

# 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 élémentaires. Nous avons étudié les accès dans les chapitres précédents. Voyons maintenant comment effectuer des calculs élémentaires, en commençant par ceux sur les entiers naturels.

### 15.1 Représentation et opérations arithmétiques

#### 15.1.1 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 0000h à FFFFh, c'est-à-dire de 0 à 65 535 ( $= 2^{16} - 1$ ).

Représentation.- On représente ces entiers tout simplement en numération binaire.

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 ».

### 15.1.2 Choix des opérations arithmétiques primitives

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'*incrément*, 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 grâce aux structures de contrôle. 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 calculent 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 calculatrice 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 Incrémentation et décrémentation

### 15.2.1 Incrémentation

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

#### 15.2.1.1 Incrémentation d'un registre de 16 bits

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

```
INC reg
```

où **reg** est un registre général 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 FFFFh 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 12.1).

Exercice corrigé.- Traduire l'instruction :

```
INC AX
```

en langage machine.

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

```
0100 0 000
```

soit 40h.

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 1000 :400.

Le programme s'écrit en langage symbolique :

```
MOV AX,1000
MOV DS,AX
MOV AX, 11
INC AX
MOV [0400],AX
RET
```

Traduisons maintenant ces instructions en langage machine :

```
B8 00 10
8E D8
B8 11 00
40
A3 00 04
C3
```

Codons ce programme :

```

C:\COM>debug
-E CS:100 B8 00 10
-E CS:103 8E D8
-E CS:105 B8 11 00
-E CS:108 40
-E CS:109 A3 00 04
-E CS:10C C3
-R BX
BX 0000
:
-R CX
CX 0000
:OD
-N INC.COM
-W
Ecriture de 0000D octets
-Q

```

Après exécution du programme `inc.com`, on vérifie que l'on a bien 18 (= 12h) à l'emplacement mémoire choisi :

```

C:\COM>debug
-D 1000:400
1000:0400 12 00 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 ..KS KSKSKSKSKSKSKS
1000:0410 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0420 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0430 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0440 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0450 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0460 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0470 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
-q

```

Remarque.- Il n'y avait pas, jusqu'à maintenant, vraiment de moyen de vérifier que le 8086 utilise la convention petitboutienne. Cet exemple, par contre, le confirme : s'il utilisait la convention grandboutienne, on se retrouverait avec 11 01h comme résultat et non 12 00h.

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

Exercice<sup>L</sup>.- Concevoir un circuit (combinatoire) qui permette d'effectuer l'incréméntation.

**15.2.1.2 Incrémentation d'un registre (de 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 code par les deux octets représentés en binaire par :

| 1111 1110 | 11 000 00 |

soit FE C0h.

## 15.2.2 Décrémentation

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

### 15.2.2.1 Décrémentation d'un registre 16 bits

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

$$\text{DEC reg}$$

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 FFFFh (et non pas à 0 comme ce serait le cas en théorie).

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

$$0100\ 1\ \text{reg}$$

où le registre est désigné de la façon maintenant habituelle (tableau de la figure 12.1).

Exercice corrigé.- Traduire l'instruction :

$$\text{DEC AX}$$

en langage machine.

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

$$0100\ 1\ 000$$

soit 48h.

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 1000 :400.

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

Exercice<sup>L</sup>.- Concevoir un circuit (combinatoire) qui permette d'effectuer la décrémentation.

**15.2.2.2 Décrémenter d'un registre (de 8 ou 16 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 décrémenter 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 code par les deux octets représentés en binaire par :

| 1111 1110 | 11 001 00 |

soit FE C8h.

## 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 sur 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 la taille du registre. 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 bas | donnée haut si w = 1 |

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

Exercice corrigé 1.- Traduire l'instruction :

ADD AL, 3h

*en langage machine.*

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

| 0000 0100 | 0000 0011 |

soit 04 03h.

Exercice corrigé 2.- Traduire l'instruction :

ADD AX, 1FC4

*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 05 C4 1Fh.



Exemple.- Écrivons un programme en langage machine qui place 17 dans l'accumulateur AX, ajoute 5 à celui-ci puis place le résultat à l'emplacement mémoire 1000 :400.

Le programme s'écrit en langage symbolique :

```
MOV AX,1000
MOV DS,AX
MOV AX,11
ADD AX,5
MOV [0400],AX
RET
```

Traduisons maintenant ces instructions en langage machine :

```
B8 00 10
8E D8
B8 11 00
05 05 00
A3 00 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 10
-E CS:103 8E D8
-E CS:105 B8 11 00
-E CS:108 05 05 00
-E CS:10B A3 00 04
-E CS:10E C3
-R BX
BX 0000
:
-R CX
CX 0000
:OF
-N ADD.COM
-W
Ecriture de 0000F octets
-Q
```

Après exécution du programme add.com, on vérifie que l'on a bien 22 (= 16h) à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 1000:400
1000:0400 16 00 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 ..KSKSKSKSKSKSKS
1000:0410 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0420 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0430 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0440 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0450 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0460 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
1000:0470 4B 53 4B 53 4B 53 4B 53-4B 53 4B 53 4B 53 4B 53 4B 53 KSKSKSKSKSKSKSKS
-q
```

**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 1.- Traduire l'instruction :

ADD CH, BL

*en langage machine.*

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

| 0000 0010 | 11 101 011 |

soit 02 EBh.

Exercice corrigé 2.- Traduire l'instruction :

ADD CX, 432

*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 81 0E 32 04h.

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 de 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 5 indicateurs (déjà présents sur le microprocesseur 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 (respectivement dix-septième) bit d'une addition ou d'une soustraction de deux entiers de huit bits (respectivement 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.  
Il n'a pas d'intérêt pour les entiers naturels.
- 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.  
C'est évidemment un résidu du 4004 conçu pour les calculettes.

L'octet de poids fort comporte quatre indicateurs (nouveaux par rapport au 8085) :

- 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.  
Il n'a pas d'intérêt pour les entiers naturels.
- 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*).  
Il n'a pas d'intérêt pour les entiers naturels.
- 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*).  
Il n'a pas d'intérêt pour les entiers naturels.

— 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 l'un au moins 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 poids faible 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 |

Exercice corrigé.- Traduire l'instruction :

ADC AX, 1FC4

en langage machine.

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

| 0001 0101 | 1100 01000 | 0001 1111 |

soit 15 C4 1Fh.

Exemple.- Écrivons un programme permettant de calculer la somme de 99FFEh et 88003h et de placer le résultat à l'emplacement mémoire 3000 :400.

Avant même d'aborder l'écriture du programme, il faut nous interroger sur une convention. Les entiers manipulés tiennent sur plus d'un mot. Quelle est la convention de stockage de ces « grands » entiers ? Cela n'a pas d'importance pour les microprocesseurs 32 bits (mais en aura pour les microprocesseurs *Intel* 64 bits). Cela a une importance, par contre, pour les systèmes d'exploitation qui permettent de manipuler, en natif, les « grands » entiers. La convention choisie est alors la continuation naturelle du stockage des mots : 12345678h est stocké de la façon suivante :

78 56 34 12

Il n'y a pas de raison, adoptons, nous aussi, cette convention.

Le mot de poids faible sera donc placé à l'emplacement mémoire 1000 :400 et celui de poids fort à l'emplacement 1000 :402. On a bien 402 et non 401 car un mot occupe deux octets.

Le programme s'écrit en langage symbolique :

```
MOV AX,3000
MOV DS,AX
MOV AX,9FFE
ADD AX,8003
MOV [0400],AX
MOV AX,9
ADC AX,8
MOV [0402],AX
RET
```

Traduisons maintenant ces instructions en langage machine :

```
B8 00 30
8E D8
B8 FE 9F
05 03 00
A3 00 04
B8 09 00
15 08 00
A3 02 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 B8 FE 9F
-E CS:108 05 03 00
-E CS:10B A3 00 04
-E CS:10E B8 09 00
-E CS:111 15 08 00
-E CS:114 A3 02 04
-E CS:117 C3
-R BX
BX 0000
:
-R CX
CX 0000
:019
-N ADC.COM
-W
Ecriture de 00019 octets
-Q
```

Après exécution du programme `adc.com`, on vérifie que l'on a bien `11A001h` à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 01 A0 11 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
```

3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....  
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....  
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....

-q

## 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 bas | donnée haut 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 1.- Traduire l'instruction :

SUB AL, 4h

*en langage machine.*

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

| 0010 1100 | 0000 0100 |

soit 2C 04h.



Exercice corrigé 2.- Traduire l'instruction :

SUB AX, 660h

*en langage machine.*

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

| 0010 1101 | 0110 0000 | 0000 0110 |

soit 2D 60 06h.

Exercice corrigé 3.- Traduire l'instruction :

SUB CH, DL

*en langage machine.*

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

| 0010 1010 | 11 101 010 |

soit 2A EAh.

Exercice corrigé 4.- Traduire l'instruction :

SUB SI, 312

*en langage machine.*

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

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

soit 81 EE 38 01h.

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 à l'emplacement mémoire 3000 :400.

Le programme s'écrit en langage symbolique :

```
MOV AX,3000
MOV DS,AX
MOV AX,11
SUB AX,5
MOV [0400],AX
RET
```

Traduisons maintenant ces instructions en langage machine :

```
B8 00 30
8E D8
B8 11 00
2D 05 00
A3 00 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 B8 11 00
-E CS:108 2D 05 00
-E CS:10B A3 00 04
-E CS:10E C3
-R BX
BX 0000
:
-R CX
CX 0000
:OF
-N SUB.COM
-W
Ecriture de 0000F octets
-Q
```

Après exécution du programme `sub.com`, on vérifie que l'on a bien 11 (= 0Ch) à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 0C 00 11 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
-q
```

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

- L'indicateur de retenue CF est levé lorsqu'il y a emprunt, 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 emprunt du quartet de poids fort pour le quartet de poids faible.
- 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 l'un au moins 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 emprunt lors de la soustraction des deux mots de poids faible dont il faut tenir compte dans la soustraction des deux mots de poids fort.

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

Soustraction avec emprunt.- L'instruction :

SBB destination, source

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

$$\text{destination} := \text{destination} - (\text{source} + \text{CF})$$

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  $w = 1$  |

Exercice corrigé.- Traduire l'instruction :

SBB AX, 1FC4

en langage machine.

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

| 0001 1101 | 1100 01000 | 0001 1111 |

soit 1D C4 1Fh.

Exemple.- Écrivons un programme permettant de calculer la différence de 98FFEh et de 89003h et de placer le résultat à l'emplacement mémoire 3000 :400.

Comme pour l'addition sur deux mots, le mot de poids faible sera placé à l'emplacement mémoire 1000 :400 et celui de poids fort à l'emplacement 1000 :402.

Le programme s'écrit en langage symbolique :

```
MOV AX,3000
MOV DS,AX
MOV AX,8FFE
SUB AX,9003
MOV [0400],AX
MOV AX,9
```

```
SBB AX,8
MOV [0402],AX
RET
```

Traduisons maintenant ces instructions en langage machine :

```
B8 00 30
8E D8
B8 FE 8F
05 03 90
A3 00 04
B8 09 00
1D 08 00
A3 02 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 B8 FE 8F
-E CS:108 2D 03 90
-E CS:10B A3 00 04
-E CS:10E B8 09 00
-E CS:111 1D 08 00
-E CS:114 A3 02 04
-E CS:117 C3
-R BX
BX 0000
:
-R CX
CX 0000
:019
-N SBB.COM
-W
Ecriture de 00019 octets
-Q
```

Après exécution du programme `sbb.com`, on vérifie que l'on a bien `FFFBh` à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 FB FF 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-q
```

## 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 registre 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 1.- Traduire l'instruction :

MUL BL (aussi écrite MUL AL, BL)

en langage machine.

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

| 1111 0110 | 11 100 011 |

soit F6 E3h.

Exercice corrigé 2.- Traduire l'instruction :

MUL BX (aussi écrite MUL AX, BX)

en langage machine.

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

| 1111 0111 | 11 100 011 |

soit F7 E3h.

Exercice corrigé 3.- Traduire l'instruction :

MUL AX, [1FC4h]

en langage machine.

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

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

soit F7 26 C4 1Fh.

Test d'un programme.- Écrivons un programme en langage machine qui place 1234h dans le registre BX, qui place 23h dans l'accumulateur AX, qui multiplie le contenu de AX par BX puis place le résultat à l'emplacement mémoire (de quatre octets) 3000 :400.

Le programme s'écrit en langage symbolique :

```
MOV AX,3000
MOV DS,AX
MOV BX,1234
MOV AX,23
MUL BX
MOV [0400],AX
MOV [0402],DX
RET
```

Traduisons maintenant ces instructions en langage machine :

```
B8 00 30
8E D8
BB 34 12
B8 23 00
F7 E3
A3 00 04
89 16 02 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 BB 34 12
-E CS:108 B8 23 00
-E CS:10B F7 E3
-E CS:10D A3 00 04
-E CS:110 89 16 02 04
-E CS:114 C3
-R BX
BX 0000
:
-R CX
CX 0000
:015
-N MUL.COM
-W
Ecriture de 00015 octets
-Q
```

Après exécution du programme mul.com, on vérifie que l'on a bien 27D1Ch à l'emplacement mémoire choisi :

```
C:\COM>debug
-D 3000:400
3000:0400 1C 7D 02 00 00 00 00 00-00 00 00 00 00 00 00 00 .}.
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
```

```

3000:0430 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0440 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0450 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0470 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
3000:0470 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-q

```

Incidence sur les indicateurs.- Si, après une multiplication, 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 se retrouvent dans un état indéfini.



## 15.6 Division euclidienne

Introduction.- 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 dont le résultat est constitué de deux entiers naturels, et non d'un seul. Pour un entier  $a$  et un entier non nul  $b$ , il existe un unique couple  $(q, r)$  d'entiers tels que :

$$a = b.q + r \text{ et } 0 \leq r < b.$$

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

Langage symbolique.- 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.

Sémantique.- Cette instruction permet d'effectuer la division euclidienne du dividende par le diviseur, le quotient exact est placé dans AL (resp. AX) et le reste dans AH (resp. 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 1.- Traduire l'instruction :

DIV BL

en langage machine.

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

| 1111 0110 | 11 110 011 |

soit F6 F3h.

Exercice corrigé 2.- Traduire l'instruction :

DIV BX

en langage machine.

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

| 1111 0111 | 11 110 011 |

soit F7 F3h.

Exercice corrigé 3.- Traduire l'instruction :

DIV byte ptr [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 code sur quatre octets représentés en binaire par :

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

soit F6 36 C4 1Fh.

Test d'un programme.- Écrivons un programme en langage machine qui place 34h dans le registre BL, qui place 1234h dans l'accumulateur AX, effectue la division euclidienne de [AX] par [BL] puis place le quotient à l'emplacement mémoire (d'un octet) 3000 :400 et le reste à l'emplacement mémoire 3000 :401.

Le programme s'écrit en langage symbolique :

```
MOV AX,3000
MOV DS,AX
MOV BL,34
MOV AX,1234
DIV BL
MOV [0400],AX
RET
```

Traduisons maintenant ces instructions en langage machine :

```
B8 00 30
8E D8
B3 34
B8 34 12
F6 F3
A3 00 04
89 16 02 04
C3
```

Codons ce programme :

```
C:\COM>debug
-E CS:100 B8 00 30
-E CS:103 8E D8
-E CS:105 B3 34
-E CS:107 B8 34 12
-E CS:10A F6 F3
-E CS:10C A3 00 04
-E CS:10F C3
-R BX
BX 0000
:
-R CX
CX 0000
:010
-N DIV.COM
-W
Ecriture de 00010 octets
-Q
```

Après exécution du programme `div.com`, on vérifie que l'on a bien un quotient de 59h et un reste de 20h :

```
C:\COM>debug
-D 3000:400
3000:0400 59 20 00 00 00 00 00 00-00 00 00 00 00 00 00 00 Y .....
3000:0410 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0420 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0430 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0440 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0450 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
3000:0470 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
-q
```

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) :

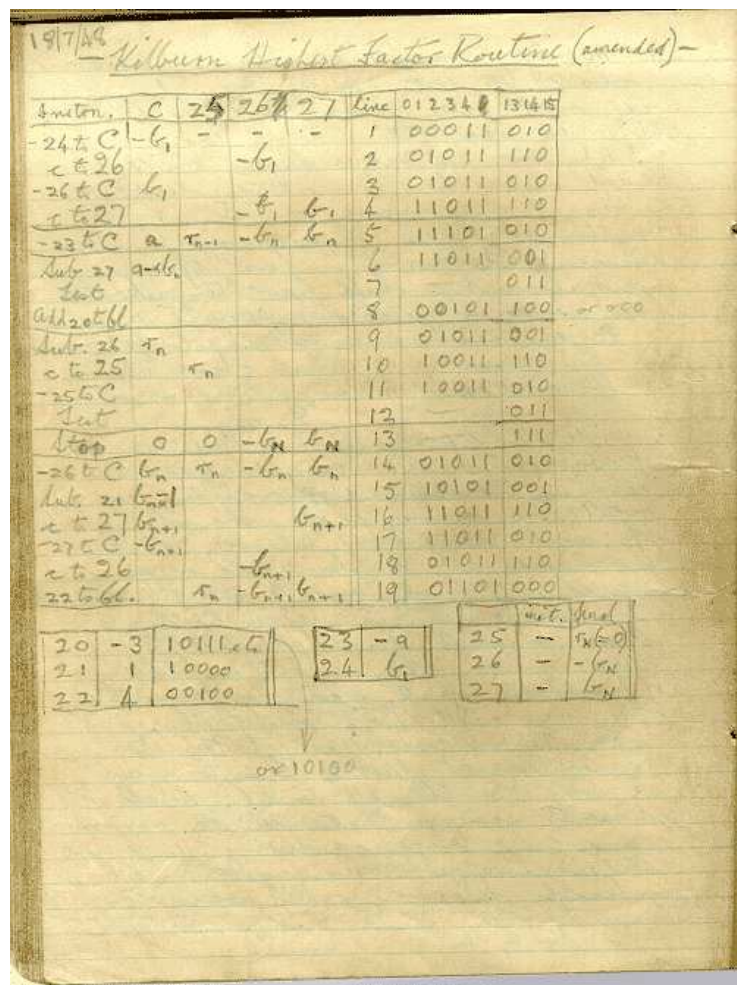


FIGURE 15.1 – Le premier programme (18 juillet 1948)

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

*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 15.1).

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

<u>Memory position.</u>	<u>Order.</u>	
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 }	Input function digits; shift to correct position in 0. $x_1 = 0$
7	I 0 S }	
8	A 0 S }	
9	R 16 S }	
10	T 0 L	
→1	I 2 S }	Input next symbol. Test for digit or discriminant.
2	A 2 S }	
3	S 5 S }	
4	E 21 S }	
5	T 3 S }	
6	V 1 S }	$10x_1 + b$ to 1.
7	L 8 S }	
8	A 2 S }	
9	T 1 S }	
20	E 11 S	
→1	R 4 S }	Length discriminant digit to acc.
2	A 1 S }	
3	L 0 L }	Form order and transfer to n.
4	A 0 S }	
5	(T 31)S }	
6	A 25 S }	$n + 1$ to n.
7	A 4 S }	
8	U 25 S }	
9	S 31 S }	
30	G 6 S }	Test for start of programme.

FIGURE 15.2 – 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 :

31 T	123 S	{ As required by	5(P S)	End print
		{ initial input.	6(P S)	Becomes x.
2 E	84 S	{ Jump to 84.	7(P S)	Becomes $x^2$ .
3(P S)		{ Use to keep count	8(P S)	Becomes $x^2$ .
		{ of subtractions.	9(P S)	Becomes $\Delta x^2$ .
4(P S)		{ Power of 10 being	80 E 110 S	
		{ subtracted.	1 E 118 S	
5 P	10000 S	} For use in the	2 P 100 S	
6 P	1000 S		83 E 95 S	
7 P	100 S		4 O 41 S	Set on print figures
8 P	10 S		5 T 129 S	Clear 129.
9 P	1 S	} Pseudo-	6 O 44 S	
40 Q	S		7 O 45 S	
1 $\pi$	S		8 A 76 S	} $x + 1$ to S(76) and
2 A	40 S		9 A 4 S	
3 $\varphi$	S		90 U 76 S	
4 $\Delta$	S		1 T 48 S	
5 $\vartheta$	S		2 A 83 S	Set switch Z.
6 O	43 S		3 T 75 S	
7 O	33 S	4 E 49 S		
8(P S)		{ Becomes number to	5 O 43 S	
		{ be printed	6 O 43 S	Double space.
→ 9 A	46 S	{ Put O 43 S in S(65)	7 H 76 S	} $x^2 \cdot 2^{15}$ to S(77).
50 T	65 S		8 V 76 S	
1 T	129 S	{ Clear S(129).	9 L 64 S	
2(A 35 S)		{ Put power of 10	100 L 32 S	
3 T	34 S	{ in S(34).	1 U 77 S	
4 E	61 S	{ Jump to 61	2 S 78 S	
→ 5 T	48 S		3 T 79 S	} $\Delta x^2$ to S(79)
6 A	47 S		4 A 77 S	
57 T	65 S		5 U 78 S	} $x^2$ to S(49) and print.
8 A	33 S		6 T 48 S	
9 A	40 S	{ To control printing	107 A 80 S	
60 T	33 S		8 T 75 S	
→ 1 A	48 S		9 E 49 S	
2 S	34 S		110 O 43 S	} Double space.
3 E	55 S		1 O 43 S	
4 A	34 S		2 A 79 S	
5(P S)			3 T 48 S	} $\Delta x^2$ to S(48) and print.
6 T	48 S	{ Print contents	4 A 81 S	
7 T	33 S	{ of S(48).	5 T 75 S	
8 A	52 S		6 E 49 S	} Test for finish.
9 A	4 S		7 A 35 S	
70 U	52 S		8 A 76 S	
1 S	42 S		9 S 82 S	
2 G	51 S		120 G 85 S	
3 A	117 S		1 O 41 S	
4 T	52 S		2 Z S	

FIGURE 15.3 – Deuxième routine (Worsley 1950)

## Démonstration de l'E.D.S.A.C.

(W. Renwick)

Rapport préparé par mademoiselle B. H. Worsley

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.

31 T	107 S		57 A	96 S		84 A	6 S	} Test p If the 5th no. printed, linefeed, carriage return.
→ 2 O	92 S	Figures	8 E	39 S		5 A	98 S	
→ 3 O	93 S	Line feed	9 T	7 S		6 G	36 S	
	4 O	94 S	{ Carriage return	60 A	96 S		7 E	33 S
	5 S	5 S	} Set p	1 U	1 S		8 P	2 S $\equiv$ 4
→ 6 T	6 S			2 A	4 S		9 P	500 S
	7 O	95 S	} Double space	3 T	96 S	} go to next factor	90 J	S $\equiv$ $\frac{8}{10}$
	8 O	95 S		4 A	99 S			1 P
→ 9 T	7 S		5 T	97 S		2 $\pi$	S	
	40 A	96 S		6 S	88 S		3 $\vartheta$	S
	1 R	4 S		→ 7 T	7 S		4 $\Delta$	S
→ 2 S	97 S			8 H	91 S		5 $\varphi$	S
	3 E	42 S	} Test whether	9 A	1 S		6(P	2 L)n
	4 L	4 S		m a factor	→ 1 V	91 S		7(P
→ 5 A	97 S		of n.	→ 2 S	89 S		8 P	L $\equiv$ 1
	6 G	45 S		→ 3 E	71 S	} If n Prime, transfer → to S(O).	9 P	1 L $\equiv$ 3
	7 S	98 S		4 A	89 S			100 T
	8 G	100 S		5 T	L		1 A	99 S
	9 T	7 S		6 O	S		2 T	97 S
	50 A	97 S	} m + 2 to m	7 H	90 S		3 A	4 S
	1 A	4 S			8 V	1 S		4 A
	2 T	97 S		9 L	4 S	} Print C(O)	5 T	96 S
	3 H	97 S		80 T	L			6 E
	4 N	97 S	} if $m > \sqrt{n}$ stop testing	1 A	7 S			
	5 L	64 S			2 A	98 S		
	6 L	64 S		3 G	67 S			

FIGURE 15.4 – Troisième routine (Worsley 1950)

#### 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.

2. Les ordres initiaux (i) (voir ci-dessous) ont alors été 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 transférés automatiquement à leurs emplacements de

stockage appropriés. Cette opération a pris à peu près 5 secondes et a été 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 s'est faite à la cadence d'un symbole en 150 millisecondes ou, en moyenne, d'un ordre et demi par seconde, un clic étant entendu pour chaque symbole.

4. Finalement on a vu le téléscripateur 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 est devenu de plus en plus appréciable à mesure que les nombres testés sont devenus de plus en plus grands. L'impression des carrés était 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 ont été 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.

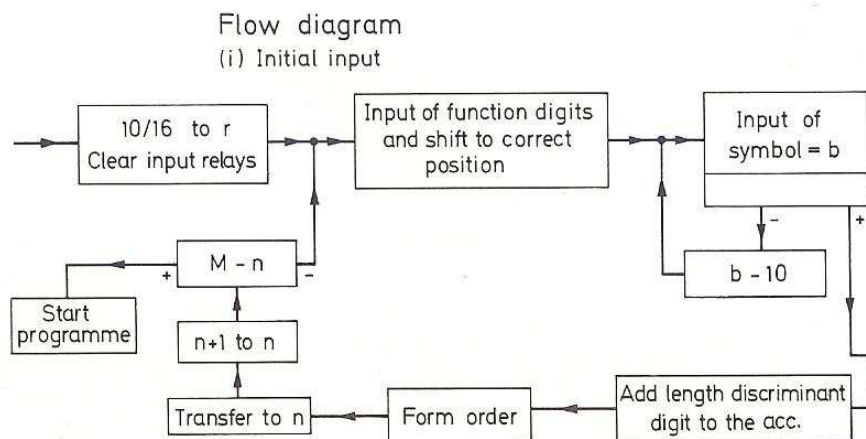


FIGURE 15.5 – Premier organigramme (Worsley 1950)

#### 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 :

(Lettre 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<sup>1</sup>.

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  $5 \times 2^{-15}$  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 nécessaires soient construits à partir d'entiers simples de cette façon.

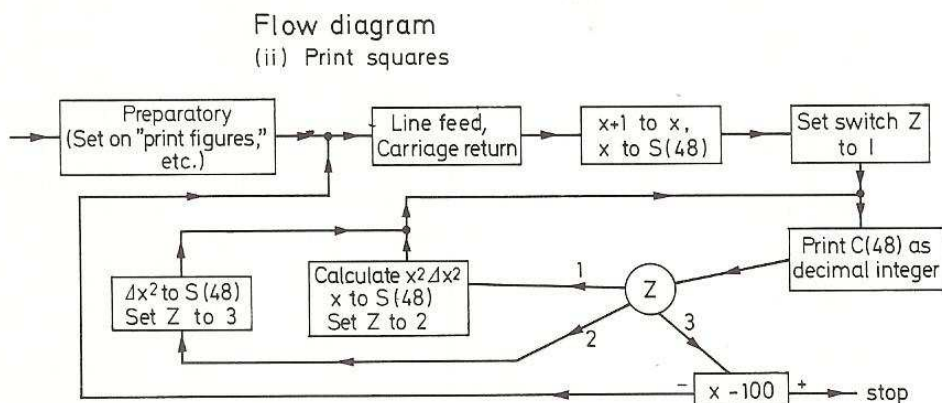


FIGURE 15.6 – Second organigramme (Worsley 1950)

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 téléscripneur. 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 téléscripneur 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 (Long) ou S (Short) 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 correctes.

[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.

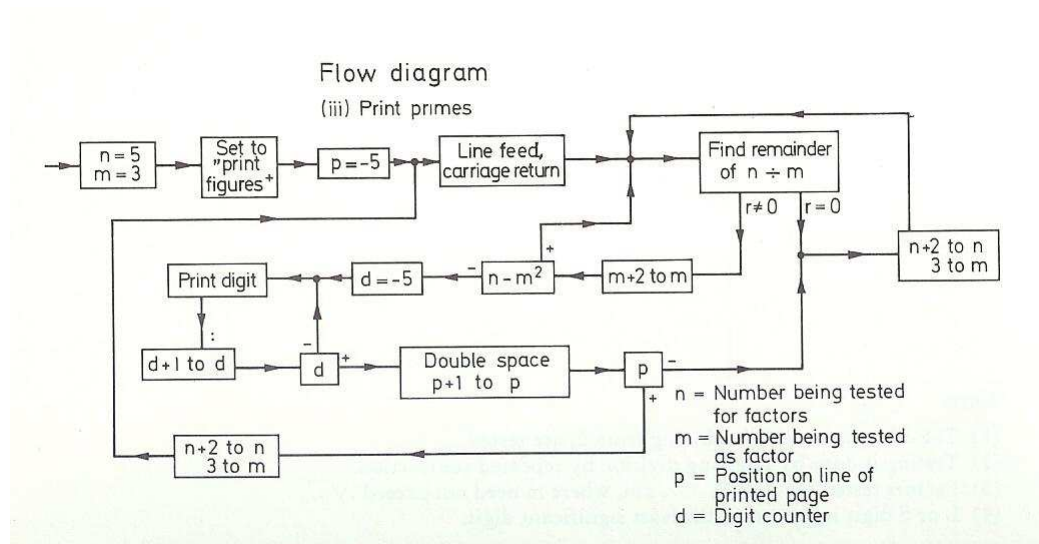


FIGURE 15.7 – Troisième organigramme (Worsley 1950)

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.

$n = 0. 1. 2. \dots 1023$ , 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'exécède 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° 1.

**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	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.

FIGURE 15.8 – Table des carrés (Worsley 1950)

1. L'instruction 1 initialise le multiplicateur à 2.
2. L'instruction 2 réinitialise l'emplacement mémoire 0 à 0 (au lieu du « T O 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

FIGURE 15.9 – Table des nombres premiers (Worsley 1950)

La façon dont les instructions sont écrites sur le ruban perforé n'est peut-être pas claire. 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 *discriminant* '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 télécriteur 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 15.6.

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

Le contenu de la pseudo-instruction 41 est ' $\pi S$ ' 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 ' $\Delta S$ ', 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 ' $\theta S$ ', 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 téléscripneur et on met en attente sur le téléscripneur 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  $\Delta x^2$** 

102. On lui soustrait le contenu de l’emplacement 78, le carré précédent, ce qui permet d’obtenir  $\Delta x^2$ .

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  $\Delta x^2$** 

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  $\Delta x^2$ , 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'analogue 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.