialiser l'accumulateur à 0.

1. L'instruction 1 initialise le multiplicateur à 2.

2. L'instruction 2 réinitialise l'emplacement mémoire 0 à 0 (au lieu du « T 0 S » fourni initialement par les unisélecteurs) et réinitialise l'accumulateur à zéro.

3. Puisque le contenu de l'accumulateur est zéro, on doit aller à l'instruction 6.

4. L'instruction 4 est une pseudo-instruction, ce que l'on peut, d'une part, déduire du fait qu'on ne l'atteint jamais en partant de l'instruction 0 et, d'autre part, puisque c'est indiqué en commentaire. Il s'agit de placer une constante entière à cet emplacement mémoire.

La façon de décoder n'ayant été précisée ni dans [WR-50] ni dans [Wor-50], il nous manque des informations pour dire de quelle constante il s'agit.

5. Même commentaire que pour l'instruction 4.

6. L'accumulateur est réinitialisé à 0. Ceci n'a pas d'importance la première fois qu'on passe par cette instruction puisque l'accumulateur a déjà la valeur nulle mais en a les fois suivantes, lorsqu'on revient à cette instruction à partir de l'instruction 30.

7. On lit la première partie de l'instruction suivante inscrite sur le ruban perforé, c'est-à-dire le code opération, et on place les cinq chiffres binaire lus en position de poids faible à l'emplacement mémoire 0.

Reproduction of Actual Tables Printed by EDSAC in the Demonstration

0005	0007	0011	0013	0017	0019	0023	0029	0031	0037
0041	0043	0047	0053	0059	0061	0067	0071	0073	0079
0083	0089	0097	0101	0103	0107	0109	0113	0127	0131
0137	0139	0149	0151	0157	0163	0167	0173	0179	0181
0191	0193	0197	0199	0211	0223	0227	0229	0233	0239
0241	0251	0257	0263	0269	0271	0277	0281	0283	0293
0307	0311	0313	0317	0331	0337	0347	0349	0353	0359
0367	0373	0379	0383	0389	0397	0401	0409	0419	0421
0431	0433	0439	0443	0449	0457	0461	0463	0467	0479
0487	0491	0499	0503	0509	0521	0523	0541	0547	0557
0563	0569	0571	0577	0587	0593	0599	0601	0607	0613
0617	0619	0631	0641	0643	0647	0653	0659	0661	0673
0677	0683	0691	0701	0709	0719	0727	0733	0739	0743
0751	0757	0761	0769	0773	0787	0797	0809	0811	0821
0823	0827	0829	0839	0853	0857	0859	0863	0877	0881
0883	0887	0907	0911	0919	0929	0937	0941	0947	0953
0967	0971	0977	0983	0991	0997	1009	1013	1019	1021

Fig. 15.10 – Table des nombres premiers (Worsley 1950)

La façon dont les instructions sont écrites sur le ruban perforé n'est peut-être pas clair. Une instruction occupe de 1 à 6 lignes de ruban perforé, c'est-à-dire de 1 à 6 caractères : le premier caractère est le code opération (représenté par une des lettres A, S, H, V, N, T, C, R, L, E, G, I, O, F, X, Y ou Z), suivi de 1 à 4 caractères représentant la valeur numérique en décimal (de 0 à 1 023) et se terminant par le *discriminateur* 'S' ou 'L'.

8. On l'ajoute à l'accumulateur, ce qui revient à la transférer tout simplement.

9. On multiplie l'accumulateur par 2^6, c'est-à-dire qu'on décale le contenu de l'accumulateur d'un caractère à droite, ce qui permet en plus d'initialiser la valeur numérique x_1 de l'instruction à 0.

10. On transfère ce qui vient d'être obtenu à l'emplacement mémoire 0 et on réinitialise l'accumulateur à 0.

11. On lit un chiffre décimal de la valeur numérique de l'instruction sur le ruban d'entrée et on le place à l'emplacement mémoire 2.

12. On l'ajoute à l'accumulateur, ce qui revient ici également tout simplement à un transfert.

13. On lui soustrait la constante située à l'emplacement mémoire 5, ce qui va permettre de déterminer s'il s'agit d'un chiffre décimal ou d'un discriminant.

14. S'il s'agit d'un discriminant, on en a terminé avec la lecture de la valeur numérique, on va à l'instruction 21 pour traiter la valeur du discriminant.

On remarquera qu'aucune vérification n'est faite sur la syntaxe de l'instruction qui a été écrite par le programmeur. Celui-ci n'a pas droit à l'erreur.

15. S'il s'agit d'un chiffre, on le transfère à l'emplacement mémoire 3 et on réinitialise l'accumulateur.

16. On multiplie le contenu de l'emplacement mémoire 1 par deux et on l'ajoute à l'accumulateur.

17. On multiplie le contenu de l'accumulateur par 2^5.

18. On lui ajoute le contenu de l'emplacement mémoire 2, c'est-à-dire, rappelons-le, le code opération.

19. On place le tout à l'emplacement mémoire 1 et on réinitialise l'accumulateur.

Les instructions 16 à 19 ont eu pour but de placer $10 \times x_1 + b$, où b est le chiffre lu, à l'emplacement mémoire 1, c'est-à-dire qu'on utilise la méthode de Hörner pour obtenir la valeur numérique.

20. On recommence à l'instruction 11 tant qu'il y a un chiffre décimal à lire.

Une fois de plus aucune vérification n'est effectuée. Il appartient au programmeur de ne pas dépasser 1 023.

21. Lorsqu'on lit un discriminant ('L' ou 'S'), on décale le nombre dans l'accumulateur de 4 places vers la droite.

22. On lui ajoute la valeur contenue à l'emplacement mémoire 1.

23. On décale le nombre dans l'accumulateur de deux places vers la gauche de façon à mettre l'instruction lue sous la forme voulue du langage machine.

24. On ajoute le nombre de l'emplacement mémoire 0 à l'accumulateur.

25. La première fois, on transfère la première instruction lue sur le téléscripteur et décodée à l'emplacement mémoire 31, c'est-à-dire tout de suite après le programme de démarrage.

26. On transfère le contenu de l'emplacement mémoire 25, c'est-à-dire l'instruction 25 qui vient d'être exécutée, à l'accumulateur.

27. On lui ajoute 4, c'est-à-dire qu'on incrémente la valeur numérique de cette instruction (la première fois on passera de 'T 31 S' à 'T 32 S').

28. On transfère cette nouvelle instruction à l'emplacement mémoire 25.

29. On soustrait le contenu de l'emplacement mémoire 31 à l'accumulateur.

30. Puisque la note relative à la routine (i) dit que la première instruction du programme doit être 'T n S', où $n - 1$ est le numéro de la dernière instruction du programme lorsqu'il sera placé en mémoire (en tenant compte des 31 instructions du programme de démarrage), si on n'a pas lu n instructions, on retourne à l'instruction 6, sinon on commence le programme par l'instruction 31 (qui ne sert pas à grand chose sinon à faire aller à l'instruction 32).

Commentaires sur la seconde routine

Commentons la seconde routine, en se référant à l'organigramme de la figure 15.7.

Étape préparatoire

31. Le 123 spécifie la taille du programme ; on remarquera d'ailleurs que les instructions de la deuxième routine sont numérotées de 31 à 122. De plus l'accumulateur prend la valeur nulle.

32. Puisque le contenu de l'accumulateur est zéro, on doit sauter à l'instruction 84.

33–83. Les instructions 33 à 83 sont des pseudo-instructions, contenant des constantes utiles pour la suite, sur lesquelles nous reviendrons en temps utile.

84. Imprime le caractère en attente sur le téléscripteur et met en attente sur le téléscripteur le caractère représenté par les cinq chiffres binaires de poids fort de l'emplacement mémoire 41.

Le contenu de la pseudo-instruction 41 est 'πS' qui, dit le commentaire, correspond à 27, c'est-à-dire *Inversion Chiffres*.

85. Efface l'emplacement mémoire 129.

Passage au début de la ligne suivante

86. Imprime le caractère en attente, ce qui correspond ici à se placer en mode *chiffres*, et met en attente le contenu de l'emplacement mémoire 44, soit 'ΔS', ou passage à la ligne d'après ce que dit le commentaire.

87. Imprime la caractère en attente, ce qui correspond ici à un passage à la ligne, et met en attente le contenu de l'emplacement mémoire 45, soit 'θS', ou passage en début de ligne comme dit le commentaire.

Incrémentation de la variable x

88. Transfère le contenu de l'emplacement mémoire 76 (appelé variable x) à l'accumulateur. Au début du programme la partie numérique de celui-ci contient zéro.

89. Ajoute le contenu de l'instruction 4, qui est une pseudo-instruction, à l'accumulateur, ce qui permet d'ajouter 1 à sa partie numérique.

90. Place le contenu de l'accumulateur dans l'emplacement mémoire 76, c'est-à-dire qu'on a incrémenté la partie numérique de x, comme le confirme le commentaire.

91. On place également cette valeur à l'emplacement mémoire 48, qui contient le nombre à imprimer par la sous-routine ci-dessus, comme le confirme le commentaire, et on met à zéro l'accumulateur.

Affichage du nombre x

92. On transfère le contenu de l'emplacement mémoire 83, c'est-à-dire 'E 95 S', dans l'accumulateur. Ceci permettra le retour depuis la sous-routine à laquelle on va sauter à l'instruction 94 à l'instruction suivante, c'est-à-dire l'instruction 95.

93. On transfère ce contenu à l'emplacement mémoire 75, qui est la dernière instruction de la sous-routine, comme le confirme le commentaire.

94. On a un saut (inconditionnel) à la sous-routine commençant à l'instruction 49.

Sous-routine d'impression de la partie numérique de la pseudo-instruction 48

49–50. On transfère le contenu de l'emplacement mémoire 46, c'est-à-dire l'instruction '0 43 S', à l'emplacement mémoire 65.

51. On place zéro, le contenu de l'accumulateur, à l'emplacement mémoire 129.

52–53. On place la puissance de 10 adéquate (10 000 la première fois puis 1 000, 100, 10, 1 et enfin 0) dans l'accumulateur, puis dans l'emplacement mémoire 34.

54. S'il ne s'agit pas de 0, on a encore des chiffres décimaux à imprimer, on saute à l'instruction 61 pour imprimer le chiffre suivant.

Impression d'un chiffre décimal

61. On transfère le nombre à imprimer, contenu dans l'emplacement mémoire 48, dans l'accumulateur.

62. On lui soustrait le contenu de l'emplacement mémoire 34, c'est-à-dire la puissance de dix correspondant à l'ordre du chiffre à imprimer.

63. Tant que le nombre reste positif, on saute à l'instruction 55.

55. On transfère le nombre obtenu, auquel on vient de soustraire une puissance de 10, à l'emplacement mémoire 48.

56–57. On transfère le contenu de l'emplacement mémoire 47, c'est-à-dire l'instruction '0 33 S', dans l'accumulateur puis à l'emplacement mémoire 65.

58. On transfère le contenu de l'emplacement mémoire 33, 0 comme partie numérique initialement, dans l'accumulateur.

59. On lui ajoute le contenu de l'emplacement mémoire 40, ce qui revient à incrémenter la partie numérique.

60. On transfère le résultat à l'emplacement mémoire 33. On a donc incrémenté la partie numérique de l'emplacement mémoire 33.

On a donc soustrait autant de fois qu'il faut la puissance de 10 adéquate au nombre à afficher et incrémenté ce même nombre de fois le contenu de l'emplacement mémoire 33. Lorsqu'on revient à l'instruction 63 pour un passage à l'instruction suivante, la partie numérique de l'emplacement 33 contient donc le chiffre à imprimer.

64. Le contenu de l'accumulateur est négatif à ce moment-là. On ajoute le contenu de l'emplacement mémoire 34, c'est-à-dire la puissance de dix dont on vient d'obtenir le chiffre pour le nombre à imprimer, dans l'accumulateur de façon à y replacer le reste dans la division euclidienne par la puissance de dix du nombre à imprimer.

65. Rappelons que l'instruction 57 y a placé l'instruction '0 33 S'. On imprime donc le caractère en attente sur le téléscripteur et on met en attente sur le téléscripteur le chiffre que l'on vient de calculer.

Passage au chiffre suivant

66. On transfère le contenu de l'accumulateur, c'est-à-dire la partie du nombre qui reste à imprimer, à l'emplacement mémoire 48.

67. On réinitialise l'emplacement mémoire 33 à zéro.

68. On transfère dans l'accumulateur le contenu de l'emplacement mémoire 52, c'est-à-dire l'ordre de la puissance de 10 que l'on vient de traiter (0 pour 10 000, 1 pour 1 000 et ainsi de suite).

69. On lui ajoute le contenu de l'emplacement mémoire 4, c'est-à-dire une constante prédéfinie qui permet d'incrémenter sa partie numérique.

70. On transfère le résultat à l'emplacement mémoire 52, c'est-à-dire que les instructions 68 à 70 ont permis d'incrémenter la partie numérique du contenu de celui-ci.

71–72. On soustrait le contenu de l'emplacement mémoire 42. Si le résultat est positif, il faut traiter la puissance de 10 suivante en retournant à l'instruction 51.

Fin de l'impression du nombre

73–74. Si le résultat est négatif, on a imprimé tous les chiffres du nombre. On ajoute le contenu de l'emplacement mémoire 117 à l'accumulateur et on transfère le résultat à l'emplacement mémoire 52.

75. On est arrivé à la fin de la sous-routine. On saute donc à l'instruction 95, comme préparé par l'instruction 93.

Calcul du carré x^2

95–96. On affiche deux fois le contenu de l'emplacement 43, c'est-à-dire un espace pour séparer du nombre suivant.

97. On transfère le nombre contenu dans l'emplacement mémoire 76, c'est-à-dire la variable x, dans le registre multiplieur.

98. On le multiplie par le contenu de l'emplacement mémoire 76, on obtient donc x^2, et on l'ajoute à l'accumulateur, qui est nul avant cette opération. La partie numérique de l'accumulateur est donc x^2.

99–101. On ajuste ce nombre et on le transfère dans l'emplacement mémoire 77.

Calcul de la différence Δx^2

102. On lui soustrait le contenu de l'emplacement 78, le carré précédent, ce qui permet d'obtenir Δx^2.

103. On transfère cette valeur à l'emplacement mémoire 79.

104–105. On transfère le contenu de l'emplacement 77, c'est-à-dire le carré, dans l'accumulateur, que l'on transfère à l'emplacement mémoire 78, ce qui servira pour la fois suivante.

Affichage du carré x^2

106. On le transfère également à l'emplacement mémoire 48.

107–108. On transfère le contenu de l'emplacement mémoire 80 dans l'emplacement mémoire 75, c'est-à-dire que la sous-routine d'impression renverra à l'instruction 110.

109. On saute à la sous-routine d'impression.

Affichage de la différence Δx^2

110–111. On affiche un double espace pour séparer du nombre suivant.

112–113. On transfère le contenu de l'emplacement mémoire 79, c'est-à-dire Δx^2, dans l'emplacement mémoire 48.

114–115. On transfère le contenu de l'emplacement mémoire 81 dans l'emplacement mémoire 75, c'est-à-dire que la sous-routine d'impression renverra à l'instruction 118.

116. On saute à la sous-routine d'impression.

Boucle jusqu'à 100

118. On transfère le contenu de l'emplacement mémoire 76, c'est-à-dire la variable x, dans l'accumulateur.

119. On lui soustrait le contenu de l'emplacement mémoire 82, c'est-à-dire 100 à la partie numérique.

120. Si le résultat est négatif, donc si x est inférieur à 100, on retourne à l'instruction 85, fin de l'étape préparatoire pour traiter le nombre suivant.

Fin

121. Lorsqu'on a calculé les cent carrés de 1 à 100, on affiche le dernier caractère en attente (et on prépare pour *Inversion Chiffres*).

122. On arrête la machine et on fait sonner la clochette d'avertissement.

Ce programme est également commenté dans le chapitre 15 de [Lav-80] et dans [Cam-80b]. Il existe trois émulateurs de l'EDSAC dont celui de Martin Campbell-Kelly disponible sur le site :

`http://www.dcs.warwick.ac.uk/~edsac/`

15.7.2 Mise en place

15.7.2.1 Addition, soustraction et multiplication

Comme nous venons de le voir, le premier ordinateur, l'EDSAC de 1949, comprend les opérations d'addition (l'analofue de ADD), de soustraction (l'analogue de SUB) et la multiplication (l'analogue de MUL) mais pas celles d'incrémentation, de décrémentation ou d'opérations avec retenues.

15.7.2.2 Registre des indicateurs

Le premier ordinateur, l'EDSAC de 1949, est destiné à des programmeurs « intelligents » par nature ou par nécessité. Aucune aide ne leur est apportée. Il n'y a donc pas de registre des indicateurs.

15.8 Bibliographie

[Cam-80] CAMPBELL-KELLY, Martin, *Programming the Manchester Mark I*, **Annals of the History of Computing**, vol. 2, avril 1980, pp. 130–168.

[Cam-80b] CAMPBELL-KELLY, Martin, *Programming the EDSAC : Early Programming Activity at the University of Cambridge*, **Annals of the History of Computing**, vol. 2, 1980, pp. 7–36 ; reprinted in vol. 20, 1998, pp 46–67.

[Lav-80] LAVINGTON, Simon, **Early British Computers**, Digital Press, Bedford, Mass., 1980.

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

[RH-00] ROJAS, Raúl, HASHAGEN, Ulf, **The First Computers : History and Architectures**, The MIT Press, 2000, XII + 457 p., ISBN 0-262-68137-4.

[SB-98] SHELBURNE, Brian J., BURTON, Christopher P., *Early Programs on the Manchester Mark I Prototype*, **Annals of the History of Computing**, vol. 20, 1998, pp. 4–15.

[Wor-50] WORSLEY, B. H., *The EDSAC Demonstration*, in **Report of a Conf. on High Speed Automatic Calculating Machines, 22-25 June 1949**, Univ. Math. Lab., Cambridge, England, Jan. 1950, pp. 12–16. Reproduit dans [Ran-82], pp. 423–429.