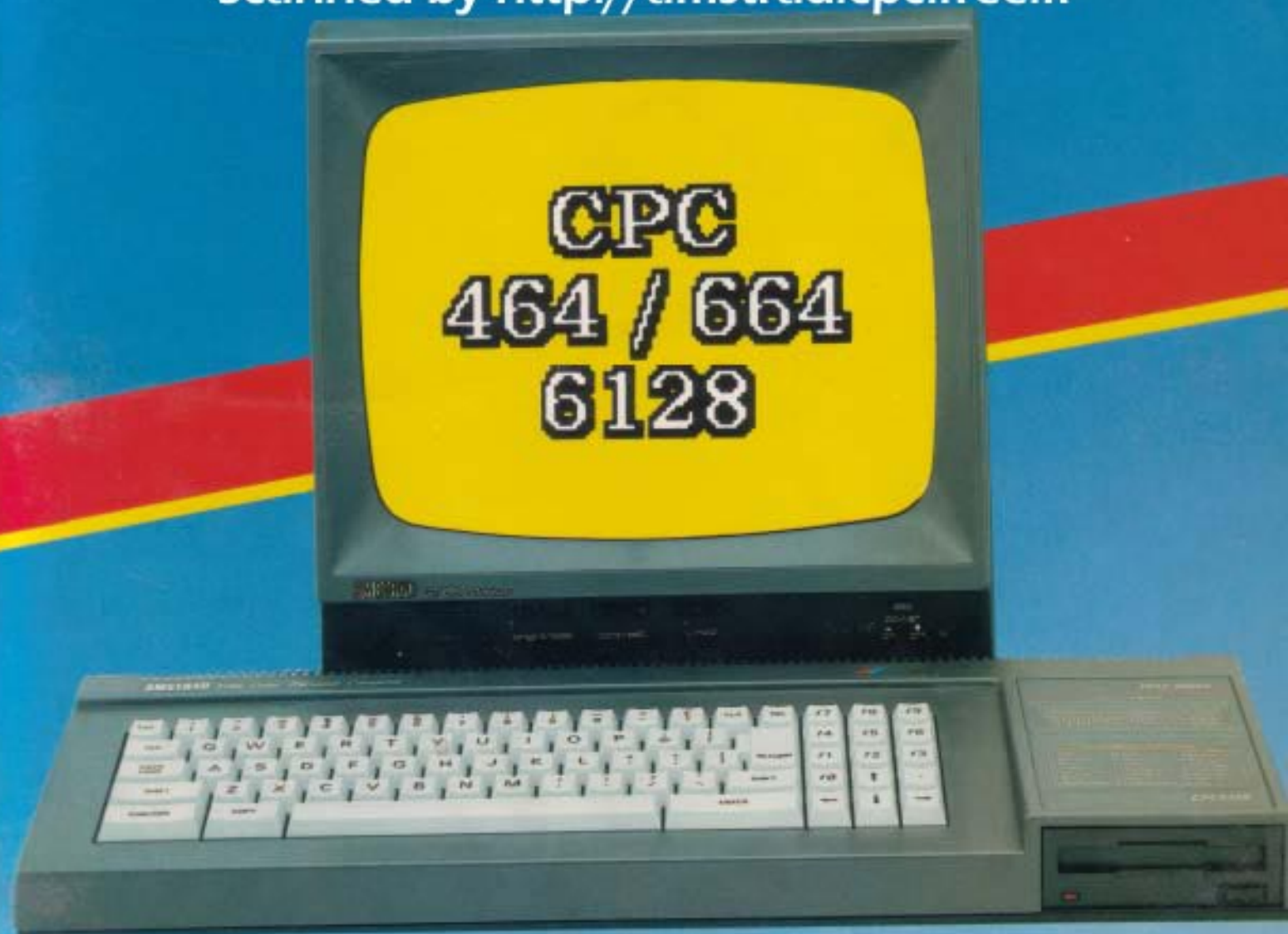


Comment exploiter
toutes les ressources
et augmenter les
performances de votre

AMSTRAD

Scanned by <http://amstrad.cpc.free.fr>



Partie 4

Langages du CPC

4/0

Table des matières

4/1	Locomotive BASIC : Définitions et rappels de base
4/1.1	Pourquoi utiliser le Basic et dans quels domaines ?
4/1.2	Version 1.0 sur CPC 464 : Mots clés et leur utilisation
	I. Affichage sur l'écran
	II. Gestion du clavier
	III. Gestion de l'unité de cassette
	IV. Manipulation de chaînes de caractères
	V. Gestion de données
	VI. Sous programmes et branchements
	VII. Gestion des erreurs
	VIII. Gestion de la mémoire
	IX. Gestion des interruptions
	X. Instructions musicales
	XI. Instructions graphiques
	XII. Fonctions à caractère mathématique
	XIII. Aides à la programmation
	XIV. Divers
4/1.3	Version 1.1 sur CPC 664 et CPC 8128 : Extensions par rapport à la version 1.0 du CPC 464
4/1.4	Rappel des ordres BASIC et de leur fonction
4/1.5	Cours de programmation
4/1.6	Basic approfondi
4/1.6.1	SYMBOL et SYMBOL AFTER
4/1.6.2	L'instruction CALL et les RSX en Basic
	I. L'instruction CALL
	II. Les RSX
4/1.6.3	La gestion des variables dans les Amstrad CPC
	I. Les variables entières
	II. Les variables en virgule flottante
	III. Les variables alphanumériques
4/1.6.4	Utilisation des vecteurs du système d'exploitation sous Basic
	I. Présentation
	II. Le programme Assembleur
	III. Le chargeur Basic
4/1.6.5	Formater une disquette sous Basic
	I. Rappel sur les disquettes et les différents formats
	II. La logique du formatage et l'instruction UFORMAT

4/1.6.6 Accélérez vos programmes Basic - Les tokens

- I. Basic-Tortue
- II. Vers le Basic-Lièvre
- III. La bonne utilisation des variables
- IV. Éliminons les blancs inutiles
- V. Fini le bla bla...

4/2 Assembleur Z80 : Définitions et rappels de base

4/2.1 Pourquoi utiliser l'Assembleur et dans quels domaines ?

4/2.2 Les modes d'adressage

4/2.3 Les mots clés de l'Assembleur et leur utilisation

- I. Usage général et interruptions
- I. Lecture et écriture en mémoire
- II. Lecture, écriture, échanges et opérations sur les registres
- IV. Ruptures de séquences
- V. Opérations arithmétiques
- VI. Opérations logiques
- VII. Manipulation de bits et de chaînes
 - A. Instructions sur bits
 - B. Instructions sur chaînes
- VIII. Opérations sur la pile
- IX. Entrées/Sorties

4/2.4 Liste alphabétique des codes opérateurs de l'Assembleur Z80

4/2.5 Cours de programmation

4/2.5.1 Initiation au langage machine

- I. Assembleur ou langage machine ?
- II. Votre première routine machine
- III. Un programme machine plus utile
- IV. Vos outils de programmation
- V. Quelques subtilités de programmation

4/2.6 Assembleurs existants

4/2.6.1 DEVPAO

4/2.7 Les banques ROM ou FIRMWARE

4/2.8 Les instructions RESTART des CPC

4/2.9 Les RSX

4/2.10 Accès aux vecteurs mathématiques en Assembleur et utilisation

4/2.11 Les interruptions sur Amstrad

- I. Comment est interrompu le CPC
- II. Les interruptions matérielles (Hard)
- II. Les interruptions logicielles (Soft)

4/3 LOGO : Définitions et rappels de base

4/3.1 Pourquoi utiliser le LOGO et dans quels domaines ?

4/3.2 Les mots clés du LOGO et leur utilisation

- I. Traitement des chaînes, nombres et opérateurs booléens
 - A. Traitement de chaînes
 - B. Traitement de nombres : opérations arithmétiques
 - C. Traitement de nombres : opérations logiques
- II. Variables et procédures
 - A. Variables
 - B. Procédures

SCRS 2004-2005 Système A avec base de III. Fonctions d'affichage : écran, couleurs et fenêtres

- III. Fonctions d'entrage : commande, données
 - A. Ecran texte
 - B. Ecran graphique
 - C. Primitives d'action directe sur l'écran
 - D. Primitives de manipulation de la tortue
- IV. Entrées/Sorties sur disquette
- V. Gestion des claviers et joystick
 - A. Clavier
 - B. Joystick
- VI. Commande du circuit sonore
- VII. Divers
 - A. Primitives d'ordre général et de mise au point de programmes LOGO
 - B. Traitement des erreurs
 - C. Edition et correction des procédures de l'espace de travail
 - D. Gestion de l'espace de travail
 - E. Fonctions d'impression

4/3.3 Liste alphabétique des primitives du LOGO

4/3.4 Franciser le Dr. LOGO de l'Amstrad

4/3.5 Programmes d'application

- 4/3.5.1 D'une mémoire en LOGO 2 et LOGO 3
- 4/3.5.2 Devine le nombre en LOGO 2 et LOGO 3
- 4/3.5.3 Chiffres romains en LOGO 2 et LOGO 3
- 4/3.5.4 LO 0 en LOGO 2 ou LOGO 3
- 4/3.5.5 Histogrammes en LOGO 2 et LOGO 3

4/4 Turbo-Pascal[®] : Définitions et rappels de base

- I. Entrées/Sorties sur écran
- II. Structure du programme
- II. Types de données comp. exes

4/4.1 Pourquoi utiliser le PASCAL et dans quels domaines ?

4/4.2 La programmation structurée en PASCAL

Structure d'un programme écrit en Turbo-Pascal

- II. Les concepts évolués du Turbo-Pascal
- A. En tête du programme
 - B. Etiquettes
 - C. Types
 - D. Constantes
 - E. Variables
 - F. Procédures et fonctions
 - G. Programme principal

4/4.3 Les mots réservés de Turbo-Pascal

4/4.4 Identificateurs standard

- I. Liste en fonction des identificateurs standard
- II. Les identificateurs standard en détail

4/4.5 Utilisation du Turbo-Pascal

- 4/4.5 Optimisation de l'arbre binaire
- 4/4.5.1 Optimisation d'écriture dans un fichier texte
- 4/4.5.2 Détermination de tous les nombres
- 4/4.5.3 Position du curseur sur l'écran
- 4/4.5.4 Programmation d'un traitement de texte

4/5	Le langage Forth 83-Standard pour Amstrad 484, 664, 6128 et PCW
	I. Genèse du langage Forth II. Domaines d'applications
4/5.1	Le langage Forth sur les Amstrad
	I. Les versions FIG II. Les versions 83 - Standard III. Premier contact A. Premières opérations arithmétiques B. La manipulation des données sur la pile
4/5.2	Le compilateur Forth
	I. Compilation de votre première définition II. Les constantes et les variables simple précision III. Les constantes et les variables double précision
4/5.3	Contrôle de l'affichage
	I. Format des nombres traités par Forth II. Formats d'affichage des nombres entiers et double précision III. Définition de nouveaux formats d'affichage IV. Affichage des caractères et chaînes de caractères V. Solutions des exercices
4/5.4	Édition des programmes écrits en Forth
	I. Création d'un fichier de blocs II. Édition d'un bloc III. Résumé des commandes d'édition A. Commandes de blocs B. Commandes de curseur C. Manipulation de texte IV. Sélection d'un terminal
4/6	Travail en Assembleur 8080 sous CP/M 2.2 ou CP/M Plus
4/6.1	Les instructions du 8080

4/1

Locomotive BASIC : Définitions et rappels de base

4/1.1

pourquoi utiliser le Basic
et dans quels domaines ?

Pourquoi utiliser le Basic et dans quels domaines ?

Le langage BASIC a été créé en 1964 aux Etats-Unis.

Ses concepteurs ont voulu mettre à la portée de programmeurs débutants ou confirmés la plupart des possibilités de la machine sur laquelle le BASIC est implanté, non seulement au niveau algorithmique, mais également au niveau des commandes des ports d'entrée/sortie et des circuits spécialisés.

Le BASIC est composé de mots-clés qui sont interprétés ou compilés selon le type de BASIC et le type de machine utilisés.

Un langage est dit « interprété » quand les instructions qui composent ses programmes sont traduites en langage exécutable au moment de l'exécution du programme.

Un langage est dit « compilé » quand la phase de traduction est faite en dehors de l'exécution, et avant celle-ci : le compilateur fabrique un programme en code objet (code machine) directement exécutable par l'ordinateur. L'avantage est évident : un langage compilé est plus rapide qu'un langage interprété puisque la traduction est faite avant l'exécution. Le gain de temps peut être plus ou moins important en fonction de la qualité du compilateur.

Sur les ordinateurs AMSTRAD, le BASIC est interprété, mais cependant assez rapide. Par contre, le TURBO PASCAL est compilé et va beaucoup plus vite que le BASIC. Si vous désirez des temps d'exécution très courts (par exemple, pour réaliser des jeux d'arcades), utilisez de préférence le TURBO PASCAL (voir Partie 4).

Le BASIC de l'AMSTRAD CPC 464 comporte 154 mots-clés répartis dans des commandes classiques : opérations arithmétiques et logiques, tests, boucles, affichage et acquisition de texte, mais aussi dans des commandes d'affichage en haute résolution, des commandes destinées au circuit sonore, à l'imprimante ou au port d'entrées/sorties.

Ce BASIC est donc très complet, et, malgré tout, facile à apprendre. Si vous vous contentez d'utiliser les instructions d'ordre général (non graphiques, non sonores et ne concernant aucun circuit spécialisé), les pro-

grammes écrits seront relativement portables, et vous pourrez facilement les implanter sur d'autres machines.

Le BASIC a donc beaucoup d'avantages. Nous vous invitons à le découvrir plus en détails.

le Basic
tomaines

Le langage BASIC a été créé en 1964 aux Etats-Unis.

Des concepteurs ont voulu mettre à la portée de programmeurs débutants un langage simple et agréable à utiliser. Ils ont donc créé le langage BASIC. Ce langage est devenu très populaire et est aujourd'hui utilisé sur de nombreuses machines.

le
BASIC
est
un
langage
de
programmation

qui
permet
de
faire
des
calculs
et
de
stocker
des
données

sur
une
machine
à
calcul

et
de
faire
des
dessins

4/1.2

Il s'agit de la version 1.0 de Locomotive BASIC sur CPC 464.

Version 1.0 sur CPC 464 : Mots clés et leur utilisation

En quelques années, le langage BASIC a énormément progressé. La version LOCOMOTIVE BASIC 1.0 de l'AMSTRAD CPC 464 est de loin différente du BASIC que l'on trouvait sur les systèmes Apple ou Commodore des années 80.

Aussi nous a-t-il semblé important de montrer l'utilisation des mots-clés de ce langage. Pour cela, nous avons réparti chaque instruction dans les groupes suivants :

I. Affichage sur l'écran :

BORDER, CLS, INK, LOCATE, MODE, PAPER, PEN, POS, PRINT, PRINT USING, SPEED INK, TAG, TAGOFF, VPOS, WRITE, ZONE.

II. Gestion du clavier :

INKEY, INKEY\$, INPUT, JOY, KEY, KEY DEF, LINE INPUT, SYMBOL, SYMBOL AFTER, SPEED KEY.

III. Gestion de l'unité de cassette :

CAT, CHAIN, CHAIN MERGE, CLOSEIN, CLOSEOUT, OPENIN, OPENOUT, SAVE, EOF, LOAD, MERGE, SPEED WRITE.

IV. Manipulation de chaînes de caractères :

ASC, CHR\$, INSTR, LEFT\$, LEN, LOWER\$, MID\$, RIGHT\$, SPACES, STR\$, STRING\$, UPPER\$, VAL.

V. Gestion de données :

DATA, READ, RESTORE.

VI. Sous-programmes et branchements :

GOSUB, GOTO, ON GOSUB, ON GOTO, RETURN.

VII. Gestion des erreurs :

ERR, ERL, ERROR, ON ERROR GOTO, RESUME.

VIII. Gestion de la mémoire :

CALL, ERASE, FRE, HIMEM, INP, MEMORY, OUT, PEEK, POKE, VARPTR.

IX. Gestion des interruptions :

AFTER, DI, EI, EVERY, REMAIN, TIME.

X. Instructions musicales :

ENT, ENV, RELEASE, SOUND, SQ, ON SQ GOSUB.

XI. Instructions graphiques :

CLG, DRAW, DRAWH, MOVE, MOVER, ORIGIN, PLOT, PLOTR, TEST, TESTR, VPOS, XPOS, YPOS.

XII. Fonctions à caractère mathématique :

ABS, AND, ATN, BIN\$, CINT, COS, CREAL, DEF FN, DEG, EXP, FIX, HEX\$, INT, LOG, LOG10, MAX, MIN, NOT, OR, PI, RAD, RANDOMIZE, RND, ROUND, SGN, SIN, SQR, TAN, UNT, XOR.

XIII. Aides à la programmation :

AUTO, CLEAR, CONT, DEFINT, DEFREAL, DEFSTR, DELETE, EDIT, END, LIST, NEW, REM, RENUM, RUN, STOP, TRON, TROFF.

XIV. Divers :

DIM, FOR, IF, LET, NEXT, WAIT, WEND, WHILE, WIDTH, WINDOW, WINDOW SWAP.

Dans la suite, nous allons brièvement décrire la ou les fonction(s) de chaque mot-clé (pour plus de détails, reportez-vous au guide de l'utilisateur AMSTRAD CPC), et surtout donner des exemples d'utilisation de ces mots-clés.

Conventions d'écriture :

Pour faciliter la description systématique des mots-clés du BASIC, nous avons adopté les conventions d'écriture suivantes :

- les mots-clés sont écrits en majuscules ;
- les caractères « < » et « > » encadrent un argument obligatoire ;
- les caractères « [» et «] » encadrent un argument facultatif ;
- les caractères « (» et «) » indiquent que le mot-clé est une fonction. Ces caractères doivent être présents dans son utilisation ;
- enfin, l'« opérateur » désigne l'utilisateur du programme.

I. Affichage sur l'écran**BORDER <a>[,]**

Change la couleur du pourtour de l'écran :

« BORDER a » donne une couleur « a » fixe au pourtour.

une ligne lue et en
une couleur différente.

Utilisation

* **BORDER a,b** * donne une couleur * a * alternant avec une couleur * b * au pourtour. La vitesse d'alternance est déterminée par l'ordre * **SPEED INK** *.

Vous voulez faire défiler toutes les couleurs possibles du **BORDER**, et appuyer sur une touche pour passer d'une couleur à la suivante :

```
100 FOR I=0 TO 31
110 CLS:PRINT"BORDER";:BORDER I
120 A$=INKEY$:IF A$="" THEN 120 'Attente appui sur une touche
130 NEXT I
```

Ligne 110 : Modification du * **BORDER** *.

Ligne 120 : Attente de l'appui sur une touche.

Autre utilisation

Pendant l'exécution d'un programme, une erreur survient. Vous, programmeur, voulez attirer l'attention de l'opérateur (utilisateur). Un moyen simple pour y arriver est de faire clignoter le **BORDER**.

Supposons que le **BORDER** à l'état normal soit **BORDER 0**:

```
100 SPEED INK 5,5 'Vitesse de clignotement
110 BORDER 2,6 'BORDER clignotant
120 FOR I=1 TO 2000:NEXT I 'Durée du clignotement
130 BORDER 0 'Retour au BORDER d'origine
```

Ligne 100 : Choix de la vitesse de clignotement

Ligne 110 : Deux couleurs pour le **BORDER** : Bleu vif (2) et Rouge vif (6).

Ligne 120 : Durée d'affichage du **BORDER** clignotant.

Ligne 130 : Retour au **BORDER** normal.

une ou deux pages.

CLS[N° canal]

Efface l'écran ou la fenêtre définie par * **WINDOW** * avec la couleur du papier (**PAPER** ou **INK 0**).

Utilisation

Vous êtes en **MODE 1** et vous utilisez tout l'écran. Un programme affiche des données numériques sur l'écran, l'une au-dessous de l'autre, et il n'y a pas assez de lignes pour afficher toutes les données. Pour faciliter la lecture de ces données, vous décidez :

- 1) d'arrêter l'affichage dès que la ligne 23 est atteinte,
- 2) d'afficher un message d'attente.
- 3) d'attendre l'appui sur une touche pour continuer l'affichage.

```
100 'Calcul et affichage d'une donnée ou fin de programme si toutes les données sont affichées
```

```
110 IF VPOS(#0)<>23 THEN 100 'Ligne 23 pas encore atteinte
```

```
120 PRINT "Appuyez sur une touche pour voir la suite"
```

130 A\$ = INKEY\$:IF A\$ = "" THEN 130 'Attente appui sur une touche

140 CLS:GOTO 100 'Effacement écran et poursuite affichage

Ligne 100 : Cette ligne matérialise le programme d'affichage de données.

Ligne 110 : Test de la position du curseur. Si la ligne 23 n'est pas atteinte, affichage d'une autre donnée.

Ligne 120 : Message signalant qu'il reste des données à afficher.

Ligne 130 : Attente d'une action au clavier.

Ligne 140 : Initialisation du prochain écran d'affichage.

Autre utilisation

Vous avez défini une fenêtre de texte, par exemple dans un jeu d'aventures, et vous voulez afficher un nouveau texte dans cette fenêtre en effaçant le précédent :

100 WINDOW 1,40,22,25 'Definition de la fenetre de texte

110 'Programme de jeu

120 CLS 'Curseur en x,y

130 PRINT "Affichage du nouveau texte"

Ligne 100 : Définition d'une fenêtre (voir mot-clé « WINDOW » Partie 4 chap. 1.2 p. 105).

Ligne 110 : Cette ligne matérialise le corps du programme de jeu.

Ligne 120 : Effacement d'écran. Curseur en haut à gauche.

INK <a>, b[, <c>]

Définit la couleur « b » fixe d'un stylo « a », ou définit les couleurs alternantes « b » et « c » d'un stylo « a ».

Utilisation

Animation d'une rivière par rotation de la palette de couleurs :

Vous êtes en MODE 0, et vous avez dessiné une rivière avec les quatre couleurs bleues que possède l'AMSTRAD. Ces couleurs ont été utilisées en PEN 1, 2, 3 et 4. Il s'agit des couleurs :

1 : Bleu, 2 : Bleu Vif, 11 : Bleu Ciel, 14 : Bleu Pastel.

Une animation très réaliste peut être obtenue en opérant une rotation de la palette des bleus, à condition bien sûr que le reste du décor n'utilise pas ces couleurs.

Rotation
des
INK

Phase 1	Phase 2	Phase 3	Phase 4
1 → 2	1 → 11	1 → 14	1 → 1
2 → 11	2 → 14	2 → 11	2 → 2
11 → 14	11 → 1	11 → 2	11 → 11
14 → 1	14 → 2	14 → 11	14 → 14

```

100 A = 1:B = 2:C = 11:D = 14 'Définition des couleurs à modifier
110 INK 1,A:INK 2,B:INK 3,C:INK 4,D 'Rotation des bleus
120 I = A:A = B:B = C:C = D:D = I 'Définition des rotations
130 FOR J = 1 TO 100:NEXT J 'Attente entre deux rotations
140 GOTO 110 'Passage à la prochaine rotation

```

Ligne 100 : Les couleurs à modifier sont rangées dans les variables A, B, C et D.

Ligne 110 : Affectation des encres aux stylos 1, 2, 3 et 4.

Ligne 120 : Rotation des encres.

Ligne 130 : Pause entre deux rotations.

Autre utilisation

Vous avez dessiné un château délabré. Il fait nuit, et, pour rendre les lieux encore plus lugubres, vous décidez d'illuminer certaines parties du château lorsque le tonnerre fait rage.

Supposons que le dessin soit en MODE 0, que le fond de l'écran soit noir et que l'éclair utilise PEN 6.

Pour obtenir l'effet souhaité :

```

100 MODE 0
110 FOR J = 1 TO 3 '3 eclairs successifs
120 INK 6,26 'On fait apparaitre l'eclair
130 INK 0,13 'Le fond de l'ecran devient blanc
140 FOR I = 1 TO 100*INT (RND(1)*10):NEXT I 'Duree de l'eclair
150 INK 6,0 'L'eclair disparaît
160 INK 0,0 'et le fond redevient noir
170 FOR I = 1 TO 100:NEXT I 'Duree entre deux eclairs
180 NEXT J

```

Ligne 120 : Affectation de l'encre « Blanc brillant » 26 au stylo 6 ; l'éclair apparaît.

Ligne 130 : Affectation de l'encre « Blanc » 13 au stylo 0 (PAPER).

Ligne 150 : Affectation de l'encre « Noir » 0 au stylo 6 ; l'éclair disparaît.

Ligne 160 : Affectation de l'encre « Noir » 0 au stylo 0 ; le PAPER redevient noir.

LOCATE[<N° de canal>], <Coord. X>, <Coord. Y>

Déplace le curseur texte (l'endroit où le prochain caractère sera affiché par une instruction du type PRINT, INPUT ou équivalent) à une position quelconque sur l'écran.

Utilisation

Ce mot-clé permet de définir et d'utiliser facilement des masques de saisie. Prenons l'exemple suivant : vous voulez constituer le fichier de vos amis et relations et, pour cela, utiliser des écrans de saisie standard de type :

Nom :
Prénom :
Adresse :
Téléphone :
Première rencontre :
Divers :

Le texte « Nom, prénom ... divers » représente les **données fixes** de l'écran de saisie et est affiché par l'ordinateur.

Appelons chacune de ces données un « champ ».

L'utilisateur devra entrer des données après chaque champ.

Le programme de saisie sera le suivant :

```

100 CLS
110 PRINT "Nom : "
120 PRINT "Prénom : "
130 PRINT "Adresse : "
140 PRINT "Telephone : "
150 PRINT "Premiere rencontre : "
160 PRINT "Divers : "
170 LOCATE 8,1:INPUT NOM$ "Saisie champ 1
180 LOCATE 11,2:INPUT PRES "Saisie champ 2
190 LOCATE 12,3:INPUT ADR$ "Saisie champ 3
200 LOCATE 14,4:INPUT TEL$ "Saisie champ 4
210 LOCATE 23,5:INPUT PRR$ "Saisie champ 5
220 LOCATE 11,6:INPUT DIV$ "Saisie champ 6

```

Lignes 110 à 160 : Affichage des champs

Lignes 170 à 220 : Positionnement du curseur et lecture des données après chaque champ.

Autre utilisation

Dans le jeu du pendu, il faut trouver un mot en un minimum d'essais en proposant des lettres au hasard. Supposons que le mot à trouver soit « ORDINATEUR » et que l'écran de jeu soit le suivant :

Mot à trouver : -----
Lettre proposée : -

La gestion de l'affichage se fera comme suit :

```

100 MODE 1
110 mot$ = "ORDINATEUR" 'Mot à trouver
120 NL = 0 'Nombre de lettres trouvées
130 PRINT "Mot à trouver : -----"
140 LOCATE 1,10:PRINT "Lettre proposée : -"
150 LOCATE 17,10:INPUT L$ 'Entrée d'une lettre
160 FOR I=1 TO LEN (MOT$)
170 IF MID$(MOT$, I, 1)=L$ THEN LOCATE I+16, 1:PRINT L$:NL(I)=1
180 NEXT I
190 NL=0:FOR I=1 TO LEN (MOT$):NL = NL + NL(I):NEXT
200 IF NL=LEN(MOT$) THEN END 'Fin du jeu, mot trouvé
210 GOTO 140 'Boucle de jeu

```

Ligne 140 : Positionnement du message « lettre proposée »
Champ d'entrée.

Ligne 150 : Positionnement du curseur après le champ d'entrée.

Ligne 160 à 190 : Corrélation entre lettre entrée et mot à trouver.

MODE <n> (où n = 0, 1 ou 2)

Définit le mode d'affichage : résolution et nombre de couleurs affichées en même temps sur l'écran.

Remarque :

L'écran est effacé avec la couleur du « PAPER » ; la fenêtre, si elle existe, est détruite et le curseur texte se trouve en haut et à gauche de l'écran après l'emploi d'un tel ordre.

Utilisation

Dans une application scientifique, vous avez défini une fenêtre graphique dans laquelle sont tracées des courbes. Ces courbes sont commentées par un texte affiché autour de la fenêtre.

L'application étant terminée, vous voulez revenir au mode d'affichage standard (MODE 1) et effacer les dernières informations affichées.

Le programme suivant réalisera ces actions :

```

100 'Définition de la fenêtre graphique
110 'Calculs et affichages divers
300 INPUT "Fin de l'application (O/N)";R$
310 IF R$ <> "O" THEN 110 'Poursuite de l'application
320 MODE 1 'Retour au mode caractère pleine page
330 END 'Fin de l'application

```

Ligne 320 : Destruction de la fenêtre et passage au MODE d'affichage 40 colonnes.

Autre exemple :

Dans un jeu d'aventures, après un écran de présentation en MODE 0 (16 couleurs), vous voulez afficher un texte pour préciser le contexte de l'aventure. Si ce texte est long, il aura intérêt à être affiché en MODE 2.

Ce qui se traduit par le programme suivant :

```
100 MODE 0:LOAD "image", &C000 'Chargement de l'écran de présentation
```

```
110 MODE 2 'Effacement de l'image et passage en 80 colonnes
```

```
120 'Affichage du texte de l'aventure
```

Ligne 100 : Chargement de l'écran de présentation.

Ligne 110 : Effacement de l'écran de présentation et passage au MODE texte 80 colonnes.

PAPER [<N° de canal>,]<encre>

Définit la couleur de fond d'écran ou de fond de fenêtre.

Utilisation

Vous voulez connaître les possibilités de couleur de fond d'écran dans les trois MODES de résolution.

Ce programme pourra vous aider dans votre étude :

```
100 MODE 0
```

```
110 FOR I=0 TO 15 '16 Couleurs possibles en MODE 0
```

```
120 PAPER I:PEN 15-I:CLS:PRINT "PAPER" ; I
```

```
130 FOR J=1 TO 500:NEXT J 'Temporisation
```

```
140 NEXT I
```

```
150 MODE 1
```

```
160 FOR I=0 TO 3 '4 Couleurs possibles en MODE 1
```

```
170 PAPER I:PEN 3-I:CLS:PRINT "PAPER" ; I
```

```
180 FOR J=1 TO 500:NEXT J 'Temporisation
```

```
190 NEXT I
```

```
200 MODE 2
```

```
210 FOR I=0 TO 1 '2 Couleurs possibles en MODE 2
```

```
220 PAPER I:PEN 1-I:CLS:PRINT "PAPER" ; I
```

```
230 FOR J=1 TO 500:NEXT J 'Temporisation
```

```
240 NEXT I
```

Ligne 120 : PAPER en MODE 0.

Ligne 170 : PAPER en MODE 1.

Ligne 220 : PAPER en MODE 2.

Remarque :

La déclaration d'un nouveau PAPER affecte l'écran à partir de la position courante du curseur et jusqu'à la fin de l'écran, sur tout le texte affiché.

Pour vous en rendre compte, faites en MODE 1, par exemple : « PAPER 2 ». Le mot « Ready » apparaît sur un fond correspondant à la couleur d'encre 2. Pour que tout l'écran prenne cette couleur, il faudra faire : « PAPER 2:CLS ».

Syntaxe :

PEN {<N° de canal> , }<encre>

Utilisation

Définit la couleur d'encre utilisée pour écrire du texte sur l'écran.

Vous voulez connaître les possibilités de couleur d'écriture dans les trois MODES de résolution.

Ce programme pourra vous aider dans vos recherches.

100 MODE 0

110 FOR I=0 TO 15

120 PEN I

130 READ A\$:PRINT A\$

140 NEXT I

150 DATA Bleu, Jaune vif, Turquoise vif, Mauve, Blanc brillant, Noir

160 DATA Bleu vif, Magenta vif, Turquoise, Jaune, Bleu pastel, Rose

170 DATA Vert vif, Vert pastel, « Bleu/Jaune vif », « Rose/Bleu ciel »

180 '

181 A\$=INKEY\$:IF A\$="" THEN 181

190 MODE 1

200 FOR I=0 TO 3

210 PEN I

220 READ A\$:PRINT A\$

230 NEXT I

240 DATA Bleu, Jaune vif, Turquoise vif, Rouge vif

250 '

251 A\$=INKEY\$:IF A\$="" THEN 251

260 MODE 2

270 FOR I=0 TO 1

280 PEN I

290 READ A\$:PRINT A\$

300 NEXT I

310 DATA Bleu, Jaune vif

Lignes 120, 210 et 280 : Choix d'un stylo en MODES 0, 1 et 2.

Lignes 130, 220 et 290 : Affichage de la couleur du stylo en MODES 0, 1 et 2.

Autre application

Vous voulez faire ressortir un mot en rouge dans un texte écrit en MODE 1 en blanc sur fond noir :

100 INK 0,0:INK 1,13:INK 2,3 'Fond noir, Coul. 1=Blanc, Coul. 2=Rouge

110 CLS:PEN 1:PRINT "... Texte ..."

120 PRINT "... Texte ..." ; :PEN2:PRINT "Mot à détacher" ;

130 PEN 1:PRINT "... suite ..."

Ligne 100 : Fond noir, encre 1 blanche, encre 2 rouge.

Ligne 120 : Affichage du message « **Mot à détacher** » en rouge.

Ligne 130 : Retour à la couleur normale PEN 1.

POS (<#N° de canal>)

Donne la position du curseur texte par rapport à la position d'origine de la fenêtre, ou donne la position du chariot sur l'imprimante.

Utilisation

Dans un masque de saisie, vous voulez limiter la longueur d'un champ alphanumérique à N caractères.

Téléphone :

100 PRINT "Telephone" ;

110 A\$=INKEY\$:IF A\$ = "" THEN 110 'Attente appui sur une touche

120 PRINT A\$; :IF POS (#0)<>11+8 THEN 110

130 SOUND 1,100:END 'BEEP de débordement et fin de programme

Ligne 120 : POS (#0) = 11 + 8 (1 = chiffre affiché colonne 11, et longueur d'affichage = 8)

Autre utilisation

Vous avez créé un programme de traitement de textes, et vous voulez limiter la largeur d'affichage à N caractères.

100 A\$=INKEY\$:IF A\$="" THEN 100

110 IF POS (#0) <>60 THEN PRINT A\$; ELSE PRINT:PRINT A\$;
'A la ligne

120 GOTO 100 'Suite de la saisie

Ligne 110 : Si la position du curseur est inférieure à 60, le caractère est affiché, sinon passage à la ligne et affichage du caractère.

PRINT[<N° de canal> .]<Liste à imprimer>

Impression de données alphanumériques sur un canal :

#0 ou rien pour l'écran,

#8 pour l'imprimante.

Remarques :

- Le caractère « , » agit comme tabulateur entre deux affichages successifs.
- Le caractère « ; » interdit le retour à la ligne en fin d'affichage et fait afficher les données les unes à la suite des autres.
- La commande « SPACE\$(N) » permet d'afficher N espaces.
- La commande « TAB(N) » permet de déplacer le curseur vers la droite de N positions.

Utilisation

Vous voulez constituer une table trigonométrique sur imprimante :

100 DEG 'On manipule des degrés

110 PRINT #8, "Angle SIN COS TAN"

120 FOR I=1 TO 360

130 PRINT #8, I ; SIN(I) ; COS(I) ; TAN(I)

140 NEXT I

Ligne 130 : Affichage sur imprimante (Canal #8).

PRINT[<N° Canal> , [USING <option>] <liste à imprimer>]

L'option « USING » du « PRINT » permet de normaliser l'affichage numérique ou alphanumérique sur écran ou imprimante.

Les options peuvent être les suivantes :

- # indique l'emplacement d'un chiffre.
- . indique la position du point décimal.
- , positionné devant le point décimal, il indique que les chiffres précédant le point seront regroupés par groupes de trois et séparés entre eux par une virgule.
- ! Le signe ! apparaîtra avant le premier chiffre ou le point décimal.
- * Les espaces non utilisés devant le nombre seront remplis par des astérisques.
- *! Cumule les options * et !.
- \$ Le signe \$ apparaîtra avant le premier chiffre ou le point décimal.
- *\$ Cumule les options * et \$.

- + Le signe du nombre sera apparent.
- Le signe est positionné derrière le nombre.
- A Indique que le nombre doit apparaître suivi d'un exposant décimal.

Utilisation

Vous voulez constituer une table trigonométrique sur imprimante, où les SINus, COSinus et TANgente ont 4 chiffres après la virgule :

100 DEG 'On manipule des degres

110 PRINT #8, "Angle SIN COS TAN"

120 FOR I = 1 TO 360

130 PRINT #8, I;

**140 PRINT #8, " " ; :PRINT #8, USING "#.####" ; SIN(I) ;
:PRINT #8, " " ;**

150 PRINT #8, USING "#.####" ; COS(I) ; :PRINT #8, " " ;

160 PRINT #8, USING "#.####" ; TAN(I)

170 NEXT I

Lignes 130 à 160 : Affichage sur imprimante de données réelles au format 1 chiffre avant la virgule, et 4 après (#.####).

SPEED INK < Nombre entier > , < Nombre entier >

Fixe la fréquence d'alternance des couleurs utilisées par les ordres PEN, PAPER ou BORDER définis par deux couleurs. Le premier nombre donne la durée d'affichage de la première couleur en 1/50 sec., le deuxième nombre donne la durée d'affichage de la deuxième couleur en 1/50 sec.

Utilisation

Dans un écran de saisie comportant plusieurs champs, vous voulez signaler à l'opérateur que l'ordinateur est en attente du champ N

Libellé 1 :	-----
Libellé 2 :	-----
Libellé 3 :	-----
Libellé 4 :	-----

Les libellés qui précèdent les champs sont de couleur fixe. Vous allez faire clignoter l'un des libellés pour indiquer le champ attendu

Par exemple, pour le champ 3 :

100 MODE 1:INK 2,2,6 'Positionnement de l'encre 2

110 SPEED INK 10,10 'Vitesse de clignotement

120 LOCATE 10,10:PEN 2:PRINT "Libelle 3" ; :PEN 1:INPUT C3\$

130 LOCATE 10,10:PRINT "Libelle 3"

Ligne 110 : Affichage du libellé 3 en PEN 2 et attente d'une réponse de l'utilisateur.

Ligne 120 : La réponse entrée, le libellé est réécrit en PEN 1.

Autre utilisation

Dans un jeu d'arcades, vous voulez faire une animation simple sur un mobile avant de le faire exploser. Par exemple, vous êtes en MODE 0, et la couleur 12 n'est utilisée que par ce mobile.

```

100 INK 12,2,8 'Couleurs du clignotement
110 FOR I = 15 TO 1 STEP - 5
120  SPEED INK I,I
130  FOR J = 1 TO I*100:NEXT J 'Boucle d'attente de clignotement
140 NEXT I

```

Ligne 100 : Définition des couleurs de clignotement.

Ligne 120 : Evolution de la vitesse de clignotement dans le temps.

Ligne 130 : Clignotement à la même cadence.

TAG[< # N° Canal >]

Permet d'obtenir un positionnement graphique (au pixel près) du curseur texte.

Utilisation

Vous voulez afficher du texte sur la périphérie d'un cercle, deux affichages consécutifs étant très rapprochés.

```

100 MODE 2
110 FOR I = 1 TO 360 STEP 3 'Pas de l'affichage
120  MOVE 270 + 190 * COS(I), 200 + 190 * SIN(I)
130  TAG:PRINT "CPC 464"
140 NEXT I

```

Ligne 120 : Positionnement du curseur graphique sur la périphérie d'un cercle.

Ligne 130 : Positionnement du curseur texte sur le curseur graphique et affichage.

Autre utilisation

Vous voulez graduer les axes d'un repère Oxy :

```

100 MODE 1
110 MOVE 60,20:DRAW 570,20 'Axe Ox
120 MOVE 60,20:DRAW 60,390 'Axe Oy
130 FOR I = 1 TO 10 '10 graduations sur Ox
140  MOVE 60 + I * 50,10
150  DRAW 0,20
160 NEXT I
170 FOR I = 1 TO 10 '10 graduations sur Oy
180  MOVE 50,20 + I * 36
190  DRAW 20,0
200 NEXT I
210 FOR I = 1 TO 10 'Annotations sur Ox

```

```

220 MOVE I*50-3+40, 15
230 TAG:PRINT I ;
240 NEXT I
250 FOR I=1 TO 10 'Annotations sur Oy
260 MOVE -10,25+I*35
270 TAG:PRINT I ;
280 NEXT I

```

Lignes 110-120 : Tracé des axes.

Lignes 130-200 : Graduations Ox et Oy.

Lignes 220 et 260 : Positionnement du curseur graphique pour annotation.

Lignes 230 et 270 : Positionnement du curseur texte et annotations.

TAGOFF(< #N° de Canal>)

Positionne le curseur à l'endroit où il était avant la commande TAG.

VPOS (< #N° Canal>)

Donne la position verticale du curseur pour le canal demandé.

WRITE [< #N° Canal>,]<Liste à écrire>

Ecrit des expressions ou des valeurs sur un canal :

Canal 8 : Imprimante

Canal 9 : Cassette/Disquette

Remarque :

Les chaînes doivent être entre guillemets et deux expressions doivent être séparées par une virgule.

Utilisation

Ecriture d'un fichier sur cassette :

```

100 NOM$ = "Dupond"
110 PRE$ = "Pierre"
120 AD$ = "20 rue Gerard, 75013 Paris"
130 TEL$ = "Inconnu"
140 OPENOUT"FILEDUP" 'Ouverture du fichier en écriture
150 WRITE#9, NOM$, PRE$, AD$, TEL$ 'Ecriture des données
160 CLOSEOUT 'Fermeture du fichier

```

Lignes 100-130 : Articles à l'écran.
Ligne 150 : Ecriture.

ZONE <Nombre entier>

Change la largeur de l'affichage à l'écran ou d'impression sur imprimante lors de l'utilisation de l'ordre « PRINT ».

Remarque :

L'effet de « ZONE » est annulé par les ordres « NEW », « LOAD », « CHAIN » et « RUN ».

Utilisation:

Vous écrivez un programme de traitement de textes, et vous voulez une sortie sur imprimante sur 80 colonnes. Une façon élégante de résoudre le problème est la suivante :

100 MODE 2 'Passage en mode 80 colonnes

110 ZONE 60 'Restriction à 60 colonnes

II. Gestion du clavier**INKEY (< Nombre entier >)**

Détermine si pour une touche particulière du clavier les fonctions [SHIFT] et [CONTROL] sont activées.

En sortie, la fonction INKEY renvoie les valeurs suivantes selon la position des touches CTRL et SHIFT.

Sortie	SHIFT	CTRL	Touche
-1	?	?	haute
0	haut	haut	basse
32	bas	haut	basse
128	haut	bas	basse
160	bas	bas	basse

Utilisation

Pour mieux comprendre le fonctionnement de la fonction INKEY, voici un programme qui indique la position des touches SHIFT et CTRL lorsque l'opérateur appuie sur la touche « Q » de code 67 (voir schéma du clavier page suivante).

100 CLS

110 IF INKEY(67) = -1 THEN PRINT "Touche Q non sollicitée" :GOTO 110

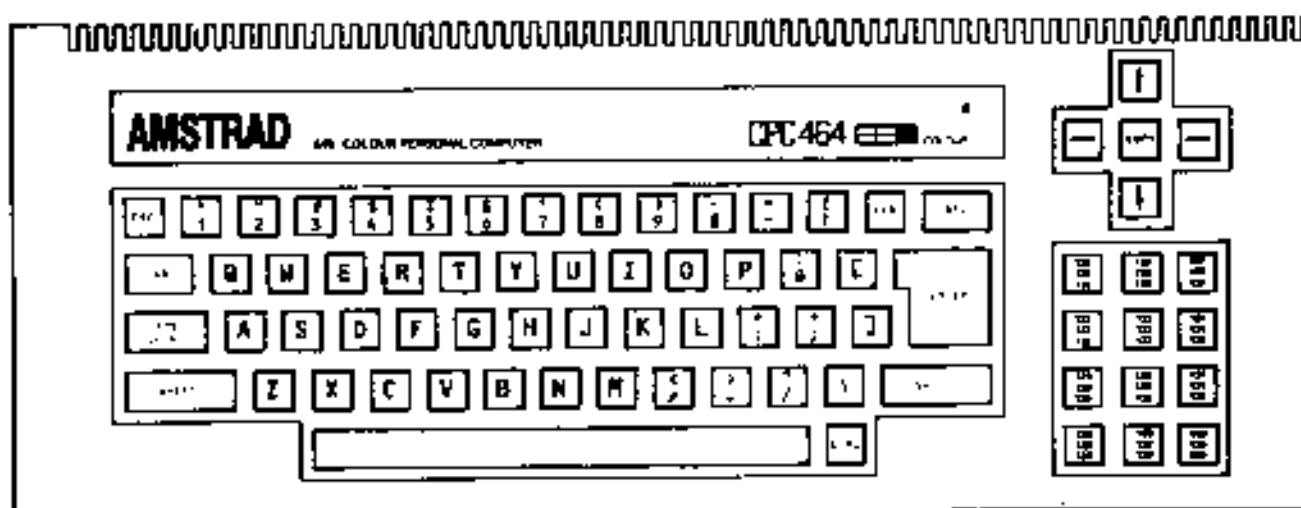
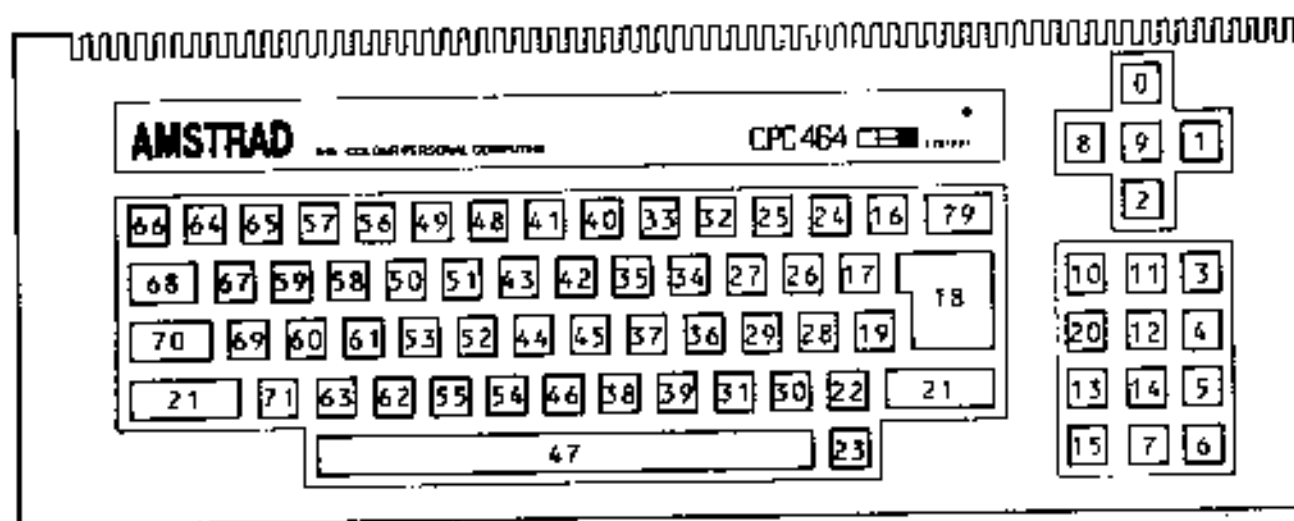
120 IF INKEY(67) = 0 THEN PRINT "Touche Q sollicitée"

130 IF INKEY (67) = 32 THEN PRINT "SHIFT + Q enfoncees"

140 IF INKEY {67} = 128 THEN PRINT "CTRL + Q enfoncees"

150 IF INKEY (67) = 160 THEN PRINT "SHIFT + CTRL + Q enfoncees"

160 GOTO 110



Autre utilisation

Dans une application de traitement de textes, nous pouvons prendre la convention suivante :

CTRL+Q = Sortie du logiciel,

CTRL+S = Sauvegarde du texte,

SHIFT+CTRL+S = Sauvegarde du texte et sortie du programme.

La touche « Q » a pour code 67, la touche « S » a pour code 60.

La détection de « CTRL+Q » se fait par `INKEY(67) = 128`,

la détection de « CTRL+S » se fait par `INKEY(60) = 128`, et

la détection de « SHIFT+CTRL+S » se fait par `INKEY(60) = 160`.

De ces constatations, nous pouvons écrire une partie de la boucle principale (Idle Loop) du traitement de textes :

```

100 A$ = INKEY$: IF A$ = "" THEN 100 'Attente d'un caractère
110 IF INKEY(67) = 128 THEN GOTO 1000 'Sortie du logiciel
120 IF INKEY(60) = 128 THEN GOTO 2000 'Sauvegarde du texte
130 IF INKEY(80) = 160 THEN GOTO 3000 'Sauvegarde et sortie
140 GOTO 100 'Bouclage de l'Idle loop

```

INKEY\$

Acquisition « volante » d'une touche du clavier.

Utilisation

Les deux domaines dans lesquels « INKEY » est souvent employée sont les suivants :

- 1) Attente de l'appui sur une touche, puis traitement ;
- 2) Insertion dans une boucle principale (« Idle Loop »).

1) Attente de l'appui sur une touche :

```
100 A$ = INKEY$: IF A$ = "" THEN 100
```

Ce programme reste en ligne 100 jusqu'à l'appui sur une touche. Dès lors, le code de la touche se trouve dans A\$. Ces codes seront affichables par « PRINT A\$ », ou non affichables par « PRINT A\$ » s'ils ont un code ASCII inférieur à 27. Dans ce deuxième cas, pour avoir leur valeur, il faudra faire :

A = ASC(A\$) ; PRINT A, et l'on obtiendra les valeurs suivantes :

Valeur de A	Touches pressées
0	CTRL + @
1	CTRL + A
2	CTRL + B
..	
26	CTRL + Z

2) Insertion d'« INKEY\$ » dans une boucle principale :

Par exemple, dans un jeu de « Space Invaders », le programme ne peut pas passer tout son temps à tester les commandes entrées au clavier. D'autres actions primordiales sont à exécuter quasiment en même temps. Il s'agit, par exemple, du déplacement des envahisseurs et des tirs de missiles, d'où le programme d'Idle loop ci-dessous.

```

100 'Idle loop
110 GOSUB DEP—ENV 'Déplacement envahisseurs
120 GOSUB TIR—MIS 'Tir des missiles
130 IF INKEY$ < > "" THEN GOSUB TRA—CLA 'Traitement clavier
160 GOTO 100 'Rebouclage de l'Idle loop

```

Ligne 130 : Exécution d'une commande clavier si le clavier est actionné.

INPUT[< # N° Canal > ,] [:] [< Chaîne > ;] < Liste de variables > , ou
INPUT[< # N° Canal > ,] [:] [< Chaîne > ,] < Liste de variables >

Lit des données en provenance du canal indiqué.

Remarque :

100 INPUT "Entrez un nombre";N :

↳ Supprime le passage à la ligne
 ↳ Un « ? » apparaît après le texte
 Remplacé par « , », aucun « ? »
 n'apparaît

Utilisation

Lecture de données numériques X, Y de points que l'on veut réunir par des droites. Les dernières coordonnées avant le tracé seront par convention 999, 999.

100 DIM X(100), Y(100) '100 points au maximum

110 I = 1 'Memorisation du premier point

120 INPUT "Entrez les coordonnées X et Y";X(I), Y(I)

130 IF X(I) < > 999 AND Y(I) < > 999 THEN I = I + 1 : GOTO 120

131 CLS

140 PLOT X(1), Y(1) 'Origine

150 FOR J = 2 TO I

160 DRAW X(J), Y(J) 'Trace

170 NEXT J

Ligne 120 : Entrée des coordonnées et stockage dans les tableaux X et Y.

Ligne 130 : Test de fin de saisie.

Lignes 131 à 170 : Affichage.

Autre application

Lecture de données sur cassette :

Reportez-vous à l'exemple donné pour l'ordre « WRITE ». On veut relire le fichier cassette ou disquette stocké dans cet exemple.

100 OPENIN "FILEDUP"

110 INPUT #9, NOM\$, PRE\$, AD\$, TEL\$

120 CLOSEIN

130 PRINT "Nom :"; NOM\$

140 PRINT "Prenom :"; PRE\$

150 PRINT "Adresse"; AD\$

160 PRINT "Telephone :"; TEL\$

Ligne 100 : Ouverture du fichier

Ligne 110 : Lecture des données (#9 = Cassette ou disquette).

Ligne 120 : Fermeture du fichier.

Lignes 130-160 : Affichage.

JOY(<Entier>)

Donne la position du manche et des boutons feu de la manette de jeu, selon le tableau ci-dessous :

Bit à 1	Position
0	Haut
1	Bas
2	Gauche
3	Droite
4	Feu 2
5	Feu 1

Utilisation

Pour se familiariser avec cette commande :

```

100 A=JOY(0)
110 IF A=0 THEN PRINT "JOYSTICK non active"
120 IF A AND 1 <> 0 THEN PRINT "Vers le haut"
130 IF A AND 2 <> 0 THEN PRINT "Vers le bas"
140 IF A AND 4 <> 0 THEN PRINT "Vers la gauche"
150 IF A AND 8 <> 0 THEN PRINT "Vers la droite"
160 IF A AND 16 <> 0 THEN PRINT "Bouton feu n° 2"
170 IF A AND 32 <> 0 THEN PRINT "Bouton feu n° 1"
180 GOTO 100

```

Ligne 100 : Acquisition de la position du joystick.

Lignes 110 à 170 : Affichage en fonction de la position du joystick

Ce programme reconnaîtra, par exemple, l'appui sur le bouton feu N° 2 et l'orientation du JOYSTICK vers la gauche en même temps grâce à l'utilisation de la fonction de test de bit « AND ».

Autre utilisation

Simulation d'une souris par JOYSTICK

La boucle principale aura l'allure suivante :

```

100 A=JOY(0)
110 IF (A AND 1) <> 0 THEN GOSUB HAUT 'Déplacement vers le haut
120 IF (A AND 2) <> 0 THEN GOSUB BAS 'Déplacement vers le bas

```

130 IF (A AND 4) <> 0 THEN GOSUB GAUCHE 'Déplacement vers la gauche

140 IF (A AND 8) <> 0 THEN GOSUB DROITE 'Déplacement vers la droite

150 GOTO 100 'Rebouclage de l'Idle loop

Ligne 100 : Acquisition de la position du joystick.

Lignes 110-140 : Activation des modules de gestion joystick.

KEY <Chaîne de caractères>

Affecte à une touche une chaîne de caractères (Longueur < 120).

Les touches de codes compris entre 128 et 159 (32 touches) sont redéfinissables.

Les codes 128 à 140 correspondent aux touches du pavé numérique :

Code 128 → Touche 0, Code 129 → Touche 1, Code 130 → Touche 2

Code 131 → Touche 3, Code 132 → Touche 4, Code 133 → Touche 5

Code 134 → Touche 6, Code 135 → Touche 7, Code 136 → Touche 8

Code 137 → Touche 9, Code 138 → point, Code 139 → ENTER

Code 140 → CTRL+ENTER.

Les codes 141 à 159 ne sont pas associés à des touches, mais peuvent l'être en utilisant la commande « KEY DEF ».

Utilisation

En tapant un programme BASIC, vous constatez que certains mots-clés reviennent souvent. Pour vous faciliter la tâche, redéfinissez des touches en leur affectant ces mots-clés répétitifs.

Par exemple :

Affectation de « INPUT » au « 0 » du pavé numérique :
KEY 128, "INPUT".

Affectation de « DATA » au « 1 » du pavé numérique :
KEY 129, "DATA".

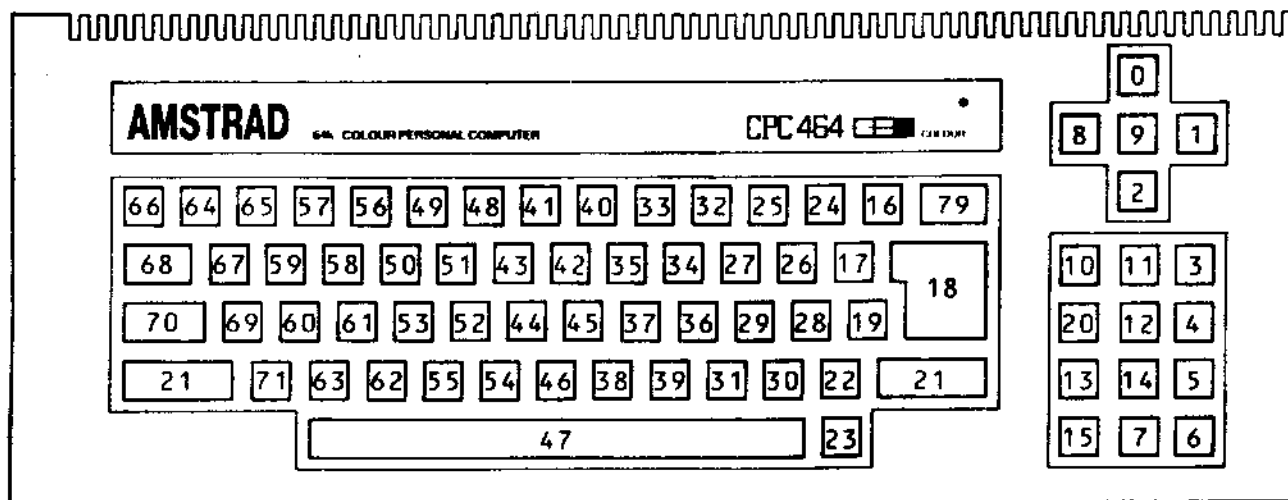
KEY DEF<N° touche>, <Répétition>, [<Normal>, [<avec SHIFT>, [<avec CTRL>]]]

Répétition : 0 ou 1 : Active ou désactive l'auto-repeat dont la vitesse est contrôlée par « SPEED KEY ».

Normal : Redéfinition s'il n'y a pas appui sur SHIFT ni CTRL.

SHIFT : Redéfinition s'il y a appui sur SHIFT.

CTRL : Redéfinition s'il y a appui sur CTRL.

**Application**

Vous voulez redéfinir votre clavier QWERTY en AZERTY, c'est-à-dire changer les touches suivantes :

QWERTY		AZERTY	
Sans SHIFT	Avec SHIFT	Sans SHIFT	Avec SHIFT
(67) q	Q	a (97)	A (65)
(59) w	W	z (122)	Z (90)
(69) a	A	q (113)	Q (81)
(29) :	*	m (109)	M (77)
(71) z	Z	w (119)	W (87)
(38) m	M	, (44)	? (63)
(39) ,	<	; (59)	. (46)
(31) .	>	: (58)	/ (47)
(30) /	?	= (61)	+ (43)

Ce qui se traduit par :

100 KEY DEF 67, 1, 97, 65 : KEY DEF 59, 1, 122, 90 : KEY DEF 69, 1, 113, 81

110 KEY DEF 29, 1, 109, 77 : KEY DEF 71, 1, 119, 87 : KEY DEF 38, 1, 44, 63

120 KEY DEF 39, 1, 59, 46 : KEY DEF 31, 1, 58, 47 : KEY DEF 30, 1, 61, 43

LINE INPUT [< #N° Canal>][;][< Chaîne>]< Variable en chaîne>

Lit une ligne sur le canal indiqué (par défaut sur l'écran).

Remarque :

Un « ; » après le « LINE INPUT » supprime le passage à la ligne en fin de lecture.

Utilisation

Vous voulez réaliser un éditeur de lignes avec une saisie simple gérant 100 lignes au maximum.

```
100 MODE 2
110 DIM A$(100)
120 PRINT "Entrez votre texte, puis 999 quand vous aurez fini."
130 I = 1 'Ligne 1
140 LINE INPUT A$(I)
150 IF A$(I) <> "999" THEN I = I + 1 : GOTO 140
160 PRINT "Voulez-vous 1) Corriger une ligne"
170 PRINT "                2) Imprimer"
180 INPUT "                3) Continuer la saisie" : A
190 IF A <> 1 THEN 240
200 INPUT "N° de ligne" : N 'Correction d'une ligne
210 PRINT A$(N)
220 LINE INPUT A$(N)
230 GOTO 160
240 IF A <> 2 THEN 290
250 FOR J = 1 TO I 'Impression
260 PRINT #8, A$(J)
270 NEXT J
280 GOTO 160
290 GOTO 140 'La saisie continue
```

Ligne 140 : Lecture d'une ligne.

Ligne 150 : Test de fin de saisie.

Lignes 160-180 : Menu de fin de saisie.

Lignes 190-230 : Correction d'une ligne.

Lignes 240-280 : Impression.

SYMBOL <N° caractère>, <8 lignes élémentaires>

Redéfinit la forme d'un caractère au pixel près.

Remarque :

Consulter le mot-clé « SYMBOL AFTER » qui va avec.

Editeur de caractères redéfinissables :

Utilisation

```

100 CLS:DIM T (8,8)
110 FOR I=1 TO 8
120 PRINT "....."
130 NEXT I
140 X=1:Y=1 'Initialisation
150 LOCATE X, Y
160 A$=INKEY$:IF A$="" THEN 160
170 A=ASC (A$)
180 IF A$="P" THEN T(Y,X)=1:PRINT "*"
190 IF A$="V" THEN T(Y,X)=0:PRINT "."
200 IF A=243 AND X<8 THEN X=X+1
210 IF A=242 AND X>1 THEN X=X-1
220 IF A=240 AND Y>1 THEN Y=Y-1
230 IF A=241 AND Y<8 THEN Y=Y+1
240 IF A<>13 THEN 150
250 LOCATE 1,15:PRINT "Donnees correspondantes :":PRINT
260 FOR I=1 TO 8
270   A=0 'Initialisation
280   FOR J=1 TO 8
290     A=A+(2^(8-J))*T(I, J)
300   NEXT J
310   PRINT A ;
320 NEXT I

```

Lignes 110-130 : Affichage de la grille de redéfinition.

Ligne 160 : Acquisition des commandes clavier.

Lignes 170-240 : Action sur une commande clavier.

Lignes 250-320 : Affichage des données de redéfinition.

Autre utilisation

Affichage de dessins définis par plusieurs caractères redéfinis mis côte à côte :

Soit la variable « HAUTEUR » définissant le nombre de caractères mis l'un sous l'autre, et « LARGEUR » le nombre de caractères mis côte à côte.

```
100 REM Affichage de motifs programmes
110 'Definition des couleurs d'encres et de papier
120 'Initialisation des coordonnees d'affichage
130 '
140 SYMBOL AFTER 129 'Definition du motif
150 FOR I= 1 TO HAUTEUR
160   FOR J= 1 TO LARGEUR
170     READ A(J)
180   NEXT J
190   SYMBOL 128+I, A(1), A(2), A(3), A(4), A(5), A(6), A(7), A(8)
200 NEXT I
210 'DATA correspondant aux symboles a redefinir
220 '
230 'Affichage du motif
240 FOR I= 1 TO HAUTEUR
250   FOR J= 1 TO LARGEUR
260     LOCATE OX+J-1, OY+I-1 'Positionnement du curseur
270     PRINT CHR$(129+(J-1)+(I-1)*LARGEUR)
280   NEXT J
290 NEXT I
300 END
```

Lignes 150 – 200 : Redéfinition des caractères.

Ligne 210 : Données de redéfinition.

Lignes 230-290 : Affichage du dessin.

SYMBOL AFTER <Entier N>

Fixe le nombre de caractères qui seront redéfinis par la commande « SYMBOL ».

Le nombre de caractères redéfinissables est $255 - N$, occupant les positions N à 255.

Par exemple, pour que les caractères 60 à 255 soient redéfinissables, taper : SYMBOL AFTER 60.

SPEED KEY <Attente>, <Période de répétition>

Définit la vitesse de répétition des touches en « auto-repeat » et le délai au bout duquel l'« auto-repeat » se déclenche en 1/50^e sec.

La valeur par défaut est 10,10 et ces deux nombres peuvent prendre des valeurs comprises entre 1 et 255.

III - Gestion de l'unité de cassette**CAT**

Donne la liste des fichiers présents sur cassette sans affecter un éventuel programme qui se trouve en mémoire.

Remarque :

Les caractères indicateurs suivants renseignent l'opérateur sur le type de fichier rencontré lors du CAT :

\$ Programme BASIC
% Programme BASIC protégé
* Fichier ASCII
& Fichier binaire

CHAIN <Nom programme>[, <N° ligne>] et CHAIN MERGE <Nom programme>[, <N° Ligne>] [, DELETE <ensemble de lignes>]

CHAIN charge un programme en effaçant celui qui était en mémoire. L'option <N° ligne> précise à quelle ligne doit commencer l'exécution.

Remarque :

CHAIN conserve la valeur des variables en mémoire, mais les fonctions définies par l'utilisateur et les fichiers ouverts sont oubliées ; quelques fonctions sont remises à zéro (reportez-vous au manuel d'utilisation AMS-TRAD pour plus de détails).

Utilisation

L'ordre « CHAIN » permet de structurer en fonctions indépendantes et successives les programmes qui ne tiennent pas en mémoire.

Supposons qu'un programme puisse être décomposé en trois fonctions séquentielles F1, F2 et F3, et que la totalité du programme ne tienne pas en mémoire. Chaque fonction pourra alors être exécutée sous forme d'un programme, F2 sera chaînée à F1 et F3 à F2. Le problème suivant se pose : F1 aura peut-être fait des calculs nécessaires à F2 et/ou F3 ; de même, F2 aura peut-être fait des calculs nécessaires à F3.

Deux solutions sont possibles :

- 1) Mettre les données ou calculs résultant d'une fonction et à destination d'une autre fonction dans un fichier.
- 2) Utiliser l'ordre « CHAIN MERGE ».

1) Passage de données par fichier :

Nous allons ouvrir un fichier séquentiel et y stocker les données à conserver puis utiliser l'ordre « CHAIN ». Supposons que les données à passer soient les suivantes : Chaîne A\$, Tableau A[1...6], Entier X%.

Le programme qui suit est à installer en fin de "F1". Il assurera la création du fichier de passage et le chaînage à "F2".

```
1000 OPENOUT "PASSAGE"
```

```
1010 WRITE #9, A$
```

```
1020 FOR I=1 TO 6
```

```
1030 WRITE #9, A[I]
```

```
1040 NEXT I
```

```
1050 WRITE #9, X%
```

```
1060 CLOSEOUT
```

```
1070 CHAIN "F2"
```

Ligne 1000 : Ouverture du fichier de transfert de données.

Lignes 1010-1050 : Ecriture des données à transférer.

Ligne 1060 : Fermeture du fichier de transfert.

Ligne 1070 : Chaînage au programme F2.

De plus, il faudra installer le programme suivant en début de « F2 ».

```
100 OPENIN "PASSAGE"
```

```
110 INPUT #9, A$
```

```
120 FOR I=1 TO 6
```

```
130 INPUT #9, A[I]
```

```
140 NEXT I
```

```
150 INPUT #9, X%
```

```
160 CLOSEIN
```

Ligne 100 : Ouverture du fichier de transfert.

Lignes 110-150 : Lecture des données.

Lignes 160 : Fermeture du fichier de transfert.

2) Utilisation de l'ordre « CHAIN MERGE » :

Supposons que "F1" occupe les lignes 1000 à 2000,
"F2" occupe les lignes 2010 à 3000,
et "F3" occupe les lignes 3010 à 4000.

En fin de "F1" : CHAIN MERGE "F2", 2010, DELETE 1000-2000
En fin de "F2" : CHAIN MERGE "F3", 3010, DELETE 2000-3000

CLOSEIN

Ferme un fichier ouvert en lecture sur cassette.

Remarque :

NEW et CHAIN MERGE ferment automatiquement tous les fichiers cassettes ouverts.

Utilisation

Voir CHAIN.

CLOSEOUT

Ferme un fichier ouvert en écriture sur cassette.

Remarque :

NEW et CHAIN MERGE ferment automatiquement tous les fichiers cassettes ouverts.

Utilisation

Voir CHAIN.

OPENIN < Nom du fichier >

Ouvre un fichier sur cassette dans le but de lire les informations qu'il contient.

OPENOUT < Nom du fichier >

Ouvre un fichier sur cassette dans le but d'y écrire des données.

Remarques concernant OPENIN et OPENOUT :

- Le « ! » supprime les messages d'entrée/sortie sur cassette de type « PRESS PLAY THEN ANY KEY » ou équivalent, et lance directement le moteur de l'unité de cassette.
- L'ordinateur AMSTRAD CPC stocke les données sur cassette par blocs de 2 kilo-octets.

SAVE < Nom Fichier > [, < Type >] [, < Paramètres binaires >]

Sauvegarde sur cassette un fichier BASIC, ASCII ou BINAIRE.

L'option Type définit la structure d'organisation des données sur cassette :

Non indiqué	Programme BASIC résident
,A	ASCII : peut être récupéré par un traitement de textes.
,P	Protégé : illisible et inexécutable après un LOAD. Ne peut être exécuté que par CHAIN ou RUN.
,B	Données binaires.

Remarque :

L'utilisation du type de données binaires (,B) oblige l'utilisateur à indiquer l'adresse de départ de la sauvegarde, la longueur (en nombre d'octets) de la sauvegarde, et éventuellement le point d'entrée du programme (si c'est un programme binaire qui est sauvegardé).

EOF

Teste la fin de fichier sur cassette.

Cette fonction booléenne vaut - 1 si la fin de fichier est atteinte, 0 sinon.

Utilisation

Lecture d'un fichier dont on connaît la structure mais pas le nombre d'enregistrements. Supposons que le fichier « DATA » ait la structure suivante :



La lecture d'un tel fichier et sa mémorisation se feront de la manière suivante :

Supposons que ce fichier comporte 100 articles au maximum.

```
100 I = 1: DIM NOM$(100), AGE(100), ADR$(100)
```

```
110 OPENIN "DATA"
```

```
120 INPUT #9, NOM$(I), AGE(I), ADR$(I)
```

```
130 IF EOF=0 THEN I=I+1: GOTO 120 'Poursuite lecture
```

```
140 CLOSEIN
```

```
150 'Suite du traitement
```

Ligne 110 : Ouverture du fichier.

Ligne 120 : Lecture des données.

Ligne 130 : Si toutes les données ne sont pas lues, lecture de l'enregistrement suivant.

Ligne 140 : Fermeture du fichier.

LOAD < Nom Fichier > [, Adresse]

Charge un programme BASIC ou un fichier binaire à partir de l'adresse donnée.

Remarque :

La commande « LOAD < Nom Fichier > » efface le programme BASIC résidant en mémoire.

Utilisation

Suite à la redéfinition des caractères standards par un programme d'édition de caractères, vous ne voulez pas réduire à néant vos efforts en atteignant la machine. Aussi décidez-vous de sauvegarder les données de redéfinition des caractères qui étaient stockées en mémoire, par exemple à partir de l'adresse hexa &H9000.

Exemple :

Vous avez redéfini 6 caractères, soit $8 * 6 = 48$ octets stockés à partir de &H9000.

Les caractères occupent donc les adresses &H9000 à &H902F. (En effet, $48D = &H30$). Ils seront sauvegardés par :

SAVE "CARACT", B, &H9000, &H30

Vous les récupérerez par :

LOAD "CARACT", &H9000

Remarque :

La commande « LOAD » destinée à charger un programme BASIC et insérée dans un programme BASIC ne lance pas le programme chargé. Pour faire cela, reportez-vous aux ordres « CHAIN » ou « CHAIN MERGE ».

MERGE < Nom Fichier >

Permet d'amalgamer deux programmes dont un est en mémoire et l'autre sur cassette.

Remarques :

a) Avant d'utiliser cette commande, assurez-vous que les deux programmes à amalgamer ne possèdent pas de numéros de lignes communs (au besoin, utilisez la commande « RENUM » à cet effet).

b) Un « ! » devant le nom du fichier à lire supprime tous les commentaires d'entrée/sortie cassette.

c) Les variables, fonctions utilisateur et dossiers ouverts sont écrasés ou abandonnés.

Pour plus de détails, consultez les commandes « CHAIN », « CHAIN MERGE » ou le manuel utilisateur AMSTRAD.

SPEED WRITE <0 ou 1>

Définit la vitesse d'écriture sur unité de cassette :

0 = 1 000 Bauds (option par défaut),

1 = 2 000 Bauds (si précisé).

Si vous n'êtes pas sûr de l'azimutage de la tête de lecture de votre magnétophone, ou si la qualité des cassettes utilisées laisse à désirer, utilisez la vitesse lente « SPEED WRITE 0 ».

Remarque :

Si vous utilisez un CPC 464 avec une qualité de bande moyenne, la vitesse rapide fonctionnera neuf fois sur dix. Le problème n'est plus le même si vous utilisez un 664 ou un 6128 avec un lecteur de cassettes externe de qualité courante...

Utilisation

Sauvegarde d'un fichier binaire en vitesse rapide :

1000 SPEED WRITE 1 'Vitesse rapide

1010 SAVE "DATA", B, &3000, &30 'Sauvegarde mémoire

IV. Manipulation de chaînes de caractères**ASC (<Chaîne Alphanumérique>)**

Donne la valeur ASCII du premier caractère d'une chaîne alphanumérique.

Utilisation :

Vous voulez mettre une majuscule sur la première lettre d'un mot. Soit un mot « MOT\$ » qui représente un nom propre, et prend donc une majuscule. Si ce mot a été entré par l'opérateur, il peut automatiquement prendre une majuscule de la façon suivante :

1000 A = ASC(MOT\$)

1010 IF A > 96 THEN MOT\$ = CHR\$(A - 32) + RIGHT\$(MOT\$, LEN(MOT\$) - 1)

2000 PRINT MOT\$

A > 96

indique que la lettre est une minuscule,

CHR\$(A - 32)

transforme une minuscule en majuscule,

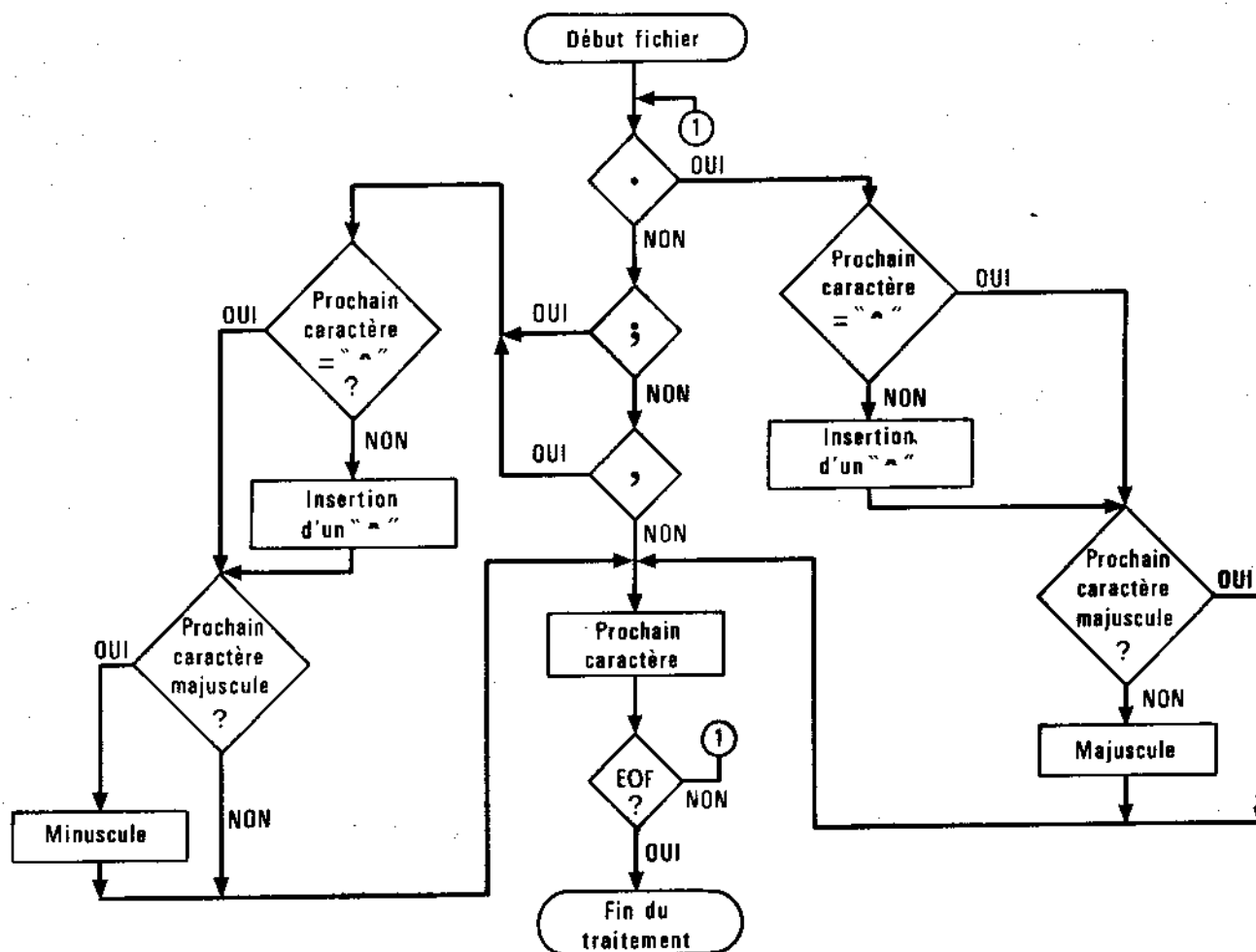
RIGHT\$(MOT\$, LEN(MOT\$) - 1) est égal à MOT\$, la première lettre exclue.

De même, on peut imaginer dans un traitement de textes une telle option qui met automatiquement des majuscules après les points, les supprime après les points-virgules, les virgules, ou met un blanc après les virgules, points-virgules et points.

Analyse du problème :

Le fichier texte étant stocké en mémoire sous forme ASCII, il suffit de le décrire du début à la fin en recherchant les caractères point, virgule et point-virgule.

Ce qui se traduit par l'organigramme suivant :



CHR\$(<Nombre entier>)

Inverse de la fonction ASC.

Transforme un code ASCII en un caractère alphanumérique correspondant.

Utilisation

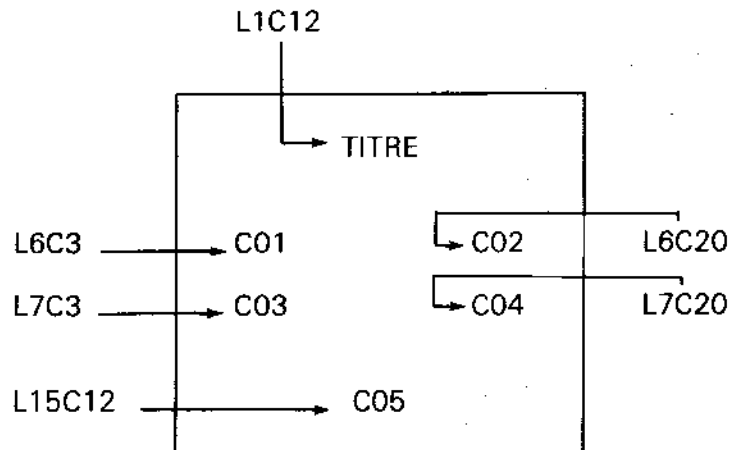
CHR\$ permet, par exemple, d'insérer des codes de contrôles dans la redéfinition d'une touche.

Par exemple, le « 0 » du pavé numérique peut être redéfini en « RUN + CR » de la façon suivante : `KEY 138, "RUN" + CHR$(13)`, ce qui provoquera l'exécution de la commande RUN.

Autre utilisation

Permet de définir des chaînes de caractères complexes (de longueur inférieure à 255 caractères) pour afficher avec un seul ordre tout un écran.

Soit l'écran suivant :



Appelons A\$ la chaîne de caractères qui représentera l'écran.

Cette chaîne aura la structure suivante :

A\$ = SPACE\$(12) + "TITRE" + CHR\$(13) + CHR\$(13) + CHR\$(13) + CHR\$(13) + CHR\$(13)

A\$ = A\$ + SPACE\$(3) + "C01" + etc.

L'ordre « PRINT A\$ » affichera l'écran que l'on vient de définir.

INSTR([< Nombre entier n > ,] < Chaîne 1 > , < Chaîne 2 >)

Cherche si une sous-chaîne (Chaîne 2) appartient à une chaîne (Chaîne 1) en commençant la scrutation à partir du caractère indiqué « n » ou du premier caractère si rien n'est indiqué.

Si « Chaîne 1 » contient « Chaîne 2 », INSTR donne la première occurrence de « Chaîne 2 » dans « Chaîne 1 » ; sinon INSTR donne 0.

Utilisation

Par exemple, dans le jeu du pendu, l'instruction INSTR peut être utilisée pour voir si la lettre proposée fait partie du mot à découvrir.

Soit M\$ le mot à découvrir, et L\$ la lettre proposée :

INSTR (M\$, L\$) = 0 si la lettre ne fait pas partie de M\$,

INSTR (M\$, L\$) = Position de la première lettre L\$ dans M\$ **sinon.**

Autres utilisations

a) Dans un environnement industriel bruité¹⁾, cette commande peut servir à la reconnaissance de données transitant sur une ligne de communications entre deux ordinateurs.

1) Un environnement industriel bruité est un local dans lequel des parasites électriques peuvent parvenir sur un ordinateur (parasités liés à des champs électrostatiques ou magnétiques, ou à des ruptures de courant de haute intensité).

Soit $D\$$ l'ensemble des mots reconnus par l'ordinateur récepteur, appelé dictionnaire par la suite.

Soit $M\$$ un message reçu par l'ordinateur récepteur, émis par l'émetteur, tel que $M\$$ appartienne au dictionnaire.

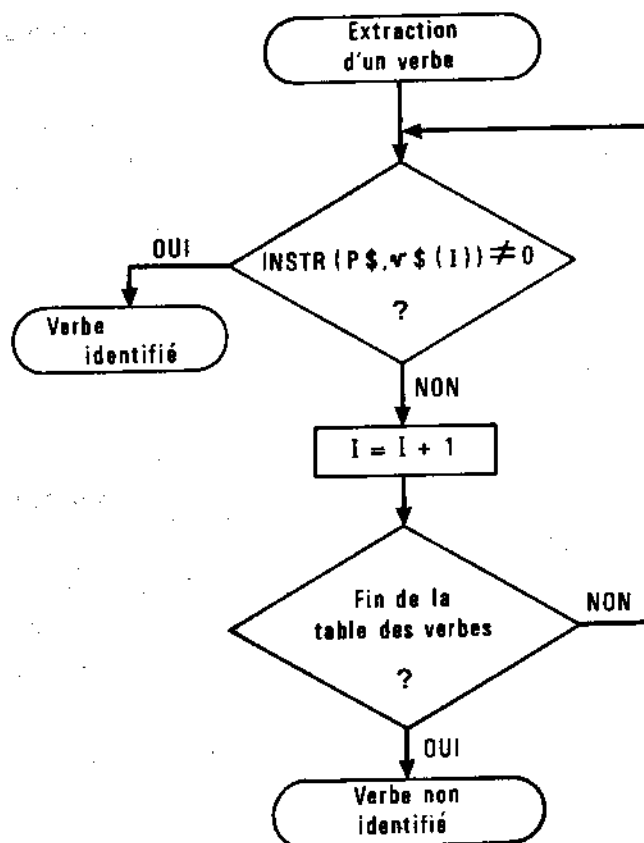
Le vocabulaire émis étant limité, l'ordre INSTR peut servir à identifier un mot émis qui a été plus ou moins bien reçu, et dont une partie seulement est identique à un des mots du dictionnaire.

Supposons que le mot reçu soit composé de n caractères, nous allons comparer ce mot à tous les mots du dictionnaire, pour essayer de le reconnaître, puis une partie du mot, jusqu'à ce que $\text{INSTR}(d\$, M\$) < > 0$ (où $d\$$ est un mot de $D\$$).

Nous aurons alors probablement identifié le mot émis.

b) Analyse syntaxique dans un jeu d'aventures :

Dans les jeux d'aventures, une grande partie du jeu est basée sur un analyseur syntaxique qui « comprend » les phrases que vous entrez au clavier. Comment fonctionne cet analyseur ? Prenons un cas très simple : il ne peut reconnaître que les phrases constituées d'un verbe et d'un sujet. (Par exemple : « PRENDRE HACHE »).



Les verbes sont définis dans un dictionnaire de verbes qui rassemble tous les verbes « compréhensibles » par la machine. Soit V\$ ce dictionnaire. De même pour les sujets. Soit S\$ le dictionnaire des sujets. Nous appellerons v\$ un élément de V\$, et s\$ un élément de S\$. Soit enfin une phrase P\$ entrée par le joueur. L'extraction des verbe et sujet de P\$ se fera de la manière suivante : (voir organigramme p. 33).

De même pour les sujets.

LEFT\$(< Chaîne>, <Entier n>)

Extrait les n caractères les plus à gauche de la chaîne soumise.

Utilisation

Reprenons l'exemple précédent : l'analyseur syntaxique.

Il peut paraître lassant au joueur de taper les mots-clés verbes et sujets en entier à chaque commande, et une bonne amélioration du jeu pourra consister à taper les quatre premières lettres (par exemple) de chaque mot-clé pour aller plus vite.

Réalisation

Au lieu d'analyser INSTR(P\$, v\$(I)), nous analyserons : INSTR(P\$, LEFT\$(v\$(I), 4)) donc les 4 premières lettres du mot-clé.

LEN(< Chaîne>)

Donne le nombre de caractères de la chaîne.

Utilisation

Une chaîne de caractères doit être stockée dans un fichier et la place maximale prévue est de 30 caractères. Si la chaîne à stocker A\$ fait plus de 30 caractères (LEN(A\$) > 30), on stockera par exemple les 30 premiers caractères (LEFT\$(A\$, 30)) dans le fichier.

Autre utilisation

De même, on peut utiliser LEN pour tester la limite supérieure d'une chaîne alphanumérique entrée au clavier, et, par exemple, interdire de dépasser une certaine limite dans un masque de saisie. Soit N cette limite.

100 B\$ = " " 'Chaîne lue

110 A\$ = INKEY\$: IF A\$ = " " THEN 110 'Attente d'un caractère

120 PRINT A\$: 'Lettre lue

130 B\$ = B\$ + A\$

140 IF LEN(B\$) = N THEN 160 'Fin de lecture

150 GOTO 110

160 'Passage à un autre champ

Ligne 110 : Acquisition au clavier

Ligne 140 : Test sur la longueur de la saisie.

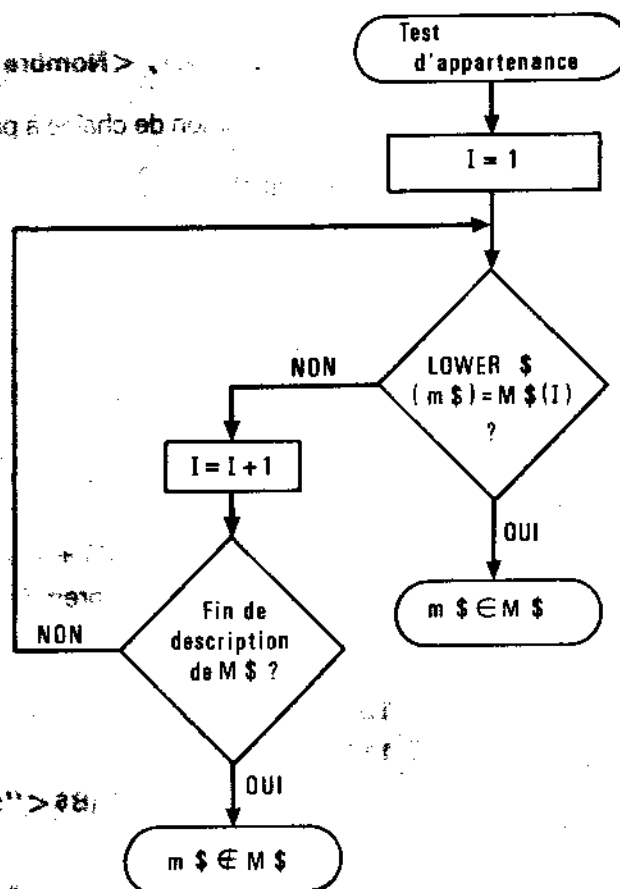
LOWER\$(< Chaîne alphanumérique>)

Change dans une chaîne alphanumérique tous les caractères majuscules en caractères minuscules.

Utilisation

Soit un ensemble de mots $M\$$, et un mot $m\$$ entre au clavier. Nous voulons savoir si $m\$$ appartient à $M\$$ en considérant que seul compte l'ordre des lettres du mot, et pas le fait qu'il soit écrit en majuscules ou en minuscules.

Si les mots de $M\$$ sont écrits en minuscules, nous pourrions tester l'appartenance de $m\$$ à $M\$$ de la façon suivante :



Ainsi, l'utilisateur pourra taper son texte en CAPS LOCK (Majuscules) ou en CAPS UNLOCK (Minuscules) ; la comparaison n'en sera pas affectée.

Autre utilisation

L'ordre LOWER facilite les tests du genre :

```

100 INPUT "Voulez-vous continuer (O/N) ";R$
110 IF R$="O" OR R$="o" THEN SUITE
120 IF R$="N" OR R$="n" THEN FIN
130 PRINT "Je ne comprends pas":GOTO 100
  
```

Ligne 110 : Test de la touche "O" avec et sans SHIFT.
Ligne 120 : Test de la touche "N" avec et sans SHIFT.

qui sera remplacé par :

```

100 INPUT "Voulez-vous continuer (O/N) ";R$
  
```

```

110 IF LOWER$(R$) = "o" THEN SUITE
120 IF LOWER$(R$) = "n" THEN FIN
130 PRINT "Je ne comprends pas": GOTO 100

```

Ligne 110 : Test de la touche « O ».

Ligne 120 : Test de la touche « N ».

MID\$(<Chaîne>, <Nombre entier n1>[, <Nombre entier n2>]).

Extrait une portion de chaîne à partir du caractère n1 jusqu'au caractère n2.

Remarque :

Cette fonction englobe les fonctions LEFT\$ et RIGHT\$.

En effet : MID\$(A\$, 1, N) = LEFT\$(A\$, N), et
MID\$(A\$, LEN(A\$) - N + 1, N) = RIGHT\$(A\$, N).

Extraction d'une sous-chaîne B\$ d'une chaîne A\$.

Supposons que B\$ soit encadrée d'astérisques. Le programme suivant réalise l'extraction de B\$:

```

100 P1 = INSTR(A$, "*" ) 'Limite inferieure de B$
110 P2 = INSTR(P1 + 1, A$, "*" ) 'Limite superieure de B$
120 C$ = MID$(A$, P1 + 1, P2 - P1 - 1) 'Chaîne extraite

```

Ligne 120 : C\$ est la première chaîne entre "*" trouvée dans A\$.

Extraction de données numériques d'une chaîne alphanumérique du type :
« La hausse du pouvoir d'achat aura été de 0,4 % en septembre ».

```

100 FOR I = 1 TO LEN(A$)
110 B$ = MID$(A$, I, 1)
120 IF B$ = "." OR (B$ < "0" AND B$ > "9") THEN PRINT B$;
130 NEXT I

```

Ligne 120 : les caractères « : » et « / » encadrent les chiffres 0 et 9 dans la table des caractères ASCII.

RIGHT\$(<Chaîne alphanumérique>, <Nombre entier N>)

Extrait les N caractères les plus à droite d'une chaîne alphanumérique.

Extraction du dernier mot d'une chaîne.

Par exemple dans la saisie d'un fichier du personnel travaillant dans une société, les articles « Nom » et « Prénom » ne sont pas forcément dissociés. Pour accéder au nom d'une personne, il faudra alors dissocier nom et prénom.

Par exemple, pour l'article A\$ « Prénom Nom », il faudra faire :

```

100 N = INSTR(A$, " ")
110 N$ = RIGHT$(A$, N + 1, LEN(A$) - N - 1) 'Extraction

```

Utilisation

Autre utilisation

Utilisation

SPACE\$(<Nombre entier N <255>)

Permet d'afficher N blancs dans une instruction PRINT ou d'affecter N blancs à une chaîne de caractères complexe (Voir l'exemple donné dans CHR\$).

Remarque :

Utiliser « SPACE\$ » au-delà de 6 caractères pour optimiser la mémoire occupée par cet ordre.

Utilisation

Vous désirez centrer un titre sur la largeur de l'écran :

```
100 MODE 2:L=80 'en MODE 2, 40 en MODE 1 et 20 en MODE 0
```

```
110 A$ = "Texte a centrer"
```

```
120 LE=LEN(A$) 'Longueur du texte a centrer
```

```
130 PRINT SPACE$ ((80-LE)/2);A$ 'Affichage centre
```

Ligne 130 : Calcul du nombre d'espaces à afficher pour centrer le texte et affichage.

STR\$(<Expression numérique>)

Convertit une donnée hexadécimale ou binaire en la même donnée exprimée en décimal.

Utilisation

Dans un utilitaire de mise au point de programmes assembleur (DEBUGGER), il peut être intéressant d'avoir de telles fonctions. Par exemple, dans le menu général, il peut y avoir une option « Changement de base ». Cette option activera le programme suivant :

```
1000 PRINT "Entrez le nombre Hexa. ou Bin. a convertir en Decimal"
```

```
1010 PRINT "sous la forme &nnnn en Hexa. et &x nnnnnnnn en Bin."
```

```
1020 INPUT "Nombre a convertir";N
```

```
1030 PRINT STRING$(N); "en decimal".
```

Ligne 1020 : Lecture du nombre à convertir.

Ligne 1030 : Conversion et affichage.

STRING\$(<Nombre entier>, <Caractère>)

Affiche N fois le même caractère. Cette fonction est une extension de « SPACE\$ » qui se contente d'afficher N blancs. Effectivement, on a : $STRING$(N,32) = SPACE(N) .

Utilisation

Par exemple pour tracer un tableau, les lignes de commentaires et de valeurs pourront être séparées par « _____ » en utilisant PRINT STR\$(Longueur tableau, "_").

UPPER\$(< Chaîne Alphanumérique >)

Transforme les caractères minuscules d'une chaîne alphanumérique en caractères majuscules.

Remarque :

Cette fonction est l'inverse de LOWER\$.

Voir LOWER\$.

Utilisation**VAL(< Chaîne de caractères >)**

Fournit la valeur numérique des premiers caractères de la chaîne indiquée. Si le premier caractère de la chaîne n'est pas un chiffre, cette fonction renvoie 0.

Remarque :

Si le signe « - » ou « . » suivi d'un caractère non numérique apparaît en début de chaîne, le message d'erreur « Type Mismatch » sera affiché à l'écran.

Exemples :

VAL("12 fois 10 = 120") = 12

et VAL("TD") produit une erreur « Type Mismatch ».

V. Gestion de données**DATA<une ou plusieurs constantes>**

Définit des données constantes à l'intérieur d'un programme. Ces données seront lues par la commande « READ » et le pointeur de données pourra être réajusté par la commande « RESTORE ».

Reportez-vous à ces commandes pour avoir plus de détails.

Les données sont séparées par le caractère « virgule ».

Remarque :

Si une donnée alphanumérique comporte un ou plusieurs des signes suivants : Virgule ou séparateur « : », elle doit être obligatoirement encadrée de doubles côtes dans un « DATA ».

10 READ A\$, B\$, C\$

20 DATA "Fonction : Supprime"

30 DATA "Le caractere le plus a droite, et repositionne"

40 DATA "Le curseur en debut de ligne"

Utilisation

Vous voulez créer un masque de saisie qui comporte n libellés. Appelons ces libellés li (avec i compris entre 1 et n). Ils doivent être affichés aux coordonnées $X1i, Y1i$ et le remplissage des champs associés doit se faire aux coordonnées $X2i, Y2i$.

et

Au lieu de répéter n fois les séquences

LOCATE X1i, Y1i : PRINT LI 'Affichage d'un libelle
et

LOCATE X2i, Y2i : INPUT CI 'Entree d'un champ

Vous pourrez faire :

110 CLS

120 READ NL 'Nombre de libelles

130 DIM C\$(NL) 'Dimensionnement tableau des champs.

140 FOR I=1 TO NL

150 READ Y1, X1, LI\$ 'Lecture pos libelles et libelles

160 LOCATE X1, Y1 : PRINT LI\$ 'Affichage libelles

170 NEXT I

180 FOR I=1 TO NL

190 READ Y2, X2

200 LOCATE X2, Y2 : INPUT C\$(I) 'Lecture des champs

210 NEXT I

220 'Traitement

230 DATA 4

240 DATA 1,5,Nom,2,5,Prenom,3,5,Age,4,5,Adresse

250 DATA 1,13,2,13,3,13,4,13

Autre utilisation :

Dans un jeu d'aventures où l'opérateur peut entrer ses commandes sous la forme « VERBE SUJET » (par exemple : « RAMASSER BATON »), vous voulez définir le dictionnaire des mots (verbes, sujets) qui seront interprétables par la machine :

100 READ NV 'Nombre de verbes dans le dictionnaire

110 DIM VE\$(NV) 'Dimensionnement du tableau des verbes

120 FOR I=1 TO NV

130 READ VE\$(I)

140 NEXT I

150 READ NS 'Nombre de sujets dans le dictionnaire

160 DIM SU\$(NS) 'Dimensionnement du tableau des sujets

et

170 FOR I=1 TO NS

180 READ SU\$(I)

190 NEXT I

1000 DATA NV 'Déclaration du nombre de verbes

1010 DATA libelles des verbes

1020 DATA NS 'Déclaration du nombre de sujets

1030 DATA libelles des sujets

READ< libellé de la ou des variable(s) à lire >

Lit une ou plusieurs donnée(s) à partir du pointeur de lecture. Ces données seront stockées dans les variables spécifiées. La commande « READ » est associée aux commandes « DATA » et « RESTORE ». Pour plus de détails, reportez-vous à ces commandes.

Utilisation

Voir la commande « DATA ».

RESTORE[< N° de ligne >]

Force le pointeur de prochaine donnée à lire

- sur la première déclaration « DATA » si « RESTORE » est utilisé sans argument,
- à la ligne spécifiée si « RESTORE » est utilisé avec argument.

Utilisation

Vous écrivez un programme musical qui comporte N morceaux. Un menu permet d'exécuter un des morceaux. Les données (pour activer l'ordre « SOUND ») sont écrites sous forme de « DATA ».

Le programme pourra avoir la structure suivante :

1000 'Affichage du menu

1010 'et choix d'un morceau

1020 'Soit « M » ce morceau

2000 FOR I=1 TO N

2010 READ LIGNE(I) 'N° de ligne ou commence chaque morceau

2020 NEXT I

2030 RESTORE LIGNE(M) 'Pos. du pointeur de données

2040 READ NN 'Nombre de notes dans le morceau

2050 FOR I=1 TO NN

2060 'Activation de l'ordre SOUND

2070 NEXT I

2080 GOTO 1000 'Retour au menu

3000 DATA definition des N° de lignes de debut de notes

3010 DATA nombre de notes morceau 1

3020 DATA notes du morceau 1

3030 DATA nombre de notes morceau 2

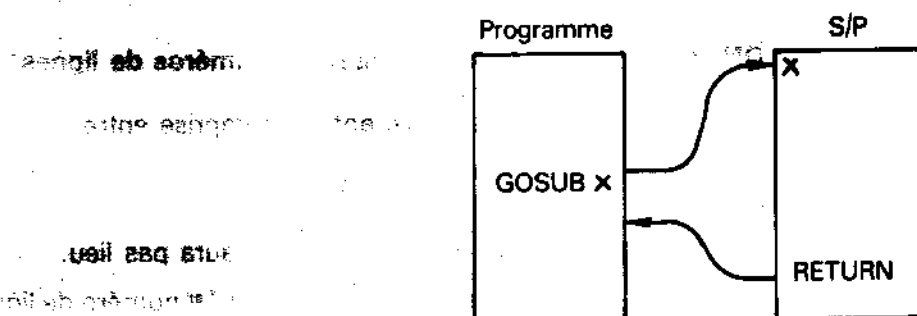
3040 DATA notes du morceau 2

3050 etc.

VI. Sous-programmes et branchements

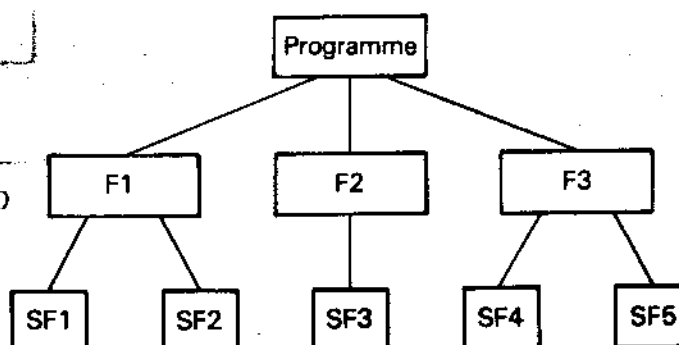
GOSUB <N° ligne>

Déroute le programme sur un sous-programme BASIC commençant à la ligne indiquée. Ce sous-programme doit se terminer par l'ordre « RETURN » qui fera reprendre l'exécution du programme à l'instruction qui suit le « GOSUB ».



L'utilisation de sous-programmes se justifie dans deux cas :

- 1) Une action doit être répétée plusieurs fois dans un programme : mettez-la dans un sous-programme au lieu de la réécrire plusieurs fois.
- 2) Vous avez écrit un programme en utilisant les concepts de programmation hiérarchisée (voir partie 4 chap. 1.4) : l'ordre « GOSUB » permet d'appeler une fonction de niveau moins élevé que le niveau actuel.

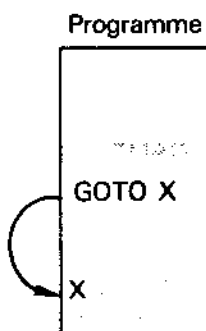


GOTO <N° Ligne>

Déroute le programme en cours d'exécution à la ligne indiquée.

Remarque :

Pour augmenter la lisibilité des programmes, évitez d'employer cet ordre trop fréquemment.

**ON <sélecteur> GOSUB <liste de numéros de lignes>**

« Sélecteur » est une variable entière comprise entre 0 et 255.

Cet ordre permet de dérouter l'exécution d'un programme en fonction de la valeur du sélecteur.

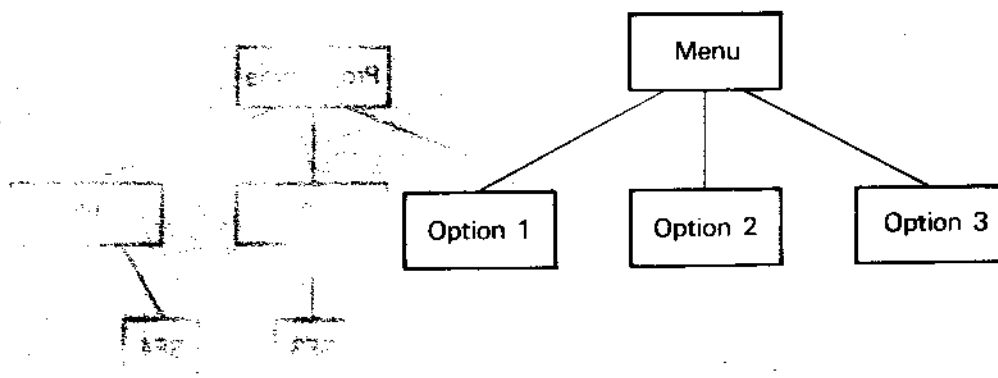
Si le sélecteur vaut 0, le déroutement n'aura pas lieu.

Si le sélecteur vaut 1, le programme ira au 1^{er} numéro de ligne spécifié.

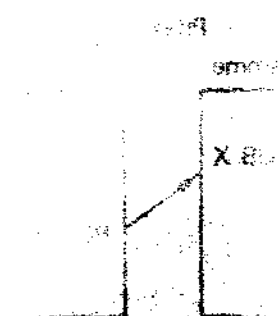
Si le sélecteur vaut n ($n > 0$), le programme ira au n^o numéro de ligne spécifié.

Cet ordre s'emploie typiquement pour activer les options d'un menu vers lequel on doit revenir après traitement de l'option choisie.

Par exemple :



Menu principal

**1000 'Affichage du menu****1010 INPUT "Option (1, 2 ou 3)"; O****1020 ON O GOSUB 2000, 3000, 4000****1030 GOTO 1000 'Retour au menu****2000 'Traitement 1****3000 'Traitement 2****4000 'Traitement 3***Remarque :*

Comme pour l'ordre « GOSUB », chacun des sous-programmes activé par l'ordre « ON GOSUB » doit se terminer par « RETURN » pour redonner le contrôle au programme appelant.

ON <sélecteur> GOTO <liste de numéros de lignes>

« Sélecteur » est une variable entière comprise entre 0 et 255.

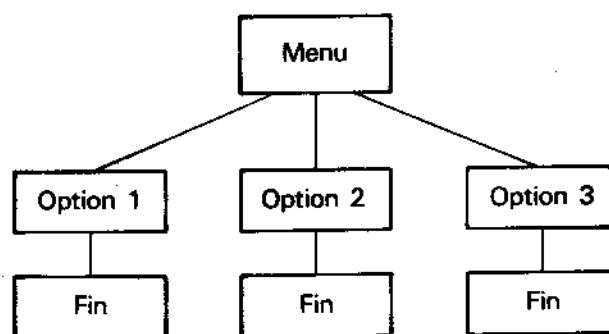
Cette instruction permet de dérouter l'exécution d'un programme en fonction de la valeur du sélecteur.

Si le sélecteur vaut 0, le déroutement n'a pas lieu.

Si le sélecteur vaut 1, le programme sera dérouté vers le premier numéro de ligne spécifié.

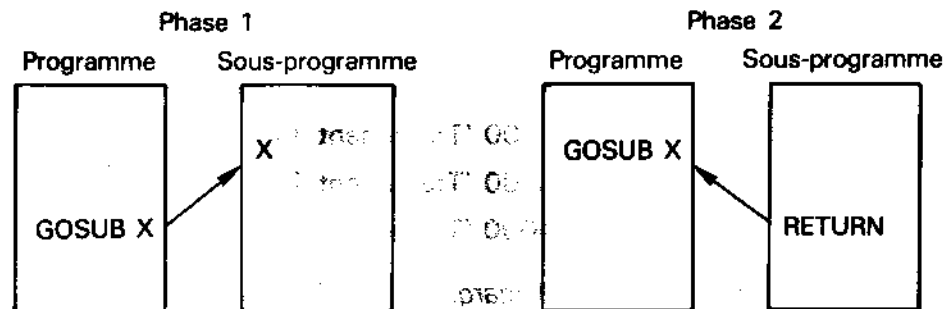
Si le sélecteur vaut n ($n > 0$), le programme sera dérouté vers le n° numéro de ligne spécifié.

Cet ordre s'emploie typiquement pour activer les options d'un menu vers lequel on ne reviendra pas après traitement de l'option choisie.

**RETURN**

Signale la fin d'un sous-programme appelé par un des ordres « GOSUB ».

Redonne le contrôle au programme appelant, à l'instruction suivant l'appel.



Remarque :

L'ordre « RETURN » est obligatoire en fin des sous-programmes activés par les ordres « GOSUB », « ON GOSUB », « ON SQ GOSUB », « AFTER GOSUB », « EVERY GOSUB » et « ON BREAK GOSUB ».

VII. Gestion des erreurs

ERR et ERL

Variables positionnées par le système lorsqu'une erreur se produit dans l'exécution d'un programme ou en mode direct.

« ERR » donne le numéro d'erreur et, le cas échéant, « ERL » donne le numéro de ligne où s'est produite l'erreur.

Les codes d'erreur sont les suivants :

- 1 NEXT in attendu
- 2 Erreur de syntaxe
- 3 RETURN in attendu
- 4 Pas assez de données pour les READ correspondants
- 5 Valeur ou argument incorrect
- 6 Débordement dans un calcul (valeur réelle $> 1.7 \text{ E } 38$)
- 7 Dépassement de la capacité mémoire
- 8 Numéro de ligne inexistant
- 9 Indice de tableau hors limites
- 10 Redimensionnement d'un tableau
- 11 Division par 0
- 12 Commande invalide en mode direct

- 13 Affectation d'une chaîne à une variable numérique ou inverse
 - 14 L'espace réservé aux chaînes n'est pas suffisant
 - 15 Chaîne trop longue
 - 16 Expression trop compliquée
 - 17 CONT impossible (une erreur s'est produite ou END rencontré)
 - 18 Fonction inconnue
 - 19 La commande RESUME manque
 - 20 La commande RESUME n'est pas attendue
 - 21 Commande directe trouvée pendant le chargement d'un programme
 - 22 Signe d'opération absent
 - 23 Ligne trop longue
 - 24 Fin de fichier (EOF) rencontrée
 - 25 Erreur dans le type du fichier
 - 26 NEXT manquant
 - 27 Fichier déjà ouvert
 - 28 Commande inconnue
 - 29 WEND manquant
 - 30 WEND inattendu
- et, sur CPC 664 et 6128 :
- 31 Fichier non ouvert
 - 32 Lecture/écriture interrompue

Exemple :

L'utilitaire qui suit facilitera la mise au point de vos programmes en indiquant en clair le type d'erreur rencontrée.

```
10000 ON ERROR GOTO 20000
10010 DIM E$(30) 'Tableau des erreurs
10020 FOR I=1 TO 30
10030 READ E$(I) 'Lecture des libelles des erreurs
10040 NEXT I
10050 DATA Next in attendu, Erreur de syntaxe, Return in attendu,
Pas assez de donnees
10060 DATA Valeur ou argument incorrect, Debordement dans un
calcul
10070 DATA Depassement de la capacite memoire, Numero de ligne
inexistant
```

10080 DATA Indice de tableau hors limites, Redimensionnement de tableau

10090 DATA Division par zero, Commande invalide en mode direct

10100 DATA Affectation d'une chaine a une variable numerique ou inverse

10110 DATA L'espace reserve aux chaines deborde, Chaine trop longue

10120 DATA Chaine trop compliquee, CONT impossible, Fonction inconnue

10130 DATA Commande RESUME absente, Commande RESUME inattendue

10140 DATA Commande directe trouvee sur le chargement d'un programme

10150 DATA Signe d'operation absent, Ligne trop longue, Fin de fichier rencontree

10160 DATA Erreur dans le type de fichier, NEXT manquant, Fichier deja ouvert

10170 DATA Commande inconnue, WEND manquant, WEND inattendu

10180 RETURN

20000 PRINT E\$(ERR) ; "ligne" ; ERL

20010 END

Pour être activé, cet utilitaire devra être appelé sur la première ligne du programme à tester par « GOSUB 10000 » et il faudra prendre garde qu'aucun tableau E\$ n'existe dans le programme que vous mettez au point.

Remarque :

Sur CPC 664 et 6128, modifiez les lignes suivantes :

10010 DIM E\$(32)

10020 FOR I=1 TO 32: **READ** E\$(I):**NEXT** I

10200 DATA Fichier non ouvert, Lecture/ecriture interrompue

ERROR<Nombre entier>

Simule une erreur en affectant à une action un numéro d'erreur compris entre 1 et 255.

Si le numéro d'erreur est compris entre 1 et 32, le message d'erreur sera le même que pour l'erreur BASIC de même numéro.

Si le numéro d'erreur est compris entre 33 et 255, l'utilisateur pourra définir ses propres messages d'erreur.

Utilisation

Définition de nouveaux messages d'erreur :

```
10 GOSUB 1010
20 ON ERROR GOTO 2000
1000 'Definition des nouvelles erreurs de No> =33
1010 DIM E$(N) 'Nombre d'erreurs definies
1020 FOR I=1 TO N
1030 READ E$(I)
1040 NEXT I
1050 DATA Libelle des nouvelles erreurs
2000 'Traitement des erreurs
```

ON ERROR GOTO<N° de Ligne>

Définit la ligne qui sera activée dès qu'une erreur est détectée.

Remarques :

- a) Cette commande est souvent placée en début de programme ou du moins sur la ligne à partir de laquelle une erreur peut se produire.
- b) L'ordre « ON ERROR GOTO 0 » supprime le déroutement du programme en cas d'erreur.

Utilisation

Voir les commandes « ERR », « ERL » et « ERROR ».

RESUME[<N° ligne>]

Redonne le contrôle au programme ayant déclenché l'erreur après l'exécution d'une routine d'erreur.

Si aucun numéro de ligne n'est indiqué, l'exécution du programme reprend à la ligne qui suit celle où une erreur a été détectée.

Si une ligne est indiquée, l'exécution du programme reprend à cette ligne.

Utilisation

Vous considérez que l'erreur est grave et qu'il faut reprendre l'exécution du programme à la première ligne :

```
10 ON ERROR GOTO 1000
20 'Programme
1000 'Traitement de l'erreur
1010 RESUME 10
```

Vous considérez que l'erreur n'est pas grave, qu'elle doit seulement provoquer l'affichage d'un message d'erreur, et être suivie de la réactivation de la ligne ayant provoqué l'erreur :

10 ON ERROR GOTO 1000

20 'Programme

1000 'Traitement de l'erreur

1010 RESUME

RESUME NEXT

Reprend l'exécution d'un programme (après le déroutement et l'exécution d'un programme de gestion d'erreur) juste après l'instruction ayant provoqué l'erreur, donc, en ne réexécutant pas l'ordre ayant provoqué l'erreur.

VIII. Gestion de la mémoire

CALL <Adresse> [, <liste de paramètres>]

Donne le contrôle à un programme en langage machine dont le point d'entrée est l'adresse donnée dans le premier argument du CALL.

Un ou plusieurs paramètres (32 au maximum) peuvent être fournis au programme activé. Ces paramètres doivent être séparés par une virgule et peuvent être :

- des constantes entières (de -32768 à 32767 ou de 0 à &FFFF),
- des variables entières ou alphanumériques,
- des pointeurs (pour plus de renseignements, reportez-vous à la description de l'ordre « VARPTR »).

Si aucun paramètre n'est fourni, le programme en langage machine est activé, et le retour sous BASIC se fait quand la commande assembleur « RTS » est rencontrée.

Si un ou plusieurs paramètres sont fournis, les registres du Z80 contiennent toutes les informations pour y accéder et sont définis comme suit au moment du débranchement :

PC contient l'adresse du point d'entrée du programme en langage machine,

A contient le nombre de paramètre(s) passé(s),

DE contient le dernier paramètre,

IX pointe sur la zone RAM du dernier paramètre passé.

Chaque paramètre est défini par deux octets : poids faible, poids fort (LSB, MSB).

Si le paramètre est entier ou contient un entier, ces deux octets donnent la valeur numérique de l'entier.

Si le paramètre est une variable alphanumérique, ces deux octets contiennent l'adresse d'implantation de la variable (VARPTR).

Par exemple :

B = 4

B\$ = "Texte"

CALL Adresse, B, &334, @B, A\$, @B\$

Registre A = 5 (nombre de paramètres passés)

Registre IX = LSB @ B\$

IX + 1 = MSB @ B\$

IX + 2 = LSB @ A\$

IX + 3 = MSB @ A\$

IX + 4 = LSB @ B

IX + 5 = MSB @ B

IX + 6 = 34 (LSB de &334 sur 16 bits)

IX + 7 = 3 (MSB de &334 sur 16 bits)

IX + 8 = 4 (LSB de 4 sur 16 bits)

IX + 9 = 0 (MSB de 4 sur 16 bits)

ERASE < Nom(s) de variable(s) >

Permet d'effacer l'espace RAM réservé à des variables numériques, alphanumériques et tableaux quand elles ne sont plus nécessaires.

FRE(< Nombre >) ou FRE(< Chaîne alphanumérique >)

- « Nombre » peut être une valeur numérique ou une variable numérique.
- « Chaîne alphanumérique » peut être le contenu d'une chaîne ou une variable représentant une chaîne.

Donne la place RAM disponible sous BASIC.

La première forme donne la valeur « brute » de la RAM disponible.

La deuxième forme donne la valeur de la RAM disponible après avoir éliminé les variables qui ne servent à rien ou sont inutilisées (GARBAGE COLLECTION).

Remarque :

La deuxième forme peut prendre une ou deux secondes, mais permet, après réarrangement de la RAM, de disposer d'un espace plus grand.

HIMEM

Donne l'adresse RAM supérieure utilisée par le BASIC. Cette adresse n'évolue pas en cours de programme sauf si vous utilisez les ordres de redéfinition de caractères « SYMBOL AFTER ».

Utilisation

Implantation et exécution d'un sous-programme en langage machine :

- Mémoriser HIMEM,
- Implanter des routines en langage machine au-delà de HIMEM,
- Exécuter ces routines sous BASIC,
- Restituer le HIMEM pour profiter pleinement du BASIC.

100 S=HIMEM 'Memo @ RAM la plus haute utilisée par BASIC

110 MEMORY XX 'On baisse cette adresse

120 'Implantation de programmes en LM entre XX et S

130 'Execution de ces programmes LM

140 MEMORY S 'Retour à l'espace RAM maximum utilisable sous BASIC

150 'Les sous-programmes ASM pourront maintenant être écrasés par BASIC

160 'Si la mémoire utilisée par le BASIC est supérieure à XX

INP (Adresse)

ad (s)mem > 32493

Donne la valeur courante du port d'entrée/sortie spécifié par l'adresse.

Cette adresse est exprimée sur 16 bits et permet donc un adressage de 2^{16} (= 65536) ports.

En réalité, très peu de ports sont accessibles en entrée sur l'AMSTRAD de base. La fonction « INP » n'est donc pas très utilisée (voir Partie 2, chap. 3 pour avoir plus de détails sur les ports d'entrée/sortie).

MEMORY < Adresse >

Définit l'adresse RAM la plus haute utilisée par le BASIC.

Cette instruction a pour but de protéger une zone mémoire afin d'y installer des programmes en langage machine ou des données en étant sûr qu'ils ne seront pas écrasés par le BASIC.

Si l'adresse fournie est trop petite ou trop grande, le message « Memory Full » apparaîtra et l'instruction n'aura aucun effet.

OUT < Adresse de sortie >, < Valeur >

Permet d'envoyer la valeur spécifiée sur le port d'entrée/sortie spécifié.

L'adresse doit être exprimée sur 16 bits (donc, 2^{16} ports sont adressables). Sur l'AMSTRAD de base, peu de ports sont utilisables par la commande « OUT » (voir Partie 2 chap: 3 pour plus de détails sur les ports d'entrée/sortie).

Remarque :

Une erreur sur cette instruction peut avoir des conséquences graves comme le « plantage »¹⁾ de l'unité centrale, ou l'écriture intempestive sur l'unité de cassette ou de disquette.

PEEK (<Adresse>)

Permet de lire dans la mémoire RAM ou ROM à l'adresse spécifiée. Reportez-vous à l'ordre POKE pour connaître la répartition de la mémoire sur les ordinateurs CPC.

Utilisation

Cet ordre peut être très utile si vous désirez faire communiquer des programmes ASSEMBLEUR et des programmes BASIC. Imaginons que vous désiriez incorporer une routine ASSEMBLEUR dans un programme BASIC, pour des raisons de vitesse d'exécution par exemple. La routine ASSEMBLEUR pourra faire des calculs et les communiquer au programme BASIC par des mémoires RAM. Le programme BASIC lira ces données avec l'ordre PEEK.

POKE <Adresse>, <Entier>

Permet d'écrire dans la mémoire RAM à l'adresse spécifiée la valeur spécifiée qui doit être comprise entre 0 et 255.

Remarque :

Une erreur sur cette instruction peut avoir des conséquences graves comme le « plantage » de l'unité centrale, ou l'écriture intempestive sur l'unité de cassette ou de disquette.

La RAM occupe les 64 kilo-octets d'adresse 0 à &FFFF de la manière suivante :

&0000 à &003F :	Copie conforme du ROM BIOS
&0040 à &3FFF :	Espace de travail pour le langage utilisé
&4000 à &BAFF :	RAM utilisateur
&BB00 à &BFFF :	Zone des variables système et table des points d'entrée des routines du FIRMWARE.
&C000 à &FFFF :	RAM écran.

Reportez-vous en partie 2 chapitre 2 pour avoir plus de détails sur la mémoire.

Utilisation

Vous pouvez insérer les codes en langage machine d'un programme écrit en assembleur dans un programme BASIC et mettre ces codes en mémoire RAM de la façon suivante :

1) L'unité centrale est « plantée » lorsqu'elle n'est plus sous le contrôle du programmeur.

100 FOR I=0 TO NB 'Nombre de codes hexa

110 READ A:POKE &9000+I, A 'Lecture du LM et implantation en RAM

120 NEXT I

130 DATA Donnees Hexa

Autre utilisation

Vous pouvez modifier la valeur d'une variable en utilisant l'ordre « POKE » pour écrire dans la zone RAM où elle est stockée. Reportez-vous à l'ordre « VARPTR » pour avoir plus de détails.

VARPTR (Adresse)

Donne l'adresse où est implantée une variable BASIC.

Format

AD = @VAR

— « VAR » est une variable entière, réelle, alphanumérique ou élément de tableau.

— AD sera l'adresse de la variable sur 16 bits.

Son contenu est fonction du type de variable fournie.

AD-1 contient le type de la variable :

Variable	AD-1
entière	1
alphanumérique	2
flottante	4

AD-2 contient le dernier caractère du nom de la variable dont le bit de poids fort a été mis à 1 (OR & 80).

AD-3 à AD-n contiennent les autres caractères du nom de la variable si celle-ci comporte n lettres.

• Variable entière :

AD : 8 bits LSB

AD + 1 : Bits 8 à 15 : 7 bits MSB

Bit 16 : Signe = 1 si < 0
= 0 si >= 0

• Variable réelle (flottante) :

Codée sur 5 octets (AD à AD+4)

Bit 7 de AD + 3 : Signe = 0 si >= 0
= 1 si < 0

AD + 4 = $\text{Log}_2(m) + 81$ ou m représente le nombre réel.

Les bits 0 à 6 de AD + 3 et les octets AD, AD + 1 et AD + 2 sont codés en sens inverse : le MSB est le bit 6 de AD + 3 et le LSB le bit 0 de AD. Ils représentent :

— bit 6 de AD + 3 = $1/2 (AD + 4)$ qui doit être ajouté à AD + 4 pour donner m,

— bit 5 de AD + 3 = $1/4 (AD + 4)$ qui doit être ajouté à AD + 4 pour donner m,

— bit 4 de $AD + 3 = 1/8 (AD + 4)$ qui doit être ajouté à $AD + 4$ pour donner m ,

etc.,

— bit 0 de $AD = 1/2^{31} (AD + 4)$ qui doit être ajouté à $AD + 4$ pour donner m .

- Variable alphanumérique :

AD = longueur de la chaîne en nombre d'octets ;

$AD + 1$ = LSB de l'adresse où se trouve la chaîne

$AD + 2$ = MSB de l'adresse où se trouve la chaîne

- Variable tableau :

Une dimension :

AD = Adresse du pointeur sur $VAR(0)$

$AD - 1$ et $AD - 2$ = MSB et LSB de la dimension du tableau

$AD - 3$ = Dimension de $VAR - 1$

$AD - 4$ et $AD - 5$ = MSB et LSB de l'offset entre deux variables

$AD - 6$ = Type de variable : 1 si entière
2 si alphanumérique
4 si flottante

$AD - 7$ = (Valeur ASCII de la dernière lettre) OR &80

$AD - 8$ = (Valeur ASCII de l'avant-dernière lettre) OR &80

$AD - n - 4$ = (Valeur ASCII de la première lettre) OR &80

x dimensions ($x > 1$) :

$AD - n$ = (Valeur ASCII de la dernière lettre) OR &80

$AD - n + 1$ = Type de la variable : 1 si entière
2 si alphanumérique
4 si flottante

$AD - n + 2$ et $AD - n + 3$ = Longueur offset entre 2 variables

$AD - n + 4$ = Nombre d'indices

$AD - n + 5$ et $AD - n + 6$ = Nombre d'éléments du 1^{er} indice

$AD - n + 7$ et $AD - n + 8$ = Nombre d'éléments du 2^e indice avec
 $n = 6 + 2 * \text{Nombre d'indices}$.

IX. Gestion des interruptions

AFTER <délai> [, <N° chronomètre>] GOSUB <N° de ligne>

— « délai » = durée de l'attente en multiples de 0.02 sec.,

— « N° chronomètre » est compris entre 0 et 3.

Cet ordre lance l'exécution d'un sous-programme BASIC après une attente définie par le paramètre « délai ».

Si le numéro du chronomètre est omis, le chronomètre 0 est pris par défaut.

DI

Interdiction des interruptions issues des compteurs d'interruptions et activées par les commandes « EVERY » et « AFTER ».

Remarques :

a) La commande « BREAK » n'est pas affectée par cette instruction.

b) Cette commande est annulée par :

— EI,

— la rencontre d'un « RETURN » en fin d'un sous-programme d'interruptions.

EI

Rétablit les interruptions issues des compteurs d'interruptions qui avaient été interdites par DI.

EVERY <délai>[, <N° de chronomètre>] GOSUB <N° ligne>

— « délai » est la durée de l'attente en multiples de 0.02 sec.,

— « N° chronomètre » est compris entre 0 et 3.

Lance cycliquement l'exécution d'un sous-programme BASIC après une attente définie par « délai ». Si le numéro du chronomètre est omis, le chronomètre 0 est pris par défaut.

Utilisation

Réalisation d'une horloge sous interruptions :

```
120 CLS:PRINT "Entrez l'heure sous la forme : HH/MM/SS"
```

```
130 PRINT:PRINT:INPUT H$
```

```
140 PRINT:INPUT "Appuyez sur <ENTER> pour valider" ; A$
```

```
150 :
```

```
160 H=VAL(LEFT$(H$,2))
```

```
170 M=VAL(MID$(H$,4,2))
```

```
180 S=VAL(RIGHT$(H$,2))
```

```
185 CLS
```

```
190 :
```

```
200 EVERY 50,0 GOSUB 300
```

```
210 :
```

220 GOTO 220

300 REM Calcul et affichage de l'heure

310 :

320 S=S+1:IF S=60 THEN S=0:M=M+1:IF M=60
THEN M=0:H=H+1:IF H=24 THEN H=0

330 LOCATE 10,10:PRINT H"/"M"/"S"/ "

340 :

350 RETURN

Lignes 120 à 130 : Initialisation du temps

Lignes 160 à 180 : Conversion de la chaîne H\$ en HH/MM/SS

Ligne 200 : EVERY

Lignes 300 à 350 : Calcul et affichage de l'heure.

REMAIN (<entier>)

L'« entier » est compris entre 0 et 3 et représente le numéro du chronomètre actif.

Donne le temps qui reste avant le lancement d'un programme sous interruptions par les instructions « EVERY » ou « AFTER ».

Remarque :

Cette commande arrête le chronomètre spécifié. L'appel du sous-programme sous interruptions affecté au chronomètre lu n'a donc pas lieu après l'utilisation de cet ordre.

TIME

Donne le temps écoulé depuis la mise en route de l'ordinateur ou la dernière commande « RESET » (CALL 0) en 1/300^e sec.

X. Instructions musicales

Reportez-vous à la partie 6 pour avoir tous les détails concernant le générateur sonore AY3-8912 (PSG).

ENT<N° enveloppe>[,<section d'enveloppe>],
[,<section d'enveloppe>], ...

Définit les variations de ton des notes pendant leur émission.

« N° enveloppe » est un entier compris entre 1 et 15.

Remarque :

Si le numéro d'enveloppe est négatif (entre -15 et -1), le son se répète jusqu'à la fin de sa durée fixée par « SOUND ».

Une section d'enveloppe peut contenir 2 ou 3 paramètres :

- nombre de pas,
- amplitude du pas,
- durée du pas.

1) La section d'enveloppe contient 3 paramètres :

Nombre de pas : de 0 à 239.

Nombre de divisions dans la tonalité à l'intérieur d'une section d'enveloppe.

Amplitude du pas : de -128 à 127

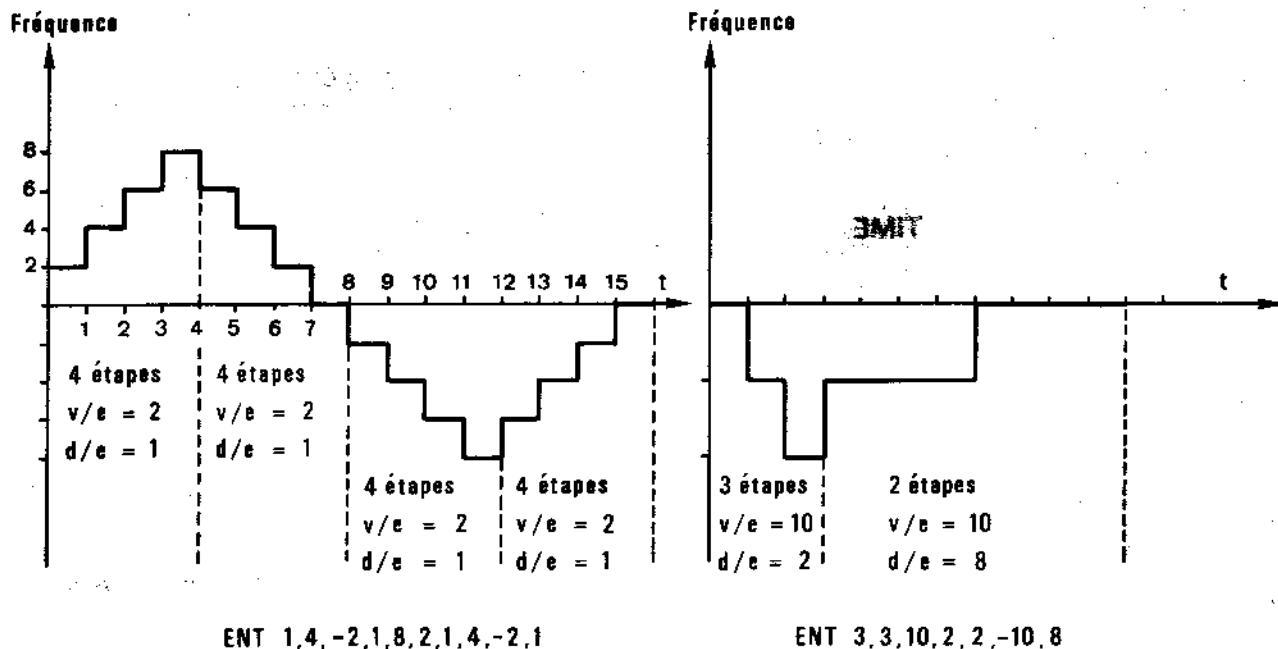
- Si ce paramètre est < 0 , augmente la hauteur de la note ;
- Si ce paramètre est > 0 , diminue la hauteur de la note.

Durée du pas : de 0 à 255 en $1/100^{\circ}$ sec.

2) La section d'enveloppe contient 2 paramètres :

Période sonore : Valeur de la période : c'est la hauteur du son (voir tableau donné à l'ordre « SOUND », p. 67).

Durée du pas : en $1/100^{\circ}$ sec.



avec v/e = variation par étape
et d/e = durée par étape

**ENV<N° d'enveloppe>[, <section d'enveloppe>]
[, <section d'enveloppe>]...**

Définit l'enveloppe de volume pour une enveloppe repérée par son numéro de 1 à 15. Cette commande est utilisée conjointement à la commande « SOUND ».

Une section d'enveloppe peut contenir 2 ou 3 paramètres.

Si elle contient 3 paramètres, ils indiquent :

- nombre de pas,
- amplitude du pas,
- durée du pas.

Si elle contient 2 paramètres, ils indiquent :

- enveloppe matérielle,
- période de l'enveloppe.

1) La section d'enveloppe contient 3 paramètres :

Nombre de pas : Nombre de divisions définissant l'évolution d'un son à l'intérieur d'une section. Ce nombre est compris entre 0 et 127.

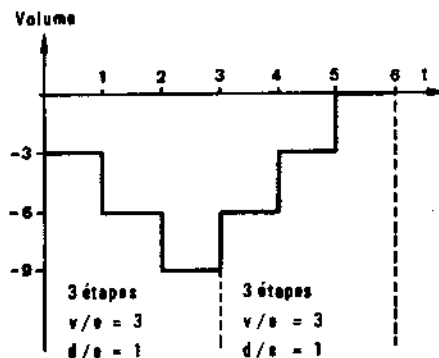
Amplitude du pas : comprise entre - 128 et 127 ; de - 128 à 0 : augmente le volume de la note ; de 0 à 127 : diminue le volume de la note.

Durée du pas : en 1/100^e sec. (de 0 à 256)

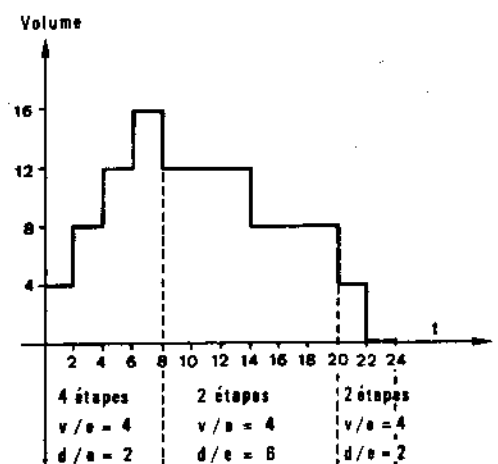
2) La section d'enveloppe contient 2 paramètres :

Enveloppe matérielle : Valeur à envoyer au registre d'enveloppe (voir Partie 6).

Période de l'enveloppe : Valeur à envoyer aux registres de période d'enveloppe (voir Partie 6).



ENV 1,3,3,1,3,-3,1



ENV 1,4,-4,2,2,4,6,2,-4,2

où v/e = variation par étape
et d/e = durée par étape

RELEASE < Canaux >

Libère les canaux sonores bloqués par la commande « SOUND ». Le paramètre « Canaux » peut prendre les valeurs suivantes :

Valeur	Canal
1	A
2	B
3	A et B
4	C
5	C et A
6	C et B
7	A et B et C

Remarque :

Si aucun canal n'est bloqué (le bit 6 de la commande « SOUND » représente le blocage d'un son : b6 = 1 → son bloqué, et b6 = 0 → son non bloqué), la commande « RELEASE » n'a aucun effet.

SOUND < état du canal >, < période sonore > [, < durée > [, < volume > [, < enveloppe de volume > [, enveloppe de ton > [, < période de bruit >]]]]

Définit les caractéristiques d'un son.

Etat du canal :

Bit	Fonction
0	Sortie canal A
1	Sortie canal B
2	Sortie canal C
3	Rendez-vous canal A
4	Rendez-vous canal B
5	Rendez-vous canal C
6	Bloquage d'un canal
7	Vidage d'un canal

Définitions :

— Rendez-vous : technique utilisée pour la synchronisation forcée de deux canaux : lorsqu'un canal s'arrête et qu'en même temps un autre commence à jouer, il y a rendez-vous entre ces deux canaux.

Reportez-vous à l'exemple ci-dessous pour avoir des détails sur l'utilisation des rendez-vous.

— Période sonore : hauteur de la note.

On a : Fréquence = $440 \cdot (2^{\uparrow (\text{Octave} + ((N - 10) / 12)))$

et : Période = $\text{ROUND} (62\,500 / \text{Fréquence})$

avec N = 1 pour DO

N = 2 pour DO #

N = 3 pour RE

N = 4 pour RE #

N = 5 pour MI

N = 6 pour FA

N = 7 pour FA #

N = 8 pour SOL

N = 9 pour SOL #

N = 10 pour LA

N = 11 pour LA #

N = 12 pour SI

Si la période sonore vaut 0, aucune fréquence n'est émise. Cette valeur est utilisée pour générer du bruit blanc.

— Durée : Longueur d'un son en 1/100^e sec. Initialisée à 20 par défaut (1/20^e sec.)

— Si « durée » = 0, c'est l'enveloppe de volume qui détermine la durée des sons.

— Si « durée » < 0, l'enveloppe de volume est répétée un nombre de fois égal à la valeur absolue du paramètre « durée ».

— Volume : volume initial d'une note. Compris entre 0 et 15. Initialisé par défaut à 12. Ce volume sera modifié par l'enveloppe de volume (ENV) si elle est définie.

— Enveloppe de volume : désigne l'une des 15 enveloppes de volume définissables par l'utilisateur et codées de 1 à 15.

— Enveloppe de ton : désigne l'une des 15 enveloppes de ton définissables par l'utilisateur et codées de 1 à 15.

— Période de bruit : compris entre 0 (aigu) et 31 (grave).

Pour illustrer l'utilisation des « rendez-vous », écrivons les données BASIC correspondant à la partition suivante :

Voie A (octave 1)

Voie B (octave 2)

Voie C (octave 3)

ou Ra = Rendez-vous avec le canal A
et Rb = Rendez-vous avec le canal B

Table des fréquences pour les octaves 0 à 3

Do	261.62	478	} Octave 0
Do #	277.18	451	
Re	293.66	426	
Re #	311.12	402	
Mi	329.62	379	
Fa	349.22	358	
Fa #	369.99	338	
Sol	391.99	319	
Sol #	415.3	301	
La	440	284	
La #	466.16	268	
Si	493.88	253	
Do	523.25	239	} Octave 1
Do #	554.36	225	
Re	587.32	213	
Re #	622.25	201	
Mi	659.25	190	
Fa	698.45	179	
Fa #	739.98	169	
Sol	783.99	159	
Sol #	830.6	150	
La	880	142	
La #	932.32	134	
Si	987.76	127	
Do	1046.5	119	} Octave 2
Do #	1108.73	113	
Re	1174.65	106	
Re #	1244.5	100	
Mi	1318.51	95	
Fa	1396.91	89	
Fa #	1479.97	84	
Sol	1567.98	80	
Sol #	1661.21	75	
La	1760	71	
La #	1864.65	67	
Si	1975.53	63	
Do	2093	60	} Octave 3
Do #	2217.46	56	
Re	2349.31	53	
Re #	2489.01	50	
Mi	2637.02	47	
Fa	2793.82	45	
Fa #	2959.95	42	
Sol	3135.96	40	
Sol #	3322.43	38	
La	3520	36	
La #	3729.31	34	
Si	3951.06	32	

SOUND 1,478	DO1	VOIE 1
SOUND 4,119	DO3	VOIE 3
SOUND 1,379	MI1	VOIE 1
SOUND 4,95	MI3	VOIE 3
SOUND 17,319	SOL1	VOIE 1 RDV B
SOUND 10,119	DO2	VOIE 2 RDV A
SOUND 4,80	SOL3	VOIE 3
SOUND 2,190	MI2	VOIE 2
SOUND 4,60	DO3	VOIE 3
SOUND 17,319	SOL1	VOIE 1 RDV B
SOUND 10,119	DO2	VOIE 2 RDV A
SOUND 4,95	MI3	VOIE 3
SOUND 1,379	MI1	VOIE 1
SOUND 4,80	SOL3	VOIE 3
SOUND 1,478	DO1	VOIE 1
SOUND 4,60	DO3	VOIE 3

RDV =
Rendez-vous

```

10 FOR I=1 TO 16
20 READ A,B
30 SOUND A,B
40 NEXT I
50 DATA 1,478,4,119,1,379,4,95,17,319,10,119,4,80,2,190
60 DATA 4,60,17,319,10,119,4,95,1,379,4,80,1,478,4,60

```

SQ (<N° de canal>)

Donne l'état de la file d'attente sonore d'un canal.

N° de canal = 1 pour le canal A

= 2 B
= 3 C

Cette fonction donne un entier dont les bits ont la signification suivante :

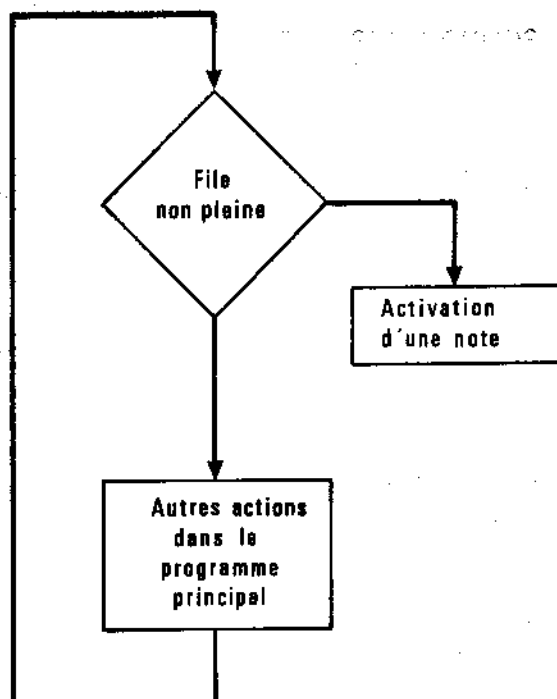
0 à 2	Nombre de places libres dans la file
3	Rendez-vous avec le canal A
4	Rendez-vous avec le canal B
5	Rendez-vous avec le canal C
6	Début de file en attente
7	Canal en activité au moment de l'activation de SQ

ON SQ (<N° canal>) GOSUB <N° ligne>

No Canal = 1 pour le canal A

_____ 2 _____ B
_____ 3 _____ C

Provoque un déroutement à la ligne indiquée si la file d'attente du canal spécifié n'est pas pleine. Cette fonction permet, par exemple, d'exécuter une musique en permanence, tout en faisant autre chose.



Si nous mettons les notes sous forme de DATA, le programme aura l'allure suivante :

1000 'Programme principal

1010 ON SQ(1) GOSUB 3000 'Activation du S/P de remplissage de file sonore

1020 GOTO 1000

3000 READ NOTE 'Lecture d'une note

3010 IF NOTE=0 THEN RESTORE:GOTO 3000

3020 SOUND 1, NOTE 'Mise en file d'attente

3030 RETURN

3040 DATA notes, 0 'Notes et terminateur

XI. Instructions graphiques

CLG[<couleur d'encre>]

Efface l'écran graphique.

Si l'encre est spécifiée, la couleur d'écran graphique devient cette couleur.

La couleur d'encre vaut 0 par défaut.

Pour comprendre la différence existant entre CLS et CLG, faites :

10 LOCATE 1,10
20 CLG

et

10 LOCATE 1,10
20 CLS

DRAW <x abs>, <y abs> [, <encre>] [, <mode d'encre>]] sur 6128
DRAW <x abs>, <y abs> [, <encre>] sur 464 et 664

— « encre » est un entier compris entre 0 et 15.

— « mode d'encre » est un entier qui prend les valeurs suivantes : 0 pour le mode normal, 1 pour le mode XOR, 2 pour le mode AND et 3 pour le mode OR.

— « x abs » est un entier compris entre 0 et 639,

— « y abs » est un entier compris entre 0 et 399.

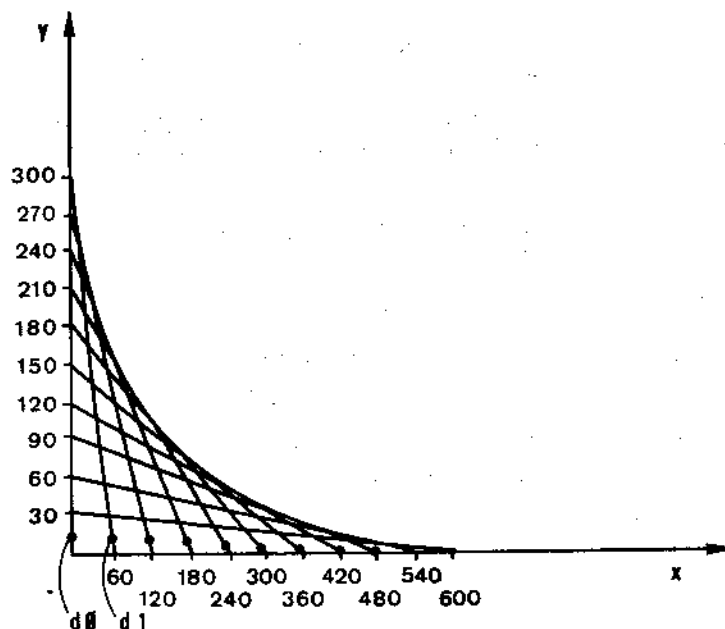
Remarque :

Voir l'utilisation du mode d'encre à l'instruction « PLOT ».

L'ordre DRAW dessine une ligne sur l'écran entre la position courante du curseur graphique et la position absolue indiquée par « x abs » et « y abs ».

Utilisation

Traçons le graphe ci-dessous à l'aide de l'instruction « DRAW ».



La droite d0 sera tracée par MOVE 0, 330: DRAW 0,0

La droite d1 sera tracée par MOVE 0, 300: DRAW 60,0

La droite di sera tracée par MOVE 0, 300 - (i * 30): DRAW (i + 1) * 60,0

d'où le programme suivant :

```
100 CLS
110 FOR I = -1 TO 10
120  MOVE 0,300-I*30
130  DRAW (I+1)*60,0
140 NEXT I
```

DRAWR <décalage en x>, <décalage en y> [, <encre>] [, <mode d'encre>] sur 6128,

DRAWR <décalage en x>, <décalage en y> [, <encre>] sur 464 et 664

- « décalage en x » est un entier compris entre -639 et 639,
- « décalage en y » est un entier compris entre -399 et 399,
- « encre » est un entier compris entre 0 et 15,
- « mode d'encre » est un entier compris entre 0 et 3 : (0=NORMAL, 1=XOR, 2=AND, 3=OR).

Remarque :

Voir l'utilisation du mode d'encre à l'instruction « PLOTS ».

Dessine une ligne sur l'écran à partir de la position courante du curseur graphique vers une position relative : Si x,y est la position courante du curseur, une ligne sera tracée entre x,y et x + décalage en x,y + décalage en y.

Utilisation

L'instruction de remplissage de surfaces à contour fermé existe dans le BASIC 6128, mais pas dans les BASIC 464 et 664. Nous allons la recréer partiellement :

Soit un rectangle défini par ses coordonnées x1,y1 en bas à gauche et x2,y2 en haut à droite.

Pour le remplir, il faudra faire :

- Curseur en x1, y1.
- Tracé d'une ligne horizontale de longueur x2-x1.
- Curseur en x1,y1+1.
- Tracé d'une ligne horizontale de longueur x2-x1.

et ce n fois jusqu'à ce que y1+n=y2.

d'où le programme suivant :

```
100 X1=100:X2=300:Y1=100:Y2=200
110 FOR I=Y1 TO Y2
120  MOVE X1, I
130  DRAWR X2-X1, 0
140 NEXT I
```

MOVE<x abs>, <y abs>[, [<encre>][, <mode d'encre>]] sur 6128

MOVE <x abs>, <y abs> sur 464 et 664

- « x abs » est compris entre 0 et 639,
- « y abs » est compris entre 0 et 399,
- « encre » est compris entre 0 et 15,
- « mode d'encre » est compris entre 0 et 3.
(0=Normal, 1=XOR, 2=AND, 3=OR)

Remarque :

Voir l'utilisation du mode d'encre à l'instruction « PLOTR »

Positionne le curseur graphique aux coordonnées absolues xabs, yabs spécifiées.

MOVER<décalage x>, <décalage y>[, [<encre>][, <mode d'encre>]] sur 6128

MOVER <décalage x>, <décalage y> sur 464 et 664

- « décalage x » est compris entre -639 et 639,
- « décalage y » est compris entre -399 et 399,
- « encre » est compris entre 0 et 15,
- « mode d'encre » est compris entre 0 et 3.
(0=Normal, 1=XOR, 2=AND, 3=OR).

Si les coordonnées courantes du curseur sont x,y, les nouvelles coordonnées seront x+décalage x, y+décalage y.

ORIGIN<x>, <y>[, <gauche>, <droite>, <haut>, <bas>]

Permet de définir :

a) Le point d'origine du curseur graphique.

Remarque :

x=0, y=0 est le point en bas à gauche de l'écran.

b) Les dimensions de la fenêtre graphique.

Remarque :

Si les coordonnées de la fenêtre dépassent la taille de l'écran, c'est l'écran qui est considéré comme fenêtre graphique.

Exemple de définition de fenêtre :

100 CLS : ORIGIN 0,0,100,540,300,100

110 CLG 3 'Affiche la fenêtre graphique d'une autre couleur que le fond de l'écran.

PLOT <x abs>,<y abs>,[,<encre>],[,<mode d'écran>]] sur 6128
PLOT <x abs>,<y abs>,[,<encre>] sur 464 et 664.

- « x abs » est compris entre 0 et 639,
- « y abs » est compris entre 0 et 399,
- « encre » est compris entre 0 et 15,
- « mode d'encre » est compris entre 0 et 3.
 (0=Normal, 1=XOR, 2=AND, 3=OR)

Utilisation du mode d'encre

Lorsque le paramètre mode d'encre est utilisé, l'opération logique est faite entre la couleur du point courant et la couleur du point que l'on veut afficher.

Les tables de vérité au niveau bit des fonctions XOR, AND et OR sont les suivantes :

x OR	0	1
0	0	1
1	1	0

AND	0	1
0	0	0
1	0	1

OR	0	1
0	0	1
1	1	1

Soient a et b deux bits ; nous aurons pour XOR :

$a \text{ XOR } b = 1$ si $a=0$ et $b=1$ ou $a=1$ et $b=0$, donc si $a \neq b$ (puisque a vaut 0 ou 1 et b également).

De même, nous aurons pour AND :

$a \text{ AND } b = 1$ si $a = b = 1$, et $a \text{ AND } b = 0$ dans les autres cas.

et, nous aurons pour OR :

$a \text{ OR } b = 0$ si $a = b = 0$, et $a \text{ OR } b = 1$ dans les autres cas.

Nous pouvons étendre les notions binaires XOR, AND et OR à des entiers en considérant leur décomposition en base 2.

Exemple :

Soient les nombres $a = 12$ 0 0 0 0 1 1 0 0
 et $b = 9$ 0 0 0 0 1 0 0 1
 nous aurons $a \text{ XOR } b$ 0 0 0 0 0 1 0 1 = 5 en décimal
 $a \text{ AND } b$ 0 0 0 0 1 0 0 0 = 8 en décimal
 $a \text{ OR } b$ 0 0 0 0 1 1 0 1 = 13 en décimal

Utilisation du mode d'encre :

En MODE 0, supposons que nous ayons :

INK 5,1 (bleu), INK 8,2 (bleu vif), INK 9,3 (rouge), INK 12,4 (magenta),
 INK 13,5 (mauve).

Si nous faisons :

PLOT x,y,12,0 → un point magenta apparaît

PLOT x,y,9,1 → un point bleu apparaît
mode XOR: $9 \text{ XOR } 12 = 5 = \text{bleu}$

PLOT x,y,12,0 → un point magenta apparaît

PLOT x,y,9,2 → un point bleu vif apparaît
mode AND: $12 \text{ AND } 9 = 8 = \text{bleu vif}$

PLOT x,y,12,0 → un point magenta apparaît

PLOT x,y,9,3 → un point mauve apparaît
mode OR: $12 \text{ OR } 9 = 13 = \text{magenta}$

Remarque :

Le mode d'encre peut être activé dans les ordres : DRAW, DRAWR, PLOT, PLOTR, MOVE, MOVER comme 4^e paramètre.

Utilisation

Tracé des courbes $Y = F(x)$

Le programme se décompose en trois phases :

- saisie de l'équation,
- saisie du domaine de définition,
- tracé.

10 REM Trace de courbes d'équations $Y = F(x)$

20 GOSUB 1000 'Saisie de l'équation

30 STOP

40 GOSUB 2000 'Saisie du domaine

50 GOSUB 3000 'Trace

60 :

70 END

80 REM -----

1000 REM Saisie de l'équation

1010 :

1020 CLS

Y3* 1030 PRINT "Tapez 2060 DEF FNA(X) ="

1040 PRINT "Suivi de l'équation a etudier."

1050 PRINT:PRINT "Tapez ensuite RUN 40"

1060 :

1070 RETURN

1080 REM -----

2000 REM Saisie du domaine de definition

2010 :

2020 CLS

2030 PRINT "Entrez le domaine d'etude:"

2040 PRINT:INPUT "X min"; X1

2050 PRINT:INPUT "X max"; X2

2060 DEF FNA(X) = SIN(X)

2070 :

2080 RETURN

2090 REM -----

3000 REM Trace de la courbe

3010 :

3020 REM Calcul de l'échelle en Y

3030 :

3035 M1 = - 1E+33:M2 = 1E+33

3040 FOR X=X1 TO X2 STEP (X2-X1)/100

3050 A = FNA(X)

3060 IF A > M1 THEN M1 = A

3070 IF A < M2 THEN M2 = A

3080 NEXT X

3090 EX = 640/(X2-X1):EY = 399/(M1-M2)

3100 PX = (X2-X1)/100

3110 :

3120 REM trace

3130 :

3140 CLS

3150 FOR X=X1 TO X2 STEP PX

3160 PLOT (X-X1)*EX, (FNA(X)-M2)*EY

3170 NEXT X

3180 :

3190 RETURN

Lignes 20 à 70 : programme principal

Lignes 1000 à 1070 : Saisie de l'équation

Lignes 2000 à 2080 : Saisie du domaine d'étude

Ligne 2060 : Définition de la fonction à étudier

Lignes 3000 à 3100 : Calcul d'échelle

Lignes 3120 à 3190 : Tracé de la courbe

PLOTR <décalage en x>, <décalage en y>, [, [<encre>], [<mode d'encre>]] sur 6128

PLOTR <décalage en x>, <décalage en y>, [, <encre>] sur 464 et 664

- « décalage x » est compris entre - 639 et 639,
- « décalage y » est compris entre - 399 et 399,
- « encre » est compris entre 0 et 15,
- « mode d'encre » est compris entre 0 et 3.
(0 = Normal, 1 = XOR, 2 = AND, 3 = OR)

Si les coordonnées du curseur graphique étaient x,y « **PLOTR** dec x, dec y » affiche un point aux coordonnées x+dec x, y+dec y.

TEST (<x>, <y>)

- x est compris entre 0 et 639,
- y est compris entre 0 et 399.

Place le curseur graphique à la position absolue x,y et donne la valeur de l'encre à cette position, modulo 16 en MODE 0, modulo 4 en MODE 1 et modulo 2 en MODE 2.

Utilisation

Soient deux points : P1(x1,y1) et P2 (x2,y2) appartenant à une ligne horizontale d'ordonnée Y. Nous voulons tracer un trait entre P1 et P2 sachant qu'ils peuvent être n'importe où sur la droite et que leur couleur est PEN 1 en MODE 2. (Evidemment, la droite n'a pas la couleur PEN 1.)

```
100 FOR I=0 TO 639
```

```
110 IF TEST (I,Y)=1 THEN IF X1=0 THEN X1=I ELSE X2=I
'Recherche de points
```

```
120 NEXT I
```

```
130 PLOT X1,Y
```

```
140 DRAW X2,Y
```

TESTR (<décalage x>, <décalage y>)

- « décalage x » est compris entre - 639 et 639
- « décalage y » est compris entre - 399 et 399.

Si le curseur graphique était en x,y , il sera positionné en $x + \text{décalage } x, y + \text{décalage } y$ après cette commande, et TESTR donnera la couleur de ce point, modulo 16 en MODE 0, modulo 4 en MODE 1 et modulo 2 en MODE 2.

VPOS

Donne la position verticale du curseur texte par rapport à la position d'origine dans la fenêtre courante.

XPOS

Donne la position horizontale du curseur graphique.

La position renvoyée est comprise entre 0 et 639.

YPOS

Donne la position verticale du curseur graphique.

La position renvoyée est comprise entre 0 et 399.

XII. Fonctions à caractère mathématique

ABS (<expression numérique>)

ou « expression numérique » est une valeur entière ou réelle.

Définition :

La valeur absolue d'un nombre « a » est celui des deux nombres a et -a qui est le plus grand.

La fonction « ABS » donne la valeur absolue de l'expression entre parenthèses.

Exemples :

$\text{ABS}(-32.5) = 32.5$
 $\text{ABS}(32.5) = 32.5$

$\text{ABS}(-1456) = 1456$
 $\text{ABS}(3.2 \text{ E } 12) = 3.2 \text{ E } 12$

<Argument 1> AND <Argument 2>

Exécute un « ET » logique entre tous les bits des deux arguments entiers fournis.

La table de vérité de la fonction AND au niveau bit est la suivante :

AND	0	1
0	0	0
1	0	1

Exemple :

```

      1 0 1 1 1 1 0 0 &BC
AND 0 1 1 1 1 1 0 1 &7D
=    0 0 1 1 1 1 0 0 &3C

```

ATN (<expression numérique>)

où « expression numérique » est une valeur réelle.

Définition :

Correspondance réciproque de la fonction tangente. $ATN(x) = b$ où b est compris entre $-\pi/2$ et $\pi/2$ et tel que $TAN(b) = x$.

Remarques :

- Les commandes « DEG » et « RAD » peuvent être utilisées pour spécifier que le résultat doit être exprimé en degrés ou en radians.
- Par défaut, la valeur sera calculée en radians.

BIN\$ (<nombre entier sans signe>[, <nombre entier>])

Donne l'équivalent binaire (base 2) d'un nombre entier décimal (base 10) sur un nombre de digit (de chiffres) indiqué par le paramètre optionnel. Ce nombre peut varier entre 0 et 16. Le nombre à convertir doit être compris entre -32768 et 65535.

Exemple :

```

BIN$(-3,8) = 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1
BIN$(3,8)  =  0 0 0 0 0 0 1 1
BIN$(3,16) =  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1

```

Remarque :

Le codage binaire d'un nombre négatif se fait de la façon suivante :

$$-n = \bar{n} + 1 \text{ sur 16 bits.}$$

Exemple :

```

n=3 sur 16 bits : 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
n̄                1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
n + 1            1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1

```

Ainsi, on a $\text{BIN}\$(-1)$: $1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1$
 $= 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ +\ 1$

CINT (<expression numérique>)

Conversion d'une valeur numérique réelle en une valeur numérique entière arrondie entre -32768 et 32767.

Utilisation

Si vous êtes un peu déconcerté devant l'abondance des instructions permettant d'extraire les parties fractionnaires ou entières d'un nombre réel, voici un programme qui pourra vous fixer les idées sur leur utilisation :

100 REM INT,CINT,CREAL,FIX,ROUND sans argument secondaire

110 :

120 MODE 2

130 PRINT "Test de INT,CINT, CREAL, FIX et ROUND sans argument secondaire" :PRINT:PRINT

140 INPUT "Entrez un nombre"; N

150 PRINT

160 PRINT "INT ("N") = "INT(N)

170 PRINT "CINT ("N") = "CINT(N)

180 PRINT "CREAL ("N") = "CREAL(N)

190 PRINT "FIX ("N") = "FIX(N)

200 PRINT "ROUND ("N") = "ROUND(N)

210 PRINT

220 LOCATE 1,20:INPUT "Un autre essai (O/N):" ; R\$

230 IF UPPER\$(R\$) <> "O" AND UPPER\$(R\$) <> "N" THEN PRINT CHR\$(7):GOTO 220

240 IF UPPER\$(R\$) = "O" THEN 100

250 END

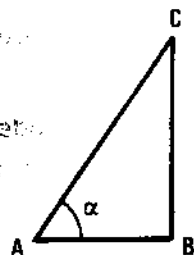
Ligne 140 : Saisie du nombre réel à convertir

Lignes 150 à 210 : Affichage du résultat des fonctions INT, CINT, CREAL, FIX et ROUND.

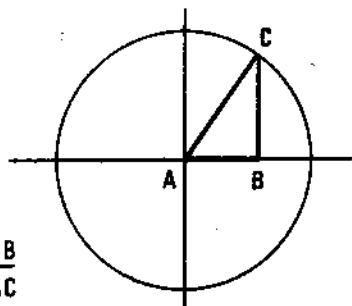
Lignes 220 à 240 : Poursuite ou arrêt du programme.

COS (<expression numérique>)*Définition :*

Rapport du côté adjacent sur l'hypoténuse.



$$\cos \alpha = \frac{AB}{AC}$$

*Remarques :*

- a) Les commandes « DEG » et « RAD » peuvent être utilisées pour spécifier que le résultat doit être exprimé en degrés ou en radians.
- b) Sauf indication contraire, la valeur sera calculée en radians.

CREAL (<expression numérique>)

ou « expression numérique » est un entier ou un réel.

Convertit l'expression numérique en un nombre réel.

Reportez-vous à la fonction « CINT » pour avoir des renseignements sur l'utilisation de « CREAL ».

DEF FN<nom>[(<paramètre(s) formel(s)>)] = <expression>

La fonction DEF FN permet de définir simplement une fonction mathématique qui sera utilisée dans la suite du programme.

Le but de cette fonction est double :

- Exécuter une seule fois la définition de cette formule et l'utiliser par la suite sous forme réduite (gain de place et de temps d'exécution),
- permettre à un utilisateur non programmeur de définir simplement la fonction qui l'intéresse.

Utilisation

Soit une fonction mathématique dont nous voulons connaître les minimum et maximum dans un intervalle donné.

```
1000 REM Recherche du minimum et du maximum
1010 REM d'une fonction sur un intervalle donne
1020 REM
1050 CLS:PRINT"Tapez 1150 DEF FNA(X)="
1060 PRINT"suivi de la fonction a etudier"
1070 PRINT"Tapez ensuite RUN 1100"
1080 STOP
1100 REM Saisie de l'intervalle d'etude
1110 PRINT:PRINT"Entrez l'intervalle d'etude:"
1120 PRINT:INPUT "Minimum";MI
1130 PRINT:INPUT "Maximum";MA
1140 REM
1150 DEF FNA(X) = COS(X)
1160 REM
1170 U = -1E+15:V = 1E+15
1180 FOR X=MI TO MA STEP (MA - MI)/100
1190   B=FNA(X)
1200   IF B>U THEN U=B 'Maximum
1210   IF B<V THEN V=B 'Minimum
1220 NEXT X
1230 REM
1240 REM Affichage des resultats
1250 PRINT:PRINT"Le maximum est :";U
1260 PRINT:PRINT"Le minimum est :";V
1270 END
```

Lignes 1050 à 1080 : Présentation,

Lignes 1100 à 1140 : Saisie de l'intervalle d'étude,

Ligne 1150 : Définition de la fonction,

Lignes 1160 à 1230 : Calcul des minimum et maximum,

Lignes 1240 à 1270 : Affichage des minimum et maximum.

DEG

Etablit le mode de calcul des fonctions trigonométriques en degrés. Par défaut, les valeurs fournies aux fonctions SIN, COS, TAN et ATN sont supposées exprimées en radians.

Remarque :

Les commandes RAD, NEW, CLEAR, LOAD, RUN, CALL O, ... annulent l'effet de la commande « DEG ».

EXP (<expression numérique>)

Donne l'exponentielle naturelle (base e) de l'expression numérique fournie.

Remarques :

a) Pour calculer la puissance dans une base autre que e, (soit a cette base), nous ferons :

$a^x = e^{(x * \text{LOG}(a))}$ où LOG est le logarithme népérien.

b) La fonction réciproque de EXP est le logarithme népérien LOG.

FIX (<expression numérique>)

Enlève la partie décimale d'un nombre réel sans l'arrondir à sa valeur la plus proche.

Reportez-vous à la fonction « CINT » pour avoir des renseignements sur l'utilisation de « FIX ».

HEX\$(<nombre entier sans signe>, [, <nombre entier>])

Le 1^{er} argument est un nombre entier compris entre -32768 et 65535.
Le 2^e argument est un nombre entier compris entre 0 et 16. Il donne le nombre de digit (chiffres) pour la conversion.

Exemples :

HEX\$(250,4)=00FA : Des « 0 » sont rajoutés à gauche du nombre exprimé en hexadécimal pour obtenir le nombre de digit demandé.

HEX\$(250,1)=FA : Impossible à exprimer sur 1 digit.

HEX\$(250,16)= 00000000000000FA

INT (<expression numérique>)

où « expression numérique » est un entier ou un réel.

Arrondit à l'entier inférieur en enlevant la partie décimale.

Le résultat est le même que pour la fonction « FIX » pour les nombres positifs, et de un inférieur à la fonction « FIX » pour les nombres négatifs.

Reportez-vous à la fonction « CINT » pour avoir des renseignements sur l'utilisation de « INT ».

LOG (<expression numérique>)

Définition :

Soit x un nombre strictement positif écrit sous la forme $x = a^y$ où :

- a est un réel différent de 1,
- y est appelé le logarithme de x dans la base a .

Dans le BASIC de l'AMSTRAD, la base est e pour la fonction « LOG » et 10 pour la fonction « LOG10 ».

Remarque :

L'expression numérique fournie à la fonction LOG doit être strictement positive.

LOG10 (<expression numérique>)

D'après la définition donnée à la fonction LOG du logarithme en base a , nous avons, pour tout nombre réel x strictement positif :

$$x = 10^y \rightarrow y = \text{LOG10}(x)$$

Remarque :

L'expression numérique fournie à la fonction LOG doit être strictement positive.

MAX (<liste d'expressions numériques>)

où « liste d'expressions numériques » représente un ensemble de variables ou de valeurs entières ou réelles.

Cette fonction donne la plus grande des expressions fournies.

Exemple :

```
10 A=4  
20 PRINT MAX(3*4,A,1)
```

affichera 12.

MIN(<liste d'expressions numériques>)

où « liste d'expressions numériques » représente un ensemble de variables ou de valeurs entières ou réelles.

Cette fonction donne la plus petite des expressions fournies.

Exemple :

```
10 A=4  
20 PRINT MIN(3*4,A,1)
```

affichera 1.

NOT <argument>

Exécute une inversion logique sur tous les bits de l'argument entier fourni. La table de vérité de la fonction NOT au niveau bit est la suivante :

A	0	1
NOT A	1	0

Exemple :

```

      1 0 1 1 1 1 0 0   &BC
NOT &BC = 0 1 0 0 0 1 1   &43

```

<Argument 1> OR <Argument 2>

Exécute un « OU » logique entre tous les bits des deux arguments entiers fournis.

La table de vérité de la fonction OR au niveau bit est la suivante :

OR	0	1
0	0	1
1	1	1

Exemple :

```

      1 0 1 1 1 1 0 0   &BC
OR    0 1 1 1 1 1 0 1   &7D
=     1 1 1 1 1 1 0 1   &FD

```

PI

Définition :

Rapport de la longueur d'un cercle à son diamètre. C'est un nombre irrationnel transcendant.

Le nombre « PI » sur AMSTRAD donne la valeur approchée du nombre PI : 3.141592653468251.

Remarque :

Les valeurs rationnelles suivantes peuvent suffire dans beaucoup de cas :
 22/7 à 2×10^{-3} près, et 315/113 à 3×10^{-7} près.

RAD

Définition :

Arc dont la longueur égale le rayon du cercle (le cercle entier mesure 2 pi radians).

Cette commande établit le mode de calcul des fonctions trigonométriques en radians.

Remarque :

On a 1 radian = $180/\pi$ degrés,
et 1 degré = $\pi/180$ radians.

RANDOMIZE[<expression numérique>]

Cette instruction permet de fixer la séquence pseudo-aléatoire du générateur de nombres aléatoires de l'AMSTRAD.

Lorsque vous désirez utiliser le générateur aléatoire dans un programme, l'instruction « RANDOMIZE » permet d'avoir des séquences aléatoires différentes.

Si vous utilisez l'argument « expression numérique », la séquence générée sera la même si vous fournissez deux fois le même argument.

Pour fixer l'argument du « RANDOMIZE », vous pouvez demander à la personne qui exécute le programme d'entrer un nombre N, et faire RANDOMIZE N, ou utiliser l'un des programmes suivants qui vous évitera cette démarche un peu lourde.

Attente de l'appui sur une touche. Pendant l'attente, un compteur s'incrémente. Quand une touche est actionnée, le programme génère un RANDOMIZE <Compteur>.

```
120 CLS:PRINT « Appuyez sur une touche »:PRINT
```

```
130 :
```

```
140 A$ = INKEY$
```

```
150 J = J + 1
```

```
160 IF A$ = "" THEN 140
```

```
170 :
```

```
180 RANDOMIZE J
```

```
190 :
```

```
200 FOR I = 1 TO 10
```

```
210 PRINT INT(RND*10) ;
```

```
220 NEXT I
```

```
230 :
```

```
240 END
```

Ligne 120 : Message à l'écran

Lignes 140 à 160 : Attente d'une action au clavier

Ligne 180 : RANDOMIZE

Lignes 200 à 230 : Affichage d'une séquence pseudo-aléatoire.

Utilisation de l'instruction « TIME » dans un RANDOMIZE :

L'instruction « RANDOMIZE TIME » donne d'assez mauvais résultats, et une expression un peu plus complexe a été choisie.

```
120 RANDOMIZE RND(TIME)*1000
```

```
130 FOR I=1 TO 10
```

```
140 PRINT INT(RND*10) ;
```

```
150 NEXT I
```

```
160 :
```

```
170 END
```

Ligne 120 : RANDOMIZE

Lignes 130 à 150 : Affichage d'une séquence pseudo-aléatoire.

RND[<expression numérique>]

Donne un nombre tiré au hasard dans la séquence pseudo-aléatoire courante si le paramètre « expression numérique » n'existe pas ou est positif.

Si l'expression numérique est nulle, RND(0) renvoie le dernier nombre tiré.

Si l'expression numérique est négative, une nouvelle séquence aléatoire est générée.

ROUND (<expression numérique>[, <nombre de décimales>])

Arrondit l'expression numérique au nombre de décimales indiqué dans le deuxième argument.

Si le deuxième argument est négatif et de valeur absolue n, l'expression est arrondie à un entier dont les n derniers chiffres sont nuls.

Exemple :

```
ROUND(12343.656, -3) = 12000
```

```
ROUND(12343.656, 2) = 12300
```

```
ROUND(12343.656, 0) ou ROUND (12343.656) = 12344
```

```
ROUND(125, -4) = 0
```

Reportez-vous à l'ordre « CINT » pour avoir des exemples de valeurs de ROUND.

SGN(<expression numérique>)

Donne le signe de l'expression numérique :

- 1 si elle est négative, 0 si elle est nulle et 1 si elle est positive.

Exemples :

```
SGN(-12.5) = -1
```

```
SGN(0) = 0
```

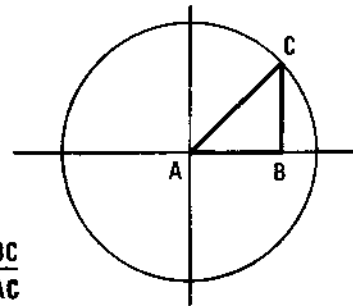
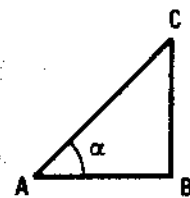
```
SGN (1.4E30) = 1
```

SIN (<expression numérique>)*Définition :*

Pour un angle aigu, rapport du côté opposé sur l'hypoténuse.

Remarques :

- a) Les commandes « DEG » et « RAD » peuvent être utilisées pour spécifier que le résultat doit être exprimé en degrés ou en radians.
- b) Sauf indication contraire, la valeur sera calculée en radians.



$$\sin \alpha = \frac{BC}{AC}$$

SQR (<expression numérique>)

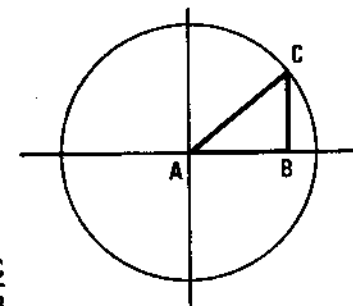
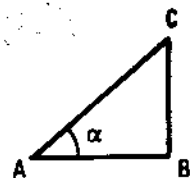
où « expression numérique » est un nombre entier ou réel.

*Définition :*a est appelé racine d'un nombre A si $a^2 = A$.

Cette fonction donne la racine d'un nombre réel ou entier.

TAN (<expression numérique>)*Définition :*

Rapport du sinus sur le cosinus.



$$\tan \alpha = \frac{BC}{AB}$$

$\sin = BC/AC$ et $\cos = AB/AC$,
 d'où $\tan = \sin/\cos = (BC/AC)/(AB/AC) = BC/AB$

Remarques :

- a) Les commandes « DEG » et « RAD » peuvent être utilisées pour spécifier que le résultat doit être exprimé en degrés ou en radians.
- b) Sauf indication contraire, la valeur sera calculée en radians.

UNT(<nombre hexadécimal>)

Convertit l'argument en un nombre entier signé sur 16 bits en complément à 2.

Exemples :

UNT(&FFFF) = -1

UNT(&FFFE) = -2

UNT(0) = 0

Définition du complément à 2 :

on a $-n = \bar{n} + 1$ où n est un entier positif

par exemple :

$$-3 = \bar{3} + 1$$

$$3 = 0000000000000011$$

$$\bar{3} = 1111111111111100$$

$$3 + 1 = 1111111111111101 = \&FFFD \text{ en hexadécimal}$$

<Argument 1> XOR <Argument 2>

Exécute un « OU EXCLUSIF » logique entre tous les bits des deux arguments entiers fournis.

La table de vérité de la fonction XOR au niveau bit est la suivante :

XOR	0	1
0	0	1
1	1	0

Exemple :

$$\begin{array}{rcl}
 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & \&BC \\
 \text{XOR} & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & \&7D \\
 = & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & \&C1
 \end{array}$$

XIII. Aides à la programmation**AUTO[<N° de ligne>[, <écart entre deux lignes>]]**

Affiche automatiquement les numéros de lignes après chaque retour ligne pour un programme BASIC.

Remarques :

a) Sur 464 et 664, si une ligne est déjà en mémoire, un « * » apparaît après son numéro pour indiquer le conflit. Tapez « Enter » si vous voulez conserver l'ancienne ligne.

Sur 6128, si une ligne est déjà en mémoire, elle apparaît en clair et est modifiable comme par la commande « EDIT ». Tapez « Enter » si vous voulez conserver l'ancienne ligne.

b) Si vous ne donnez aucun argument à cette commande, la numérotation commence en ligne 10 et l'écart entre deux lignes est de 10.

CLEAR

Efface des variables BASIC entières, réelles, tableaux et fonctions utilisateur, ferme les fichiers ouverts, et force le calcul des fonctions trigonométriques en radians.

CONT

Permet de continuer l'exécution d'un programme après avoir rencontré l'instruction « STOP » ou après que la touche « ESC » ait été pressée deux fois.

Remarques :

a) Si le programme est modifié entre son arrêt et la demande de redémarrage, ou s'il est protégé, la commande « CONT » sera mise en défaut.

b) L'utilisation des ordres « STOP » et « CONT » est très efficace pour aider à la mise au point des programmes écrits en BASIC. En effet, si le programme à tester est de grande taille, il pourra être judicieux de le tester par petits blocs en y insérant des commandes « STOP ».

DEFINT < liste de lettres concernées >

Définit le type des variables par défaut comme étant entier. Si une variable est entrée sans marqueur du type « ! » pour un nombre binaire, « % » pour un entier et « \$ » pour une chaîne, elle est automatiquement définie entière si sa première lettre commence par une des lettres citées dans la commande « DEFINT ».

Exemple :

DEFINT A → Toutes les variables de nom commençant par « A » et non marquées seront de type entier.

DEFINT A-Z → Toutes les variables non marquées seront de type entier.

DEFREAL < Liste de lettres concernées >

Définit le type des variables par défaut comme étant réel. Si une variable

est entrée sans marqueur du type « ! » pour un nombre binaire, « % » pour un entier et « \$ » pour une chaîne, elle est automatiquement définie réelle si sa première lettre commence par une des lettres citées dans la commande « DEFREAL ».

Exemple :

DEFREAL A → Toutes les variables de nom commençant par « A » et non marquées seront de type réel.

DEFREAL A-Z → Toutes les variables non marquées seront de type réel.

DEFSTR <liste de lettres concernées>

Définit le type des variables par défaut comme étant alphanumérique. Si une variable est entrée sans marqueur du type « ! » pour un nombre binaire, « % » pour un entier et « \$ » pour une chaîne, elle est automatiquement définie chaîne si sa première lettre commence par une des lettres citées dans la commande « DEFSTR ».

Exemple :

DEFSTR A → Toutes les variables de nom commençant par « A » et non marquées seront de type alphanumérique.

DEFSTR A-Z → Toutes les variables non marquées seront de type alphanumérique.

DELETE <une ligne ou un ensemble de lignes>

Efface la ou les ligne(s) spécifiée(s).

Exemples :

DELETE Efface la totalité des lignes en mémoire.
Identique à la commande « NEW » à ceci près que DELETE n'efface pas la valeur des variables en mémoire.

DELETE a - b Efface toutes les lignes comprises entre « a » et « b ».

DELETE - a Efface toutes les lignes du début du programme jusqu'à la ligne « a ».

DELETE a - Efface toutes les lignes à partir de la ligne « a » jusqu'à la fin du programme.

Remarque :

Utilisez cette commande avec précaution, car il n'est pas possible de récupérer les lignes effacées par une simple commande BASIC.

EDIT <N° de ligne>

Affiche une des lignes d'un programme BASIC en mémoire. Le curseur

est positionné sur le premier caractère du premier mot BASIC de la ligne, et le mode d'édition est lancé : toute lettre tapée sera insérée avant le curseur et les caractères suivant le curseur seront décalés d'un caractère vers la droite.

Remarques :

- a) Lors de l'exécution d'un programme BASIC, s'il survient une erreur, le mode EDIT est lancé automatiquement sur la ligne ayant provoqué l'erreur.
- b) Si vous tentez d'éditer une ligne qui n'existe pas, le message suivant apparaîtra : « Line does not exist ».

END

Marque la fin d'un programme. Plusieurs ordres « END » peuvent être présents dans un programme, mais il n'y en aura qu'un si vous utilisez les concepts de programmation hiérarchisée-structurée exposés en partie 4 chapitre 1.4.

LIST[<ensemble de lignes>][. <numéro de canal>]

où <ensemble de lignes> peut prendre les valeurs suivantes :

Inexistant : liste la totalité des lignes du programme en mémoire.

– a : liste toutes les lignes du début du programme à la ligne a.

a – : liste toutes les lignes du programme à partir de la ligne a jusqu'à la fin du programme.

a – b : liste toutes les lignes comprises entre a et b.

et « numéro de canal » = #0 pour lister sur l'écran,

= #8 pour lister sur imprimante.

Cette commande permet d'afficher une ligne ou un ensemble de lignes sur le canal spécifié.

NEW

Efface le programme et les données en mémoire et ferme tous les fichiers ouverts. Les touches redéfinies ne sont pas effacées.

Remarque :

Utilisez cette commande avec précaution, car il n'est pas possible de récupérer les lignes effacées par une simple commande BASIC.

REM <texte>

Abréviation du mot anglais « remark ». Permet d'insérer des commentaires dans un programme BASIC.

Remarques :

- a) Tous les caractères rencontrés après la commande « REM » sont considérés comme des commentaires, y compris le séparateur d'instructions « : » qui ne permet pas de valider une instruction derrière un REM.
- b) L'ordre REM peut être remplacé par une apostrophe, mais il produira une erreur s'il est utilisé de cette façon dans un ordre DATA.

RENUM[<nouveau numéro de ligne>][, <ancien numéro de ligne>][, <incrément>]

où « ancien numéro de ligne » est la 1^{re} ligne touchée par le RENUM, « nouveau numéro de ligne » définit le numéro de la 1^{re} ligne à la sortie de la commande RENUM.

« incrément » définit l'espacement entre deux lignes consécutives.

Pour comprendre la façon dont fonctionne cette commande, le plus simple est de raisonner sur un exemple.

Soit le programme suivant :

10 REM Programme principal

20 :

30 GOSUB 80 'Entree

40 GOSUB 100 'Traitement

50 GOSUB 120 'Sortie

60 :

70 END

80 REM Entree

90 RETURN

100 REM Traitement

110 RETURN

120 REM Sortie

130 RETURN

que nous voulons renuméroter de la façon suivante :

10 REM Programme principal

20 :

30 GOSUB 1000 'Entree

40 GOSUB 2000 'Traitement

50 GOSUB 3000 'Sortie

60 :

70 END

1000 REM Entree

1100 RETURN

2000 REM Traitement

2100 RETURN

3000 REM Sortie

3100 RETURN

Il faudra faire RENUM 1000, 80, 100.

Cette commande aura l'effet suivant :

ab ordonné neiens> .H

10 REM Programme principal

20 :

30 GOSUB 1000 'Entree

40 GOSUB 1200 'Traitement

50 GOSUB 1400 'Sortie

60 :

70 END

1000 REM Entree

1100 RETURN

1200 REM Traitement

1300 RETURN

1400 REM Sortie

1500 RETURN

Puis RENUM 2000, 1200, 100

Cette commande aura l'effet suivant :

10 REM Programme principal

20 :

30 GOSUB 1000 'Entree

40 GOSUB 2000 'Traitement

50 GOSUB 2200 'Sortie

60 :

70 END

1000 REM Entree

1100 RETURN

2000 REM Traitement

2100 RETURN

2200 REM Sortie

2300 RETURN

Et enfin **RENUM 3000, 2200, 100.**

Et nous retrouverons la numérotation voulue :

10 REM Programme principal

20 :

30 GOSUB 1000 'Entree

40 GOSUB 2000 'Traitement

50 GOSUB 3000 'Sortie

60 :

70 END

1000 REM Entree

1100 RETURN

2000 REM Traitement

2100 RETURN

3000 REM Sortie

3100 RETURN

RUN < chaîne alphanumérique >

Charge et exécute un programme présent sur unité de cassette sur 464 ou de disquette (selon la direction des entrées/sorties définie par ICAS ou I DISC).

Remarque :

Les programmes BASIC protégés doivent être exécutés par cette commande directement, sans être chargés en mémoire par l'instruction « LOAD ».

RUN[< N° de ligne >]

Exécute un programme BASIC présent en mémoire à partir de la ligne indiquée ou à partir de la première ligne si aucun numéro de ligne n'est précisé.

STOP

Cet ordre arrête l'exécution d'un programme quand il est rencontré. Utilisé conjointement à la commande CONT, cet ordre permet de « debugger » les programmes BASIC que vous mettez au point.

TRON

Permet de suivre l'exécution d'un programme BASIC ligne par ligne en affichant entre crochets le numéro de la ligne en cours d'exécution.

Cette commande peut être activée par programme ou en mode direct. TRON est à utiliser avec TROFF, STOP et CONT pour aider à mettre au point vos programmes BASIC.

TROFF

Dévalide l'utilisation du mode « trace » lancé par l'ordre TRON. Cette commande peut être activée par programme ou en mode direct.

XIV. Divers

DIM <une ou plusieurs variables>

Dimensionnement d'un ou de plusieurs tableaux. Le ou les tableaux spécifié(s) peuvent contenir des entiers, des réels, ou des chaînes alphanumériques selon l'indicateur accolé à leur nom comme suit :

% indique que le tableau ne pourra contenir que des entiers,

\$ indique que le tableau pourra contenir des chaînes alphanumériques.

Par exemple :

DIM A%(50) déclare un tableau d'entiers de 50 valeurs,

DIM A(50) déclare un tableau de 50 valeurs qui pourra contenir des nombres entiers ou réels.

DIM A\$(50) déclare un tableau de 50 valeurs alphanumériques. Les tableaux peuvent être multidimensionnés. Dans ce cas, chaque dimension sera indiquée dans la parenthèse, séparée des autres par une virgule. Par exemple DIM VIR\$(10, 12, 4) déclare un tableau à trois dimensions qui pourra contenir $10 * 12 * 4 = 480$ éléments alphanumériques. Chaque élément sera accédé par VIR\$(x, y, z) où $x \leq 10$, $y \leq 12$, $z \leq 4$.

FOR <variable> = début TO fin [STEP <pas>]

Exécute une ou plusieurs commande(s) répartie(s) sur un ou plusieurs numéro(s) de ligne entre le mot-clé « FOR » et le mot-clé « NEXT » n fois, où $n = \text{INT}((\text{fin-début})/\text{pas})$.

Remarque :

L'argument « pas » peut être positif ou négatif, du moment que l'expression $((\text{fin-début})/\text{pas})$ est positive.

IF <expression logique> THEN <instruction(s) 1> [ELSE <instruction(s) 2>]

où « expression logique » est une entité qui doit être vraie ou fausse, par exemple $A = 5$ ou $\text{MID}\$(A$, 6, 2) > "A"$, etc..

« instruction(s) 1 » et « instruction(s) 2 » représentent un ou plusieurs ordre(s) BASIC.

Si « expression logique » est vraie, « instruction(s) 1 » sera exécuté.
Si « expression logique » est fausse, « instruction(s) 2 » sera exécuté si ELSE est présent. Sinon, aucune action ne se produira.

LET <variable> = <expression>

où « variable » est du type entier, réel ou chaîne
et « expression » est du même type que « variable ».
Affecte une expression à une variable.

Cet ordre peut être omis et, par exemple, A=4 sera équivalent à LET A=4.

NEXT[<Une ou plusieurs variables>]

Marque la fin d'une boucle FOR. Si aucun nom de variable n'est déclaré, la variable par défaut sera celle qui a été déclarée dans le dernier FOR rencontré. Plusieurs variables peuvent être précisées. Dans ce cas, elles doivent être séparées par des virgules.

WAIT <Adresse port>, <masque>[, <octet de sélection>]

Lit le contenu du port dont l'adresse est spécifiée. Soit A ce contenu. L'ordre « WAIT » fait (A AND <masque>) XOR (<octet de sélection>) et ne rend la main à l'interpréteur BASIC que lorsque le résultat est non nul.

Exemples :

Attente jusqu'à ce que le bit 4 du port &3000 soit <> 0 : WAIT &3000,4.
Attente jusqu'à ce que le bit 4 du port &3000 soit égal à 0 : Wait &3000, 4, 4.

WEND

Marque la fin d'une section d'instructions répétitives qui a débuté par l'ordre WHILE.

WHILE <expression logique>

où « expression logique » est une entité qui doit être vraie ou fausse : par exemple A=5 ou MID\$(A\$, 6, 2)>"A", etc.

Répète une section d'instructions tant que « expression logique » est vraie.

Exemple :

Répéter le calcul de la décomposition polynomiale $\sum_{n=1}^{\alpha} \frac{n}{2^n}$ tant que le taux d'erreur est supérieur à 0.001.

```
1000 X = 0:N = 1:TE = 1
1010 WHILE TE > 0.001
1020  X = X + N/(2^N)
1030  N = N + 1
1040  TE = N/(2^N)
1050 WEND
1060 PRINT X;"a";TE;"pres."
```

WIDTH <nombre entier>

Donne le nombre maximum de caractères par ligne lors d'une sortie sur imprimante. La valeur par défaut est 132.

WINDOW # <N° de canal>,]<gauche>, <droite>, <haut>, <bas>

Donne le numéro et la dimension de la fenêtre d'écran.

Par défaut, les paramètres <gauche>, <droite>, <haut> et <bas> prennent les valeurs suivantes :

```
1, 20, 1, 25 en MODE 0
1, 40, 1, 25 en MODE 1
1, 80, 1, 25 en MODE 2
```

Le numéro de canal vaut #0 par défaut, et peut prendre une valeur comprise entre 0 et 7.

WINDOW SWAP <N° de canal>, <N° de canal>

Définit le numéro de fenêtre courante où vont s'afficher tous les messages du BASIC.

Par exemple, si vous voulez que ce soit la fenêtre 5, il faudra faire WINDOW SWAP 0,5.

4/1.3

Version 1.1 sur CPC 664 et CPC 6128 : Extensions par rapport à la version 1.0 du CPC 464

Quatre fonctions sont rajoutées et sept instructions, soit onze mots-clés.

FONCTIONS

COPYCHR\$(# <Numéro de canal>)

Copie un caractère (qui se trouve sur l'écran, à l'endroit pointé par le curseur texte) sur le canal spécifié.

Le canal peut être l'écran (#0) ou l'imprimante (#8).

Utilisation

Cette commande peut être très utile si vous décidez d'écrire un logiciel d'édition de textes en pleine page : la fonction COPY des CPC pourra être reproduite grâce à la fonction COPYCHR\$. Le caractère à copier sera lu et stocké dans une variable (par exemple, C\$) par C\$ = COPYCHR\$(#0), puis recopié à la position voulue en positionnant le nouveau curseur par l'ordre LOCATE.

Ces diverses actions pourront se faire de la manière suivante :

```
1000 REM Nous supposons ici que le caractere a copier est en
```

```
1001 REM X1, Y1 et l'endroit ou le copier en X2, Y2
```

```
1010 C$ = COPYCHR$(#0) 'Memorisation du caractere a copier
```

```
1020 LOCATE X2, Y2:PRINT C$ 'Copie du caractere
```

```
1030 LOCATE X1, Y1 'Retour à la position initiale du curseur
```

DEC\$(<expression numérique>, <format>)

— <expression numérique> est un nombre entier ou réel,

— **<format>** appartient à l'ensemble des caractères suivants :
+ - £ \$ * # , . ^

Reportez-vous à l'ordre PRINT USING (voir Partie 4 chap. 1.2 p. 11) pour connaître la signification des signes précédents.

Cette fonction permet de définir une donnée numérique selon un certain format : n chiffre(s) avant le point décimal, p chiffre(s) après le point décimal.

Exemples :

```
A$=DEC$(DONNEE, "#####.####")
```

```
PRINT DEC$(DONNEE, "*$###.####")
```

DERR

Donne le dernier code d'erreur renvoyé par le driver de disquettes.

Les erreurs possibles sont les suivantes :

0 ou 22	Activation de ESCape pendant un accès disque
142	Etat du canal incohérent
143	Fin de fichier matérielle
144	Mauvaise commande
145	Fichier déjà existant
146	Fichier non existant
147	Catalogue saturé
148	Disquette pleine
149	Disquette changée sans refermer le(s) fichier(s) ouvert(s)
150	Fichier accessible en lecture seulement
154	Fin de fichier logicielle

SPC(<Nombre entier>)

Définit un nombre d'espaces à afficher dans l'instruction PRINT.

Exemple :

```
PRINT SPC(12) "Douze espaces affichés avant le texte"
```

INSTRUCTIONS

CLEAR INPUT

Efface les données entrées au clavier et se trouvant dans le buffer clavier.

Utilisations

Dans un jeu où la boucle principale comporte une lecture du clavier, on peut éliminer le buffer clavier en utilisant CLEAR INPUT si cela est nécessaire.

Supposons qu'un programme doive faire n traitements dans un temps limité. Un de ces traitements porte sur la lecture du clavier. Ayant fait une estimation de la durée de chaque traitement, on sait qu'on ne pourra traiter que p caractères par seconde. L'ordre CLEAR INPUT peut servir à éliminer les données exédentaires situées dans le buffer clavier.

CURSOR[<Système>][, <Utilisateur>]

<Système> et <Utilisateur> valent 0 ou 1.

Active ou désactive le pavé curseur système ou utilisateur.

Un des deux indicateurs peut être omis ; il prend alors sa valeur par défaut.

FILL <Numéro d'encre>

Remplit une zone de l'écran délimitée par un contour fermé. Si le contour n'est pas fermé, tout l'écran est rempli par la couleur d'encre spécifiée. Le remplissage commence à partir de la position courante du curseur graphique.

FRAME

Synchronise l'affichage graphique avec les trames d'affichage vidéo. De cette manière, l'effet de « flash » est supprimé lors d'un affichage.

Cet ordre est couramment employé dans les jeux où plusieurs objets se déplacent sur l'écran sans distorsion ni scintillement.

GRAPHICS PAPER <encre>

<encre> est compris entre 0 et 15.

Définit la couleur du fond graphique. C'est cette couleur qui sera utilisée pour remplir l'écran lorsqu'un ordre CLG sera exécuté.

GRAPHICS PEN[<encre>][, <mode d'affichage du fond>]

<encre> est compris entre 0 et 15,

<mode d'affichage du fond> vaut 0 pour un fond opaque, et 1 pour un fond transparent.

Définit la couleur du stylo graphique courant utilisé pour réaliser des graphismes sur l'écran (affichage de points et lignes).

Pour mieux comprendre la fonction du paramètre **<mode d'affichage du fond>**, affichez des caractères avec l'ordre TAG en mode opaque et transparent.

MASK[<entier>][, <premier point tracé ou non>]

<entier> est compris entre 0 et 255,

<premier point tracé ou non> vaut 0 ou 1.

Détermine le mode de tracé des lignes affichées par DRAW.

Le paramètre **<entier>** est à considérer comme un nombre sur 8 bits où chaque bit indique si un pixel est allumé (1) ou non (0).

Le paramètre **<premier point tracé ou non>** indique si le premier point doit être affiché (1) ou non (0).

ON BREAK CONT

Permet de dévalider la prise en compte de l'appui sur la touche BREAK.

Cet ordre doit être utilisé avec prudence lors de la mise au point de vos programmes : il n'existe aucun moyen de stopper un programme qui est lancé si cette commande a été exécutée, si ce n'est l'appui simultané sur les touches CONTROL, SHIFT et ESC, ce qui détruit le programme en mémoire !

Nous vous conseillons de rajouter cette instruction en début de programme après l'avoir sauvegardé, ou si vous êtes sûr qu'il s'agit de la version définitive.

GRAPHICS PEN[<entier>][, <mode d'affichage du fond>]

4/1.4

Rappel des ordres BASIC et de leur fonction

Numéro de page

- | | |
|----|---|
| 70 | ABS(<expression numérique>)
Donne la valeur absolue de l'expression numérique. |
| 53 | AFTER<entier>[, <No compteur>] GOSUB <numéro de ligne>
Va en « numéro de ligne » après <entier> * 50 sec. |
| 71 | <Argument> AND <Argument>
ET logique entre les deux arguments |
| 30 | ASC(<chaîne alphanumérique>)
Donne le code ASCII du 1 ^{er} caractère de la chaîne alphanumérique. |
| 71 | ATN(<expression numérique>)
Donne l'arc tangente de l'expression numérique. |
| 81 | AUTO[<premier N° de ligne>][, <différence>]
Numérote automatiquement les lignes d'un programme BASIC en phase de saisie. |
| 71 | BIN\$(<nombre entier sans signe>[, <nombre entier>])
Convertit en base 2 un nombre exprimé en base 10. |
| 2 | BORDER<numéro de couleur 1>[, <numéro de couleur 2>]
Définit la ou les couleur(s) de la bordure d'écran. |
| 48 | CALL<adresse>[, <liste de paramètres>]
Transmet le contrôle à un programme assembleur en lui communiquant ou non des arguments. |

Numéro
de page

- 25 **CAT**
Donne le répertoire des fichiers présents sur l'unité magnétique.
- 25 **CHAIN**<nom du dossier>[, <numéro de ligne>]
Charge un programme en effaçant le programme en mémoire.
- 25 **CHAIN MERGE**<nom du dossier>[, <numéro de ligne>] [.DELETE
<ensemble de lignes>]
Amalgame un programme sur support magnétique et un programme présent en mémoire.
- 31 **CHR\$**(<nombre entier>)
Donne le caractère alphanumérique correspondant au code ASCII entré.
- 72 **CINT**(<expression numérique>)
Convertit l'expression numérique en un entier.
- 82 **CLEAR**
Efface les variables mémoire et ferme les fichiers ouverts.
- 62 **CLG**[<encre masquée>]
Efface l'écran graphique.
- 27 **CLOSEIN**
Ferme un fichier ouvert en lecture.
- 27 **CLOSEOUT**
Ferme un fichier ouvert en écriture.
- 3 **CLS**[<numéro de canal>]
Efface l'écran (ou la fenêtre spécifiée par son numéro de canal).
- 82 **CONT**
Reprend l'exécution d'un programme après un BREAK ou un STOP.
- 73 **COS**(<expression numérique>)
Calcule le COSinus de l'expression numérique fournie.
- 73 **CREAL**(<expression numérique>)
Convertit une valeur en un nombre réel.
- 38 **DATA**<liste de constantes>
Déclare une ou plusieurs données constantes.

**Numéro
de page**

- 73 DEF FN<nom> [(<paramètre(s) formel(s)>)] = <expression mathématique>**
Définit une fonction mathématique.
- 82 DEFINT<type de lettre(s) concernée(s)>**
- 83 DEFSTR<type de lettre(s) concernée(s)>**
- 82 DEFREAL<type de lettre(s) concernée(s)>**
Définit le type des variables non marquées par défaut.
- 74 DEG**
Définit le mode de calcul des fonctions trigonométriques.
- 83 DELETE(<ensemble de lignes>)**
Efface une ou plusieurs ligne(s) BASIC.
- 54 DI**
Dévalide les interruptions BASIC autres que BREAK.
- 88 DIM<liste de variables>**
Réserve de l'espace en RAM pour déclarer un ou plusieurs tableau(x).
- 63 DRAW<coordonnée en X>, <coordonnée en Y>[, <encre masquée>]**
Dessine une ligne sur l'écran.
X et Y sont des coordonnées absolues.
- 64 DRAWR<offset en X>, <offset en Y>[, <encre masquée>]**
Dessine une ligne sur l'écran.
X et Y sont des coordonnées relatives.
- 83 EDIT<numéro de ligne>**
Permet la modification d'une ligne BASIC.
- 54 EI**
Rétablit le fonctionnement des interruptions BASIC.
- 84 END**
Marque la fin d'un programme.

**Numéro
de page**

- ENT<N° d'enveloppe> [, <sections d'enveloppe>]**
Définit une enveloppe de ton.
- ENV<N° d'enveloppe> [, <sections d'enveloppe>]**
Définit une enveloppe de volume.
- EOF**
Fonction de test de fin de fichier.
- ERASE<liste de variables>**
Efface l'espace occupé par une ou plusieurs variable(s).
- ERR**
Donne le numéro d'erreur.
- ERL**
Donne le numéro de ligne ayant provoqué l'erreur.
- ERROR<nombre entier>**
Permet de définir une ou plusieurs erreurs non répertoriées dans le BASIC.
- EVERY<entier> [, <N° de compteur>] GOSUB <numéro de ligne>**
Provoque l'exécution d'un programme toutes les <entier> * 50 sec.
- EXP<expression numérique>**
Donne l'EXponentielle de l'expression numérique fournie.
- FIX(<expression numérique>)**
Donne la valeur arrondie de l'expression numérique fournie.
- FOR<variable> = <début> TO <fin> [STEP<pas>]**
Exécute un bloc d'instruction INT ((fin-début)/pas) fois.
- FRE(<expression numérique>)**
Donne la quantité de RAM libre.
- FRE(<expression alphanumérique>)**
Donne la quantité de RAM libre après réarrangement.
- GOSUB<numéro de ligne>**
Lance l'exécution d'un sous-programme à partir du « numéro de ligne » spécifié.

**Numéro
de page**

- 42 GOTO <numéro de ligne>**
Déroute l'exécution séquentielle d'un programme en exécutant la ligne spécifiée.
- 75 HEX\$ <entier> [, <nombre de digit>]**
Convertit un nombre (exprimé en base 2, 10 » ou 16) en base 16.
- 49 HIMEM**
Donne l'adresse de l'octet le plus élevé utilisé par le BASIC.
- 88 IF <expression logique> THEN <option 1> [ELSE <option 2>]**
Si l'expression logique est vraie, exécute option 1, sinon exécute option 2.
- 88 IF <expression logique> GOTO <ligne> [ELSE <option>]**
Si l'expression logique est vraie, va à la ligne spécifiée, sinon exécute l'option.
- 4 INK <encre> , <couleur> [, <couleur>]**
Définit la couleur d'une encre.
- 15 INKEY(<nombre entier>)**
Indique si une touche particulière du clavier est pressée.
- 17 INKEY\$**
Lit le clavier et donne le code alphanumérique de la touche pressée.
- 50 INP(<numéro de port>)**
Donne la valeur d'un port défini en entrée.
- 18 INPUT[# <N° de canal> , [;] <chaîne> ;] <liste de variables> , ou
INPUT[# <N° de canal> , [;] <chaîne> ,] <liste de variables>**
Lit les données en provenance du canal indiqué.
- 32 INSTR([<nombre entier> ,] <chaîne 1> , <chaîne 2>)**
Cherche si la deuxième chaîne fait partie de la première.
- 75 INT(<expression numérique>)**
Donne la valeur arrondie de l'expression numérique.
- 19 JOY(<nombre entier>)**
Lit le bit significatif du JOYSTICK.

**Numéro
de page**

- 20 KEY[<nombre entier>, [CHR\$(n) +]<chaîne>[+ CHR\$(n)]**
Définit une touche de fonction.
- 20 KEY DEF <N° de touche>, <répétition>[, <normal>[, <avec SHIFT>[, <avec CTRL>]]]**
Définit le caractère associé à une touche du clavier.
- 34 LEFT\$(<chaîne>, <nombre entier>)**
Donne les <nombre entier> caractères les plus à gauche de <chaîne>.
- 34 LEN(<chaîne>)**
Donne le nombre de caractères de la chaîne fournie.
- 89 LET(<expression logique d'égalité>)**
Affecte une valeur à une variable.
- 22 LINE INPUT[# <numéro de canal>,][;][<chaîne> ;]<variable chaîne>**
Lit une ligne sur le canal spécifié.
- 84 LIST[<ensemble de lignes>][, # <numéro de canal>]**
Liste une ou plusieurs ligne(s) sur le canal spécifié.
- 29 LOAD<nom de fichier>[, <adresse>]**
Lit un programme sur K7 ou disquette.
- 5 LOCATE[# <N° de canal>,]<coordonnée en X>, <coordonnée en Y>**
Déplace le curseur texte vers la position X, Y spécifiée.
- 75 LOG(<expression numérique>)**
Calcule le logarithme népérien de l'expression numérique fournie.
- 75 LOG10(<expression numérique>)**
Calcule le logarithme en base 10 de l'expression numérique fournie.
- 34 LOWER\$(<chaîne alphanumérique>)**
Tous les caractères majuscules de la chaîne alphanumérique sont changés en caractères minuscules identiques.
- 76 MAX(<liste d'expressions numériques>)**
Donne la valeur la plus grande de la liste fournie.

**Numéro
de page**

- 50 **MEMORY** <adresse>
Définit l'adresse la plus haute, utilisable par le BASIC.
- 29 **MERGE**[<nom de fichier>]
Amalgame un programme présent en mémoire et un programme lu sur K7 ou disquette.
- 36 **MID\$**(<chaîne>, <entier 1>[, <entier 2>])
Extrait de la chaîne fournie une sous-chaîne commençant au caractère <entier 1> et de longueur <entier 2>.
- 76 **MIN**(<liste d'expressions numériques>)
Donne la plus petite des expressions numériques fournies.
- 7 **MODE** <nombre entier>
Définit le MODE d'affichage sur l'écran (20, 40 ou 80 colonnes).
- 65 **MOVE** <coordonnée X>, <coordonnée Y>
Positionne le curseur graphique aux coordonnées absolues X, Y.
- 65 **MOVER** <offset X>, <offset Y>
Positionne le curseur graphique aux coordonnées relatives X, Y.
- 84 **NEW**
Efface le programme et les variables en mémoire.
- 76 **NOT** <argument>
Négation logique.
- 89 **NEXT**[<liste de variables>]
Marque la fin d'une boucle définie par « FOR ».
- 42 **ON** <expression entière> **GOSUB** <liste de N° de ligne>
- 43 **ON** <expression entière> **GOTO** <liste de N° de ligne>
Déroute l'exécution séquentielle d'un programme vers le n^{ème} numéro de ligne spécifié, où n est la valeur de l'<expression entière> fournie.
- 44 **ON BREAK GOSUB** <N° de ligne>
Appelle un sous-programme lorsqu'un BREAK est activé.

**Numéro
de page****ON BREAK STOP**

Annule les autres commandes ON BREAK.

47 ON ERROR GOTO <N° de ligne>

Définit le point d'entrée d'un programme de gestion d'erreurs.

61 ON SQ GOSUB <N° de ligne>

Définit le point d'entrée d'un sous-programme de gestion de file sonore.

27 OPENIN <nom de fichier>

Ouvre un fichier en lecture.

27 OPENOUT <nom de fichier>

Ouvre un fichier en écriture.

76 <argument> OR <argument>

OU logique entre les deux arguments.

65 ORIGIN <X>'<Y>[, <gauche>, <droite>, <haut>, <bas>]

Détermine le point de référence du curseur graphique.

50 OUT <N° de canal>, <donnée entière>

Envoie la donnée entière définie sur 8 bits vers le canal spécifié.

8 PAPER[# <N° de canal>,]<encre masquée>

Définit la couleur de fond d'écran.

PEEK(<adresse>)

Donne le contenu de la mémoire dont l'adresse est spécifiée.

9 PEN[# <N° de canal>,]<encre masquée>

Détermine le numéro de stylo utilisé.

77 PI

Valeur approchée de PI (3.141592653468251).

66 PLOT <coordonnée X>, <coordonnée Y>[, <encre masquée>]

Affiche un point aux coordonnées absolues spécifiées.

69 PLOTR <offset X>, <offset Y>[, <encre masquée>]

Affiche un point aux coordonnées relatives spécifiées.

**Numéro
de page**

- 51 **POKE<adresse>, <valeur>**
Place la valeur sur 8 bits fournie à l'adresse spécifiée.
- 10 **POS(# <numéro de canal>)**
Donne la position horizontale du curseur texte pour le canal demandé.
- 12 **PRINT[# <N° de canal>,][<liste à imprimer>]
[USING <option>][<séparateur>]**
Définit :
— la ou les valeurs à afficher,
— le numéro de canal en sortie,
— éventuellement les options d'affichage (USING).
- 77 **RAD**
Définit le mode de calcul des fonctions trigonométriques en RADians.
- 77 **RANDOMIZE[<expression numérique>]**
Définit le point de départ d'une série pseudo-aléatoire.
- 40 **READ<liste de variables>**
Lit des données constantes définies dans un ordre DATA.
- 58 **RELEASE<canal sonore>**
Libère un son dans un canal sonore.
- 84 **REM<commentaire>**
Définit un commentaire sur une ligne BASIC.
- 55 **REMAIN(<nombre entier>)**
Supprime le chronomètre indiqué.
- 85 **RENUM[<Nouveau numéro de ligne>][<ancien numéro de ligne>]
[<différence entre deux lignes>]**
Renumérote un programme BASIC présent en mémoire.
- 40 **RESTORE[<numéro de ligne>]**
Positionne le pointeur de lecture de données sur la ligne spécifiée.
- 47 **RESUME[<N° de ligne>]**
Redonne la main au programme dérouté lorsqu'une erreur s'est produite.

**Numéro
de page**

- 43 **RETURN**
Marque la fin d'un sous-programme.
- 36 **RIGHT\$(<chaîne alphanumérique> , <nombre entier>)**
Donne les <nombre entier> caractères les plus à droite de <chaîne alphanumérique>.
- 79 **RND(<expression numérique>)**
Donne un nombre lu dans la séquence pseudo-aléatoire courante.
- 79 **ROUND(<expression numérique> [, <nombre entier>])**
Arrondit l'expression numérique fournie avec un nombre de décimales défini par <nombre entier>.
- 87 **RUN <chaîne alphanumérique>**
Charge un programme (présent sur support magnétique) en mémoire et l'exécute.
- 87 **RUN[<numéro de ligne>]**
Exécute un programme présent en mémoire à partir de la ligne spécifiée ou à partir de la première ligne si aucun numéro de ligne n'est spécifié.
- 27 **SAVE <nom de fichier> [, <type de fichier>] [, <délimiteurs>]**
Sauve un programme BASIC ou une portion de mémoire définie par ses délimiteurs.
- 79 **SGN(<expression numérique>)**
Détermine le SGN d'une expression numérique.
- 79 **SIN(<expression numérique>)**
Calcule le SINus de l'expression numérique fournie.
- 58 **SOUND <canal> , <période> [, <durée> [, <volume> [, <ENV> [, <ENT> [, <période de bruit>]]]]**
Met un son en file d'attente sonore.
- 37 **SPACE\$(<nombre entier>)**
Crée une chaîne de <nombre entier> blancs.
- 12 **SPEED INK <nombre entier> , <nombre entier>**
Définit la vitesse de clignotement des encres.
- 25 **SPEED KEY <attente avant répétition> , <délai des répétitions>**
Définit les caractéristiques de l'auto-repeat.

**Numéro
de page**

- 30 SPEED WRITE <nombre entier>**
Définit la vitesse d'écriture sur unité de cassette.
- 61 SQ(<canal sonore>)**
Donne le nombre de places libres dans une file d'attente sonore.
- 80 SQR(<expression numérique>)**
Calcule la racine carrée d'un nombre.
- 87 STOP**
Arrête l'exécution d'un programme.
- 37 STR\$(<expression numérique>)**
Convertit l'expression numérique (exprimée en base 2, 10 ou 16) en base 10.
- 37 STRING\$(<nombre entier>, <caractère>)**
Crée une chaîne de <nombre entier> <caractère>s du même type.
- 23 SYMBOL <N° de caractère>, <liste de lignes élémentaires>**
Redéfinit un caractère.
- 24 SYMBOL AFTER <nombre entier>**
Fixe le nombre de caractères redéfinissables.
- 13 TAG[# <N° de canal>]**
Affiche du texte à une position graphique.
- 14 TAGOFF[# <N° de canal>]**
Annule la commande TAG.
- 80 TAN(<expression numérique>)**
Calcule la TANGente de l'expression numérique fournie.
- 69 TEST(<coordonnée X>, <coordonnée Y>)**
Donne la valeur de l'encre du point défini par les coordonnées absolues X, Y.
- 69 TESTR(<offset X>, <offset Y>)**
Donne la valeur de l'encre du point défini par les coordonnées relatives X, Y.

**Numéro
de page****55 TIME**

Donne le temps écoulé en 1/300 seconde depuis la mise en marche ou le boot de l'ordinateur.

87 TRON**88 TROFF**

Valide ou dévalide le mode trace.

81 UNT(<adresse>)

Convertit un entier 16 bits sans signe en un entier 15 bits signé.

38 UPPER\$(<chaîne alphanumérique>)

Tous les caractères minuscules de la chaîne alphanumérique sont changés en caractères majuscules identiques.

38 VAL(<chaîne alphanumérique>)

Extrait une expression numérique si elle est présente au début de la <chaîne alphanumérique> fournie.

14 VPOS(# <N° de canal>)

Donne la position verticale du curseur texte pour le canal demandé.

89 WAIT<adresse>, <masque>[, <inversion>]

Attend que ((adresse) OR <masque>) XOR <inversion> soit non nul.

89 WEND

Marque la fin d'un bloc d'instructions ayant commencé par WHILE.

89 WHILE<expression logique>

Exécute un bloc d'instructions tant que <expression logique> est vraie.

90 WIDTH<nombre entier>

Définit la largeur d'impression.

90 WINDOW[# <N° de canal>,]<gauche>, <droite>, <haut>, <bas>

Définit les dimensions d'une fenêtre.

90 WINDOW SWAP<N° de canal>, <N° de canal>

Définit le numéro de fenêtre courante.

**Numéro
de page**

14 **WRITE[# <N° de canal> ,][<liste de données>]**

Affiche ou écrit les données sur le canal spécifié.

81 **<argument> XOR <argument>**

OU EXCLUSIF entre les deux arguments.

70 **XPOS**

Donne la position horizontale du curseur graphique.

70 **YPOS**

Donne la position verticale du curseur graphique.

15 **ZONE<nombre entier>**

Définit la longueur de la zone d'affichage ou d'impression utilisée avec l'ordre PRINT.

4/1.5

Cours de programmation

Cette partie s'adresse à tous les programmeurs, débutants ou confirmés, qui désirent être plus performants dans la conception puis dans la réalisation de logiciels de tous types en employant une méthode simple et efficace...

Avant de parler de méthodologie, nous pouvons remarquer que l'évolution des langages de programmation a suivi de près l'évolution des ordinateurs. Il y a une vingtaine d'années seulement, les informaticiens ne disposaient que du « langage machine » pour réaliser des programmes de longueur aussi importante que le respect qu'on vouait à leur personne, à juste titre d'ailleurs.

Mais ces temps héroïques sont bien lointains, et le programmeur « moderne » dispose d'outils qui, certes, le ramènent au rang du commun des mortels, mais également, lui facilitent grandement la tâche.

Ces outils sont de deux natures différentes :

- les méthodologies qui visent à optimiser les temps de conception/réalisation/maintenance des logiciels. Nous allons en présenter une dans ce paragraphe.
- les langages évolués qui permettent de manipuler des entités complexes à moindres frais. Dans ce qui suit, nous analyserons les possibilités du langage BASIC des CPC.

Comment vous y prenez-vous quand vous décidez d'écrire un programme ?

- Vous branchez l'ordinateur et vous commencez à taper ?
- Vous réfléchissez d'abord au problème que vous voulez résoudre, puis vous passez à la programmation ?

La première méthode peut certainement convenir si le programme à écrire est court et/ou manipule peu de concepts différents. Mais, même dans ce cas, nous vous la déconseillons. A moins que vous n'ayez une grande habitude du langage de programmation utilisé et une connaissance a priori du problème que vous voulez résoudre, vous risquez :

- d'aligner les instructions de manière non optimale et désordonnée,
- de perdre un temps non négligeable dans la phase de mise au point.

La deuxième méthode est de loin plus saine. C'est celle dont nous allons parler dans la suite. Elle consiste à diviser la résolution du problème en quatre phases successives :

- 1) Analyse du problème,
- 2) Recherche d'une structure,
- 3) Programmation,
- 4) Mise au point.

1) Analyse du problème

Il est nécessaire de bien définir le problème que vous vous proposez de résoudre avant d'entamer la phase de programmation. C'est même le point le plus important — et souvent le plus négligé — de l'analyse.

En effet, comment prétendre écrire un programme sain si l'on n'a pas, dès le départ, formalisé toutes les données, finalités et traitements ?

Une bonne méthode consiste à définir le problème comme étant une boîte noire dans laquelle entrent et de laquelle sortent des informations. Cette boîte noire contient le traitement nécessaire à appliquer aux entrées (par l'utilisateur, par un port d'entrée, ...) pour les transformer en sorties désirées (affichages, calculs, impressions, ...)

Entrée → Traitement → Sortie

Posez-vous les questions suivantes :

- Quelles données dois-je entrer pour obtenir les sorties qui m'intéressent ?
- Quels traitements appliquer aux entrées pour les transformer en résultats, en sortie du programme ?

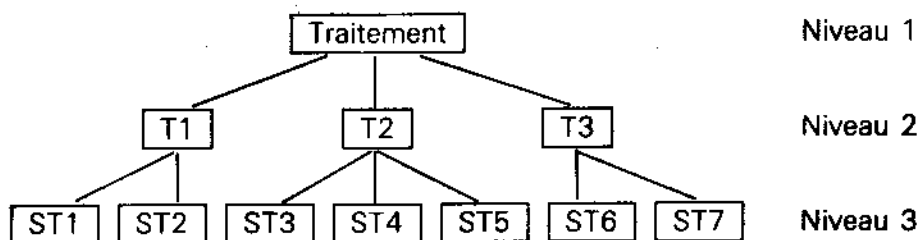
2) Recherche d'une structure

La première phase définie, vous devez en être arrivé au point où vous connaissez la liste

- des entrées,
- des traitements,
- des sorties.

Dans la mesure du possible, essayez de décomposer les traitements en sous-traitements plus élémentaires et indépendants. De même, décomposez les sous-traitements en tâches indépendantes et encore plus élémentaires, et ainsi de suite pour arriver à créer de petits modules qui comportent seulement quelques dizaines d'instructions. Cette manière de décomposer un problème en sous-problèmes porte le nom d'*analyse descendante*. Cette méthode a pour avantage d'obliger le concepteur/programmeur à se poser les bonnes questions dès le départ de l'analyse : pour arriver à décomposer le problème en tâches de quelques dizaines d'instructions, il est effectivement nécessaire de l'avoir clairement défini.

Le traitement se décompose donc en une série de modules de niveaux hiérarchiques de plus en plus faibles et réalisant de moins en moins d'opérations.



Remarques :

a) Essayez de limiter la décomposition au niveau 3.

Cette remarque se justifie pour de petits programmes qui logent dans les 64 ou 128 kilo-octets de votre ordinateur. Descendez éventuellement au niveau 4 si le programme est très complexe, mais cela risque d'être inutile et risque de nuire à la bonne compréhension du logiciel.

b) Essayez de ne pas avoir plus de 5 sous-modules par module.

En effet, il est difficile de mémoriser plus de 5 concepts en même temps. Quand vous êtes à un niveau hiérarchique donné, pour un module donné, vous devez être capable d'avoir en tête les n sous-modules qui vont le composer.

Pour être sûr de ne rien oublier de la décomposition, limitez-vous à 3 ou 4 sous-modules par module et, au besoin, descendez d'un niveau supplémentaire si un ou plusieurs des sous-modules vous semble(nt) trop complexe(s).

3) Programmation

Cette phase doit être quasi immédiate si vous avez respecté les consignes données dans les deux phases précédentes. A titre indicatif, elle ne devrait pas dépasser 30 % du temps total que vous passerez à la définition, la programmation et le test du programme.

Les modules et sous-modules seront organisés en procédures et/ou fonctions, et le programme principal fera appel aux procédures d'entrée, traitement et sortie.

A titre indicatif, prenons l'exemple suivant : *Tracé d'une courbe d'équation $Y=f(x)$.*

L'analyse du problème nous montre que les entrées sont :

- domaine d'étude,
- équation de la courbe.

La sortie est :

- affichage de la courbe.

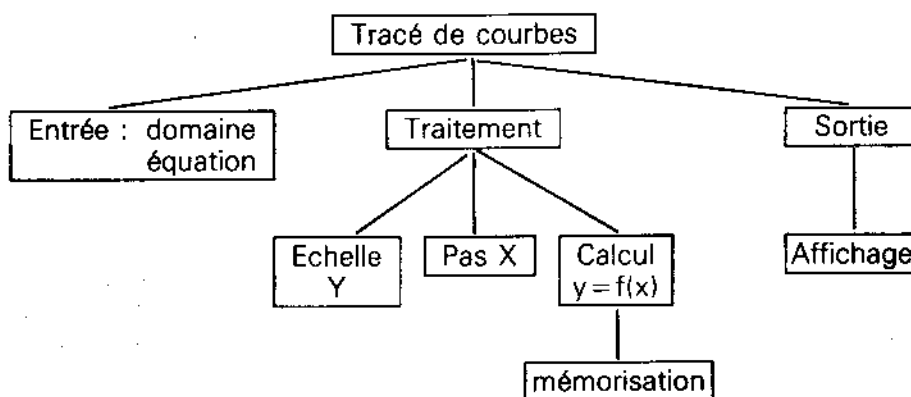
Le traitement est :

- calcul de Y en fonction de x dans le domaine d'étude,
- mémorisation du calcul.

Si nous poursuivons l'analyse, le traitement peut se décomposer en

- recherche de l'échelle en Y,
- recherche du pas du tracé,
- calcul et mémorisation des coordonnées y en fonction de x sur le domaine donné.

Ce qui se représente par le graphe structuré suivant :



Le programme qui en résulte a la structure suivante :

```

10 REM Trace de courbes
20 '
30 GOSUB 1000 'Entree
40 GOSUB 2000 'Traitement
50 GOSUB 3000 'Sortie
60 '
70 END
80 ' -----
1000 REM Entree
1010 '
1020 REM saisie de l'equation et du domaine d'etude
1030 '
1040 RETURN
1050 ' -----
2000 REM Traitement
  
```

```
2010 '  
2020 GOSUB 4000 'Calcul echelle en Y  
2030 GOSUB 5000 'Calcul pas en x  
2040 FOR I= 1 TO NBPTS STEP PAS  
2050     GOSUB 6000 'Memorisation des coordonnees  
2060 NEXT I  
2070 '  
2080 RETURN  
2090 '-----  
3000 REM Sortie  
3010 '  
3020 REM Affichage de la courbe en lisant le tableau  
3030 '  
3040 RETURN  
3050 '-----  
4000 REM Echelle en Y  
4010 '  
4020 REM Recherche de l'echelle  
4030 '  
4040 RETURN  
4050 '-----  
5000 REM Pas en x  
5010 '  
5020 REM Calcul du pas  
5030 '  
5040 RETURN  
5050 '-----  
6000 REM memorisation d'un point  
6010 '  
6020 REM memorisation de  $Y = f(x)$  dans le tableau Y  
6030 '  
6040 RETURN
```

4) Mise au point

Cette partie doit être quasi inexistante si vous avez bien suivi les conseils précédents. Si vous n'obtenez pas les résultats escomptés, déterminez dans quelle branche se trouve le problème et descendez de niveau hiérarchique jusqu'à trouver le module ou sous-module responsable du mauvais fonctionnement.

des coordonnées

3080 NEXT I

3070

3060

3050

3040 M SORTIE

3030

3020 REM Affichage de la coupe en fonction du tableau

3030

3040

3050

4000 Y

4010

4020 REM Recherche de l'échelle

4030

4040

REM Pas en

5000 M Calcul du pas

5030

5040 RETURN

5050

6000 M mémorisation d'un point

6010

6020 M mémorisation de (x) de Y

6030

6040

4/1.6

Basic approfondi

4/1.6.1

SYMBOL et SYMBOL AFTER

Votre Logo personnalisé — Aller plus loin avec la redéfinition des caractères

Rappels et compléments

Nous allons dans ce chapitre, créer un petit programme affichant un logo au nom des Editions Weka, et que vous pourrez modifier afin de le personnaliser.

Ce logo permettra d'étudier la fonction SYMBOL ainsi que SYMBOL AFTER, et d'en cerner un peu les restrictions d'utilisation.

Avant toute définition de caractères, il est souhaitable de le signaler au Basic par l'instruction SYMBOL AFTER n. Cette fonction indique au Basic que des caractères dont le numéro commence en n + 1 vont être redéfinis. Après cette déclaration, vous pourrez redéfinir vos caractères par la commande SYMBOL N° de caractère (le petit programme de la Partie 5, chapitre 9 page 2 vous y aidera).

Chaque caractère ainsi défini est accessible par la commande :

PRINT CHR\$(N° de caractère)

Vous pourrez même les associer dans une chaîne de caractères, par exemple :

A\$ = CHR\$(N°1) + CHR\$(N°2) + ... etc ...

Il est même possible d'y associer des caractères de contrôle permettant un affichage sur plusieurs lignes (les caractères **CHR\$(8)** = retour d'un caractère ; **CHR\$(9)** = avance d'un caractère ; **CHR\$(11)** = remonte d'une ligne sur la même colonne).

Essayez par exemple, en utilisant les caractères prédéfinis dans le moniteur (c'est-à-dire en ne vous préoccupant pas encore de redéfinir un caractère — voir Annexe 3 pages 1 à 12) :

```
A$ = CHR$(32) + CHR$(240) + CHR$(10) + CHR$(8) + CHR$(8)
      + CHR$(242) + CHR$(32) + CHR$(243) + CHR$(10) + CHR$(8)
      + CHR$(8) + CHR$(241)
```

```
LOCATE 10,10
```

```
PRINT A$
```

Il est même possible de redéfinir tous les caractères de l'alphabet, comme dans certains logiciels du commerce, en gothique par exemple. Le numéro de réservation sera alors :

```
SYMBOL AFTER 65
```

Programme exemple

Revenons au programme de présentation de notre logo :

```
10 ***** PRESENTATION VEK
A *****
20 MODE 1
30 INK 0,26:INK 1,1:INK 2,24
40 PAPER 0:PEN 1
50 CLS
60 SYMBOL AFTER 230
70 SYMBOL 231,0,0,0,0,0,0,0,63
80 SYMBOL 232,0,0,0,0,0,0,0,255
90 SYMBOL 233,0,0,0,0,0,0,0,224
100 SYMBOL 234,32,32,32,32,32,32,32
110 SYMBOL 235,224,0,0,0,0,0,0
120 SYMBOL 236,255,0,0,0,0,0,0
130 SYMBOL 237,63,0,0,0,0,0,0
140 SYMBOL 241,0,0,0,0,0,0,255,146
150 SYMBOL 242,0,0,0,0,0,0,255,64
160 SYMBOL 243,0,0,0,0,0,0,255,32
170 SYMBOL 244,0,0,0,0,0,0,255,48
180 SYMBOL 245,0,0,0,0,0,0,248,8
190 SYMBOL 246,146,146,128,128,128,128,1
28,255
200 SYMBOL 247,64,64,64,64,64,64,255

210 SYMBOL 248,32,224,33,33,224,32,32,25
5
220 SYMBOL 249,80,144,16,16,147,82,50,25
5
230 SYMBOL 250,8,8,8,8,200,72,72,248
240 BORDER 26:CLS
```

```

250 PRINT:PRINT:PRINT:PRINT:PRINT:PRINT:
PRINT:PRINT:PRINT:PRINT:PRINT:PRINT
260 PRINT "      "+CHR$(231)+CHR$(232)+CHR$
(232)+CHR$(232)+CHR$(232)+CHR$(232)+CHR$
(233)
270 PRINT "      "+CHR$(234)+"      "+CHR$(2
34)
280 PRINT "      "+CHR$(234)+"      "+CHR$(2
34)
290 PRINT "      "+CHR$(234)+"      "+CHR$(2
34)
300 PRINT "      "+CHR$(234)+CHR$(241)+CHR
$(242)+CHR$(243)+CHR$(244)+CHR$(245)+CHR
$(234)
310 PRINT "      "+CHR$(234)+CHR$(246)+CHR
$(247)+CHR$(248)+CHR$(249)+CHR$(250)+CHR
$(234)
320 PRINT "      "+CHR$(234)+"      "+CHR$(
234)
330 PRINT "      "+CHR$(237)+CHR$(236)+CHR
$(236)+CHR$(236)+CHR$(236)+CHR$(236)+CHR
$(235)
340 REM ORIGIN 150,150:FILL 2
350 LOCATE 12,13:PRINT"Editions WEKA"
360 LOCATE 12,14:PRINT"82, rue Curial"
370 LOCATE 12,15:PRINT"75019 Paris"
380 DATA 53,79,6E,74,61,78,20,65
390 DATA 72,72,6F,72,20,69,6E,20
400 DATA 32,30,0A,0D,32,30,20,18
410 DATA 67,18,6F,64,75,62,20,31
420 DATA 30,30,30
430 RESTORE 380
440 FOR i=1 TO 35
450 READ a$
460 a$="&"+a$
470 c$=c$+CHR$(VAL(a$))
480 NEXT i
490 LOCATE 1,20
500 PRINT c$
510 CALL &BB06
520 PRINT CHR$(72)+CHR$(73)+CHR$(72)+CHR
$(73)
530 RUN "PROGRAM.BAS"

```

- Ligne 60 : réservation de redéfinition pour les caractères à partir de 231.
- Lignes 70 à 180 : redéfinition de caractères permettant de tracer un cadre.

- Lignes 190 à 230 : redéfinition de caractères d'écriture du mot WEKA.
- Lignes 250 à 330 : affichage du logo grâce à tous ces caractères.
- Ligne 340 : en REM, mais peut être placée en ligne normale pour les CPC possédant l'instruction FILL.
- Lignes 380 à 480 : exemple de création d'une variable texte possédant des caractères de contrôle qui vous réservent un effet surprenant, que nous vous laissons découvrir.
- Ligne 530 : commande lançant le programme que vous désirez, la page restant à l'écran durant le temps de chargement de votre programme (intéressant si celui-ci est long). Rien ne vous empêche, de plus, d'insérer votre programme à la suite de cette ligne, il suffira de la remplacer par une boucle d'attente, du type

```
FOR i = 1 to 3000 : next i
```

Nous espérons que votre curiosité et votre perspicacité vous inciteront à étudier ce programme et à le modifier pour créer votre propre logo.

Les restrictions d'utilisation

RÉCUPÉRATION DE MÉMOIRE

Sans que vous le sachiez, le CPC effectue, lors de la mise sous tension, ou d'une réinitialisation, la réservation de la place mémoire pour 16 caractères redéfinissables.

Ce qui veut dire que vous pouvez vous passer de la commande SYMBOL AFTER si vous ne désirez utiliser que les caractères de 240 à 255.

Par contre, si vous ne redéfinissez aucun caractère, il est possible de récupérer cette place mémoire (un caractère occupant 8 octets, on peut récupérer : $8 \times 16 = 128$ octets), grâce à la commande SYMBOL AFTER 256.

SYMBOL AFTER ET MEMORY

La commande SYMBOL AFTER est réputée pour faire très mauvais ménage avec l'instruction MEMORY.

Essayez donc :

```
MEMORY &9FFF  
SYMBOL AFTER 200
```

et vous voilà face à un message d'erreur vous signalant :

```
IMPROPER ARGUMENT
```

alors que tout semble correct.

Cela est très fâcheux lorsque vous désirez implanter des routines en langage machine et que vous désirez les protéger.

Un premier remède qui semble fonctionner est d'inverser ces deux instructions. Mais ici aussi il y a un problème : si vous avez placé ces deux instructions dans un programme, dans cet ordre :

```
10 SYMBOL AFTER 200  
20 MEMORY &9FFF
```

et que vous en êtes encore aux essais de votre programme, tout se passe correctement au premier « RUN », mais de nouveau le message d'erreur précédent s'affiche au deuxième « RUN ».

En parcourant les différents vecteurs du système d'exploitation (voir Partie 4, chapitre 2.7), on trouve l'appel à une routine appelée TXT-INITIALISE à l'adresse &BB4E.

Cette routine effectue une réinitialisation complète de l'écran, des variables texte, des indirections de l'écran, de la table de contrôle, mais, ce qui nous intéresse avant tout : la table des caractères redéfinis est réinitialisée. Après tant de réinitialisation, on comprendra que l'appel de TXT-INITIALISE doit être la première instruction du programme.

Suite à cet appel, on pourra redéfinir les caractères utilisateurs et la position de la valeur haute de la mémoire, mais dans quel ordre ?

L'expérience indique que placer l'instruction SYMBOL AFTER avant la réservation mémoire provoque, lors de l'exécution de cette dernière un assez désagréable message d'erreur MEMORY FULL.

On retiendra donc la séquence d'instructions suivante qui résout les problèmes de redéfinition de caractères :

```
10 CALL &BB4E  
20 MEMORY &3FFF : REM (ou ce dont vous aurez besoin)  
30 SYMBOL AFTER 200 : REM (idem)
```

4/1.6.2

L'instruction CALL et les RSX en Basic

Introduction

Nous allons, dans ce chapitre, nous consacrer à la réalisation d'instructions supplémentaires au Basic des AMSTRAD-CPC.

Nous procéderons par étapes successives, qui vous permettront d'accéder à la personnalisation du jeu d'instructions, en y introduisant vos propres commandes, au nom que vous aurez choisi, et qui appellent bien sûr, des routines en langage machine permettant de les traiter.

Quelques exemples vous permettront de créer vos premières instructions simples sans grandes connaissances en langage machine.

Vous devrez, ensuite, connaître un minimum de programmation en assembleur, afin de créer des routines plus élaborées ; mais une lecture assidue de votre ouvrage, vous ouvrira les portes du microprocesseur Z80 (on se reportera avec intérêt à la Partie 4, chapitre 2).

Certes, la puissance de l'interpréteur Basic de l'Amstrad vous permet de créer des fonctions élaborées (voir l'instruction DEF FN, si souvent ignorée des programmeurs en Basic), mais rien n'égale la rapidité et le gain de place en mémoire RAM fournis par le code machine.

Entrons donc sans tarder, mais progressivement, dans la personnalisation de vos instructions.

I - L'instruction CALL

PREMIERS PAS AVEC L'INSTRUCTION CALL

Vous aurez déjà remarqué dans certains programmes de votre ouvrage, l'appel à des routines en langage machine grâce à l'instruction Basic:

CALL adresse

Cette instruction lance l'exécution d'une routine en langage machine située à partir de l'adresse spécifiée, et, ce, en général, dans la mémoire vive de votre CPC.

Cette adresse peut être décrite en décimal, signé ou non, (par exemple CALL 47878 ou CALL - 17658), éventuellement en binaire (CALL &X1011101100000110, mais trop complexe pour être utilisée), ou, le plus souvent, en hexadécimal (par exemple CALL &BB06).

Il est même possible d'appeler une routine dont l'adresse est contenue dans une variable numérique (CALL A ; A contenant l'adresse de la routine à appeler).

Dans les exemples indiqués ci-dessus, les adresses données sont rigoureusement identiques, et provoquent donc l'appel à une même routine, qui est un vecteur appelant une routine en ROM attendant l'appui sur une touche du clavier (voir Partie 4, chapitre 2.7 page 3).

A titre d'exemple, nous allons créer notre propre programme qui affichera le nom de WEKA sur l'écran, par un CALL adresse tout simplement.

Les explications seront données de façon à ce que vous puissiez modifier le programme, et inscrire votre propre nom à la place de celui de WEKA.

Il faut tout d'abord savoir que pour afficher des caractères sur l'écran, l'AMSTRAD ne comprend que les codes ASCII de ces caractères (codes hexadécimaux).

Notre message, qui sera stocké en mémoire vive, sera donc créé ainsi :

```
PRINT  CHR$(57);  CHR$(45);CHR$(4B);  CHR$(41)
           W           E           K           A
```

Nous indiquerons la fin du message par le nombre hexadécimal &FF qui ne correspond à aucun caractère alphanumérique.

AFFICHAGE DU MESSAGE SUR L'ECRAN

Attaquer l'écran pixel par pixel, en allumant chacun des pixels des caractères dans la mémoire écran, demanderait une programmation fastidieuse (mais pas impossible sur le CPC).

En examinant de plus près la liste des vecteurs du système d'exploitation inscrits en RAM (voir Partie 4, chapitre 2.7.), nous y découvrons le vecteur TXT-OUTPUT (Partie 4, chapitre 2.7 p. 12) situé à l'adresse &BB5A, qui affiche la caractère dont le code ASCII est chargé dans le registre A.

Remarque :

La routine TXT-WR CHAR située en &BB5D aurait aussi pu nous convenir, car nous n'utiliserons pas de caractères de contrôle.

Il nous suffira donc de lire chacun des caractères du message dans le registre A, puis d'appeler la routine TXT-OUTPUT, qui les affichera.

L'algorithme du programme s'écrira ainsi :

— DEBUT

Pointer la première lettre du message à afficher

TANT QUE le caractère lu n'est pas le caractère de fin de message (&FF)

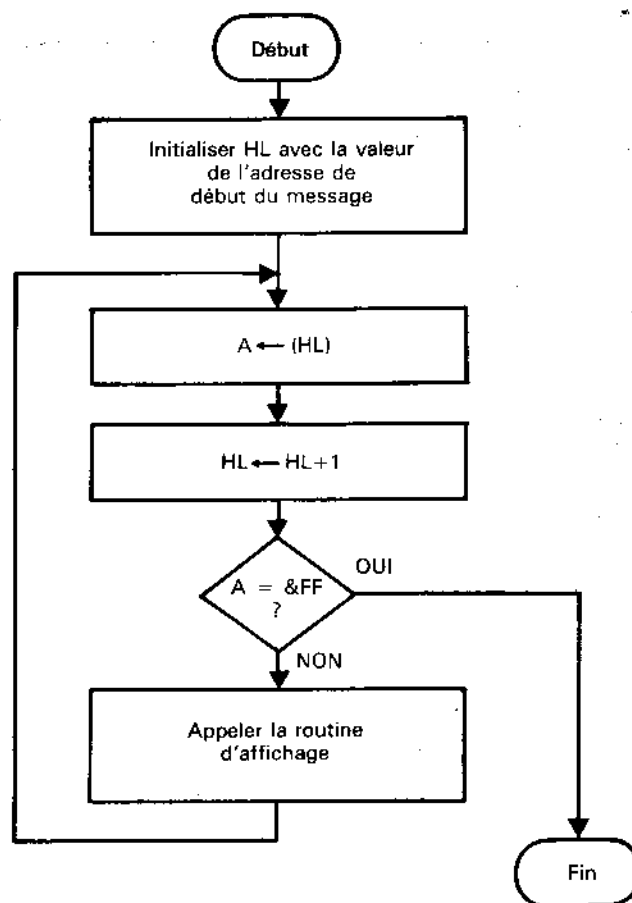
- Lire le caractère pointé
- Pointer le caractère suivant
- PROCEDURE afficher le caractère (TXT-OUTPUT)

FIN TANT QUE

— FIN

L'assembleur ne possédant pas l'instruction évoluée permettant de créer une boucle du type TANT QUE ... FIN TANT QUE, nous devons la simuler par un test qui nous fera sortir de la boucle de lecture et d'affichage de caractère dès que le nombre &FF est rencontré.

L'ordinogramme se traduira ainsi :



Vous trouverez ci-après le listing d'assemblage largement commenté afin que vous puissiez l'analyser en fonction de l'algorithme et de l'ordino-gramme présentés plus haut.

```

1      ;*****
2      ; * AFFICHAGE D'UN MESSAGE PAR      *
3      ; * UN APPEL DE SOUS PROGRAMME     *
4      ; * A L'AIDE DE L'INSTRUCTION      *
5      ; *   CALL adresse                  *
6      ;*****
7      ;
8      ;
9      ;**   ORIGINE D'ASSEMBLAGE   **
10     ;
11     ;       ORG   &A000H
12     ;
13     ;
14     ;**   ADRESSE DE CHARGEMENT DU   **
15     ;**   CODE MACHINE               **
16     ;
17     ;       LOAD  &A000H
18     ;
19     ;
20     ;*****
21     ; *   TRAITEMENT DE L'INSTRUCTION *
22     ;*****
23     ;
24 A000 210DA0  NOM:      LD   HL,MESSAG      ; POINTE
25             ; L'ADRESSE DE DEBUT DU
26             ; MESSAGE A AFFICHER
27 A003 7E      NOM1:     LD   A,(HL)        ; CHARGE
28             ; CODE CARACTERE DANS A
29 A004 23             INC  HL                ; POINTE CARACTERE SUIVANT
30 A005 FEFF      CP   OFFH                ; DERNIER CARACTERE ?
31 A007 CB      RET  Z                    ; OUI ALORS FIN
32 A008 CD5AB8    CALL  OBBSAH              ; CARACTERE A
33             ; L'AFFICHAGE
34 A00B 18F6      JR    NOM1                ; CARACTERE SUIVANT
35             ;
36             ;*****
37             ;
38             ;
39             ;*****
40             ; *   MESSAGE A AFFICHER   *
41             ;*****
42             ;
43 A00D 57454B41  MESSAG:  DEFB  "WEKA"
44 A011 FF      DEFB  OFFH
45             ;
46             ;
47             END

```

— Ligne 11 : ORG indique à l'assembleur de débiter l'assemblage à l'adresse &A000.

— Ligne 17 : indique l'adresse de début de chargement du code machine.

— Lignes 24 à 34 : programme principal.

— Ligne 30 : le test sur le caractère de fin &FF qui permet le retour en cas d'égalité (ligne 31) — retour au Basic, si la routine a été appelée depuis le Basic.

— Ligne 43 : message que vous pouvez modifier comme bon vous semble. Il faudra obligatoirement ajouter l'octet &FF après le message pour signaler la fin (ligne 44).

Vous trouverez ci-après le listing du chargeur Basic des codes machine.

```
10 REM *****
20 REM * AFFICHAGE D'UN MESSAGE *
30 REM * PAR APPEL D'UN SOUS      *
40 REM * PROGRAMME MACHINE        *
50 REM *****
60 REM
70 REM *** CHARGEUR BASIC ***
80 ADR = &A000:I = 0
90 RESTORE 300
100 READ A$
110 IF A$ = "XX" THEN 190
120 B$ = "&" + A$
130 B = VAL(B$)
140 POKE ADR,B
150 ADR = ADR + 1
160 I = I + 1
170 GOTO 100
180 REM
190 REM *** SAUVEGARDE ***
200 PRINT "POUR SAUVEGARDER LA ROUTINE E
N BINAIRE"
210 PRINT "SAVE ";CHR$(34);"ROUTINE.BIN"
;CHR$(34);",B,&A000,";RIGHT$(STR$(I),LEN
(STR$(I))-1)
220 PRINT
230 PRINT "CHARGEMENT PAR LOAD ";CHR$(34
);"ROUTINE.BIN";CHR$(34);",&A000"
240 PRINT "MEMORY &9FFF"
250 PRINT
260 PRINT "EXECUTION PAR CALL &A000"
270 STOP
280 REM
290 REM *** ROUTINE L.M. ***
300 DATA 21,0D,A0,7E,23,FE,FF,CB
310 DATA CD,5A,BB,1B,F6
320 REM
330 REM *** MESSAGE ***
340 DATA 57,45,4B,41
350 REM
360 REM *** FIN DE MESSAGE ***
370 DATA FF
380 REM
390 REM *** FIN DE DATAs ***
400 DATA XX
```

Vous pouvez modifier la ligne de données du message (ligne 340) afin d'inscrire votre message. Le tableau des codes ASCII vous aidera à inscrire votre propre message, en y prenant les codes hexadécimaux des caractères désirés. Vous pouvez insérer autant de lignes de DATAs que vous voulez (donc un message aussi long que vous voulez).

Seules les lignes 370 et 400 doivent obligatoirement être les dernières lignes du programme.

Cette première approche de l'instruction CALL est relativement simple, puisqu'elle ne demande qu'un appel direct et effectue des traitements simples.

POUR ALLER PLUS LOIN AVEC CALL

Les routines machine peuvent, et sont plus souvent utilisées pour traiter des chiffres souvent différents, aussi faut-il pouvoir les transmettre au programme, et pourquoi pas, récupérer le résultat.

La première idée qui vient à l'esprit est d'utiliser les instructions qui vont de pair avec CALL, PEEK et POKE.

POKE adresse, valeur permet de sauvegarder une valeur à une adresse précise en mémoire vive, que le programme en langage machine récupère à cette adresse.

Après traitement, il est possible de récupérer un résultat mémorisé à une adresse (par la routine) grâce à l'instruction PEEK adresse.

Cette méthode de programmation peut être intéressante lors du développement et de la mise au point des routines, mais devient vite lourde à gérer quand il faut transmettre et récupérer plusieurs paramètres à la routine.

Il est possible d'éviter cette façon de procéder en fournissant directement les valeurs avec l'instruction CALL. On appelle cela le passage de paramètre.

LE PASSAGE DE VALEURS FIXES ENTIÈRES

Nous allons, dans un premier temps étudier la manière de fournir des valeurs entières.

Il est en effet possible d'écrire l'instruction CALL sous la syntaxe suivante :

CALL adresse,valeur1,valeur2, ..., valeurn

Trente deux valeurs peuvent être fournies de cette façon à la routine en assembleur.

Récupération des paramètres dans la routine

Il faut d'abord savoir que lors de l'exécution de ce type d'instruction, l'interpréteur Basic considère que les valeurs fournies sont codées sur 16 bits (donc des chiffres compris entre 0 et 65535 ou -32768 et +32767).

Il empile ensuite ces valeurs en mémoire RAM (chacun occupera donc deux cases mémoire). L'endroit où sont empilées ces valeurs peut être connu grâce au registre d'index IX du Z80, on appelle cette adresse : adresse de base des paramètres. Le nombre de paramètres peut lui aussi être connu : il est inscrit dans le registre accumulateur A.

LE PASSAGE D'UNE SEULE VALEUR NUMÉRIQUE

Nous considérerons dans un premier temps une instruction ne comportant qu'un paramètre de la forme :

CALL adresse,valeur

Après exécution, dans le programme en langage machine, nous pourrions vérifier qu'un seul paramètre a été envoyé en testant le registre A.

Pour obtenir la valeur transmise, il suffira de lire le contenu de IX + 00H pour avoir l'octet bas, la contenu de IX + 01H nous donnera l'octet haut.

Exemple :

- CALL adresse,3425 placera &A1 à l'adresse IX+00h et &0D à l'adresse IX+01H (car $3425_{\text{decimal}} = 0DA1_{\text{hexa}}$).
- CALL adresse,36 placera &24 en IX+00H et &00 en IX+01H.

Nous vous proposons le programme ci-dessous afin d'illustrer les explications ci-dessus. Ce programme exécutera un scrolling vertical d'une ligne de caractères (huit pixels) dans le sens que vous aurez précisé : sens montant si on transmet le paramètre 1, sens descendant si vous placez un paramètre différent (on prendra -1).

Le programme se trouvera à l'adresse &A000, ce qui donnera donc les instructions suivantes :

- CALL &A000,1 pour un scrolling montant,
- CALL &A000,-1 pour un scrolling descendant.

Voici dans un premier temps l'algorithme du programme :

- DEBUT
 - SI le nombre de paramètres est différent de 1
 - ALORS
 - Retourner au Basic
 - FIN SI
 - Récupérer l'octet bas du paramètre
 - SI sa valeur est 1
 - ALORS
 - Indiquer un scrolling haut
 - SINON

- Indiquer un scrolling bas
- FIN SI
- Effectuer le scrolling
- FIN

Pour effectuer ce scrolling, il sera intéressant d'utiliser le vecteur SCR-HW-ROLL situé à l'adresse &BC4D, qui requiert dans le registre B le sens du scrolling, et dans A le numéro d'encre qui remplira la nouvelle ligne ainsi créée (on se reportera Partie 4, Chap. 2.7, page 36).

Le listing assemblé donne donc :

```

1      ;*****
2      ; * SCROLLING VERTICAL MONTANT *
3      ; * PAR CALL &A000,1 *
4      ; * SCROLLING VERTICAL DESCENDANT *
5      ; * PAR CALL &A000,-1 *
6      ;*****
7      ;
8      ;
9      ;** ORIGINE D'ASSEMBLAGE **
10     ;
11     ORG 0A000H
12     ;
13     ;
14     ;** CHARGEMENT DU CODE MACHINE **
15     ;
16     LOAD 0A000H
17     ;
18     ;
19     ;** PROGRAMME SCROLLING **
20     ;
21 A000 FE01      CP 01H      ;UN CARACTERE ?
22 A002 C0        RET NZ     ;NON ALORS FIN
23 A003 DD7E00    LD A,(IX+00H) ;RECUPERE CHIFFRE
24 A006 FE01      CP 01H     ;SCROLLING MONTANT?
25 A008 2804      JR Z,HAUT  ;OUI ALORS EXECUTER
26     ;
27 A00A 0600      BAS:      LD B,00H      ;NON ALORS PREPARER
28 A00C 1802      JR SCROLL ;SCROLLING BAS
29     ;
30 A00E 0601      HAUT:     LD B,01H     ;SCROLLING HAUT
31     ;
32 A010 3E00      SCROLL:   LD A,00      ;COULEUR DE FOND
33     ; EGAL ENCRE 0
34 A012 CD4DBC    CALL 0BC4DH ;EXECUTER SCROLLING
35     ; DE HUIT PIXEL SELON SENS
36 A015 C9        RET      ;FIN RETOUR
37     ;
38     ;
39     ;*****
40     ;
41     END

```

Le chargeur Basic est donné ci-dessous :

```

10 REM *****
20 REM * SCROLLING VERTICAL MONTANT *
30 REM * PAR CALL &A000,1 *
40 REM * SCROLLING VERTICAL DESCEND *
50 REM * PAR CALL &A000,-1 *
60 REM *****
70 REM
80 REM *** CHARGEUR BASIC ***
90 ADR = &A000:I = 0
100 RESTORE 310
110 READ A$
120 IF A$ = "XX" THEN 200
130 B$ = "&" + A$
140 B = VAL(B$)
150 POKE ADR,B
160 ADR = ADR + 1
170 I = I + 1
180 GOTO 110
190 REM
200 REM *** SAUVEGARDE ***
210 PRINT "POUR SAUVEGARDER LA ROUTINE E
N BINAIRE"
220 PRINT "SAVE ";CHR$(34);"ROUTINE.BIN"
;CHR$(34);",B,&A000,";RIGHT$(STR$(I),LEN
(STR$(I))-1)
230 PRINT
240 PRINT "CHARGEMENT PAR LOAD ";CHR$(34)
);"ROUTINE.BIN";CHR$(34);",&A000"
250 PRINT "MEMORY &9FFF"
260 PRINT
270 PRINT "EXECUTION PAR CALL &A000,VALE
UR"
280 STOP
290 REM
300 REM *** ROUTINE L.M. ***
310 DATA FE,01,CO,DD,7E,00,FE,01
320 DATA 28,04,06,00,18,02,06,01
330 DATA 3E,00,CD,4D,BC,C9
340 REM
350 REM *** FIN DE DATA ***
360 DATA XX

```

LE PASSAGE DE PLUSIEURS VALEURS NUMÉRIQUES

Dans le cas où plusieurs valeurs numériques sont fournies à la routine en langage Machine, ces valeurs sont empilées dans l'ordre de frappe de l'instruction.

En exécutant l'instruction générale :

CALL adresse, valeur1, valeur2, ... valeur n-1, valeur n ces valeurs seront empilées ainsi :

$IX + 2 \cdot (n - 1) + 1$	octet haut de valeur 1
$IX + 2 \cdot (n - 1)$	octet bas de valeur 1
$IX + 2 \cdot (n - 1 - 1) + 1$	octet haut de valeur 2
$IX + 2 \cdot (n - 1 - 1)$	octet bas de valeur 2
....	...
$IX + 03H$	octet haut de valeur n-1
$IX + 02H$	octet bas de valeur n-1
$IX + 01H$	octet haut de valeur n
$IX + 00H$	octet bas de valeur n

La dernière valeur de l'instruction étant toujours à la base de IX.

Par exemple: CALL &adresse,1,2 place en :

$IX + 00H$	la valeur 02
$IX + 01H$	la valeur 00
$IX + 02H$	la valeur 01
$IX + 03H$	la valeur 00

LE PASSAGE D'UNE VARIABLE NUMÉRIQUE ENTIÈRE

Il est possible de transmettre à une routine en langage machine des valeurs entières contenues dans des variables numériques entières, c'est-à-dire des variables du type A% ou définies par l'instruction DEFINT variable.

En fait ce ne sont pas les valeurs contenues dans ces variables qui sont transmises, mais les adresses où ces valeurs sont sauvegardées en mémoire vive.

La syntaxe d'appel de la routine devra donc être la suivante :

CALL adresse, @variable%

qui transmet l'adresse du contenu de cette variable. Il faut remarquer que l'utilisation de cette séquence doit absolument être précédée d'une utilisation de la variable sous peine de générer un message d'erreur.

Sitôt dans la routine, il sera possible alors de récupérer l'adresse de l'octet faible de la valeur dans le registre HL, en général, puis d'accéder à la valeur, si elle est sur un octet, lisant le contenu de HL dans le registre accumulateur A.

Nous prendrons comme exemple le programme suivant :

```

1          ;*****
2          ;* AFFICHAGE CARACTERE ASCII *
3          ;*****
4          ;
5          ;** ADRESSE D'ASSEMBLAGE **
6          ORG 0A000H
7          ;
8          ;** ADRESSE DE CHARGEMENT **
9          LOAD 0A000H
10         ;
11         ;** ROUTINE **
12 A000 FE01 CP 01
13 A002 C0 RET NZ ; SINON RETOUR
14 A003 DD6E00 LD L,(IX+0) ; OCTET BAS
15 A006 DD6601 LD H,(IX+1) ; OCTET HAUT
16 A009 7E LD A,(HL) ; VALEUR
17 A00A CD5ABB CALL 0BB5AH ; A L'AFFICHAGE
18 A00D C9 RET ; FIN
19
20         END

```

Dans ce programme, IX+00H contient la partie basse de l'adresse de la variable, il suffit de la lire dans le registre L (LD L,(IX+00H)), puis on lit la partie haute dans le registre H (LD H,(IX+01H)), enfin le contenu de cette adresse est lu dans le registre A (LD A,(HL)) et est envoyé dans la routine moniteur d'impression (&BB5A).

Lors de l'exécution des instructions suivantes :

A% = &41 : CALL &A000,@A%

on obtient l'impression du caractère A (&41 = code ASCII de A) ; en fait, l'exécution de cette routine, pour tout autre valeur de A% provoque l'affichage du caractère correspondant au code ASCII contenu dans la variable [équivalent Basic : PRINT CHR\$(A%)].

```

10 REM *****
20 REM * AFFICHAGE CARACTERE ASCII *
30 REM *****
40 REM
50 FOR I=1 TO 14
60 READ A
70 POKE &9FFF+I,A
80 NEXT I
90 STOP
100 REM
110 DATA &FE,&01,&C0,&DD,&6E,&00,&DD
120 DATA &66,&01,&7E,&CD,&5A,&BB,&C9

```

Il est, comme lors du passage de valeurs numériques entières, possible de passer plusieurs variables numériques entières sous la forme d'une instruction :

CALL &adresse, @variable1, @variable2, ..., @variablen

Ce sont cette fois-ci les adresses de ces variables qui sont empilées l'une après l'autre et que l'on récupérera grâce au registre IX.

LE PASSAGE DE CHAINES DE CARACTERES

Il est aussi possible de passer une chaîne de caractères dans une instruction :

CALL &adresse, « CHAINE DE CARACTERE »

Cette syntaxe est prévue pour fonctionner sans problème sur le CPC-6128, nous verrons ultérieurement l'adaptation nécessaire aux modèles CPC-664 et CPC-464.

Lors de l'arrivée dans la routine machine, l'interpréteur Basic aura empilé l'adresse où se trouvent *uniquement* les caractéristiques de la chaîne de caractères.

A cette adresse se trouvent le nombre de caractères contenus dans la chaîne. A l'adresse immédiatement supérieure se trouve l'octet bas de l'adresse du premier caractère de la chaîne, puis à l'adresse supérieure, l'octet haut.

Pour récupérer la chaîne de caractères, il suffira donc de lire l'adresse contenue en IX + 01H et IX + 00H, qui nous donnera l'adresse des caractéristiques de la chaîne. A cette adresse, on lira le nombre de caractères de la chaîne (pour le traitement, on place généralement ce nombre dans le registre B, car il existe une instruction en assembleur Z80 de test sur B qui permet un comptage : DJNZ), on récupère ensuite l'adresse de la chaîne (aux octets supérieurs, en général dans un registre double HL ou DE).

Pour exemple, nous vous proposons un programme effectuant un scrolling horizontal du texte de la dernière ligne de l'écran grâce à une instruction de la forme :

CALL &A000, « MESSAGE »

Dans ce programme nous aurons besoin de positionner le message à afficher en toute dernière ligne. Il est possible de faire cela à l'aide des vecteurs en RAM (TXT-SET-CURSOR à l'adresse &BB75), mais il existe aussi certains caractères de contrôle qui sont pris en compte par la routine TXT-OUTPUT. Nous vous proposons ci-dessous la liste des codes correspondants et leur effet.

Tous les caractères présentés ci-après sont accessibles à partir du Basic par la commande

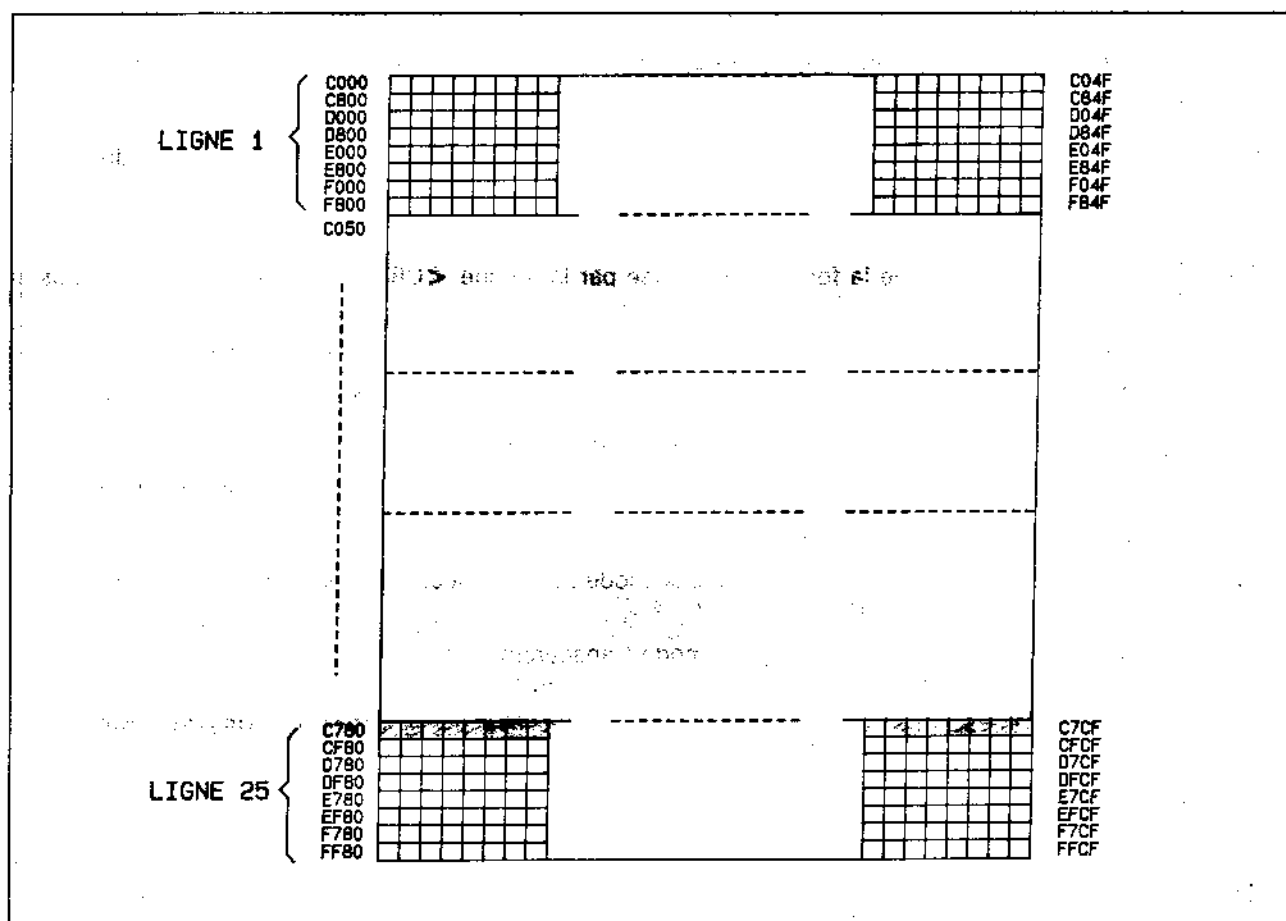
PRINT CHR\$(caractère) <éventuellement CHR\$(complément)>

Hexa	Décimal	Fonction
00	0	Caractère nul - aucune action.
01	1	Visualise sur l'écran les caractères de contrôle. Par exemple en Basic PRINT CHR\$(1);CHR\$(10) affiche une flèche vers le bas.
02	2	Permet de ne pas visualiser le curseur en mode programme (lors d'un INPUT par exemple).
03	3	Rétablit le curseur.
04	4	Change le mode graphique, comme l'instruction MODE. Exemple : PRINT CHR\$(4) ; CHR\$(2) passe en MODE 2.
05	5	Ecrit un caractère à l'emplacement du curseur graphique.
06	6	Annule la commande de masquage sur écran (voir le code 21 décimal).
07	7	Emet le BIP de la cloche.
08	8	Effectue un déplacement d'un caractère vers la gauche.
09	9	Effectue un déplacement d'un caractère vers la droite.
0A	10	Effectue un déplacement d'un caractère vers le bas.
0B	11	Effectue un déplacement d'un caractère vers le haut.
0C	12	Efface la fenêtre courante (équivalent de CLS du Basic).
0D	13	Retour Chariot : place le curseur à la gauche de la ligne courante.
0E	14	Equivalent à l'instruction PAPER PRINT CHR\$(14);CHR\$(2), place la couleur du fond de l'écran à la couleur de l'encre 2.
0F	15	Equivalent à l'instruction PEN.
10	16	Equivalent de la fonction effectuée par la touche <CRL> : efface le caractère sous le curseur.
11	17	Efface tous les caractères précédant le curseur dans la ligne courante.
12	18	Efface tous les caractères suivant le curseur dans la ligne courante.
13	19	Efface tous les caractères depuis le début de la fenêtre courante jusqu'au curseur.
14	20	Efface tous les caractères depuis le curseur jusqu'à la fin de la fenêtre courante.
15	21	En relation avec le code 06 : interdit l'affichage à l'écran.
16	22	Permet d'écrire des caractères en mode transparence, c'est-à-dire d'écrire sur un caractère déjà affiché : PRINT CHR\$(22);CHRS\$1 : mode transparent, PRINT CHR\$(22);CHR\$(0) : mode normal.
17	23	Modifie le mode graphique (accompagné de CHR\$(1) ; CHR\$(2) ; CHR\$(3) ; CHR\$(4).
18	24	Fonctionne en bascule pour établir ou annuler l'inversion vidéo.
19	25	Equivalent de la commande SYMBOL en Basic. Ce code nécessite d'être suivi de 9 paramètres (le numéro du caractère puis les 8 valeurs le définissant).
1A	26	Equivalent de la commande WINDOW en Basic, mais uniquement avec les quatre chiffres définissant la fenêtre.

Hexa	Décimal	Fonction
1B	27	Caractère d'échappement (ESCAPE), utile pour la communication, avec un Minitel par exemple, ou l'imprimante.
1C	28	Equivalent à la fonction INK du Basic mais demande à être suivie obligatoirement des trois chiffres (numéro d'encre, couleur 1, couleur 2).
1D	29	Equivalent à BORDER avec deux couleurs obligatoirement.
1E	30	Place le curseur à la position originelle de la fenêtre.
1F	31	Equivalent de LOCATE Basic, donc suivi de deux caractères.

La méthode retenue est de déplacer tous les pixels un à un d'une position.

Il faut savoir que ces pixels sont inscrits en RAM ECRAN entre les adresses &C000 et &FFFF. Voici la carte de la mémoire écran suite à une initialisation du CPC ou après une instruction MODE n:

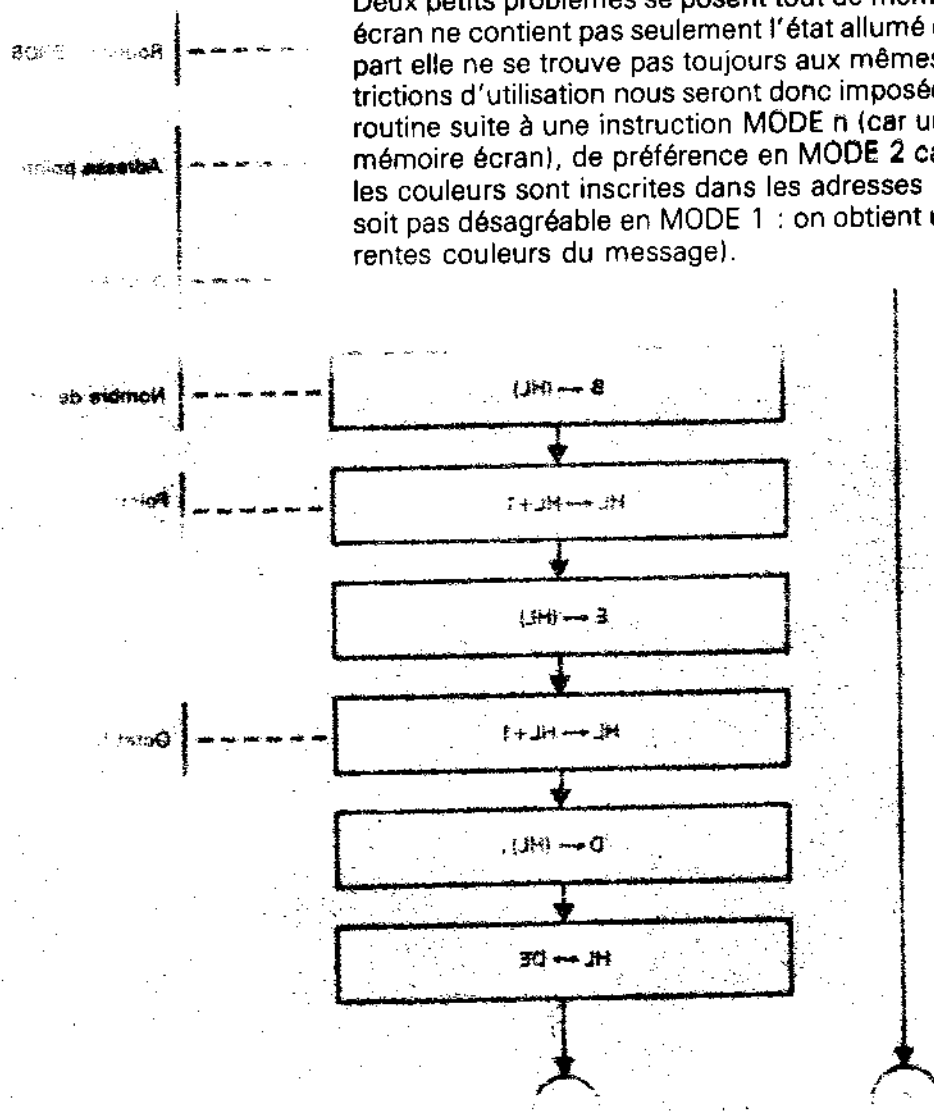


Organisation des adresses écran des CPC après l'instruction MODE.

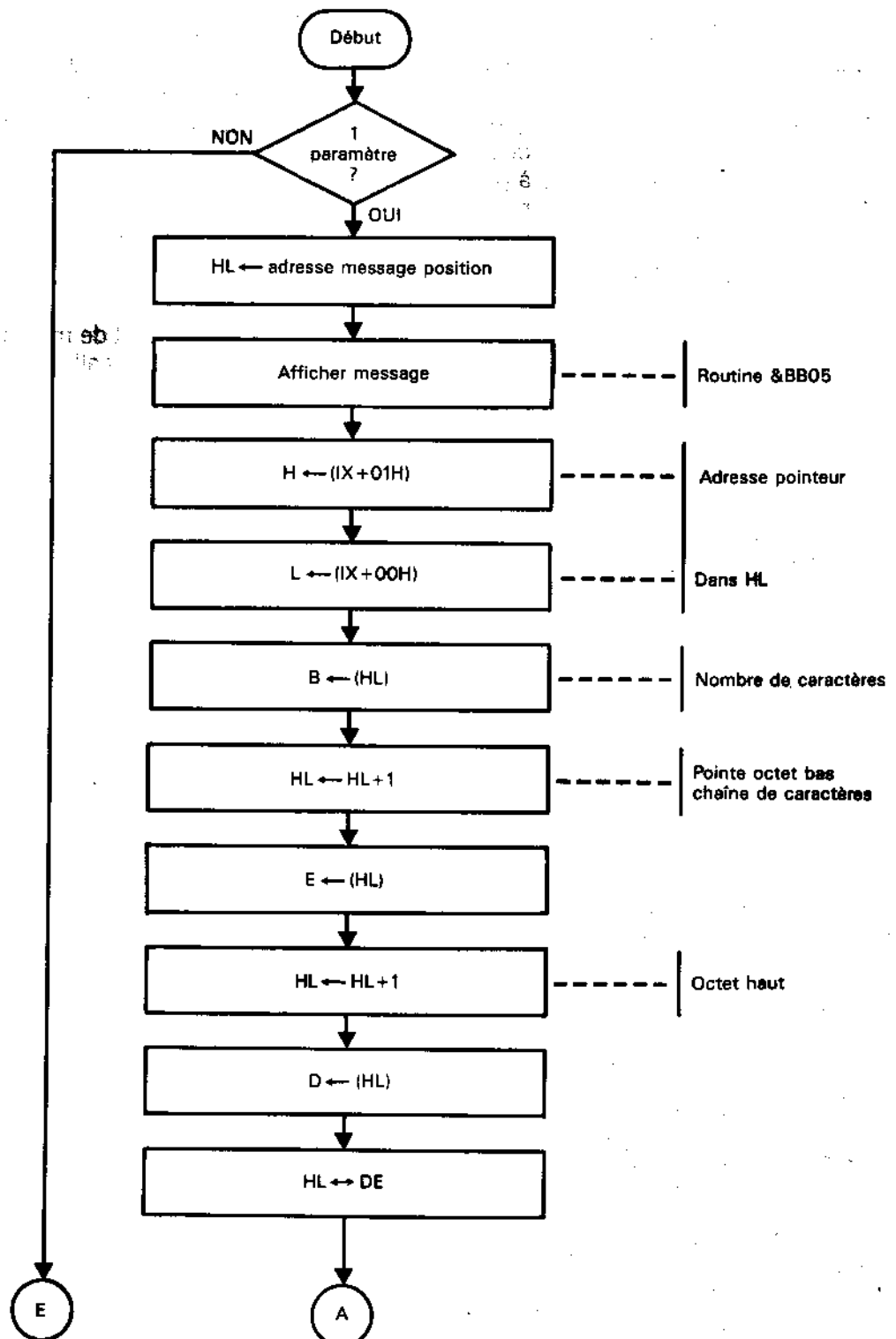
On trouvera donc les pixels de la dernière ligne de caractères entre les adresses &C780-&C7CF, &CF80-&CFCF, ..., &FF80-&FFCF. Lorsqu'on lira une de ces adresses (grâce à PEEK (adresse)) on obtiendra la valeur hexadécimale de huit pixels de cette adresse.

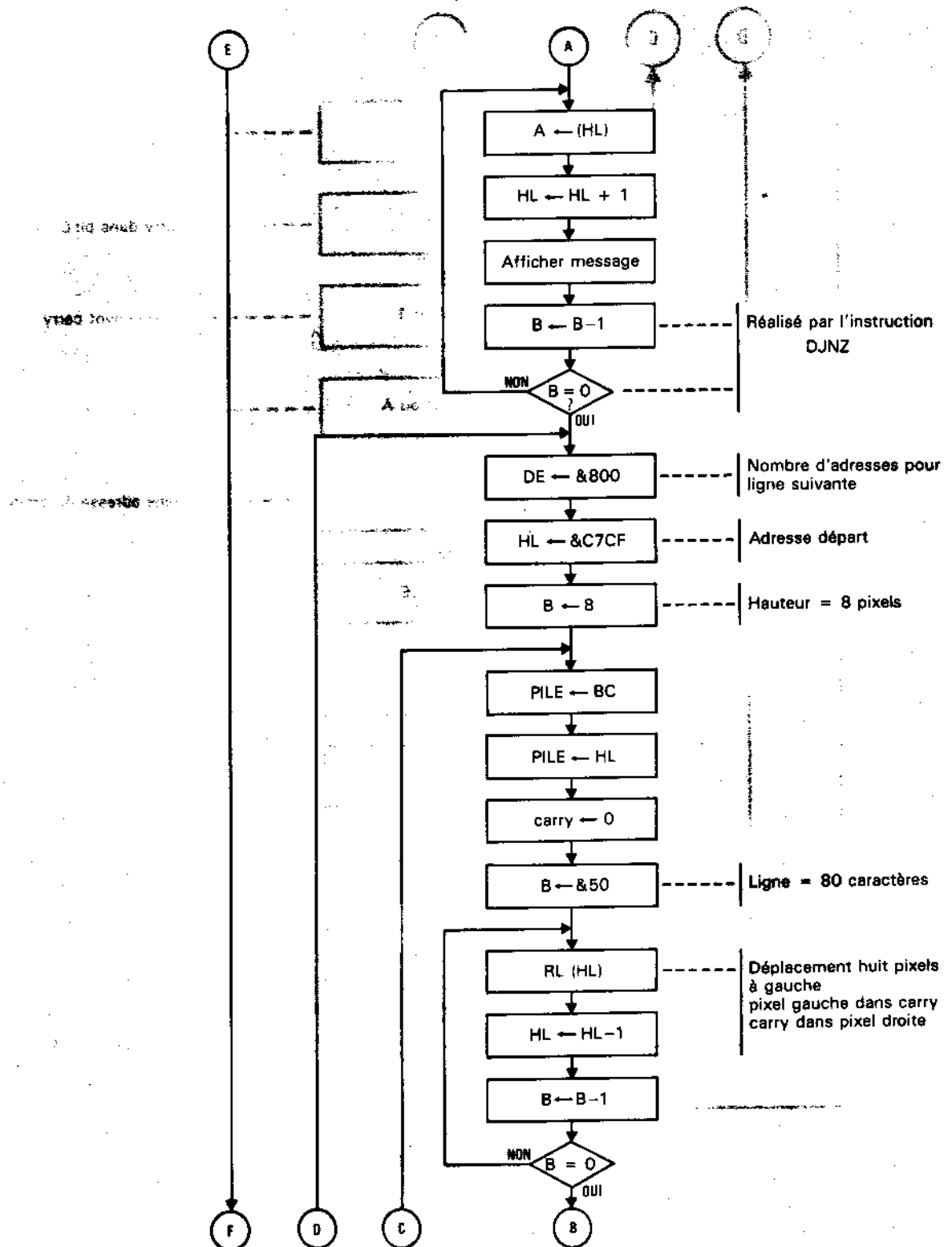
Notre méthode de décalage consistera à lire l'adresse la plus à droite d'une ligne de pixel, décaler ces pixels un à un, récupérer le pixel de gauche, l'inscrire en tant que pixel de droite lors du décalage des pixels de l'adresse immédiatement à gauche, ... etc., jusqu'à l'adresse la plus à gauche. Il ne faudra pas oublier ensuite de replacer le pixel décalé le plus à gauche à la place du pixel le plus à droite sous peine de perdre un pixel à chaque décalage, ce qui ferait disparaître notre message. Les décalages sur une ligne de pixels étant effectués, il faudra passer à la ligne suivante, et ainsi de suite.

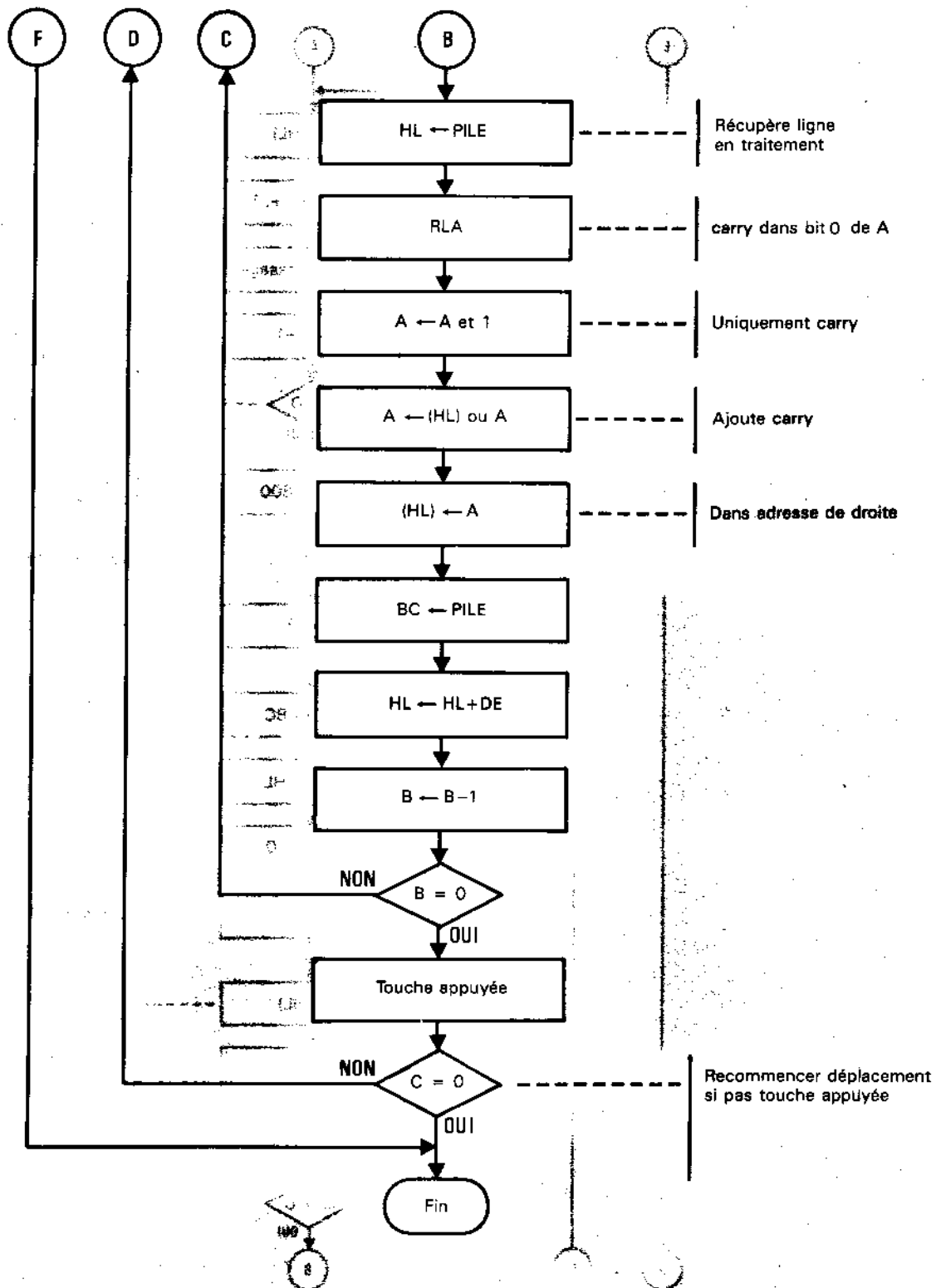
Deux petits problèmes se posent tout de même : d'une part la mémoire écran ne contient pas seulement l'état allumé ou non des pixels, d'autre part elle ne se trouve pas toujours aux mêmes adresses. Certaines restrictions d'utilisation nous seront donc imposées : appeler toujours cette routine suite à une instruction MODE n (car un scrolling décale toute la mémoire écran), de préférence en MODE 2 car dans les autres modes, les couleurs sont inscrites dans les adresses écran (bien que l'effet ne soit pas désagréable en MODE 1 : on obtient un passage dans les différentes couleurs du message).



Nous vous proposons maintenant l'ordinogramme du scrolling horizontal :







On remarquera dans cet ordinogramme l'appel de la routine &BB09 qui va contrôler si un caractère a été frappé sans l'attendre (équivalent de INKEY\$ du Basic), pour nous permettre de quitter la routine de décalage.

Le programme assembleur commenté sera donc :

```

1          ;*****
2          ; * EXECUTION D'UN SCROLLING SUR *
3          ; * UN MESSAGE ECRIT LE LIGNE 25 *
4          ; * PAR L'INSTRUCTION *
5          ; * CALL adresse,"MESSAGE" *
6          ; * DU CALL adresse,a$ *
7          ;*****
8          ;
9          ;
10         ;** ORIGINE D'ASSEMBLAGE **
11         ;
12         ; ORG 0A000H
13         ;
14         ;
15         ;** ADRESSE DE CHARGEMENT DU CODE*
16         ;
17         ; LOAD 0A000H
18         ;
19         ;
20         ;** MISE EN POSITION MESSAGE **
21         ;
22         A000 FE01          CP 01H          ;UN CARACTERE ?
23         A002 C0           RET NZ          ;NON ALORS FIN
24         A003 2147A0       LD HL,POSITI    ;POINTE LES CARACTERE
25         ; DE CONTROLE DE POSITION EN LIG.25
26         A006 7E          AFFICH: LD A,(HL) ;PREND LES
27         ; CARACTERES POINTE
28         A007 23          INC HL           ;POINTE ADRESSE SUIVANTE
29         A008 FEFF        CP OFFH         ;DERNIER CARACTERE ?
30         A00A 2805        JR Z,AFFIC1     ;OUI ALORS SUITE
31         A00C CDSABB      CALL 0BBSAH     ;NON ALORS AFFICHER
32         A00F 18F5        JR AFFICH       ;RECOMMENCER POUR
33         ; CARACTERE SUIVANT
34         ;
35         ;
36         ;** RECUPERATION PARACMETRES **
37         ;
38         A011 DD6601      AFFIC1: LD H,(IX+1) ;RECUPERE
39         ; OCTET HAUT DE L'ADRESSE OU
40         ; SE TROUVENT LES CARACTERIS-
41         ; -TIQUES DE LA CHAINE DE
42         ; CARACTERES.
43         A014 DD6E00      LD L,(IX+0)      ;RECUPERE OCTETS
44         ; BAS DE L'ADRESSE
45         A017 46          LD B,(HL)        ;RECUPERE LE NOMBRE
46         ; DE CARACTERES DE LA CHAINE
47         A018 23          INC HL           ;POINTE ADRESSE SUIVANTE
48         A019 5E          LD E,(HL)        ;RECUPERE OCTET BAS DE
49         ; L'ADRESSE OU DEBUTE LA CHAINE
50         A01A 23          INC HL           ;POINTE ADRESSE SUIVANTE
51         A01B 56          LD D,(HL)        ;RECUPERE OCTET HAUT
52         A01C EB          EX DE,HL         ;MET L'ADRESSE DANS HL
53         ; POUR POINTER LA ZONE
54         ;
55         ;

```

```

56      ;** AFFICHAGE DU MESSAGE **
57      ;
58 A01D 7E      AFFIC2:      LD      A,(HL)      ;PREND UN
59      ;CARACTERE DE LA CHAINE
60 A01E 23      INC      HL      ;POINTE CARACTERE SUIVAN
61 A01F CD5ABB   CALL OBB5AH      ;A L'AFFICHAGE
62 A022 10F9     DJNZ AFFIC2      ;SI ENCORE UN
63      ;CARACTERE ALORS AFFICHER
64      ;SINON EXECUTER SCROLLIG SUIVANT
65      ;
66      ;
67      ;** EXECUTION DU SCROLLING **
68      ;
69 A024 11000B   SCRHB1:      LD      DE,00800H      ;NOMBRE
70      ;D'ADRESSE POUR LIGNE SUIVANTE
71 A027 21CFC7   LD      HL,0C7CFH      ;CHARGE ADRESSE
72      ;SUPERIEURE DROITE
73 A02A 0608     LD      B,08H      ;HUIT LIGNES DE
74      ;PIXELS A TRAITER
75 A02C C5      SCRHB2:      PUSH BC      ;SAUVE NOMBRE DE
76      ;LIGNES A TRAITER
77 A02D E5      PUSH HL      ;SAUVE L'ADRESSE DE
78      ;TRAITEMENT
79 A02E AF      XOR      A      ;MET LE CARRY A ZERO
80 A02F 0650     LD      B,050H      ;NOMBRE DE BLOCS DE HUIT
81      ;PIXELS A TRAITER
82 A031 CB16     SCRHB1:      RL      (HL)      ;DEPLACE LES PIXELS
83      ;DU BLOC POINTE A GAUCHE-LE PIXEL
84      ;DE GAUCHE DANS LE CARRY-L'ETAT
85      ;PRECEDENT DU CARRY ANNULE OU NOM
86      ;LE PIXEL DU BLOC POINTE
87 A033 2B      DEC      HL      ;POINTE L'ADRESSE SUIVANT
88      ;A TRAITER
89 A034 10FB     DJNZ SCRHB1      ;TRAITE ADRESSE
90      ;SUIVANTE SI COMPTE NON NUL
91 A036 E1      POP      HL      ;SINON RECUPERER ADRESSE
92      ;DE LA LIGNE EN TRAITEMENT
93 A037 CB17     RL      A      ;RECUPERER LE CARRY DANS
94 A039 E601     AND      01H      ;MASQUER POUR AVOIR
95      ;UNIQUEMENT LE CARRY
96 A03B B6      OR      (HL)      ;AJOUTER LE BLOC POINTE
97 A03C 77      LD      (HL),A      ;RESAUVEGARDER LE BLOC
98 A03D C1      POP      BC      ;RECUPERER LE NOMBRE DE
99      ;LIGNES RESTANT A TRAITER
100 A03E 19      ADD      HL,DE      ;POINTER LIGNE SUIVANTE
101 A03F 10EB     DJNZ SCRHB2      ;ENCORE UNE LIGNE A
102      ;TRAITER? OUI ALORS DEPLACER
103 A041 CD09BB   CALL OBB09H      ;NON ALORS VOIR SI
104      ;UNE TOUCHE A ETE APPUYEE
105 A044 30DE     JR      NC,SCRHB1      ;SI OUI,FIN
106 A046 C9      RET
107      ;
108      ;
109      ;** CODES DE CONTROLE POUR **
110      ;** POSITIONNER EN LIGNE 25 **
111      ;
112 A047 1F0119FF POSITI:      DEFB 01FH,01H,019H,OFFH
113      ;
114      ;
115      ;*****
116      ;
117      ;
118      ;
119      ;
120      ;
121      ;
122      ;
123      ;
124      ;
125      ;
126      ;
127      ;
128      ;
129      ;
130      ;
131      ;
132      ;
133      ;
134      ;
135      ;
136      ;
137      ;
138      ;
139      ;
140      ;
141      ;
142      ;
143      ;
144      ;
145      ;
146      ;
147      ;
148      ;
149      ;
150      ;
151      ;
152      ;
153      ;
154      ;
155      ;
156      ;
157      ;
158      ;
159      ;
160      ;
161      ;
162      ;
163      ;
164      ;
165      ;
166      ;
167      ;
168      ;
169      ;
170      ;
171      ;
172      ;
173      ;
174      ;
175      ;
176      ;
177      ;
178      ;
179      ;
180      ;
181      ;
182      ;
183      ;
184      ;
185      ;
186      ;
187      ;
188      ;
189      ;
190      ;
191      ;
192      ;
193      ;
194      ;
195      ;
196      ;
197      ;
198      ;
199      ;
200      ;
201      ;
202      ;
203      ;
204      ;
205      ;
206      ;
207      ;
208      ;
209      ;
210      ;
211      ;
212      ;
213      ;
214      ;
215      ;
216      ;
217      ;
218      ;
219      ;
220      ;
221      ;
222      ;
223      ;
224      ;
225      ;
226      ;
227      ;
228      ;
229      ;
230      ;
231      ;
232      ;
233      ;
234      ;
235      ;
236      ;
237      ;
238      ;
239      ;
240      ;
241      ;
242      ;
243      ;
244      ;
245      ;
246      ;
247      ;
248      ;
249      ;
250      ;
251      ;
252      ;
253      ;
254      ;
255      ;
256      ;
257      ;
258      ;
259      ;
260      ;
261      ;
262      ;
263      ;
264      ;
265      ;
266      ;
267      ;
268      ;
269      ;
270      ;
271      ;
272      ;
273      ;
274      ;
275      ;
276      ;
277      ;
278      ;
279      ;
280      ;
281      ;
282      ;
283      ;
284      ;
285      ;
286      ;
287      ;
288      ;
289      ;
290      ;
291      ;
292      ;
293      ;
294      ;
295      ;
296      ;
297      ;
298      ;
299      ;
300      ;
301      ;
302      ;
303      ;
304      ;
305      ;
306      ;
307      ;
308      ;
309      ;
310      ;
311      ;
312      ;
313      ;
314      ;
315      ;
316      ;
317      ;
318      ;
319      ;
320      ;
321      ;
322      ;
323      ;
324      ;
325      ;
326      ;
327      ;
328      ;
329      ;
330      ;
331      ;
332      ;
333      ;
334      ;
335      ;
336      ;
337      ;
338      ;
339      ;
340      ;
341      ;
342      ;
343      ;
344      ;
345      ;
346      ;
347      ;
348      ;
349      ;
350      ;
351      ;
352      ;
353      ;
354      ;
355      ;
356      ;
357      ;
358      ;
359      ;
360      ;
361      ;
362      ;
363      ;
364      ;
365      ;
366      ;
367      ;
368      ;
369      ;
370      ;
371      ;
372      ;
373      ;
374      ;
375      ;
376      ;
377      ;
378      ;
379      ;
380      ;
381      ;
382      ;
383      ;
384      ;
385      ;
386      ;
387      ;
388      ;
389      ;
390      ;
391      ;
392      ;
393      ;
394      ;
395      ;
396      ;
397      ;
398      ;
399      ;
400      ;
401      ;
402      ;
403      ;
404      ;
405      ;
406      ;
407      ;
408      ;
409      ;
410      ;
411      ;
412      ;
413      ;
414      ;
415      ;
416      ;
417      ;
418      ;
419      ;
420      ;
421      ;
422      ;
423      ;
424      ;
425      ;
426      ;
427      ;
428      ;
429      ;
430      ;
431      ;
432      ;
433      ;
434      ;
435      ;
436      ;
437      ;
438      ;
439      ;
440      ;
441      ;
442      ;
443      ;
444      ;
445      ;
446      ;
447      ;
448      ;
449      ;
450      ;
451      ;
452      ;
453      ;
454      ;
455      ;
456      ;
457      ;
458      ;
459      ;
460      ;
461      ;
462      ;
463      ;
464      ;
465      ;
466      ;
467      ;
468      ;
469      ;
470      ;
471      ;
472      ;
473      ;
474      ;
475      ;
476      ;
477      ;
478      ;
479      ;
480      ;
481      ;
482      ;
483      ;
484      ;
485      ;
486      ;
487      ;
488      ;
489      ;
490      ;
491      ;
492      ;
493      ;
494      ;
495      ;
496      ;
497      ;
498      ;
499      ;
500      ;
501      ;
502      ;
503      ;
504      ;
505      ;
506      ;
507      ;
508      ;
509      ;
510      ;
511      ;
512      ;
513      ;
514      ;
515      ;
516      ;
517      ;
518      ;
519      ;
520      ;
521      ;
522      ;
523      ;
524      ;
525      ;
526      ;
527      ;
528      ;
529      ;
530      ;
531      ;
532      ;
533      ;
534      ;
535      ;
536      ;
537      ;
538      ;
539      ;
540      ;
541      ;
542      ;
543      ;
544      ;
545      ;
546      ;
547      ;
548      ;
549      ;
550      ;
551      ;
552      ;
553      ;
554      ;
555      ;
556      ;
557      ;
558      ;
559      ;
560      ;
561      ;
562      ;
563      ;
564      ;
565      ;
566      ;
567      ;
568      ;
569      ;
570      ;
571      ;
572      ;
573      ;
574      ;
575      ;
576      ;
577      ;
578      ;
579      ;
580      ;
581      ;
582      ;
583      ;
584      ;
585      ;
586      ;
587      ;
588      ;
589      ;
590      ;
591      ;
592      ;
593      ;
594      ;
595      ;
596      ;
597      ;
598      ;
599      ;
600      ;
601      ;
602      ;
603      ;
604      ;
605      ;
606      ;
607      ;
608      ;
609      ;
610      ;
611      ;
612      ;
613      ;
614      ;
615      ;
616      ;
617      ;
618      ;
619      ;
620      ;
621      ;
622      ;
623      ;
624      ;
625      ;
626      ;
627      ;
628      ;
629      ;
630      ;
631      ;
632      ;
633      ;
634      ;
635      ;
636      ;
637      ;
638      ;
639      ;
640      ;
641      ;
642      ;
643      ;
644      ;
645      ;
646      ;
647      ;
648      ;
649      ;
650      ;
651      ;
652      ;
653      ;
654      ;
655      ;
656      ;
657      ;
658      ;
659      ;
660      ;
661      ;
662      ;
663      ;
664      ;
665      ;
666      ;
667      ;
668      ;
669      ;
670      ;
671      ;
672      ;
673      ;
674      ;
675      ;
676      ;
677      ;
678      ;
679      ;
680      ;
681      ;
682      ;
683      ;
684      ;
685      ;
686      ;
687      ;
688      ;
689      ;
690      ;
691      ;
692      ;
693      ;
694      ;
695      ;
696      ;
697      ;
698      ;
699      ;
700      ;
701      ;
702      ;
703      ;
704      ;
705      ;
706      ;
707      ;
708      ;
709      ;
710      ;
711      ;
712      ;
713      ;
714      ;
715      ;
716      ;
717      ;
718      ;
719      ;
720      ;
721      ;
722      ;
723      ;
724      ;
725      ;
726      ;
727      ;
728      ;
729      ;
730      ;
731      ;
732      ;
733      ;
734      ;
735      ;
736      ;
737      ;
738      ;
739      ;
740      ;
741      ;
742      ;
743      ;
744      ;
745      ;
746      ;
747      ;
748      ;
749      ;
750      ;
751      ;
752      ;
753      ;
754      ;
755      ;
756      ;
757      ;
758      ;
759      ;
760      ;
761      ;
762      ;
763      ;
764      ;
765      ;
766      ;
767      ;
768      ;
769      ;
770      ;
771      ;
772      ;
773      ;
774      ;
775      ;
776      ;
777      ;
778      ;
779      ;
780      ;
781      ;
782      ;
783      ;
784      ;
785      ;
786      ;
787      ;
788      ;
789      ;
790      ;
791      ;
792      ;
793      ;
794      ;
795      ;
796      ;
797      ;
798      ;
799      ;
800      ;
801      ;
802      ;
803      ;
804      ;
805      ;
806      ;
807      ;
808      ;
809      ;
810      ;
811      ;
812      ;
813      ;
814      ;
815      ;
816      ;
817      ;
818      ;
819      ;
820      ;
821      ;
822      ;
823      ;
824      ;
825      ;
826      ;
827      ;
828      ;
829      ;
830      ;
831      ;
832      ;
833      ;
834      ;
835      ;
836      ;
837      ;
838      ;
839      ;
840      ;
841      ;
842      ;
843      ;
844      ;
845      ;
846      ;
847      ;
848      ;
849      ;
850      ;
851      ;
852      ;
853      ;
854      ;
855      ;
856      ;
857      ;
858      ;
859      ;
860      ;
861      ;
862      ;
863      ;
864      ;
865      ;
866      ;
867      ;
868      ;
869      ;
870      ;
871      ;
872      ;
873      ;
874      ;
875      ;
876      ;
877      ;
878      ;
879      ;
880      ;
881      ;
882      ;
883      ;
884      ;
885      ;
886      ;
887      ;
888      ;
889      ;
890      ;
891      ;
892      ;
893      ;
894      ;
895      ;
896      ;
897      ;
898      ;
899      ;
900      ;
901      ;
902      ;
903      ;
904      ;
905      ;
906      ;
907      ;
908      ;
909      ;
910      ;
911      ;
912      ;
913      ;
914      ;
915      ;
916      ;
917      ;
918      ;
919      ;
920      ;
921      ;
922      ;
923      ;
924      ;
925      ;
926      ;
927      ;
928      ;
929      ;
930      ;
931      ;
932      ;
933      ;
934      ;
935      ;
936      ;
937      ;
938      ;
939      ;
940      ;
941      ;
942      ;
943      ;
944      ;
945      ;
946      ;
947      ;
948      ;
949      ;
950      ;
951      ;
952      ;
953      ;
954      ;
955      ;
956      ;
957      ;
958      ;
959      ;
960      ;
961      ;
962      ;
963      ;
964      ;
965      ;
966      ;
967      ;
968      ;
969      ;
970      ;
971      ;
972      ;
973      ;
974      ;
975      ;
976      ;
977      ;
978      ;
979      ;
980      ;
981      ;
982      ;
983      ;
984      ;
985      ;
986      ;
987      ;
988      ;
989      ;
990      ;
991      ;
992      ;
993      ;
994      ;
995      ;
996      ;
997      ;
998      ;
999      ;
1000     ;

```

On retrouve la routine d'affichage de caractères aux adresses &A006-&A00F qui positionne le curseur en ligne 25 colonne 1, ainsi qu'aux adresses &A01D-&A022 pour afficher le message.

A l'adresse &A024, le nombre hexadécimal &800 chargé dans le double registre DE correspond au nombre d'adresses à ajouter pour passer à la ligne de pixels suivante.

A l'adresse &A02F, le chargement dans B de la valeur &50 correspond au nombre d'adresses d'une ligne (&50 = 80_{decimal}).

Pour ceux qui ne possèdent pas d'assembleur, mais que ce programme intéresse, nous vous soumettons le chargeur Basic des instructions ci-dessous :

```

10 REM *****
20 REM * EXECUTION D'UN SCROLLING *
30 REM * SUR UN MESSAGE EN LIGNE 25 *
40 REM * PAR L'INSTRUCTION *
50 REM * CALL &A000,"MESSAGE" *
60 REM *****
70 REM
80 REM *** CHARGEUR BASIC ***
90 ADR = &A000:I = 0
100 RESTORE 320
110 READ A$
120 IF A$ = "XX" THEN 200
130 B$ = "&" + A$
140 B = VAL(B$)
150 POKE ADR,B
160 ADR = ADR + 1
170 I = I + 1
180 GOTO 110
190 REM
200 REM *** SAUVEGARDE ***
210 PRINT "POUR SAUVEGARDER LA ROUTINE E
N BINAIRE"
220 PRINT "SAVE ";CHR$(34);"ROUTINE.BIN"
;CHR$(34);",B,&A000,";RIGHT$(STR$(I),LEN
(STR$(I))-1)
230 PRINT
240 PRINT "CHARGEMENT PAR LOAD ";CHR$(34)
);"ROUTINE.BIN";CHR$(34);",&A000"
250 PRINT "MEMORY &9FFF"
260 PRINT
270 PRINT "EXECUTION PAR CALL &A000,"MES
SAGE"
280 STOP
290 REM
300 REM

```

```

310 REM *** POSITIONNER MESSAGE ***
320 DATA FE,01,C0,21,47,A0,7E,23
330 DATA FE,FF,28,05,CD,5A,BB,18
340 DATA F5
350 REM
360 REM *** RECUPERATION PARAMETRES ***
370 DATA DD,66,01,DD,6E,00,46,23
380 DATA 5E,23,56,EB,7E,23,CD,5A
390 DATA BB,10,F9
400 REM
410 REM *** EXECUTION DU SCROLLING ***
420 DATA 11,00,0B,21,CF,C7,06,0B
430 DATA C5,E5,AF,06,50,CB,16,2B
440 DATA 10,FB,E1,CB,17,E6,01,B6
450 DATA 77,C1,19,10,EB
460 REM
470 REM *** VOIR TOUCHE ENFONCEE ***
480 DATA CD,09,BB,30,DE,C9
490 REM
500 REM *** CODES POSITION LIGNE 25 ***
510 DATA 1F,01,19,FF
520 REM
530 REM *** FIN DE MESSAGE ***
540 DATA FF
550 REM
560 REM
570 REM *** FIN DE DATA ***
580 DATA XX

```

L'appel de l'instruction se fera sous la forme :

CALL &A000,« MESSAGE »

si vous possédez un CPC-6128.

Les possesseurs d'un CPC-464 devront ruser pour faire digérer cette instruction à leur ordinateur. Ils devront en effet passer par l'intermédiaire d'une variable alphanumérique, la routine sera donc appelée en deux temps :

A\$ = « MESSAGE »
CALL &A000,@A\$

Ce qui ne change rien aux explications données plus haut, c'est cette fois le pointeur de la variable A\$ qui est transmis aux travers des adresses IX+00H et IX+01H.

Tout comme pour le passage des variables numériques entières, il est possible de passer à la routine plusieurs chaînes de caractères par les instructions :

CALL &A000, « CHAINE1 », « CHAINE2 », ..., « CHAINE_n »
ou **CALL &A000,@A1\$,@2\$, ... @An\$**

LA RECUPERATION DES VARIABLES NUMERIQUES

S'il est possible de transmettre à la routine des variables, il est inversement possible de récupérer des résultats suite à un traitement, dans une variable.

Il suffit en effet de passer à la routine le pointeur d'une variable pour qu'elle aille y sauvegarder des valeurs, que l'on pourra relire sous Basic dans la variable.

Pour passer le pointeur d'une variable numérique, il suffit qu'elle existe, elle sera donc créée par l'instruction `VARIABLE% = 0` par exemple.

Lors de l'appel de la routine il suffira d'y adjoindre son pointeur `@VARIABLE%`, qui sera lu dans la pile pointée par IX, l'adresse sera ensuite connue et les éventuels résultats rangés à cette adresse.

LA RECUPERATION DES VARIABLES ALPHANUMERIQUES

Il est également possible de récupérer des chaînes de caractères traitées dans une routine en langage machine.

Pour une seule chaîne de caractères, il suffit de créer une variable par `A$ = SPACES(n)`, n étant généralement le nombre de caractères maximal que pourra contenir la chaîne, puis on transmet le pointeur de cette chaîne par l'instruction :

CALL &adresse,@A\$

Dans la routine il suffira de récupérer l'adresse de la variable à l'aide du pointeur, et de la traiter, en n'oubliant pas de spécifier le nombre de caractères la composant.

Nous allons prendre à titre d'exemple une routine permettant de convertir une chaîne de caractères contenant un message, en une autre chaîne contenant la conversion ASCII de tous ces caractères. Cette routine pourra vous être utile pour modifier le programme d'affichage d'un message.

Nous nous proposons d'appeler la routine sous la forme :

A\$ = "CHAINE DE CARACTERES"

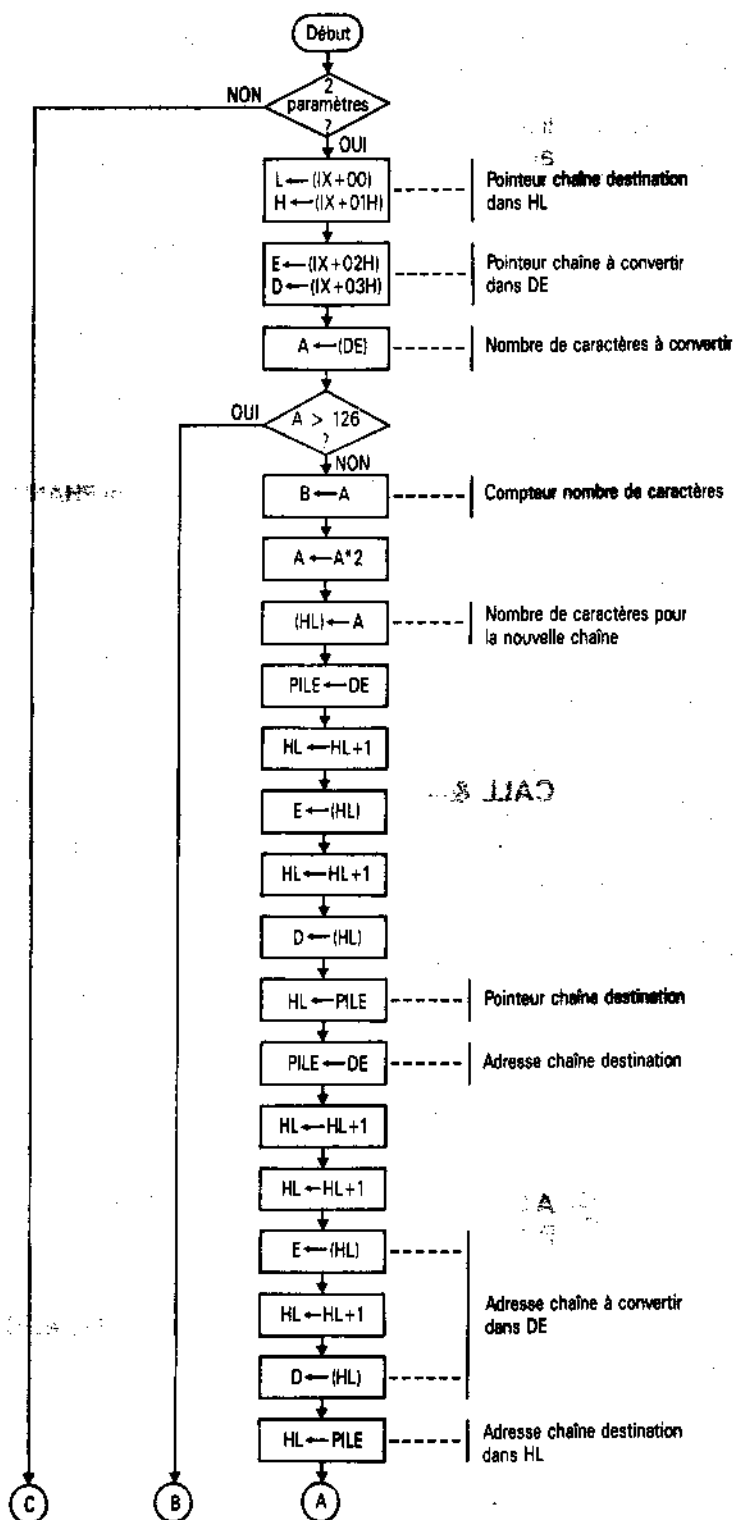
B\$ = SPACES (254)

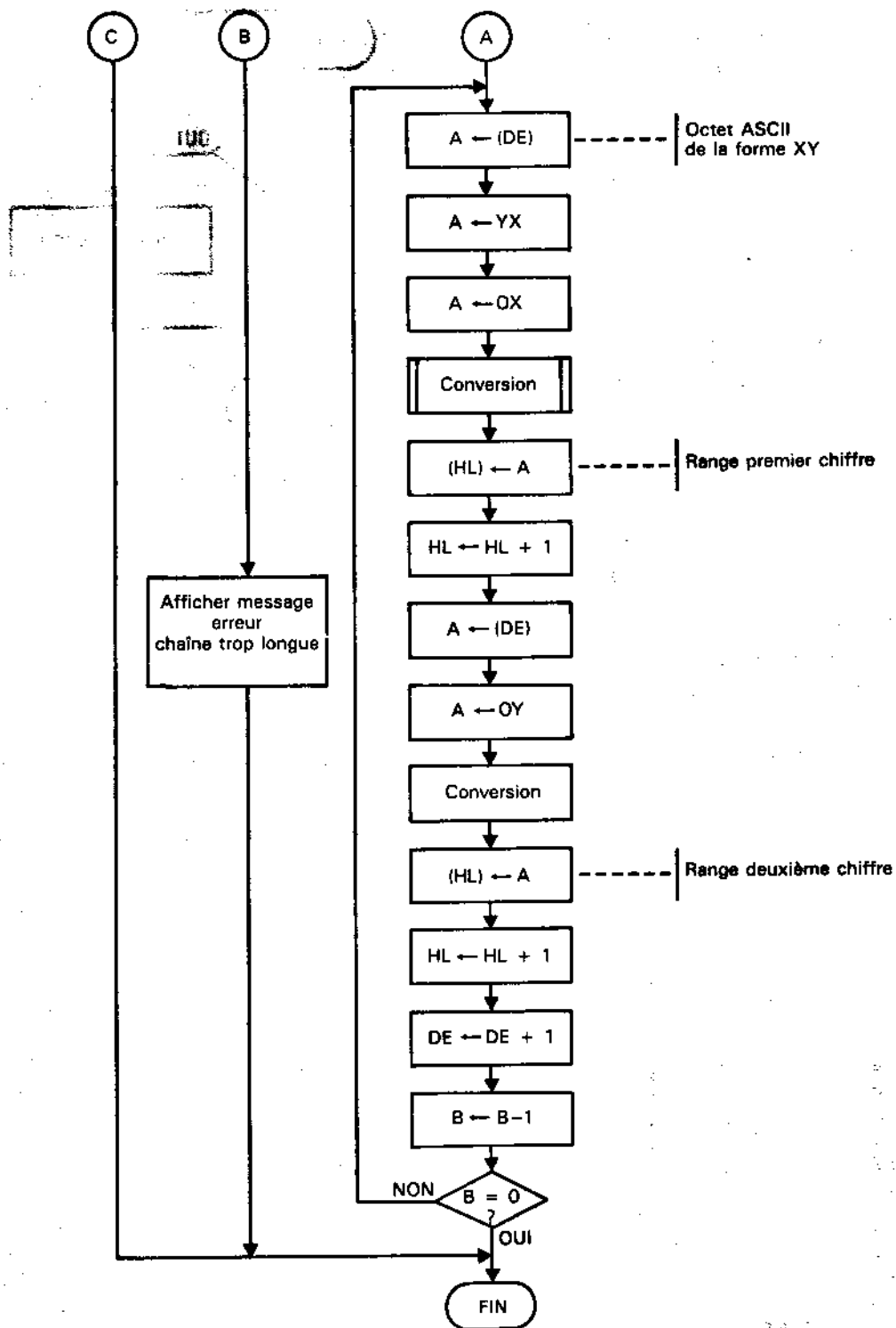
CALL &A000,@A\$,@B\$

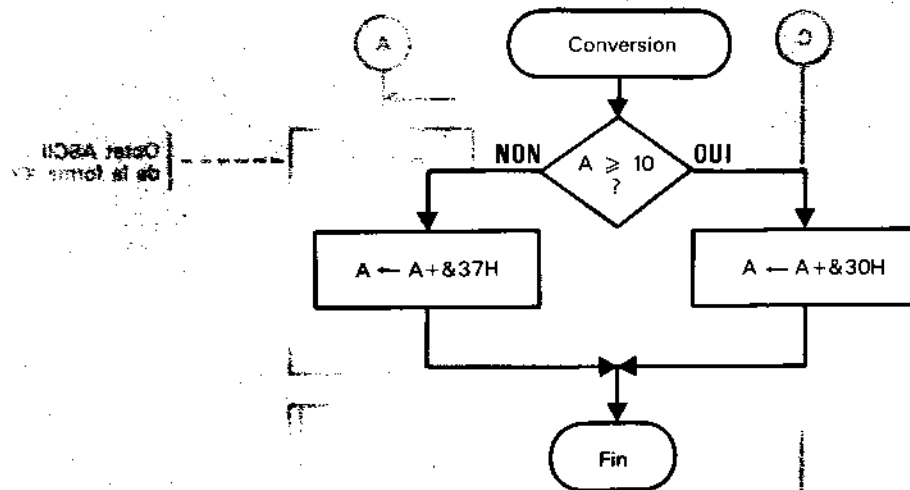
où A\$ contient la chaîne à convertir, et B\$ la conversion de tous les caractères.

Il faut d'abord savoir qu'une chaîne de caractères peut contenir au maximum 255 caractères, or, comme la conversion d'un caractère alphanumérique se traduit en deux chiffres hexadécimaux, B\$ ne pouvant ainsi contenir qu'au maximum 254 caractères, A n'en contiendra pas plus de 127.

Voici l'ordinogramme de conversion proposé :







L'ordinogramme d'affichage du message d'erreur n'est pas ici explicité car déjà vu précédemment.

On retrouve la lecture des deux pointeurs des variables qui sont traités afin de lire la chaîne de caractères et d'inscrire la chaîne convertie aux adresses correspondantes.

Le sous-programme de conversion est intéressant, car très rapide dans son exécution, et doit être retenu pour d'éventuelles utilisations ultérieures (procédures de communications, par exemple). Ce type de traitement provient d'une réflexion mathématique sur les différents codes ASCII des caractères. Il impose malheureusement une limitation aux caractères alphabétiques de A à F et aux chiffres de 0 à 9 (hexadécimal oblige).

Le programme en langage d'assemblage qui en découle est le suivant :

```

1      ; *****
2      ; * PROGRAMME DE CONVERSION D'UNE *
3      ; * CHAÎNE DE CARACTÈRE EN UNE   *
4      ; * AUTRE CHAÎNE DONNANT LES     *
5      ; * VALEURS ASCII DE CHACUN DES  *
6      ; * CARACTÈRES                    *
7      ; *****
8
9
10     ; ***  ORIGINE D'ASSEMBLAGE  ***
11
12     ORG  0A000H
13
14     ; ***  ADRESSE DE CHARGEMENT  ***
15     LOAD 0A000H
16
17     ; ***  DEBUT DE PROGRAMME    ***
18
19 A000 FE02      CP  02H              ; DEUX PARAMETRES TRANSMIS
20 A002 C0        RET  NZ
21
22 A003 DD6E00    LD  L, (IX+00H)      ; OCTET BAS DU
23 ; POINTEUR DE LA CHAÎNE DESTINATION
24 A006 DD6601    LD  H, (IX+01H)      ; OCTET HAUT
  
```

```

25 A009 DD5E02          LD  E,(IX+02H)          ;OCTET BAS DU
26                      ;POINTEUR DE LA CHAINE DE CARACTER
27 A00C DD5603          LD  D,(IX+03H)          ;OCTET HAUT
28
29 A00F 1A              LD  A,(DE)              ;RECUPERE NOMBRE DE
30                      ;DE CARACTERE DE LA CHAINE
31 A010 CB7F            BIT  7,A                ;MAXIMUM 127 CARACTERES?
32 A012 2030            JR   NZ,ERREUR          ;SINON ERREUR
33 A014 47              LD  B,A                ;SAUVEGARDE NOMBRE DE
34                      ;CARACTERES POUR BOUCLE
35 A015 07              RLCA                    ;MULTIPLICATION PAR DEUX
36                      ;DU NOMBRE DE CARACTERES
37 A016 77              LD  (HL),A             ;POUR LA LONGUEUR DE
38                      ;LA CHAINE DESTINATION
39 A017 D5              PUSH DE                 ;SAUVEGARDE TEMPORAIRE
40                      ;ADRESSE DU POINTEUR DE LA CHAINE
41                      ;A TRAITER
42 A018 23              INC  HL                 ;POINTE OCTET BAS DE
43                      ;L'ADRESSE DE LA CHAINE DESTIN.
44 A019 5E              LD  E,(HL)             ;POUR CHARGER
45 A01A 23              INC  HL                 ;POINTE OCTET HAUT
46 A01B 56              LD  D,(HL)             ;POUR LE CHARGER
47 A01C E1              POP  HL                ;RECUPERE ADRESSE DE LA
48                      ;CHAINE A TRAITER
49 A01D D5              PUSH DE                 ;SAUVE TEMPORAIREMENT
50                      ;L'ADRESSE DE LA CHAINE DESTINATIO
51 A01E 23              INC  HL                 ;COMME CI DESSUS RECUPERE
52 A01F 5E              LD  E,(HL)             ;L'ADRESSE COMPLETE
53 A020 23              INC  HL                 ;DE LA CHAINE A TRAITER
54 A021 56              LD  D,(HL)             ;DANS DE
55 A022 E1              POP  HL                ;RECUPERE ADRESSE DESTIN.
56
57 A023 1A              PROG1: LD  A,(DE)        ;PREND UN CARACTER
58 A024 07              RLCA                    ;DECALE QUATRE
59 A025 07              RLCA                    ;FOIS POUR
60 A026 07              RLCA                    ;INVERSER OCTET HAUT
61 A027 07              RLCA                    ;ET BAS
62 A028 E60F            AND  0FH                ;GARDE OCTET HAUT
63 A02A CD3AA0          105-10 CALL CONVER        ;A CONVERTIR EN ASCII
64 A02D 77              LD  (HL),A             ;SAUVEGARDE DANS DESTIN.
65 A02E 23              INC  HL                 ;ADRESSE SUIVANTE POUR
66                      ;DEUXIEME OCTET
67 A02F 1A              LD  A,(DE)             ;REPREND CARACTERE
68 A030 E60F            AND  0FH                ;GARDE OCTET BAS
69 A032 CD3AA0          CALL CONVER            ;A CONVERTIR
70 A035 77              LD  (HL),A             ;PUIS SAUVEGARDE
71 A036 23              INC  HL                 ;ADRESSE SUIVANTE
72 A037 13              INC  DE                 ;CARACTERE SUIVANT
73 A038 10E9            DJNZ PROG1              ;SI ENCORE UN
74                      ;CARACTERE ALORS RECOMMENCER
75
76                      ;
77                      ;*** SOUS PROGRAMME DE CONVERSION
78                      ;
79 A03A FE0A            CONVER: CP  0AH          ;COMPARE A 10
80 A03C 3003            JR   NC,SUP            ;SI SUPERIEUR OU EGAL
81                      ;ALORS ALLER A SUP
82 A03E C630            ADD  A,030H            ;SINON CHIFFRE
83 A040 C9              RET                    ;RETOUR

```

```

84 A041 C637      SUP:      ADD  A,037H      ; CARACTERE
85                ; ALPHABETIQUE
86 A043 C9        RET                ; RETOUR
87
88                ; *** TRAITEMENT CHAINE TROP LONGUE
89                ;
90 A044 2151A0     ERREUR:    LD    HL,MSGERR
91 A047 7E        ERR1:      LD    A,(HL)
92 A048 FEFF      CP    OFFH
93 A04A C8        RET    Z
94 A04E CD5ABB    CALL  OBB5AH
95 A04E 23        INC    HL
96 A04F 1BF6      JR     ERR1
97                ;
98                ;
99                ; *** MESSAGE ERREUR ***
100               ;
101 A051 43484149  MSGERR:    DEFB  "CHAINE"
101 A055 4E45
102 A057 2054524F  DEFB  " TROP "
102 A05B 5020
103 A05D 4C4F4E47  DEFB  "LONGUE"
103 A061 5545
104 A063 FF        DEFB  OFFH
105
106                END

```

A l'adresse &A010 le test du nombre de caractères est effectué sur le bit le plus à gauche du nombre, car celui-ci passe à 1 dès que ce nombre est supérieur ou égal à 128.

On retiendra la partie de programme entre les adresses &A019 et &A01B qui consiste à mettre dans DE le contenu des adresses pointées par HL et HL + 1. Si l'on y ajoute l'instruction EXG, DE, HL, on obtient un résultat correspondant à $HL \leftarrow (HL + 1)(LH)$.

Le listing du chargeur Basic est donné ci-après.

II - Les RSX

(Sur ce sujet reportez-vous également à la Partie 4, Chap. 2.9).

Après avoir étudié différentes façon de transmettre des paramètres à une routine en langage Machine, nous pouvons nous lancer dans la création des RSX.

QU'EST-CE QU'UNE RSX ?

Mot bien souvent magique, qui rebute beaucoup de programmeurs Basic, mais qui va être sans secret pour vous d'ici quelques instants.

RSX vient de la contraction du terme *Resident System eXtension*, qui, mot à mot veut dire extension résidente dans le système. Ces extensions sont dans notre cas des commandes Basic qui sont intégrées aux commandes déjà existantes, à la seule particularité qu'elles doivent être précédées du caractère ù (ou la barre verticale I) de code ASCII 124).

```

1      ;*****
2      ;* REALISATION D'UNE INSTRUCTION *
3      ;*      PERSONNELLE      *
4      ;*  <NOM> -> MESSAGE PERSONNEL *
5      ;*      *
6      ;*      MODIFIABLE FACILEMENT *
7      ;*      PAR SON UTILISATEUR   *
8      ;*      *
9      ;*****
10     ;
11     ;**      ORIGINE D'ASSEMBLAGE      **
12     ;
13     ;          ORG  0A000H
14     ;
15     ;
16     ;**      ADRESSE DE CHARGEMENT DU      **
17     ;**      CODE MACHINE      **
18     ;
19     ;          LOAD 0A000H
20     ;
21     ;
22     ;*****
23     ;* INSTALLATION DE L'INSTRUCTION *
24     ;*  SOUS LA FORME D'UNE R.S.X.  *
25     ;*****
26     ;
27     A000 2119A0  DEBUT:      LD  HL,KERNAL      ;POINTE 4 OCTET
28     ;UTILISES PAR LA ROUTINE MONITEUR
29     A003 010FA0  ;          LD  BC,VECTEU      ;POINTE LA TABLE DES
30     ; INSTRUCTIONS
31     A006 CDD1B0  ;          CALL 0BCD1H      ;AJOUTE LA R.S.X.
32     A009 3EC9    ;          LD  A,0C9H      ;CHARGE RET POUR
33     A00B 3200A0  ;          LD  (DEBUT),A    ;INTERDIRE AUTRE APPE
34     A00E C9      ;          RET            ;FIN INSTALLATION
35     ;
36     A00F 14A0    VECTEU:    DEFW TABLE      ;ADRESSE DE LA
37     ;TABLE DES INSTRUCTIONS
38     A011 C31DA0  ;          JP  NOM          ;SAUT AU TRAITEMENT
39     ;DE L'INSTRUCTION <NOM>
40     ;
41     A014 57454B  TABLE:    DEFB "WEK"      ;DEBUT DE LA
42     ;DEFINITION DE L'INSTRUCTION
43     A017 C1      ;          DEFB 080H+"A"    ;DERNIERE LETTRE
44     A01B 00      ;          DEFB 0          ;FIN DE LA TABLE
45     ;
46     KERNAL:      DEFS 4      ;4 OCTETS POUR LA
47     ;ROUTINE KL-LOG-TEXTE
48     ;
49     ;*****
50     ;
51     ;*****
52     ;*      TRAITEMENT DE L'INSTRUCTION *
53     ;*****
54     ;
55     ;
56     A01D 212AA0  NOM:      LD  HL,MESSAG     ;POINTE
57     ;L'ADRESSE DE DEBUT DU
58     ;MESSAGE A AFFICHER
59     A020 7E      NOM1:     LD  A,(HL)      ;CHARGE
60     ;CODE CARACTERE DANS A

```

POURQUOI LES RSX ?

Oui, pourquoi créer de nouveaux noms d'instructions alors que pour appeler une routine en langage Machine, il suffit d'effectuer un appel à cette routine par CALL ?

Imaginez un Basic uniquement composé d'instructions CALL à différentes adresses, quelle mémoire devrez-vous posséder avant d'utiliser celle de votre ordinateur ! Il est bien plus simple de retenir une instruction par un nom que par un numéro, qui plus est hexadécimal !

COMMENT CREER UNE RSX ?

La création d'une RSX est très simple, maintenant que vous savez manier l'instruction CALL.

Un vecteur du système d'exploitation Basic est spécialement implanté pour cela :

KL-LOG-EXT

(voir Partie 4, Chap. 2.7 page 54).

Ce vecteur s'utilise dans une petite routine en langage machine qui définit la ou les extensions à intégrer au Basic.

Il requiert différents paramètres qui sont :

- l'adresse de la table des instructions,
- l'adresse d'une zone libre de quatre octets.

La routine de liaison des RSX se présentera toujours sous la forme de l'algorithme suivant :

- DEBUT
 - Charger HL avec l'adresse de la zone de quatre octets
 - Charger BC avec l'adresse de la table des instructions
 - PROCEDURE KL-LOG-EXT
 - Placer le code de RET au début de la routine
- FIN

Ce qui nous donne en langage d'assemblage :

```

DEBUT  LD      HL,KERNEL
        LD      BC,VECTEU
        CALL   0BCD1H
        LD      A,0C9H
        LD      (DEBUT),A
        RET
  
```

Le chargement du code RET à l'adresse de début de la routine de définition des instructions n'est pas obligatoire, mais il est vivement recommandé, car un deuxième appel à cette routine provoquerait un plantage du microprocesseur, et bien souvent l'obligation d'effectuer un RESET et de perdre toutes les données, pour reprendre la main.

Il faudra ensuite faire suivre cette routine des différents vecteurs et table des instructions, de la façon suivante :

VECTEU	DEFW	TABLE
	JP	INSTRUCTION1
	JP	INSTRUCTION2
		
	JP	INSTRUCTIONn
TABLE	DEFB	« INSTRUCTION1 moins la dernière lettre »
	DEFB	080H + « dernière lettre de l'instruction 1 »
		
	DEFB	« INSTRUCTIONn moins la dernière lettre »
	DEFB	080H + « dernière lettre de l'instruction n »
	DEFB	00H
KERNEL	DEFS	4

Nous trouvons dans cette partie de la routine à l'étiquette VECTEU la définition de l'adresse de la table des instructions, suivie de la table des sauts au traitement des instructions. Par exemple JP INSTRUCTION1, effectue un saut à la routine de traitement de l'instruction 1, ce traitement sera identique au traitement par l'instruction CALL.

Vient ensuite la table des instructions, qui doit être dans l'ordre de la table de saut des instructions. Dans cette table, les noms des instructions devront être obligatoirement écrits en MAJUSCULE, avec, de plus, le nombre hexadécimal &80 ajouté au dernier caractère, ceci pour que le moniteur reconnaisse la fin de l'instruction.

On trouve en tout dernier la réservation de quatre octets pour la routine.

Il sera possible d'installer, après cette routine les différents traitements prévus.

Prenons un exemple simple : notre première instruction consistera à inscrire le nom de ùWEKA ou IWEKA, et à obtenir un message. Pour ceux qui ne posséderaient pas encore d'assembleur, nous vous donnerons le chargeur Basic, ainsi que toutes les explications pour modifier cette routine.

Le programme de traitement de cette instruction ne sera pas très différent du programme des premiers pas effectués avec l'instruction CALL, si ce n'est la définition de cette instruction. Aussi, nous vous donnons ci-dessous, directement, le listing assembleur :

```

1          ; *****
2          ; * REALISATION D'UNE INSTRUCTION *
3          ; *          PERSONNELLE          *
4          ; * UNOM --> MESSAGE PERSONNEL   *
5          ; *                               *
6          ; * MODIFIABLE FACILEMENT        *
7          ; * PAR SON UTILISATEUR          *
8          ; *****
9          ;
10         ;
11         ; **      ORIGINE D'ASSEMBLAGE      **
12         ;
13         ;          ORG 0A000H

```

```

14      ;
15      ;
16      ;**  ADRESSE DE CHARGEMENT DU  **
17      ;**          CODE MACHINE      **
18      ;
19      LOAD 0A000H
20      ;
21      ;
22      ;*****
23      ;*  INSTALLATION DE L'INSTRUCTION *
24      ;*  SOUS LA FORME D'UNE R.S.X.   *
25      ;*****
26      ;
27  A000 2119A0  DEBUT:      LD  HL,KERNAL      ;POINTE 4 OCTET
28      ;UTILISES PAR LA ROUTINE MONITEUR
29  A003 010FA0      LD  SC,VECTEU      ;POINTE LA TABLE DES
30      ;INSTRUCTIONS
31  A006 CDD1BC      CALL OBCD1H      ;AJOUTE LA R.S.X.
32  A009 3EC9      LD  A,0C9H      ;CHARGE RET POUR
33  A00B 3200A0      LD  (DEBUT),A      ;INTERDIRE AUTRE APPE
34  A00E C9      RET      ;FIN INSTALLATION
35      ;
36  A00F 14A0  VECTEU:  DEFW TABLE      ;ADRESSE DE LA
37      ;TABLE DES INSTRUCTIONS
38  A011 C31DA0      JP  NOM      ;SAUT AU TRAITEMENT
39      ;DE L'INSTRUCTION ANOM
40      ;
41  A014 57454B  TABLE:  DEFB "WEK"      ;DEBUT DE LA
42      ;DEFINITION DE L'INSTRUCTION
43  A017 C1      DEFB 080H+"A"      ;DERNIERE LETTRE
44  A018 00      DEFB 0      ;FIN DE LA TABLE
45      ;
46      KERNAL:  DEFS 4      ;4 OCTETS POUR LA
47      ;ROUTINE KL-LOG-TEXTE
48      ;
49      ;*****
50      ;
51      ;
52      ;*****
53      ;*  TRAITEMENT DE L'INSTRUCTION *
54      ;*****
55      ;
56  A01D 212AA0  NOM:      LD  HL,MESSAG      ;POINTE
57      ;L'ADRESSE DE DEBUT DU
58      ;MESSAGE A AFFICHER
59  A020 7E      NOM1:      LD  A,(HL)      ;CHARGE
60      ;CODE CARACTERE DANS A
61      ;
62      ;
63      ;
64      ;
65      ;
66  A028 18F6      INC  HL      ;POINTE CARACTERE SUIVANT
67      ;
68      ;
69      ;
70      ;
71      ;
72      ;
73      ;
74      ;
75      ;
76      ;
77      ;
78      ;
79      ;
80      ;
81      ;
82      ;
83      ;
84      ;
85      ;
86      ;
87      ;
88      ;
89      ;
90      ;
91      ;
92      ;
93      ;
94      ;
95      ;
96      ;
97      ;
98      ;
99      ;
100     ;
101     ;
102     ;
103     ;
104     ;
105     ;
106     ;
107     ;
108     ;
109     ;
110     ;
111     ;
112     ;
113     ;
114     ;
115     ;
116     ;
117     ;
118     ;
119     ;
120     ;
121     ;
122     ;
123     ;
124     ;
125     ;
126     ;
127     ;
128     ;
129     ;
130     ;
131     ;
132     ;
133     ;
134     ;
135     ;
136     ;
137     ;
138     ;
139     ;
140     ;
141     ;
142     ;
143     ;
144     ;
145     ;
146     ;
147     ;
148     ;
149     ;
150     ;
151     ;
152     ;
153     ;
154     ;
155     ;
156     ;
157     ;
158     ;
159     ;
160     ;
161     ;
162     ;
163     ;
164     ;
165     ;
166     ;
167     ;
168     ;
169     ;
170     ;
171     ;
172     ;
173     ;
174     ;
175     ;
176     ;
177     ;
178     ;
179     ;
180     ;
181     ;
182     ;
183     ;
184     ;
185     ;
186     ;
187     ;
188     ;
189     ;
190     ;
191     ;
192     ;
193     ;
194     ;
195     ;
196     ;
197     ;
198     ;
199     ;
200     ;
201     ;
202     ;
203     ;
204     ;
205     ;
206     ;
207     ;
208     ;
209     ;
210     ;
211     ;
212     ;
213     ;
214     ;
215     ;
216     ;
217     ;
218     ;
219     ;
220     ;
221     ;
222     ;
223     ;
224     ;
225     ;
226     ;
227     ;
228     ;
229     ;
230     ;
231     ;
232     ;
233     ;
234     ;
235     ;
236     ;
237     ;
238     ;
239     ;
240     ;
241     ;
242     ;
243     ;
244     ;
245     ;
246     ;
247     ;
248     ;
249     ;
250     ;
251     ;
252     ;
253     ;
254     ;
255     ;
256     ;
257     ;
258     ;
259     ;
260     ;
261     ;
262     ;
263     ;
264     ;
265     ;
266     ;
267     ;
268     ;
269     ;
270     ;
271     ;
272     ;
273     ;
274     ;
275     ;
276     ;
277     ;
278     ;
279     ;
280     ;
281     ;
282     ;
283     ;
284     ;
285     ;
286     ;
287     ;
288     ;
289     ;
290     ;
291     ;
292     ;
293     ;
294     ;
295     ;
296     ;
297     ;
298     ;
299     ;
300     ;
301     ;
302     ;
303     ;
304     ;
305     ;
306     ;
307     ;
308     ;
309     ;
310     ;
311     ;
312     ;
313     ;
314     ;
315     ;
316     ;
317     ;
318     ;
319     ;
320     ;
321     ;
322     ;
323     ;
324     ;
325     ;
326     ;
327     ;
328     ;
329     ;
330     ;
331     ;
332     ;
333     ;
334     ;
335     ;
336     ;
337     ;
338     ;
339     ;
340     ;
341     ;
342     ;
343     ;
344     ;
345     ;
346     ;
347     ;
348     ;
349     ;
350     ;
351     ;
352     ;
353     ;
354     ;
355     ;
356     ;
357     ;
358     ;
359     ;
360     ;
361     ;
362     ;
363     ;
364     ;
365     ;
366     ;
367     ;
368     ;
369     ;
370     ;
371     ;
372     ;
373     ;
374     ;
375     ;
376     ;
377     ;
378     ;
379     ;
380     ;
381     ;
382     ;
383     ;
384     ;
385     ;
386     ;
387     ;
388     ;
389     ;
390     ;
391     ;
392     ;
393     ;
394     ;
395     ;
396     ;
397     ;
398     ;
399     ;
400     ;
401     ;
402     ;
403     ;
404     ;
405     ;
406     ;
407     ;
408     ;
409     ;
410     ;
411     ;
412     ;
413     ;
414     ;
415     ;
416     ;
417     ;
418     ;
419     ;
420     ;
421     ;
422     ;
423     ;
424     ;
425     ;
426     ;
427     ;
428     ;
429     ;
430     ;
431     ;
432     ;
433     ;
434     ;
435     ;
436     ;
437     ;
438     ;
439     ;
440     ;
441     ;
442     ;
443     ;
444     ;
445     ;
446     ;
447     ;
448     ;
449     ;
450     ;
451     ;
452     ;
453     ;
454     ;
455     ;
456     ;
457     ;
458     ;
459     ;
460     ;
461     ;
462     ;
463     ;
464     ;
465     ;
466     ;
467     ;
468     ;
469     ;
470     ;
471     ;
472     ;
473     ;
474     ;
475     ;
476     ;
477     ;
478     ;
479     ;
480     ;
481     ;
482     ;
483     ;
484     ;
485     ;
486     ;
487     ;
488     ;
489     ;
490     ;
491     ;
492     ;
493     ;
494     ;
495     ;
496     ;
497     ;
498     ;
499     ;
500     ;
501     ;
502     ;
503     ;
504     ;
505     ;
506     ;
507     ;
508     ;
509     ;
510     ;
511     ;
512     ;
513     ;
514     ;
515     ;
516     ;
517     ;
518     ;
519     ;
520     ;
521     ;
522     ;
523     ;
524     ;
525     ;
526     ;
527     ;
528     ;
529     ;
530     ;
531     ;
532     ;
533     ;
534     ;
535     ;
536     ;
537     ;
538     ;
539     ;
540     ;
541     ;
542     ;
543     ;
544     ;
545     ;
546     ;
547     ;
548     ;
549     ;
550     ;
551     ;
552     ;
553     ;
554     ;
555     ;
556     ;
557     ;
558     ;
559     ;
560     ;
561     ;
562     ;
563     ;
564     ;
565     ;
566     ;
567     ;
568     ;
569     ;
570     ;
571     ;
572     ;
573     ;
574     ;
575     ;
576     ;
577     ;
578     ;
579     ;
580     ;
581     ;
582     ;
583     ;
584     ;
585     ;
586     ;
587     ;
588     ;
589     ;
590     ;
591     ;
592     ;
593     ;
594     ;
595     ;
596     ;
597     ;
598     ;
599     ;
600     ;
601     ;
602     ;
603     ;
604     ;
605     ;
606     ;
607     ;
608     ;
609     ;
610     ;
611     ;
612     ;
613     ;
614     ;
615     ;
616     ;
617     ;
618     ;
619     ;
620     ;
621     ;
622     ;
623     ;
624     ;
625     ;
626     ;
627     ;
628     ;
629     ;
630     ;
631     ;
632     ;
633     ;
634     ;
635     ;
636     ;
637     ;
638     ;
639     ;
640     ;
641     ;
642     ;
643     ;
644     ;
645     ;
646     ;
647     ;
648     ;
649     ;
650     ;
651     ;
652     ;
653     ;
654     ;
655     ;
656     ;
657     ;
658     ;
659     ;
660     ;
661     ;
662     ;
663     ;
664     ;
665     ;
666     ;
667     ;
668     ;
669     ;
670     ;
671     ;
672     ;
673     ;
674     ;
675     ;
676     ;
677     ;
678     ;
679     ;
680     ;
681     ;
682     ;
683     ;
684     ;
685     ;
686     ;
687     ;
688     ;
689     ;
690     ;
691     ;
692     ;
693     ;
694     ;
695     ;
696     ;
697     ;
698     ;
699     ;
700     ;
701     ;
702     ;
703     ;
704     ;
705     ;
706     ;
707     ;
708     ;
709     ;
710     ;
711     ;
712     ;
713     ;
714     ;
715     ;
716     ;
717     ;
718     ;
719     ;
720     ;
721     ;
722     ;
723     ;
724     ;
725     ;
726     ;
727     ;
728     ;
729     ;
730     ;
731     ;
732     ;
733     ;
734     ;
735     ;
736     ;
737     ;
738     ;
739     ;
740     ;
741     ;
742     ;
743     ;
744     ;
745     ;
746     ;
747     ;
748     ;
749     ;
750     ;
751     ;
752     ;
753     ;
754     ;
755     ;
756     ;
757     ;
758     ;
759     ;
760     ;
761     ;
762     ;
763     ;
764     ;
765     ;
766     ;
767     ;
768     ;
769     ;
770     ;
771     ;
772     ;
773     ;
774     ;
775     ;
776     ;
777     ;
778     ;
779     ;
780     ;
781     ;
782     ;
783     ;
784     ;
785     ;
786     ;
787     ;
788     ;
789     ;
790     ;
791     ;
792     ;
793     ;
794     ;
795     ;
796     ;
797     ;
798     ;
799     ;
800     ;
801     ;
802     ;
803     ;
804     ;
805     ;
806     ;
807     ;
808     ;
809     ;
810     ;
811     ;
812     ;
813     ;
814     ;
815     ;
816     ;
817     ;
818     ;
819     ;
820     ;
821     ;
822     ;
823     ;
824     ;
825     ;
826     ;
827     ;
828     ;
829     ;
830     ;
831     ;
832     ;
833     ;
834     ;
835     ;
836     ;
837     ;
838     ;
839     ;
840     ;
841     ;
842     ;
843     ;
844     ;
845     ;
846     ;
847     ;
848     ;
849     ;
850     ;
851     ;
852     ;
853     ;
854     ;
855     ;
856     ;
857     ;
858     ;
859     ;
860     ;
861     ;
862     ;
863     ;
864     ;
865     ;
866     ;
867     ;
868     ;
869     ;
870     ;
871     ;
872     ;
873     ;
874     ;
875     ;
876     ;
877     ;
878     ;
879     ;
880     ;
881     ;
882     ;
883     ;
884     ;
885     ;
886     ;
887     ;
888     ;
889     ;
890     ;
891     ;
892     ;
893     ;
894     ;
895     ;
896     ;
897     ;
898     ;
899     ;
900     ;
901     ;
902     ;
903     ;
904     ;
905     ;
906     ;
907     ;
908     ;
909     ;
910     ;
911     ;
912     ;
913     ;
914     ;
915     ;
916     ;
917     ;
918     ;
919     ;
920     ;
921     ;
922     ;
923     ;
924     ;
925     ;
926     ;
927     ;
928     ;
929     ;
930     ;
931     ;
932     ;
933     ;
934     ;
935     ;
936     ;
937     ;
938     ;
939     ;
940     ;
941     ;
942     ;
943     ;
944     ;
945     ;
946     ;
947     ;
948     ;
949     ;
950     ;
951     ;
952     ;
953     ;
954     ;
955     ;
956     ;
957     ;
958     ;
959     ;
960     ;
961     ;
962     ;
963     ;
964     ;
965     ;
966     ;
967     ;
968     ;
969     ;
970     ;
971     ;
972     ;
973     ;
974     ;
975     ;
976     ;
977     ;
978     ;
979     ;
980     ;
981     ;
982     ;
983     ;
984     ;
985     ;
986     ;
987     ;
988     ;
989     ;
990     ;
991     ;
992     ;
993     ;
994     ;
995     ;
996     ;
997     ;
998     ;
999     ;
1000    ;

```

```

74                                     ;
75 A02A 0402      MESSAG:      DEFB 04H,02H      ; ECRAN MODE 2
76 A02C 1C001A1A      DEFB 01CH,00H,1AH,1AH      ; ENCRE
77                                     ; NO 0 EN BLANC
78 A030 0E00      DEFB 0EH,00H      ; COULEUR FOND =
79                                     ; ENCRE 0
80 A032 1C010000      DEFB 01CH,01H,00H,00H      ; ENCRE
81                                     ; NO 1 EN NOIR
82 A036 0F01      DEFB 0FH,01H      ; COULEUR PEN 1
83 A038 1F130A      DEFB 01FH,013H,0AH      ; POSITION
84 A03B 436F6D6D      DEFB "Comment ameliorer
84 A03F 656E7420
84 A043 616D656C
84 A047 696F7265
84 A04B 72206574
85 A04F 20617567      DEFB " augmenter"
85 A053 6D656E74
85 A057 6572
86 A059 1F160B      DEFB 01FH,016H,0BH      ; POSITION
87 A05C 6C657320      DEFB "les performances
87 A060 70657266
87 A064 6F726D61
87 A068 6E636573
87 A06C 20646520
87 A070 766F7472
87 A074 65
88 A075 1F1F0D      DEFB 01FH,01FH,0DH      ; POSITION
89 A07B 414D5354      DEFB "AMSTRAD-CPC"
89 A07C 5241442D
89 A080 435043
90 A0B3 FF      DEFB 0FFH
91                                     ;
92                                     ;
93                                     END

```

À l'adresse A014, nous trouvons les trois premières lettres de l'instruction WEK, puis, la lettre A à laquelle on ajoute &80 à l'adresse A017.

Le programme de traitement est identique à tous les programmes d'affichage de chaîne de caractères et de caractères de contrôle vus plus haut.

Après sauvegarde et assemblage, on procède à l'initialisation de l'instruction par les commandes :

```

MEMORY &9FFF
CALL &A000

```

Vous trouverez ci-dessous le listing Basic du chargeur de cette instruction :

```

10 REM *****
20 REM * CREATION D'UNE INSTRUCTION *
30 REM * AFFICHANT UN MESSAGE *
40 REM * PERSONNEL GRACE A LA RSX *
50 REM * &NOM *
60 REM *****
70 REM
80 REM *** CHARGEUR BASIC ***
90 ADR = &A000: I = 0
100 RESTORE 320
110 READ A$
120 IF A$ = "XX" THEN 200
130 B$ = "&" + A$
140 B = VAL(B$)
150 POKE ADR,B
160 ADR = ADR + 1
170 I = I + 1
180 GOTO 110
190 REM
200 REM *** SAUVEGARDE ***
210 PRINT "POUR SAUVEGARDER LA ROUTINE E
N BINAIRE"
220 PRINT "SAVE ";CHR$(34);"ROUTINE.BIN"
;CHR$(34);"",B,&A000,"";RIGHT$(STR$(I),LEN
(STR$(I))-1)
230 PRINT
240 PRINT "CHARGEMENT PAR LOAD ";CHR$(34
);"ROUTINE.BIN";CHR$(34);"",&A000"
250 PRINT "MEMORY &9FFF"
260 PRINT
270 PRINT "INSTALLATION PAR CALL &A000,V
ALEUR"
280 PRINT "PUIS EXECUTION PAR &nom (ici,
nom = WEKA)"
290 STOP
300 REM
310 REM *** ROUTINE L.M. ***
320 DATA 21,26,A0,01,0F,A0,CD,D1
330 DATA BC,3E,C9,32,00,A0,C9
340 REM
350 REM *** VECTEUR ***
360 DATA 14,A0,C3,2A,A0
370 REM
380 REM *** NOM INSTRUCTION ***
390 DATA 57,45,4B,C1,00,00,00,00
400 DATA 00,00,00,00,00,00,00,00
410 DATA 00,00,00,00,00,00,00,00
420 REM
430 REM *** FIN DE TABLE ***

```

```
440 DATA 00
450 REM
460 REM *** ROUTINE MACHINE ***
470 DATA 21,37,A0,7E,23,FE,FF,CB
480 DATA CD,5A,BB,18,F6
490 REM
500 REM *** MESSAGE ***
510 DATA 04,02,1C,00,1A,1A,0E,00
520 DATA 1C,01,00,00,0F,01,1F,13
530 DATA 0A,43,6F,6D,6D,65,6E,74
540 DATA 20,61,6D,65,6C,69,6F,72
550 DATA 65,72,20,65,74,20,61,75
560 DATA 67,6D,65,6E,74,65,72,1F
570 DATA 16,0B,6C,65,73,20,70,65
580 DATA 72,66,6F,72,6D,61,6E,63
590 DATA 65,73,20,64,65,20,76,6F
600 DATA 74,72,65,1F,1F,0D,41,4D
610 DATA 53,54,52,41,44,2D,43,50
620 DATA 43
630 REM
640 REM *** FIN DE MESSAGE ***
650 DATA FF
660 REM
670 REM
680 REM *** FIN DE DATAs ***
690 DATA XX
```

Une fois sauvegardé, le programme sera exécuté par RUN, puis installé par :

```
MEMORY &9FFF
CALL &A000
```

Il sera ensuite possible d'effacer le chargeur Basic par NEW, et d'appeler l'instruction ùWEKA (ou IWEKA).

Ceux d'entre vous qui possèdent un assembleur n'auront aucun mal pour adapter l'instruction à leur nom par exemple, par contre voici les explications pour la modifier dans le chargeur Basic :

En lignes 390 à 410, il est réservé 21 octets pour le nom de l'instruction. Il vous suffira de modifier les quatre premiers octets, puis de remplacer les 00 par les codes ASCII de noms que vous désirez inscrire. Vous obtiendrez les codes ASCII du nom grâce à la table de conversion donnée plus haut, il vous suffira de frapper la commande :

```
PRINT HEX$ (&dernieroctet + &80)
```

pour obtenir la valeur du dernier caractère du nom à inscrire.

Attention :

Le nombre d'octets de ces trois lignes doit impérativement être égal à 21 (les zéros compris).

Vous pourrez modifier ensuite le message entre les lignes 510 et 620. Vous pourrez cette fois-ci inscrire autant d'octets ASCII que vous désirez à la condition que les deux dernières valeurs en DATA (lignes 650 et 690) soient, dans l'ordre FF et XX.

4/1.6.5 DISQUETTE

Formater une disquette sous Basic

Nous vous proposons dans ce chapitre d'utiliser les connaissances que vous avez acquises sur la programmation de nouvelles instructions supplémentaires au Basic, de type RSX (voir Partie 4, chapitre 1.6.2 p. 1 à 35) pour écrire une instruction de formatage. Nous vous conseillons par ailleurs de charger cette nouvelle instruction chaque fois que vous mettrez au point, ou que vous commencerez à entrer un programme Basic.

Beaucoup d'entre vous ont certainement rencontré le problème qui sera résolu par cette nouvelle commande. En effet, il arrive que dans la mise au point, ou l'amélioration d'un programme, celui-ci, de part le nombre de lignes ajoutées, ne permette plus la sauvegarde sur la disquette d'origine (n'oublions pas le **.BAK** qui est créé à chaque sauvegarde d'un fichier déjà existant), signalée par l'horrible message **Disk Full**. A ce moment arrive l'angoisse pour celui (nous en fîmes partie) qui, dans l'empressement n'a pas prévu de disquette préalablement formatée.

Vous aurez beau avoir à votre disposition autant de disquettes vierges nouvellement achetées, aucun recours possible avec celles-ci.

La seule solution est d'utiliser la disquette du système d'exploitation CP/M (CP/M 2.2 ou CP/M 3.0), et de lancer l'un des utilitaires **FORMAT.COM**, **DISCKIT2.COM** ou **DISCKIT3.COM**.

Le résultat : des dizaines, voir des centaines de lignes de programmes laborieusement entrées au clavier qui s'envolent parmi la multitude d'octets de la mémoire.

Nous avons donc réalisé pour vous une nouvelle instruction résidente, que nous appellerons **FORMAT**, dont nous étudierons la syntaxe exacte ultérieurement, et qui vous permettra, de plus, d'améliorer vos connaissances sur le fonctionnement logiciel d'un formatage sur votre **AMSTRAD-CPC**.

Les débutants en langage machine pourront parfaire leurs connaissances en étudiant l'algorithme et l'algorithme décrit, trouveront des utilisations possibles d'un nouveau vecteur du système d'exploitation ; les passionnés du code hexadécimal y trouveront certainement de quoi personnaliser leurs disquettes en ne formatant par exemple qu'une piste unique.

1. Rappel sur les disquettes et les différents formats

Précisons d'abord, afin d'utiliser le même vocabulaire, que nous appellerons **FORMAT** le terme en relation directe avec le formatage, et **TAILLE** le mot en relation avec les dimensions physiques de la disquette (3 pouces, 3 pouces et 1/2, 5 pouces 1/4).

DISQUETTE - PISTE - SECTEUR

Sans nous étendre sur l'organisation de la surface de la disquette, rappelons que lorsque vous achetez une disquette, celle-ci est entièrement vierge de toute donnée, et elle n'est pas, hormis sa taille (3 pouces dans le cas des disquettes pour AMSTRAD-CPC), prévue pour fonctionner sur un seul type de micro-ordinateur. La disquette 3 pouces faisant exception à la règle, car la société Amstrad, espérant vainement imposer un standard, est quasiment la seule à utiliser cette dimension.

C'est l'ordinateur et son lecteur de disquettes qui va inscrire et organiser, lors du formatage, les données nécessaires à son utilisation, sur le support magnétique. Chaque type d'ordinateur aura donc un formatage différent.

Dans le cas de l'AMSTRAD-CPC, et dans pratiquement tous les micro-ordinateurs travaillant encore avec le système d'exploitation CP/M, l'organisation de la disquette est représentée sur la figure 1.

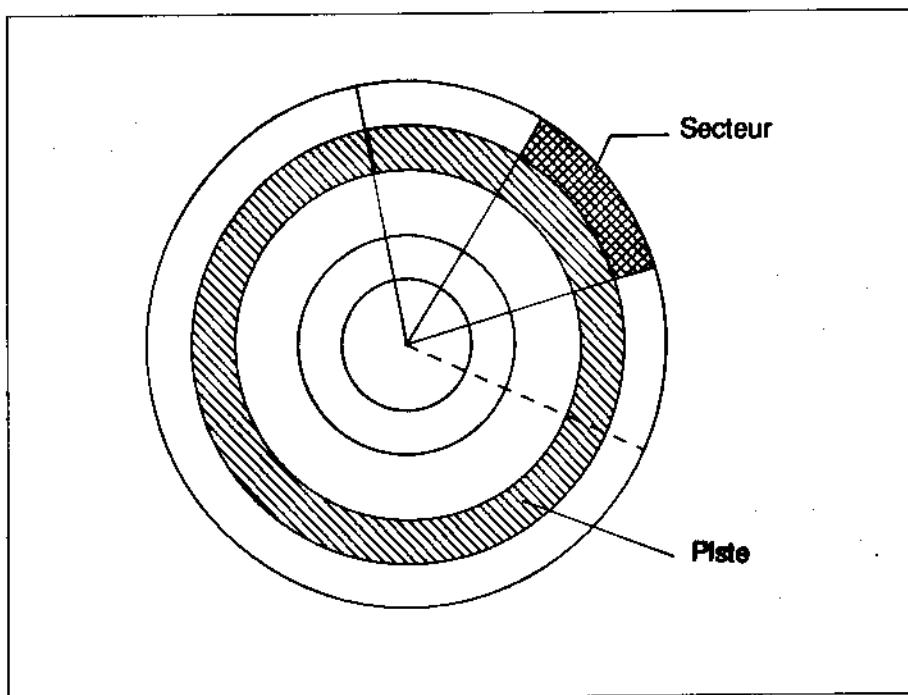


Fig. 1 : Organisation de la surface d'une disquette.

Le nombre de pistes sous CP/M est dans notre cas limité à 40 par face, qui sont numérotées de 0 à 39, la piste 0 se trouvant sur l'extérieur de la surface, et la piste 39 à l'intérieur.

Chacune des pistes est découpée en portions, tel un camembert, appelées secteurs (huit ou neuf secteurs selon le format), chaque secteur pouvant supporter 512 octets de données. Ces secteurs sont numérotés de façons différentes selon les formats.

Dans certains logiciels ou certaines revues, vous trouverez les appellations anglo-saxonnes : **TRACK** pour PISTE et **SECTOR** pour SECTEUR.

LES FORMATS

Selon l'utilisation requise, il vous est possible de formater vos disquettes sous quatre formats différents : IBM, SYSTEME CP/M, VENDOR ou DATA:

Le format IBM

Ce format qui n'est quasiment pas utilisé sur Amstrad CPC.

Il travaille sur 40 pistes de 8 secteurs chacune, numérotées de 01 à 08.

La première piste est réservée au système d'exploitation.

Chaque secteur peut comporter 512 octets de données.

Le format SYSTEME

Le format SYSTEME est le format standard des disquettes sur AMSTRAD-CPC, et sur PCW, car c'est celui prévu pour travailler sous CP/M, et permettant de travailler avec des fichiers sous AMSDOS, donc en Basic.

Les caractéristiques de ce format sont :

- 40 pistes par face numérotées de 0 à 39 ;
- 9 secteurs par piste numérotés de &41 (hexadécimal) à &49 ;
- taille d'un secteur : 512 octets de données ;
- 64 fichiers au maximum au catalogue ;
- deux pistes réservées au système : les pistes 0 et 1 :

Piste 0 secteur &41 : secteur BOOT (démarrage) ;

Piste 0 secteur &42 : secteur configuration ;

Piste 0 secteur &43 à &47 : non utilisés ;

Piste 0 secteur &48 et &49 ainsi que toute la Piste 1 sont réservés à CCP (Console Command Processeur) et BDOS (Basic Disk Operating System).

Ce format permet de démarrer le CP/M 2.2 par la commande `ùCPM`. Cette même commande démarre aussi CP/M 3.0 (ou CP/M+) mais avec un secteur BOOT différent et l'utilitaire `C10CPM3.EMS`.

La capacité disponible se calcule facilement par : $(40 \text{ pistes} - 2 \text{ pistes réservées}) \times 9 \text{ secteurs} \times 512 \text{ octets} - 2 \times 1024 \text{ octets au catalogue} = 173056 \text{ octets} = 169 \text{ k-octets}$.

Le format VENDOR

Le format VENDOR correspond à une préparation identique au format SYSTEME avec la seule particularité que les pistes réservées ne contiennent pas le système CP/M.

En effet, le système d'exploitation **CP/M** est la propriété de la société DIGITAL RESEARCH, il est donc interdit par la loi de le distribuer ou de le vendre, sans payer des droits, ce qui augmenterait le prix du logiciel sur la disquette.

Tout utilisateur de logiciel fonctionnant sous CP/M possède ce système d'exploitation, qu'il est censé avoir acheté. Il lui sera ainsi facile de l'installer à l'aide de l'utilitaire SYSGEN.COM qui lui a été livré.

Le format DATA

Ce format est couramment employé par les programmeurs n'utilisant pas CP/M sur leurs disquettes, car il permet de récupérer la place des deux premières pistes.

Les caractéristiques en sont :

- 40 pistes par faces numérotées de 0 à 39 ;
- 9 secteurs par pistes numérotés de &C1 à &C9 ;
- taille d'un secteur : 512 octets ;
- 64 fichiers au catalogue ;
- pas de pistes réservées, ce qui permet d'obtenir : $2 * 9 * 512 = 9216$ octets = 9 k-octets supplémentaires pour obtenir une capacité formatée $169 + 9 = 178$ k-octets par face.

k-octets par face.

Sous ce format il est hélas impossible de ré-implanter CP/M sauf par un nouveau formatage.

OÙ SONT LES DONNÉES SUR LA DISQUETTE ?

La disquette est découpée en secteurs et en pistes, lors du formatage, afin que votre CPC retrouve les données cherchées grâce au marquage de leur emplacement.

Les fichiers ou les programmes seront ainsi découpés en blocs de 512 octets, enregistrés dans les secteurs.

Mais tout ceci n'est pas aussi simple que cela : une piste n'est pas uniquement constituée de $9 * 512 = 4608$ octets, mais de beaucoup plus, afin de pouvoir les relire et les inscrire avec fiabilité.

Voyons comment est préparé le contenu de la disquette lors du formatage, pour vous donner une idée de l'ampleur du travail effectué par le composant supervisant le tout : le μ PD 765 AC (les mordus d'électronique trouveront les affectations de ses différentes broches en Partie 2 chapitre 3.5 — notez que son appellation la plus courante est FDC 765 pour *Floppy Disk Controller*).

Le numéro de la piste est connu grâce à l'utilisation d'un moteur pas à pas d'une très grande précision.

Une fois sur la piste qu'il a choisie, le μ PD 765 AC doit d'abord être prévenu du début d'une piste par la détection du signal donné par une cel-

lule au passage d'un trou sur la disquette : le trou d'INDEX (vous pouvez voir apparaître ce trou en faisant tourner la disquette avec les doigts dans les perforations du plastique enveloppant le support magnétique).

Au début de la piste signalée, 80 octets blancs sont inscrits pour laisser à l'électronique et la mécanique le temps de réagir. On appelle ces octets des octets de GAP (jeu, espace).

Ensuite, 12 octets permettant de synchroniser le μ PD 765 AC sont inscrits ainsi que 3 + 1 autres octets de repérage de la piste, puis viennent 50 autres octets de GAP.

Nous avons donc là : $80 + 12 + 4 + 50 = 146$ octets figés pour le début de la piste.

Viennent ensuite les secteurs.

Chaque secteur sera composé, dans l'ordre, par :

- 12 octets de synchronisation ;
- 3 + 1 octets de repérage du secteur ;
- 4 octets d'identification, dont le premier indique le numéro de piste, le deuxième le numéro de tête, le troisième le numéro de secteur, le quatrième la taille du secteur ;
- 2 octets permettant de contrôler la validité des données précédentes par leur somme (appelée en langage informatique CHECKSUM) ;
- 22 octets de GAP ;
- 12 octets de synchronisation ;
- 3 + 1 octets de repérage des données ;
- les 512 tant attendus octets de données (lors du formatage, contiendront tous la valeur &E5) ;
- 2 octets de contrôle ;
- 54 octets de GAP pour un éventuel débordement des données lors d'une prochaine procédure d'écriture.

Vous retrouverez en figure 2 la composition d'une piste de la disquette.

Vous vous apercevrez ainsi que sur une disquette au format DATA, il y a ainsi :

$40 \cdot (146 + 9 \cdot 628) = 231920$ octets, soit plus de 226 kilo-octets qui y sont inscrits.

ACCÈS À L'INSTRUCTION 408

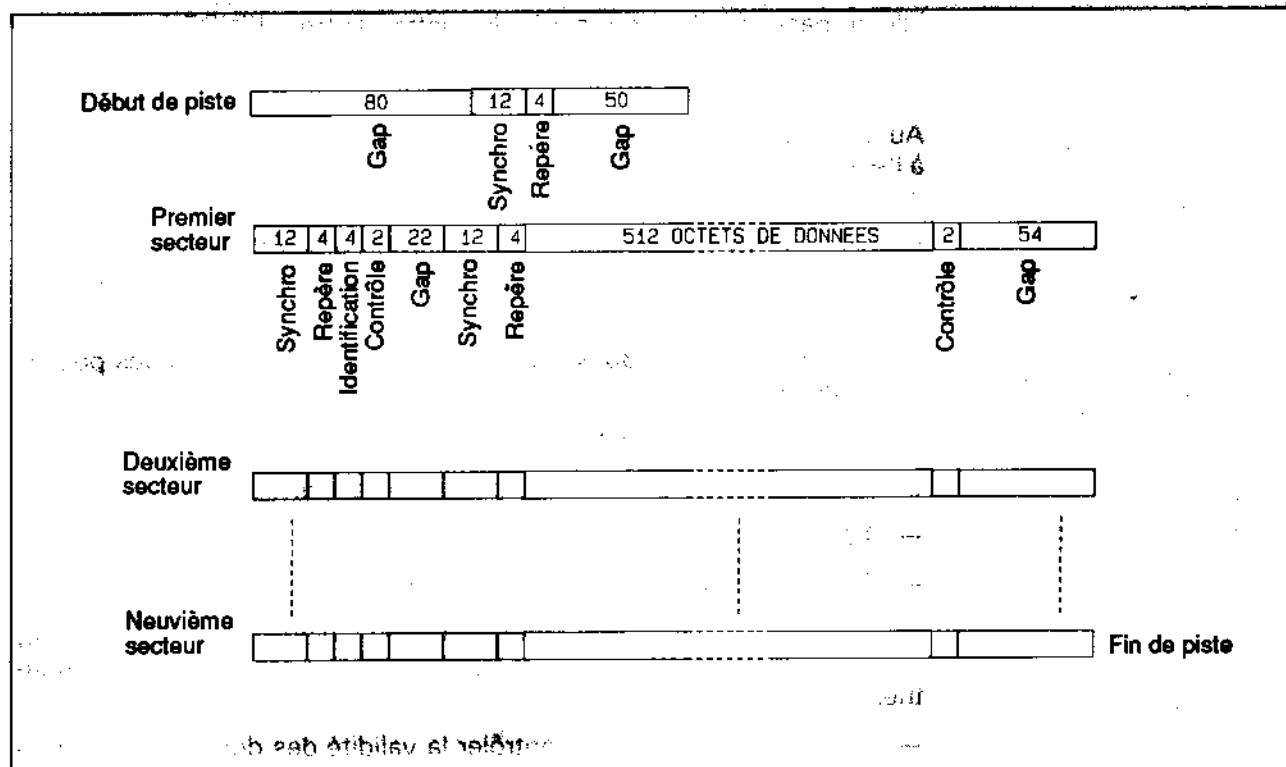


Fig. 2 : Organisation des informations logiques placées sur une piste de disquette 3 pouces Amstrad.

II. La logique du formatage et l'instruction `FORMAT`

Nous allons ici découvrir que pour réaliser nous-même, notre programme de formatage, l'introduction de tous les octets précédemment décrits ne sera pas nécessaire. Seuls les 4 octets d'identification seront à fournir. Il nous faudra ainsi une petite connaissance de la ROM sous l'AMSDOS.

Ceux d'entre-vous, passionnés de langage machine, ont certainement découvert, en décortiquant le programme Partie 9, chapitre 8.10, la possibilité d'utiliser certaines instructions cachées de la ROM AMSDOS, pour lire ou écrire sur un secteur particulier de la disquette.

Il existe de plus une instruction permettant de formater une piste en particulier, que nous utiliserons donc.

ACCÈS À L'INSTRUCTION `&06`

Cette instruction est nommée par un numéro, mais n'est malheureusement pas accessible directement depuis le Basic.

On peut en trouver la définition dans le listing de la ROM à l'adresse `C0BC` (dans les premières versions de ROM, il est possible qu'elle soit diffé-

rente sur certains AMSTRAD-CPC) ; nous vous donnons d'ailleurs le fruit de nos recherches entre les adresses et &C0B6 et &C0BF, par un desassemblage de cette zone :

C0B6	81	DEFB	01H + 80H ; INSTRUCTION 01
C0B7	82	DEFB	02H + 80H ; INSTRUCTION 02
C0B8	83	DEFB	03H + 80H ; INSTRUCTION 03
C0B9	84	DEFB	04H + 80H ; INSTRUCTION 04
C0BA	85	DEFB	05H + 80H ; INSTRUCTION 05
C0BB	86	DEFB	06H + 80H ; INSTRUCTION 06
C0BC	87	DEFB	07H + 80H ; INSTRUCTION 07
C0BD	88	DEFB	08H + 80H ; INSTRUCTION 08
C0BE	89	DEFB	09H + 80H ; INSTRUCTION 09
C0BF	00	DEFB	00H FIN DE TABLE

Vous pouvez ainsi remarquer, en vous référant Partie 4, chapitre 1.6.2, que sont définies ici 9 instructions numérotées de 01 à 09, les utilisateurs chevronnés de celles-ci prennent plutôt l'habitude de les nommer de &81 à &89, de par leur particularité.

Une telle définition implique bien sûr une table de saut, voici celle inscrite dans notre ROM aux adresses C033 à C04D :

C033	C3	72	CA	JP	0CA72H	MESSAGES ON/OFF (&01)
C036	C3	0D	C6	JP	0C60DH	PARAMETRE DISQUE (&02)
C039	C3	81	C5	JP	0C581H	PARAMETRE FORMAT (&03)
C03C	C3	66	C6	JP	0C666H	LECTURE SECTEUR (&04)
C03F	C3	4E	C6	JP	0C64EH	ECRITURE SECTEUR (&05)
C042	C3	52	C6	JP	0C652H	FORMATAGE PISTE (&06)
C045	C3	63	C7	JP	0C763H	CHERCHE PISTE (&07)
C048	C3	30	C6	JP	0C630H	DISQUE PRESENT ? (&08)
C04B	C3	03	C6	JP	0C603H	TENTATIVES LECTURE (&09)

On constate, dans cette table, que le saut pour l'instruction, que nous appellerons maintenant &86, s'effectue à l'adresse &C652. Seule une routine en langage machine permettra d'y accéder car elle exige une commutation de la ROM supérieure.

Mais même avec une commutation, cette adresse n'est pas à utiliser telle quelle, car telle ou telle autre version de ROM ne sera pas forcément identique, donc ne permettra pas une transposition automatique sur certains modèles de CPC, au cas où des améliorations de ROM auraient été effectuées.

Heureusement, il existe un vecteur en RAM, théoriquement immuable, qui permet d'accéder à une procédure déterminant l'adresse de n'importe quelle extension en ROM pour l'exécuter ultérieurement, il s'agit du vecteur KL-FIND-COMMAND à l'adresse &BCD4 (voir partie 4, chapitre 2.7 page 54).

Ce vecteur requiert un paramètre dans le registre double HL et qui contient l'adresse où se trouve définie l'instruction &86 (en RAM cette fois-ci).

Au retour, l'indicateur de retenue (ou *Flag Carry*) contient la valeur 1 si l'extension existe, le registre C le numéro de ROM où elle se trouve, et HL l'adresse effective.

Le programme correspondant se présentera ainsi :

```
LD      HL,ADR86
CALL    KL-FIND-COMMAND
... suite ...
```

Une fois connus le numéro de ROM, qui est normalement 7, et l'adresse, effectuons un saut à ces coordonnées.

La méthode évidente passe par l'utilisation des ports d'entrées sorties, donc du GATE ARRAY, pour effectuer une commutation de la ROM supérieure AMSDOS. Ce serait oublier la possibilité laissée par les concepteurs du moniteur, qui ont placé à notre disposition des vecteurs sous forme d'instruction dites "RESTART". Celle qui nous intéresse est dénommée couramment RST &18.

RST &18 est une interruption du microprocesseur Z80. Elle a été implantée dans le moniteur de l'AMSTRAD-CPC de façon à permettre d'appeler une routine en ROM ou en RAM. Derrière cette instruction doit se trouver l'adresse d'une zone mémoire en RAM qui contient dans l'ordre :

- l'octet bas de l'adresse en ROM ;
- l'octet haut de l'adresse en ROM ;
- le numéro de la ROM supérieure.

La forme du programme d'accès sera donc :

```
RST      018H
DEFW     SAUT
... suite du programme...
```

SAUT : octet bas de l'adresse
 octet haut de l'adresse
 numéro de ROM

Il suffira ainsi de ranger dans la zone de SAUT les paramètres remis suite à l'appel de KL-FIND-COMMAND.

Quels paramètres lui transmettre ?

Comme beaucoup de routines et vecteurs du CPC, l'instruction &86 doit connaître certains paramètres pour pouvoir être exécutée.

Ces paramètres sont :

- le numéro du lecteur dans le registre E ;
- le numéro de la piste à formater dans le registre D ;
- le numéro du premier secteur dans le registre C ;
- l'adresse d'une table nécessaire au formatage.

Dernière étape de notre étude : la composition de cette table de formatage. Dans cette table, se trouve, pour chaque secteur : les quatre octets nécessaires à leur identification, ceux qui portent le même nom en figure 2.

Dans l'ordre nous aurons :

- numéro de piste ;
- numéro de tête ;
- numéro du secteur ;
- taille du secteur ;

et ainsi de suite pour tous les secteurs.

Le numéro de piste sera celui de la piste dont le formatage est requis. Le numéro de tête est toujours 0 car le lecteur Amstrad ne possède qu'une tête. Le numéro du secteur sera, selon le formatage, l'un des numéros de &41 à &49 pour le format VENDOR ou SYSTEME, &C1 à &C9 pour le format DATA, &01 à &08 pour le format IBM (que nous passerons dorénavant sous silence !), et la taille du secteur sera toujours égale à 2 pour des secteurs de 512 octets.

Une ultime précision, par exemple pour le format DATA, la logique voudrait que le premier secteur soit numéroté &C1, le deuxième &C2, le troisième &C3, ... etc. En fait, supposons le cas d'une lecture, comme c'est souvent le cas, d'un fichier sauvegardé sur les secteurs &C1, puis &C2 et enfin &C3, la rapidité de traitement des informations ferait en sorte que le secteur &C1 serait géré bien avant que la disquette n'ait effectué à peine plus d'un quart de tour. Il a été convenu pour optimiser les performances que les secteurs seraient placés tous les demi-tours de la surface magnétique environ. L'ordre est ainsi :

&C1 - &C3 - &C5 - &C7 - &C9 - &C2 - &C4 - &C6 - &C8

ce qui donne pour le format SYSTEME ou VENDOR :

&41 - &43 - &45 - &47 - &49 - &42 - &44 - &46 - &48

La table de formatage pour une piste aura ainsi l'aspect suivant :

SECTEUR1:	DEFB	Nodepiste	; DE 0 à 028H
	DEFB	0	; TETE ZERO
	DEFB	&C1	; DANS LE CAS D'UN FORMAT DATA
	DEFB	2	; TAILLE POUR 512 OCTETS
SECTEUR2:	DEFB	Nodepiste	
	DEFB	0	
	DEFB	&C3	
	DEFB	2	
SECTEUR3:	DEFB	Nodepiste	
	DEFB	0	
	DEFB	&C5	
	DEFB	2	
SECTEUR4:	DEFB	Nodepiste	
	DEFB	0	
	DEFB	&C7	
	DEFB	2	
SECTEUR5:	DEFB	Nodepiste	
	DEFB	0	
	DEFB	&C9	
	DEFB	2	
SECTEUR6:	DEFB	Nodepiste	
	DEFB	0	
	DEFB	&C2	
	DEFB	2	
SECTEUR7:	DEFB	Nodepiste	
	DEFB	0	
	DEFB	&C4	
	DEFB	2	
SECTEUR8:	DEFB	Nodepiste	
	DEFB	0	
	DEFB	&C6	
	DEFB	2	
SECTEUR9:	DEFB	Nodepiste	
	DEFB	0	
	DEFB	&C8	
	DEFB	2	

L'INSTRUCTION BASIC ÛFORMAT

La syntaxe

L'instruction nouvelle que nous allons créer, aura pour syntaxe :

ûFORMAT, lecteur, "type"

où le caractère ù correspond au caractère "BARRE VERTICALE" code ASCII 124) sur certains modèles d'AMSTRAD-CPC et situé sur la même touche que le "a commercial" : @.

Le numéro de lecteur sera 0 pour le lecteur A et 1 pour le lecteur B.

Le type, à placer obligatoirement entre guillemets pourra être **VENDOR** ou **DATA**, ou tout autre nom commençant par les deux lettres **V** ou **A**. Nous vous conseillons les quatre utilisations suivantes :

- `ùFORMAT,0,"V"` ou `ùFORMAT,0,"v"` pour formater sur le lecteur A une disquette au format **VENDOR** ;
- `ùFORMAT,0,"D"` ou `ùFORMAT,0,"d"` pour formater sur le lecteur A une disquette au format **DATA** ;
- `ùFORMAT,1,"V"` ou `ùFORMAT,0,"v"` pour formater sur le lecteur B une disquette au format **VENDOR** ;
- `ùFORMAT,1,"D"` ou `ùFORMAT,0,"d"` pour formater sur le lecteur B une disquette au format **DATA**.

Nous nous sommes imposés de demander confirmation à l'utilisateur pour démarrer le formatage, celle-ci n'étant acceptée que si la réponse est **O** majuscule ou **o** minuscule pour **OUI** ; toute autre réponse renvoyant au Basic.

La confirmation passée, le logiciel attend automatiquement une disquette dans le lecteur, si elle ne s'y trouve déjà. Mais, si par hasard il n'y en avait pas, vous serez prié d'en placer une par le message :

Drive X: Disc missing

Retry, Ignore or Cancel ?

et la seule alternative serait de placer une disquette puis d'appuyer sur la touche **R**, vous aviez bien confirmé ?!

L'algorithme

Nous allons à partir de maintenant, réinvestir **toutes les connaissances** acquises précédemment sur le formatage d'une piste, pour formater les 40 pistes d'une face de disquette, et ce en utilisant une démarche algorithmique pour arriver au programme.

Voici, présenté dans ses grandes lignes, l'algorithme général du programme :

— DEBUT

- Vérifier la validité du secteur (0 ou 1)
et sauvegarder

CO

le cas échéant retourner au BASIC

FINCO

- Vérifier la validité du type de formatage
et sauvegarder

CO

le cas échéant retourner au BASIC

FINCO

- Demander confirmation du formatage

CO

dans la négative retourner au BASIC

FINCO

- Signaler le formatage

- REPETER

— REPETER

- Préparer la table de formatage

- JUSQU'A ce que les 9 secteurs soient prêts

- Ordonner le formatage de la piste

- JUSQU'A ce que la piste 39 soit atteinte

- FIN

Vous pouvez déjà apercevoir les principales boucles dans les termes REPETER ... JUSQU'A qui laissent entrevoir que la table des secteurs sera créée par une structure itérative (donc par le calcul), ainsi que le formatage de toutes les pistes. En effet, il aurait été inconcevable d'écrire et d'entrer entièrement manuellement toute la table de formatage, ceci nous aurait coûté $9 * 4 * 40 = 1440$ octets supplémentaires !

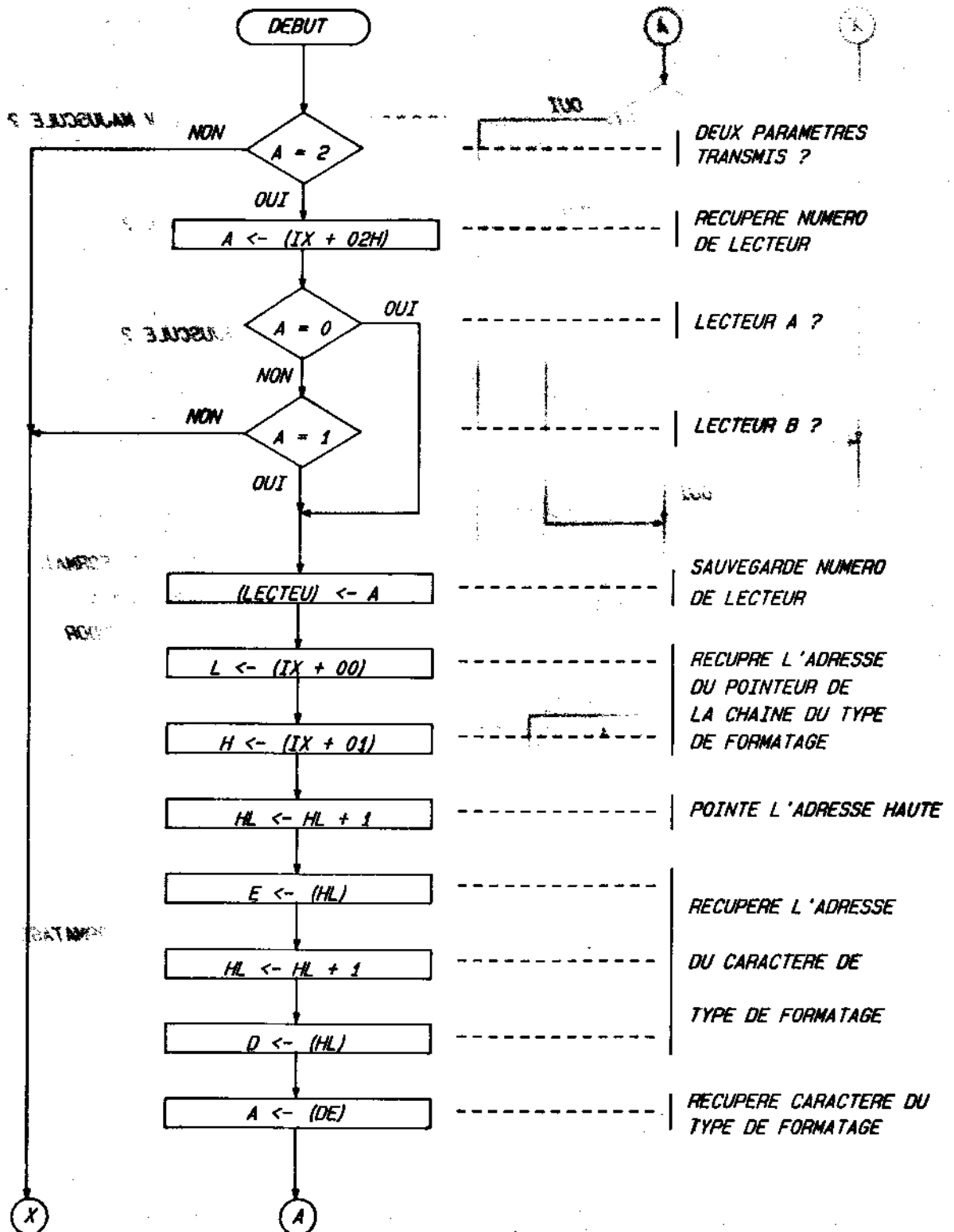
Nous ne détaillerons pas davantage l'algorithme dans ce chapitre, mais nous vous conseillons de le faire en vous aidant entre autre de l'ordinogramme que nous avons ci-dessous décrit.

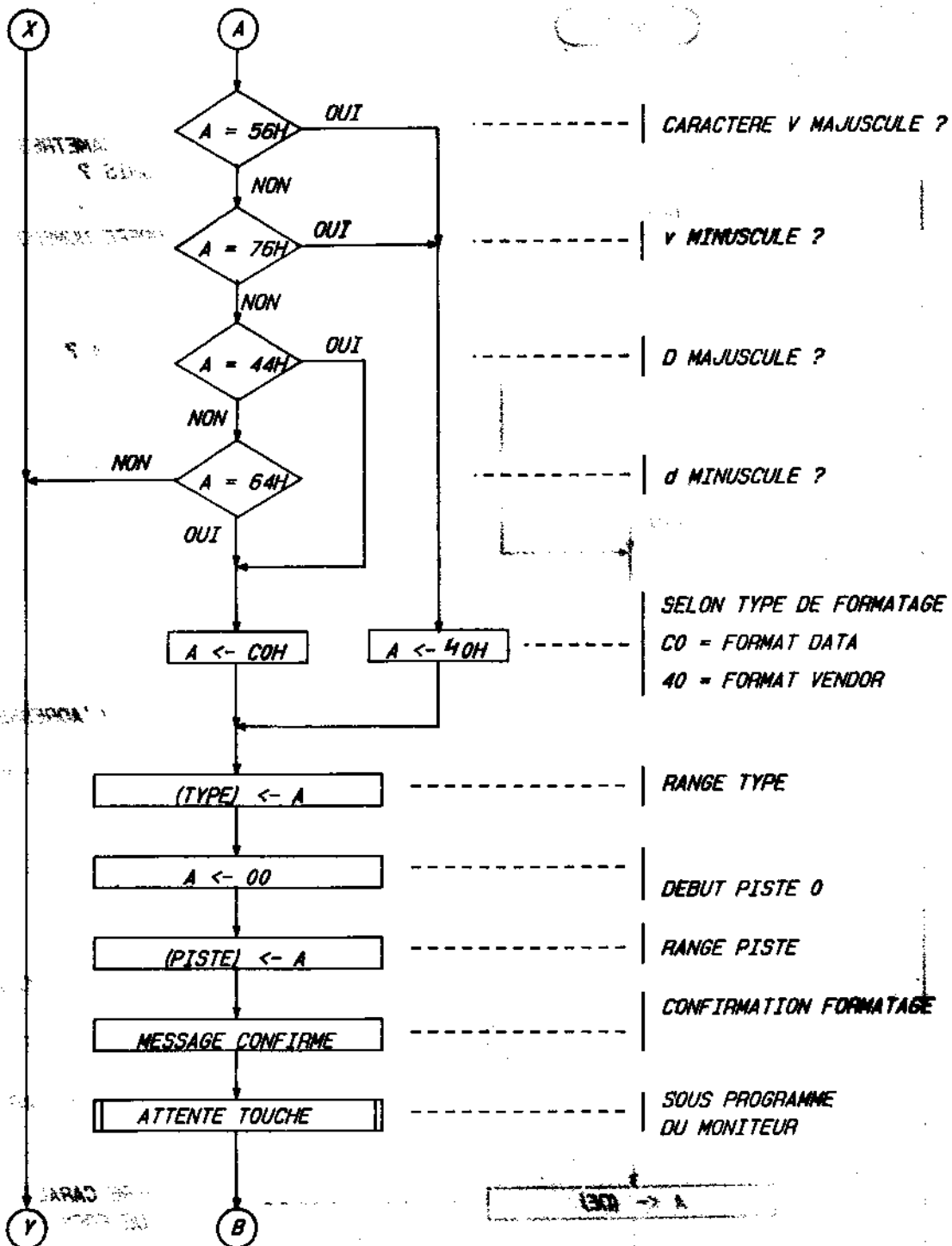
L'ordinogramme

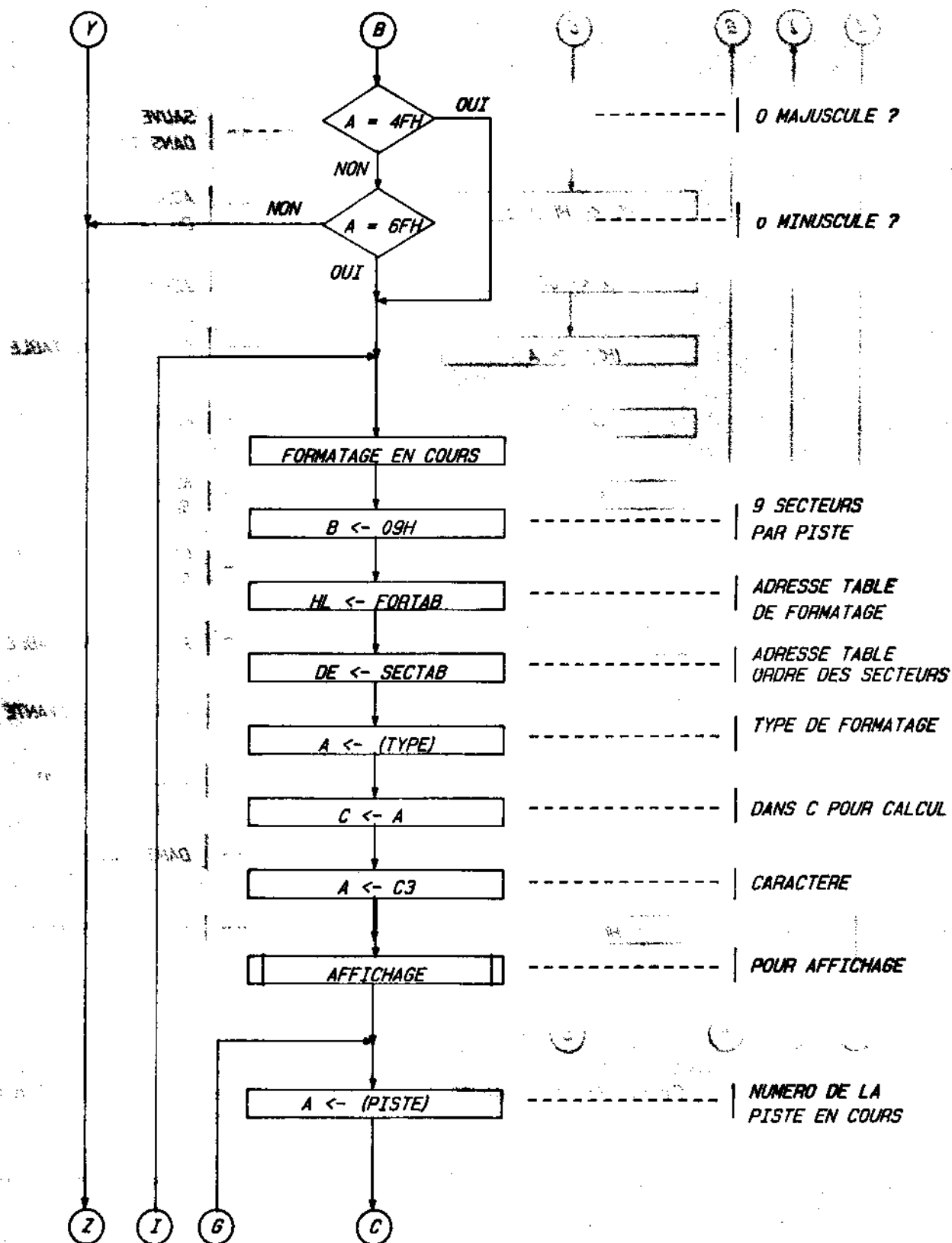
L'ordinogramme ou encore algorithme (on trouve parfois l'appellation organigramme, qui est plus générale), schématise la structure générale d'un programme ici en assembleur Z80.

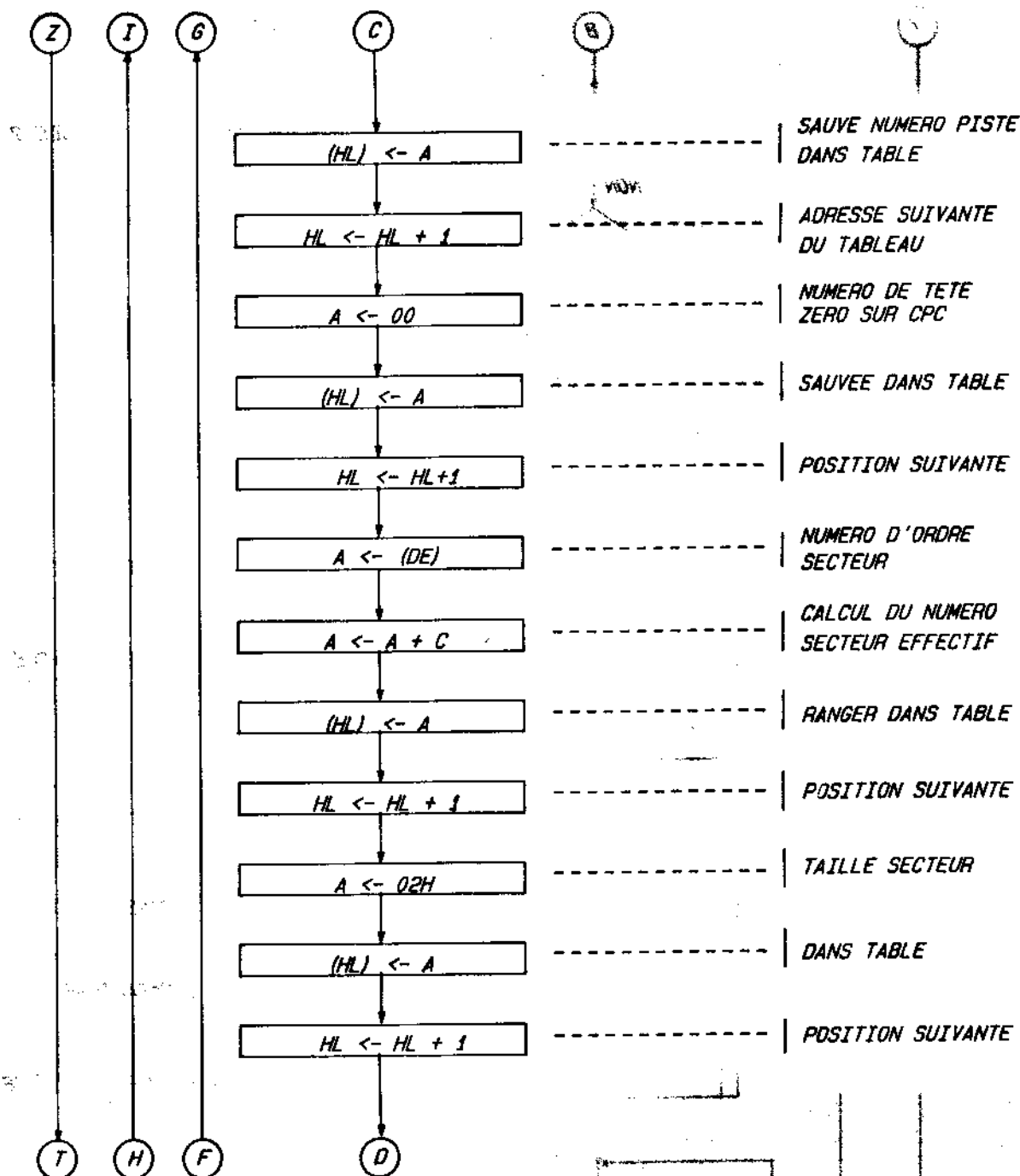
En premier lieu, un test sur le nombre de paramètres, envoyés lors de la frappe de ùFORMAT, lecteur, "type", qui se trouve dans le lecteur A.

On trouve ensuite deux tests concernant le numéro de lecture sur lequel on désire effectuer un formatage.





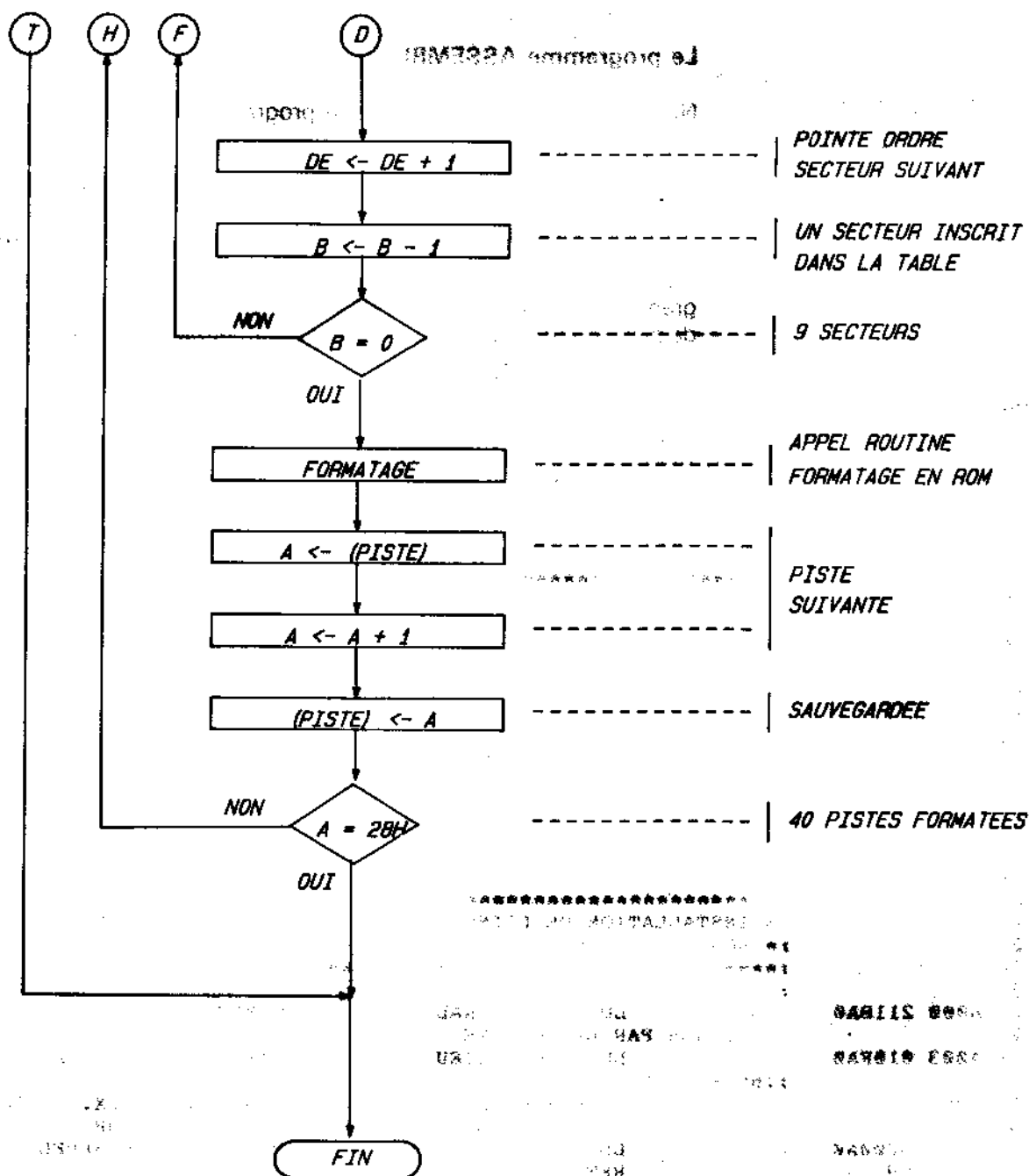




Ce numéro sauvegardé, le type de formatage est récupéré grâce à l'adresse de la chaîne de caractères, placée dans la pile paramètre, puis testé (4 possibilités pour le premier caractère).

Selon le type de formatage, le quartet haut est sauvegardé provisoirement (&40 ou &C0)...

On range ensuite le numéro de la première piste à formater avant de demander confirmation par l'envoi d'un message et l'attente de la frappe d'une touche (se reporter Partie 4, chapitre 1.6.2, pages 2 à 6). La rou-



tine utilisée pour l'attente de la frappe d'une touche est le classique vecteur &BB06, suite à l'appui d'une touche revient avec le code ASCII de celle-ci dans le registre A. Celui-ci est testé par rapport au caractère 0 ou 0.

En cas de confirmation, le formatage est signalé, puis la table de formatage créée pour une piste. Le formatage est lancé par l'appel de l'instruction &86. Et ainsi de suite pour chacune des 40 pistes.

La fin du formatage implique un retour au Basic bien sûr.

Le programme ASSEMBLEUR

Nous vous proposons ci-après le programme assembleur développé à partir de l'ordinogramme précédent.

Ce programme est commenté pour que chacun d'entre-vous puisse l'étudier et approfondir ses connaissances, tant sur la création des RSX, que dans le remplissage d'une table, ou l'utilisation de la nouvelle instruction.

Nous allons uniquement décrire ici ce qui n'apparaît pas dans l'ordinogramme, (car cela n'est pas toujours évident à insérer) : les réservations de place mémoire pour les différents paramètres éventuellement variables.

```

1      ;*****
2      ;* REALISATION DE L'INSTRUCTION *
3      ;*   DE FORMATAGE EN BASIC   *
4      ;*   |FORMAT, LECTEUR, "TYPE" | *
5      ;* LECTEUR A = 0 - LECTEUR B = 1 *
6      ;*   TYPE DATA = D - VENDOR = V *
7      ;*****
8      ;
9      ;
10     ;** ORIGINE D'ASSEMBLAGE      **
11     ;
12     ;           ORG 0A000H
13     ;
14     ;
15     ;** ADRESSE DE CHARGEMENT DU  **
16     ;**   CODE MACHINE           **
17     ;
18     ;           LOAD 0A000H
19     ;
20     ;
21     ;*****
22     ;* INSTALLATION DE L'INSTRUCTION *
23     ;* SOUS LA FORME D'UNE R.S.X.   *
24     ;*****
25     ;
26 A000 211BA0 DEBUT: LD HL, KERNAL ;POINTE 4 OCTET
27 ;UTILISES PAR LA ROUTINE
28 A003 010FA0 LD BC, VECTEU ;POINTE LA TABLE DES
29 ;INSTRUCTIONS
30 A006 CDD1BC CALL 0BCD1H ;AJOUTE LA R.S.X.
31 A009 3EC9 LD A, 0C9H ;CHARGE RET POUR
32 A00B 3200A0 LD (DEBUT), A ;INTERDIR AURE APPEL
33 A00E C9 RET ;FIN INSTALLATION
34 ;
35 A00F 14A0 VECTEU: DEFW TABLE ;ADRESSE DE LA
36 ;TABLE D'INSTRUCTION
37 A011 C31FA0 JP FORMAT ;SAUT AU TRAITEMENT
38 ;
39 A014 464F524D TABLE: DEFB "FORMA" ;DEBUT DE LA
39 A018 41
40 ;DEFINITION DE L'INSTRUCTION
41 A019 D4 DEFB 080H+"T" ;DERNIERE LETTRE
42 A01A 00 DEFB 0 ;FIN DE TABLE
43 ;

```

```

44      KERNAL:      DEFS 4      ;4 OCTETS POUR LA
45      ;ROUTINE KL-LOG-TEXTE
46      ;
47      ;*****
48      ;
49      ;
50      ;*****
51      ;* EXECUTION DE L'INSTRUCTION *
52      ;*****
53      ;
54      ;
55      ;
56      ;** RECUPERATION DES PARAMETRES **
57      ;
58      A01F FE02      FORMAT:      CP      02H      ;DEUX PARAMETRES?
59      A021 C0      RET      NZ      ;NON ALORS FIN
60      A022 DD7E02      LD      A,(IX+02H)      ;RECUPERE LECTEUR
61      A025 FE00      CP      00H      ;LECTEUR A?
62      A027 2803      JR      Z,FORMA1      ;OUI ALORS SUITE
63      A029 FE01      CP      01H      ;LECTEUR B?
64      A02B C0      RET      NZ      ;NON ALORS RETOUR
65      A02C 32D3A0      FORMAL:      LD      (LECTEU),A      ;SAUVE LE
66      ;NUMERO DU LECTEUR
67      A02F DD6E00      LD      L,(IX+00H)      ;RECUPERE ADRESSE
68      ;BASSE DU POINTEUR DE TYPE
69      A032 DD6601      LD      H,(IX+01H)      ;RECUPERE ADRESSE
70      ;HAUTE DU POINTEUR DE TYPE
71      A035 23      INC      HL      ;POINTE SUR ADRESSE BASSE
72      A036 5E      LD      E,(HL)      ;CHARGE OCTET BAS
73      ;DE L'ADRESSE DU TYPE
74      A037 23      INC      HL      ;POINTE SUR ADRESSE HAUTE
75      A038 56      LD      D,(HL)      ;CHARGE OCTET HAUT
76      ;DE L'ADRESSE DE TYPE
77      A039 1A      LD      A,(DE)      ;CHARGE TYPE DE FORMATAG
78      ;
79      ;
80      ;** CONVERSION TYPE FORMATAGE **
81      A03A FE56      CP      056H      ;FORMAT VENDOR
82      A03C 280D      JR      Z,VENDOR      ;oui alors traitement
83      A03E FE76      CP      076H      ;format vendor minuscule
84      A040 2809      JR      Z,VENDOR      ;oui alors traitement
85      A042 FE44      CP      044H      ;format DATA majuscule
86      A044 2809      JR      Z,DATA      ;oui alors traitement
87      A046 FE64      CP      064H      ;format data minuscule
88      A048 2805      JR      Z,DATA      ;oui alors traitement
89      A04A C9      RET      ;non alors fin
90      A04B 3E40      VENDOR:      LD      A,040H      ;charger poids
91      ;fort du type vendor
92      A04D 1802      JR      FORMA2      ;puis suite
93      A04F 3EC0      DATA:      LD      A,0C0H      ;charger poids fort
94      ;du formatage de type data
95      A051 32D5A0      FORMA2:      LD      (TYPE),A      ;SAUVER LE TYPE
96      A054 3E00      LD      A,00H      ;CHARGER PISTE DEPART
97      A056 32D4A0      LD      (PISTE),A      ;ET LA SAUVEGARDER
98      ;
99      ;
100     ;*****
101     ;* ATTEND CONFIRMATION DU *
102     ;* FORMATAGE DE LA DISQUETTE *
103     ;*****
104     ;
105     A059 21E3A0      LD      HL,METDIS      ;POINTE MESSAGE

```

```

106      ;CONFIRMATION OU ANNULATION
107 A05C 7E      METD11: LD A,(HL)      ;PREND CARACTERE
108 A05D 23      INC HL      ;POINTE CARACTERE SUIVANT
109 A05E FEFF      CP 0FFH      ;DERNIER CARACTERE?
110 A060 2805      JR Z,METD12      ;OUI ALORS SUITE
111 A062 CD5ABB      CALL 0BB5AH      ;AFFICHE CARACTERE
112 A065 18F5      JR METD11      ;CARACTERE SUIVANT
113 A067 CD06BB      METD12: CALL 0BB06H      ;ATTENT LA
114      ;FRAPPE D'UNE TOUCHE
115 A06A FE4F      CP 04FH      ;TOUCHE O MAJUSCULE?
116 A06C 2803      JR Z,FORMAC      ;OUI ALORS FORMATAGE
117 A06E FE6F      CP 06FH      ;TOUCHE o minuscule?
118 A070 C0      RET NZ      ;NON ALORS FIN
119 A071 211FA1      FORMAC: LD HL,ENCOUR      ;POINTE MESSAG
120      ;FORMATAGE EN COURS
121 A074 7E      ENCOU1: LD A,(HL)      ;PREND CARACTERE
122 A075 23      INC HL      ;POINTE CARACTERE SUIVANT
123 A076 FEFF      CP 0FFH      ;DERNIER CARACTERE?
124 A078 2805      JR Z,FORMA5      ;OUI ALORS SUITE
125 A07A CD5ABB      CALL 0BB5AH      ;AFFICHE CARACTERE
126 A07D 18F5      JR ENCOU1      ;CARACTERE SUIVANT
127      ;
128      ;
129      ;*****
130      ;*CREER LA TABLE DE FORMATAGE *
131      ;* POUR CHACUNE DES PISTES *
132      ;*****
133      ;
134 A07F 0609      FORMA5: LD B,09H      ;NOMBRE DE SECTEUR
135 A081 2139A1      LD HL,FORTAB      ;HL POINTE LA TABLE
136      ;DE FORMATAGE
137 A084 11D6A0      LD DE,SECTAB      ;DE POINTE LA TABLE
138      ;D'ORDRE DES SECTEURS
139 A087 3AD5A0      LD A,(TYPE)      ;RECUPERER LE TYPE
140 A08A 4F      LD C,A      ;RANGER DANS C
141 A08B 3EE3      LD A,0E3H      ;AFFICHAGE D'UN
142 A08D CD5ABB      CALL 0BB5AH      ;CARACTERE POUR
143      ;INDIQUER LA PISTE EN COURS
144 A090 3AD4A0      FORMA4: LD A,(PISTE)      ;RECUPERER LE
145      ;NUMERO DE PISTE ACTUEL
146 A093 77      LD (HL),A      ;INCRIRE LE NUMERO DE
147      ;PISTE DANS LA TABLE DE FORMATAGE
148 A094 23      INC HL      ;POSITION SUIVANTE DE TAB
149 A095 3E00      LD A,00      ;NUMERO DE TETE D'ECRITUR
150      ;** TOUJOURS ZERO **
151 A097 77      LD (HL),A      ;SAUVER NO TETE -> TABLE
152 A098 23      INC HL      ;POSITION SUIVANTE
153 A099 1A      LD A,(DE)      ;RECUPERE LE NUMERO
154      ;D'ORDRE DE SECTEUR
155 A09A 81      ADD A,C      ;CREER LE NUMERO DE
156      ;SECTEUR AVEC LE TYPE
157 A09B 77      LD (HL),A      ;INSCRIRE NE NUMERO
158      ;DE SECTEUR CORRECT DANS TABLE
159 A09C 23      INC HL      ;POSITION SUIVANTE
160 A09D 3E02      LD A,02      ;TAILLE DU SECTEUR (2)
161 A09F 77      LD (HL),A      ;INSCRIT DANS TABLE
162      ;DE FORMATAGE
163 A0A0 23      INC HL      ;POSITION SUIVANTE
164 A0A1 13      INC DE      ;DE POINTE LE PROCHAIN
165      ;NUMERO D'ORDRE DE SECTEUR
166 A0A2 10EC      DJNZ FORMA4      ;DECREMENTER B POUR
167      ;SAVOIR SI LES NEUF SECTEURS

```

```

168 ; SONT INSCRITS, SINON RECOMMENCER
169 ;
170 ;
171 ;*****
172 ;* FORMATAGE D'UNE PISTE *
173 ;* GRACE A L'INSTRUCTION &86 *
174 ;* DE LA ROM DE AMSDOS *
175 ;*****
176 ;
177 A0A4 21DFA0 LD HL,COMMAN ; POINTE LE NOM DE LA
178 ; COMMANDE &86H
179 A0A7 CDD4BC CALL 0BCD4H ; VA CHERCHER SES
180 ; COORDONNEES EN ROM
181 A0AA 22E0A0 LD (ADRINS),HL ; SAUVE L'ADRESSE
182 A0AD 79 LD A,C ; RECUPERE LE NUMERO DE RO
183 A0AE 32E2A0 LD (NOROM),A ; LE SAUVE
184 A0B1 3AD3A0 LD A,(LECTEU) ; CHARGE NO LECTEUR
185 A0B4 5F LD E,A ; DANS REGISTRE E
186 A0B5 3AD4A0 LD A,(PISTE) ; CHARGE NO PISTE
187 A0B8 57 LD D,A ; DANS D
188 A0B9 3AD5A0 LD A,(TYPE) ; CHARGE TYPE
189 A0BC 4F LD C,A ; DANS C
190 A0BD 3E01 LD A,01 ; PREMIER SECTEUR
191 A0BF 81 ADD A,C ; NUMERO PREMIER SECTEUR
192 A0C0 4F LD C,A ; DANS REGISTRE C
193 A0C1 2139A1 LD HL,FORTAB ; HL POINTE LA TABLE
194 ; DE FORMATAGE
195 A0C4 DF RST 018H ; RESTART RST3
196 A0C5 E0A0 DEFB ADRINS ; VECTEUR POUR
197 ; RESTART
198 ;
199 ;
200 ;*****
201 ;* ENCORE UNE PISTE A FORMATER ? *
202 ;*****
203 ;
204 A0C7 3AD4A0 LD A,(PISTE) ; RECUPERE NO DE PISTE
205 A0CA 3C INC A ; PISTE SUIVANTE
206 A0CB 32D4A0 LD (PISTE),A ; SAUVEGARDE
207 A0CE FE28 CP 028H ; 40-IEME PISTE?
208 A0D0 20AD JR NZ,FORMA5 ; NON ALORS CONTINUER
209 ; A FORMATER
210 A0D2 C9 RET ; OUI ALORS FIN.
211 ;
212 ;*****
213 ;* FIN ROUTINE *
214 ;*****
215 ;
216 ;
217 ;*****
218 ;* ALLOCATION DES PARAMETRES *
219 ;*****
220 ;
221 A0D3 00 LECTEU: DEFB 00 ; NUMERO DE LECTEUR
222 A0D4 00 PISTE: DEFB 00 ; NUMERO DE PISTE
223 A0D5 00 TYPE: DEFB 00 ; TYPE DE FORMATAGE
224 ;
225 A0D6 01 SECTAB: DEFB 01 ; PREMIER SECTEUR
226 A0D7 03 DEFB 03 ; DEUXIEME
227 A0D8 05 DEFB 05 ; TROISIEME
228 A0D9 07 DEFB 07
229 A0DA 09 DEFB 09

```

```

230 A0DB 02      ;&86 DE AMSDOS      ;&86 DE AMSDOS      ;&86 DE AMSDOS      ;&86 DE AMSDOS
231 A0DC 04      DEF 04
232 A0DD 06      DEF 06
233 A0DE 08      DEF 08      ;NEUVIEME SECTEUR
234              ;A FORMATER
235              ;
236 A0DF 86      COMMAN: DEF 086H      ;INSTRUCTION
237              ;&86 DE AMSDOS      ;&86 DE AMSDOS      ;&86 DE AMSDOS      ;&86 DE AMSDOS
238              ;
239              ADRINS: DEF 2      ;ADRESSE INSTRUCTION
240 A0E2 00      NOROM: DEF 00      ;NUMERO DE ROM
241              ;
242 A0E3 0D0A      METDIS: DEF 0DH,0AH      ;
243 A0E5 494E5345      DEF "INSERER UNE"
243 A0E9 52455220
243 A0ED 554E45
244 A0F0 20444953      DEF " DISQUETTE"
244 A0F4 51554554
244 A0F8 5445
245 A0FA 0D0A0A      DEF 0DH,0AH,0AH
246 A0FD 45542041      DEF "ET APPUYER SUR"
246 A101 50505559
246 A105 45522053
246 A109 5552
247 A10B 203C4F3E      DEF " <O> POUR "
247 A10F 20504F55
247 A113 5220      ;
248 A115 434F4E46      DEF "CONFIRMER"
248 A119 49524D45
248 A11D 52
249 A11E FF      DEF 0FFH
250              ;
251 A11F 0D      ENCOUR: DEF 0DH      ;
252 A120 464F524D      DEF "FORMATAGE" ;
252 A124 41544147      ;
252 A128 45
253 A129 20454E20      DEF " EN COURS ..."
253 A12D 434F5552
253 A131 53202E2E
253 A135 2E
254 A136 0D0AFF      DEF 0DH,0AH,0FFH
255              ;
256              FORTAB: DEF 024H      ;36 OCTETS
257              ;RESERVES POUR LA TABLE
258              ;DE FORMATAGE
259              ;
260              ;*****
261              ;
262              END

```

— Lignes 221 à 223 : se trouvent les 3 adresses qui recevront les données lecteur, piste en cours de formatage et type de formatage (&40 pour le format VENDOR, &C0 pour le format DATA.

— Lignes 225 à 233 : on trouve la table d'ordre des secteurs 1, 3, 5, 7, 9, 2, 4, 6, 8, chiffres qui seront additionnés au nombre en relation avec le type de formatage avant d'être rangés dans la table de formatage.

— Ligne 239 : on réserve une zone de deux octets où sera rangée l'adresse de l'instruction &86 suite à l'appel de KL-FIND-COMMAND.

- Ligne 240 : ici viendra le numéro de ROM dans laquelle elle se trouve.
- Lignes 241 à 249 : message de confirmation.
- Lignes 251 à 254 : message signalant le formatage.
- Ligne 256 : une zone de 36 octets est réservée pour la table de formatage qui sera remplie pour chaque piste à partir de la table d'ordre des secteurs, du numéro de tête, de la taille des secteurs et des informations fournies (type, lecteur).

Le chargeur Basic

Ce programme permettra aux lecteurs, qui ne se sont pas encore laissés tenter par le langage d'assemblage, de bénéficier malgré tout de l'utilitaire de formatage.

Les directives d'utilisation et de sauvegardes, que vous prendrez soin de frapper dans leur intégralité, car des calculs y sont effectués (examinez de près la ligne 310), figurent dans le listing.

```

10 REM *****
20 REM * CREATION D'UNE INSTRUCTION *
30 REM * DE FORMATAGE SOUS BASIC *
40 REM * SOUS LA FORME DE LA RSX *
50 REM * |FORMAT,lecteur,"format" *
60 REM *****
70 MODE 2
80 PRINT "PATIENCE,CHARGEMENT EN COURS"
90 REM
100 REM *** CHARGEUR BASIC ***
110 ADRESSE = &A000
120 I = 0
130 SOMME = 0
140 RESTORE 470
150 READ AS
160 IF AS = "XX" THEN 260
170 BS = "&" + AS
180 B = VAL(BS)
190 SOMME = SOMME + B
200 POKE ADRESSE,B
210 ADRESSE = ADRESSE + 1
220 PRINT I;CHR$(13);
230 I = I + 1
240 GOTO 150
250 REM
260 REM ***** CONTROLE *****
270 IF SOMME <> 29611 THEN PRINT CHR$(12);CHR$(7);"ERREUR DANS LES DATAS":STOP
280 REM
290 REM *** SAUVEGARDE ***

```

```

300 PRINT "POUR SAUVEGARDER LA ROUTINE E
N BINAIRE"
310 PRINT "SAVE ";CHRS(34);"FORMAT.BIN";
CHRS(34);".B.&A000.";RIGHT$(STR$(I),LEN(
STR$(I))-1)
320 PRINT
330 PRINT "CHARGEMENT PAR LOAD ";CHRS(34
);"FORMAT.BIN";CHRS(34);".&A000"
340 PRINT "MEMORY &9FFF"
350 PRINT
360 PRINT "INITIALISATION PAR CALL &A000
"
370 PRINT
380 PRINT "UTILISATIONS POSSIBLES:"
390 PRINT "      - |FORMAT,0,"D" : FORMATA
GE DATA SUR DISQUE A"
400 PRINT "      - |FORMAT,0,"V" : FORMATA
GE VENDOR SUR DISQUE A"
410 PRINT "      - |FORMAT,1,"D" : FORMATA
GE DATA SUR DISQUE B"
420 PRINT "      - |FORMAT,1,"V" : FORMATA
GE VENDOR SUR DISQUE B"
430 PRINT
440 STOP
450 REM
460 REM *** CREATION RSX ***
470 DATA 21,1B,A0,01,0F,A0,CD,D1
480 DATA BC,3E,C9,32,00,A0,C9,14
490 DATA A0,C3,1F,A0,46,4F,52,4D
500 DATA 41,D4,00,00,00,00,00
510 REM *** RECUPERER PARAMETRES ***
520 DATA FE,02,C0,DD,7E,02,FE,00
530 DATA 28,03,FE,01,C0,32,D3,A0
540 DATA DD,6E,00,DD,66,01,23,5E
550 DATA 23,56,1A
560 REM *** CONVERSION TYPE FORMAT ***
570 DATA FE,56,28,0D,FE,76,28,09
580 DATA FE,44,28,09,FE,64,28,05
590 DATA C9,3E,40,18,02,3E,C0,32
600 DATA D5,A0,3E,00,32,D4,A0
610 REM *** AFFICHAGE MESSAGE ***
620 DATA 21,E3,A0,7E,23,FE,FF,28
630 DATA 05,CD,5A,BB,18,F5,CD,06
640 DATA BB,FE,4F,28,03,FE,6F,C0
650 DATA 21,1F,A1,7E,23,FE,FF,28
660 DATA 05,CD,5A,BB,18,F5
670 REM *** CREATION TABLE ***
680 DATA 06,09,21,39,A1,11,D6,A0
690 DATA 3A,D5,A0,4F,3E,E3,CD,5A
700 DATA BB,3A,D4,A0,77,23,3E,00
710 DATA 77,23,1A,81,77,23,3E,02

```

```
720 DATA 77,23,13,10,EC
730 REM *** FORMATAGE PISTE ***
740 DATA 21,DF,A0,CD,D4,BC,22,E0
750 DATA A0,79,32,E2,A0,3A,D3,A0
760 DATA 5F,3A,D4,A0,57,3A,D5,A0
770 DATA 4F,3E,01,81,4F,21,39,A1
780 DATA DF,E0,A0
790 REM *** PISTE SUIVANTE ET FIN ***
800 DATA 3A,D4,A0,3C,32,D4,A0,FE
810 DATA 28,20,AD,C9
820 REM *** ALLOCATION PARAMETRES ***
830 DATA 00,00,00
840 REM *** TABLE SECTEUR ***
850 DATA 01,03,05,07,09,02,04
860 DATA 06,08
870 REM *** RESERVATION FORMATAGE ***
880 DATA 86,00,00,00
890 REM
900 REM *** ZONE MESSAGES ***
910 REM *** INSERER DISQUETTE ***
920 DATA 0D,0A,49,4E,53,45,52,45
930 DATA 52,20,55,4E,45,20,44,49
940 DATA 53,51,55,45,54,54,45,0D
950 REM *** CONFIRMATION ? ***
960 DATA 0A,0A,45,54,20,41,50,50
970 DATA 55,59,45,52,20,53,55,52
980 DATA 20,3C,4F,3E,20,50,4F,55
990 DATA 52,20,43,4F,4E,46,49,52
1000 DATA 4D,45,52,FF,0D
1010 REM *** FORMATAGE EN COURS ***
1020 DATA 46,4F,52,4D,41,54,41,47
1030 DATA 45,20,45,4E,20,43,4F,55
1040 DATA 52,53,20,2E,2E,2E,0D,0A
1050 DATA FF
1060 REM
1070 REM *** RESERVE TABLE FORMATAGE ***
1080 DATA 00,00,00,00,00,00,00,00
1090 DATA 00,00,00,00,00,00,00,00
1100 DATA 00,00,00,00,00,00,00,00
1110 DATA 00,00,00,00,00,00,00,00
1120 DATA 00,00,00,00
1130 REM *** FIN DE DATA ***
1140 DATA XX
1150 END
1160 REM
```

Nous vous conseillons de frapper le programme et de le charger sur disquette aussitôt (une disquette préalablement formatée, bien sûr).

Vous effectuerez ensuite une réinitialisation (RESET) de votre micro-ordinateur (par SHIFT-CONTROL-ESC), puis, avant d'effectuer

RUN "Nomduprogramme"

vous frapperez

MEMORY &9FFF

afin d'éviter tout conflit avec la gestion des fichiers.

Vous pourrez ensuite sauvegarder le fichier binaire comme indiqué, ou initialiser le RSX par

CALL &A000

Enfin, NEW, et finis les petits tracas de disquettes non formatées ! Vous pouvez désormais entrer vos programmes Basic sans soucis.

Limites d'utilisation

Utilisez cette nouvelle instruction dans un programme si cela est nécessaire ; veillez tout de même à effectuer un RESET pour toute utilisation d'un programme écrivant dans la zone de mémoire entre &A000 et HIMEM. Vous rechargerez ce performant utilitaire avant de débiter une nouvelle saisie.

Un petit défaut, le programme ne vérifie pas si la disquette est correctement formatée, ce qui peut arriver (mais très rarement) avec des disquettes possédant un vice de fabrication, ou usagées. Dans le premier cas, faites valoir la garantie auprès de votre revendeur, dans le deuxième, quelques nouvelles tentatives de formatage en viendront peut-être à bout, sinon, mieux vaut la jeter, elle a fait son temps.

```
***
**SS
**
```

```
*** NEW TP
**
```

```
*** SYSTEM
```

```
*** IDATANE
```

```
***
**
```

```
**
**SS
**SS
**SS
**SS
**SS
```

```
*** RUN
**
**SS
**
```

```
*** IDATANE
```

```
**SS
```

```
**SS
```

```
*** ATAN
```

```
XX
```

```
MAN
```

4/1.6.6 **Les tokens**

Accélérez vos programmes Basic Les tokens

Vous le savez tous, le langage de programmation Basic n'est pas la Formule 1 des langages existants. Ceux qui désirent écrire des programmes performants, en vitesse d'exécution, ce qui est le cas pour bon nombre de jeux d'arcades, de programmes de tri de données, ou de gestion graphique, se sont vite heurtés à la lenteur du Basic.

Beaucoup se sont alors intéressés aux langages compilables auxquels le système d'exploitation CP/M leur permet d'accéder : TURBO-PASCAL et C-BASIC. D'autres, au vu des faibles possibilités graphiques de ces langages, ont dû recourir à l'assembleur.

Outre une mise en œuvre plus fastidieuse (il faut d'abord charger CP/M, appeler ensuite le langage, puis compiler, et enfin revenir à CP/M pour lancer le programme créé), ces programmes demandent une étude structurée qui peut sembler rebutante à certains, surtout à ceux qui ont une expérience de longue date sur Basic.

Nous allons dans ce chapitre vous proposer des trucs, des commandes et un programme qui vous permettront d'accélérer les performances des vôtres. Hormis les quelques trucs utilisables sur tous CPCs, il vous faudra néanmoins posséder une unité de disquette pour l'utilisation du programme que nous avons dénommé "ANTI-REM".

PRÉAMBULE

LES TOKENS

Votre première impression après avoir lu les lignes précédentes est peut-être de vous demander pourquoi nous n'avons pas utilisé toutes les astuces que nous détaillerons dans les paragraphes suivants (élimination des espaces inutiles au fonctionnement, des instructions REM ou ').

Sachez que c'est tout d'abord dans un esprit de clarté que nous introduisons parfois des espaces, séparant le nom d'une variable d'un opérateur. Il en est de même dans les noms des variables, quelquefois longs, mais certainement plus significatifs que l'utilisation de la suite A1, A, A3, ..., B\$, C\$, ... n'ayant pas de rapport avec l'affectation qui leur sont données. Notre souci est aussi de vous permettre de relire et d'étudier plus facilement les programmes que nous vous proposons.

I. Basic-Tortue

Le langage Basic est lent parce que c'est un langage interprété.

Sans nous étendre sur le fonctionnement précis de l'interpréteur Basic de votre AMSTRAD, sachez que chaque ligne d'un programme, avant exécution, est traduite par les logiciels des ROMs. Chaque instruction est étudiée puis comparée aux instructions en ROM, les variables sont lues une à une pour reformer leur nom qui sera ensuite recherché dans la RAM, éventuellement créé s'il n'existe pas, etc.

Même dans une boucle, tout ce processus est réitéré à chaque passage ainsi dans les lignes :

```
10 WHILE A < 10000
20 VARIABLE = VARIABLE + 1:REM INCREMENTATION
30 WEND
```

La recherche de la variable, après lecture est effectuée 2×10000 fois. De plus les différents espaces entre le signe = et le signe + sont lus chacun 10000 fois, de même pour l'instruction REM.

II. Vers le Basic-Lièvre

Pour modifier les programmes Basic afin de les rendre plus performants, nous allons devoir travailler sur les chaînes de caractères composant les lignes numérotées.

Or il est impossible de travailler facilement sur un programme directement en mémoire. Il est aussi très difficile de travailler sur le programme sauvegardé sous la forme classique en fichier .BAS.

LES TOKENS

Pourquoi ? L'interpréteur Basic, après chaque frappe sur la touche <RETURN> ou <ENTER> suivant chaque ligne de programme, compare toutes les instructions qu'il rencontre par rapport à une table qu'il a en ROM, et les traduit en un code, prenant moins de place mémoire, appelé token.

Nous vous proposons ci-après un court programme permettant de lire certains tokens.

```
10 MODE 2
20 PRINT "LECTURE DE TOKENS"
30 FOR compteur = &170 TO &200
40   FOR J = 0 TO 8
50     A = PEEK(I+J)
60     A$ = HEX$(A)
```

```

70      IF LEN(A$) = 1 THEN A$ = "0" + A$
80      PRINT A$ : SPACE$(2);
90      NEXT J
100     PRINT SPACE$(5)
110     FOR J = 0 TO 8
120     IF (PEEK(I+J) < 32 OR PEEK(I+J) > 127)
        THEN A = 46 ELSE A = PEEK(I+J)
130     NEXT J
140     PRINT
150 NEXT compteur

```

Il faut d'abord savoir qu'un programme Basic débute à l'adresse &170. Le programme précédent va donc lire à partir de &170 tous les octets jusqu'à l'adresse &200 (arbitraire), et les affiche ainsi que leur traduction ASCII quand elle est possible (sinon il affiche un point).

Après lancement, vous pourrez lire comme première liste, la ligne suivante :

```

octet   08   00   0A   00   AD   20   10   00
adresse &170 &171 &172 &173 &174 &175 &176 &177

```

où 08 00 indique le nombre d'octets comportant la ligne, y compris ces deux valeurs (à remettre dans l'ordre : 0008), 0A00 est le numéro de la ligne (retranscrit en inversant : 000A = 10), AD est le code de MODE (voir le tableau ci-après), 20 l'espace, 10 le code de la constante 2 et 00 signalant la fin de la ligne. Ensuite débute la ligne suivante sous le même principe.

Vous pourrez remarquer que pour le nom des variables, il est ajouté la valeur &80 à la valeur ASCII de la dernière lettre.

Vous pouvez ainsi modifier directement le contenu d'une ligne, par exemple en remplaçant le token AD de MODE, en position &174 par le token de PAPER: &BA. Frappez :

```
POKE &174,&BA puis LIST
```

et admirez le résultat.

Il est même possible de modifier un numéro de ligne en pokant à l'adresse &173 ou &172 une autre valeur. Si vous listez, vous aurez la surprise de constater que le numéro de ligne est modifié, par contre sa position est identique.

Nous vous fournissons ci-dessous la liste des tokens utilisés par l'interpréteur Basic, pour vous permettre d'approfondir le sujet.

Grâce à un désassemblage de la ROM du CPC 6128, nous avons retrouvé la liste des tokens Basic de ce modèle, qui ne doivent pas différer des autres, à part les quelques instructions supplémentaires.

80	AFTER	81	AUTO	82	BORDER
83	CALL	84	CAT	85	CHAIN
86	CLEAR	87	CLG	88	CLOSEIN
89	CLOSEOUT	8A	CLS	8B	CONT
8C	DATA	8D	DEF	8E	DEFINT
8F	DEFREAL	90	DEFSTR	91	DEG
92	DELETE	93	DIM	94	DRAW
95	DRAWR	96	EDIT	97	ELSE
98	END	99	ENT	9A	ENV
9B	ERASE	9C	ERROR	9D	EVERY
9E	FOR	9F	GOSUB	A0	GOTO
A1	IF	A2	INK	A3	INPUT
A4	KEY	A5	LET	A6	LINE
A7	LIST	A8	LOAD	A9	LOCATE
AA	MEMORY	AB	MERGE	AC	MID\$
AD	MODE	AE	MOVE	AF	MOVER
B0	NEXT	B1	NEW	B2	ON
B3	ON BREAK	B4	ON ERROR	B5	ON SO
B6	OPENIN	B7	OPENOUT	B8	ORIGIN
B9	OUT	BA	PAPER	BB	PEN
BC	PLOT	BD	PLOTR	BE	POKE
BF	PRIN	CO		C1	RAD
C2	RANDOMIZE	C3	READ	C4	RELEASE
C5	REM	C6	RENUM	C7	RESTORE
C8	RESUME	C9	RETURN	CA	RUN
CB	SAVE	CC	SOUND	CE	STOP
CF	SYMBOL	DO	TAG	D1	TAGOFF
D2	TRON	D3	TROFF	D4	WAIT
D5	WEND	D6	WHILE	D7	WIDTH
D8	WINDOW	D9	ZONE	DA	WRITE
DB	DI	DC	EI	DD	FILL
DE	GRAPHICS	DF	MASK	EO	FRAME
E1	CURSOR	E2	non défini	E3	ERL
E4	FN	E5	SPS	E6	STEP
E7	SWAP	E8	non défini	E9	non défini
EA	TAB	EB	THEN	EC	TO
ED	USING	EE	>	EF	=
FO	> =	F1	<	F2	< >
F3	< =	F4	+	F5	-
F6	*	F7	/	F8	
F9	\	FA	AND	FB	MOD
FC	OR	FD	XOR	FE	NOT
FF 00	ABS	FF 01	ASC	FF 02	ATN
FF 03	CHR\$	FF 04	CINT	FF 05	COS
FF 06	CREAL	FF 07	EXP	FF 08	FIX
FF 09	FRE	FF 0A	INKEY	FF 0B	INP
FF 0C	INT	FF 0D	JOY	FF 0E	LEN
FF 0F	LOG	FF 10	LOG10	FF 11	LOWER\$
FF 12	PEEK	FF 13	REMAIN	FF 14	SGN
FF 15	SIN	FF 16	SPACE\$	FF 17	SQ
FF 18	SQR	FF 19	STR\$	FF 1A	TAN
FF 1B	UNT	FF 1C	UPPER\$	FF 1D	VAL
FF 1E	à FF 3F	NON	AFFECTES	FF 42	HIMEM
FF 40	EOF	FF 41	ERR	FF 45	RND
FF 43	INKEY\$	FF 44	PI	FF 48	YPOS
FF 46	TIME	FF 47	XPOS	FF 70	NON AFFECTES
FF 49	DERR	FF 4A	à	FF 73	HEX\$
FF 71	BIN\$	FF 72	DEC\$	FF 76	MAX
FF 74	INSTR	FF 75	LEFT\$	FF 79	RIGHT\$
FF 77	MIN	FF 78	POS	FF 7C	TEST
FF 7A	ROUND	FF 7B	STRING\$	FF 7F	VPOS
FF 7D	TESTR	FF 7E	COPYCHR\$		

En plus de ces tokens concernant les instructions, on trouve les tokens signalant les fins de blocs d'instructions, les variables, les constantes, les valeurs de tout type, et fin de ligne :

- 00 fin de ligne
- 01 fin de bloc d'instruction
- 02 variable entière suivie du nom codé en ASCII
(plus &80 pour le code ASCII du dernier caractère)
- 03 variable de caractère (identique à ci-dessus)
- 04 variable réelle (identique à ci-dessus)
- 0D variable non définie (identique à ci-dessus)
- 0E chiffre 0
- 0F chiffre 1
- 10 chiffre 2
- 11 chiffre 3
- 12 chiffre 4
- 13 chiffre 5
- 14 chiffre 6
- 15 chiffre 7
- 16 chiffre 8
- 17 chiffre 9
- 18 non défini
- 19 valeur sur un octet suivi de la valeur
- 1A valeur décimale (2 octets)
- 1B valeur binaire (2 octets)
- 1C valeur hexadécimale (2 octets)
- 1D adresse de ligne
- 1E numéro de ligne
- 1F valeur en virgule flottante (5 octets)

Remarques :

Les tokens &1D et &1E :

Lorsque vous entrez une ligne du type :

40 GOTO 20

le renvoi à la ligne 20 est codé une première fois par le numéro de ligne en hexadécimal (on trouvera donc le code &1E le précédant). Puis, lors d'une exécution, l'interpréteur recherche la position de la ligne dans la

mémoire, puis remplace le ~~numéro~~ de ligne par son adresse effective, ce qui permettra d'accélérer ensuite l'exécution. Il signale l'adresse en remplaçant le code &1E par le code &1D.

Lors d'un listing, d'une insertion de ligne, ou d'une destruction, il lui sera ensuite facile de retrouver le numéro de ligne, puisqu'il se trouve en troisième et quatrième position à partir de l'adresse.

LA SAUVEGARDE ASCII

Sachant, que, lorsque vous ~~savegardez~~ le programme Basic sous la forme habituelle :

SAVE "PROG.BAS"

vous comprendrez après toutes les explications précédentes qu'il va devenir fastidieux de traiter tous ces tokens. De plus l'utilisation de l'instruction :

OPENIN "PROG.BAS"

pour récupérer les codes provoque le message d'erreur :

File type error

La solution est de sauvegarder le fichier sous sa forme ASCII, c'est-à-dire caractère par caractère par la commande :

SAVE "PROG.ASC",A

III. La bonne utilisation des variables

Une première modification possible, mais manuelle, vous permettra de gagner en vitesse d'exécution, notamment dans les boucles de type **FOR ... NEXT**, **WHILE ... WEND** et dans les sous-programmes souvent appelés par **GOSUB** ou **ON GOSUB...**

Il vous faudra manuellement repérer les variables qui sont utilisées dans le format **ENTIER**, c'est-à-dire qui ne comporteront que des valeurs de - 32768 à + 32767. Vous pourrez alors les définir de type **ENTIER** par l'instruction :

DEFINT liste de variables

Par la même occasion, repérer les variables réelles et chaîne de caractères que vous définirez par :

DEFREAL et DEFSTR.

Si cela est possible, utilisez des noms de variables courts, car le programme les relit entièrement avant de les rechercher dans la mémoire.

Vous pourrez aussi regrouper des lignes de programmes non concernées par les branchements de type **GOTO**, **GOSUB...** ou les tests (condition **IF... THEN... ELSE**), en une seule ligne, mais ceci est préjudiciable à la clarté du programme ; vous éviterez donc de l'effectuer sur la version originale.

D'autres possibilités vous sont offertes pour améliorer les temps d'exécutions, nous vous proposons ci-après des programmes qui permettront d'automatiser les actions à effectuer.

IV. Eliminons les blancs inutiles

Un truc connu probablement par beaucoup d'entre vous, pour permettre à l'interpréteur d'éliminer automatiquement les espaces indésirables, est de frapper la commande :

POKE &AC00,1

avant toute entrée de ligne de programme.

Seulement, pour des facilités de corrections ultérieures en cas d'erreur, nous vous conseillons d'entrer nos programmes tels qu'ils sont décrits, ou de frapper les vôtres sans être avare d'espaces.

Nous vous conseillons dans un premier temps de sauvegarder le programme sous une forme de fichier ASCII par la commande :

SAVE "PROG.ASC",A

Une autre possibilité pour sauvegarder le programme sous forme ASCII est d'ouvrir un fichier au nom du programme par l'instruction OPENOUT, puis de lister le programme par l'intermédiaire de l'unité de sauvegarde :

**OPENOUT "PROG.ASC"
LIST #9
CLOSEOUT**

Vous pourrez ensuite utiliser la case mémoire précédente (&AC00) pour éliminer les blancs, mais sous une forme différente. Le truc étant de recharger le programme précédent après avoir frappé le POKE salutaire.

Voici donc l'ordre des opérations à effectuer :

**SAVE "PROG.ASC",A
POKE &AC00,1
LOAD "PROG.ASC"
SAVE "PROG.BAS"**

Vous disposerez ainsi sur votre unité de sauvegarde de deux versions du programme : la version ASCII que vous pourrez réutiliser ultérieurement afin d'y effectuer des modifications ou des études sur certaines portions, et la version Basic sans blanc, plus optimisée pour l'exécution.

V. Fini le blabla...

Tout comme les espaces, l'instruction REM, utilisée pour ajouter des commentaires, allonge considérablement les temps d'exécution, surtout dans les boucles.

Nous avons donc conçu un programme qui enlève automatiquement les instructions REM et leurs commentaires associés.

La réflexion, qui s'est imposée, concerna les différentes façons d'écrire l'instruction dans un programme.

- Premièrement REM peut se trouver seul dans une ligne Basic, telle la ligne

10 REM *** PROGRAMME MACHIN ***

Si l'on supprime toute l'instruction, il nous restera une ligne vide portant le numéro 10. Deux cas peuvent alors se présenter :

- soit il est possible de supprimer la ligne sans porter préjudice au fonctionnement du programme ;
- soit une autre ligne fait référence à celle-ci, et dans ce cas il est hors de question de la supprimer, sous peine de plantage.

Il vous appartiendra de vérifier ce deuxième cas, et d'effectuer les modifications nécessaires, pour utiliser l'une ou l'autre version du programme que nous vous proposons.

- Deuxièmement REM peut se trouver en fin d'une ligne comportant une ou plusieurs instructions, comme par exemple :

20 PRINT "PROGRAMME MACHIN":REM AFFICHAGE

Notre programme devra donc supprimer la fin de la ligne REM + commentaire, mais aussi le caractère : devenu inutile.

- Troisièmement REM peut faire partie du nom d'une variable :

30 FLAGREM = 2

Il ne faudrait pas supprimer le REM et sa suite.

- Quatrièmement il faudra être vigilant à ne pas supprimer les REMarques insérées entre guillemets, qui sont réservées à l'affichage :

40 INPUT "FRAPPEZ REM POUR LA SUITE";A\$

Ce qui serait quelque peu embarrassant pour la compréhension du programme, voire son déroulement.

Nous avons de plus réservé la possibilité quelquefois intéressante de pouvoir mettre des instructions en remarques, instructions qui se trouveraient replacées dans le programme, suite à l'utilisation du logiciel que nous vous proposons. Il suffirait d'insérer le caractère : (deux points) entre la remarque et l'instruction.

50 REM: ON BREAK CONT ou

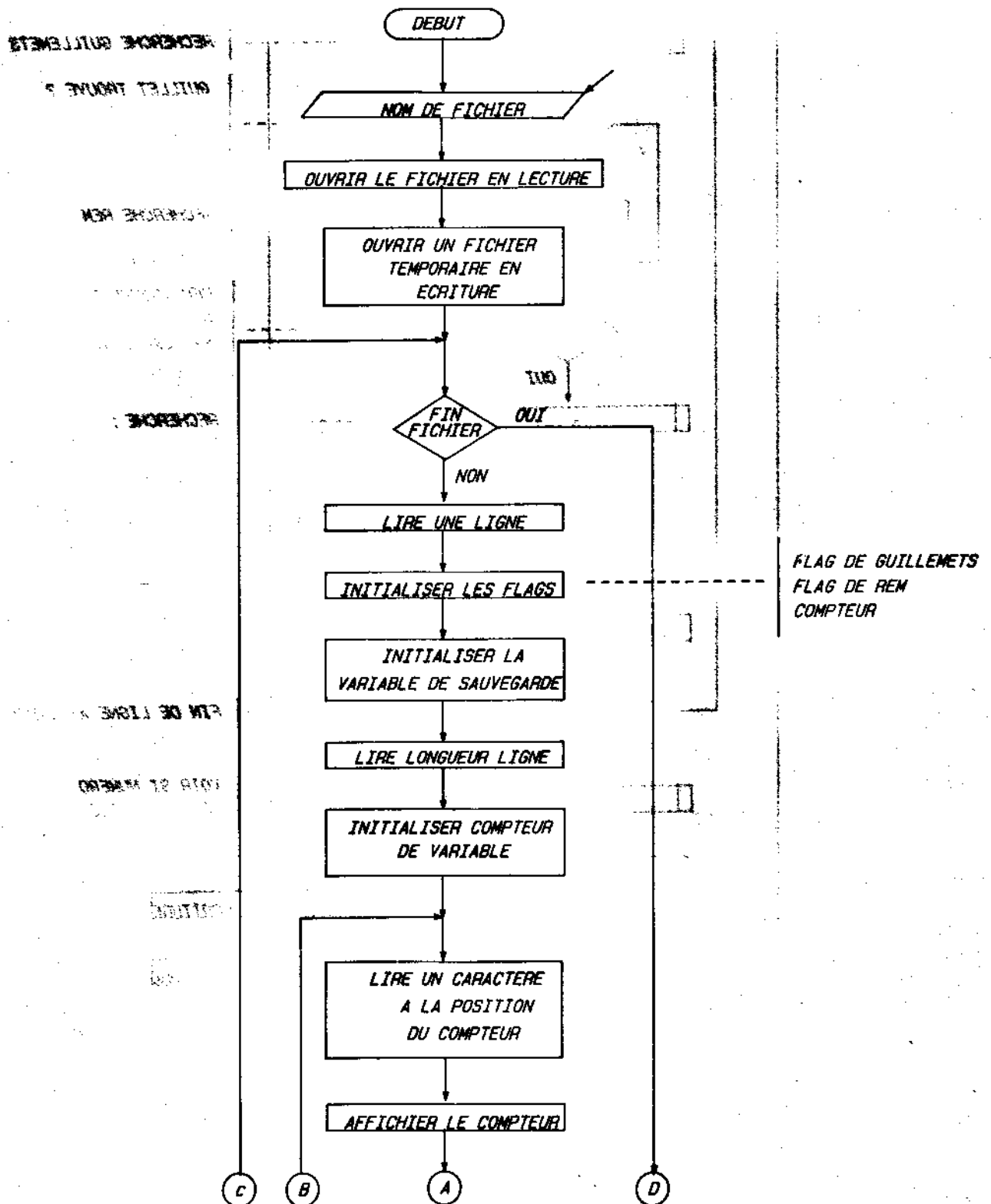
50 REM A ELIMINER: ON BREAK CONT deviendront

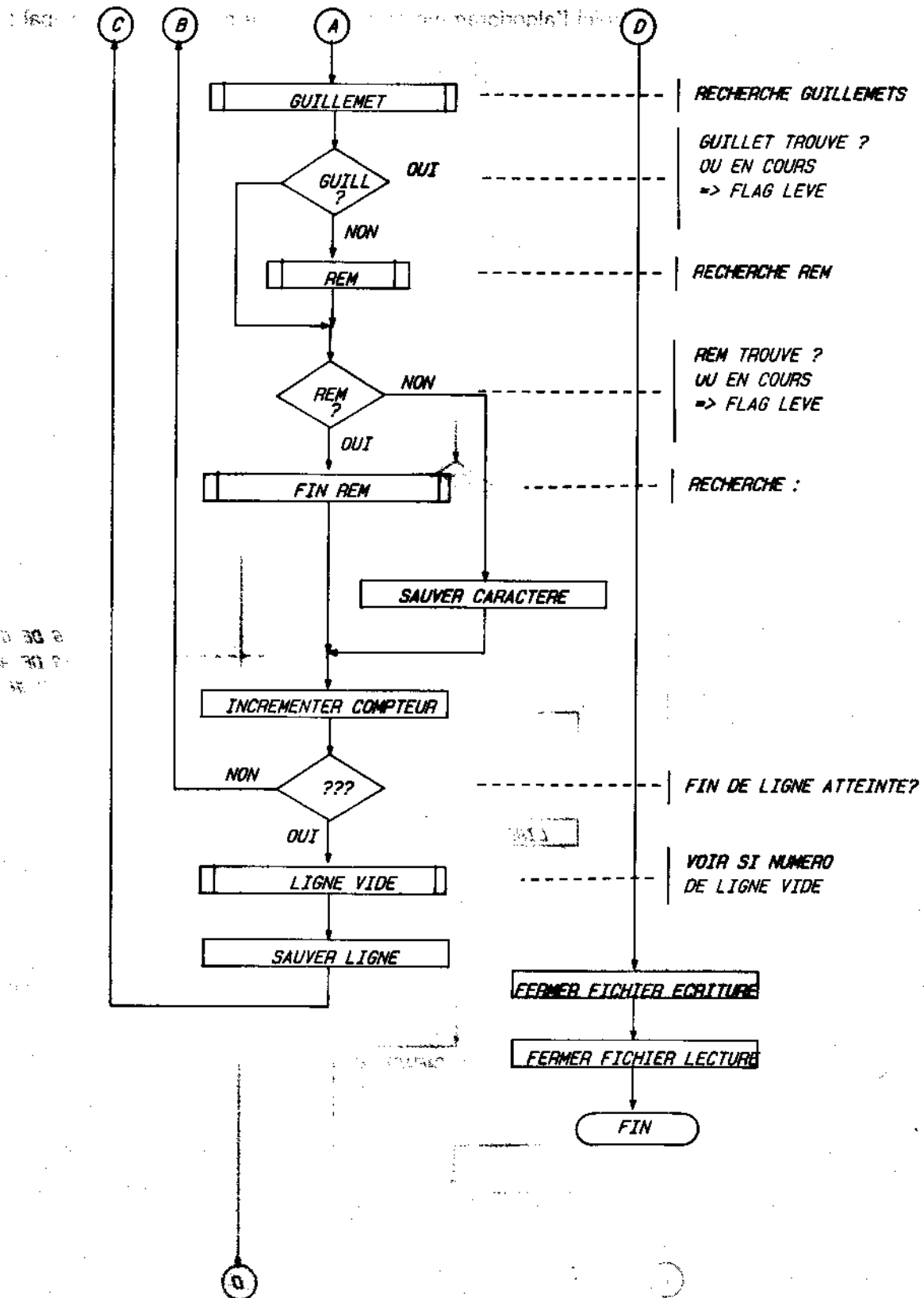
50 ON BREAK CONT

L'ALGORIGRAMME

L'étude des quatre cas précédents nous a permis de préciser le schéma général du programme, que nous avons décomposé en sous-programmes reconnaissant les différentes possibilités.

Voici l'algorithme proposé pour le programme principal :



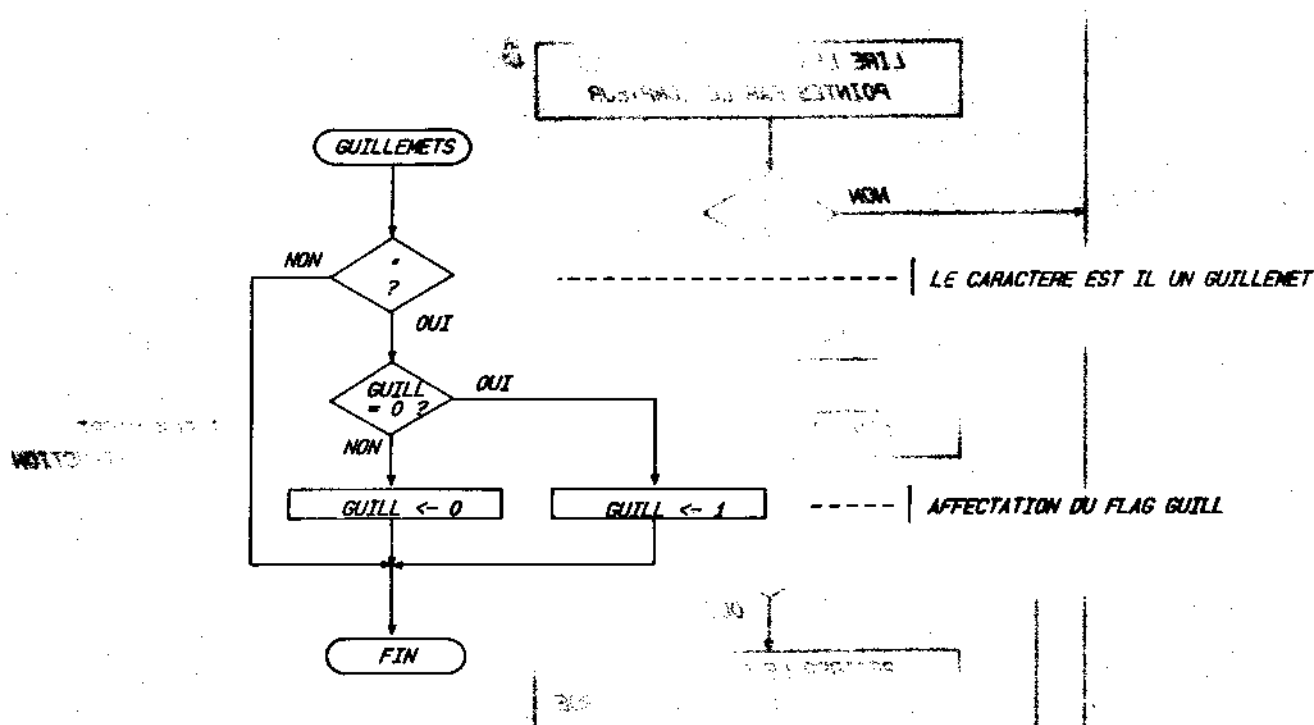


On s'aperçoit, qu'après avoir **demandé le nom du programme à traiter** (en ASCII) le programme va ouvrir un fichier temporaire en écriture pour y sauvegarder les lignes traitées.

Tant que la fin du fichier n'est pas atteinte, chaque ligne sera lue et traitée caractère par caractère. Le compteur est affiché pour signaler au programmeur que le programme n'est pas planté (dans le cas de programmes longs).

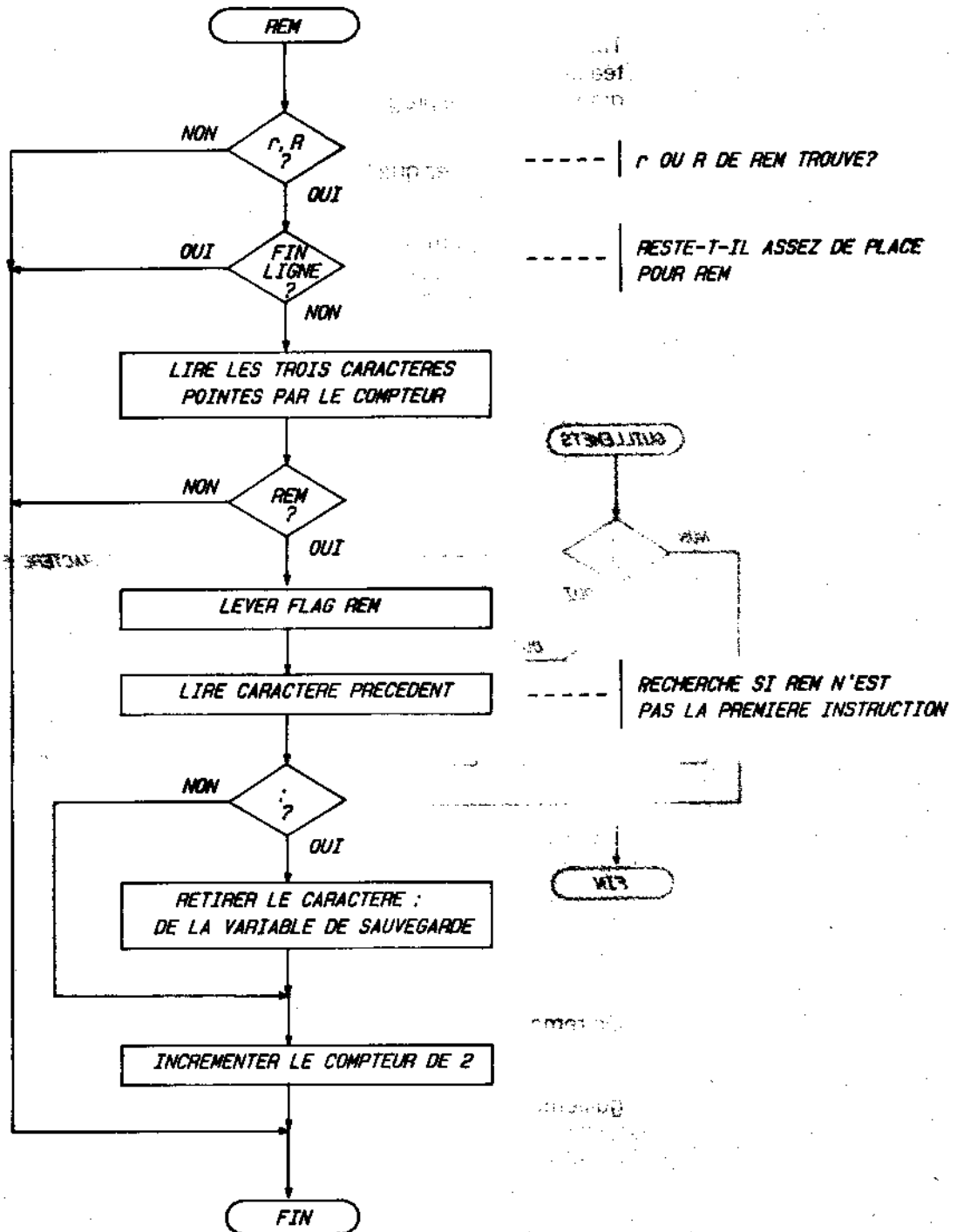
Vous remarquerez que lorsque l'instruction REM a été détectée, le compteur est incrémenté sans sauvegarde du caractère en cours de traitement.

Le sous-programme de recherche de guillemets aura la structure suivante :

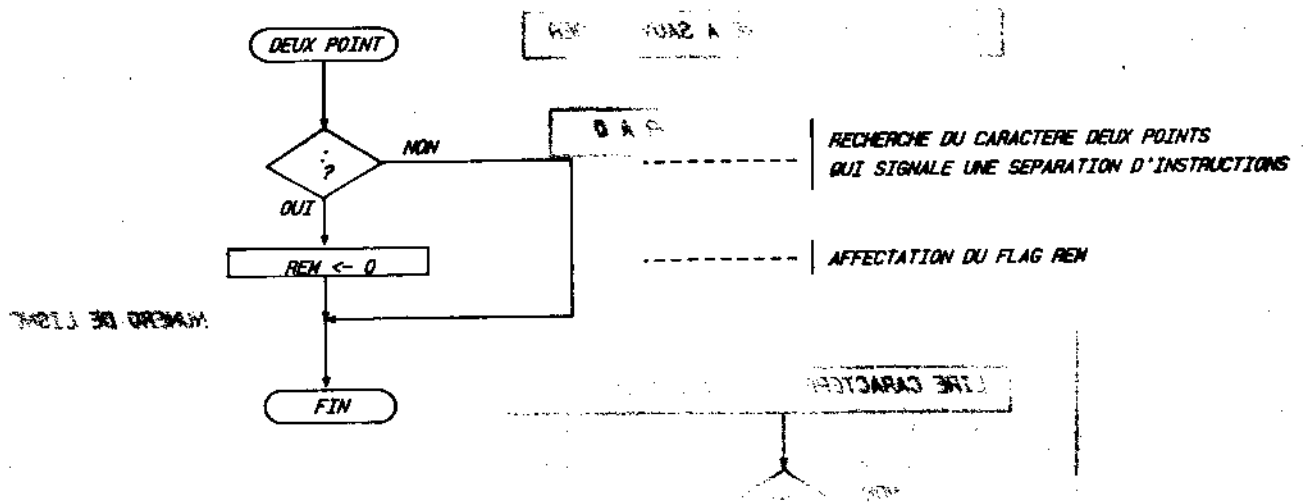


On remarquera que la variable FLAG (ou indicateur) est une variable que l'on pourrait appeler à bascule, puisqu'elle prendra la valeur 1 si un guillemet est détecté une première fois, puis 0 lorsque l'on refermera ces guillemets.

Le caractère **REM** sera traité selon l'algorithme suivant :



La reconnaissance des deux points signalant la fin d'une instruction REM aura la forme :



Enfin avant de sauvegarder chaque ligne, il faudra vérifier si elle est vide, ce qui sera réalisé par le sous-programme page suivante.

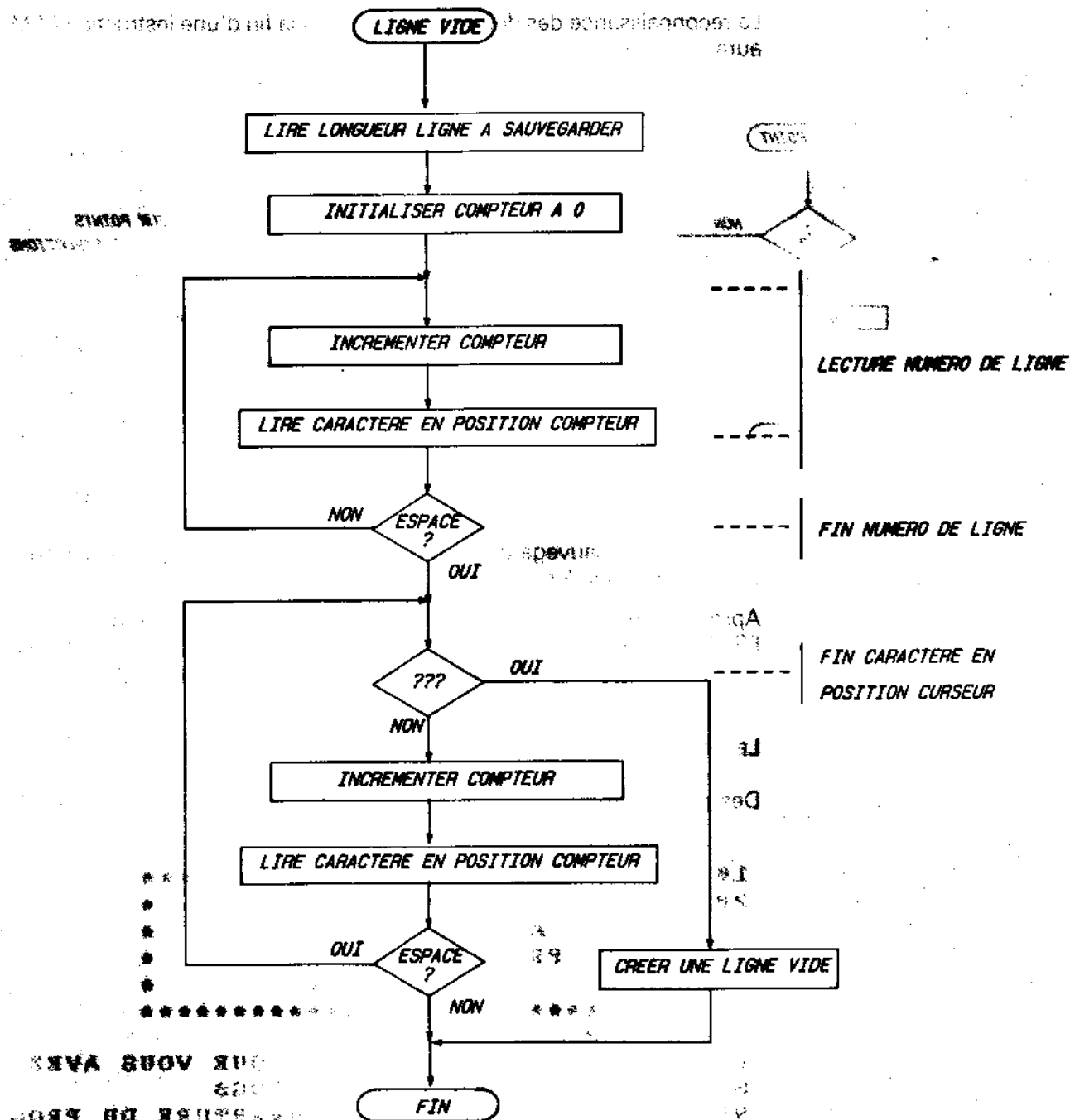
Après lecture du numéro de ligne, signalée par la présence du premier ESPACE de la ligne, le programme recherche si un caractère se présente avant la fin de la ligne, sinon il crée une variable de sauvegarde vide.

LE PROGRAMME

Des algorithmes précédents, voici le programme Basic commenté.

```

10 REM *****
20 REM *   PROGRAMME   ANTI-REM   *
30 REM *   A UTILISER SUR UN   *
40 REM *   PROGRAMME SAUVEGARDE *
50 REM *           EN ASCII     *
60 REM *****
70 MODE 2
80 INPUT "NOM DU PROGRAMME QUE VOUS AVEZ
SAUVEGARDE EN ASCII"; NOMPROGS
90 OPENIN NOMPROGS: REM OUVERTURE DU PROG
RAMME ASCII
100 OPENOUT "TEMP.ASC": REM PROGRAMME TEMP
ORAIRE A RENOMMER ENSUITE
110 WHILE NOT EOF
120   LINE INPUT #9, LIGNES: REM LECTURE L
IGNE
130   PRINT LIGNES
  
```



```

140      REM INITIALISATION VARIABLES
150      GUILL=0
160      SAUVS=""
170      FLAGREM=0
180      LONG = LEN(LIGNES)
190      COMPT =1
200      AS = MIDS(LIGNES,COMPT,1)
210      PRINT COMPT;CHRS(13);
220      GOSUB 420:REM VOIR SI GUILLEMETS
230      IF GUILL = 1 THEN 250
240      GOSUB 480:REM VOIR SI REM?
250      IF FLAGREM = 1 THEN GOSUB 630:GOT
O 270
260      SAUVS = SAUVS + AS
270      COMPT = COMPT + 1:REM CARACTERE S
Uivant
280      IF COMPT <= LONG THEN 200: A ALL
E TRAITER
290      GOSUB 690:REM VOIR SI LIGNE VIDE
300      PRINT #9,SAUVS
310      PRINT SAUVS+SPACES(10)
320      WKND
330      CLOSEOUT
340      CLOSIN
350      PRINT"FICHER SAUVEGARDE"
360      PRINT"CHARGER PAR : LOAD ";CHRS(34);
"TEMP.ASC";CHRS(34)
370      PRINT"PUIS SAUVEGARDE AU NOM CHOISI"
380      STOP
390      REM
400      REM *** RECONNAISSANCE GUILLEMETS **
*
410      REM
420      IF AS <> CHRS(34) THEN GOTO 440
430      IF GUILL = 0 THEN GUILL = 1 ELSE GUI
LL = 0
440      RETURN
450      REM
460      REM *** RECHERCHE REM ***
470      REM
480      IF AS <> "R" AND AS <> "r" THEN RETU
RN
490      IF LONG - COMPT < 2 THEN RETURN
500      CS = UPPER$(MIDS(LIGNES,COMPT,3))
510      IF CS <> "REM" THEN RETURN
520      CS = UPPER$(MIDS(LIGNES,COMPT-1,1))
530      IF (CS <> ":" AND CS <> CHRS(32)) TH
EN RETURN
540      FLAGREM = 1
560      IF CS <> ":" THEN 580
570      SAUVS = LEFT$(SAUVS,COMPT-2)

```

```

580 COMPT = COMPT + 2
590 RETURN
600 REM
610 REM *** RECHERCHE SEPARATEUR ***
620 REM
630 IF AS <> " " THEN RETURN
640 FLAGREM = 0
650 RETURN
660 REM
670 REM *** TRAITEMENT LIGNE VIDE ***
680 REM
690 LONG = LEN(SAUVS)
700 COMPT = 0
710 COMPT = COMPT + 1
720 AS = MIDS(SAUVS,COMPT,1)
730 IF AS <> CHR$(32) THEN 710
740 IF COMPT = LONG THEN SAUVS = " ":RETURN
750 COMPT = COMPT + 1
760 AS = MIDS(SAUVS,COMPT,1)
770 IF AS <> CHR$(32) THEN RETURN
780 GOTO 740

```

Le listing ci-dessus considère qu'aucun saut n'est effectué à une ligne contenant uniquement une instruction REM, donc élimine ces dernières sans pitié.

Au cas où le programme contient des sauts à des adresses de commentaires, vous pouvez modifier la ligne contenant les instructions :

```

IF COMPT = LONG THEN SAUV = " ":RETURN
** BYNEMJLJUG ED par IF COMPT = LONG THEN SAUV = CHR$(39):RETURN

```

Nous allons récapituler la procédure d'amélioration de la rapidité de vos programmes :

- Sauvegardez une version originale sous forme ASCII par **SAVE "PROG1.ASC",A**
- Déclarez si possible toutes les variables selon leur utilisation.
- Regroupez les lignes pouvant l'être en une seule ligne.
- Sauvegardez le programme sous forme ASCII par **SAVE "PROG2.ASC",A**
- Entrez **POKE &AC00,1** puis chargez le programme par **LOAD "PROG2.ASC"**, que vous resauvegardez aussitôt sous le même nom, et en ASCII comme ci-dessus.
- Lancez la version que vous aurez choisi du programme supprimeur de REM (il faut obligatoirement un lecteur de disquette à partir de cette étape).
- Chargez la version temporaire du programme, qui est sauvegardée en ASCII (**LOAD "TEMP.ASC"**) et sauvegardez-la aussitôt en Basic sous le nom final (**SAVE "PROG.BAS"**).

4/2

Assembleur Z80 : Définitions et rappels de base

4/2.1

Pourquoi utiliser l'assembleur et dans quels domaines

Le circuit intégré principal de votre ordinateur est le microprocesseur Z80. Il émet des commandes visibles ou invisibles (pour l'utilisateur) vers ses circuits périphériques. Grâce à lui, vous pouvez :

- entrer une commande ou un programme au clavier,
- voir du texte ou des graphismes s'afficher sur l'écran,
- dialoguer avec des périphériques, etc.

Les microprocesseurs ne comprennent qu'un langage : le binaire. Le Z80 qui est un microprocesseur 8 bits (les données manipulées sont codées sur 8 bits : de 0 à 255) est activé par des instructions en langage machine codées sur 1, 2, 3 ou 4 octets.

Les langages dits « évolués » (BASIC, LOGO, PASCAL, FORTH, etc.) sont composés d'un ensemble de mots-clés de haut niveau. Chaque mot-clé peut correspondre à un assemblage de plusieurs dizaines d'instructions en langage machine.

Pour être exécutables, les programmes écrits dans ces langages doivent être traduits en langage machine. Deux solutions sont possibles :

- la traduction est faite instruction par instruction, au moment où l'instruction est exécutée : on parle alors de *langage interprété*.
- la traduction est faite avant l'exécution : on parle alors de *langage compilé*.

La deuxième solution, quoique plus lourde à mettre en œuvre, procure des temps d'exécution plus courts que la première. En effet, la phase de traduction n'est pas faite pendant mais avant l'exécution.

Définition :

On appelle « taux d'expansion » le rapport entre le nombre de codes en langage machine produits par le compilateur et le nombre de codes en langage machine nécessaires pour produire l'action demandée.

Quel que soit le compilateur utilisé, il aura un **taux d'expansion supérieur à 1**. En effet, les ordres évolués d'un langage ont souvent plusieurs significations possibles. Par exemple, en BASIC, l'ordre « PRINT » peut envoyer des informations sur l'écran ou sur une imprimante. Les séquences d'ordres générés en langage machine ne sont pas toujours optimisées. Pour cela, il faudrait différencier chaque utilisation possible d'un même ordre, et lui affecter un traitement spécifique ; ce qui augmenterait sensiblement la taille du compilateur.

Malgré l'utilisation de compilateurs à **faible taux d'expansion**, certaines tâches ont besoin d'être exécutées très rapidement, et la seule solution pour les réaliser consiste à les écrire directement en langage machine, ou à utiliser un assembleur.

Qu'est-ce qu'un assembleur ?

C'est un compilateur à **taux d'expansion unitaire** qui traduit des **codes opératoires** en binaire codé en hexadécimal. Programmer en assembleur est strictement équivalent à manipuler des codes en binaire, à ceci près que l'utilisation de codes opératoires facilitera grandement la tâche du programmeur.

4/2.2

Les modes d'adressage

Le Z80 est un microprocesseur 8 bits capable de faire des opérations logiques et arithmétiques sur 8 ou 16 bits avec ses registres internes, ou avec une mémoire de taille maximum 64 kilo-octets.

Les registres sont des mémoires de 8 ou 16 bits internes au microprocesseur. Ils sont identifiés par une lettre s'ils font 8 bits et par deux lettres s'ils font 16 bits. On les appelle alors *registres pairs*.

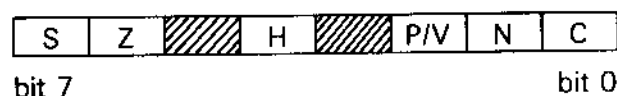
Les registres disponibles portent les noms suivants :

A, F, B, C, (ou BC), D, E, (ou DE), H, L, (ou HL), IX, IY, SP, I, R pour les registres primaires,

A', F', BC', DE' et HL' pour les registres secondaires.

Le registre A ou « Accumulateur » est celui qui est le plus souvent utilisé, et son accès est généralement plus rapide que celui à un autre registre. Les opérations arithmétiques, logiques et comparaisons font souvent appel à lui.

Le registre F contient les indicateurs (flags) qui donnent des renseignements sur la dernière opération effectuée.



BIT 0 : C = Carry (retenue)

Positionné par les opérations arithmétiques, décalages, comparaisons et les instructions SCF et CCF.

Les opérations logiques AND, OR et XOR le mettent à 0.

BIT 1 : N = Negative (négatif)

Ce bit est à 1 si l'opération précédente était une soustraction ou une décrémentation.

BIT 2 : P/V = Parity/Overflow (parité/débordement)

Ce bit joue un double rôle en fonction du type d'instruction utilisée :

P est positionné par les instructions logiques :

- il vaut 1 si le nombre de bits à 1 de A est pair,
- il vaut 0 si le nombre de bits à 1 de A est impair,

V est positionné par les instructions arithmétiques :

- il vaut 1 si un débordement s'est produit.

BIT 4 : H = Half-Carry (demi-retenue)

Utilisé par l'instruction d'ajustement décimal DAA.

Reportez-vous à cette instruction pour avoir plus de détails.

H=1 signale qu'une retenue s'est produite sur le LSQ (Last Significant Quartet = quartet de poids faible) du registre A.

BIT 6 : Z = Zéro

Positionné par toutes les instructions qui peuvent produire un résultat nul (AND, BIT, CP, DEC, etc.)

Positionné à 1 par les instructions INDR, INIR, OTDR et OTIR.

BIT 7 : S = Signe

Donne le signe de la valeur testée.

Positionné par les opérations logiques, arithmétiques, rotations et décalages.

Si S=1, la valeur est négative (bit 7 à 1 pour les valeurs sur 8 bits et bit 15 à 1 pour les valeurs sur 16 bits).

Si S=0, la valeur est positive (bit 7 à 0 pour les valeurs sur 8 bits et bit 15 à 0 pour les valeurs sur 16 bits).

Le registre B peut être associé au registre C pour former le registre pair BC. Mais il est manipulé comme registre 8 bits par les instructions du type :

- Faire une action, décrémenter B, et répéter l'action tant que B n'est pas nul, comme DJNZ, par exemple.

Les registres C, D, E, H et L peuvent être utilisés sous forme de registres pairs BC, DE et HL. Ils permettent de faire des opérations sur 8 ou 16 bits avec ou sans retenue.

Les registres IX et IY sont des registres 16 bits. Ils peuvent être utilisés comme les registres pairs BC, DE ou HL, mais sont obligatoires dans certains modes d'adressages indexés.

Le registre SP (Stack Pointer ou pointeur de pile) donne l'adresse de l'élément mémoire le plus extérieur à la pile.

Le registre I (ou IFF) donne le poids fort de l'adresse où va se débrancher le programme en cas d'interruption.

Le registre R donne l'adresse de rafraîchissement dynamique des blocs de mémoire RAM. Nous n'en parlerons pas dans ce manuel, car il est quasiment inutilisé en programmation.

Modes d'adressage

Le terme « mode d'adressage » désigne la façon qui va être utilisée par le microprocesseur pour accéder à une information.

Le Z80 possède 7 modes d'adressage :

Immédiat, registre, indirect sur registre, direct, relatif, indirect indexé et bit.

Examinons en détails chacun de ces modes :

- Adressage immédiat

La valeur se trouve dans l'instruction.

Exemple : LD A, 10H
ou OR 45H

LD A, 10H Registre A ← 10H

OR 45H A ← A OR 45H

- Adressage registre

Les opérations se font entre registres internes sans impliquer la mémoire.

Exemple : LD A, C
ou CP B

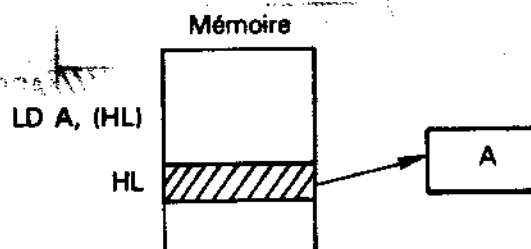
LD A, C A ← C

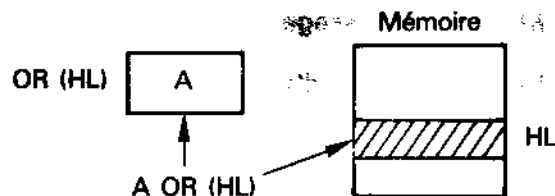
CP, B A B
) Comparaison

- Adressage indirect sur registre

Le registre pair spécifié pointe sur une mémoire et l'opération est faite sur cette mémoire.

Exemple : LD A, (HL)
ou OR (HL)

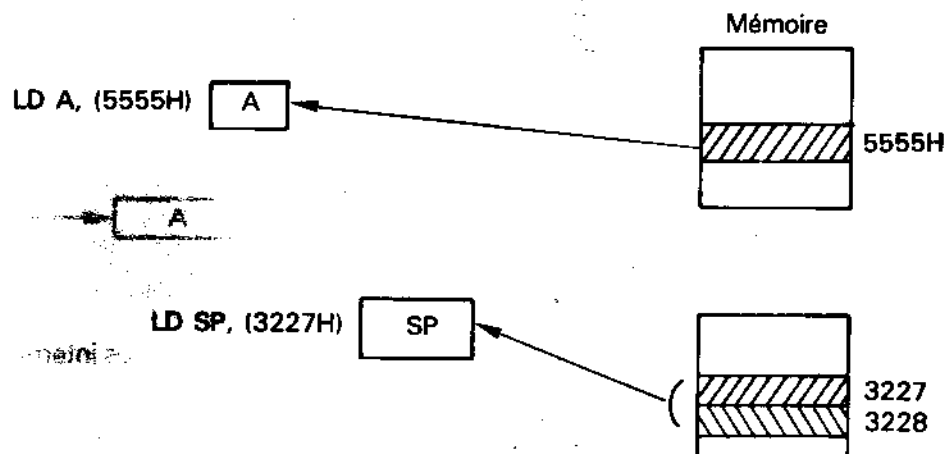




- Adressage direct

Une adresse mémoire spécifiée dans l'instruction pointe sur la donnée qui sera manipulée.

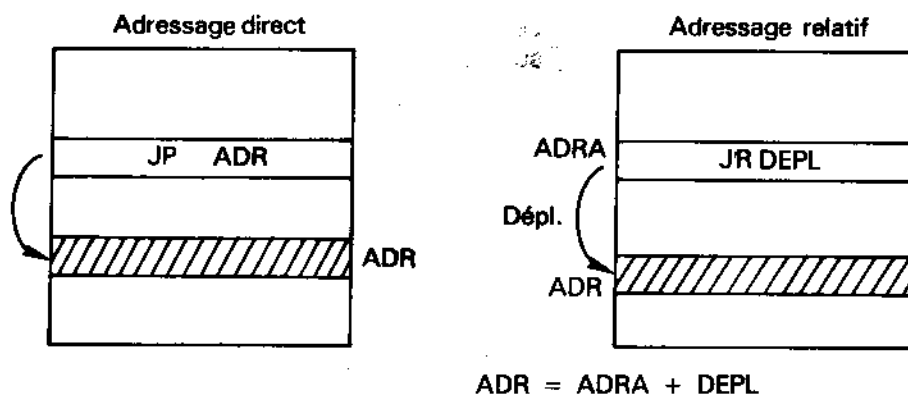
Exemple : LD A,(5555H)
ou LD SP,(3227H)



- Adressage relatif

Concerne les instructions de saut du type JR.

Quand l'adresse à laquelle doit s'effectuer le débranchement est assez proche (entre + 127 et - 128 octets) de l'adresse courante, ces instructions peuvent être employées pour « économiser » un octet en codage. En effet, une instruction JP est codée sur 3 octets, alors que l'instruction JR équivalente est codée sur 2 octets.

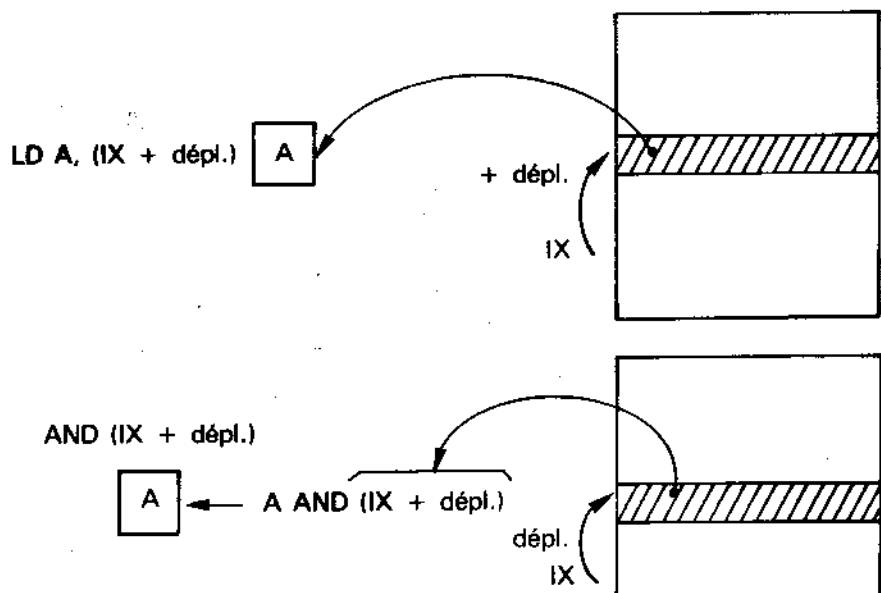


Remarque :

L'utilisation d'un ordre JR pour accéder à un label situé hors des limites permises (entre +127 et -128 octets) produira une erreur lors de l'assemblage du programme.

• Adressage indirect indexé

Accès à une information d'adresse IX + déplacement ou IY + déplacement de la forme LD A, (IX + dépl.)
ou AND (IX + dépl.)

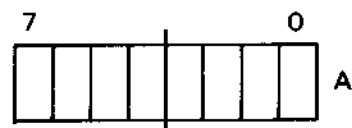


• Adressage bit

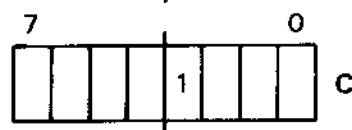
Positionnement ou test d'un bit :

Les instructions « BIT » testent un bit,
les instructions « SET » mettent un bit à 1,
et les instructions « RES » mettent un bit à 0.

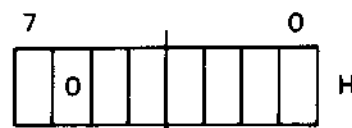
BIT 1, A donne la valeur
du bit 1 du registre A



SET 3, C met le bit 3 du
registre C à 1



RES 6, H met le bit 6
du registre H à 0



4/2.3

Les mots-clés de l'assembleur Z80 et leur utilisation

Le nombre de codes opératoires (d'instructions) du Z80 est assez important, et nous avons pensé qu'il serait judicieux de les répartir en plusieurs groupes pour faciliter leur utilisation.

Vous trouverez donc, dans les pages qui suivent, l'analyse détaillée et illustrée de chaque code opératoire. Les codes-op sont classés par groupes de fonctions :

- usage général et interruptions,
- lecture et écriture en mémoire,
- lecture, écriture, échanges et opérations sur les registres,
- ruptures de séquences,
- opérations arithmétiques,
- opérations logiques,
- manipulation de bits et de chaînes,
- opérations sur la pile,
- entrées/sorties.

Les codes opératoires examinés dans chaque groupe sont les suivants :

I. Usage général et interruptions :

NOP, HALT, EI, DI, IM 0, IM 1, IM 2.

II. Lecture et écriture en mémoire :

LD, LDI, LDD.

Remarque :

Une liste des codes opératoires, et leur fonction résumée est proposée dans le chapitre 2.4 de la partie 4. Les références des pages où ils sont mentionnés sont précisées. La liste alphabétique des codes-op et leur codage en hexadécimale se situe en Annexe 2 de la partie 11.

III. Lecture, écriture, échanges et opérations sur les registres :

LD, EX, EXX

IV. Ruptures de séquences :

JP, JR, DJNZ, CALL, RET, RETI, RETN, RST

V. Opérations arithmétiques :

Additions : ADD, ADC.

Soustractions : SUB, SBC.

Incrémentation et décrémentation : INC, DEC, DAA.

VI. Opérations logiques :

Opérations logiques élémentaires : AND, OR, XOR.

Comparaison et tests : CPL, NEG, CP, CPI, CPD.

Rotations et décalages : RLC, RRC, RLA, RRA, RL, SLA, SRL, RLD, RRD, SCF, CCF.

VII. Manipulation de bits et de chaînes :

Instructions sur bits : BIT, RES, SET.

Instructions sur chaînes : LDIR, LDDR, CPIR, CPDR.

VIII. Opérations sur la pile :

PUSH, POP

IX. Entrées/sorties

OUT, IN, INI, INIR, IND, INDR, OUTI, OUTD, OTDR, OTIR.

I. Usage général et interruptions**NOP : N° opération**

Indicateurs modifiés : aucun.

Aucune action n'est effectuée, si ce n'est une perte de temps pour analyser le code et l'exécuter.

Applications :

C'est une instruction idéale pour créer des temporisations.

Sachant qu'une instruction NOP prend 1 cycle machine (voir table en annexe) et que, sur AMSTRAD, un cycle machine prend 0,25 microsecondes, un NOP est donc exécuté en 0.25 microsecondes.

Nous pourrions réaliser des temporisations de courte durée en mettant à la suite plusieurs NOP.

1	ORG	9000H
2	LOAD	9000H

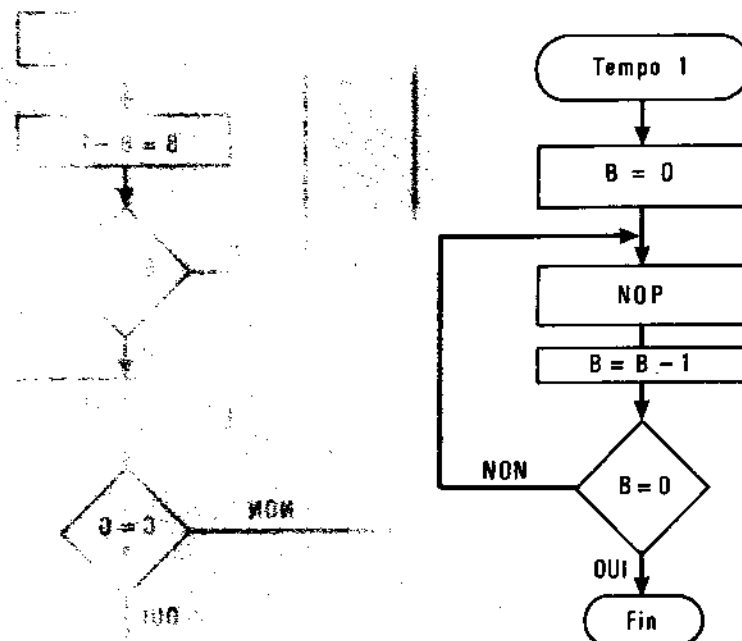
```

3 9000 00    NOP    ;0.25 microsec passees
4 9001 00    NOP    ;0.25 microsec passees
5 9002 00    NOP    ;0.25 microsec passees
6            END
    
```

Pour réaliser des temporisations plus longues, nous pourrions insérer une ou plusieurs instruction(s) NOP dans une boucle.

Reportez-vous à la description de l'ordre DJNZ si vous ne connaissez pas son utilisation (voir partie 4 chapitre 2.3 p. 32).

LD B,0 prend 2 cycles donc 0.5 microsecondes.
DJNZ prend 2 cycles donc 0.5 microsecondes.



```

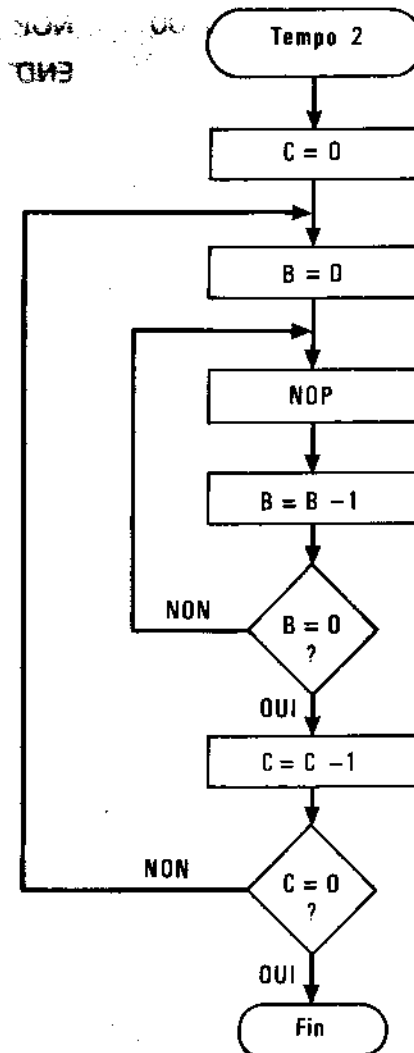
1          ORG      9000H
2          LOAD     9000H  ORG
3 9000 0600    LD B,0      ; Nombre de boucles
4          BOUCLE:  EQU $
5 9002 00      NOP        ; histoire de passer le temps
6 9003 10FD    DJNZ     BOUCLE
7          END
    
```

La temporisation fera :

LD B,0 &FF*(NOP+DJNZ)

2 cycles + &FF*(1+2) cycles = 767 cycles soit 383.5 microsecondes.

Pour réaliser des temporisations encore plus longues, nous pourrions imbriquer deux ou plus de deux boucles de la façon suivante :



1		ORG	9000H	
2		LOAD	9000H	
3	9000 0E00	LD	C, 0	; Nombre de boucles *FF
4	B1:	EQU	\$	
5	9002 0600	LD	B, 0	; Nombre de boucles
6	B2:	EQU	\$	
7	9004 00	NOP		; Attente
8	9005 10FD	DJNZ	B 2	; 1 ^{re} boucle
9	9007 0D	DEC	C	; C = C - 1
10	9008 20FB	JR	NZ, B 1	; 2 ^e boucle
11		END		

La temporisation fera :

```
LD C,0      &FF*(LD B,0 + &FF*(NOP + DJNZ) + DEC C + JP NZ,@)
2 + &FF*(2 + &FF*(1 + 2) + 1 + 3)
```

19 607 cycles, soit 98.3 millisecondes, soit environ 0,1 sec.

HALT

Indicateurs modifiés : aucun.

Arrête le déroulement du programme qui ne pourra être relancé que par une action matérielle (interruption ou RESET).

Remarque :

Les RAM dynamiques ne sont plus rafraîchies suite à l'exécution de cet ordre, et les informations qu'elles contiennent peuvent être perdues.

INTERRUPTIONS

• Avant de voir la fonction des ordres dédiés aux interruptions, nous allons définir ce qu'est une interruption. Pour mieux cerner le problème, nous pouvons faire un parallèle avec une situation similaire mais peut-être plus courante pour vous.

Supposez que vous êtes en train de lire un livre, et que le téléphone se met à sonner. Vous pouvez réagir de trois manières différentes :

- vous pouvez ignorer le téléphone et continuer votre lecture ;
- vous pouvez finir la phrase que vous êtes en train de lire, puis décrocher le téléphone ;
- vous pouvez arrêter immédiatement votre lecture et décrocher le téléphone.

Dans le deuxième et le troisième cas, vous reprendrez sans doute la lecture du livre quand la conversation téléphonique sera finie.

Dans un ordinateur, les interruptions sont interprétées par le microprocesseur de la même manière que la sonnerie du téléphone dans l'exemple ci-dessus, et la lecture du livre correspond à l'exécution d'un programme.

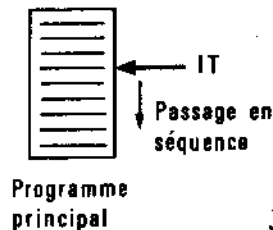
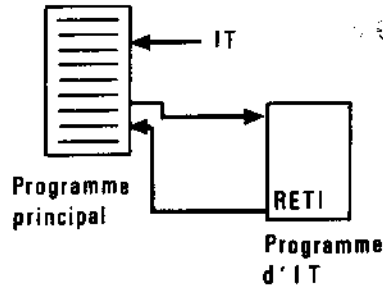
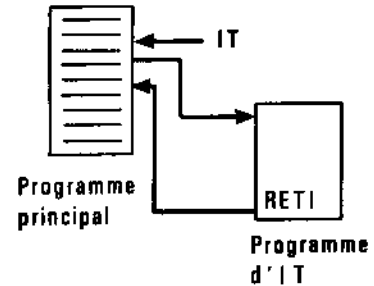
Si le programme est ininterrompible (cas N° 1 de notre exemple), l'interruption ne sera pas prise en compte et le programme continuera son exécution sans aucun arrêt.

Si le programme est interrompible, selon la priorité de l'interruption, il sera interrompu immédiatement (cas n° 3 de notre exemple), ou après un certain temps (cas n° 2 de notre exemple) — Voir page suivante —.

• Nous savons ce qu'est une interruption, mais pas encore ce qui peut la provoquer.

En général, tous les périphériques de l'ordinateur peuvent émettre des interruptions dès qu'ils ont besoin :

- de la puissance de calcul du microprocesseur,
- d'accéder à la mémoire,
- de communiquer avec un autre périphérique.

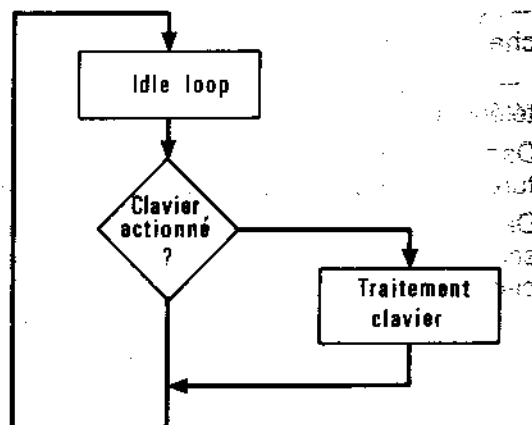
1^{er} cas : Interruption ignorée2^e cas : IT acceptée avec délai3^e cas : IT acceptée immédiatement

Deux méthodes peuvent cependant être employées par le microprocesseur pour savoir si un périphérique a besoin de lui.

Première méthode : Intervention dans l'Idle Loop

Quand il n'a « rien de spécial à faire », le microprocesseur exécute une série d'instructions qui servent à rafraîchir la RAM, ou à tester si un périphérique sollicite sa présence. Cette série d'instructions porte le nom d'« Idle Loop » ou boucle d'attente principale.

Le test d'activation du clavier pourra être fait dans l'Idle Loop de la manière suivante :



Deuxième méthode : Attente d'une interruption

Le microprocesseur se trouve soit dans l'Idle Loop, soit en train d'exécuter un programme. L'équipement qui requiert le microprocesseur émet une interruption. Reprenons l'exemple du clavier. Quand une touche est pressée, une interruption est générée, et selon l'importance (appelée « priorité ») du programme en cours d'exécution, l'appui sur la touche sera pris en compte immédiatement, sous un certain délai, ou pas du tout (par exemple, lors de l'accès au lecteur de cassettes ou de disquettes).

- Avant de passer à l'utilisation des ordres d'interruptions, il reste à définir les divers types d'interruptions que peut gérer le Z80.

Quatre pattes du circuit intégré Z80 sont dédiées aux interruptions. Il s'agit de NMI, INT, BUSRQ et RESET. Quand le niveau de tension passe de 5 volts à 0 volt ou quand un niveau 0 volt est détecté sur l'une de ces pattes, une interruption est générée.

Interruptions non masquables :

Ces interruptions sont prises en compte immédiatement (à la fin du cycle d'instructions en cours), quel que soit le programme en cours d'exécution. Il s'agit des interruptions NMI.

Les actions suivantes se produisent :

- empilement du PC (Program Counter = Pointeur d'adresse de l'instruction en cours d'exécution) ;
- sauvegarde de l'état de masquage d'interruption (IFF : c'est l'information qui indique si le programme en cours d'exécution est ou n'est pas interruptible) ;
- interdiction de prise en compte d'autres interruptions pendant le traitement de l'interruption NMI par action sur l'information de masquage d'interruption IFF ;
- exécution du programme d'interruption situé en &0066.

La fin d'interruption non masquable est repérée par l'ordre RETN qui permettra de revenir à l'état de masquage d'interruption précédant l'activation de la routine d'interruption NMI.

Remarque :

Si l'instruction RET est rencontrée, la routine de traitement d'interruption sera considérée comme finie, et le masque d'interruption ne sera pas restauré.

Interruptions masquables :

La prise en compte des interruptions de type INT peut être masquée (DI : Disable Interrupts) ou autorisée (EI : Enable Interrupts) par programme.

Contrairement à la patte NMI qui provoque une interruption sur front descendant (5 V → 0 V), la patte INT provoquera une interruption sur niveau bas. La tension sur cette patte devra être 0 volt jusqu'à ce que le Z80 décide de la traiter, et devra être mise à 5 volts quand l'interruption aura été traitée. Sinon, une autre interruption sera générée.

EI : Enable Interrupts

Cette instruction permet de valider la prise en compte des interruptions masquables de type INT.

DI : Disable Interrupts

Cette instruction permet de dévalider la prise en compte des interruptions masquables de type INT.

IM 0 : Interrupt Mode 0

Permet de valider le mode d'interruption 0.

Ce mode est actif à la mise sous tension du microprocesseur, quand une commande de remise à zéro du Z80 est émise, ou quand l'ordre IM 0 est rencontré.

Si la demande d'interruption (IT) est acceptée (IT validées par EI et aucune interruption en cours) :

- les demandes d'interruptions sont inhibées et l'état de masquage des interruptions est écrasé (IFF=0),
- les pattes $\overline{M1}$ et $\overline{IORQ}$ sont mises à 0. Ce qui provoque l'émission vers le Z80 d'un code opératoire de type « RST 0 » à « RST 7 » ou « CALL Adresse »,
- le Z80 saute à l'emplacement correspondant à l'ordre ReStart qu'il a reçu,

RST	Adresse du débranchement
0	&0000
1	&0008
2	&0010
3	&0018
4	&0020
5	&0028
6	&0030
7	&0038

ou à l'adresse spécifiée dans les deux octets (poids faible, poids fort) qui suivent le code opératoire CALL.

IM 1 : Interrupt Mode 1

Permet de valider le mode d'interruption 1.

Si la demande d'IT est acceptée (IT validées par EI et aucune interruption en cours) :

- les demandes d'interruptions sont inhibées et l'état de masquage des interruptions est écrasé (IFF=0),
- le PC est stocké dans la pile,
- le Z80 exécute la routine d'interruption située en &0038.

La routine d'interruption doit se terminer par l'instruction RET ou RETI.

IM 2 : Interrupt Mode 2

Permet de valider le mode d'interruption 2.

Ce mode est actif quand l'ordre IM 2 est rencontré.

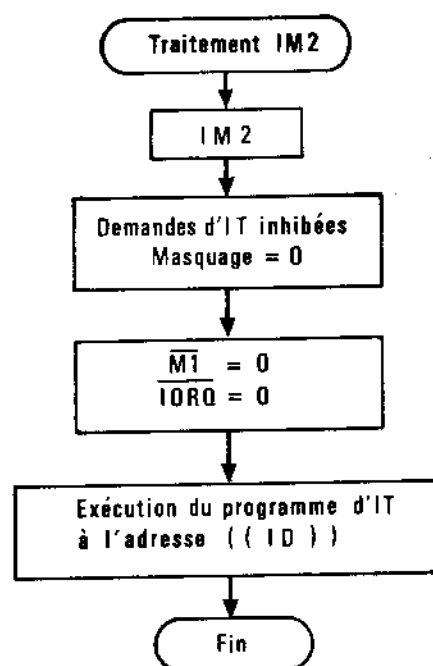
Si la demande d'IT est acceptée (IT validées par EI et aucune interruption en cours) :

- les demandes d'interruptions sont inhibées et l'état de masquage des interruptions est écrasé (IFF = 0),

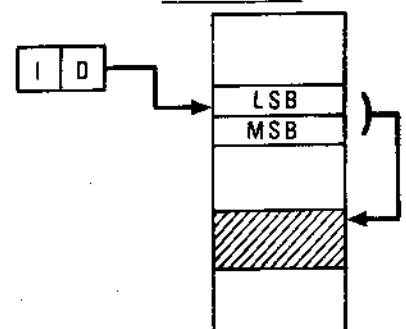
- les pattes $\overline{M1}$ et $\overline{IORQ}$ sont mises à 0. Ce qui provoque l'émission d'un octet sur le bus de données. Le Z80 forme un mot de 16 bits où les 8 bits de poids fort sont constitués par le registre I, et les 8 bits de poids faible par l'octet émis sur le bus de données.

Le contenu de l'adresse constituée par le mot de 16 bits précédent donne l'adresse à laquelle se trouve le programme de gestion d'interruptions.

Ce mode d'indirections pour trouver l'adresse d'exécution de la routine d'IT peut sembler complexe, mais il a l'avantage de permettre d'exécuter une routine située à un endroit quelconque en mémoire.



Exécution du programme d'interruption :



LSB = Last significative byte
= poids faible

MSB = Most significative byte
= poids fort

 Programme de traitement d'interruption

II. Lecture et écriture en mémoire

Sur le Z80, la lecture et l'écriture en mémoire sont réalisées par le même code opératoire. Il s'agit de LD (Load) et de ses variantes LDI et LDD.

Reportez-vous à la partie 4 chapitre 2.2 pour être au courant des modes d'adressage utilisés par le Z80, car ce qui suit fait largement appel à ces notions.

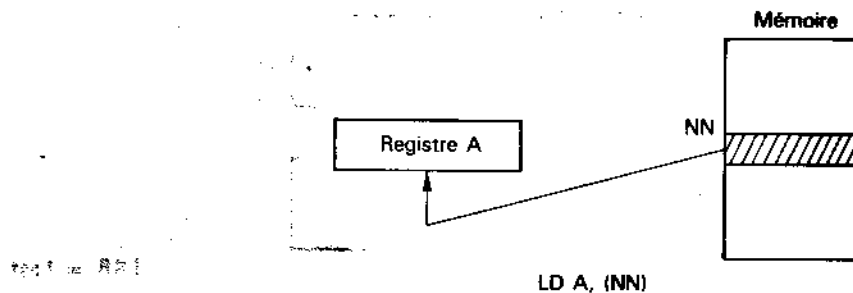
Conventions d'écriture :

- NN** représente une donnée ou une adresse sur 16 bits,
N représente une donnée sur 8 bits,
(VAR) indique que la donnée manipulée est le contenu de VAR.
d représente un offset sur 8 bits, c'est-à-dire une valeur qui est ajoutée à une adresse absolue pour obtenir un adressage dit indexé.
VAL8 représente une valeur quelconque sur 8 bits.
VAL16 représente une valeur quelconque sur 16 bits.

- Adressage direct sur 8 bits :

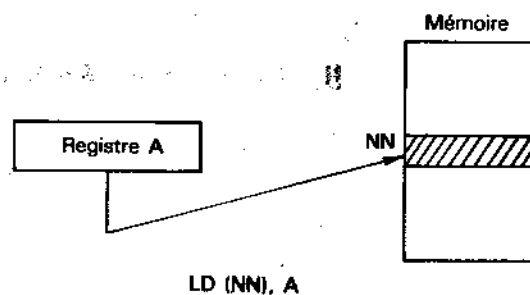
Lecture en mémoire : **LD A,(NN)**

Les indicateurs ne sont pas affectés



Ecriture en mémoire : **LD (NN),A**

Les indicateurs ne sont pas affectés

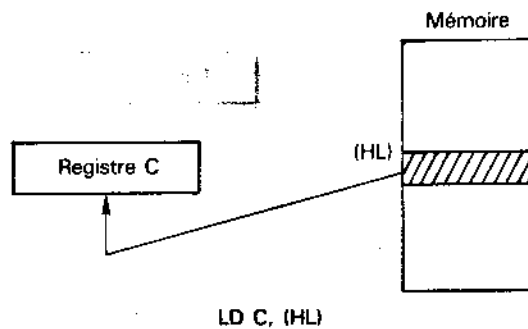


- Adressage indirect sur 8 bits par l'intermédiaire de HL :

Lecture en mémoire : **LD X, (HL)**

où X peut être un des registres suivants :
A, B, C, D, E, H, L.

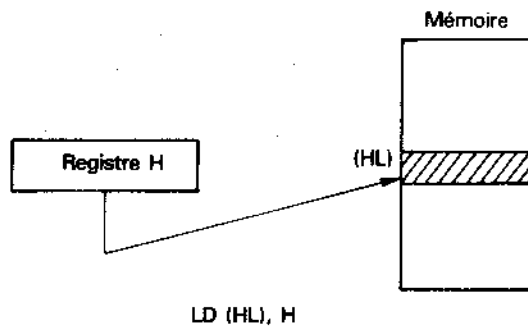
Les indicateurs ne sont pas affectés.



Ecriture en mémoire : **LD (HL), X**

où X peut être un des registres suivants :
A, B, C, D, E, H, L.

Les indicateurs ne sont pas affectés.

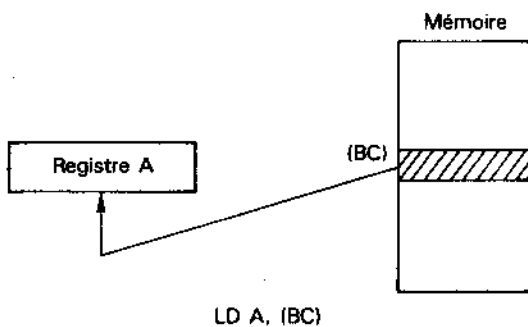


- Adressage indirect sur 8 bits par l'intermédiaire de BC ou DE :

Seul le registre A est utilisable à cet effet.

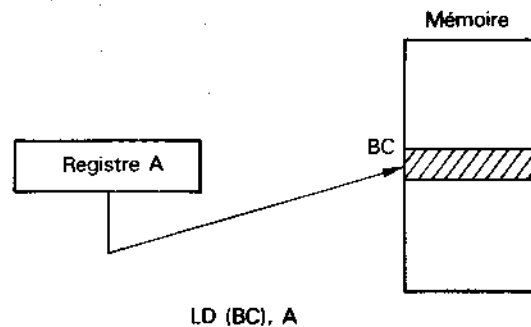
Lecture en mémoire : **LD A, (BC) et LD A, (DE)**

Les indicateurs ne sont pas affectés.



Ecriture en mémoire : LD (BC),A et LD (DE),A

Les indicateurs ne sont pas affectés.

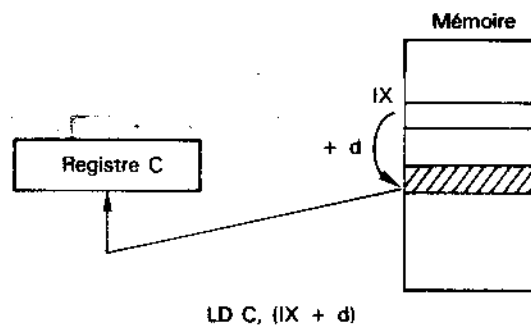


- Adressage indirect indexé sur 8 bits :

Lecture en mémoire : LD X, (IX+d) ou LD X, (IY+d)

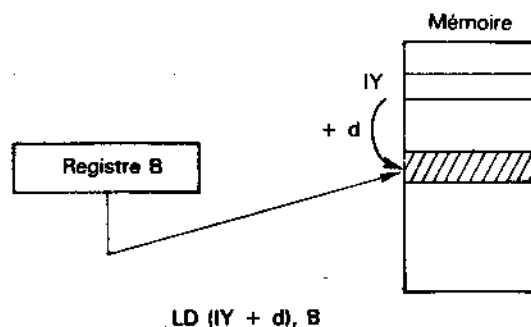
où X peut être un des registres suivants :
A, B, C, D, E, H, L.

Les indicateurs ne sont pas affectés.

**Ecriture en mémoire : LD (IX+d),X ou LD (IY+d),X**

où X peut être un des registres suivants :
A, B, C, D, E, H, L.

Les indicateurs ne sont pas affectés.

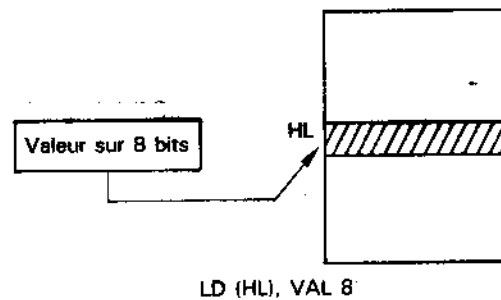


- Adressage immédiat indirect sur 8 bits :

Aucun ordre de lecture en mémoire n'est prévu dans ce mode d'adressage.

Ecriture en mémoire : **LD (HL), VAL8**

Les indicateurs ne sont pas affectés.

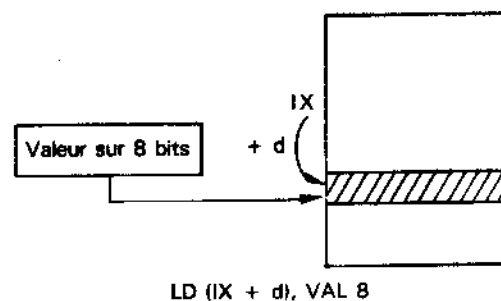


- Adressage immédiat indirect indexé sur 8 bits :

Aucun ordre de lecture en mémoire n'est prévu dans ce mode d'adressage.

Ecriture en mémoire **LD (IX + d), VAL8** et **LD (IY + d), VAL8**

Les indicateurs ne sont pas affectés.

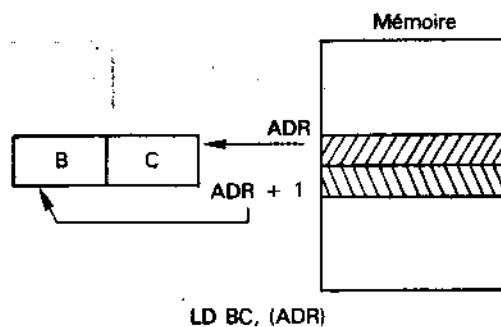


- Adressage direct sur 16 bits :

Lecture en mémoire : **LD XX, (ADR)**

où XX est un des registres pairs suivants : BC, DE, HL, SP, IX, IY.

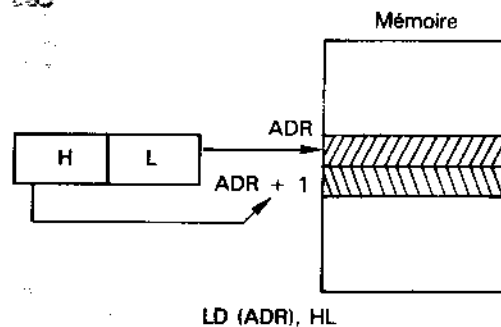
Les indicateurs ne sont pas affectés.



Ecriture en mémoire : **LD (ADR),XX**

où XX est un des registres pairs suivants :
BC, DE, HL, SP, IX, IY.

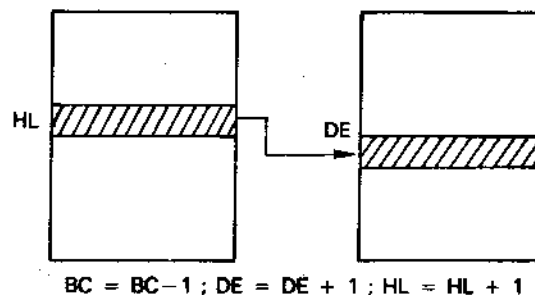
Les indicateurs ne sont pas affectés.



- Adressage indirect de mémoire à mémoire :

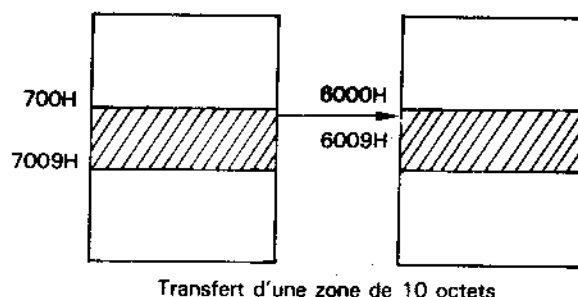
LDI

L'octet pointé par l'adresse HL est chargé à l'adresse pointée par DE,
BC est décrémenté de 1,
HL et DE sont incrémentés de 1.



Utilisation

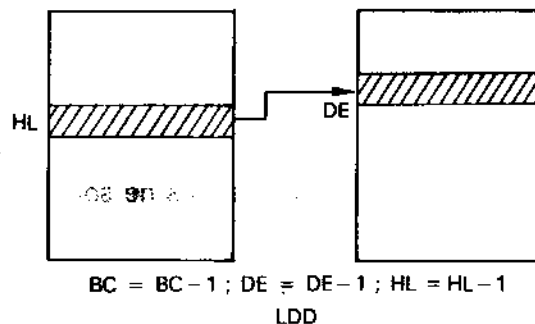
Supposons que nous voulions copier 10 octets à partir de l'adresse &7000 en &6000 :



1		ORG	9000H	
2		LOAD	9000H	
3	9000 110065	LD	DE,6500H	;@Destinataire
4	9003 210070	LD	HL,7000H	;@Source
5	9006 010A00	LD	BC,10	;10 Octets
6		BOUCLE:	EQU	\$
7	9009 EDA0	LDI		;Transfere 1 octet
8	900B 78	LD	A,B	
9	900C B1	OR	C	;BC = 0 ?
10	900D 20FA	JR	NZ,BOUCLE	;Oui
11		END		;Non

LDD

L'octet pointé par l'adresse HL est chargé à l'adresse pointée par DE,
BC est décrémenté de 1,
HL et DE sont décrémentés de 1.



III. Lecture, écriture, échanges et opérations sur les registres

La différence avec le groupe d'instructions précédent vient du fait que les opérations sont faites entre registres et ne font jamais intervenir une mémoire.

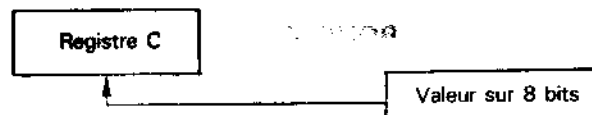
Reportez-vous à la partie 4 chapitre 2.2 pour prendre connaissance des modes d'adressage utilisés par le Z80, car ce qui suit fait largement appel à ces notions.

- Adressage immédiat sur 8 bits :

LD X, VAL8

où X peut être un des registres suivants : A, B, C, D, E, H, ou L.

Les indicateurs ne sont pas affectés.



08;

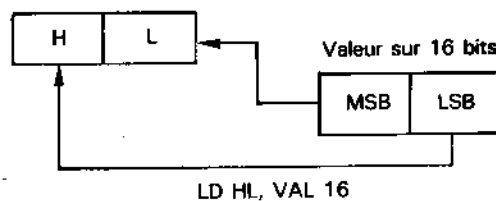
LDC, VAL 8

- Adressage immédiat sur 16 bits :

LD XX, VAL 16

où XX peut être un des registres pairs suivants : BC, DE, HL, SP, IX ou IY.

Les indicateurs ne sont pas affectés.

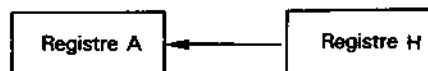


LD HL, VAL 16

- Entre registres sur 8 bits :

LD R1,R2 où R1 et R2 sont l'un des registres suivants : A, B, D, E, H ou L.

Les indicateurs ne sont pas affectés.



LD A, H

- Entre A et les registres I ou R :

LD A,I charge dans A le registre d'interruptions.

Les indicateurs sont modifiés :

H=0, N=0, C inchangé, S et Z modifiés, P/V = état de masquage des interruptions (IFF).

LD A,R charge dans A le registre de rafraîchissement mémoire.

Les indicateurs sont modifiés :

H=0, N=0, C inchangé, S et Z modifiés.

LD I,A modifie le registre d'interruption, et
LD R,A modifie le registre de rafraîchissement mémoire.

Les indicateurs ne sont pas affectés par ce deux dernières instructions.

- Entre deux registres sur 16 bits :

LD SP, HL
LD SP,IX et
LD SP,IY

Les indicateurs ne sont pas affectés.

Ces ordres permettent de charger le pointeur de pile en une fois par les registres HL, IX ou IY.

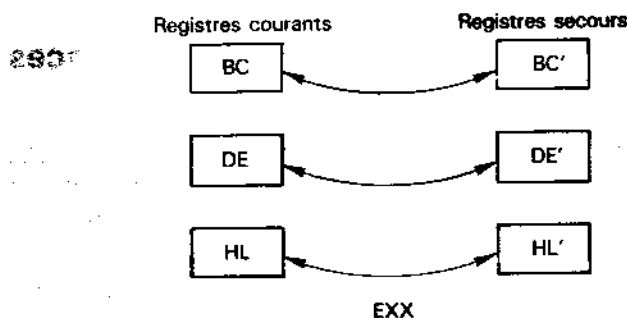
Généralement, la pile est placée le plus haut possible en mémoire, ou du moins à un endroit qui ne risque pas d'écraser des données ou d'être écrasé par des données.

- Echange de registres 16 bits :

1) Echange des registres courants avec les registres secours :

EXX Echange les registres BC, DE et HL avec les registres de secours BC', DE' et HL'.

Les indicateurs ne sont pas affectés.



EX AF,AF' Echange le registre AF avec le registre de secours AF'. Ces ordres sont pratiques si l'on exécute un sous-programme classique ou de gestion d'interruptions. Si on les place en début et en fin de sous-programme, ils permettent de travailler sur tous les registres sans détruire les valeurs qui y étaient au moment de l'appel du sous-programme.

2) Echange des registres DE et HL :

EX DE,HL

Le registre DE prend la valeur du registre HL, et le registre HL celle du registre DE.

Les indicateurs ne sont pas affectés.

3) Echange indirect sur SP :

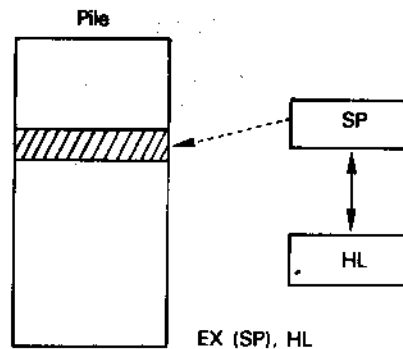
EX (SP),HL

EX (SP),IX

EX (SP),IY

La mémoire pointée par le registre pointeur de pile SP prend la valeur contenue dans HL, IX, ou IY.

Les indicateurs ne sont pas affectés.



IV. Ruptures de séquences

Ce groupe comprend toutes les instructions de débranchements (conditionnels ou non) à une autre instruction que celle qui suit l'instruction courante, ou à un sous-programme.

Reportez-vous à la partie 4 chapitre 2.2 pour prendre connaissance des modes d'adressage utilisés par le Z80, car ce qui suit fait largement appel à ces notions.

Remarque :

Les indicateurs ne sont pas modifiés par l'utilisation des ordres de débranchement.

Les modes d'adressage sont : absolu, indirect et relatif.

- Adressage absolu :

Dans la suite, ADR représente une adresse absolue exprimée sur 16 bits.

JP ADR

absolute Jump : Débranchement inconditionnel à l'adresse indiquée.

JP C,ADR

absolute Jump if Carry : saut absolu si C = 1.

Si l'indicateur Carry (dernier résultat supérieur à 255 ou comparaison entre 2 nombres dont le 1^{er} est strictement inférieur au second) est positionné, le débranchement se produit.

```

1          ORG      9000H
2          LOAD     9000H
3 9000 3E0A    LD      A,10
4 9002 FE0B    CP      11          ;C positionne
5 9004 DA0790  JP      C,ADR      ;Saut effectue
6          ;Traitement Carry off
7          ADR:    EQU      $
8          ;Traitement Carry on
9          END
    
```

JP Z,ADR

absolute Jump if Zero : saut absolu si Z = 1.

Si l'indicateur Zéro est positionné (le dernier calcul a donné un résultat nul ou une comparaison a été faite entre deux nombres identiques), le débranchement se produit.

```

1          ORG      9000H
2          LOAD     9000H
3 9000 3E0A    LD      A,10
4 9002 D60A    SUB     10          ;Z positionne
5 9004 CA0790  JP      Z,ADR      ;Saut effectue
6          ;Traitement Zero off
7          ADR:    EQU      $
8          ;Traitement Zero on
9          END
    
```

JP M,ADR

absolute Jump if Minus : saut si résultat négatif.

Si le dernier calcul a donné un résultat négatif, le débranchement est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3 9000 3E05          LD          A,5
4 9002 FE0C          CP          12          ;12 > 5
5 9004 FA0790        JP          M,ADR      ;Saut effectuée
6          ;Traitement si minus faux
7 ADR:      EQU          $
8          ;Traitement si minus vrai
9          END

```

JP PE,ADR

absolute Jump if parity Even : saut absolu si le bit P des indicateurs est égal à 1.

Si sur une opération logique le nombre de bits à 1 du résultat est pair, le débranchement est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3 9000 3E11          LD          A,11H
4 9002 B7           OR          A          ;P = 1:2 bits a 1 dans 11H
5 9003 EA0690        JP          PE,ADR    ;Saut effectuée
6          Traitement parite impaire
7 ADR:      EQU          $
8          ;Traitement parite paire
9          END

```

JP NC,ADR

absolute Jump if Not Carry : saut absolu si C = 0.

Si l'indicateur Carry n'est pas positionné (dernier résultat inférieur à 255 ou comparaison entre deux nombres dont le premier est supérieur ou égal au second) le débranchement se produit.

```

1          ORG          9000H
2          LOAD         9000H
3 9000 3E12      LD      A,12H
4 9002 FE05      CP      5          ;C non positionne
5 9004 D20790    JP      NC,ADR     ;Saut effectue
6              ;Traitement Carry off
7          ADR:      EQU      $
8              ;Traitement Carry on
9          END

```

JP NZ,ADR

absolute Jump if Not Zero : saut absolu si Z = 0.

Si l'indicateur Zéro n'est pas positionné (le dernier calcul a donné un résultat non nul ou une comparaison a été faite entre deux nombres différents), le débranchement est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3 9000 3E0C      LD      A,12
4 9002 FE14      CP      20         ;Z non positionne
5 9004 C20790    JP      NZ,ADR     ;Saut effectue
6              ;Traitement si Zero
7          ADR:      EQU      $
8              ;Traitement si non Zero
9          END

```

JP P,ADR

absolute Jump if Plus : saut absolu si résultat positif.

Si le dernier calcul a donné un résultat strictement positif, le débranchement est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3 9000 3E7B      LD      A,123
4 9002 C604      ADD     A,4         ;Resultat positif
5 9004 F20790    JP      P,ADR      ;Saut effectue
6              ;Traitement resultat negatif
7          ADR:      EQU      $
8              ;Traitement resultat positif
9          END

```

JP PO,ADR

absolute Jump if Parity Odd : saut absolu si le bit P des indicateurs est égal à 0.

Si sur une opération logique, le nombre de bits à 1 du résultat est impair, le débranchement se produit.

1		ORG	9000H	
2		LOAD	9000H	
3	9000 3E13	LD	A,13H	
4	9002 A7	AND	A	;P=0:3 bits a 1
5	9003 E20690	JP	PO,ADR	;Saut effectue
6	;Traitement	parite paire		
7	ADR:	EQU	\$	
8	;Traitement	parite impaire		
9		END		

- Adressage indirect absolu :

JP (HL)

JP (IX)

JP (IY)

Les indicateurs ne sont pas modifiés.

- Adressage relatif :

Dans la suite, DEPL représente un déplacement relatif signé sur 7 bits. Le bit 8 (bit de poids le plus fort) représente le signe. S'il est égal à un, le déplacement se fait vers le haut. S'il est égal à 0, le déplacement se fait vers le bas.

JR DEPL

Jump Relative : débranchement inconditionnel au déplacement indiqué.

JR C, DEPL

Jump Relative if Carry : saut relatif si le bit C des indicateurs est égal à 1.

Si l'indicateur Carry est positionné (dernier résultat supérieur à 255 ou comparaison entre 2 nombres dont le 1^{er} est strictement inférieur au second), le débranchement se produit.

1		ORG	9000H
2		LOAD	9000H

```

3 9000 3E12      LD      A,12H
4 9002 0616      LD      B,16H
5 9004 B8        CP      B      ;C positionne
6 9005 3800      JR      C,ADR   ;Saut effectue
7                Traitement si C = 0
8                ADR: EQU      $
9                ;Traitement si C = 1
10               END

```

JR Z,DEPL

Jump Relative if Zero : saut relatif si le bit Z des indicateurs est égal à 1.

Si l'indicateur zéro est positionné (le dernier calcul a donné un résultat nul ou une comparaison a été faite entre deux nombres identiques), le débranchement se produit.

```

1                ORG      9000H
2                LOAD     9000H
3 9000 3E7A      LD      A,122
4 9002 D67A      SUB      122    ;Z positionne
5 9004 2800      JR      Z,ADR   ;Saut effectue
6                ;Traitement si non zero
7                ADR: EQU      $
8                ;Traitement si zero
9                END

```

JR NC,DEPL

Jump Relative if Not Carry : saut relatif si le bit C des indicateurs est égal à 0.

Si l'indicateur Carry n'est pas positionné (dernier résultat inférieur à 255 ou comparaison entre deux nombres dont le premier est supérieur ou égal au second), le débranchement se produit.

```

1                ORG      9000H
2                LOAD     9000H
3 9000 3E12      LD      A,12H

```

```

4 9002 0E04      LD A      C,4
5 9004 B9        CP      C      ;C non positionne
6 9005 3000      JR      NC,ADR  ;Saut effectue
7                ;Traitement si C = 1
8                ADR:      EQU    $      ;remet à 0
9                ;Traitement si C = 0
10               END

```

JR NZ, DEPL

Jump Relative if Not Zero : saut relatif si le bit Z des indicateurs vaut 0.

Si l'indicateur Zéro n'est pas positionné (le dernier calcul a donné un résultat non nul ou une comparaison a été faite entre deux nombres différents), le débranchement se produit.

```

1                ORG      9000H
2                LOAD     9000H
3 9000 3E7A      LD      A,122
4 9002 FE01      CP      1      ;Z non positionne
5 9004 C20790    JP      NZ,ADR  ;Saut effectue
6                ;Traitement si Zero
7                ADR:      EQU    $      ;remet à 0
8                ;Traitement si non Zero
9                END

```

DJNZ DEPL

Decrement B and Jump if Not Zero : Décrémente le registre B de 1 et saute à l'adresse relative indiquée si $B < > 0$.

Cette instruction est très pratique pour effectuer des calculs ou des tests répétitifs, comme le montre l'exemple suivant.

Utilisation

Supposons que nous voulions chercher l'occurrence de l'octet 54H sur un espace mémoire commençant en 7000H et sur une longueur de 40 octets.

Pour réaliser ce test, il sera commode d'utiliser l'ordre DJNZ de la façon suivante :

```

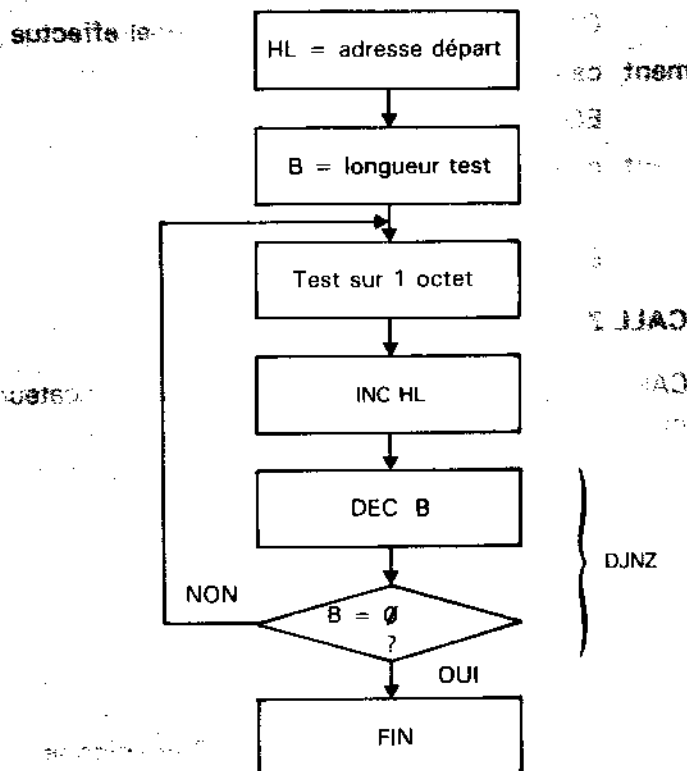
1                ORG      9000H
2                LOAD     9000H

```

```

3 9000 210070      LD      HL,7000H      ;@ depart
4 9003 0628         LD      B,40          ;Lgr test
5                   BOUCLE: EQU      $
6 9005 7E           LD      A,(HL)        ;octet teste
7 9006 23           INC     HL            ;memoire suivante
8 9007 FE54         CP      54H           ;54 hexa ?
9 9009 CA0E90       JP      Z,TROUVE      ;oui
10 900C 10F7        DJNZ    BOUCLE       ;non
11
12                 TROUVE: EQU      $
13                 ;A ce niveau, si Z = 0, HL = @ octet + 1
14                 ;si Z = 0, l'octet n'est pas trouve
15                 END

```



• Appel à un sous-programme :

Débranchement du programme en forçant la valeur du PC (Program Counter) à la valeur indiquée dans l'argument. L'adresse de retour du sous-programme est mise dans la pile, et le retour sera effectué sur la rencontre d'une instruction RET ou équivalent.

L'adressage est absolu.

CALL ADR

Débranchement inconditionnel au sous-programme indiqué.

CALL C,ADR

CALL if Carry : Appel au S/P si le bit C des indicateurs est égal à 1.

Si l'indicateur Carry est positionné (dernier résultat supérieur à 255 ou comparaison entre 2 nombres dont le 1^{er} est strictement inférieur au second), le débranchement se produit.

1		ORG	9000H	
2		LOAD	9000H	
3	9000 210080	LD	HL,8000H	
4	9003 110080	LD	DE,8000H	
5	9006 ED5A	ADC	HL,DE	;C positionne
6	9008 DC0B90	CALL	C,ADR	;Appel effectue
7				
	;Traitement	carry off		
8	ADR:	EQU	\$	
9				
	;Traitement	carry on		
10	900B C9	RET		
11		END		

CALL Z,ADR

CALL if Zero : Appel au S/P si le bit Z des indicateurs est égal à 1.

Si l'indicateur Zéro est positionné (le dernier calcul a donné un résultat nul ou une comparaison a été faite entre deux nombres identiques), le débranchement se produit.

1		ORG	9000H	
2		LOAD	9000H	
3	9000 3EC4	LD	A,0C4H	
4	9002 063C	LD	B,3CH	
5	9004 80	ADD	A,B	;Z positionne
6	9005 CC0890	CALL	Z,ADR	;Appel effectue
7				
	;Traitement	Z = 0		
8	ADR:	EQU	\$	
9				
	;Traitement	Z = 1		
10	9008 C9	RET		
11		END		

CALL M,ADR

CALL if Minus : Appel au S/P si résultat négatif.

Si le dernier calcul a donné un résultat négatif, le débranchement est effectué.

1		ORG	9000H	
2		LOAD	9000H	
3	9000 3E0C	LD	A,12	
4	9002 FE7F	CP	127	;127>12
5	9004 FC0790	CALL	M,ADR	;Appel effectue
6				
7		ADR:	EQU	\$
8				
9	9007 C9	RET		
10		END		

CALL PE,ADR

CALL if Parity Even : Appel au S/P si le bit P des indicateurs est égal à 1.

Si sur une opération logique le nombre de bits à 1 du résultat est pair, le débranchement est effectué.

1		ORG	9000H	
2		LOAD	9000H	
3	9000 3E12	LD	A,12H	
4	9002 0605	LD	B,5	
5	9004 B0	OR	B	;4 b a 1 → P=1
6	9005 EC0890	CALL	PE,ADR	;Appel effectue
7				
8		ADR:	EQU	\$
9				
10	9008 C9	RET		
11		END		

CALL NC, ADR

CALL if Not Carry : Appel au S/P si le bit C des indicateurs est égal à 0.

Si l'indicateur Carry n'est pas positionné (dernier résultat inférieur à 255 ou comparaison entre deux nombres dont le premier est supérieur ou égal au second), le débranchement se produit.

```

1          ORG      9000H
2          LOAD     9000H
3 9000 3E76      LD      A,76H
4 9002 0E0E      LD      C,0EH
5 9004 81        ADD     A,C      ;C non positionne
6 9005 D40890    CALL     NC,ADR   ;Appel effectue
7              ;Traitement Carry on
8  ADR:      EQU     $
9              ;Traitement Carry off
10 9008 C9       RET
11              END

```

CALL NZ,ADR

CALL if Not Zero : Appel au S/P si le bit Z des indicateurs est égal à 0.

Si l'indicateur Zéro n'est pas positionné (le dernier calcul a donné un résultat non nul ou une comparaison a été faite entre deux nombres différents), le débranchement est effectué.

```

1          ORG      9000H
2          LOAD     9000H
3 9000 3E0C      LD      A,12
4 9002 1E04      LD      E,4
5 9004 83        ADD     A,E      ;Z non positionne
6 9005 C40890    CALL     NZ,ADR   ;Appel effectue
7              ;Traitement Z = 1
8  ADR:      EQU     $
9              ;Traitement Z = 0
10 9008 C9       RET
11              END

```

CALL P,ADR

CALL if Plus : Appel au S/P si le résultat est positif.

Si le dernier calcul a donné un résultat strictement positif, le débranchement est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3 9000 3E0C      LD      A,12
4 9002 0E04      LD      C,4
5 9004 91        SUB     C      ;resultat > 0
6 9005 F40890    CALL    P,ADR  ;Appel effectue
7              ;Traitement si resultat <= 0
8  ADR:         EQU     $
9              ;Traitement si resultat > 0
10 9008 C9       RET
11              END

```

CALL PO,ADR

CALL if Parity Odd : Appel au S/P si le bit P des indicateurs est égal à 0.

Si sur une opération logique, le nombre de bits à 1 du résultat est impair, le débranchement se produit.

```

1          ORG          9000H
2          LOAD         9000H
3 9000 3E02      LD      A,2
4 9002 17        RLA                ;1 b a 1 → P=0
5 9003 E40690    CALL    PO,ADR  ;Appel effectue
6              ;Traitement parite paire
7  ADR:         EQU     $
8              ;Traitement parite impaire
9 9006 C9       RET
10          END

```

• Retour de sous-programme :

L'utilisation d'une instruction de retour de sous-programme — ou d'interruptions — provoque un débranchement à l'instruction qui a suivi l'appel. C'est-à-dire à l'adresse de l'ordre ayant provoqué l'appel + 3. En effet, l'ordre utilisé pour appeler un sous-programme est un CALL qui est codé sur 3 octets :

1 ^{er} octet	2 ^e octet	3 ^e octet
Code Op	LSB	MSB

Cette adresse est retrouvée en dépilant le PC (Program Counter) de la pile. L'adressage est indirect au travers de la pile.

RET

RETurn from subroutine : Retour inconditionnel.

RETI

RETurn from Interrupt : Retour d'interruption.

RETN

RETurn from Non masquable Interrupt : Retour d'interruption non masquable.

RET C

RETurn if Carry : Retour d'interruption si le bit C des indicateurs est à 1. Si l'indicateur Carry est positionné (dernier résultat supérieur à 255 ou comparaison entre 2 nombres dont le 1^{er} est strictement inférieur au second), le retour est effectué.

1		ORG	9000H	
2		LOAD	9000H	
3	SOUP:	EQU	\$	
4	;Corps du sous-programme			
5	9000 FE30	CP	30H	
6	9002 D8	RET	C	;Retour si A < 30
7	9003 18FB	JR	SOUP	;Poursuite sinon
8		END		

RET Z

RETurn if Zero : Retour si le bit Z des indicateurs est à 1.

Si l'indicateur Zéro est positionné (le dernier calcul a donné un résultat nul ou une comparaison a été faite entre deux nombres identiques), le retour est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3          SOUP:      EQU          $
4          ;Corps du sous-programme
5 9000 0D          DEC          C          ;C = C - 1
6 9001 C8          RET          Z          ;retour si C = 0
7 9002 C30090      JP          SOUP      ;Poursuite sinon
8          END

```

RET M

RETurn if Minus : Retour si résultat négatif.

Si le résultat du dernier calcul a donné un nombre négatif, le retour est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3          SOUP:      EQU          $
4          ;Corps du sous-programme
5 9000 35          DEC          (HL)      ;Mem (HL) - 1
6 9001 C8          RET          M          ;retour si résultat < 0
7 9002 18FC        JR          SOUP      ;Poursuite sinon
8          END

```

RET PE

RETurn if Parity Even : Retour si le bit P des indicateurs est à 1.

Si sur une opération logique le nombre de bits à 1 du résultat est pair, le retour est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3          SOUP:      EQU          $
4          ;Corps du sous-programme
5 9000 CB27        SLA          A
6 9002 E8          RET          PE      ;si nbre de b a 1 pair
7 9003 18FB        JR          SOUP      ;Poursuite traitement
8          END

```

RET NC

RETurn if Not Carry : Retour si le bit C des indicateurs est à 0.

Si l'indicateur Carry n'est pas positionné (dernier résultat inférieur à 255 ou comparaison entre deux nombres dont le premier est supérieur ou égal au second), le retour est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3          SOUP: EQU     $
4          ;Corps du sous-programme
5 9000 34          INC   (HL)
6 9001 D0          RET   NC      ;Retour si (HL) <= 255
7 9002 18FC        JR    SOUP    ;Poursuite sinon
8          END

```

RET NZ

RETurn if Not Zero : Retour si le bit Z des indicateurs est à 0.

Si l'indicateur Zéro n'est pas positionné (le dernier calcul a donné un résultat non nul ou une comparaison a été faite entre deux nombres différents), le retour est effectué.

```

1          ORG          9000H
2          LOAD         9000H
3          SOUP: EQU     $
4          ;Corps du sous-programme
5 9000 35          DEC   (HL)
6 9001 C0          RET   NZ      ;Retour si (HL) <> 0
7 9002 18FC        JR    SOUP    ;Poursuite sinon
8          END

```

RET P

RETurn if Plus : Retour si résultat positif.

Si le dernier calcul a donné un résultat strictement positif, le débranchement est effectué.

```

1          ORG          9000H
2          LOAD         9000H

```

```

3      SOUP:      EQU      $
4      ;Corps du sous-programme
5 9000 FE40      CP      40H
6 9002 F0      RET      P      ;retour si A >= 40H
7 9003 18FB      JR      SOUP      ;poursuite sinon
8      END

```

RET PO

RETurn if Parity Odd : Retour si le bit P des indicateurs est à 0.

Si sur une opération logique, le nombre de bits à 1 du résultat est impair, le débranchement se produit.

```

1      ORG      9000H
2      LOAD     9000H
3      SOUP:    EQU      $
4      ;Corps du sous-programme
5 9000 B7      OR      A
6 9001 E0      RET      PO      ;si nb de b a 1 impair
7 9002 18FC      JR      SOUP      ;poursuite sinon
8      END

```

Utilisation des retours conditionnels :

Réalisation d'un sous-programme d'attente paramétrable. Le nombre de millisecondes d'attente est passé au sous-programme dans le registre C.

```

1      ORG      9000H
2      LOAD     9000H
3 9000 1606      LD      D,6
4 9002 CD0690      CALL     SOUP      ;6 ms d'attente
5 9005 C9      RET
6      SOUP:    EQU      $      ;S/P d'attente
7 9006 0E03      LD      C,3      ; 3 boucles
8      B1:      EQU      $
9 9008 0600      LD      B,0
10     B2:      EQU      $

```

11 900A 00	NOP
12 900B 10FD	DJNZ B2
13 900D 0D	DEC C
14 900E 20F8	JR NZ,B1
15 9010 15	DEC D
16 9011 20F3	JR NZ,S0UP
17 9013 C9	RET
18	END

V. Opérations arithmétiques

A. ADDITIONS

ADD et ADC

Additions sur 8 ou 16 bits avec ou sans retenue.

Remarque :

Les additions sur 8 bits travaillent sur 7 bits de données et 1 bit de signe.
Les additions sur 16 bits travaillent sur 15 bits de données et 1 bit de signe.
Les modes d'adressage sont : immédiat, registre 8 ou 16 bits, indirect ou indirect indexé.

Pour les instructions ADD et ADC sur 8 bits, les indicateurs sont modifiés de la façon suivante : N = 0, C, Z, V, S et H affectés selon le résultat de l'addition.

Pour les instructions ADD et ADC sur 16 bits, l'indicateur H n'a aucun sens.

- Adressage immédiat sur 8 bits :

ADD A, VAL8

Additionne la valeur VAL8 à l'accumulateur sans tenir compte de la retenue (indicateur C).

Le résultat se trouve dans l'accumulateur.

0	0	1	1	0	1	1	1
---	---	---	---	---	---	---	---

A = 37H

+

0	1	1	1	0	1	0	1
---	---	---	---	---	---	---	---

VAL 8 = 75H

=

1	0	1	0	1	1	0	0
---	---	---	---	---	---	---	---

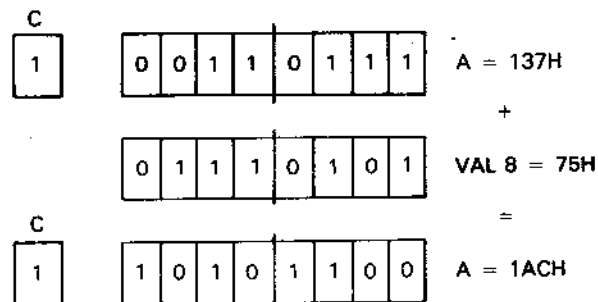
A = ACH

LD A, 37H
ADD 75H
A = ACH

ADC A,VAL8

Additionne la valeur VAL8 à l'accumulateur en tenant compte de la retenue (indicateur C).

Le résultat se trouve dans l'accumulateur.



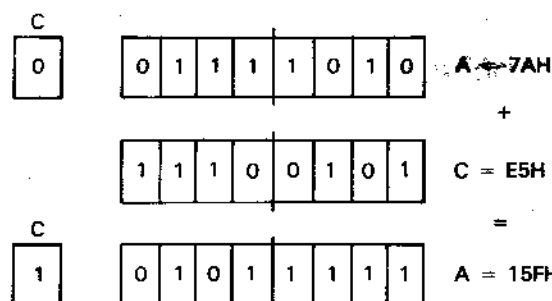
LD A, 37H
ADC 75H
A = 1ACH

- Adressage registre 8 bits :

ADD A,X

où X est l'un des registres suivants : A, B, C, D, E, H ou L.

Additionne le registre spécifié à l'accumulateur sans tenir compte de la retenue (indicateur C). Le résultat est dans l'accumulateur.



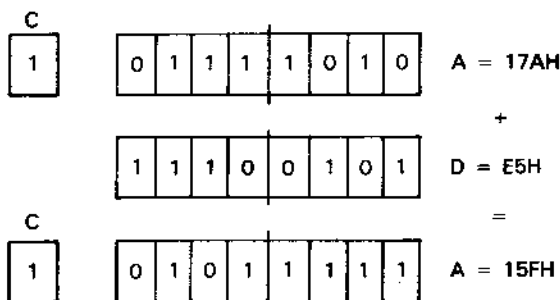
LD A, 7AH
LD C, E5H
ADD A,C
A = 15FH

ADC A,X

où X est l'un des registres suivants : A, B, C, D, E, H ou L.

Additionne le registre spécifié à l'accumulateur en tenant compte de la retenue (indicateur C).

Le résultat est dans l'accumulateur.



LD A, 7AH
LD D, E5H
ADC A,D
A = 15FH

- Adressage registre 16 bits : **ADD HL,XX**

ADD HL,XX

où XX est un des registres pairs suivants : BC, DE, HL, SP.

ADD IX,XX

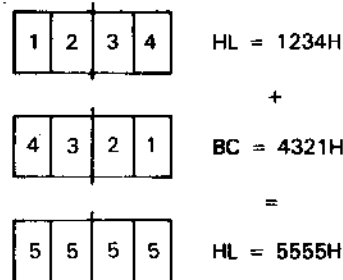
où XX est un des registres pairs suivants : BC, DE, IX, SP.

ADD IY,XX

où XX est un des registres pairs suivants : BC, DE, IY, SP.

Additionne les deux registres pairs spécifiés sans tenir compte de la retenue (indicateur C).

Le résultat est dans HL.

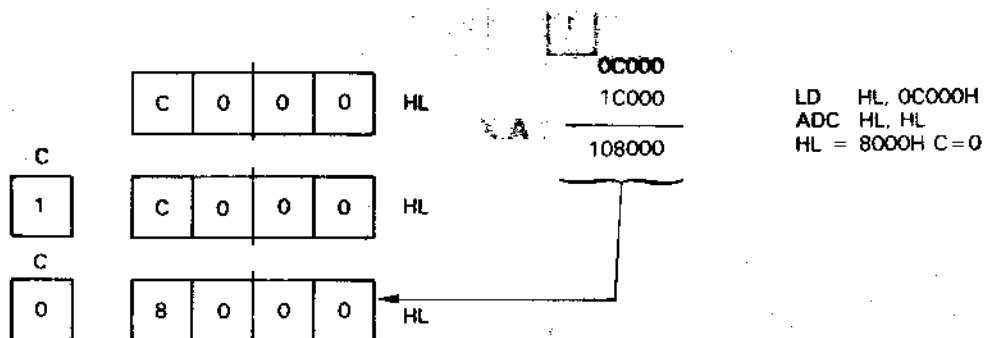


```
LD HL, 1234H
LD BC, 4321H
ADD HL, BC
HL = 5555H
```

ADC HL,XX

où XX est un des registres pairs suivants : BC, DE, HL, SP.

Additionne les deux registres pairs spécifiés en tenant compte de la retenue (indicateur C).



- Adressage indirect sur 8 bits :

ADD A,(HL)

Additionne l'octet pointé par HL à l'accumulateur sans tenir compte de la retenue (indicateur C).

Le résultat est dans A.

ADC A,(HL)

Additionne l'octet pointé par HL à l'accumulateur en tenant compte de la retenue (indicateur C).

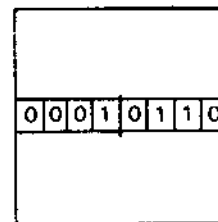
Le résultat est dans A.

1 1 1 0 0 0 0 0 A = E0H

+ (HL) → 4000H

1 1 1 1 0 1 1 0 A = F6H

```
LD A, 0E0H
LD HL, 4000H
ADD A, (HL)
A = F6H
```



- Adressage indirect indexé sur 8 bits :

ADD A,(IX + d)

ADD A,(IY + d)

Additionne l'octet pointé par IX + d ou IY + d à l'accumulateur sans tenir compte de la retenue (indicateur C).

d est un déplacement dont la valeur est comprise entre -127 et 127.

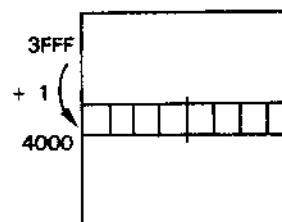
Le résultat est dans A.

1 1 1 0 0 0 0 0 A = E0H

+ (IY + 1)

1 1 1 1 0 1 1 0 A = F6H

```
LD A, 0E0H
LD IY, 3FFFH
ADD A, (IY + 1)
A = F6H
```



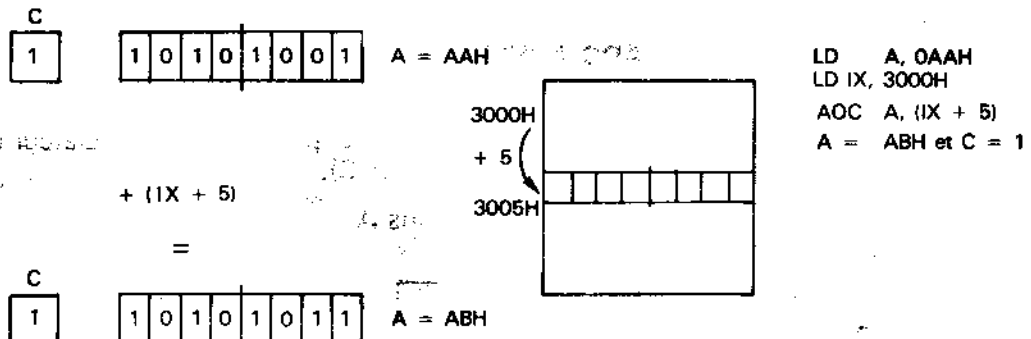
ADC A,(IX + d)

ADC A,(IY + d)

Additionne l'octet pointé par IX + d ou IY + d à l'accumulateur en tenant compte de la retenue (indicateur C).

d est un déplacement dont la valeur est comprise entre -127 et 127.

Le résultat est dans A.



B. SOUSTRACTIONS

SUB et SBC

Soustraction sur 8 ou 16 bits avec ou sans retenue.

Les modes d'adressage sont : immédiat, registre 8 ou 16 bits, indirect ou indirect indexé.

Remarques :

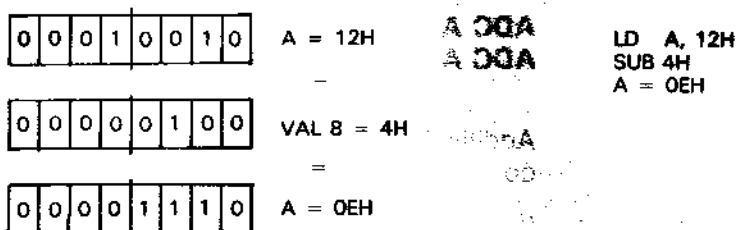
- Les soustractions sur 8 bits travaillent sur 7 bits de données et 1 bit de signe. Les soustractions sur 16 bits travaillent sur 15 bits de données et 1 bit de signe.
- Pour les instructions SUB et SBC sur 8 bits, les indicateurs sont modifiés de la façon suivante : N = 1, C, Z, V, S et H affectés selon le résultat de la soustraction.
- Pour les instructions SUB et SBC sur 16 bits, l'indicateur H n'a aucun sens.

- Adressage immédiat sur 8 bits :

SUB VAL8

Soustrait la valeur VAL8 de l'accumulateur sans tenir compte de la retenue avant soustraction (indicateur C).

Le résultat se trouve dans l'accumulateur.



SBC A, VAL8

Soustrait la valeur VAL8 de l'accumulateur en tenant compte de la retenue avant soustraction (indicateur C).
Le résultat se trouve dans l'accumulateur.

	0	0	0	1	0	0	1	0	A = 12H	
C	1	0	0	0	0	0	0	1	VAL 8 = 1H et C = 1	LD A, 12H Supposons que C = 1 SUB 1H A = 11H C = 0
C	0	0	0	1	0	0	0	1	A = 11H	

- Adressage registre 8 bits :

SUB X

où X est l'un des registres suivants : A, B, C, D, E, H ou L.
Soustrait le registre spécifié de l'accumulateur sans tenir compte de la retenue avant soustraction.
Le résultat est dans l'accumulateur.

	1	1	1	1	1	1	1	1	A = FFH	
	0	0	0	0	0	0	0	1	C = 1	LD A, FFH LD C, 1H SUB C A = FEH
	1	1	1	1	1	1	1	0	A = FEH	

SBX A,X

où X est l'un des registres suivants : A, B, C, D, E, H ou L.
Soustrait le registre spécifié de l'accumulateur en tenant compte de la retenue avant soustraction (indicateur C).
Le résultat est dans l'accumulateur.

	0	0	0	0	0	0	0	1	A = 1H	
C	1	1	1	0	1	1	1	1	D = FEH	LD A, 1H Supposons que C = 1 LD D, OFEH SBX A, D A = 3H C = 0
C	0	0	0	0	0	0	0	1	A = 3H	

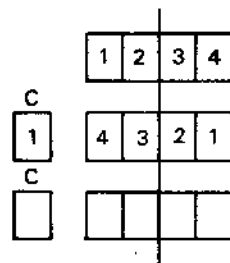
- Adressage registre 16 bits :

SBC HL,XX

où XX est un des registres pairs suivants : BC, DE, HL, SP.

Soustrait les deux registres pairs spécifiés en tenant compte de la retenue avant soustraction (indicateur C).

Le résultat est dans HL.



```
LD HL, 1234H
LD BC, 4321H
Supposons que C = 1
SBC HL, BC
HL = CF 12H C = 1
```

Remarque :

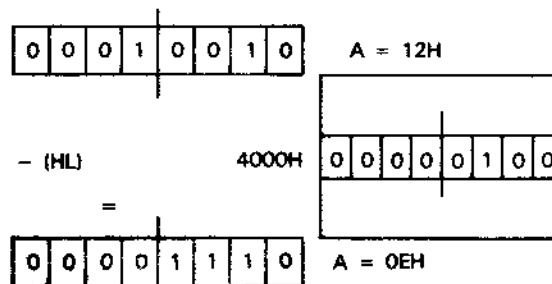
L'ordre de soustraction de deux registres pairs sans tenir compte de la retenue n'existe pas. Vous serez donc obligé d'utiliser un SBC pour soustraire deux registres pairs. Pour ne pas avoir de mauvaise surprise, nous vous conseillons d'effacer l'indicateur Carry avant de faire la soustraction.

- Adressage indirect sur 8 bits :

SUB (HL)

Soustrait l'octet pointé par HL de l'accumulateur sans tenir compte de la retenue avant la soustraction (indicateur C).

Le résultat est dans A.

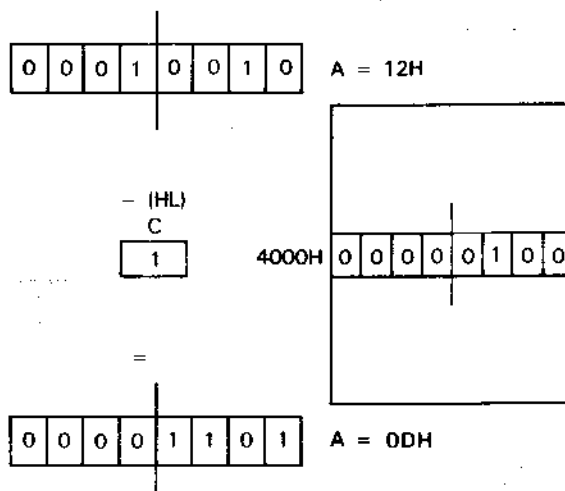


```
LD A, 12H
LD HL, 4000H
SUB (HL)
A = 0EH
```

SBC A,(HL)

Soustrait l'octet pointé par HL de l'accumulateur en tenant compte de la retenue avant la soustraction (indicateur C).

Le résultat est dans A.



```
LD A, 12H
LD HL, 4000H
Supposons C = 1
SBC (HL)
A = 0DH
```

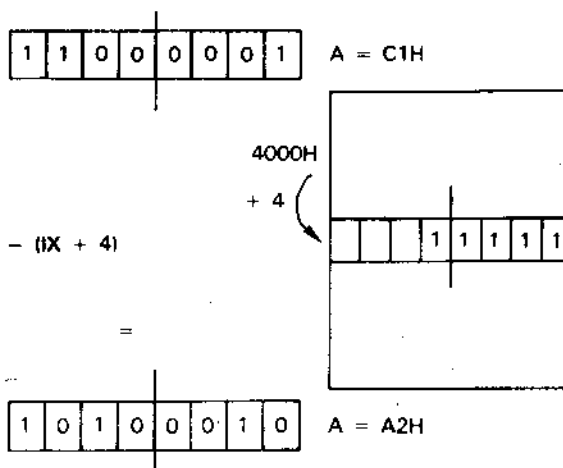
- Adressage indirect indexé sur 8 bits :

SUB A,(IX + d) SUB A,(IY + d)

Soustrait l'octet pointé par IX + d ou IY + d de l'accumulateur sans tenir compte de la retenue avant la soustraction (indicateur C).

d est un déplacement dont la valeur est comprise entre -127 et 127.

Le résultat est dans A.



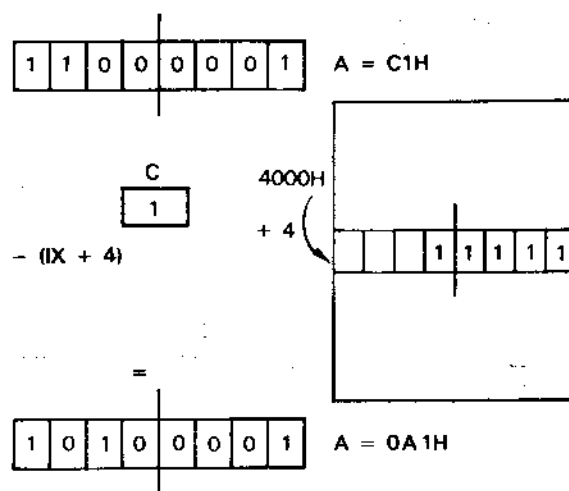
```
LD A, C1H
LD IX, 4000H
SUB (IX + 4)
A = A2H
```

SBC A,(IX + d)
SBC A,(IY + d)

Soustrait l'octet pointé par IX + d ou IY + d de l'accumulateur en tenant compte de la retenue avant la soustraction (indicateur C).

d est un déplacement dont la valeur est comprise entre - 127 et 127.

Le résultat est dans A.



```
LD  A, 0C1H
LD  IX, 4000H
SBC (IX + 4)
A = 0A1H
```

C. INCRÉMENTATION ET DÉCRÉMENTATION

INC et DEC

Agissent sur un registre 8 ou 16 bits ou sur une mémoire 8 ou 16 bits.

Les modes d'adressage sont : adressage registre 8 bits ou 16 bits, indirect 8 bits ou indirect indexé 8 bits.

Remarque :

Pour l'instruction INC, sauf indication contraire, les indicateurs sont modifiés de la façon suivante :

N=0, C inchangé, Z, V, S et H affectés selon le résultat de l'incréméntation.

- Adressage registre 8 bits :

INC X

où X = A, B, C, D, E, H ou L.

Incréménte de 1 la valeur du registre spécifié.

Indicateurs : N=0, C inchangé, Z, V, S et H modifiés.

```
LD      C,12 H
INC      C           ;C = 13H
LD      B, 0 FFH
INC      B ;           B=0 Flag Z = 1
```

DEC X

où X = A, B, C, D, E, H ou L.

Décrémente de 1 la valeur du registre spécifié.

Indicateurs : N = 1, C inchangé, Z, V, S et H modifiés.

```
LD      D,12H
DEC      D           ;D = 11H
LD      D,0
DEC      D           ;D = 0FFH
```

- Adressage registre 16 bits :

INC XX

où XX = BC, DE, HL, IX, IY ou SP.

Incrémente de 1 la valeur du registre spécifié.

Les indicateurs ne sont pas modifiés.

```
LD      HL, 3000H
INC      HL           ;HL = 3001H
LD      HL,0FFFFH
INC      HL           ;HL = 0
```

DEC XX

où XX = BC, DE, HL, IX, IY ou SP.

Décrémente de 1 la valeur du registre spécifié.

Les indicateurs ne sont pas modifiés.

```
LD      HL, 3000H
DEC      HL           ;HL = 2FFFFH
LD      HL,0
DEC      HL           ;HL = FFFFFH
```

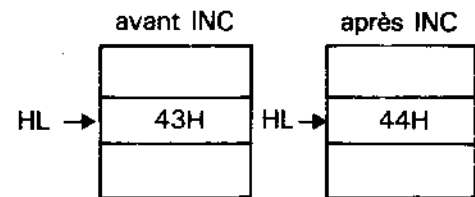
- Adressage indirect sur 8 bits :

INC (HL)

Incrémente de 1 l'octet pointé par HL.

Indicateurs : N=0, C inchangé, Z, V, S et H modifiés.

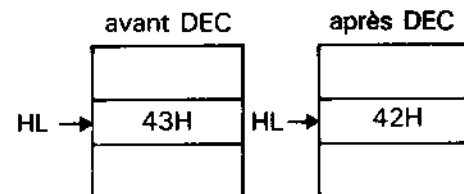
```
LD HL, 3000H
INC (HL)
```

**DEC (HL)**

Décrémente de 1 l'octet pointé par HL.

Indicateurs : N=1, C inchangé, Z, V, S et H modifiés.

```
LD HL, 3000H
DEC (HL)
```



- Adressage indirect indexé sur 8 bits :

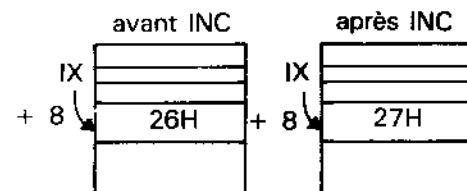
INC (IX + d)**INC (IY + d)**

Incrémente de 1 l'octet pointé par IX + d ou IY + d

d est un déplacement dont la valeur est comprise entre -127 et 127.

Indicateurs : N=0, C inchangé, Z, V, S et H modifiés.

```
LD IX, 3000H
INC (IX + 8)
```

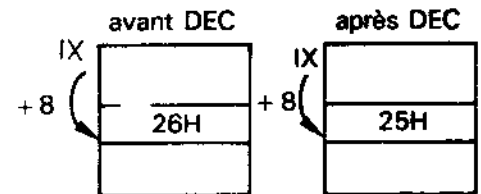
**DEC (IX + d)****DEC (IY + d)**

Décrémente de un l'octet pointé par IX + d ou IY + d.

d est un déplacement dont la valeur est comprise entre -127 et 127.

Indicateurs: N=1, C inchangé, Z, V, S et H modifiés.

```
LD IX, 3000H
DEC (IX + 8)
```



DAA

Decimal Adjust Accumulator : Ajustement décimal.

Transforme une donnée 8 bits exprimée en binaire et stockée dans le registre A en la même donnée codée en décimal dans le registre A.

L'adressage est implicite.

Indicateurs : N inchangé, C, Z, P, S, H modifiés.

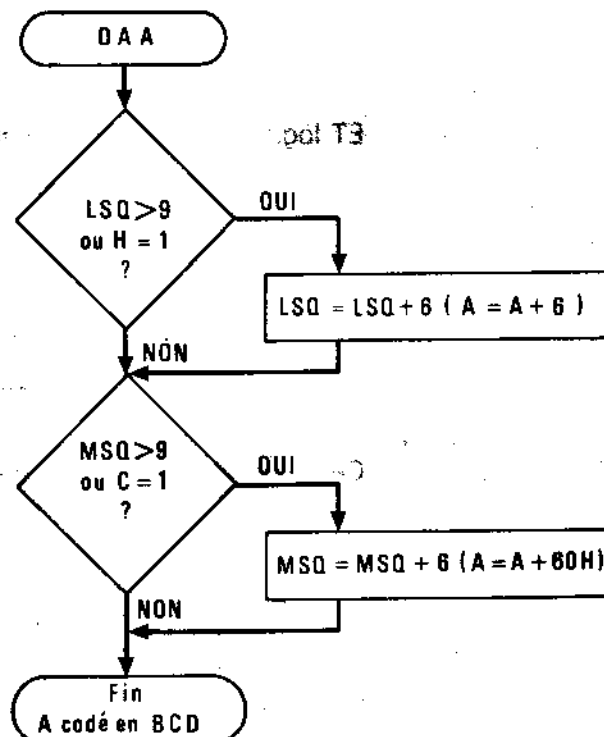
Comment fonctionne l'instruction DAA ?

Elle divise le registre A en deux quartets que nous appellerons MSQ et LSQ (Most Significant quartet et Last Significant quartet, c'est-à-dire quartets de poids fort et de poids faible).

Si LSQ est supérieur à 9 ou si H = 1 (voir remarque), LSQ est incrémenté de 6, c'est-à-dire que A est incrémenté de 6.

Si MSQ est supérieur à 9 ou si C = 1, MSQ est incrémenté de 6, c'est-à-dire que A est incrémenté de 60H.

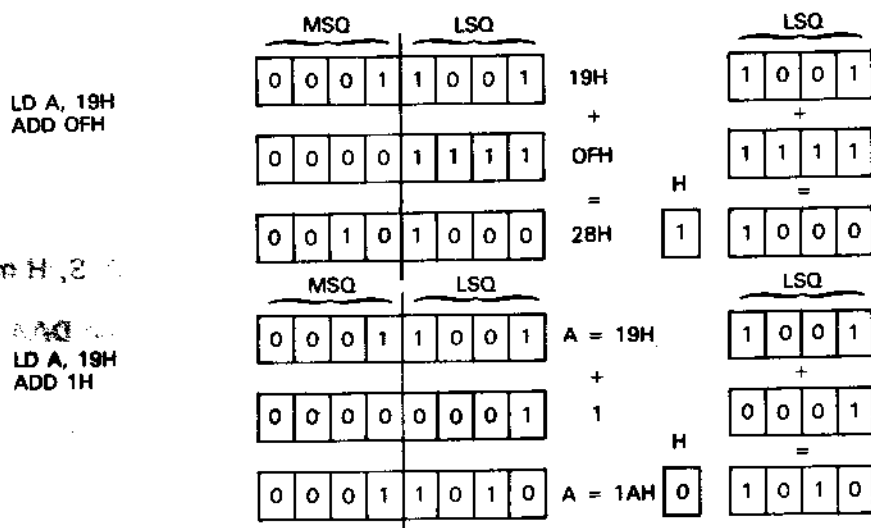
Ce qui se traduit par l'organigramme suivant :



Remarque :

Le bit H est appelé « Half Carry » ou demi-retenue.

Il est positionné sur les opérations arithmétiques, décalages et comparaisons si une retenue est obtenue sur le quartet de poids faible LSQ.

**VI. Opérations logiques****A. OPÉRATIONS LOGIQUES ÉLÉMENTAIRES****AND**

ET logique sur l'accumulateur.

L'adressage peut être immédiat, registre 8 bits, indirect, ou indirect indexé.

La table de vérité de AND au niveau bit est la suivante :

AND	0	1
0	0	0
1	0	1

Cette logique est appliquée entre chaque bit de même rang de l'accumulateur et de l'argument fourni.

Remarque :

Pour l'instruction AND, sauf indication contraire, les indicateurs sont modifiés de la façon suivante :

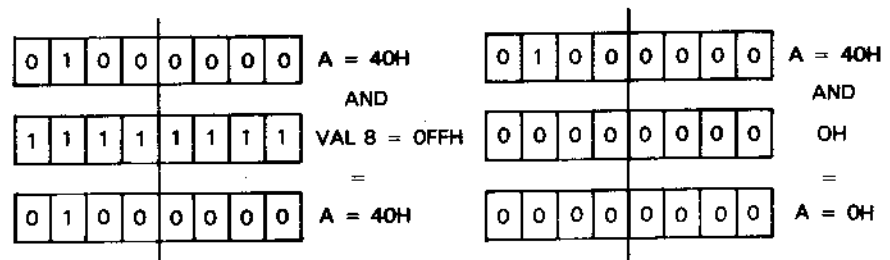
C=0, N=0, H=1, Z, P, et S affectés selon le résultat du AND.

• Adressage immédiat :

AND VAL8

ET logique entre l'accumulateur et VAL8.

Le résultat est dans A.



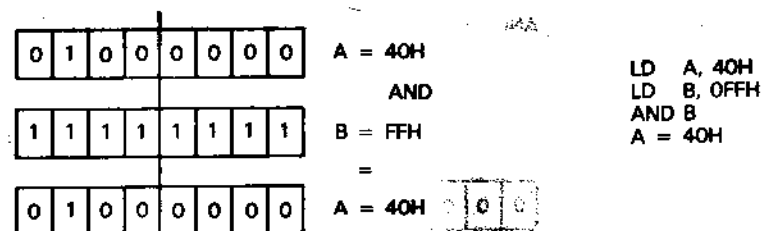
• Adressage registre 8 bits :

AND X

où X peut être A, B, C, D, E, H ou L.

ET logique entre l'accumulateur et le registre spécifié.

Le résultat est dans A.

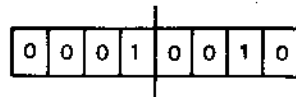


• Adressage indirect :

AND (HL)

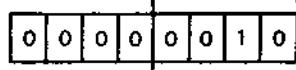
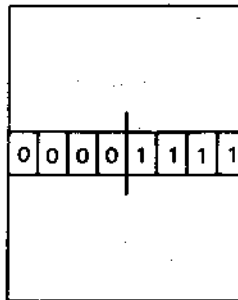
ET logique entre l'accumulateur et l'octet pointé par HL.

Le résultat est dans A.



A = 12H

AND (HL) 3000H



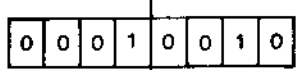
A = 12H

```
LD A, 12H
LD HL, 3000H
LD (HL), 0FH
AND (HL)
A = 02H
```

- Adressage indirect indexé :

AND (IX + d)
AND (IY + d)

ET logique entre l'accumulateur et l'octet pointé par IX + d ou IY + d.
 d est un déplacement dont la valeur est comprise entre - 127 et 127.
 Le résultat est dans A.

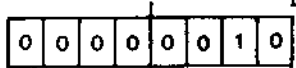
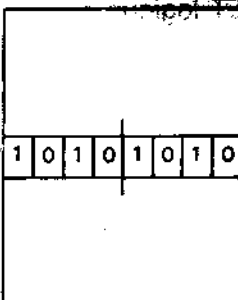
X DMA

A = 12H

AND (IX - 4)

-4

3000H



A = 2H

```
LD A, 12H
LD IX, 3000H
AND (IX - 4)
A = 2H
```

OR

OU logique sur l'accumulateur.

L'adressage peut être immédiat, registre 8 bits, indirect, ou indirect indexé.

La table de vérité de OR au niveau bit est la suivante :

OR	0	1
0	0	1
1	1	1

Cette logique est appliquée entre chaque bit de même rang de l'accumulateur et de l'argument fourni.

Remarque :

Pour l'instruction OR, sauf indication contraire, les indicateurs sont modifiés de la façon suivante :

C=0, N=0, H=0, Z, P et S affectés selon le résultat du OR.

- Adressage immédiat :

OR VAL8

OU logique entre l'accumulateur et VAL8.

Le résultat est dans A.

1	1	1	1	0	0	1	0	A = F2H	
								OR	
0	0	0	0	0	1	0	1	VAL 8 = 5H	LD A, 0F2H
								=	OR 5H
1	1	1	1	0	1	1	1	A = F7H	A = F7H

- Adressage registre 8 bits :

OR X

où X peut être A, B, C, D, E, H ou L.

OU logique entre l'accumulateur et le registre spécifié.

Le résultat est dans A.

0	0	0	1	0	0	1	0	A = 12H	
								OR	
0	1	0	0	0	1	1	0	C = 46H	LD A, 12H
								=	LD C, 46H
0	1	0	1	0	1	1	0	A = 56H	OR C
									A = 56H

- Adressage indirect :

OR (HL)

OU logique entre l'accumulateur et l'octet pointé par HL.

Le résultat est dans A.

0 0 0 1 0 0 1 0 A = 12H

LD A, 12H
LD HL, 3000H
OR (HL)
A = DEH

OR (HL) 3000H 1 1 0 0 1 1 1 0

1 1 0 1 1 1 1 0 A = DEH

- Adressage indirect indexé :

OR (IX + d)
OR (IY + d)

OU logique entre l'accumulateur et l'octet pointé par IX + d ou IY + d.
d est un déplacement dont la valeur est comprise entre -127 et 127.
Le résultat est dans A.

0 0 1 1 0 1 0 0 A = 34H

LD A, 34H
LD IY, 3000H
OR (IY + 5)
A = F4H

IY
OR (IY + 5) + 5 3005H 1 1 0 0 0 0 0 0

1 1 1 1 0 1 0 0 A = F4H

XOR

OU EXCLUSIF sur l'accumulateur.

L'adressage peut être immédiat, registre 8 bits, indirect, ou indirect indexé.
La table de vérité de XOR au niveau bit est la suivante :

XOR	0	1
0	0	1
1	1	0

Remarques :

- a) Si a et b sont des variables binaires, on a :
a XOR b = $\bar{a}b + a\bar{b}$,

c'est-à-dire que $a \text{ XOR } b = 1$ si $a=0$ et $b=1$
ou si $a=1$ et $b=0$.

Cette logique est appliquée entre chaque bit de même rang de l'accumulateur et de l'argument fourni.

b) Pour l'instruction XOR, sauf indication contraire, les indicateurs sont modifiés de la façon suivante :

C=0, N=0, H=0, Z, P et S affectés selon le résultat du XOR.

- Adressage immédiat :

XOR VAL8

OU EXCLUSIF entre l'accumulateur et VAL8.

Le résultat est dans A.

0	0	0	1	0	0	1	0
A = 12H							
1	1	0	0	0	0	0	0
VAL 8 = C0H							
=							
1	1	0	1	0	0	1	0
A = D2H							

LD A, 12H
XOR C0H
A = D2H

- Adressage registre 8 bits :

XOR X

où X peut être A, B, C, D, E, H ou L.

OU EXCLUSIF entre l'accumulateur et le registre spécifié.

Le résultat est dans A.

0	0	0	1	0	0	1	0
A = 12H							
1	1	0	0	0	0	0	0
D = C0H							
=							
1	1	0	1	0	0	1	0
A = D2H							

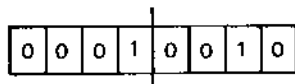
LD A, 12H
LD D, C0H
X OR D
A = D2H

- Adressage indirect :

XOR (HL)

OU EXCLUSIF entre l'accumulateur et l'octet pointé par HL.

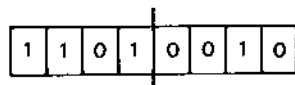
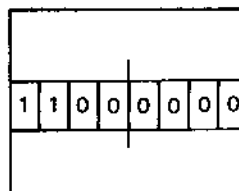
Le résultat est dans A.



A = 12H

XOR (HL)

3000H



A = D2H

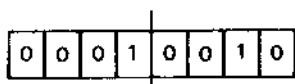
```
LD  A, 12H
LD  HL, 3000H
XOR (HL)
A = D2H
```

ROX

- Adressage indirect indexé :

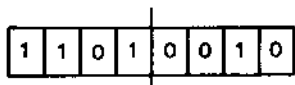
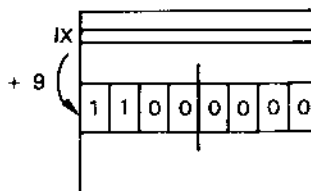
XOR (IX + d)**XOR (IY + d)**

OU EXCLUSIF entre l'accumulateur et l'octet pointé par IX + d ou IY + d.
d est un déplacement dont la valeur est comprise entre - 127 et 127.
Le résultat est dans A.



A = 12H

XOR (IX + 9)



A = D2H

```
LD  A, 12H
LD  IX, 3000
XOR (IX + 9)
A = D2H
```

CPL

Complement Accumulator : Complément à 1 de A.

L'adressage est implicite sur A.

Exemple :

```
A  0 1 1 1 0 1 1 0 = 76H
CPL 1 0 0 0 1 0 0 1 = 89H
```

NEG

Negate Accumulator : Complément à 2 de A.

L'adressage est implicite sur A.

Exemple :

$A = 01110110 = 76H$
 $\bar{A} = 10001001 = 89H$
 $NEG = \bar{A} + 1 = 10001010 = 8AH$

B. COMPARAISONS ET TESTS

Comparaison de l'accumulateur à une valeur exprimée sur 8 bits.

L'adressage peut être immédiat, registre 8 bits, indirect 8 bits ou indirect indexé 8 bits.

Indicateurs : N = 1, C, Z, V, S et H modifiés.

- Adressage immédiat :

CP VAL8

Compare l'accumulateur et VAL8 sans modifier l'accumulateur.

0	0	0	1	0	0	1	0	A = 12H	LD A, 12H CP 13H Indicateurs : 093 H
0	0	0	1	0	0	1	1	VAL 8 = 13H	

- Adressage registre 8 bits :

CP X

où X peut être A, B, C, D, E, H ou L.

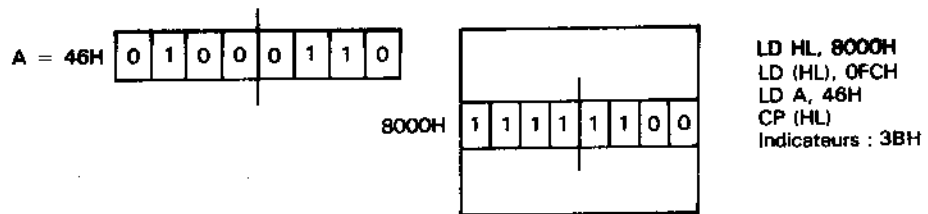
Compare l'accumulateur et le registre indiqué sans modifier l'accumulateur.

0	0	0	1	0	0	1	0	A = 12H	LD A, 12H LD B, 12H CP B Indicateurs : 42H
0	0	0	1	0	0	1	0	B = 12H	

- Adressage indirect :

CP (HL)

Compare l'accumulateur et l'octet pointé par HL sans modifier l'accumulateur.



2T83T T3 T3 T3 T3 T3 T3 T3 T3

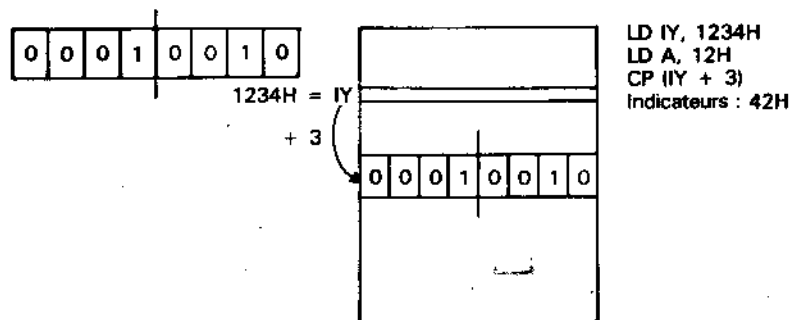
- Adressage indirect indexé :

CP (IX + d)

CP (IY + d)

- 20 Compare l'accumulateur et l'octet pointé par IX + d ou IY + d sans modifier l'accumulateur.

d est un déplacement dont la valeur doit être comprise entre - 127 et 127.



CPI

X 90

ComPare and Increment

Compare le contenu de HL et l'accumulateur.

Positionne le registre Z en conséquence (Z = 1 si (HL) = A et Z = 0 si (HL) < > A). Incrémente HL et décrémente BC. Si BC = 0, l'indicateur P/V = 0.

P/V = 1 pour les autres valeurs de BC.

Supposons que BC = 10 H
HL = 3020H
(HL) = F1H
et A = F1H

Les instructions LD A, F1H
CPI

produiront l'effet suivant :

Z = 1 car (HL) = A

HL = 3021H, BC = 0FH

CPD

ComPare and Decrement

Compare le contenu de HL et l'accumulateur.

Positionne le registre Z en conséquence (Z=1 si (HL)=A et Z=0 si (HL) < > A). Décrémente HL et décrémente BC. Si BC=0, l'indicateur P/V=0. P/V=1 pour les autres valeurs de BC.

Supposons que BC = 10 H
HL = 3020H
(HL) = F1H
et A = 10H

les instructions LD A, 10 H
CPD

produiront l'effet suivant :

Z = 0 car (HL) ≠ A

HL = 301F H et BC = 0 FH

Application :

Recherche d'un octet dans une zone mémoire, à partir de &7000, et sur une longueur de 40 octets.

1		ORG 9000H	
2		LOAD 9000H	
3	9000 013200	LD BC,50	
4	9003 210050	LD HL,5000H	;@de depart
5	9006 3E12	LD A,12H	;octet recherche
6	BOU:	EQU \$	
7	9008 EDA1	CPI	
8	900A 2803	JR Z,TROUVE	;z=1→oct trouve
9	900C EA0890	JP PE,BOU	;P/V=1→BC<>0
10	TROUVE:	EQU \$	
11			;a ce niveau, si Z=1, HL=@+1
12			;de l'octet recherche
13			;si Z=0, l'octet n'est pas trouve
14		END	

C. ROTATIONS ET DÉCALAGES

On distingue les décalages :

- circulaires à gauche et à droite ;
- circulaires à gauche et à droite à travers le bit de retenue ;

- arithmétiques à gauche et à droite ;
- logiques à droite ;
- circulaires en binaire codé décimal à gauche et à droite.

1) Décalages circulaires :

L'adressage peut être registre 8 bits, indirect ou indirect indexé.

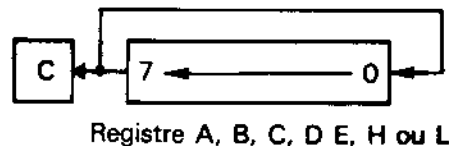
- Adressage registre 8 bits :

RLC X

Rotate Left Circular : Rotation circulaire à gauche où X peut être A, B, C, D, E, H ou L.

Les bits du registre concerné sont décalés vers la gauche selon le dessin ci-dessous :

Les indicateurs H et N sont à 0, S, Z, C et P sont modifiés.

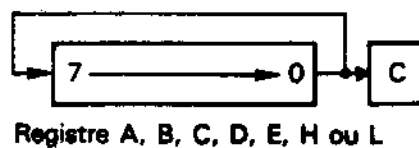


RRC X

Rotate Right Circular : Rotation circulaire à droite où X peut être A, B, C, D, E, H ou L.

Les bits du registre concerné sont décalés vers la droite selon le dessin ci-dessous :

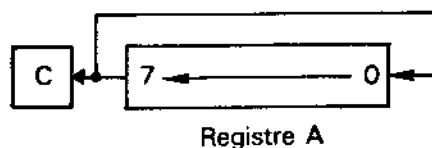
Les indicateurs H et N sont à 0, S, Z, C et P sont modifiés.



RLCA

Rotate Left Circular Accumulator : Rotation circulaire à gauche de l'accumulateur.

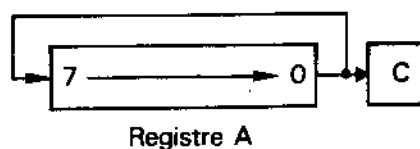
Les bits du registre A sont décalés vers la gauche comme pour RLC A. Les indicateurs H et N sont à 0 ; S, Z et P sont inchangés et C est modifié.



RRCA

Rotate Right Circular Accumulator : Rotation circulaire à droite de l'accumulateur.

Les bits du registre A sont décalés vers la droite comme pour RRC A. Les indicateurs H et N sont à 0 ; S, Z et P sont inchangés et C est modifié.

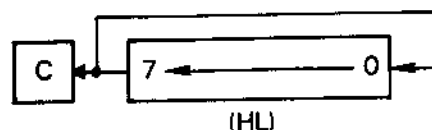


- Adressage indirect :

RLC (HL)

Rotate Left Circular : Rotation circulaire à gauche de la mémoire pointée par HL comme pour RLC A.

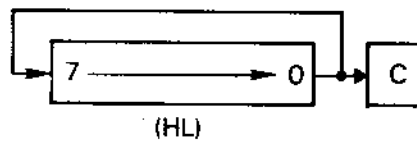
Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



RRC (HL)

Rotate Right Circular : Rotation circulaire à droite de la mémoire pointée par HL comme pour RRC A.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



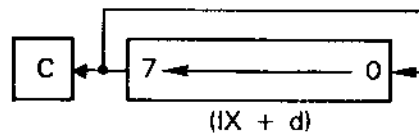
- Adressage indirect indexé :

RLC (IX + d)

Rotate Left Circular : Rotation circulaire à gauche de la mémoire pointée par IX + d comme pour RLC A.

d est un déplacement compris entre -127 et 127.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

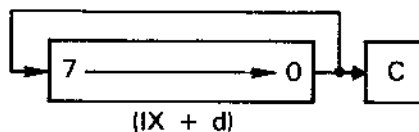


RRC (IX + d)

Rotate Right Circular : Rotation circulaire à droite de la mémoire pointée par IX + d comme pour RRC A.

d est un déplacement compris entre -127 et 127.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



2) Décalages circulaires à travers le bit de retenue :

L'adressage peut être registre 8 bits, indirect ou indirect indexé.

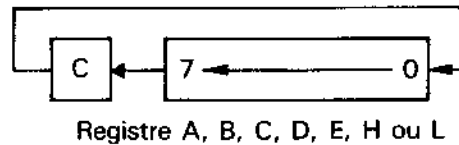
- Adressage registre 8 bits :

RL X

Rotate Left : Rotation circulaire à gauche à travers le bit de retenue, où X peut être A, B, C, D, E, H ou L.

Les bits du registre concerné sont décalés vers la gauche selon le dessin ci-dessous.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

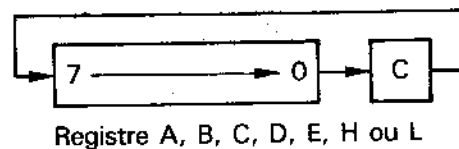


RR X

Rotate Right : Rotation circulaire à droite à travers le bit de retenue.
où X peut être A, B, C, D, E, H ou L.

Les bits du registre concerné sont décalés vers la droite selon le dessin ci-dessous.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

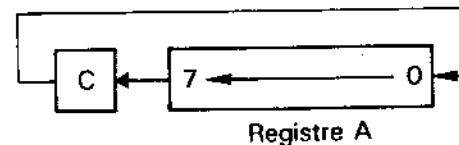


RLA

Rotate Left Accumulator : Rotation circulaire à gauche de l'accumulateur à travers le bit de retenue.

Les bits du registre A sont décalés vers la gauche comme pour RL A.

Les indicateurs H et N sont à 0 ; S, Z et P sont inchangés et C est modifié.

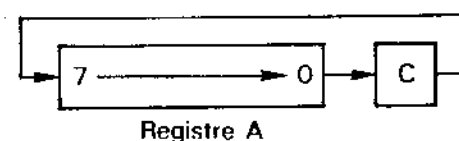


RRA

Rotate Right Accumulator : Rotation circulaire à droite de l'accumulateur à travers le bit de retenue.

Les bits du registre A sont décalés vers la droite comme pour RR A.

Les indicateurs H et N sont à 0 ; S, Z et P sont inchangés et C est modifié.

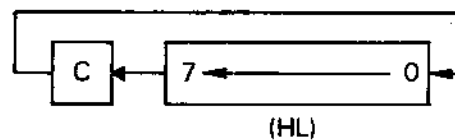


- Adressage indirect :

RL (HL)

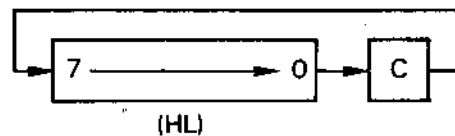
Rotate Left : Rotation circulaire à gauche de la mémoire pointée par HL comme pour RL A.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

**RR (HL)**

Rotate Right : Rotation circulaire à droite de la mémoire pointée par HL comme pour RR A.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



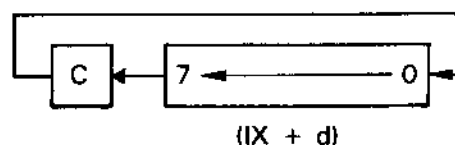
- Adressage indirect indexé :

RL (IX + d)

Rotate Left : Rotation circulaire à gauche de la mémoire pointée par IX + d comme pour RL A.

d est un déplacement compris entre -127 et 127.

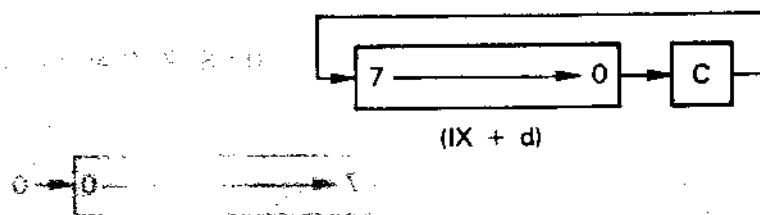
Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

**RR (IX + d)**

Rotate Right : Rotation circulaire à droite de la mémoire pointée par IX + d comme pour RR A.

d est un déplacement compris entre -127 et 127.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



3) Décalages arithmétiques : (HL) AR2

L'adressage peut être registre 8 bits, indirect ou indirect indexé.

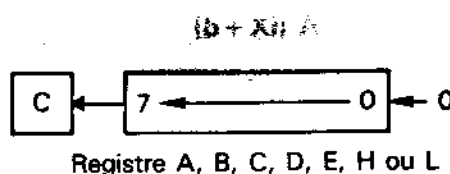
- Adressage registre 8 bits :

SLA X

Shift Left Arithmetic : Décalage arithmétique à gauche, où X peut être A, B, C, D, E, H ou L.

Les bits du registre concerné sont décalés vers la gauche selon le dessin ci-dessous.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



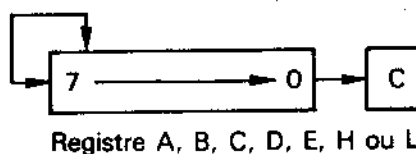
SRA X

Shift Right Arithmetic : Décalage arithmétique à droite

Où X peut être A, B, C, D, E, H ou L.

Les bits du registre concerné sont décalés vers la droite selon le dessin ci-dessous.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

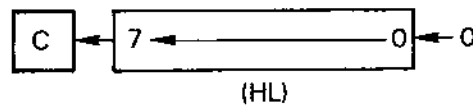


- Adressage indirect :

SLA (HL)

Shift Left Arithmetic : Décalage arithmétique à gauche de la mémoire pointée par HL comme pour SLA A.

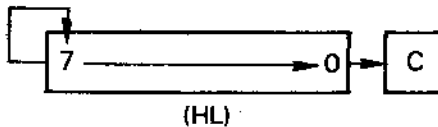
Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



SRA (HL)

Shift Right Arithmetic : Décalage arithmétique à droite de la mémoire pointée par HL comme pour SLA A.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



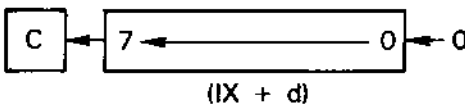
- Adressage indirect indexé :

SLA (IX + d)

Shift Left Arithmetic : Décalage arithmétique à gauche de la mémoire pointée par IX + d comme pour SLA A.

d est un déplacement compris entre -127 et 127.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

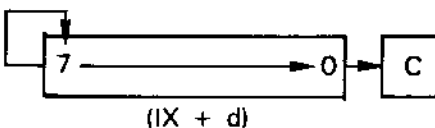


SRA (IX + d)

Shift Right Arithmetic : Décalage arithmétique à droite de la mémoire pointée par IX + d comme pour SLA A.

d est un déplacement compris entre -127 et 127.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



4) Décalages logiques à droite :

L'adressage peut être registre 8 bits, indirect ou indirect indexé.

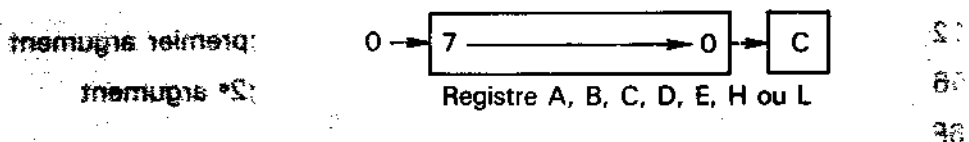
- Adressage registre 8 bits :

SRL X

Shift Right Logical : Décalage logique à droite,
où X peut être A, B, C, D, E, H ou L.

Les bits du registre concerné sont décalés vers la droite selon le dessin ci-dessous.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

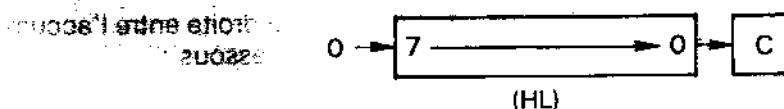


- Adressage indirect :

SRL (HL)

Shift Right Logical : Décalage logique à droite de la mémoire pointée par HL comme pour SRL A.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.



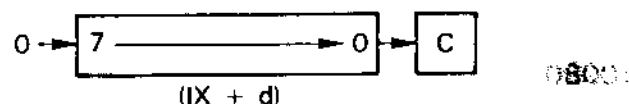
- Adressage indirect indexé :

SRL (IX + d)

Shift Right Logical : Décalage logique à droite de la mémoire pointée par IX + d comme pour SRL A.

d est un déplacement compris entre -127 et 127.

Les indicateurs H et N sont à 0 ; S, Z, C et P sont modifiés.

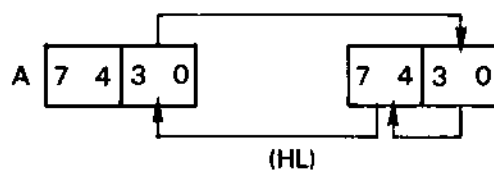


5) Décalages circulaires en binaire codé décimal :

- Adressage indirect sur 8 bits :

RLD

Rotate Left Digit : Rotation BCD à gauche entre l'accumulateur et l'octet pointé par HL selon le dessin page suivante.



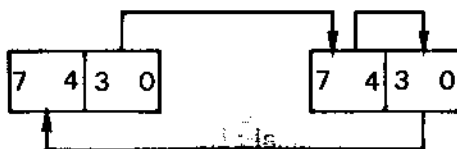
```

1          ORG    9000H
2          LOAD   9000H
3 9000 210080   LD    HL,8000H
4 9003 3612     LD    (HL),12H    ;premier argument
5 9005 3E56     LD    A,56H      ;2° argument
6 9007 ED6F     RLD
7              ;a ce niveau A = 51H
8              ;      HL = 26H
9              END

```

RRD

Rotate Right Digit : Rotation BCD à droite entre l'accumulateur et l'octet pointé par HL selon le dessin ci-dessous :



Exemple :

```

1          ORG    9000H
2          LOAD   9000H
3 9000 210080   LD    HL,8000H
4 9003 3612     LD    (HL),12H    ;premier argument
5 9005 3E56     LD    A,56H      ;2° argument
6 9007 ED67     RRD
7              ;a ce niveau A = 52H
8              ;      HL = 61H
9              END

```

D. OPÉRATIONS SUR LE BIT DE RETENUE C

SCF

Set Carry Flag : Met à 1 le bit indicateur C.

CCF

Complement Carry Flag : Inverse l'état du bit indicateur C.

VII. Manipulation de bits et de chaînes

A. INSTRUCTIONS SUR BITS

Les opérations possibles sont les suivantes :

- test d'un bit ;
- mise à 0 d'un bit ;
- mise à 1 d'un bit.

1) Test de la valeur d'un bit :

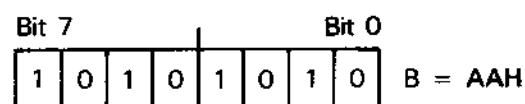
L'adressage peut être registre 8 bits, indirect ou indirect indexé.

- Adressage registre 8 bits :

BIT b,X

Test BIT b of register X : Teste le bit b du registre X où b peut prendre les valeurs 0 à 7 (0 est le poids faible et 7 le poids fort) et X peut être un des registres suivants : A, B, C, D, E, H, L.

Indicateurs : N=0, H=1, C inchangé, Z modifié.



- BIT 5, B donne un indicateur Z = 0
- BIT 4, B donne un indicateur Z = 1

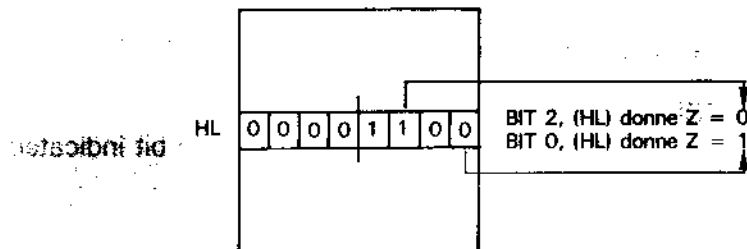
On voit donc que le bit Z prend la valeur inverse du bit testé

- Adressage indirect :

BIT b,(HL)

Test BIT b of location (HL) : Teste le bit b de la mémoire pointée par HL. b peut prendre les valeurs de 0 à 7 (0 est le poids faible et 7 le poids fort).

Indicateurs : N=0, H=1, C inchangé, Z modifié.



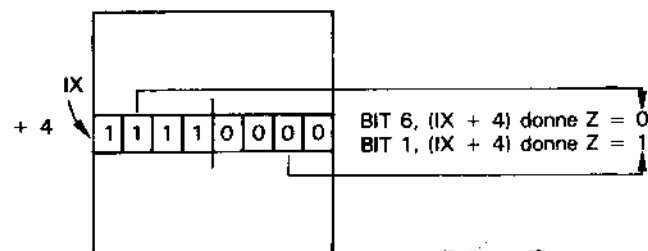
Adressage indirect indexé

BIT b, (IX + d)

BIT b, (IY + d)

Test BIT b of location (IX + d) or (IY + d) : Teste le bit b de la mémoire pointée par IX + d ou IY + d. b peut prendre les valeurs 0 à 7 (0 est le poids faible et 7 le poids fort). d est un déplacement compris entre -127 et 127.

Indicateurs : N=0, H=1, C inchangé, Z modifié.



2) Mise à zéro d'un bit :

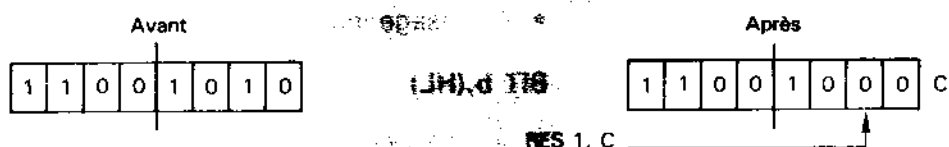
L'adressage peut être registre 8 bits, indirect ou indirect indexé.

• Adressage registre 8 bits

RES b,X

RESet bit b of register X : Met à 0 le bit b du registre X où b peut prendre les valeurs 0 à 7 (0 est le poids faible et 7 le poids fort) et X peut être un des registres suivants : A, B, C, D, E, H, L.

Indicateurs : inchangés.

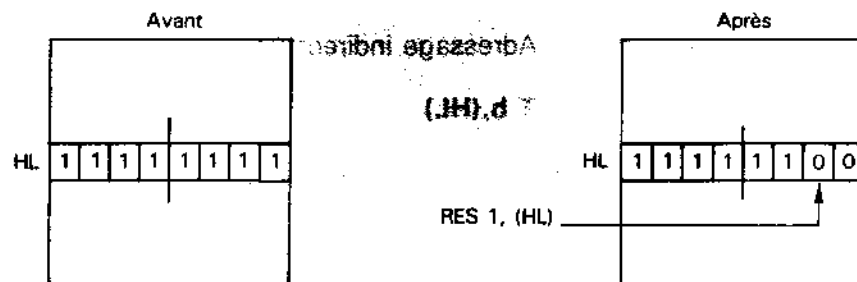


- Adressage indirect :

RES b,(HL)

RESet bit b of location (HL) : Met à 0 le bit b de la mémoire pointée par HL. b peut prendre les valeurs 0 à 7 (0 est le poids faible et 7 le poids fort).

Indicateurs : inchangés.



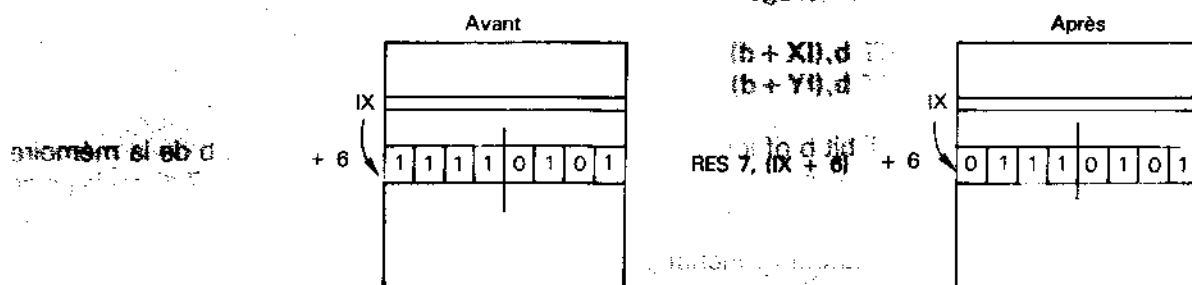
- Adressage indirect indexé :

RES b,(IX + d)

RES b,(IY + d)

RESet bit b of location (IX + d) or (IY + d) : Met à 0 le bit b de la mémoire pointée par IX + d ou IY + d. b peut prendre les valeurs 0 à 7 (0 est le poids faible et 7 le poids fort). d est un déplacement compris entre - 127 et 127.

Indicateurs : inchangés.



3) Mise à 1 d'un bit :

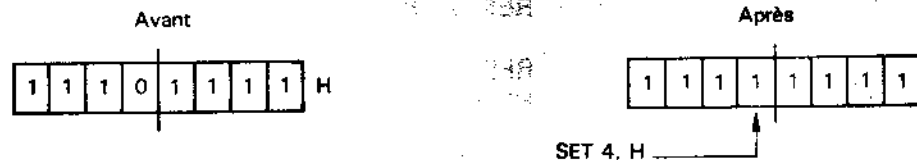
L'adressage peut être registre 8 bits, indirect ou indirect indexé.

- Adressage registre 8 bits :

SET b,X

SET bit b of register X : Met à 1 le bit b du registre X ou b peut prendre les valeurs 0 à 7 (0 est le poids faible et 7 le poids fort) et X peut être un des registres suivants : A, B, C, D, E, H, L.

Indicateurs : inchangés.

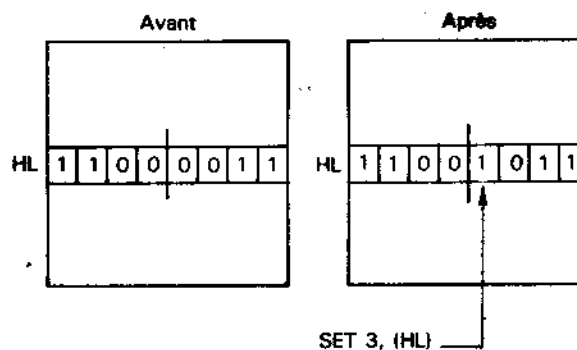


- Adressage indirect :

SET b,(HL)

SET bit b of location (HL) : Met à 1 le bit b de la mémoire pointée par HL. b peut prendre les valeurs 0 à 7 (0 est le poids faible et 7 le poids fort).

Indicateurs : inchangés.



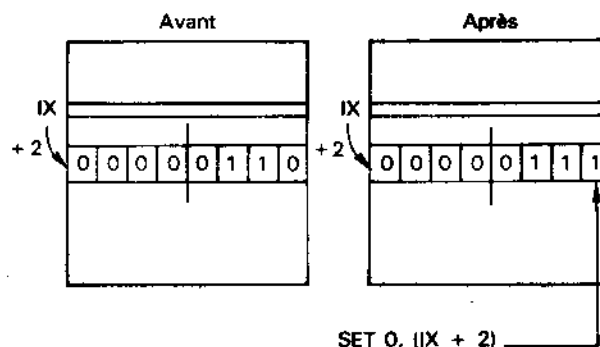
- Adressage indirect indexé :

SET b,(IX + d)

SET b,(IY + d)

SET bit b of location (IX + d) or (IY + d) : Met à 1 le bit b de la mémoire pointée par IX + d ou IY + d. b peut prendre les valeurs 0 à 7 (0 est le poids faible et 7 le poids fort). d est un déplacement compris entre -127 et 127.

Indicateurs : inchangés.



B. INSTRUCTIONS SUR CHAINES

LDIR

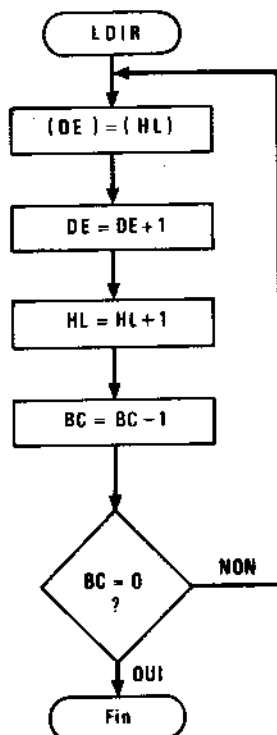
LoaD, Increment And Repeat.

Copie une zone mémoire de longueur BC et commençant à l'adresse HL à partir de l'adresse DE.

HL pointe sur le début de la zone à copier.

Indicateurs : N, P et H=0, les autres ne sont pas modifiés.

Copions la zone mémoire commençant en 7000H, de longueur 30H, à partir de 8000H.



1		ORG	9000H
2		LOAD	9000H
3	9000 013000	LD	BC,30H ;longueur
4	9003 210070	LD	HL,7000H ;@source
5	9006 110080	LD	DE,8000H ;@destination
6	9009 EDB0	LDIR	;copie
7		END	

LDDR

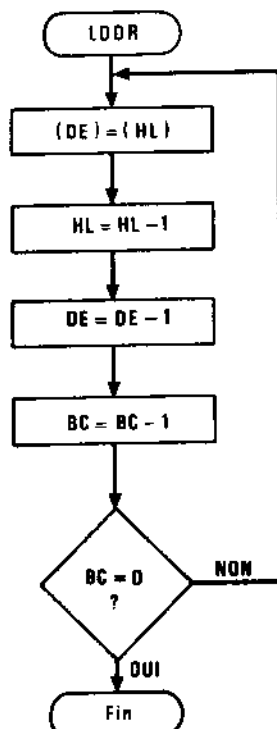
LoaD, Decrement and Repeat.

Copie une zone mémoire de longueur BC et commençant à l'adresse HL à partir de l'adresse DE.

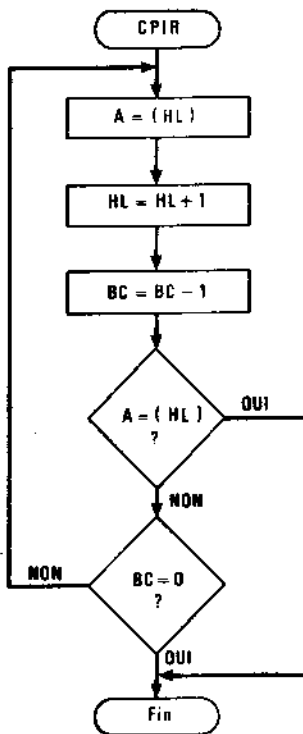
HL pointe sur la fin de la zone à copier.

Indicateurs : N, P et H=0, les autres ne sont pas modifiés.

Copions la zone mémoire finissant en 7040, de longueur 41H en 8000H.



1		ORG	9000H
2		LOAD	9000H
3	9000 014100	LD	BC,41H ;Longueur
4	9003 214070	LD	HL,7040H ;@fin source
5	9006 110080	LD	DE,8000H ;@destination
6	9009 EDB8	LDDR	;copie
7		END	

**CPIR**

ComPare Increment and Repeat.

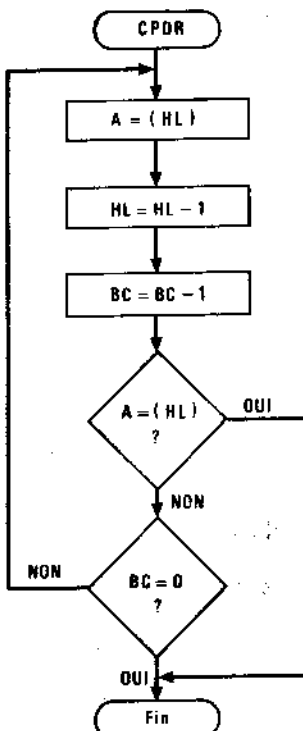
Compare le contenu de HL à l'accumulateur.

Incrémente HL, décrémente BC et répète le test jusqu'à ce que BC = 0 ou que A = (HL). Si A = (HL), l'indicateur Z est mis à 1. Sinon, BC arrive à 0 et l'indicateur V passe à 0.

Indicateurs : N = 1, C inchangé, S, H, V et Z modifiés.

Recherchons l'octet 24H dans une zone mémoire commençant en 8000H et de longueur 40H.

1	ORG	9000H	
2	LOAD	9000H	
3	9000 3E24	LD	A,24H ;Octet recherche
4	9002 210080	LD	HL,8000H ;@debut recherche
5	9005 014000	LD	BC,40H ;longueur recherche
6	9008 EDB1	CPIR	;Recherche
7		END	

**CPDR**

ComPare Decrement and Repeat.

Compare le contenu de HL à l'accumulateur.

Décrémente HL, décrémente BC et répète le test jusqu'à ce que BC = 0 ou que A = (HL). Si A = (HL), l'indicateur Z est mis à 1. Sinon, BC arrive à 0 et l'indicateur V passe à 0.

Indicateur : N = 1, C inchangé, S, H, V et Z modifiés.

Recherchons l'octet 24H dans une zone mémoire finissant en 8040H et de longueur 41H.

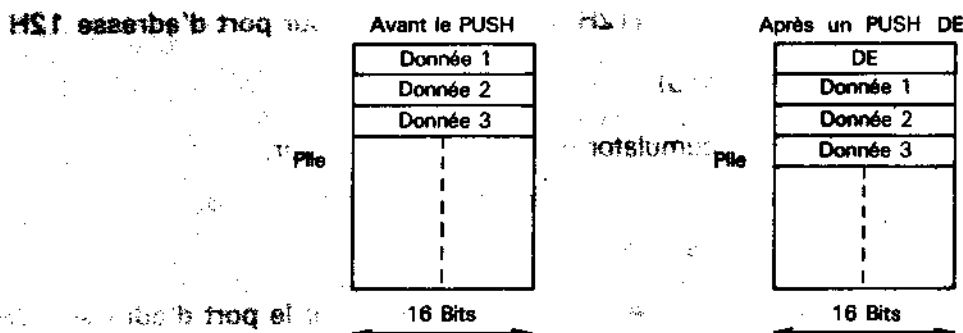
1	ORG	9000H	
2	LOAD	9000H	
3	9000 3E24	LD	A,24H ;Octet recherche
4	9002 214080	LD	HL,8040H ;@fin recherche
5	9005 014100	LD	BC,41H ;Longueur recherche
6	9008 EDB9	CPDR	;Recherche
7		END	

VIII. Opérations sur la pile

Ces instructions permettent d'empiler ou de dépiler un registre pair de la pile.

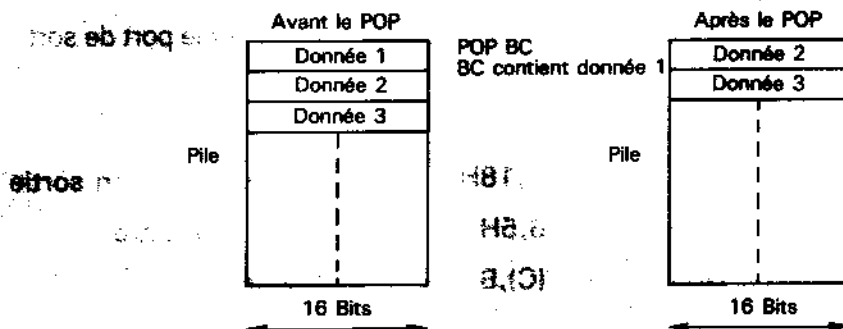
PUSH XX

Où XX peut être un des registres pairs suivants : AF, BC, DE, HL, IX, IY.
Met en pile le registre pair spécifié selon le schéma ci-dessous.



POP XX

Où XX peut être un des registres pairs suivants : AF, BC, DE, HL, IX, IY.
Dépile le registre pair spécifié selon le schéma ci-dessous.



IX. Entrées/Sorties

Ces instructions permettent d'envoyer un ou plusieurs octet(s) vers un périphérique pour les instructions du type OUT, et de recevoir un ou plusieurs octet(s) d'un périphérique pour les instructions du type IN.

L'adressage peut être direct, indirect sur registre 8 bits, indirect sur registre avec incrément/décrément ou indirect sur chaîne.

- Adressage direct :

OUT (VAL8),A

Load OUTput port with Accumulator.

Envoie le contenu de l'accumulateur vers le périphérique d'adresse VAL8.

Indicateurs : non affectés.

LD A, 56H ;Donnée à émettre

OUT (12H),A ;Emission sur port d'adresse 12H

IN A,(VAL8)

Load Accumulator with INput port register.

Lit un octet sur le périphérique d'adresse VAL8.

Indicateurs : non affectés.

IN A,(12H) ;Lecture sur le port d'adresse 12H
;la donnée lue est dans A.

- Adressage indirect sur registre 8 bits :

OUT(C),X

Load OUTput port (C) with register X.

Où X peut être A, B, C, D, E, H ou L.

Envoie le contenu du registre spécifié sur le port de sortie dont l'adresse est donnée dans C.

Indicateurs : non modifiés.

LD C,18H ;Adresse du port en sortie

LD B,5H ;Donnée à émettre

OUT (C),B ;Emission

IN X,(C)

Load register X with INput from port (C).

Où X peut être A, B, C, D, E, H ou L.

Lit un octet sur le périphérique d'adresse (C) et stocke la valeur lue dans le registre spécifié.

LD C,18H ;Adresse du port en entrée

IN (D),C ;Lecture sur le port d'adresse 18H
;La donnée est dans le registre D

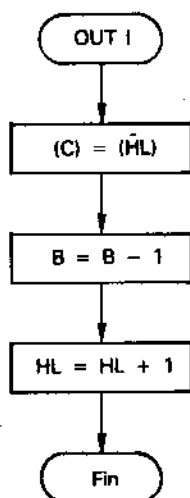
- Adressage indirect sur registre avec incrément/décément :

OUTI

Envoie le contenu de la mémoire pointée par HL (C), décrémente B et incrémente HL.

Indicateurs : N=1, V, S, H et C inchangés, Z modifié.

Par exemple, si nous voulons émettre 15 données situées en mémoire à partir de l'adresse 8000 H, sur le port d'adresse 14 H, il faudra faire :



1	ORG	9000H	
2	LOAD	9000H	
3	9000 0E14	LD	C,14H ;@port sortie
4	9002 060F	LD	B,15 ;Nombre de donnees
5	9004 210080	LD	HL,8000H ;@debut donnees
6	BOUCLE: EQU	\$	
7	9007 EDA3	OUTI	;Sortie d'1 donnee
8	9009 20FC	JR	NZ,BOUCLE
9	END		

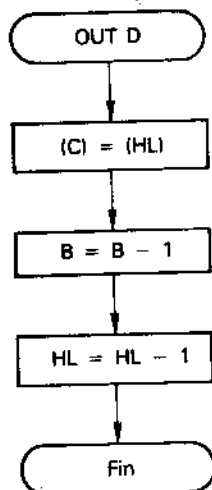
OUTD

OUTput and Decrement.

Envoie le contenu de la mémoire pointée par HL sur le port dont l'adresse est définie par (C), décrémente B et décrémente HL.

Indicateurs : N=1, V, S, H et C inchangés, Z modifié.

Par exemple, si nous voulons émettre 15 données situées en mémoire (la dernière étant en 800FH) sur le port d'adresse 14H, il faudra faire :



1	ORG	9000H	
2	LOAD	9000H	
3	9000 0E14	LD	C,14H ;@port sortie
4	9002 060F	LD	B,15 ;Nombre de donnees
5	9004 210F80	LD	HL,800FH ;@derniere donnee
6	BOUCLE: EQU	\$	
7	9007 EDAB	OUTD	;Emission
8	9009 20FC	JR	NZ,BOUCLE
9	END		

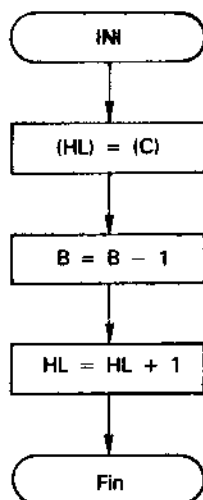
INI

INput and Increment.

Charge la mémoire pointée par HL par la valeur lue sur le port en entrée d'adresse (C), décrémente B et incrémente HL.

Indicateurs : N = 1, V, S, H et C inchangés, Z modifié.

Par exemple, si nous voulons lire quelques données et les stocker en mémoire, à partir de l'adresse 8000H sur un port d'entrée d'adresse 14H, il faudra faire :



1	ORG	9000H	
2	LOAD	9000H	
3	9000 0E14	LD	C,14H ;@port entree
4	9002 060F	LD	B,15 ;Nombre de donnees
5	9004 210080	LD	HL,8000H ;@1 ^{re} donnee
6	BOUCLE: EQU	\$	
7	9007 EDA2	INI	;Lecture
8	9009 20FC	JR	NZ,BOUCLE
9	END		

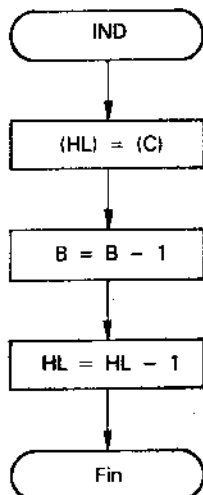
IND

INput and Decrement.

Charge la mémoire pointée par HL par la valeur lue sur le port en entrée d'adresse (C), décrémente B et décrémente HL.

Indicateurs : N = 1, V, S, H et C inchangés, Z modifié.

Par exemple, si nous voulons lire 15 données et les stocker en mémoire, la dernière en 800FH, la première en 8000H sur un port d'entrée d'adresse 14H, il faudra faire :



1	ORG	9000H	
2	LOAD	9000H	
3	9000 0E14	LD	C,14H ;@port entree
4	9002 060F	LD	B,15 ;Nombre de donnees
5	9004 210F80	LD	HL,800FH ;@derniere donnee
6	BOUCLE: EQU	\$	
7	9007 EDAA	IND	;Lecture
8	9009 20FC	JR	NZ,BOUCLE
9	END		

- Adressage indirect sur chaîne :

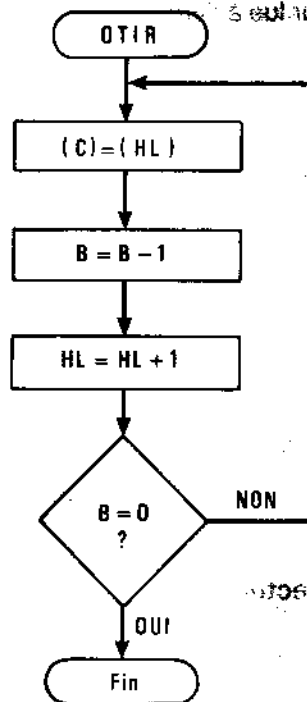
OTIR

Output, Increment and Repeat.

Envoie le contenu de la mémoire pointée par HL sur le port dont l'adresse est définie par (C), décrémente B, incrémente HL et répète ces actions jusqu'à ce que B soit nul.

Indicateurs : N = 1, Z = 1, V, S, H et C inchangés.

Si nous reprenons l'exemple donné pour l'ordre OUTI, nous pouvons utiliser l'ordre OTIR de la façon suivante



1	0000	ORG	9000H	
2	0000	LD	C,14H	;@port sortie
3	0002	LD	B,15	;Nombre de donnees
4	0004	LD	HL,8000H	;@1 ^{re} donnee
5	0007	OTIR		;Emission
6		END		

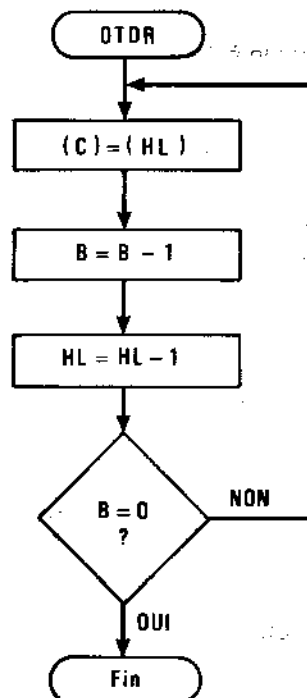
INDR

OTDR

Envoie le contenu de la mémoire pointée par HL sur le port dont l'adresse est définie par (C), décrémente B, décrémente HL et répète ces actions jusqu'à ce que B soit nul.

Indicateurs : N = 1, Z = 1, V, S, H et C inchangés.

Si nous reprenons l'exemple donné pour l'ordre OUTD, nous pouvons utiliser l'ordre OTDR de la façon suivante :



1	0000	ORG	9000H	
2	0000	LD	C,14H	;@port sortie
3	0002	LD	B,15	;Nombre de donnees
4	0004	LD	HL,800FH	;@derniere donnee
5	0007	OTDR		;Emission
6		END		

INIR

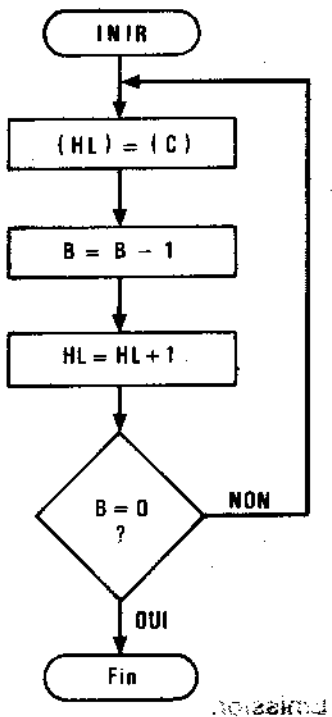
INput, Increment and Repeat.

Lit le port en entrée d'adresse (C) et stocke la valeur lue à l'adresse pointée par HL.

Décrémente N et incrémente HL. Répète ces opérations jusqu'à ce que B soit nul.

Indicateurs : N = 1, Z = 1, V, S, H et C inchangés.

Si nous reprenons l'exemple donné par l'ordre INI, nous pouvons utiliser l'ordre INIR, de la façon suivante :



1	ORG	9000H	
2	LOAD	9000H	
3	9000 0E14	LD	C,14H ;@port entree
4	9002 060F	LD	B,15 ;Nombre de donnees
5	9004 210080	LD	HL,8000H ;@1 ^{re} donnee
6	9007 EDB2	INIR	;Lecture
7	END		

INDR

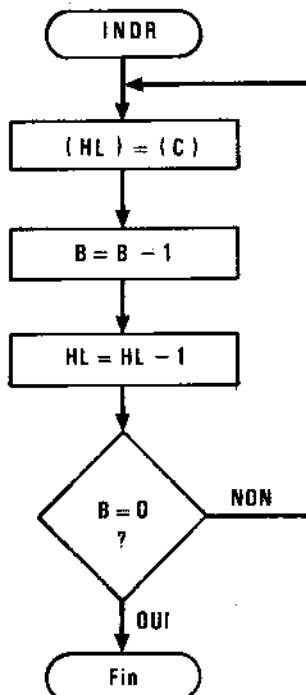
INput, Decrement and Repeat.

Lit le port en entrée d'adresse (C) et stocke la valeur lue à l'adresse pointée par HL.

Décrémente B et décrémente HL. Répète ces opérations jusqu'à ce que B soit nul.

Indicateurs : N = 1, Z = 1, V, S, H et C inchangés.

Si nous reprenons l'exemple donné pour l'ordre IND, nous pouvons utiliser l'ordre INDR de la façon suivante :



1	ORG	9000H	
2	LOAD	9000H	
3	9000 0E14	LD	C,14H ;@port entree
4	9002 060F	LD	B,15 ;Nombre de donnees
5	9004 210F80	LD	HL,800FH ;@derniere donnee
6	9007 EDBA	INDR	;Lecture
7	END		

4/2.4

Liste alphabétique des codes opératoires de l'assembleur Z80

		Número de page
ADC	ADditiOn with Carry. Addition avec retenue.	34
ADD	ADditiOn	34
AND	ET logique	46
BIT	Test de bit.	65
CALL	Appel à un sous-programme en langage machine.	26
CCF	Clear Carry Flag. L'indicateur « retenue » est mis à zéro.	65
CP	ComPare. Comparaison de deux nombres et positionnement des indicateurs en conséquence.	53
CPD	ComPare and Decrement. (DE) < - HL, Comparaison, BC - 1, DE - 1, HL - 1	55
CPDR	ComPare and Decrement Register. Recherche d'un octet en mémoire.	70
CPI	ComPare and Increment. (DE) < - HL, Comparaison, BC - 1, DE + 1, HL + 1	54
CPIR	ComPare and Increment Register. Recherche d'un octet en mémoire.	70
CPL	ComPLement. Complémentation à 1 du registre A.	52
DAA	Decimal Adjustment on A.	45
DEC	DECrement. Décrémente d'1 un registre.	42
DI	Disable Interrupt. Dévalide les interruptions masquables.	7
DJNZ	Decrement and Jump if Not Zero. B = B - 1, Débranchement si B > 0.	24
EI	Enable Interrupt. Valide les interruptions masquables.	7

EX	EXchange. Echange deux registres pairs.	18
EX	AF, AF'. Echange les registres AF et AF'.	17
EX	DE, HL. Echange les registres DE et HL.	17
EXX	Echange BC et BC', DE et DE', HL et HL'.	17
HALT	Stoppe l'exécution d'un programme.	5
IM0	Interrupt Mode 0. Mode d'interruption 0 validé.	8
IM1	Interrupt Mode 1. Mode d'interruption 1 validé.	8
IM2	Interrupt Mode 2. Mode d'interruption 2 validé.	8
IN	INput. Lecture sur un port d'entrée/sortie.	72
INC	INCrement. Incrémente d'1 un registre.	42
IND	INput and Decrement. Lit une donnée sur un périphérique, décrémente B et décrémente HL.	74
INDR	Lit sur un périphérique plusieurs données et les stocke en mémoire.	76
INI	INput and Increment. Lit une donnée sur un périphérique, décrémente B et incrémente HL.	74
INIR	Lit sur un périphérique plusieurs données et les stocke en mémoire	76
JP	JumP. Débranchement absolu.	19
JR	Jump Relative. Débranchement relatif avec un offset de 7 bits.	22
LD	LoaD. Charge un registre.	10, 16
LDD	LoaD and Decrement. (DE) < - (HL), BC - 1, DE - 1, HL - 1	15
LDDR	Transfert d'un bloc de mémoire.	69
LDI	LoaD and Increment. (DE) < - (HL), BC - 1, DE + 1, HL + 1	14
LDIR	Transfert d'un bloc de mémoire.	69
NEG	NEGation. Complément à 2 du registre A.	52
NOP	NO oPeration. Aucune action n'est effectuée.	2
OR	OU logique	48
OTDR	Ecrit sur un périphérique plusieurs données stockées en mémoire.	75
OTIR	Ecrit sur un périphérique plusieurs données stockées en mémoire.	75
OUT	Sortie d'une donnée en mémoire sur un port.	72

OUTD	Sortie d'une donnée en mémoire sur un port. C = @ périph, donnée = (HL), B - 1, HL - 1.	73
OUTI	Sortie d'une donnée en mémoire sur un port. C = @ périph, donnée = (HL), B - 1, HL + 1.	73
POP	Dépile la dernière information entrée dans la pile FIFO.	71
PUSH	Empile une information dans la pile FIFO.	71
RES	Mise à zéro d'un bit.	66
RET	Retour de sous-programme.	30
RETI	Retour de sous-programme d'interruption.	30
RETN	Retour de sous-programme d'interruption.	30
RL	Rotate Left. Rotation vers la gauche d'un registre.	60
RLA	Rotate Left Arithmetic. Rotation arithmétique vers la gauche d'un registre.	59
RLC	Rotate Left Circular. Rotation circulaire vers la gauche d'un registre.	56
RLD	Rotate Left Decimal. Rotation décimale vers la gauche d'un registre.	63
RR	Rotate Right. Rotation vers la droite d'un registre.	60
RRA	Rotate Right Arithmetic. Rotation arithmétique vers la droite d'un registre.	59
RRC	Rotate Right Circular. Rotation circulaire vers la droite d'un registre.	56
RRD	Rotate Right Decimal. Rotation décimale vers la droite d'un registre.	64
RST	ReSTart.	8
SBC	SuBstract with Carry. Soustraction avec retenue.	38
SCF	Set Carry Flag. Indicateur « retenue » mis à 1.	65
SET	Mise à 1 d'un bit.	67
SLA	Shift Left Arithmetic. Décalage arithmétique vers la gauche d'un registre.	61
SRA	Shift Right Arithmetic. Décalage arithmétique vers la droite d'un registre.	65
SRL	Shift Right Logical. Décalage logique vers la droite d'un registre.	63
SUB	SUBstraction. Soustraction sans retenue.	38
XOR	OU exclusif.	50

Nous avons réparti les codes opératoires donnés ci-dessus en neuf groupes :

— Usage général et interruptions.

4/2.5

Cours de programmation

Reportez-vous au chapitre 1.5 de la partie 4 où les concepts de base de la programmation hiérarchisée descendante sont exposés. Ces concepts s'appliquent également au langage Assembleur. Soyez encore plus perfectionniste si vous voulez que la phase de mise au point soit de courte durée et définissez des tâches totalement indépendantes comme pour la programmation en BASIC ; mais ici, pour chaque tâche, définissez les paramètres en entrée, en sortie, et notez si les registres sont ou ne sont pas écrasés. Au besoin, utilisez les instructions d'échanges pour travailler sur les registres secondaires, ou mettez systématiquement les registres en pile à l'entrée d'une tâche par l'instruction PUSH et dépilez ces registres dans l'ordre inverse d'empilement à la sortie de la tâche par l'instruction POP.

Plus encore que pour la programmation en BASIC, passez la plus grande partie du temps sur l'analyse : définition détaillée du problème et division du problème en sous problèmes indépendants et plus simples à mettre en œuvre.

D'autres langages « évolués » (Logo, Forth, etc.) représentent d'autres combinaisons d'instructions Z 80, qui présentent certains avantages et certains inconvénients par rapport au Basic. La grande force de ces langages évolués dont le Basic est le plus répandu est leur facilité d'apprentissage : les instructions sont formulées pratiquement « en clair », à l'aide de mots anglais très simples et de règles peu contraignantes. Les erreurs commises sont signalées au programmeur qui peut les corriger commodément.

Les choses sont très différentes en langage machine : l'équivalent du programme Basic d'une ligne 10 GOTO 10 s'écrira :

195 64 156 ou C3 40 9C

C'est déjà moins parlant, mais il y a pire : il ne suffit pas de le frapper au clavier puis de faire RUN pour le lancer ! Il faut réfléchir pour déterminer à quel endroit de la mémoire on ira l'implanter, et « appeler » cette « adresse » pour déclencher l'exécution. En cas d'erreur ou de « bouclage », on ne pourra pas reprendre « la main » par un BREAK : l'ordinateur sera bloqué ou « planté » et pour le débloquer, il faudra effacer le programme !

Pas question non plus de faire LIST pour examiner le programme : il faut aller inspecter la mémoire à l'aide d'un logiciel « moniteur » ou avec des PEEK !

La contrepartie de cette lourdeur est que l'on peut réaliser en langage machine des opérations inaccessibles au Basic : seules l'imagination et la compétence du programmeur limitent les possibilités logicielles, pourvu que l'on ne cherche pas à aller plus loin que ne le permet le matériel (le langage machine ne fera pas apparaître de la couleur sur un écran monochrome !). Egalement, à possibilités égales, un programme machine est bien plus rapide que son équivalent Basic : n'oublions pas que l'interpréteur Basic doit traduire chaque instruction en langage machine avant de l'exécuter... Si une boucle FOR-NEXT renouvelle mille fois la même tâche, la séquence de lignes correspondante sera traduite mille fois : c'est neuf cent quatre-vingt-dix-neuf fois de trop, ou même mille fois puisqu'il est possible d'écrire directement cette « routine » en langage machine ! Cette économie de temps s'accompagne aussi d'une importante diminution d'encombrement mémoire : bien que plus rapide, un programme machine occupera considérablement moins de place que son équivalent Basic.

Le langage machine se prête donc particulièrement bien à l'écriture de logiciels exceptionnellement performants, mais au prix d'un travail considérable. Une forme de simplification consiste à écrire le langage machine non plus « à la main » comme nous allons apprendre à le faire, mais à l'aide d'un logiciel « assembleur » : il s'agit d'un programme écrit dans un langage a priori quelconque, qui peut « traduire » en langage machine des instructions écrites à l'aide de « mnémoniques », abréviations rappelant un peu les « mots-clés » du Basic. En assembleur, notre petit programme exemple s'écrirait : **JP 40000**, ce qui est déjà plus parlant si on considère que JP est l'abréviation du mot anglais JUMP qui signifie « sauter » !

Un bon logiciel assembleur est généralement accompagné d'un « désassembleur » capable de reconstituer un « listing » de mnémoniques à partir du programme machine terminé (utile par exemple pour étudier le contenu de la ROM de l'Amstrad, ou tout simplement pour contrôler le travail de l'assembleur).

Un débogueur est également utile, permettant de modifier un programme machine sans le récrire en entier, et de le faire « tourner » pas à pas en évitant les blocages sur erreurs.

Ces puissants outils transforment l'Amstrad en un véritable « système de développement de Z 80 », analogue à ce qu'utilisent les programmeurs professionnels. Pour les rentabiliser, il faut toutefois avoir à écrire des logiciels machine de plusieurs centaines ou milliers d'octets, ce qui suppose une profonde maîtrise de la programmation du Z 80.

Si vous lisez ces lignes, c'est fort probablement parce que vous ne maîtrisez précisément pas cette discipline ! Commençons donc par le commencement : entre le « tout Basic » et le « tout assembleur » existe un domaine relativement facile à explorer, et qui peut vous apprendre beaucoup.

Cherchons donc à écrire de courts « sous-programmes » ou « routines » en langage machine, que le Basic pourra « appeler » pour « sous-traiter » des tâches qu'il ne sait pas exécuter, ou qu'il exécute dans de mauvaises conditions.

Si alors les horizons ouverts par la programmation machine vous séduisent, vous souhaiterez probablement cesser d'assembler vos instructions à la main et vous vous offrirez un assembleur-désassembleur-debugger... Mais ne brûlez pas les étapes : c'est en forgeant que l'on devient forgeron, mais c'est aussi en assemblant manuellement de courtes routines que l'on assimile les notions de base sans lesquelles on ne pourra rien tirer de bon d'un assembleur, aussi performant soit-il.

II. Votre première routine machine

Puisque nous avons vu que l'Amstrad attend des instructions Basic dès sa mise sous tension, il est clair que nous allons devoir « partir » du Basic pour accéder au langage machine.

Nous utiliserons donc les instructions suivantes :

POKE pour « implanter » des « octets » en mémoire

PEEK pour aller lire en mémoire

MEMORY pour réserver des zones de mémoire pour le langage machine

CALL pour « appeler » nos routines en langage machine

L'Amstrad venant d'être mis sous tension, frappons la commande :

PRINT HIMEM (+ ENTER)

Une valeur apparaît, par exemple **43903**, qui indique l'adresse mémoire (c'est-à-dire le numéro de « case ») maximale dans laquelle le Basic peut

avoir à écrire : en dessous de 43903, donc, la mémoire n'est pas « sûre » et ce qu'on y écrit risque tôt ou tard de se trouver « écrasé » par autre chose.

Protégeons donc une zone qui nous sera strictement réservée, en frappant :

MEMORY 39999 (+ ENTER)

Il est facile de vérifier que HIMEM vaut dorénavant 39999 : toute la zone mémoire entre 40000 et 43903 soit plus de 3 Ko est maintenant à notre disposition sans surprise possible.

Un programme machine n'est rien d'autre qu'une suite d'octets, c'est-à-dire de valeurs comprises entre 0 et 255 (en décimal) ou entre 0 et FF (en hexadécimal) ou encore entre 00000000 et 11111111 (en binaire).

Notre exemple (195 64 156) n'échappe pas à cette règle même s'il ne comporte que trois octets en tout et pour tout !

Il s'agit donc de l'implanter en mémoire, puis de l'exécuter. Faisons donc successivement :

POKE 40000, 195 (+ ENTER)

POKE 40001, 64 (+ ENTER)

POKE 40002, 156 (+ ENTER)

Nos trois octets de programme machine sont désormais rangés à la suite les uns des autres à partir de l'adresse 40000, qu'il ne s'agit plus d'oublier car elle devra être spécifiée pour « lancer » l'exécution.

Faites **NEW** si vous voulez pour bien montrer que vos octets sont en sécurité, puis :

PRINT PEEK (40000) (+ ENTER)

PRINT PEEK (40001) (+ ENTER)

PRINT PEEK (40002) (+ ENTER)

Notre programme est bien là !

Pour le lancer, il suffit de faire :

CALL 40000 (+ ENTER)

Allons-y, et constatons qu'il ne se passe rien. Ce qui est pire, c'est que le clavier n'agit plus et que même la touche **BREAK** est inopérante : l'ordinateur est « planté » ! Ce n'est pas grave, mais pour reprendre « la main », il va falloir faire un « reset », c'est-à-dire un **CTRL-SHIFT-ESC** qui effacera toute la mémoire, y compris notre programme : n'oublions pas qu'en faisant un **CALL**, nous avons arrêté l'exécution du programme « interpréteur Basic » de la ROM pour travailler complètement « sans filet ». Perdre un programme de trois octets n'est rien, en perdre un de 300 ou 500 octets, représentant des heures de travail, est une tout autre affaire : il faudra être prudent lorsque vous en arriverez là et effectuer des sauvegardes !

En fait, ce qui est arrivé était voulu : ce programme est une « boucle » utilisant l'instruction de « saut » **JP NN**.

195 est le code de l'instruction JP (jump); 64 et 156 indiquent l'adresse à laquelle il faut faire « sauter » l'exécution, soit $64 + (256 \times 156) = 40000$.

Notre CALL 40000 a donc appelé une instruction appelant elle-même sa propre exécution !

En Basic, un RUN déclenchant l'exécution de 10 GOTO 10 aurait le même effet, à ceci près que BREAK serait efficace, et que cette ligne occupe largement plus de trois octets en mémoire. Toutes les instructions machine n'occupent pas le même nombre d'octets : elles sont plus ou moins longues selon leur complexité. Cependant, toute suite d'octets ne correspond pas nécessairement à un programme machine vraisemblable : un CALL vers une adresse mémoire quelconque a toutes les chances de « planter » l'ordinateur : sachez exactement ce que vous faites... Certaines adresses de la ROM correspondent toutefois à des routines pré-programmées qu'il peut être utile d'exploiter (Voir partie 4, chap. 2.7).

III. Un programme machine plus utile

Pratique pour faire comprendre un certain nombre de notions importantes, ce premier programme n'a certes pas une grande utilité pratique !

Attaquons-nous donc maintenant à la résolution d'un problème précis, difficile à traiter en Basic. Vous connaissez certainement le code ASCII, qui affecte un octet déterminé à chaque caractère susceptible d'être affiché par l'Amstrad : en décimal, le « A » est représenté par l'octet 65, et le « Z » par 90, par exemple. (Voir les fonctions BASIC, ASC et CHR\$.)

Le tableau suivant rappelle qu'un octet peut être représenté sous la forme de huit « bits » pouvant être soit à 1 soit à 0, et dont chacun possède un « poids » bien précis : celui de droite, le moins « significatif », vaut 1 lorsqu'il est à 1 et évidemment 0 lorsqu'il est à 0.

	27	26	25	24	23	22	21	20	
	128	64	32	16	8	4	2	1	
A	0	1	0	0	0	0	0	1	65
Z	0	1	0	1	1	0	1	0	90

Celui de gauche, le plus significatif, vaut 128 lorsqu'il est à 1 et toujours 0 lorsqu'il est à 0. De droite à gauche, les huit bits de l'octet valent respectivement 1, 2, 4, 8, 16, 32, 64 et 128, soit un total maximal de 255 s'ils sont tous à 1.

Or, 127 combinaisons suffisent largement pour coder tous les caractères de l'alphabet informatique normalisé : lettres majuscules et minuscules, chiffres et signes divers. Sept bits suffisent donc, et si l'on persiste à travailler avec des octets de huit bits (le matériel est fait pour cela), le bit de poids fort (128) reste normalement à 0.

Il a été imaginé de se servir de ce huitième bit pour contrôler les altérations pouvant survenir dans les transmissions de données : ce bit, appelé *bit de parité*, est mis à 1 ou à 0 selon le nombre de bits qui sont eux-mêmes à 1 dans le « septet » représentant le caractère. Le Minitel, par exemple, travaille en *parité paire* : le huitième bit est positionné de façon à ce que le nombre total de bits à 1 dans l'octet soit *pair*. A l'arrivée, on compte les bits à 1 et si le nombre trouvé est impair, on déduit qu'il y a eu erreur dans la transmission !

Bien évidemment, cela n'a rien à voir avec la « parité » de la valeur décimale de l'octet, au sens arithmétique du terme. Le tableau suivant montre le résultat de ce traitement sur les octets valant, en décimal, de 0 à 8. Cette valeur d'origine se trouve augmentée de 128 lorsque le bit de parité doit être mis à 1. En Basic, l'adjonction du bit de parité serait une opération lourde et lente. En langage machine, une routine de douze octets suffit, soit l'équivalent d'une seule très courte ligne de Basic.

Bit de parité									
	27	26	25	24	23	22	21	20	
Avant	128	64	32	16	8	4	2	1	Après
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	1	129
2	1	0	0	0	0	0	1	0	130
3	0	0	0	0	0	0	1	1	3
4	1	0	0	0	0	1	0	0	132
5	0	0	0	0	0	1	0	1	5
6	0	0	0	0	0	1	1	0	6
7	1	0	0	0	0	1	1	1	135
8	1	0	0	0	1	0	0	0	136
9	0	0	0	0	1	0	0	1	9
etc.									

Principe de codage en parité paire : le huitième bit de l'octet sert de bit de parité

Etudions donc en détail ce programme sous une forme qui vous sera bientôt familière :

40000	LD A,0	62	0	
40002	AND A	167		
40003	JP PE 40008	234	72	156
40006	SET 7,A	203	255	
40008	LD 40001,A	50	65	156
40011	RET	201		

La colonne de gauche indique l'adresse mémoire à laquelle est implanté le premier octet (ou l'octet unique) de chaque instruction : la première instruction comportant deux octets, on comprend pourquoi on passe de 40000 à 40002 !

Par contre, la seconde instruction ne possédant qu'un octet, on passe de 40002 à 40003.

Insistons bien sur le fait qu'une routine machine est en général écrite pour être implantée à partir d'une adresse mémoire bien déterminée : la déplacer équivaldrait à renuméroter les lignes d'un programme Basic sans corriger les GOTO et les GOSUB !

Certaines techniques de programmation permettent d'écrire des routines « relogeables », pouvant fonctionner n'importe où en mémoire, mais on ne peut pas toujours les employer.

La seconde colonne donne le *mnémonique* de l'instruction, c'est-à-dire la description très abrégée de ce qu'elle fait : c'est sous cette forme qu'on programme en « assembleur ».

Pour programmer « à la main » en langage machine, il faut faire le travail du logiciel assembleur à l'aide d'un crayon et de papier : aller chercher dans les recueils d'instructions les *codes opératoires* (octets) correspondant à chaque mnémonique, et calculer à l'aide de tables ou à la machine les octets servant à « pointer » des adresses mémoire. C'est long et fastidieux, mais extrêmement formateur : pour bien utiliser un assembleur, il faut savoir comment il fonctionne !

La première instruction du programme charge la valeur 0 dans le « registre » A du Z 80. Ce microprocesseur possède plusieurs de ces registres, 22 pour être précis. On trouvera leur description précise en partie 4, chapitre 2.2, mais sachez pour le moment que le plus utilisé est A, nommé « accumulateur ».

En fait, la valeur 0 sera remplacée juste avant l'exécution par l'octet à traiter : il suffira pour cela de faire, en Basic, POKE 40001, (octet) juste avant le CALL 40000.

Vous pouvez donc mettre n'importe quelle valeur inférieure à 256 à la place de 0, mais ne touchez pas à 62 qui est le code opératoire de l'instruction LD, A,N (LOAD A with N) qui charge A avec N.

L'instruction suivante, AND A, est utilisée de façon particulière : normalement, AND effectue un ET logique bit à bit entre l'octet contenu dans A, et celui contenu dans le registre spécifié. Ici, le ET est effectué entre le contenu de A et le contenu de A ! Bien évidemment, le contenu de A n'est pas modifié, mais l'exécution de l'instruction AND positionne le « drapeau » de parité du Z 80 : cet indicateur se « souviendra » donc de la parité du contenu de A, c'est-à-dire s'il contient un nombre pair ou impair de bits.

L'instruction suivante fonctionne comme un IF-THEN en Basic : JP PE 40008 (jump to 40008 if parity even) déclenche un saut à l'instruction logée à l'adresse 40008 si et seulement si le drapeau de parité est positionné sur « pair ». Dans le cas contraire, c'est l'instruction suivante qui est exécutée, celle qui est logée en 40006.

234 est le code opératoire de **JP PE NN**, tandis que 72 et 156 représentent l'adresse 40008 ($72 + (256 \times 156)$) : l'octet le moins significatif de l'adresse est toujours placé en premier, suivi de l'octet le plus significatif, dont le « poids » est 256.

Si la parité est impaire (nombre impair de bits), l'instruction **SET 7,A** est exécutée : elle force à 1 le bit de poids fort (N° 7) du registre A, ce qui rend pair le nombre total de bits.

Si la parité était déjà paire, c'est l'instruction **LD 40001,A** qui est exécutée : le contenu de A est chargé dans l'adresse mémoire 40001 ($65 + (256 \times 156)$) où le Basic pourra venir le chercher par un **PEEK 40001**.

Encore faut-il pour cela revenir en Basic, ce qui est déclenché par l'instruction **RET**, équivalente au **RETURN** du Basic. Le programme suivant se charge à la fois de l'implantation en mémoire de cette routine (lignes 10 à 60), et de son exploitation pratique (lignes 100 à 140).

```

10 MEMORY 39999
20 DATA 62,0,167,234,72,156,203,255,50,65,156,201
30 FOR f=40000 TO 40011
40 READ c
50 POKE f,c
60 NEXT f
70 REM=====
80 REM (c)1988 Patrick GUEULLE
90 REM=====
100 FOR f=0 TO 127
110 POKE 40001,f
120 CALL 40000
130 PRINT f;" ";PEEK(40001)
140 NEXT f

```

Programme Basic d'implantation et de lancement

Dans un premier temps, de la place mémoire est protégée, bien plus qu'il n'en faut d'ailleurs, mais au diable l'avarice !

Les douze octets, rangés dans une ligne DATA, sont alors implantés en mémoire par une suite de **POKE** déclenchés par une boucle **FOR-NEXT** opérant directement sur les adresses mémoire 40000 à 40011.

La routine peut maintenant être appelée à volonté par un simple **CALL 40000**, à charge pour l'utilisateur de placer d'abord l'octet à traiter en 40001 par un **POKE**.

C'est ce que fait la seconde partie du programme, qui affiche à l'écran les octets 0 à 127 enrichis d'un bit de parité paire.

Comparez donc le début de cette liste avec la colonne de droite du tableau page 6 et, si le cœur vous en dit, essayez de faire aussi bien et aussi rapide sous Basic : bonne chance !

IV. Vos outils de programmation

Le microprocesseur Z 80 offre tant de possibilités que ce n'est qu'après plusieurs années de pratique que l'on peut prétendre en maîtriser vraiment la programmation.

Il est cependant possible de l'utiliser de façon déjà intéressante en se limitant à une partie seulement de ses fonctions.

Le tableau suivant rassemble une sélection de trente instructions, choisies en fonction de leur similitude avec le Basic. Nous vous conseillons de les utiliser en priorité dans l'écriture de vos premières routines personnelles.

Table des 30 instructions machine « préférentielles »

Mnémonique	Code	Equivalent Basic
ADD A,B	128	LET A = A + B
ADD A,N	198	LET A = A + OMS
CALL NN	205	GOSUB (OMS + 256 × OPS)
CALL NZ,NN	196	IF Z = 0 THEN GOSUB (OMS + 256 × OPS)
CALL Z,NN	204	IF Z = 1 THEN GOSUB (OMS + 256 × OPS)
CP B	184	IF B = A THEN LET Z = 1
CP N	254	IF A = N THEN LET Z = 1
DEC A	061	LET A = A - 1
DEC B	005	LET B = B - 1
IN A,(N)	219	IN (N)
INC A	060	LET A = A + 1
INC B	004	LET B = B + 1
JP NN	195	GOTO (OMS + 256 × OPS)
JP NZ,NN	194	IF Z = 0 THEN GOTO (OMS + 256 × OPS)
JP Z,NN	202	IF Z = 1 THEN GOTO (OMS + 256 × OPS)
LD A,B	120	LET A = B
LD A,N	062	LET A = OMS
LD A,(NN)	058	LET A = PEEK (OMS + 256 × OPS)
LD B,A	071	LET B = A
LD B,N	006	LET B = OMS
LD (NN),A	050	POKE (OMS + 256 × OPS),A
NOP	000	(REM)
OUT(N),A	211	OUT N,A
RET	201	RETURN
RET NZ	192	IF Z = 0 THEN RETURN
RET Z	200	IF Z = 1 THEN RETURN
SBC A,B	152	LET A = A - B - Cy
SBC A,N	222	LET A = A - OMS - Cy
SUB B	144	LET A = A - B
SUB N	214	LET A = A - OMS

OMS = octet le moins significatif (poids 1)
OPS = octet le plus significatif (poids 256)

d'une valeur comprise
entre 0 et 65535

~~cela ne vous empêche évidemment pas~~ de recourir à des instructions plus spécifiques, mais il vous faudra alors faire l'effort de comprendre leur fonctionnement dans le détail, ce qui n'est pas toujours simple !

La partie 4, chapitre 2.3 est là pour vous y aider, ainsi que le tableau suivant qui donne la signification des principales abréviations entrant dans la composition de mnémoniques d'assemblage.

Pour incorporer dans un programme les instructions que vous aurez choisies, vous devrez vous reporter à une table des *codes opératoires* donnant les octets à utiliser pour représenter, en mémoire, chaque instruction.

Signification des abréviations utilisées dans les mnémoniques

Abréviation	Signification	Equivalence Basic
ADC	Addition avec retenue	+
ADD	Addition sans retenue	+
AND	ET logique bit à bit	AND
BIT	Lecture d'un bit spécifié d'un octet donné	spécial
CALL	Appel sous-programme	GOSUB
CP	Comparaison	IF-THEN
CC	Complémentation	spécial
DAA	Conversion en BCD	spécial
DEC	Décrémenter	LET L = L - 1
DI	Annulation interruptions	spécial
DJNZ	Décrémenter B et brancher si B = 0	LET B = B - 1 et GOTO
EI	Autorisation interruptions	spécial
EX	Echanger deux opérandes	double LET
EXX	Echange complexe	spécial
HALT	Arrêt machine	STOP
IM	Choix mode interruption	spécial
IN	Entrée par port	IN
INC	Incrémentation	LET L = L + 1
IND	Entrée + décrémenter	spécial
INI	Entrée + incrémentation	spécial
JP ou JR	Branchement	GOTO
LD	Chargement	LET
LDD	Chargement + décrémenter	spécial
LDI	Chargement + incrémentation	spécial
NEG	Complémentation à 2	spécial
NOP	Pas d'opération	(REM)
OR	OU logique bit à bit	OR
OTD	Sortie + décrémenter	spécial
OTI	Sortie + incrémentation	spécial
OUT	Sortie sur port	OUT
OUTD	Sortie + décrémenter	spécial
OUTI	Sortie + incrémentation	spécial
POP	Récupérer données pile	spécial
PUSH	Entrer données pile	spécial
RES	Mettre bit spécifié à 0	spécial (LET)
RET	Retour au prog principal	RETURN

Abréviation	Signification	Equivalence Basic
RL	Permutation de bits à gauche	spécial
RR	Permutation de bits à droite	spécial
RST	Repartir à adresse spécifiée	spécial (RUN)
SBC	Soustraction avec retenue	—
SCF	Forcer la retenue à 1	spécial (LET)
SET	Mettre à 1 bit spécifié	spécial (LET)
SLA	Décalage de bits à gauche	spécial
SRA	Décalage de bits à droite	spécial
SRL	Décalage de bits à droite	spécial
SUB	Soustraction sans retenue	—
XOR	OU exclusif bit à bit	spécial

Les premières lettres des mnémoniques des instructions Z 80 sont l'abréviation de mots anglais décrivant la fonction réalisée. Il suffit d'en connaître la signification, ainsi que les conventions de notation des opérandes (registres, cellules mémoire, ports) pour comprendre l'essentiel du fonctionnement de l'instruction. Pour les détails, il faudra bien sûr consulter le recueil complet des instructions Z 80 (chapitre 4/2.3 de cet ouvrage).

Abréviation	Signification
()	Cellule mémoire dont l'adresse est stockée, sous la forme OMS OPS, dans les deux registres ou les deux nombres mentionnés entre les parenthèses
A	Registre A (accumulateur)
B	Registre B
C	Registre C
D	Registre D
E	Registre E
H	Registre H
L	Registre L
N	Valeur numérique (octet) contenue dans l'instruction après le code opératoire
IX	Registre d'index IX (à deux octets)
IY	Registre d'index IY (à deux octets)
HL	Paire de registres H et L utilisés ensemble
BC	Paire de registres B et C utilisés ensemble
DE	Paire de registres D et E utilisés ensemble
SP	Registre SP (pointeur de pile)
d	Déplacement (quantité contenue, sous forme d'un octet, dans une instruction et intervenant selon des modalités diverses, dans le calcul d'une adresse (à étudier cas par cas).
NN	Paire de valeurs numériques (2 octets OMS et OPS)
Z	Si drapeau Z à 1 (résultat nul ou égalité)
NZ	Si drapeau Z à 0 (résultat 0 ou inégalité)
C	Si drapeau C à 1 (retenue)
NC	Si drapeau C à 0 (pas de retenue)
M	Si drapeau S à 1 (résultat négatif)
P	Si drapeau S à 0 (résultat positif)
AF	Paire de registres A et F
AF'	Paire de registres A' et F'

Abréviation	Signification
NC	Si drapeau C à 0 (pas de retenue)
PO	Si drapeau P à 0 (résultat pair ou pas de dépassement)
PE	Si drapeau P à 1 (résultat impair ou dépassement)
0	Bit N° 0 de l'octet spécifié (registre)
...	
7	Bit N° 7 de l'octet spécifié (registre)
10H	Adresse 16 (après RST)
18H	Adresse 24
20H	Adresse 32
28H	Adresse 40
30H	Adresse 48
38H	Adresse 56
8	Adresse 8
()	Numéro d'un port d'entrée-sortie
.	Indique un registre du second groupe

Ces abréviations sont employées dans la suite des mnémoniques pour en préciser le sens, ou pour désigner les opérandes (registres ou cellules mémoire sur lesquels agit l'instruction).

Une telle table existe en annexe 2 (partie 11), un classement alphabétique facilitant la recherche d'après les mnémoniques.

Les codes opératoires y sont cependant présentés en *hexadécimal*, forme universellement adoptée par les programmeurs avertis mais qui risque de dérouter encore un peu plus le débutant en langage machine !

La table de conversion de la page suivante est donc là pour permettre d'opérer rapidement une conversion entre des octets exprimés en décimal ou en « hexa » : vérifiez par exemple que 124 en décimal correspond à 7C en hexa, et vice-versa.

Une petite table annexe permet les conversions entre « chiffres » hexa, et « nibbles » ou demi-octets de quatre bits : sachant que 7 correspond à 0111 et C à 1100, on déduit immédiatement que 7C (ou 124) vaut 01111100 en binaire !

Si vous décidez de travailler en décimal, vous devrez calculer la valeur des 2 octets représentant l'adresse. Prenons l'exemple de l'adresse 40000 : cherchez le multiple de 256 immédiatement inférieur à 40000, soit $39936 = 156 \times 256$. Soustrayez 39936 à 40000, ce qui donne 64 : codée sur deux octets, l'adresse 40000 se compose donc de 64 (octet le moins significatif, de « poids » 1) et de 156 (octet le plus significatif, de « poids » 256).

Convertis en hexadécimal, ces deux octets s'écrivent respectivement 40 et 9C, ce qui permet d'écrire l'adresse 40000 sous la forme hexadécimale 9C40.

Table de conversion décimal - hexadécimal

		2 ^e chiffre															
1 ^{er} chiffre		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
	6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
	7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
	8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
	9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
	A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
	C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
	D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
	F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

*Equivalence
entre codes
hexadécimaux
et décimaux*

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

*Correspondance
entre chiffres
hexadécimaux
et binaires*

L'adresse suivante, 40001 en décimal, s'écrirait donc 9D40 en hexadécimal : à vous de choisir ce qui vous semble le plus pratique, mais vous viendrez sûrement tôt ou tard à l'hexa !

Un autre exemple :

Le programme suivant est un autre exemple de la facilité avec laquelle un programme Basic peut faire exécuter une tâche par une routine écrite en langage machine et incorporée dans son listing.

Il s'agit d'un petit logiciel de codage et décodage de messages secrets, très simplifié mais dont le principe est intéressant à étudier car il peut être largement extrapolé.

Un texte est entré en machine sous la forme d'une chaîne m\$: chaque caractère de cette chaîne est représenté en mémoire par un octet, c'est-à-dire huit bits. L'idée consiste à modifier la position de ces bits dans l'octet grâce à une « permutation circulaire ». Une telle opération serait extrêmement lourde et lente en Basic, mais ne pose aucun problème en assembleur, des instructions machine existant tout spécialement à cet effet.

```

5 REM MESSAGES
10 MEMORY 39999
20 DATA 62,0,15,50,65,156,201
30 FOR a=40000 TO 40006
40 READ d
50 POKE a,d
60 NEXT a
100 INPUT m$
110 FOR f=1 TO LEN(m$)
120 c$=MID$(m$,f,1)
130 c=ASC(c$)
140 POKE 40001,c
150 CALL 40000
160 c=PEEK(40001)
170 PRINT c;" ";
180 NEXT f
190 PRINT
200 m$=""
210 POKE 40002,7
220 INPUT c
230 IF c=0 THEN 290
240 POKE 40001,c
250 CALL 40000
260 c=PEEK(40001)
270 m$=m$+CHR$(c)
280 GOTO 220
290 PRINT m$
300 REM (c)1988 Patrick GUEULLE

```

Les lignes 10 à 60 installent en mémoire la routine détaillée qui se limite à sept octets !

40000 LD A,0	62	0	
40002 RRC A	15		(codage)
40003 LD 40001,A	50	65	156
40006 RET	201		

variante :	40002 RLC A	7	(décodage)
------------	-------------	---	------------

Avant de la lancer par un **CALL 40000**, on **POKE** à l'adresse 40001 (c'est-à-dire à la place du zéro d'origine) l'octet qu'il s'agit de transformer.

L'instruction machine de l'adresse 40000 charge donc cet octet dans l'accumulateur du Z 80, et l'instruction suivante, **RRC A**, effectue une rotation à droite de ses bits. Vous avez compris qu'au décodage, il faudra faire l'inverse c'est-à-dire une rotation à gauche par l'instruction **RLC A** (rotate left circular A).

En 40003, on recharge le résultat contenu dans A à l'adresse 40001, puis on revient au Basic. Celui-ci va alors chercher le résultat du traitement à l'adresse 40001 par un **PEEK**.

chaque caractère de m\$ est traité de la sorte, et il s'affiche une liste de

valeurs inférieures à 256 qui forment le message codé proprement dit.

Après avoir été noté et éventuellement transmis par tout moyen approprié, le message peut être décodé grâce aux lignes 200 à 300 : en chargeant un 7 à la place du 15 à l'adresse 40002, le POKE de la ligne 210 transforme la routine de codage en une routine de décodage !

Il est donc bien sûr nécessaire que la routine de codage soit présente en mémoire, c'est-à-dire que les lignes 10 à 190 aient été exécutées au moins une fois, même « à vide ».

La liste de codes peut alors être frappée octet par octet, et sera traduite au fur et à mesure, le texte décodé étant stocké dans m\$, préalablement vidée. Il suffira de donner la valeur zéro au dernier octet de la liste pour que le texte décodé s'affiche à l'écran.

Pour améliorer la sécurité de ce codage finalement assez naïf, on pourrait songer à effectuer des rotations tantôt à droite et tantôt à gauche, dans un ordre fixé par une liste incluse dans les programmes de codage comme de décodage, et qui serait la « clef du code », à changer évidemment assez fréquemment...

V. Quelques subtilités de programmation

Nous n'avons utilisé jusqu'à présent qu'une infime partie des possibilités du Z 80. Ajoutons donc à notre panoplie quelques outils puissants qui se révéleront indispensables dans bon nombre de situations plus délicates.

Le programme suivant permet, sur un simple CALL 40000, d'obtenir l'impression sur l'écran de cinq lignes de « \$ ».

```

10 MEMORY 39999
20 DATA 62,36,6,200,197,245,205,93,187,241,193,16,247,201
30 FOR a=40000 TO 40013
40 READ d
50 POKE a,d
60 NEXT a
100 CALL 40000
500 STOP
1000 FOR f=1 TO 200
1010 PRINT"$";
1020 NEXT f
    
```

Les lignes 1000 à 1020, accessibles par RUN 1000, permettent d'effectuer le même travail en Basic à des fins de comparaison. Dans ce cas précis, le gain de rapidité est faible, mais il existe. Cela provient du fait que notre routine machine se sert du même programme extrait de la ROM que l'instruction PRINT du Basic : on ne gagne donc que le temps d'interprétation de la boucle FOR-NEXT.

On pourrait faire beaucoup plus rapide en réécrivant de toutes pièces le sous-programme d'affichage, mais là n'est pas notre propos.

Le programme ci-dessous utilise plusieurs techniques de programmation qu'il est intéressant d'étudier en détail.

DEBUT	40000	LD A,36	62	36	
\$	40002	LD B,200	6	200	
BOUCLE	40004	PUSH BC	197		
	40005	PUSH AF	245		
PRINT	40006	CALL BB5D	205	93	187
	40009	POP AF	241		
	40010	POP BC	193		
	40011	DJNZ BOUCLE	16	247	
RETOUR	40013	RET	201		

L'instruction **CALL BB5D** « appelle » un sous-programme lui-même écrit en langage machine, et implanté en mémoire à partir de l'adresse décimale 47965 ($93 + (256 \times 187)$). Remarquons qu'en hexadécimal, 93 s'écrit 5D et 187, BB...

L'adresse BB5D se situe dans la ROM Amstrad : c'est le début de la routine **TXT WR CHAR**, (Voir partie 4, chap. 2.7 page 12).

Toutes les routines de ce chapitre peuvent être utilisées à partir du langage machine grâce aux instructions de la famille **CALL**. Il faut cependant se méfier du fait que leur exécution perturbe souvent profondément le « contexte », c'est-à-dire le contenu des registres du microprocesseur.

Les instructions **PUSH** et **POP** sont précisément prévues pour remédier à ce problème : **PUSH BC** se charge de « mettre à l'abri » le contenu des registres B et C dans une zone mémoire sûre, directement gérée par le Z 80, la « pile machine ».

PUSH AF fait de même avec les registres A et F, tandis que les instructions **POP AF** et **POP BC** font le travail inverse : les contenus des registres sont reconstitués à partir des informations relues dans la pile. L'ordre de sortie de la pile doit cependant être inverse de celui d'entrée : le dernier registre mis à l'abri doit être ressorti le premier, comme un journal dans une pile, d'où cette dénomination !

L'instruction **DJNZ** (adresse 40011) est tout un programme à elle seule : c'est l'une des instructions les plus puissantes du Z 80, qui fait cruellement défaut à beaucoup d'autres microprocesseurs.

A chaque fois qu'elle s'exécute, le contenu du registre B est « décrémenté », c'est-à-dire diminué d'une unité. Tant qu'on n'arrive pas à zéro, on part exécuter une instruction située en mémoire non pas à une adresse explicitement précisée, mais à une adresse située *un certain nombre d'octets avant ou après* celle qui suit **DJNZ** : c'est ce qu'on appelle *l'adressage relatif*.

L'avantage majeur de ce système par rapport aux instructions du type **JP** est que les routines écrites de cette façon sont « relogeables » : on peut les implanter à n'importe quelle adresse mémoire sans modifications !

Ess

Essayez par exemple d'ajouter 1000 à toutes les adresses du programme page 8 (lignes 30 et 100, 10 si vous voulez) : tout fonctionnera comme avant.

Un code particulier doit cependant être utilisé pour désigner un « déplacement » qui peut être positif ou négatif, à l'aide d'un octet qui, lui, est toujours positif.

Ce code « complément à deux » est donné par la table suivante :

0	:	0	65	:	65						
1	:	1	66	:	66	128	:	-128	193	:	-63
2	:	2	67	:	67	129	:	-127	194	:	-62
3	:	3	68	:	68	130	:	-126	195	:	-61
4	:	4	69	:	69	131	:	-125	196	:	-60
5	:	5	70	:	70	132	:	-124	197	:	-59
6	:	6	71	:	71	133	:	-123	198	:	-58
7	:	7	72	:	72	134	:	-122	199	:	-57
8	:	8	73	:	73	135	:	-121	200	:	-56
9	:	9	74	:	74	136	:	-120	201	:	-55
10	:	10	75	:	75	137	:	-119	202	:	-54
11	:	11	76	:	76	138	:	-118	203	:	-53
12	:	12	77	:	77	139	:	-117	204	:	-52
13	:	13	78	:	78	140	:	-116	205	:	-51
14	:	14	79	:	79	141	:	-115	206	:	-50
15	:	15	80	:	80	142	:	-114	207	:	-49
16	:	16	81	:	81	143	:	-113	208	:	-48
17	:	17	82	:	82	144	:	-112	209	:	-47
18	:	18	83	:	83	145	:	-111	210	:	-46
19	:	19	84	:	84	146	:	-110	211	:	-45
20	:	20	85	:	85	147	:	-109	212	:	-44
21	:	21	86	:	86	148	:	-108	213	:	-43
22	:	22	87	:	87	149	:	-107	214	:	-42
23	:	23	88	:	88	150	:	-106	215	:	-41
24	:	24	89	:	89	151	:	-105	216	:	-40
25	:	25	90	:	90	152	:	-104	217	:	-39
26	:	26	91	:	91	153	:	-103	218	:	-38
27	:	27	92	:	92	154	:	-102	219	:	-37
28	:	28	93	:	93	155	:	-101	220	:	-36
29	:	29	94	:	94	156	:	-100	221	:	-35
30	:	30	95	:	95	157	:	-99	222	:	-34
31	:	31	96	:	96	158	:	-98	223	:	-33
32	:	32	97	:	97	159	:	-97	224	:	-32
33	:	33	98	:	98	160	:	-96	225	:	-31
34	:	34	99	:	99	161	:	-95	226	:	-30
35	:	35	100	:	100	162	:	-94	227	:	-29
36	:	36	101	:	101	163	:	-93	228	:	-28
37	:	37	102	:	102	164	:	-92	229	:	-27
38	:	38	103	:	103	165	:	-91	230	:	-26
39	:	39	104	:	104	166	:	-90	231	:	-25
40	:	40	105	:	105	167	:	-89	232	:	-24

41	:	41	106	:	106	168	:	-88	233	:	-23
42	:	42	107	:	107	169	:	-87	234	:	-22
43	:	43	108	:	108	170	:	-86	235	:	-21
44	:	44	109	:	109	171	:	-85	236	:	-20
45	:	45	110	:	110	172	:	-84	237	:	-19
46	:	46	111	:	111	173	:	-83	238	:	-18
47	:	47	112	:	112	174	:	-82	239	:	-17
48	:	48	113	:	113	175	:	-81	240	:	-16
49	:	49	114	:	114	176	:	-80	241	:	-15
50	:	50	115	:	115	177	:	-79	242	:	-14
51	:	51	116	:	116	178	:	-78	243	:	-13
52	:	52	117	:	117	179	:	-77	244	:	-12
53	:	53	118	:	118	180	:	-76	245	:	-11
54	:	54	119	:	119	181	:	-75	246	:	-10
55	:	55	120	:	120	182	:	-74	247	:	-9
56	:	56	121	:	121	183	:	-73	248	:	-8
57	:	57	122	:	122	184	:	-72	249	:	-7
58	:	58	123	:	123	185	:	-71	250	:	-6
59	:	59	124	:	124	186	:	-70	251	:	-5
60	:	60	125	:	125	187	:	-69	252	:	-4
61	:	61	126	:	126	188	:	-68	253	:	-3
62	:	62	127	:	127	189	:	-67	254	:	-2
63	:	63				190	:	-66	255	:	-1
64	:	64				191	:	-65			
						192	:	-64			

Codage des déplacements (offset) en « complément à deux »

Dans notre cas, c'est l'instruction de l'adresse 40013 qui doit normalement s'exécuter après **DJNZ**. Or, nous voulons dévier l'exécution vers l'adresse 40004, soit neuf adresses en arrière (-9). La table nous indique que le déplacement est représenté par l'octet de valeur décimale 247. Lors de l'écriture d'un tel programme, on ne sait pas, en général, quelle sera l'adresse de renvoi au moment où on écrit l'instruction qui le déclenchera. On utilise alors la technique des « labels » ou « étiquettes », qui permet de ne les déterminer que plus tard (comme les « assembleurs » qui travaillent en deux « passes »).

Les labels sont des noms librement choisis (mais pas trop longs tout de même) que l'on inscrit en tête des lignes de programme vers lesquelles on pense avoir à effectuer un renvoi (mais il n'y a pas d'inconvénients, au contraire, à ce que chaque ligne possède son label).

Lorsqu'une instruction de branchement (par exemple un **JP** ou un **DJNZ**) sera écrite, ce sera sous la forme **JP LABEL** ou **DJNZ LABEL**, et on remplacera l'octet (ou les deux octets) désignant l'adresse par un petit rectangle vide.

C'est une fois le programme entièrement écrit en mnémoniques que l'on procédera à la deuxième « passe », au cours de laquelle on écrira les adresses définitives et que l'on calculera les octets manquants qui seront alors placés dans ces rectangles. Cette façon de procéder, même si elle semble lourde, fait gagner du temps surtout lorsque les modifications sont faites en cours d'écriture.

C'est exactement de cette façon que travaillent les logiciels assembleurs : il est logique d'opérer de même lorsque l'on travaille « à la main ».

Vous en savez maintenant assez pour utiliser avec profit la documentation très complète sur le Z 80, mais ne perdez pas de vue que vous n'en êtes encore qu'au stade de l'initiation : commencez par écrire et faire fonctionner des routines de quelques dizaines d'octets, qui seront d'ailleurs capables de résoudre bon nombre de problèmes pratiques.

Augmentez petit à petit la taille de ces logiciels, mettez à contribution de nouvelles instructions, utilisez hardiment les différents registres qui sont à votre disposition, jonglez avec les « modes d'adressage » ou même avec les « interruptions » et vous deviendrez progressivement un bon programmeur de Z 80. Attendez-vous cependant à ce que vos progrès en langage machine soient plus lents et plus laborieux qu'en Basic, mais ils seront probablement aussi encore plus passionnants !

Et lorsque vous atteindrez un certain seuil d'importance de programmes, nul doute que vous vous offrirez un assembleur : votre aptitude à la programmation du Z 80 se trouvera alors considérablement secondée, et vous pourrez aller très loin...

4/2.6

Assembleurs existants

4/2.6.1

DEVPAC

Cet assembleur fonctionne sur CPC 464, 664 et 6128.

Deux parties le composent :

- GENA3 : Editeur de texte et Assembleur,
- MONA3 : Désassembleur et Debugger.

— GENA3

Présentation :

C'est grâce à ce programme que vous pourrez saisir le « source » de vos programmes assembleur.

C'est également avec GENA3 que vous fabriquerez le « binaire exécutable », c'est-à-dire le code hexadécimal exécutable.

- Syntaxe des codes reconnus par le compilateur de GENA3 : Le format est le suivant :

LABEL : Mnémo. Opérande Commentaire
ADR : LD A,10 ;A contient 10

Les caractères spéciaux reconnus par GENA3 sont les suivants :

; pour signaler le début d'un commentaire

* pour définir une commande :

- *E affiche 3 lignes blanches à l'écran ou un saut de page sur l'imprimante.
- *Hs affiche l'entête alphanumérique « s » après chaque saut de page.
- *S stoppe le listage sur cette ligne. Le listage redémarre par l'appui sur une touche quelconque.

*L- Le listage à l'écran et l'impression sont arrêtés à partir de cette ligne.

*L+ Le listage à l'écran et l'impression sont validés à partir de cette ligne.

*D+ Le compteur d'adresse est donné en décimal au début de chaque ligne.

*D- Le compteur est donné en hexadécimal au début de chaque ligne.

*F<Nom de fichier> permet d'assembler le fichier spécifié sur cassette ou disquette.

*T+ <Nom de fichier> permet d'obtenir la sauvegarde du code machine généré par la compilation sous le nom précisé.

*T- ferme le fichier objet courant.

• Directives d'assemblage :

Ce sont des mots-clés qui ne concernent que le compilateur et qui ne font pas partie des mnémoniques du Z80. Ces directives varient d'un assembleur à l'autre.

ORG <expression>

Donne l'adresse d'implantation du programme objet.

<Label:> EQU <expression>

Place la valeur de l'expression dans la variable précédant le mot clé EQU.

DEFB <expression> [, <expression> [..., <expression>]...]

Réserve une zone de n octets (n est le nombre d'expressions spécifiées), et y place la ou les expression(s) spécifiée(s).

DEFW <expression> [, <expression> [..., <expression>]...]

Réserve une zone de 2*n octets (n est le nombre d'expressions spécifiées) et y place la ou les expression(s) spécifiée(s).

DEFS <expression>

Définit une zone dont la longueur est donnée par l'expression spécifiée et la remplit de zéro.

DEFM <chaîne alphanumérique>

Réserve une zone de n caractères (n est la longueur de la chaîne) et y place l'expression chaîne spécifiée.

ENT <expression>

Définit l'adresse d'exécution du programme (point d'entrée).

IF <expression logique> <BLOC D'INSTRUCTIONS> ELSE <BLOC D'INSTRUCTIONS>

Assemblage conditionnel : le premier bloc d'instructions est assemblé si l'expression logique est vraie. Dans le cas contraire, le deuxième bloc d'instructions est assemblé.

Les bases de numération et les opérateurs suivants peuvent être utilisés :
aucun caractère spécial pour signaler la manipulation de nombres décimaux,

% indique que le nombre qui suit est exprimé en binaire,

indique que le nombre qui suit est exprimé en hexadécimal.

+ addition

- soustraction

& et logique (AND)

@ ou logique (OR)

! ou exclusif (XOR)

* multiplication entière

/ division entière

? fonction MODULO

• Chargement du logiciel :

Tapez RUN "GENA3" sur 664 et 6128, ou "RUN" <cr> sur 464. Le message suivant apparaît : "Load Address ?".

Entrez l'adresse où doit être stocké le premier octet de GENA3.

L'entête suivant apparaît :

AMSOFT PRESENTS

HISOFT DEVPAC

GENA3.1 Assembler Loader

Copyright Hisoft 1984

Load address ?

Load MONA now ?

Please wait... Loading GENA3.1

Donnez une adresse d'implantation et répondez N(o) à la question « Load MONA now ? ».

Lorsque GENA est en mémoire, le menu suivant apparaît :

HISOFT GENA 3.1 ASSEMBLER AMSTRAD CPC464

Copyright HISOFT 1984

All rights reserved

Assemble	Bye
Current State	Delete
Edit	Find
Get text	Help
Insert	CTRL/Jump to MONA
List	Move
reNumber	Object
Put text	Q put ASCII
Run	Separator
Tape speed	Upper line
teXt info	Width
Z print text	Y print length

ASSEMBLE : Option A, Format A

Cette option permet de compiler le programme source en mémoire. Le message suivant apparaît :

Table size : <dim table des symboles>

Entrez la dimension en octets de la table des symboles. Si vous tapez <ENTER>, GENA affectera une valeur par défaut à cette table.

Le message « Options : » apparaît vous invitant à donner une option de compilation.

Cette option peut être :

- 1 si vous désirez un listing de la table des symboles,
- 2 si le code objet ne doit pas être généré,
- 4 si vous ne désirez pas avoir de listing,
- 8 si vous voulez un listing sur imprimante,

16 si le code objet généré doit être placé juste après la table des symboles,
32 si vous ne désirez effectuer aucune vérification sur l'adresse d'implantation du code objet (le temps de compilation est plus court).

Ces options peuvent être cumulées. Par exemple, en choisissant l'option 17, vous produirez les actions 1 et 16 ($17 = 1 + 16$), etc.

Le message suivant apparaît quand vous avez donné l'option de compilation :

Hisoft GENA 3.1 Assembler. Page 1.

Pass 1 errors : 00

Pass 2 errors : 00

Table used : xx From yy

xx indique la taille de la table utilisée, et yy la taille de la table réservée par l'option « table size ».

Si une erreur se produit durant la compilation, un numéro d'erreur sera affiché et la phase d'assemblage sera stoppée :

- provisoirement jusqu'à l'appui sur une touche quelconque,
- définitivement si vous appuyez simultanément sur CTRL et C.

BYE : Option B, Format B

Redonne le contrôle au BASIC.

CURRENT STATE : Option C, Format C

Affiche la valeur courante des délimiteurs N1, N2, S1 et S2.

DELETE : Option D, Format Dn,m

Efface les lignes comprises entre n et m.

EDIT : Option E, Format En

Edite la ligne n.

FIND : Option F, Format Fn, m, f[,s]

Cherche la chaîne « f » entre les lignes n et m.

Si la chaîne « s » est présente, remplace « f » par « s ».

GET TEXT : Option G, Format G,,s

Cherche le fichier source de nom « s » sur l'unité de cassette ou de disquette.

HELP : Option H, Format H

Affiche le résumé des commandes.

INSERT : Option I, Format I

Insère du texte à la position courante du pointeur de ligne.

CTRL/J : Option CTRL+J

Format CTRL et J pressées en même temps. Exécute MONA s'il est en mémoire ; sinon, ne produit aucune action.

LIST : Option L, Format L

Liste les lignes de la position courante du pointeur de ligne à la fin du fichier.

MOVE : Option M, Format Mn,m,d

Déplace le bloc compris entre les lignes n et m en d et efface les lignes déplacées.

RENUMBER : Option N, Format Nn,m

Renumérote la première ligne en lui donnant le numéro n, et donne un espacement entre deux lignes de m.

OBJECT : Option O, Format O,,s

Sauvegarde le programme source sur cassette ou disquette en lui donnant le nom s.

PUT TEXT : Option P, Format Pn,m,s

Sauvegarde le programme source sur cassette ou disquette à partir de la ligne n, jusqu'à la ligne m sous le nom s.

PUT ASCII : Option Q, Format Q,,s

Sauvegarde le programme source en ASCII sur cassette ou disquette en lui donnant le nom s.

RUN : Option R, Format R

Exécution du programme en mémoire.

SEPARATOR : Option S, Format S,,d

Permet de choisir le caractère séparateur utilisé sur une ligne de commande. Le caractère séparateur par défaut est la virgule.

TAPE SPEED : Option T, Format Tn

Définit la vitesse de sauvegarde sur cassette.

T suivi de <ENTER> définit la vitesse lente, et T suivi d'un nombre supérieur à zéro définit la vitesse rapide.

UPPER LINE : Option U, Format U

Affiche la dernière ligne de texte entrée.

VERIFY : Option V, Format V,,s

Vérifie que le fichier texte en mémoire est identique au fichier cassette ou disquette lu de nom s.

TEXT INFO : Option X, Format X

Donne les adresses de début et de fin d'implantation du texte en mémoire.

WIDTH : Option W, Format W

Switch de passage en mode 40 ou 80 colonnes.

PRINT TEXT : Option Z, Format Z

Imprime le fichier texte actuellement en mémoire.

PRINT LENGTH : Option Y, Format Yn

Fixe le nombre de lignes par page pour les sorties sur imprimante.

II — MONA3

Présentation :

Comme GENA3, MONA3 peut être implanté à une adresse quelconque en mémoire vive.

Quand MONA3 est en mémoire, l'écran suivant apparaît :

```

0000 01 89 7F          LD BC, #7F89
>PC 0F50      22  DC  1A  2A  E1  1A  22  CF  1A
   SP BFF2      A2  B9  85  7F  8E  F2  7B  96  E8
   IY AE70      CB  00  00  7A  8A  6A  8E  FF  FF
   IX BFF8      7B  96  E8  03  98  3A  6F  DE  00
   HL 0000      01  89  7F  ED  49  C3  91  05  C3
   DE 1AB7      00  00  00  00  00  00  00  00  00
   BC 0000      01  89  7F  ED  49  C3  91  05  C3
   AF 0044      01  89  7F  ED  49  C3  91  05  C3
   IR 007D      Z   V
    
```

FFF4	00	FFFC	00	0004	49	000C	84
FFF5	00	FFFD	00	0005	C3	000D	B9
FFF6	00	FFFE	00	0006	91	000E	C5
FFF7	00	FFFF	00	0007	05	000F	C9
FFF8	00	>0000	01<	0008	C3	0010	C3
FFF9	00	0001	89	0009	8A	0011	10
FFFA	00	0002	7F	000A	B9	0012	BA
FFFB	00	0003	ED	000B	C3	0013	C3

Les commandes possibles sont les suivantes :

→ Pointeur mémoire + 1

← Pointeur mémoire - 1

↑ Pointeur mémoire - 8

↓ Pointeur mémoire + 8

! Définition d'un point d'arrêt permettant au programmeur de visualiser l'état des registres ou/et de la mémoire (par exemple).

CTRL A Désassemble une page à partir du pointeur de mémoire.

CTRL C Continue l'exécution du programme à partir de l'adresse du pointeur de programme PC.

CTRL D Choix de la base de numération (10 ou 16).

CTRL J Permet d'entrer sous GENA3 s'il est en mémoire : sinon, ne produit aucune action.

CTRL L Commande identique à la commande L, mais le bloc de mémoire est dirigé vers l'imprimante.

CTRL S Exécution d'un programme pas à pas. Cette commande exécute l'instruction courante, met à jour les données affichées sur l'écran et attend une autre commande.

CTRL X Retour sous BASIC.

G[2A]<cr> Recherche d'un ou plusieurs octets.

[FF]<cr> Terminer la définition du ou des octets recherchés par <cr>.

H[:32123] Conversion décimale/hexadécimale.

! Option conversationnelle de copie d'un bloc de mémoire d'un endroit à un autre.

J[:932A] Exécution du code machine situé à l'adresse indiquée.

L Liste sur écran un bloc de mémoire à partir de l'adresse du pointeur de mémoire.

Tapez ESC pour retourner au menu général, une autre touche pour poursuivre le listage.

M[:3212]	Affecte la valeur spécifiée au pointeur de mémoire.
N	Cherche la prochaine occurrence des octets définis dans la commande G.
O	Affiche la mémoire pointée par l'instruction de débranchement relatif située au pointeur de mémoire.
P	Option conversationnelle de remplissage d'un bloc de mémoire avec l'octet spécifié.
S	Chargement du pointeur de mémoire avec le pointeur de programme SP.
T	Option conversationnelle de désassemblage d'un bloc de mémoire.
U	Utilisée avec l'option O. Permet de retourner à l'adresse où l'option O a été tapée.
V	Utilisée avec l'option X. Permet de retourner à l'adresse où l'option X a été tapée.
W	Option conversationnelle d'écriture d'un bloc de mémoire sur cassette ou disquette.
X	Affiche la mémoire pointée par l'instruction de débranchement absolu située au pointeur de mémoire.
Y	Permet d'entrer des caractères ASCII à partir du pointeur de mémoire.

Remarque :

Les valeurs entre crochets servent d'illustration aux commandes dans lesquelles ils apparaissent. Les crochets ne doivent pas être entrés lors de l'utilisation réelle de la commande.

III — CONCLUSIONS

GENA3 :

- L'éditeur est de bonne qualité. Mais ce n'est qu'un éditeur de lignes et non un éditeur de texte pleine page, avec tout l'inconfort qu'un éditeur de lignes implique.
- Les possibilités de l'éditeur sont multiples et il est assez simple à manipuler.
- La commande H(elp) permet à tout moment d'avoir le sommaire des commandes disponibles. Ce sommaire est parfait quand on sait se servir de l'éditeur, mais le manuel utilisateur sera nécessaire pour le programmeur débutant.

— Le compilateur à deux passes offre des options intéressantes que l'on trouve rarement sur les compilateurs disponibles sur les micro-ordinateurs de la catégorie des AMSTRAD CPC. Un reproche cependant, il ne gère pas les macro-instructions.

MONA3 :

— Les possibilités de ce debugger sont multiples, et lorsqu'on s'est familiarisé avec le maniement des PC et pointeur de mémoire, son emploi est souple et agréable.

En conclusion, cet éditeur/compilateur/debugger est d'un niveau convenable et nous vous le conseillons si vous possédez un CPC 664 ou 6128. Sur 464, son emploi est laborieux à cause de la lenteur de chargement de GENA3 et MONA3.

D'autre part, son implantation en mémoire est quasiment quelconque, ce qui permet de développer des programmes, qui ne seraient pas relogeables, à leur adresse réelle en RAM.

AMSTRAD — III

GENA3

4/4.5

Utilisation du Turbo-Pascal

Afin de mieux assimiler les aspects théoriques, nous proposons dans ce chapitre des programmes types écrits en Turbo-Pascal. Certains d'entre eux nous ont été demandés par nos lecteurs, d'autres illustrent un ou plusieurs points fondamentaux et/ou complexes du Turbo-Pascal.

Lorsque cela est nécessaire, un ou plusieurs programmes écrits en Assembleur accompagnent le listing en Turbo-Pascal.

En effet, malgré les grandes possibilités de ce dernier, certaines ressources matérielles (points d'entrée du Firmware ou du CP/M, accès aux ports d'entrée/sortie, à la mémoire de l'écran), et d'une manière plus générale tous les circuits de base de la carte mère ne peuvent pas être accédés directement par les ordres du Turbo-Pascal.

Enfin, lorsque les programmes décrits sont complexes, ils sont agrémentés d'un ou de plusieurs organigrammes dans le but de bien visualiser leur structure.

4/4.5.1

Optimisation d'écriture dans un fichier texte

Il existe essentiellement deux ordres en Turbo-Pascal pour écrire dans un fichier texte. Il s'agit de « Writeln » et de « Write ». (Voir Partie 4, chap. 4.) Le premier ordre sépare les données écrites par un <CR> <LF> (passage à la ligne). Pour des raisons de rapidité de lecture de fichiers texte, il est intéressant d'utiliser la seconde forme d'écriture.

Mais alors se pose un problème lorsque l'on veut relire les données écrites. Effectivement, aucun séparateur n'existe entre deux données, et une lecture du type

Read(fichier,variable)

où variable est une chaîne, qui a toutes les chances de produire une erreur 10 (erreur dans la longueur d'une chaîne) car, une fois lue, cette dernière dépasse les 255 caractères.

Une solution pour résoudre ce problème consiste à séparer chaque donnée écrite par un signe séparateur (qui ne fait pas partie des caractères que l'on peut trouver dans le fichier). La lecture d'un tel fichier se fait caractère par caractère avant de rencontrer le séparateur. Nous avons pris le séparateur « /\ », car il n'apparaît sûrement pas dans les données que vous ou tout autre informaticien manipule tous les jours. Vous pouvez bien sûr changer ce séparateur en effectuant des modifications mineures dans les listings.

Ecriture dans un fichier texte en utilisant l'ordre Write

Nous donnons comme exemple l'écriture des 300 premiers nombres entiers. Les données sont écrites dans le fichier « g.doc ».

```

Program Ecriture_texte_avec_separateur;

(* ----- *)
(* Ce programme permet d'ecrire dans un fichier texte *)
(* des donnees par l'instruction Write. Ces donnees *)
(* sont separees par les caracteres /\ *)
(* ----- *)

Var
  Fichier      : Text;
  I            : Integer;

begin
  Assign(Fichier, 'g.doc');
  Rewrite(Fichier);
  For I:=1 to 300 do
  begin
    Write(Fichier, I);
    Write(Fichier, '/\');
  end;
  Close(Fichier);
end.

```

Le contenu du fichier g.doc doit être le suivant après exécution du programme :

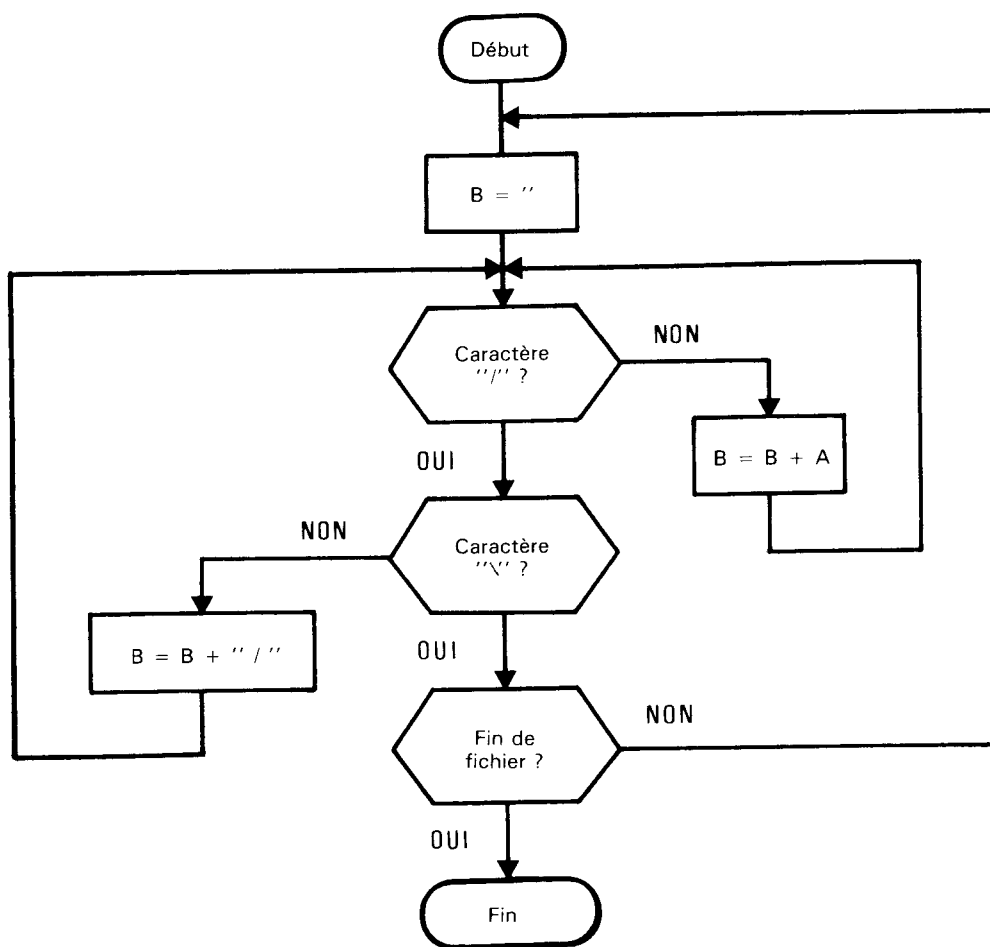
```

1/\2/\3/\4/\5/\6/\7/\8/\9/\10/\11/\12/\13/\14/\15/\16/\17/\18/\19/\20/\21/\22/\2
3/\24/\25/\26/\27/\28/\29/\30/\31/\32/\33/\34/\35/\36/\37/\38/\39/\40/\41/\42/\4
3/\44/\45/\46/\47/\48/\49/\50/\51/\52/\53/\54/\55/\56/\57/\58/\59/\60/\61/\62/\6
3/\64/\65/\66/\67/\68/\69/\70/\71/\72/\73/\74/\75/\76/\77/\78/\79/\80/\81/\82/\8
3/\84/\85/\86/\87/\88/\89/\90/\91/\92/\93/\94/\95/\96/\97/\98/\99/\100/\101/\102
/\103/\104/\105/\106/\107/\108/\109/\110/\111/\112/\113/\114/\115/\116/\117/\118
/\119/\120/\121/\122/\123/\124/\125/\126/\127/\128/\129/\130/\131/\132/\133/\134
/\135/\136/\137/\138/\139/\140/\141/\142/\143/\144/\145/\146/\147/\148/\149/\150
/\151/\152/\153/\154/\155/\156/\157/\158/\159/\160/\161/\162/\163/\164/\165/\166
/\167/\168/\169/\170/\171/\172/\173/\174/\175/\176/\177/\178/\179/\180/\181/\182
/\183/\184/\185/\186/\187/\188/\189/\190/\191/\192/\193/\194/\195/\196/\197/\198
/\199/\200/\201/\202/\203/\204/\205/\206/\207/\208/\209/\210/\211/\212/\213/\214
/\215/\216/\217/\218/\219/\220/\221/\222/\223/\224/\225/\226/\227/\228/\229/\230
/\231/\232/\233/\234/\235/\236/\237/\238/\239/\240/\241/\242/\243/\244/\245/\246
/\247/\248/\249/\250/\251/\252/\253/\254/\255/\256/\257/\258/\259/\260/\261/\262
/\263/\264/\265/\266/\267/\268/\269/\270/\271/\272/\273/\274/\275/\276/\277/\278
/\279/\280/\281/\282/\283/\284/\285/\286/\287/\288/\289/\290/\291/\292/\293/\294
/\295/\296/\297/\298/\299/\300/\
A>

```

Lecture dans un fichier texte en utilisant l'ordre Write

La recherche des deux caractères séparateurs se fait selon l'algorithme suivant. Remarquez que la rencontre du signe slash (/) ne suffit pas pour affirmer qu'un séparateur est rencontré.



L'application de cet algorithme pour retrouver les données enregistrées précédemment dans le fichier « g.doc » donne lieu au programme suivant :

```

Program Lecture_Texte_avec_Separateur;

(* ----- *)
(* Ce programme permet de lire un fichier texte *)
(* écrit par une suite d'instructions Write, et *)
(* dont les données sont séparées par les *)
(* caractères /\ *)
(* ----- *)

Var
  Result,
  I      : Integer;
  Fichier : Text;
  A      : Char;
  Nom    : String[20];
  B      : String[40];
  Bis    : Boolean;

begin
  ClrScr;
  Repeat
    Write('Nom du fichier à lire : ');
    Read(Nom);
    Assign(Fichier, Nom);
    {$I-}
    Reset(Fichier);
    {$I+}
    Result := IoResult;
    If Result <> 0 then begin
      Writeln;
      Writeln('Fichier inexistant');
    end;
  until Result = 0;

  Writeln;
  Repeat
    B := '';
    Repeat
      Read(Fichier, A);
      If A <> '/' then B := Concat(B, A);
    until A = '/'; (* Premier caractère terminateur *)
    Read(Fichier, A);
    If A = '\ ' then Bis := False
    else begin
      Bis := True;
      B := Concat(B, '/' + A);
    end;
  until Not Bis;
  Write(B, ' '); (* Affichage de la donnée lue *)
  until Eof(Fichier);
  Close(Fichier);
end.

```

4/4.5.2

Définition de routines sonores

Contrairement au Turbo-Pascal implanté sur les IBM PC, la version Amstrad ne contient pas de procédure sonore. Nous allons pallier ce problème en réalisant :

- une procédure d'émission de Beep fixe.

Cette procédure servira par exemple à indiquer que l'utilisateur a fait une action incorrecte...

- une procédure d'émission de Beep paramétrable.

Les fréquence et puissance du son émis pourront être choisies à volonté par l'utilisateur.

Emission d'un Beep de fréquence fixe

Pour réaliser un tel programme, intégrons un court programme écrit en Assembleur dans le programme Pascal en utilisant l'ordre « InLine ».

Structure du programme Assembleur

Une macro instruction du Firmware des CPC permet de programmer directement les registres du PSG (AY3-8912). Il s'agit de l'instruction **MC SOUND REGISTER** dont le point d'entrée se trouve en #BD34. (Voir Partie 4, Chap. 2.7). Pour ce qui nous concerne, il suffit de savoir que :

- le numéro du registre du PSG à accéder doit être placé dans le registre A avant l'appel ;
- la valeur à envoyer dans ce registre doit être placée dans le registre C avant l'appel.

Un appel à **MC SOUND REGISTER** se fera donc de la façon suivante :

```
LD    A,NR    ; No registre à accéder
LD    C,VAL    ; Valeur à envoyer dans ce registre
CALL  SOUND   ; Appel à MC SOUND REGISTER
```

Pour réaliser un Beep, nous utilisons les registres PSG suivants :

- registres 0 et 1 pour définir la fréquence du Beep ;
- registre 8 pour définir le volume du Beep ;
- registre 7 pour valider le canal d'émission.

Le Beep est émis pendant une durée qui correspond au temps nécessaire pour faire passer le registre HL de #FFFF à #0000 en opérant une décrémentation unitaire à chaque pas.

Le Beep est arrêté en dévalidant le canal A grâce au registre 7 du PSG.

Le programme Assembleur d'émission d'un Beep est donc le suivant :

```

1          ORG 9000H
2          LOAD 9000H
3          ; -----
4          ; Emission d'un BEEP parametuable
5          ; en utilisant la macro
6          ; MC SOUND REGISTER
7          ; -----
8          ;
9          ; Declaration de constante
10         ;
11         SOUND:      EQU 0BD34H
12         FREQH:      EQU 9F00H          ;MSB frequence
13         FREQL:      EQU 9F01H          ;LSB frequence
14         VOL:        EQU 9F02H          ;Volume du son emis
15         ;
16         ;
17 9000 3A009F        LD  A,(FREQH)      ;MSB Frequence
18 9003 4F            LD  C,A             ;dans le registre C
19 9004 3E00          LD  A,0            ;Registre 0
20 9006 CD34BD        CALL SOUND
21 9009 3A019F        LD  A,(FREQL)      ;LSB Frequence
22 900C 4F            LD  C,A             ;dans C
23 900D 3E01          LD  A,1            ;Registre 1
24 900F CD34BD        CALL SOUND
25 9012 3A029F        LD  A,(VOL)        ;Volume sonore
26 9015 4F            LD  C,A             ;Dans C

```

```
27 9016 3E08          LD   A,8           ;Registre 8
28 9018 CD34BD        CALL SOUND
29 901B 3E07          LD   A,7           ;Registre validation
30 901D 0E08          LD   C,8           ;canal A
31 901F CD34BD        CALL SOUND
32                   ;
33                   ;Boucle d'attente
34                   ;
35 9022 21FFFF        LD   HL,OFFFHH
36                   BOU:   EQU  $
37 9025 2B            DEC   HL
38 9026 7C            LD   A,H
39 9027 B5            OR    L
40 9028 20FB          JR    NZ,BOU
41                   ;
42                   ;Arret du son emis
43                   ;
44 902A 3E08          LD   A,8           ;Registre amplitude
45 902C 0E00          LD   C,0           ;mis a zero
46 902E CD34BD        CALL SOUND
47 9031 C9            RET
48                   END
```

Le programme Pascal d'émission du Beep consiste à incorporer les données hexadécimales correspondant au programme Assembleur précédent dans une procédure. Remarquez que le code #C9 correspondant au RET de fin de routine n'a pas été reporté dans le programme Pascal. En effet, le « end; » de la procédure a le même effet.

Program Definition_Beep;

```
(* ----- *)
(* Ce programme propose une routine d'émission de Beep *)
(* de durée et fréquence fixes sous la forme d'une *)
(* procédure de nom Beep *)
(* ----- *)
```

Procedure Beep;

```
(* ----- *)
(* PSG: Reg 0 = 0, Reg 1 = 1, Reg 2 = 6, Reg 7 = 8 *)
(* ----- *)
```

begin

```
  inline($3e/$00/      ( LD      A,0      )
    $0e/$00/          ( LD      C,0      )
    $cd/$34/$bd/      ( CALL    $BD34    )
    $3e/$01/          ( LD      A,1      )
    $0e/$01/          ( LD      C,1      )
    $cd/$34/$bd/      ( CALL    $BD34    )
    $3e/$08/          ( LD      A,8      )
    $0e/$08/          ( LD      C,6      )
    $cd/$34/$bd/      ( CALL    $BD34    )
    $3e/$07/          ( LD      A,7      )
    $0e/$08/          ( LD      C,8      )
    $cd/$34/$bd/      ( CALL    $BD34    )
    $21/$ff/$ff/      ( LD      HL,$FFFF )
    $2b/              ( DEC     HL      )
    $7c/              ( LD      A,H      )
    $b5/              ( OR      L        )
    $20/$fb/          ( JR      NZ,BOU   )
    $3e/$08/          ( LD      A,8      )
    $0e/$00/          ( LD      C,0      )
    $cd/$34/$bd/      ( CALL    SOUND   )
```

end;

begin

```
  Beep:              ( Activation du Beep )
```

end.

As

Emission d'un Beep de fréquence et intensité variables

Le principe est le même, à ceci près que la fréquence doit être placée dans les variables **FREQH** et **FREQL** qui correspondent respectivement au poids fort et au poids faible de la fréquence à émettre. Il convient également de placer la valeur du volume dans la variable **Vol** juste avant l'appel. Un exemple est donné dans le programme suivant :

```
A>
Program Definition Beep_parametrable;

(* ----- *)
(* Ce programme propose une routine d'émission de Beep *)
(* de durée et fréquence variables sous la forme d'une *)
(* procédure de nom Beep *)
(* ----- *)

var
  FreqH   : byte absolute $9f00; { Poids fort   fréquence }
  FreqL   : byte absolute $9f01; { Poids faible fréquence }
  Vol     : byte absolute $9f02; { Volume }

Procedure Beep;
(* ----- *)
(* FSG: Reg 0, 1 et 8 variables, Reg 7 = 8 *)
(* ----- *)

begin
  inline($3a/$01/$9f/      { LD      A,$9F01  }
    $4f/                   { LD      C,A      }
    $3e/$00/               { LD      A,0      }
    $cd/$34/$bd/           { CALL    $BD34    }
    $3a/$00/$9f/           { LD      A,$9F00  }
    $4f/                   { LD      C,A      }
    $3e/$01/               { LD      A,1      }
    $cd/$34/$bd/           { CALL    $BD34    }
    $3a/$02/$9f/           { LD      A,$9F02  }
    $4f/                   { LD      C,A      }
    $3e/$08/               { LD      A,8      }
    $cd/$34/$bd/           { CALL    $BD34    }
    $3e/$07/               { LD      A,7      }
    $0e/$08/               { LD      C,8      }
    $cd/$34/$bd/           { CALL    $BD34    }
    $01/$ff/$ff/           { LD      HL,$FFFF }
    $0b/                   { DEC     HL      }
    $01/                   { LD      A,H      }
    $0f/                   { OR      L       })
```

```
      $20/$fb/          ( JR      NZ,BOU    )
      $3e/$08/          ( LD      A,8       )
      $0e/$00/          ( LD      C,0       )
      $cd/$34/$bd);     ( CALL    SOUND    )
end;

begin
  FreqH:=1;             ( Poids fort   Frequence )
  FreqL:=100;           ( Poids faible Frequence )
  Vol:=8;               ( Volume du son emis )
  Beep;                 ( Activation du Beep    )
end.

A>
```

4/4.5.3

Position du curseur sur l'écran

Deux procédures bien utiles (WhereX et WhereY) sont implantées sur IBM PC et non sur CPC. Là encore, nous allons pallier ce manque en écrivant une routine assembleur unique que nous appellerons **WhereXY** et dont le but sera de donner la position courante du curseur dans les variables PX (abscisse) et PY (ordonnée).

Une macro instruction du firmware donne la position courante du curseur. Il s'agit de **TXT GET CURSOR** dont le point d'entrée se trouve en #BB78. Elle ne demande aucun paramètre en entrée et fournit l'abscisse du curseur dans le registre H et l'ordonnée du curseur dans le registre L.

Le programme de lecture de la position du curseur en assembleur consistera à activer la routine **TXT GET CURSOR** et à placer la valeur des registres H et L dans des cases mémoires accessibles en Turbo-Pascal.

```

1          ORG 9000H
2          LOAD 9000H
3          ; -----
4          ; Lecture de la position curseur
5          ; sur l'ecran. L'abscisse est mise
6          ; dans la memoire &4000 et
7          ; l'ordonnee dans la memoire &4001
8          ; -----
9          ; Programme totalement relogeable
10         ; -----
11         ; -----
12         ; Definition de constantes
13         ; -----
14         GETCUR:      EQU 0BB78H          ;TXT GET CURSOR
15         X:           EQU 4000H          ;Abcisse curseur
16         Y:           EQU 4001H          ;Ordonnee curseur
17         ; -----
18         ; Programme
19         ; -----
20 9000 CD78BB          CALL GETCUR          ;Lecture pos curseur
21 9003 7C              LD A,H
22 9004 320040          LD (X),A            ;Abcisse
23 9007 7D              LD A,L
24 9008 320140          LD (Y),A            ;Ordonnee
25 900B C9              RET
26                     END

```

Le programme Turbo-Pascal qui intègre cette routine est le suivant :

```
Program Position_Curseur;
```

```
(* ----- *)
(* Ce programme donne une extension du Turbo Pascal *)
(* v3.0 CPC: La procedure WhereXY *)
(* donne la position en X et en Y du curseur. *)
(* ----- *)
```

```
var
```

```
  PX      : Byte Absolute $9d00; ( Abcisse  Curseur )
  PY      : Byte Absolute $9d01; ( Ordonnee Curseur )
```

```
Procedure WhereXY;
```

```
(* ----- *)
(* En sortie de la procedure: *)
(* L'abscisse du curseur se trouve dans la variable PX *)
(* L'ordonnee du curseur se trouve dans la variable PY *)
(* ----- *)
```

```
begin
```

```
  Inline($cd/$78/$bb/      { CALL GETCUR }
    $7c/                    { LD   A,H   }
    $32/$00/$9d/           { LD   (X),A }
    $7d/                    { LD   A,L   }
    $32/$01/$9d);          { LD   (Y),A }
```

```
end;
```

```
begin
```

```
  WhereXY;
  Writeln('Au moment de l'appel, le curseur se trouvait en :');
  Writeln('X=',PX,' Y=',PY);
end.
```

4/5

Le langage Forth 83-Standard¹⁾ pour Amstrad 464, 664, 6128 et PCW

I. Genèse du langage Forth

La formidable évolution de la micro-informatique tend à nous faire oublier qu'il n'y a pas si longtemps, c'est-à-dire une dizaine d'années, pouvoir disposer de 16K de mémoire et d'un lecteur de disquette sur un système de développement constituait un véritable luxe. A cette époque, les langages informatiques les plus « performants », tels que Pascal ou Fortran, ne pouvaient pratiquement être exploités que sur de gros systèmes.

Bien des tentatives ont été faites pour créer des langages à la fois compacts et très performants. On ne parle bien entendu que de langages compilés. Un de ces langages a eu la faveur de la communauté astronomique internationale : le langage Forth.

Forth est un langage surprenant :

- il génère un code pseudo-compilé plus compact qu'un assembleur classique ;
- il est à la fois interpréteur et compilateur ;
- il peut être résident en mémoire ROM ou chargé en mémoire vive ;
- il est très structuré ;
- son jeu « d'instructions » est extensible à l'infini (en fait limité à la capacité mémoire disponible) ;

1) Comment se procurer Forth 83-Standard ?

L'Association JEDI propose la version F83 Laxen et Perry sur disquette 3" pour AMSTRAD CPC 464, 664 et 6128, PCW 8256, 8512 contre l'envoi de 10 timbres à 3,70 Fr.
Contact : ASSOCIATION JEDI - 17, rue de la Lancette - 75012 Paris.

- la rapidité d'exécution d'un programme compilé est à peine moins rapide qu'un programme équivalent écrit en langage machine ;
- il autorise la cohabitation de routines assemblées et de routines compilées dans son dictionnaire ;
- Forth est écrit à 80 % en Forth.

II. Domaines d'applications

Forth a commencé sa carrière dans le domaine de l'astronomie et plus particulièrement celui du traitement de données en temps réel en radio-astronomie à l'observatoire de Kit Peak en Arizona (USA).

Ne quittons pas les étoiles. La première application spectaculaire de Forth a été d'assurer le pilotage de caméras pour le film « La Guerre des Etoiles » de Georges Lucas. Jusqu'alors, la complexité des mouvements de plusieurs maquettes dans une séquence d'animation ne permettait que la prise de vues au plan fixe. Avec une caméra asservie à des servomoteurs, les mouvements de prises de vues sont calculés pour donner l'illusion du mouvement des maquettes et du décor comme si l'on se trouvait aux commandes d'un appareil inter-sidéral.

Aujourd'hui les domaines d'applications de Forth se sont considérablement élargis.

A Auxerre, les pompiers agissent selon un plan d'intervention calculé par un programme écrit en Forth.

Plusieurs sociétés de conception en automatisme industriel exploitent Forth intensivement.

Des jeux vidéos renommés sont écrits en Forth. Et même le dernier né des logiciels de gestion de base de données Rapidfile, diffusé par Ashton Tate, est écrit en Forth. VP-Planer, un clone de Lotus 123, est également écrit en Forth et a été agréé par des sociétés renommées (EDF par exemple). Un générateur de système expert, Forthlog II, est exploité par le département formation d'un ministère.

La prochaine génération de micro-ordinateurs sera peut-être équipée d'un micro-processeur de la série Novix en remplacement des micro-processeurs conventionnels. L'énorme intérêt du Novix 4016 est de permettre la programmation d'applications directement en langage Forth sans passer par un code machine et des séquences d'instructions issues d'un assemblage. L'architecture Risc ayant servi à la fabrication du Novix 4016 permet des vitesses d'exécution jamais atteintes sur un micro-ordinateur ordinaire (une boucle à vide est exécutée un million de fois en 0,16 seconde).

4/5.1

Le langage Forth sur les Amstrad

I. Les versions FIG

Plusieurs versions ont été développées, d'abord en cassette, puis en disquette. La première version d'origine britannique, Kuma-Forth, ne travaillait qu'en cassette et était lancée depuis Basic.

Cette version a rapidement été supplantée par une version d'origine française tournant sous CP/M, Amsforth, mais disponible exclusivement pour les systèmes Amstrad CPC 464, 664 et 6128.

II. Les versions 83-Standard

Deux versions similaires sont apparues en Europe :

- la première, nommée Volksforth, d'origine germanique, distribuée par le Forth Interest Group de Hambourg (RFA).
- la seconde, nommée F83 et diffusée par l'Association Jedi. Dans la suite de ce chapitre nous nous consacrerons exclusivement à cette version.

Le système Forth F83 a été écrit par deux Américains, Henri Laxen et Michael Perry, et placé dans le domaine public. Il tourne sous CP/M (versions 2.x, 3.x) et a été adapté de manière à fonctionner sur toute la gamme Amstrad sans restriction (CPC et PCW). Une version F83 est également disponible pour les nouveaux Amstrad compatibles IBM.

Sur tous ces systèmes, le langage Forth F83 utilise la même syntaxe, ce qui garantit une très grande portabilité des programmes.

Sur Amstrad, le langage Forth F83-Standard (que l'on abrègera par la suite par F83), dispose des outils suivants :

- décompilateur de définition Forth ;
- assembleur 8080 au format Forth ;
- éditeur de blocs de programmes source ;
- méta-générateur de programme ;
- tous les fichiers du code source ayant servi à la génération du langage Forth. En clair, F83 est écrit et généré par F83.

III. Premier contact

Vous venez de recevoir votre disquette contenant le langage F83. La première précaution est de copier son contenu sur une autre disquette, car un accident est vite arrivé.

Une fois votre copie faite, passez sous CP/M:

- depuis Basic en utilisant la commande CPM précédée de la barre verticale pour les systèmes Amstrad CPC 464, 664 et 6128 ;
- en chargeant CP/M à la mise sous tension pour les systèmes Amstrad PCW.

Retirez la disquette CP/M et introduisez la disquette contenant F83. Tapez **F83** et appui sur la touche **RETURN**. F83 se présente :

8080 Forth 83 Model 1.0.0

Modified 16oct83

Un appui sur la touche de retour chariot provoque l'affichage du message :

ok

Ce message indique que Forth est disponible et attend une commande. Il apparaîtra chaque fois que Forth aura terminé l'exécution d'une commande ou d'un programme.

La première commande que vous allez essayer est **WORDS**. Ce mot provoque l'affichage de tous les mots Forth disponibles dans le vocabulaire courant. Tapez **WORDS** et appuyez sur **RETURN**, vous voyez à l'affichage :

EMPTY MARK HELLO EXTEND80.BLK META80.BLK BACKGROUND

ACTIVATE SET-TASK ...etc...

...jusqu'à... UNNEST EXIT RP FORTH ok

L'appui sur une touche quelconque interrompt l'exécution de **WORDS**. Vous avez essayé ? Ah, il y a un problème, vous avez tapé en minuscule et Forth a affiché :

words words ?

C'est parce que Forth ne reconnaît que les commandes tapées en majuscules. Pour arranger ceci, tapez en caractères majuscules :

CAPS ON et appui sur **RETURN** (on ne le redira plus !)

A partir de maintenant, Forth ne se fâchera plus si vous tapez en minuscule ou en majuscule. Essayez à nouveau **WORDS**, **words** ou **WoRdS**.

A. PREMIÈRES OPÉRATIONS ARITHMÉTIQUES

Le langage FORTH stocke les nombres sur une pile nommée « pile de données ». L'empilage d'un nombre est très simple :

55

empile le nombre 55. Pour dépiler ce nombre, il y a plusieurs méthodes. La première consiste à afficher le contenu du dernier élément empilé.

```
. affiche 55
```

Si vous empilez plusieurs nombres, voici ce qui se passe :

11 45 6543

Empilage du
premier nombre

11

Empilage du
second nombre

45

11

le nombre 11 passe en dessous

Empilage du
troisième nombre

6543

45

11

le nombre 45 passe en dessous

le nombre 11 passe en 3^e position

Le dépilage successif des nombres affiche ceux-ci dans l'ordre inverse de leur empilage :

```
. affiche 6543  
. affiche 54  
. affiche 11
```

Pour mieux visualiser le mécanisme d'empilage et de dépilage des nombres, pensez à une pile d'assiette : la dernière assiette déposée sur la pile sera la première reprise. A tout moment vous pouvez prendre connaissance du contenu de la pile sans avoir à provoquer le dépilage des valeurs qui y sont stockées en utilisant le mot .S

1 2 3 .S

affiche

1

2

3

Les opérateurs arithmétiques agissent en majorité sur les deux valeurs situées au sommet de la pile. Pour additionner deux nombres, il faut d'abord les déposer sur la pile de données :

22 44 + . affiche 66

Les valeurs 22 et 44 ne sont pas conservées. La somme de trois nombres ou plus obéit aux mêmes règles :

55 1 + 3 + . affiche 59 et peut aussi s'écrire
55 1 3 + + .

Ce principe de calcul est appelé Notation Polonaise Inverse (RPN dans la littérature anglaise, pour *Reverse Polish Notation*).

Soit deux nombres a et b, voici les syntaxes Forth des quatre opérations de base :

a b + pour a + b
a b - pour a - b
a b * pour a * b
a b / pour a / b

Pour la division, seul le quotient entier est conservé sur la pile de données :

22 7 / . affiche 3

Le reste de la division peut être obtenu en appliquant la fonction modulo :

22 7 MOD . affiche 1

Une fonction /MOD combine les actions de / et de MOD:

22 7 /MOD . . affiche 3 1

Les opérations peuvent être combinées mais une opération en notation arithmétique comportant des parenthèses doit être convertie en notation RPN en tenant compte de l'ordre de priorité des opérations. Forth n'utilise pas les parenthèses dans les opérations arithmétiques :

soit l'expression algébrique	(2 + 5) * (7 - 2)
elle s'écrit en Forth	2 5 + 7 2 - *

B. LA MANIPULATION DES DONNÉES SUR LA PILE

La pile est l'élément fondamental du langage Forth pour le traitement de données. Son fonctionnement est identique à celle gérée par le micro processeur. Dans certaines situations, les données traitées par les différentes définitions doivent être réordonnées ou dupliquées. Ces opérations sont exécutées par les mots suivants :

DUP	(n --- n n)	Duplique le contenu du sommet de la pile
OVER	(n1 n2 --- n1 n2 n1)	Duplique le second élément de la pile
SWAP	(n1 n2 --- n2 n1)	Inverse les deux éléments du sommet de la pile
ROT	(n1 n2 n3 --- n2 n3 n1)	Effectue une rotation sur trois éléments
ROT	(n1 n2 n3 --- n3 n1 n2)	Effectue une rotation inverse sur trois éléments. Similaire à ROT ROT
PICK	(n --- n n)	Dépose au sommet de la pile le n ^e élément de la pile, n non compris
ROLL	(n ---)	Effectue une rotation sur les n premiers éléments de la pile de données, n non compris.

Nous ne nous étendrons guère sur ces opérateurs et vous laissons le soin de les tester pour mieux vous familiariser avec eux. Nous aurons l'occasion de les utiliser fréquemment dans les différents exemples qui suivront.

4/5.2

Le compilateur Forth

Le langage Forth est d'un fonctionnement tout à fait paradoxal car il intègre à la fois un interpréteur, un compilateur et un assembleur. La version F83 dispose même de trois interpréteurs :

- l'interpréteur externe (voir Partie 4, chap. 5.1) ;
- l'interpréteur d'exécution pas à pas qui opère sur une définition et permet la mise au point des routines délicates ;
- l'interpréteur d'adresse interne.

Les deux premiers sont écrits en Forth, le dernier est écrit en langage machine.

Le compilateur Forth travaille en une seule passe et sans gérer de table de référence. Ce compilateur est plus exactement un pseudo-compilateur dont la seule fonction est d'accroître les fonctions disponibles dans le dictionnaire.

L'assembleur Forth travaille également en une seule passe et sans gérer de table de référence. Il peut traiter les branchements avant en utilisant des structures de contrôle évoluées à la place d'étiquettes. Une séquence peut être assemblée entre deux définitions compilées en Forth.

Au démarrage de Forth, tout part de l'interpréteur externe (celui qui vous dit OK) et dont le fonctionnement est très simple :

```
si un mot est tapé au clavier
  et si ce mot existe
    et si on est en interprétation
      alors on l'exécute
    sinon si on est en compilation
      alors on le compile
  sinon on essaie de le convertir en nombre 16 ou 32 bits
    et si c'est un nombre
      et si on est en interprétation
        alors on empile ce nombre 16 ou 32 bits
      sinon si on est en compilation
        alors on compile ce nombre 16 ou 32 bits
  sinon si ce n'est ni un mot ni un nombre,
    alors on envoie un message d'erreur.
```

I. Compilation de votre première définition

A titre d'exemple, nous allons compiler une définition que l'on appellera **AFFICHE-BINAIRE** et dont le rôle sera d'afficher en binaire un nombre préalablement déposé au sommet de la pile de données :

```
: AFFICHE-BINAIRE  
  2 BASE ! . DECIMAL ;
```

Le mot **:** (deux-points) ouvre une séquence de compilation et est suivi du libellé du mot à définir. Le nom peut être composé de tous les caractères ASCII dont le code est compris entre 33 et 126 à l'exclusion de tout espace ou caractère de contrôle. La longueur d'un mot est comprise entre 1 et 32 caractères.

Le nom est suivi de la définition :

```
2 BASE !    passage en base numérique binaire  
.          affichage du nombre précédemment empilé  
DECIMAL    restitution de la base numérique décimale
```

La fin de la séquence de compilation est marquée par le mot **;** (point-virgule). Vous êtes à nouveau en mode interprétation.

Maintenant votre nouvelle définition fait partie du dictionnaire Forth, comme vous pouvez le voir en tapant **WORDS**. Pour essayer le fonctionnement du mot que vous venez de créer, il suffit de taper :

```
14 AFFICHE-BINAIRE affiche 1110  
756 AFFICHE-BINAIRE affiche 1011110100
```

Si vous trouvez qu'**AFFICHE-BINAIRE** est trop long, réécrivez votre définition en l'appelant **B.** ou mieux, en définissant **B.** comme suit :

```
: B. AFFICHE-BINAIRE ;
```

Tout mot compilé dans le dictionnaire Forth peut être intégré à une nouvelle définition.

II. Les constantes et les variables simple précision

Tout programme écrit en Forth, aussi sophistiqué soit-il, ne peut pas toujours traiter des données provenant de la pile de données. Dans certains cas on fera appel à des constantes et des variables. Les constantes sont définies à l'aide du mot **CONSTANT**:

```
1988 CONSTANT ANNEE
```

Les variables sont définies à l'aide du mot **VARIABLE**:

```
VARIABLE JOURS
```

La constante ANNEE et la variable JOURS figurent maintenant dans le dictionnaire Forth, c'est-à-dire qu'elles sont disponibles au même titre qu'un mot compilé par : (deux-points) ou n'importe quelle autre primitive déjà définie dans Forth. Seule l'exécution de ces mots diffère de celle d'un mot défini par : (deux-points).

Une constante dépose au sommet de la pile de données la valeur affectée au moment de sa définition :

ANNEE . affiche 1988

Une variable dépose au sommet de la pile de données l'adresse contenant la valeur qui lui a été affectée ou qui devra y être affectée.

JOURS . affiche une adresse 16 bits

Le contenu de cette adresse est déposé sur la pile par l'exécution du mot @ :

JOURS @ . affiche 0

Une valeur numérique 16 bits peut être stockée dans une variable par l'exécution du mot ! :

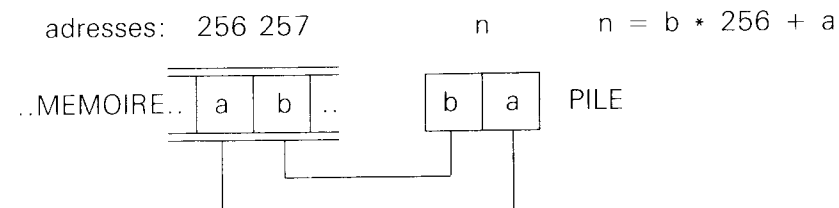
15 JOURS ! puis

JOURS @ . affiche 15

En fait, les mots @ et ! agissent sur n'importe quelle adresse mémoire. Si vous tapez :

256 @ .

Forth affichera le contenu 16 bits des adresses 256 et 257 et dont voici le mécanisme d'exécution :



L'inversion des valeurs a et b est valable pour les versions F83 tournant sur des systèmes équipés de microprocesseurs Z80, 8080 ou 8086. Sur les autres systèmes, le mécanisme diffère mais le résultat figurant au sommet de la pile de données sera identique.

Forth dispose d'un certain nombre de variables pré-définies et utilisées par le système. L'une d'elle, la variable **BASE** que nous avons déjà utilisée, peut être modifiée au gré de l'utilisateur. Voyons, avec un nouvel exemple, comment afficher un nombre dans une base numérique autre

que la base numérique courante, ceci quelle que soit la base de départ et sans altération de celle-ci :

VARIABLE NOUVELLE-BASE 2 NOUVELLE-BASE !

: NB. (n ---)

BASE @ NOUVELLE-BASE @ BASE !

SWAP (on récupère la valeur n)

. BASE ! ;

Exemple d'utilisation :

DECIMAL 355 NB. affiche 101100011

HEX 2DF NB. affiche 1011011111

Dans le second cas, on sélectionne la base hexadécimale, on empile un nombre hexadécimal et on exécute NB. On modifie la base numérique dans laquelle le nombre sera affiché en affectant une nouvelle valeur à la variable **NOUVELLE-BASE**.

Remarquez qu'un nombre n'a pas de syntaxe particulière désignant la base numérique à laquelle il appartient. C'est la base numérique que l'on sélectionne avant empilage du ou des nombres. Cette simplicité fait partie de la philosophie du langage Forth.

III. Les constantes et les variables double précision

Si l'intervalle de définition d'une constante ou d'une variable définie par **CONSTANT** ou **VARIABLE** est trop restreint pour une application particulière, on peut faire appel aux constantes et aux variables double précision.

Une constante double précision est définie par le mot **2CONSTANT**:

150000. 2CONSTANT JACKPOT

Et une variable double précision est définie par **2VARIABLE**:

2VARIABLE SCORE

Une valeur double précision est affectée à une variable double précision par le mot **2!** :

120000. SCORE 2!

Et le contenu d'une variable double précision est empilé par le mot **2@**:

SCORE 2@ D. affiche 120000

4/5.3

Contrôle de l'affichage

I. Format des nombres traités par Forth

Forth ne traite pas n'importe quels nombres. Ceux que vous avez utilisés en essayant les précédents exemples sont des entiers signés 16 bits. Le domaine de définition des entiers 16 bits est compris entre - 32768 et 32767.

Exemple :

32767 . affiche 32767

32768 . affiche - 32768

65535 . affiche - 1

Ces nombres peuvent être traités dans n'importe quelle base numérique, toutes les bases numériques situées entre 2 et 32 étant valides :

255 HEX . DECIMAL affiche FF

La base numérique courante est contrôlée par une variable nommée **BASE** et dont le contenu peut être modifié. Ainsi, pour passer en binaire, il suffit de stocker la valeur **2** dans **BASE** :

2 BASE !

et de taper **DECIMAL** pour revenir à la base numérique décimale.

Forth ne connaît pas les nombres en virgule flottante. Cependant, des routines peuvent être compilées pour exécuter des calculs sur ce type de nombres, mais ce sera au détriment de la vitesse de traitement. Si vous voulez traiter des quantités importantes, comme votre compte en banque ou le prix de votre prochaine voiture, vous pouvez utiliser les nombres double précision également nommés nombres 32 bits. Ceux-ci se distinguent des nombres entiers 16 bits en y accolant un point, une virgule ou une barre de fraction :

135246. D. affiche 135246

135246, D. affiche 135246

Le point, la virgule ou la barre de fraction peuvent être placés ailleurs qu'à l'extrémité du nombre :

135,246 D. affiche 135246

Les opérations possibles sur les nombres 32 bits sont :

3. 5. D+ correspond à 3 + 5
3. 5. D- correspond à 3 - 5

II. Formats d'affichage des nombres entiers et double précision

A première vue, F83 semble bien rustique concernant les possibilités d'affichage des nombres. Il n'en est rien. Si d'autres langages, comme Basic, utilisent des compléments d'instructions tel Using pour forcer un format d'affichage de nombre, F83 utilise des mots variés dont les combinaisons permettent de dépasser largement les performances des langages conventionnels dans ce domaine.

Les nombres entiers 16 bits situés au sommet de la pile peuvent être affichés de diverses manières par les mots suivants :

- . Prononcer 'point'. Affiche un entier signé (intervalle -32768,32767)
- U. Prononcer 'U-point'. Affiche un entier non signé (intervalle 0,65535)
- .R Prononcer 'point-R'. Affiche un entier signé cadré à droite dans un champ de n caractères.
- U.R Prononcer 'U-point-R'. Affiche un entier non signé cadré à droite dans un champ de n caractères.

Exemples :

7 - 10 . .	affiche 7 - 10
45656 U.	affiche 45656
- 3 6 .R	affiche - 6
154 6 .R	affiche 154
45656 6 U.R	affiche 45656

Les nombres double précision 32 bits situés au sommet de la pile peuvent être affichés par les mots suivants :

- D. Prononcer 'D-point'. Affiche un nombre double précision signé.
- UD. Prononcer 'U-D-point'. Affiche un nombre double précision non signé.
- D.R Prononcer 'D-point-R'. Affiche un nombre double précision signé cadré à droite dans un champ de n caractères.
- UD.R Prononcer 'UD-point-R'. Affiche un nombre double précision non signé cadré à droite dans un champ de n caractères.

Exemples :

5. D.	affiche 5	
- 1 UD.	affiche 4294967295	
10. 12 D.R	affiche	10
35.3 12 UD.R	affiche	353

III. Définition de nouveaux formats d'affichage

Outre ces mots pré-définis, F83 dispose de primitives permettant d'adapter l'affichage d'un nombre à un format quelconque. Ces primitives ne traitent que des nombres double précision :

<# Début une séquence de définition de format
Insère un digit dans une séquence de définition de format
#S Equivaut à une succession de #
HOLD Insère un caractère dans une définition de format
#> Achève une définition de format et laisse sur la pile l'adresse et la longueur de la chaîne contenant le nombre à afficher.

Exemple, soit à afficher un nombre exprimant un montant libellé en francs avec la virgule comme séparateur décimal :

: FRANCS (ud ---)

<# # # ASCII , HOLD #S #> TYPE ." Fr " ;

Exemples d'exécution :

0.35 FRANCS	affiche 0,35 Fr
35.75 FRANCS	affiche 35,75 Fr
10.15 35.75 D+ FRANCS	affiche 45,90 Fr

La définition de FRANCS ne prend pas en compte le signe du résultat, ni la position de la virgule. Si vous tapez

35. FRANCS

Vous aurez à l'affichage **0,35 Fr**.

Voici un exemple plus complexe démontrant la compacité du Forth. Il s'agit d'écrire un programme convertissant un nombre quelconque de secondes au format **HH:MM:SS**:

: :00 (---)

DECIMAL # 6 BASE ! # ASCII : HOLD DECIMAL ;

: HMS (ud ---)

<# :00 :00 #S #> TYPE SPACE ;

Exemples d'exécution :

59. HMS affiche 0:00:59

60. HMS affiche 0:01:00

4500. HMS affiche 1:15:00

Le système d'affichage des secondes et des minutes est appelé système sexagésimal. Les 'unités' sont exprimées dans la base numérique décimale, les 'dizaines' sont exprimées dans la base six. Le mot :00 gère la conversion des 'unités' et des 'dizaines' dans ces deux bases pour la mise au format des digits correspondant aux secondes et aux minutes. Pour les heures, les digits sont tous décimaux.

Exercice 1

Définissez un programme convertissant un nombre entier 16 bits décimal en binaire et l'affichant au format bbbb bbbb bbbb bbbb.

Solution page 6.

IV. Affichage des caractères et chaînes de caractères

L'affichage d'un caractère est réalisé par le mot **EMIT**:

65 EMIT affiche A

Les caractères affichables sont compris dans l'intervalle 32..255. Les codes compris entre 0 et 31 seront interprétés comme des codes de contrôle :

13 EMIT 10 EMIT exécute l'équivalent de CR (retour chariot)

Les chaînes de caractères sont affichées de diverses manières. La première, utilisable en compilation seulement, affiche une chaîne de caractères délimitée par le caractère " (guillemet) :

: TITRE ." MENU GENERAL" ;

TITRE affiche MENU GENERAL

La seconde, utilisable en interprétation seulement, affiche une chaîne de caractères délimitée par le caractère) (parenthèse fermée) :

.(MENU GENERAL) affiche MENU GENERAL

Dans les deux cas, la chaîne est séparée du mot ." ou .(par au moins un caractère espace.

Une chaîne de caractères peut aussi être compilée par le mot " (guillemet) et délimitée par le caractère " (guillemet) :

: LIGNE1 " E..Enregistrement de données" ;

L'exécution de LIGNE1 dépose sur la pile de données l'adresse et la longueur de la chaîne compilée dans la définition. L'affichage est réalisé par le mot **TYPE**:

LIGNE1 TYPE affiche E..Enregistrement de données

En fin d'affichage d'une chaîne de caractères, le retour à la ligne doit être provoqué s'il est souhaité :

TITRE LIGNE1 affiche MENU GENERAL..Enregistrement de données

CR TITRE CR CR LIGNE1 CR affiche

MENU GENERAL

E.. Enregistrement de données

Un ou des espaces peuvent être rajoutés en début ou fin d'affichage d'une chaîne alphanumérique :

SPACE affiche un caractère espace

n SPACES affiche n caractères espaces

Exercice 2

Créez un mot qui affiche une chaîne de caractères centrée par rapport à la largeur de l'écran.

Solution page 6.

V. Solutions des exercices

Exercice 1

```
: BBBB ( ----)
  # # # # 32 HOLD ;
: UB. ( u ---)
  0 < # BBBB BBBB BBBB BBBB # > TYPE SPACE ;
```

- Utilisation :
33 UB. affiche 0000 0000 0010 0001
256 UB. affiche 0000 0001 0000 0000

Exercice 2

```
80 CONSTANT LARGEUR-ECRAN
: .CENTRE ( adr long ---)
  LARGEUR-ECRAN OVER - 2/ SPACES TYPE ;
```

- Utilisation :
: AFFICHE-CENTRE
CR "Ce texte".CENTRE
CR "est centré sur 80 caractères" CENTRE CR ;
AFFICHE-CENTRE affiche
Ce texte est centré sur 80 caractères.

4/4.5.4

Programmation d'un traitement de texte

Afin de mieux comprendre l'utilisation des commandes du Turbo-Pascal, nous vous présentons un programme qui réalise la fonction de traitement de texte sur une ou plusieurs pages d'écran. Il est écrit totalement en Turbo-Pascal et présente l'avantage d'utiliser de nombreuses commandes.

La mémoire nécessaire pour entrer la source de ce programme, pour le compiler et l'exécuter n'est disponible que sur les CPC 6128. Ce programme ne concerne donc que les possesseurs de 6128.

D'une manière générale, Turbo-Pascal n'est vraiment intéressant que sur les CPC 6128. En effet, sur les CPC 464 et 664, seulement 6 Ko de mémoire sont réservés pour le programme exécutable et les données, ce qui est bien peu. Pour donner un ordre d'idées, le traitement de texte dont le listing est donné ici demande environ 5 Ko de mémoire pour les données, 9 Ko de mémoire pour le code exécutable et 15.6 Ko pour le code source.

Il s'agit d'un traitement de texte pleine page dont les possibilités sont les suivantes :

En mode « un écran » :

- saisie de texte en majuscule ou minuscule ;
- déplacement du curseur grâce aux touches du pavé numérique ;
- touche « back space » effective, qui permet d'effacer le caractère qui se trouve à gauche du curseur ;
- insertion d'une ligne avant la position courante du curseur ;
- suppression de la ligne courante ;
- fin d'édition et sauvegarde magnétique.

En mode « plusieurs écrans » :

Les fonctions disponibles sont les mêmes que celles du mode « un écran ».

Le scrolling d'écran (haut et bas) est automatique lorsque le curseur se trouve positionné respectivement sur la dernière ligne ou sur la première ligne de l'écran.

Le programme est organisé en plusieurs procédures dont voici les listes et les descriptions :

Lecture

Cette procédure lit un caractère tapé au clavier et le renvoie dans la variable caractère **Ch**.

On notera l'utilisation de l'ordre **keypressed** de la manière suivante :

While not keypressed do;

qui signifie : tant qu'aucune touche du clavier n'est pressée, attendre.

On notera également l'élimination du code ESCape (127). En effet, ce caractère précède le code de certaines touches pour préciser que leur fonction est spéciale.

```

Procédure Lecture;
(*-----*)
(* Lecture d'une touche      *)
(*-----*)
(* Entree : Aucune          *)
(* Sortie : Ch=Caractere lu *)
(*-----*)
begin
  while not keypressed do; (* Attente appui sur une touche *)
    read(Kbd,Ch);
end;
```

Lit

Cette procédure lit plusieurs caractères tapés au clavier et les affiche sur l'écran. Le retour chariot en fin de saisie est automatique. La longueur du texte à saisir est placée dans la variable **longueur** par l'appelant, et le texte entré au clavier est disponible dans la variable **A** en sortie de la procédure.

```

Procédure Lit(Longueur:Integer);
(*-----*)
(* Lecture d'une chaine au clavier      *)
(*-----*)
(* Entree : Longueur=Nbre de caracteres a lire *)
(* Sortie : Chaine lue dans A           *)
(*-----*)
begin
  A:= ''; B:= ' '; I:=1;
  repeat
    Lecture;
    Case Ord(Ch) of
      CRLF:  I:=Longueur+1; (* Appui sur Return *)
      BS:    begin (* Appui sur Backspace *)
                If I>1 then
                  begin
                    WhereX:=WhereX-1; I1:=WhereX; I2:=WhereY;
                    GotoXY(I1,I2); Write(' '); GotoXY(I1,I2);
                    I:=I-1;
                    A:=Copy(A,1,Length(A)-1);
                  end;
                end;
            else

```

(Programme suite)

```

begin
  WhereX:=WhereX+1;
  A:=Concat(A,Ch);
  Write(Ch);
  I:=I+1;
end;
end;
until (I>Longueur)
end;

```

Saisie 1

Cette procédure réalise la saisie de texte plein écran en mode **une page**. Le curseur peut être déplacé en utilisant les touches du pavé numérique.

La touche **BS** (Back Space) peut être utilisée pour effacer le caractère qui se trouve à gauche du curseur.

La touche **home** positionne le curseur au début de la ligne courante, et la touche **end** à la fin de la ligne courante. Une ligne peut être insérée avant la ligne courante en appuyant simultanément sur les touches **CTRL et I**.

La ligne courante peut être effacée en appuyant simultanément sur les touches **CTRL et X**.

Enfin, la fin de saisie de texte est signalée par l'appui simultané des touches **CTRL et F**.

```

Procédure Saisie1;
(*-----*)
(* Saisie d'un texte sur 24 lignes sans scrolling *)
(*-----*)
(* Entree : Aucune *)
(* Sortie : Texte entre dans Buf_Ecran *)
(*-----*)
begin
  Clrscr;
  GotoXY(1,1);
  WhereX:=1; WhereY:=1;
  Max:=24;
  If Not Vierge then
  begin
    for I:=1 to 23 do
    begin
      Ch2:=Buf_Ecran[I];
      Writeln(Ch2);
    end;
    Write(Buf_Ecran[24]);
    GotoXY(1,1);
  end
  else Vierge:=False;

```

```

Repeat
  Lecture;                                (* Lecture d'un caractere *)
  Case ord(Ch) of                          (* Caractere affichable  *)
    32..126      : begin
      If WhereX<80 then
        begin
          I1:=WhereY; I2:=WhereX;
          Une_Ligne:=Buf_Ecran[I1];
          Une_Ligne[I2]:=Ch;
          Buf_Ecran[I1]:=Une_Ligne;
          Write(Ch);
          WhereX:=WhereX+1;
          If I2=79 then GotoXY(I2,I1);
        end;
      end;

  BS:          begin      (* Appui sur Backspace *)
    If WhereX>1 then
      begin
        WhereX:=WhereX-1;
        I1:=WhereX; I2:=WhereY;
        GotoXY(I1,I2);
        Write(Blanc);
        GotoXY(I1,I2);
        I1:=WhereY; I2:=WhereX;
        Une_Ligne:=Buf_Ecran[I1];
        Une_Ligne[I2]:=Blanc;
        Buf_Ecran[I1]:=Une_Ligne;
      end
    end;

  Ins_Ligne:   begin      (* Insertion d'une ligne *)
    Incline;
    I3:=WhereY; I3:=I3+1;
    For I:=24 downto I3 do
      begin
        I1:=I-1;
        Buf_Ecran[I1]:=Buf_Ecran[I1];
      end;
    I1:=WhereY;
    Buf_Ecran[I1]:=Lblanc;
  end;

  Eff_Ligne:   begin      (* Effacement de la ligne courante *)
    Sauv_Y:=WhereY;
    I1:=WhereY;
    For I:=I1 to 24 do
      begin
        I1:=I+1;
        Buf_Ecran[I1]:=Buf_Ecran[I1];
      end;
    Buf_Ecran[24]:=LBlanc;
    Deline;
  end;

```

```

        GotoXY(1,24);
        I1:=24;
        Write(Buf_Ecran[I1]);
        GotoXY(1,Sauv_Y);
    end;

CRLF:   begin (* Appui sur Return *)
        I1:=WhereY;
        If I1<>24 then begin
            WhereX:=1; WhereY:=WhereY+1;
            Writeln;
        end
        else GotoXY(1,24);
    end;

Haut:   begin (* Deplacement vers le haut *)
        If WhereY>1 then
        begin
            WhereY:=WhereY-1;
            I1:=WhereX; I2:=WhereY;
            GotoXY(I1,I2);
        end;
    end;

Bas:    begin (* Deplacement vers le bas *)
        If WhereY<24 then
        begin
            WhereY:=WhereY+1;
            I1:=WhereX; I2:=WhereY;
            GotoXY(I1,I2);
        end;
    end;

Droite: begin (* Deplacement vers la droite *)
        If WhereX<79 then
        begin
            WhereX:=WhereX+1;
            I1:=WhereX; I2:=WhereY;
            GotoXY(I1,I2);
        end;
    end;

Gauche: begin (* Deplacement vers la gauche *)
        If WhereX>1 then
        begin
            WhereX:=WhereX-1;
            I1:=WhereX; I2:=WhereY;
            GotoXY(I1,I2);
        end;
    end;

    end;
until ord(Ch)=Fin_de_saisie;
end;

```

Saisie 2

Cette procédure reprend les facilités d'édition de **Saisie 1** mais intègre en plus un scrolling automatique vers le haut et vers le bas lorsque le curseur se trouve respectivement sur la dernière ou sur la première ligne de l'écran.

```

Procédure Saisie2;
(*-----*)
(* Saisie d'un texte sur n lignes avec scrolling *)
(*-----*)
(* Sortie : Texte entre dans Buf_Ecran *)
(*-----*)
begin
  Clrscr;
  GotoXY(1,1);
  Dec_Y:=0;
  WhereX:=1; WhereY:=1;
  If Not Vierge then
  begin
    For I:=1 to 23 do
      Writeln(Buf_Ecran[I]);
    Write(Buf_Ecran[24]);
    GotoXY(1,1);
  end
  else Vierge:=False;

  Repeat
    Lecture;
    Case ord(Ch) of

      32..126   : begin
                    If WhereX<80 then
                    begin
                      I1:=WhereY; I1:=I1+Dec_Y; I2:=WhereX;
                      Une_Ligne:=Buf_Ecran[I1];
                      Une_Ligne[I2]:=Ch;
                      Buf_Ecran[I1]:=Une_Ligne;
                      Write(Ch);
                      WhereX:=WhereX+1;
                      If I2=79 then GotoXY(I2,WhereY);
                    end;
                  end;

      BS        : begin
                    If WhereX>1 then
                    begin
                      WhereX:=WhereX-1;
                      I1:=WhereX; I2:=WhereY;
                      GotoXY(I1,I2);
                      Write(Blanc);
                      GotoXY(I1,I2);

```

```

        I1:=WhereY; I1:=I1+Dec_Y; I2:=WhereX;
        Une_Ligne:=Buf_Ecran[I1];
        Une_Ligne[I2]:=Blanc;
        Buf_Ecran[I1]:=Une_Ligne;
    end;
end;

Ins_Ligne : begin
    InsLine;
    I3:=WhereY; I3:=I3+1+Dec_Y;
    For I:=Max downto I3 do
    begin
        I1:=I-1;
        Buf_Ecran[I]:=Buf_Ecran[I1];
    end;
    I1:=WhereY; I1:=I1+Dec_Y;
    Buf_Ecran[I1]:=Lblanc;
end;

Eff_Ligne : begin
    Sauv_Y:=WhereY;
    I1:=WhereY; I1:=I1+Dec_Y;
    For I:=I1 to (Max-1) do
    begin
        I1:=I+1;
        Buf_Ecran[I]:=Buf_Ecran[I1];
    end;
    Buf_Ecran[Max]:=Lblanc;
    DelLine;
    GotoXY(1,24);
    I1:=24+Dec_Y;
    Write(Buf_Ecran[I1]);
    WhereX:=1;
    GotoXY(1,Sauv_Y);
end;

CRLF      : A_la_ligne;

Haut      : Depl_haut;

Bas       : Depl_bas;

Droite    : begin
    If WhereX<79 then
    begin
        WhereX:=WhereX+1;
        I1:=WhereX; I2:=WhereY;
        GotoXY(I1,I2);
    end;
end;

```

(Programme procédure Saisie 2 suite)

```

        Gauche      : begin
                        If WhereX>1 then
                        begin
                            WhereX:=WhereX-1;
                            I1:=WhereX; I2:=WhereY;
                            GotoXY(I1,I2);
                        end;
                    end;
    until ord(Ch)=Fin_de_Saisie;
end;

```

Depl_haut

Cette procédure déplace le curseur vers le haut et réalise éventuellement un scrolling d'écran si nécessaire (curseur positionné sur la première ligne de l'écran).

```

Procedure Depl_Haut;
(*-----*)
(* Déplacement vers le haut *)
(*-----*)
begin
    If WhereY<>1
    then
        begin
            WhereY:=WhereY-1;
            I1:=WhereX; I2:=WhereY;
            GotoXY(I1,I2);
        end
    else
        if Dec_Y<>0 then
            begin
                Sauv_X:=WhereX;
                Inline;
                GotoXY(1,1);
                Write(Buf_Ecran[Dec_Y]);
                GotoXY(Sauv_X,1);
                Dec_Y:=Dec_Y-1;
            end;
    end;
end;

```

A_la_ligne

Cette procédure positionne le curseur au début de la ligne qui suit la ligne courante, et réalise éventuellement un scrolling d'écran si nécessaire (curseur positionné sur la dernière ligne d'écran).

```

Procédure A_la_ligne;

(*-----*)
(* Passage a la ligne *)
(*-----*)
begin
  I1:=WhereY;
  If I1<>24 then begin
    Writeln;
    WhereY:=WhereY+1;
    WhereX:=1;
  end
  else if Dec_Y<>(Max-24) then
  begin
    I1:=Dec_Y+25;
    Write(Buf_Ecran[I1]);
    GotoXY(1,24);
    Dec_Y:=Dec_Y+1;
    WhereX:=1;
  end;
end;

```

Depl_bas

Cette procédure déplace le curseur vers le bas et réalise éventuellement un scrolling d'écran si nécessaire (curseur positionné sur la dernière ligne de l'écran).

```

Procédure Depl_bas;
(*-----*,
(* Déplacement vers le bas *)
(*-----*)
begin
  I1:=WhereY;
  If I1<>24 then
  begin
    WhereY:=WhereY+1;
    I1:=WhereX; I2:=WhereY;
    GotoXY(I1,I2)
  end
  else If Dec_Y<>(Max-24) then
  begin
    Sauv_X:=WhereX;
    I1:=Dec_Y+25;
    GotoXY(80,24);
    Writeln;
    Write(Buf_Ecran[I1]);
    GotoXY(Sauv_X,24);
    Dec_Y:=Dec_Y+1;
  end;
end;

```

Initialisation

Cette procédure initialise les diverses variables du programme en début d'exécution.

```

Procédure Initialisation;
(*-----*)
(* Initialisation du programme *)
(*-----*)
begin
  Lblanc:=''; b:=' ';
  For I:=1 to 79 do Lblanc:=Concat(Lblanc,b);
  For I:=1 to Max do Buf_Ecran[I]:=Lblanc;
  Vierge:=True;
end;

```

Beep

Cette procédure émet un beep sonore. Elle est appelée par le programme principal ou par les diverses procédures lorsqu'une erreur d'ordre général se produit. Voir la liste de la procédure Beep partie 4 chapitre 4.5.2 pages 4 à 6.

Sauvegarde

Cette procédure sauvegarde sur disquette ou disque dur le texte entré. La sauvegarde se fait dans un fichier texte. La première ligne du fichier texte précise le nombre de lignes du fichier. Les autres lignes représentent le texte entré, en clair.

```

Procédure Sauvegarde;
(*-----*)
(* Sauvegarde d'un fichier texte *)
(*-----*)
begin
  Clrscr;
  Assign(Fichier_Ecr,Nom_Ecr);
  Rewrite(Fichier_Ecr);
  Str(Max,S1);
  Writeln(Fichier_Ecr,S1);
  S1:=Nom_Ecr;
  For I:=1 to Max do
    writeln(Fichier_Ecr,Buf_Ecran[I]);
  Close(Fichier_Ecr);
end;

```

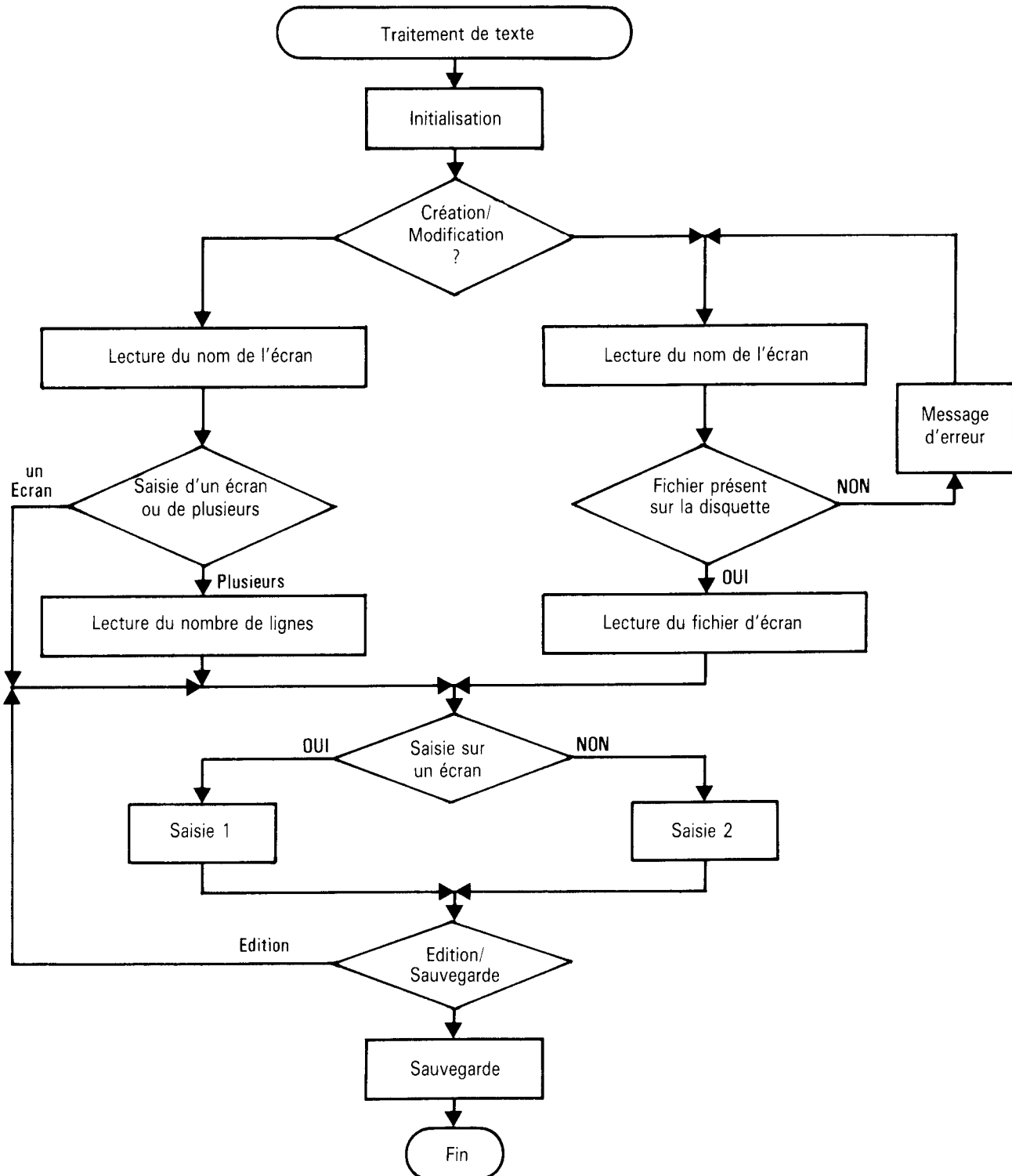
Retrouve

Cette procédure lit le fichier écran dont le nom a été précisé. Les 25 premières lignes sont affichées à l'écran, et le curseur est positionné en haut et à gauche de l'écran.

```
Procedure Retrouve;  
(*-----*)  
(* Lecture d'un fichier texte *)  
(*-----*)  
begin  
  Vierge:=False;  
  Clrscr;  
  Assign(Fichier_Ecr,Nom_Ecr);  
  Reset(Fichier_Ecr);  
  Readln(Fichier_Ecr,S1);  
  Val(S1,Max,Result);  
  If Max=24 then Choix:=1 else Choix:=2;  
  For I:=1 to Max do  
    Readln(Fichier_Ecr,Buf_Ecran[I]);  
  Close(Fichier_Ecr);  
end;
```

Programme principal

Il fait appel aux diverses procédures que nous venons d'énumérer selon l'ordre précisé dans l'ordinogramme ci-dessous :



Ordinogramme du traitement de texte

Le programme de traitement de texte en Turbo-Pascal ®

```
Program Traitement_de_texte_1;
```

```
const
  Fin_de_saisie = 6;      (* Ctrl F : Marque la fin de saisie *)
  BS            = 127;    (* Caractere Back Space *)
  Ins_Ligne     = 9;      (* Ctrl I: Insere une ligne blanche *)
  CRLF          = 13;    (* Passage a la ligne *)
  Eff_Ligne     = 24;    (* Ctrl X: Efface la ligne courante *)
  Haut          = 240;   (* Caractere deplacement vers le haut *)
  Bas           = 241;   (* Caractere deplacement vers le bas *)
  Droite        = 243;   (* Caractere deplacement vers la droite *)
  Gauche        = 242;   (* Caractere deplacement vers la gauche *)
  DELETE        = 127;   (* Caractere effacement *)
  Blanc         = ' ';   (* Caractere espace *)
  NL            = 66;    (*
```

```
type
  Ligne      = String[79];
```

```
var
  Buf_Ecran: Array[1..100] of Ligne; (* Buffer d'ecran *)
  Ch: Char; (* Sauv caractere lu au clavier *)
  Une_Ligne: Ligne; (* Buffer ecran une ligne *)
  I,J: Integer;
  Sauv_X,
  Sauv_Y: Integer; (* Sauvegarde Pos en X et en Y *)
  Dec_Y: Integer; (* Decalage en Y ds la fenetre *)
  Lblanc: Ligne; (* Ligne blanche *)
  Max: Integer; (* Nombre de lignes a traiter *)
  Result: Integer; (* Resultat des proc standard *)
  I1,I2,I3,I5: Integer;
  Ch1,B: Char;
  Ch2,Ch3,S1: Ligne;
  Vierge: Boolean;
  A: String[20];
  Car: Char;
  Nom_Ecr: String[9];
  Fichier_Ecr: Text; (* Nom logique fichier ecran *)
  Operation: Array[1..5] of Char;
  Choix,
  Choix2: Integer;
  Mod_Cre: Integer;
  WhereX,
  WhereY: Integer;
```

```

(* Programme principal *)
begin
  Clrscr;
  GotoXY(1,1);
  Initialisation;
  Writeln('S'agit-il: 1) d'une creation,');
  repeat
    GotoXY(1,4);
    Write('          2) d'une modification: -',chr(8));
    Lit(1);
  until (A='1') or (A='2');
  Val(A,Mod_Cre,Result);

  If A='2' then
  repeat
    GotoXY(1,6);
    Write('Entrez le nom de l'ecran: -----',chr(8),chr(8),chr(8),chr(8));
    Write(chr(8),chr(8),chr(8),chr(8),chr(8));
    Lit(9);
    I1:=Pos(' ',A);
    If I1=0 then Nom_Ecr:=A
      else Nom_Ecr:=Copy(A,1,I1-1);
    Assign(Fichier_Ecr,Nom_Ecr);
    {#I-}
    Reset(Fichier_Ecr);
    {#I+}
    I1:=IOResult;
    If I1<>0 then
    begin
      GotoXY(1,8);
      Write('Fichier Inexistant');
    end
    else Close(Fichier_Ecr);
  until I1=0
  else
  begin
    GotoXY(1,6);
    Write('Entrez le nom de l'ecran: -----',chr(8),chr(8),chr(8),chr(8));
    Write(chr(8),chr(8),chr(8),chr(8),chr(8));
    WhereX:=27; WhereY:=6;
    Lit(9);
    I1:=Pos(' ',A);
    If I1=0 then Nom_Ecr:=A
      else Nom_Ecr:=Copy(A,1,I1-1);
  end;
  If Mod_Cre=2 then Retrouve
  else
  begin
    Vierge:=True;
    Clrscr;
    GotoXY(1,1);
    Writeln('La saisie se fera-t-elle sur:');
    writeln('      1) un ecran,');
    writeln('      2) plusieurs ecrans');
    Repeat
      GotoXY(1,7);
      Write('Votre choix : -',chr(8));
      Lit(1); Val(A,Choix,Result);
    until (Choix=1) or (Choix=2);
    Clrscr;
  end;
end;

```

(Programme principal suite)

```
    If (Choix=2) then begin
        Writeln; Writeln;
        Write('Nombre de lignes: ---',chr(8),chr(8),chr(8));
        Lit(3); Val(A,Max,Result);
    end;

Repeat
    If Choix=1 then Saisie1
    else Saisie2;

    Clrscr;
    GotoXY(1,10);
    Writeln('Voulez-vous: ');
    Writeln('    1) Retourner a l''edition,');
    Repeat
        GotoXY(1,12);
        Write('    2) Sortir de l''editeur : -',chr(8));
        Lit(1); Val(A,Choix2,Result);
    until (Choix2=1) or (Choix2=2);
until Choix2=2;
Sauvegarde;
end.
```

4/5.4

Edition des programmes écrits en Forth

Jusqu'à présent, nous nous sommes contentés de définir des mots depuis le clavier. Mais si nous voulons modifier une définition, il faut la retaper entièrement. Si cela n'est guère gênant pour une première phase visant à l'apprentissage de Forth, il n'en est plus de même pour un programme plus conséquent. Comme tous les autres langages, Forth permet la sauvegarde des programmes source dans des fichiers.

I. Création d'un fichier de blocs

Les programmes écrits en Forth sont sauvegardés dans des blocs de 1 024 caractères constitués de 16 lignes comprenant 64 caractères. Un fichier contiendra au moins deux blocs. Pour créer un fichier destiné à éditer votre travail, il faut utiliser le mot **CREATE-FILE** précédé du nombre de blocs à initialiser dans le fichier :

10 CREATE-FILE TRAVAIL.BLK

crée un fichier nommé **TRAVAIL.BLK** de 10 blocs de longueur numérotés de 0 à 9. L'ouverture du fichier est ensuite activée par le mot **OPEN** suivi du nom du fichier à ouvrir :

OPEN TRAVAIL.BLK

Si vous disposez d'un second lecteur de disquette, vous pouvez traiter un fichier provenant du second lecteur :

10 CREATE-FILE B:TRAVAIL.BLK

OPEN B:TRAVAIL.BLK

L'exécution de **OPEN** crée un mot dans le dictionnaire nommé **TRAVAIL.BLK** ou **B:TRAVAIL.BLK**. Si ce mot existe déjà, le mot n'est pas recréé. Pour rouvrir ultérieurement ce fichier, il suffira, au cours d'une même session de travail en FORTH, de taper **TRAVAIL.BLK**.

La taille maximale d'un fichier créé par **CREATE-FILE** dépend de l'espace mémoire disponible sur le disque courant (A: ou B:).

II. Edition d'un bloc

Dès qu'un fichier est ouvert, on peut visualiser son contenu en exécutant le mot **LIST** précédé du numéro du bloc concerné. Le numéro du bloc doit être compris entre 0 et la valeur délivrée par l'exécution de **CAPACITY** moins une unité :

OPEN TRAVAIL.BLK

CAPACITY 1 - . affiche 9

Pour le fichier **TRAVAIL.BLK**, le numéro du bloc auquel on désire accéder doit être situé entre 0 et 9.

Comme notre fichier ne contient encore aucun programme, passons directement en édition en exécutant le mot **EDIT** précédé du numéro de bloc à éditer :

1 EDIT

III. Résumé des commandes d'édition

A. COMMANDES DE BLOCS

A	Fonction de bascule entre le bloc courant et le bloc commentaire associé.
B	Appelle comme bloc courant le bloc précédent
DONE	Sortie du mode éditeur avec mise à jour du fichier
ED	Edite le bloc courant
n EDIT	Edite le bloc numéro n
L	Liste le bloc courant
n LIST	Liste le bloc numéro n
N	Appelle comme bloc courant le bloc suivant
QUIT	Sortie de l'éditeur sans sauvegarde des modifications du contenu du bloc
W	Sauvegarde sur disque le contenu du bloc courant
WIPE	Efface le contenu du bloc courant en totalité

B. COMMANDES DE CURSEUR

n C	Déplace le curseur de n caractères vers la droite si n est positif, de n caractères vers la gauche si n est négatif
n +T	Déplace le curseur de n lignes depuis la ligne courante
n T	Positionne le curseur au début de la ligne n

TOP Positionne le curseur au début de la ligne 0
n NEW Commande combinant de manière répétitive les commandes T et P

C. MANIPULATION DE TEXTE

D txt Cherche le texte et l'efface
E txt Efface le texte trouvé par F ou S
F txt Cherche le texte dans le bloc courant et place le curseur à la suite
I txt Insère le texte à la suite du curseur
O txt Surimpressionne le texte à la suite du curseur
P txt Remplace le texte sur la ligne courante
R txt Remplace le texte trouvé par F ou S
n S txt Cherche le texte jusqu'à l'écran n ou à la fin du fichier et place le curseur à sa suite

IV. Sélection d'un terminal

F83 est étudié pour fonctionner sur de nombreux types de terminaux :

HEAT	FALCO	TELEVIDEO
QUME	ANSI	PERKIN
DUMB		

C'est le dernier terminal qui est sélectionné par défaut. Dans cette configuration, votre Amstrad réagira comme un terminal télétype, ce qui n'est pas très pratique.

Selon que vous disposez d'un Amstrad CPC ou PCW, il faut sélectionner un de ces terminaux en tapant simplement son nom. Pour initialiser correctement les attributs de positionnement du curseur pour les Amstrad de type CPC, il faut en plus compiler ces définitions :

```
: LOCATE 31 EMIT EMIT EMIT ;  
' LOCATE IS AT  
: CLS 12 EMIT #LINE OFF #OUT OFF ;  
' CLS IS DARK
```

Ces définitions peuvent être sauvegardées définitivement sous forme compilée en tapant **BYE** ; prenez note du nombre de pages affiché après exécution de **BYE**. Sous CP/M, tapez ensuite :

nn SAVE NF83.COM où nn est la valeur affichée après exécution de **BYE**.

Pour revenir sous Forth, il suffit maintenant de taper **NF83**. Vos définitions compilées ont été conservées, vos sélections exécutées avant **BYE** restent opérationnelles.

4/6

Travail en Assembleur 8080 sous CP/M 2.2 ou CP/M Plus

Les programmes utilitaires ASM, LINK et DDT fournis avec le CP/M 2.2 ou le CP/M Plus permettent de travailler en Assembleur 8080. Nous allons dans un premier temps analyser en détails les instructions du 8080. Nous poursuivrons notre étude en commentant l'utilisation de ASM, LINK et DDT. Et enfin, nous la cloturerons par de nombreux exemples d'applications.

4/6.1

Les instructions du 8080

Liste résumée des instructions du 8080

Pour augmenter la lisibilité des instructions, nous avons adopté les conventions suivantes :

r et r'	désignent un des registres du microprocesseur,
nn	désigne une constante de longueur un octet,
nnnn	désigne une constante de longueur deux octets.

Le registre des indicateurs contient les informations suivantes :

C	Retenue
H	Semi-retenue (bit 3 vers bit 4)
N	Négatif
O	Débordement
P	Parité
S	Signe
Z	Zéro

A

ACI nn	Addition de l'accumulateur, de la retenue et de la constante nn
ADC M	Addition de la mémoire d'adresse HL, de l'accumulateur et de la retenue
ADC r	Addition de l'accumulateur, du registre spécifié et de la retenue
ADD M	Addition de la mémoire d'adresse HL et de l'accumulateur
ADD r	Addition de l'accumulateur et du registre r
ADI nn	Addition de l'accumulateur et de la constante nn
ANA M	ET logique entre l'accumulateur et la mémoire d'adresse HL
ANA r	ET logique entre l'accumulateur et le registre r
ANI nn	ET logique entre l'accumulateur et la constante nn

C

CALL nnnn	Appel inconditionnel du sous-programme d'adresse nnnn
CC nnnn	Appel du sous-programme d'adresse nnnn si C = 1
CM nnnn	Appel du sous-programme d'adresse nnnn si S = 1
CNC nnnn	Appel du sous-programme d'adresse nnnn si C = 0
CNZ nnnn	Appel du sous-programme d'adresse nnnn si Z = 0
CP nnnn	Appel du sous-programme d'adresse nnnn si S = 0
CPE nnnn	Appel du sous-programme d'adresse nnnn si P = 1
CPO nnnn	Appel du sous-programme d'adresse nnnn si P = 0
CZ nnnn	Appel du sous-programme d'adresse nnnn si Z = 1

CMA	Complément à un de l'accumulateur
CMC	Complémentation de l'indicateur de retenue
CMP M	Compare l'accumulateur et la mémoire d'adresse HL
CMP r	Compare l'accumulateur et le registre r
CPI nn	Compare l'accumulateur et la constante nn

D

DAA	Ajustement décimal de l'accumulateur
DAD B	Additionne les registres HL et BC
DAD D	Additionne les registres HL et DE
DAD H	Additionne les registres HL et HL
DAD SP	Additionne les registres HL et SP
DCR M	Décrémente la mémoire d'adresse HL
DCR r	Décrémente le registre r
DCX B	Décrémente le registre BC
DCX D	Décrémente le registre DE
DCX H	Décrémente le registre HL
DCX SP	Décrémente le registre SP
DI	Dévalide les interruptions

E

EI	Autorise les interruptions
----	----------------------------

H

HLT	Suspend l'activité du CPU jusqu'à la prochaine interruption ou jusqu'au RESET
-----	---

I

IN nn	Lit le port nn et place la valeur lue dans l'accumulateur
INR M	Incrémente la mémoire d'adresse HL
INR r	Incrémente le registre r
INX B	Incrémente le registre BC
INX D	Incrémente le registre DE
INX H	Incrémente le registre HL
INX SP	Incrémente le registre SP

J

JMP nnnn	Saut inconditionnel à l'adresse nnnn
JC nnnn	Saut à l'adresse nnnn si C = 1
JM nnnn	Saut à l'adresse nnnn si S = 1
JNC nnnn	Saut à l'adresse nnnn si C = 0
JNZ nnnn	Saut à l'adresse nnnn si Z = 0
JP nnnn	Saut à l'adresse nnnn si S = 0
JPE nnnn	Saut à l'adresse nnnn si P = 1
JPO nnnn	Saut à l'adresse nnnn si P = 0
JZ nnnn	Saut à l'adresse nnnn si Z = 1

L

LDA nnnn	Charge l'accumulateur avec le contenu de l'adresse nnnn
LDAX B	Charge l'accumulateur avec le contenu de l'adresse qui se trouve dans le registre BC
LDAX D	Charge l'accumulateur avec le contenu de l'adresse qui se trouve dans le registre DE
LHLD nnnn	Charge le registre L avec le contenu de l'adresse nnnn et le registre H avec le contenu de l'adresse nnnn + 1
LXI B,nnnn	Charge le registre BC avec la constante nnnn
LXI D,nnnn	Charge le registre DE avec la constante nnnn
LXI H,nnnn	Charge le registre HL avec la constante nnnn
LXI SP,nnnn	Charge le registre SP avec la constante nnnn

M

MOV M,r	Stocke le contenu du registre r dans la mémoire d'adresse HL
MOV r,M	Stocke dans le registre r le contenu de la mémoire d'adresse HL
MOV r,r'	Stocke le contenu du registre r' dans le registre r
MVI M,nn	Charge la constante nn dans la mémoire d'adresse HL
MVI r,nn	Charge la constante nn dans le registre r

N

NOP	Aucune opération
-----	------------------

O

ORA M	OU logique entre l'accumulateur et la mémoire d'adresse HL
ORA r	OU logique entre l'accumulateur et le registre r
ORI nn	OU logique entre l'accumulateur et la constante nn
OUT nn	Le contenu de l'accumulateur est placé sur le port d'adresse nn

P

PCHL	Charge le compteur de programme (PC) avec la valeur contenue dans le registre HL
POP B	Charge le registre BC avec les deux octets supérieurs de la pile
POP D	Charge le registre DE avec les deux octets supérieurs de la pile
POP H	Charge le registre HL avec les deux octets supérieurs de la pile
POP PSW	Charge les indicateurs avec l'octet au sommet de la pile et l'accumulateur avec l'octet suivant
PUSH B	Empile le contenu du registre BC
PUSH D	Empile le contenu du registre DE
PUSH H	Empile le contenu du registre HL
PUSH PSW	Empile le contenu des indicateurs et de l'accumulateur

R

RAL	Rotation vers la gauche de l'accumulateur
RAR	Rotation vers la droite de l'accumulateur
RET	Retour de sous-programme
RC	Retour de sous-programme si C = 1
RM	Retour de sous-programme si S = 1
RNC	Retour de sous-programme si C = 0
RNZ	Retour de sous-programme si Z = 0
RP	Retour de sous-programme si S = 0
RPE	Retour de sous-programme si P = 1
RPO	Retour de sous-programme si P = 0
RZ	Retour de sous-programme si Z = 1

RLC	Rotation vers la gauche de l'accumulateur
RRC	Rotation vers la droite de l'accumulateur
RST 0 à RST 7	Débranchement aux sous-programmes d'adresse (respectivement) #00, #08, #10, #18, #20, #28, #30, #38

S

SBB M	Soustrait de l'accumulateur l'octet d'adresse HL et la retenue
SBB r	Soustrait de l'accumulateur le registre r et la retenue
SBI nn	Soustrait de l'accumulateur la constante nn et la retenue
SHLD nnnn	Stocke à l'adresse nnnn le contenu du registre L et à l'adresse nnnn + 1 le contenu de l'adresse nnnn + 1
SPHL	Stocke le contenu du registre HL dans SP
STA nnnn	Stocke l'accumulateur dans la mémoire d'adresse nnnn
STAX B	Stocke l'accumulateur dans la mémoire d'adresse BC
STAX D	Stocke l'accumulateur dans la mémoire d'adresse DE
STC	Met à un l'indicateur de retenue
SUB M	Soustrait de l'accumulateur l'octet d'adresse HL
SUB r	Soustrait de l'accumulateur le contenu du registre r
SUI nn	Soustrait de l'accumulateur la constante nn

X

XCHG	Echange le contenu des registres DE et HL
XRA M	OU exclusif entre l'accumulateur et la mémoire d'adresse HL
XRA r	OU exclusif entre l'accumulateur et le registre r
XRI nn	OU exclusif entre l'accumulateur et la constante nn
XTHL	Echange d'une part le contenu de SP et le registre L, et d'autre part le contenu de SP + 1 et le registre H.

Les instructions du 8080 en détail

Dans un souci d'uniformisation, chacune des instructions du 8080 est présentée selon le même modèle, à savoir :

Code Op

Explication détaillée de l'instruction
Pseudo-code
Indicateurs modifiés par l'instruction
Instruction équivalente en Z80

L'instruction équivalente Z80 est donnée pour faciliter l'apprentissage du 8080 aux programmeurs Z80.

ACI nn

Addition de l'accumulateur, de la retenue et de la constante nn
 $A \leftarrow A + C + nn$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADC A,nn

ADC M

Addition de la mémoire d'adresse HL, de l'accumulateur et de la retenue
 $A \leftarrow A + C + (HL)$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADC A,(HL)

ADC r

Addition de l'accumulateur, du registre spécifié et de la retenue
 $A \leftarrow A + r + C$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADC A,r

ADD M

Addition de la mémoire d'adresse HL et de l'accumulateur
 $A \leftarrow A + (HL)$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADD A,(HL)

ADD r

Addition de l'accumulateur et du registre r
 $A \leftarrow A + r$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADD A,r

ADI nn

Addition de l'accumulateur et de la constante nn

$A \leftarrow A + nn$

Indicateurs modifiés : C, H, O, S, Z, N

Equivalent Z80 : ADD A,nn

ANA M

ET logique entre l'accumulateur et la mémoire d'adresse HL

$A \leftarrow A \text{ AND } (HL)$

Indicateurs modifiés : P, S, Z, C, N, H

Equivalent Z80 : AND (HL)

ANA r

ET logique entre l'accumulateur et le registre r

$A \leftarrow A \text{ AND } r$

Indicateurs modifiés : P, S, Z, C, N, H

Equivalent Z80 : AND r

ANI nn

ET logique entre l'accumulateur et la constante nn

$A \leftarrow A \text{ AND } nn$

Indicateurs modifiés : P, S, Z, C, N, H

Equivalent Z80 : AND nn

CALL nnnn

Appel inconditionnel du sous-programme d'adresse nnnn

$PC \leftarrow nnnn$

Indicateurs modifiés : Aucun

Equivalent Z80 : CALL nnnn

CC nnnn

Appel du sous-programme d'adresse nnnn si C = 1

If C = 1 then $PC \leftarrow nnnn$

Indicateurs modifiés : Aucun

Equivalent Z80 : CALL C,nnnn

CM nnnn

Appel du sous-programme d'adresse nnnn si S = 1

If S = 1 then PC $\leftarrow$ nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : CALL M,nnnn

CNC nnnn

Appel du sous-programme d'adresse nnnn si C = 0
If C = 0 then PC $\leftarrow$ nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : CALL NC,nnnn

CNZ nnnn

Appel du sous-programme d'adresse nnnn si Z = 0
If Z = 0 then PC $\leftarrow$ nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : CALL NZ,nnnn

CP nnnn

Appel du sous-programme d'adresse nnnn si S = 0
If S = 0 then PC $\leftarrow$ nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : CALL P,nnnn

CPE nnnn

Appel du sous-programme d'adresse nnnn si P = 1
If P = 1 then PC $\leftarrow$ nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : CALL PE,nnnn

CPO nnnn

Appel du sous-programme d'adresse nnnn si P = 0
If P = 0 then PC $\leftarrow$ nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : CALL PO,nnnn

CZ nnnn

Appel du sous-programme d'adresse nnnn si Z = 1
If Z = 1 then PC $\leftarrow$ nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : CALL Z,nnnn

CMA

Complément à un de l'accumulateur

$A \leftarrow \bar{A}$

Indicateurs modifiés : H, N

Equivalent Z80 : CPL

CMC

Complémentation de l'indicateur de retenue

If C = 1 then C = 0 else C = 1

Indicateurs modifiés : C, N

Equivalent Z80 : CCF

CMP M

Compare l'accumulateur et la mémoire d'adresse HL

$A - (HL)$

Indicateurs modifiés : C, H, O, S, Z, N

Equivalent Z80 : CP (HL)

CMP r

Compare l'accumulateur et le registre r

$A - r$

Indicateurs modifiés : C, H, O, S, Z, N

Equivalent Z80 : CP r

CPI nn

Compare l'accumulateur et la constante nn

$A - nn$

Indicateurs modifiés : C, H, O, S, Z, N

Equivalent Z80 : CP nn

DAA

Ajustement décimal de l'accumulateur

Indicateurs modifiés : C, H, O, S, Z

Equivalent Z80 : DAA

DAD B

Additionne les registres HL et BC

$HL \leftarrow HL + BC$

Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADD HL,BC

DAD D

Additionne les registres HL et DE
 $HL \leftarrow HL + DE$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADD HL,DE

DAD H

Additionne les registres HL et HL
 $HL \leftarrow HL + HL$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADD HL,HL

DAD SP

Additionne les registres HL et SP
 $HL \leftarrow HL + SP$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : ADD HL,SP

DCR M

Décrémente la mémoire d'adresse HL
 $(HL) \leftarrow (HL) - 1$
Indicateurs modifiés : H, O, S, Z, N
Equivalent Z80 : DEC (HL)

DCR r

Décrémente le registre r
 $(r) \leftarrow (r) - 1$
Indicateurs modifiés : H, O, S, Z, N
Equivalent Z80 : DEC r

DCX B

Décrémente le registre BC
 $BC \leftarrow BC - 1$
Indicateurs modifiés : Aucun
Equivalent Z80 : DEC BC

DCX D

Décrémente le registre DE
 $DE \leftarrow DE - 1$
Indicateurs modifiés : Aucun
Equivalent Z80 : DEC DE

DCX H

Décrémente le registre HL
 $HL \leftarrow HL - 1$
Indicateurs modifiés : Aucun
Equivalent Z80 : DEC HL

DCX SP

Décrémente le registre SP
 $SP \leftarrow SP - 1$
Indicateurs modifiés : Aucun
Equivalent Z80 : DEC SP

DI

Dévalide les interruptions
Indicateurs modifiés : Aucun
Equivalent Z80 : DI

EI

Autorise les interruptions
Indicateurs modifiés : Aucun
Equivalent Z80 : EI

HLT

Suspend l'activité du CPU jusqu'à la prochaine interruption ou jusqu'au RESET
Indicateurs modifiés : Aucun
Equivalent Z80 : HALT

IN nn

Lit le port nn et place la valeur lue dans l'accumulateur
 $A \leftarrow \text{Port}(nn)$
Indicateurs modifiés : Aucun
Equivalent Z80 : IN A,(nn)

INR M

Incrémente la mémoire d'adresse HL
 $HL \leftarrow (HL) + 1$
Indicateurs modifiés : H, O, S, Z, N
Equivalent Z80 : INC (HL)

INR r

Incrémente le registre r
 $(r) \leftarrow (r) + 1$
Indicateurs modifiés : H, O, S, Z, N
Equivalent Z80 : INC r

INX B

Incrémente le registre BC
 $BC \leftarrow BC + 1$
Indicateurs modifiés : Aucun
Equivalent Z80 : INC BC

INX D

Incrémente le registre DE
 $DE \leftarrow DE + 1$
Indicateurs modifiés : Aucun
Equivalent Z80 : INC DE

INX H

Incrémente le registre HL
 $HL \leftarrow HL + 1$
Indicateurs modifiés : Aucun
Equivalent Z80 : INC HL

INX SP

Incrémente le registre SP
 $SP \leftarrow SP + 1$
Indicateurs modifiés : Aucun
Equivalent Z80 : INC SP

JMP nnnn

Saut inconditionnel à l'adresse nnnn
 $PC \leftarrow nnnn$

Indicateurs modifiés : Aucun
Equivalent Z80 : JP nnnn

JC nnnn

Saut à l'adresse nnnn si C = 1
If C = 1 then PC = nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : JP C,nnnn

JM nnnn

Saut à l'adresse nnnn si S = 1
If S = 1 then PC = nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : JP M,nnnn

JNC nnnn

Saut à l'adresse nnnn si C = 0
If S = 0 then PC = nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : JP NC,nnnn

JNZ nnnn

Saut à l'adresse nnnn si Z = 0
If Z = 0 then PC = nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : JP NZ,nnnn

JP nnnn

Saut à l'adresse nnnn si S = 0
If S = 0 then PC = nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : JP P,nnnn

JPE nnnn

Saut à l'adresse nnnn si P = 1
If P = 1 then PC = nnnn
Indicateurs modifiés : Aucun
Equivalent Z80 : JP PE,nnnn

JPO nnnn

Saut à l'adresse nnnn si $P = 0$
If $P = 0$ then $PC = nnnn$
Indicateurs modifiés : Aucun
Equivalent Z80 : JP PO,nnnn

JZ nnnn

Saut à l'adresse nnnn si $Z = 1$
If $Z = 0$ then $PC = nnnn$
Indicateurs modifiés : Aucun
Equivalent Z80 : JP Z,nnnn

LDA nnnn

Charge l'accumulateur avec le contenu de l'adresse nnnn
 $A \leftarrow (nnnn)$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD A,(nnnn)

LDAX B

Charge l'accumulateur avec le contenu de l'adresse qui se trouve dans le registre BC
 $A \leftarrow (BC)$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD A,(BC)

LDAX D

Charge l'accumulateur avec le contenu de l'adresse qui se trouve dans le registre DE
 $A \leftarrow (DE)$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD A,(DE)

LHLD nnnn

Charge le registre L avec le contenu de l'adresse nnnn et le registre H avec le contenu de l'adresse $nnnn + 1$
 $L \leftarrow (nnnn)$ et $H \leftarrow (nnnn + 1)$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD HL,(nnnn)

LXI B,nnnn

Charge le registre BC avec la constante nnnn

$BC \leftarrow nnnn$

Indicateurs modifiés : Aucun

Equivalent Z80 : LD BC,nnnn

LXI D,nnnn

Charge le registre DE avec la constante nnnn

$DE \leftarrow nnnn$

Indicateurs modifiés : Aucun

Equivalent Z80 : LD DE,nnnn

LXI H,nnnn

Charge le registre HL avec la constante nnnn

$HL \leftarrow nnnn$

Indicateurs modifiés : Aucun

Equivalent Z80 : LD HL,nnnn

LXI SP,nnnn

Charge le registre SP avec la constante nnnn

$SP \leftarrow nnnn$

Indicateurs modifiés : Aucun

Equivalent Z80 : LD SP,nnnn

MOV M,r

Stocke le contenu du registre r dans la mémoire d'adresse HL

$(HL) \leftarrow (r)$

Indicateurs modifiés : Aucun

Equivalent Z80 : LD (HL),r

MOV r,M

Stocke dans le registre r le contenu de la mémoire d'adresse HL

$(r) \leftarrow (HL)$

Indicateurs modifiés : Aucun

Equivalent Z80 : LD r,(HL)

MOV r,r'

Stocke le contenu du registre r' dans le registre r

$(r) \leftarrow (r')$

Indicateurs modifiés : Aucun
Equivalent Z80 : LD r,r'

MVI M,nn

Charge la constante nn dans la mémoire d'adresse HL
 $(HL) \leftarrow nn$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD (HL),nn

MVI r,nn

Charge la constante nn dans le registre r
 $(r) \leftarrow nn$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD r,nn

NOP

Aucune opération
Indicateurs modifiés : Aucun
Equivalent Z80 : NOP

ORA M

OU logique entre l'accumulateur et la mémoire d'adresse HL
 $A \leftarrow A \text{ OR } (HL)$
Indicateurs modifiés : P, S, Z, C, H, N
Equivalent Z80 : OR (HL)

ORA r

OU logique entre l'accumulateur et le registre r
 $A \leftarrow A \text{ OR } (r)$
Indicateurs modifiés : P, S, Z, C, H, N
Equivalent Z80 : OR r

ORI nn

OU logique entre l'accumulateur et la constante nn
 $A \leftarrow A \text{ OR } nn$
Indicateurs modifiés : P, S, Z, C, H, N
Equivalent Z80 : OR nn

OUT nn

Le contenu de l'accumulateur est placé sur le port d'adresse nn

$\text{Port}(\text{nn}) \leftarrow A$

Indicateurs modifiés : Aucun

Equivalent Z80 : OUT (nn),A

PCHL

Charge le compteur de programme (PC) avec la valeur contenue dans le registre HL

$\text{PC} \leftarrow \text{HL}$

Indicateurs modifiés : Aucun

Equivalent Z80 : JP (HL)

POP B

Charge le registre BC avec les deux octets supérieurs de la pile

$\text{BC} \leftarrow (\text{SP})$ et $\text{SP} \leftarrow \text{SP} + 2$

Indicateurs modifiés : Aucun

Equivalent Z80 : POP BC

POP D

Charge le registre DE avec les deux octets supérieurs de la pile

$\text{DE} \leftarrow (\text{SP})$ et $\text{SP} \leftarrow \text{SP} + 2$

Indicateurs modifiés : Aucun

Equivalent Z80 : POP DE

POP H

Charge le registre HL avec les deux octets supérieurs de la pile

$\text{HL} \leftarrow (\text{SP})$ et $\text{SP} \leftarrow \text{SP} + 2$

Indicateurs modifiés : Aucun

Equivalent Z80 : POP HL

POP PSW

Charge les indicateurs avec l'octet au sommet de la pile et l'accumulateur avec l'octet suivant

$F \leftarrow (\text{SP})$ $A \leftarrow (\text{SP} + 1)$ et $\text{SP} \leftarrow \text{SP} + 2$

Indicateurs modifiés : Tous

Equivalent Z80 : POP AF

PUSH B

Empile le contenu du registre BC
 $(SP) \leftarrow BC$ et $SP \leftarrow SP - 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : PUSH BC

PUSH D

Empile le contenu du registre DE
 $(SP) \leftarrow DE$ et $SP \leftarrow SP - 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : PUSH DE

PUSH H

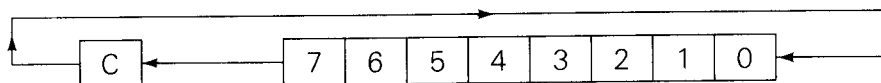
Empile le contenu du registre HL
 $(SP) \leftarrow HL$ et $SP \leftarrow SP - 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : PUSH HL

PUSH PSW

Empile le contenu des indicateurs et de l'accumulateur
 $(SP) \leftarrow F$ $(SP + 1) \leftarrow A$ et $SP \leftarrow SP - 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : PUSH AF

RAL

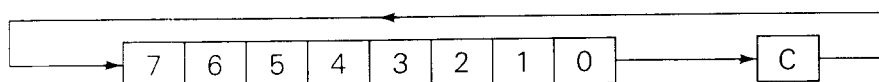
Rotation vers la gauche de l'accumulateur



Indicateurs modifiés : C, H, N
Equivalent Z80 : RLA

RAR

Rotation vers la droite de l'accumulateur



Indicateurs modifiés : C, H, N
Equivalent Z80 : RRA

RET

Retour de sous-programme
 $PC \leftarrow (SP)$ et $SP \leftarrow SP + 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : RET

RC

Retour de sous-programme si C = 1
If C = 1 then $PC \leftarrow (SP)$, $SP \leftarrow SP + 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : RET C

RM

Retour de sous-programme si S = 1
If S = 1 then $PC \leftarrow (SP)$ et $SP \leftarrow SP + 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : RET M

RNC

Retour de sous-programme si C = 0
If C = 0 then $PC \leftarrow (SP)$, $SP \leftarrow SP + 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : RET NC

RNZ

Retour de sous-programme si Z = 0
If Z = 0 then $PC \leftarrow (SP)$, $SP \leftarrow SP + 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : RET NZ

RP

Retour de sous-programme si S = 0
If S = 0 then $PC \leftarrow (SP)$, $SP \leftarrow SP + 2$
Indicateurs modifiés : Aucun
Equivalent Z80 : RET P

RPE

Retour de sous-programme si P = 1
If P = 1 then PC $\leftarrow$ (SP), SP $\leftarrow$ SP + 2
Indicateurs modifiés : Aucun
Equivalent Z80 : RET PE

RPO

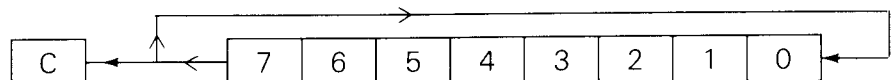
Retour de sous-programme si P = 0
If P = 0 then PC $\leftarrow$ (SP), SP $\leftarrow$ SP + 2
Indicateurs modifiés : Aucun
Equivalent Z80 : RET PO

RZ

Retour de sous-programme si Z = 1
If Z = 1 then PC $\leftarrow$ (SP), SP $\leftarrow$ SP + 2
Indicateurs modifiés : Aucun
Equivalent Z80 : RET Z

RLC

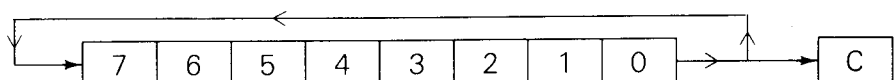
Rotation vers la gauche de l'accumulateur



Indicateurs modifiés : C, H, N
Equivalent Z80 : RLCA

RRC

Rotation vers la droite de l'accumulateur



Indicateurs modifiés : C, H, N
Equivalent Z80 : RRCA

RST 0 à RST 7

Débranchement aux sous-programmes d'adresse (respectivement)
#00, #08, #10, #18, #20, #28, #30, #38
Indicateurs modifiés : Aucun
Equivalent Z80 : RST 00H à RST 38H

SBB M

Soustrait de l'accumulateur l'octet d'adresse HL et la retenue
 $A \leftarrow A - (HL) - C$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : SBC A,(HL)

SBB r

Soustrait de l'accumulateur le registre r et la retenue
 $A \leftarrow A - (r) - C$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : SBC A,r

SBI nn

Soustrait de l'accumulateur la constante nn et la retenue
 $A \leftarrow A - nn - C$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : SBC A,nn

SHLD nnnn

Stocke à l'adresse nnnn le contenu du registre L et à l'adresse nnnn + 1
le contenu du registre H
 $(nnnn) \leftarrow L$ et $(nnnn + 1) \leftarrow H$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD (nnnn),HL

SPHL

Stocke le contenu du registre HL dans SP
 $SP \leftarrow HL$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD SP,HL

STA nnnn

Stocke l'accumulateur dans la mémoire d'adresse nnnn
 $(nnnn) \leftarrow A$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD (nnnn),A

STAX B

Stocke l'accumulateur dans la mémoire d'adresse BC
 $(BC) \leftarrow A$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD (BC),A

STAX D

Stocke l'accumulateur dans la mémoire d'adresse DE
 $(DE) \leftarrow A$
Indicateurs modifiés : Aucun
Equivalent Z80 : LD (DE),A

STC

Met à un l'indicateur de retenue
 $C \leftarrow 1$
Indicateurs modifiés : C, H, N
Equivalent Z80 : SCF

SUB M

Soustrait de l'accumulateur l'octet d'adresse HL
 $A \leftarrow A - (HL)$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : SUB (HL)

SUB r

Soustrait de l'accumulateur le contenu du registre r
 $A \leftarrow A - (r)$
Indicateurs modifiés : C, H, O, S, Z, N
Equivalent Z80 : SUB r

SUI nn

Soustrait de l'accumulateur la constante nn

$A \leftarrow A - nn$

Indicateurs modifiés : C, H, O, S, Z, N

Equivalent Z80 : SUB nn

XCHG

Echange le contenu des registres DE et HL

$Temp \leftarrow DE, DE \leftarrow HL, HL \leftarrow Temp$

Indicateurs modifiés : Aucun

Equivalent Z80 : EX DE,HL

XRA M

OU exclusif entre l'accumulateur et la mémoire d'adresse HL

$A \leftarrow A \text{ XOR } (HL)$

Indicateurs modifiés : P, S, Z, C, H, N

Equivalent Z80 : XOR (HL)

XRA r

OU exclusif entre l'accumulateur et le registre r

$A \leftarrow A \text{ XOR } (r)$

Indicateurs modifiés : P, S, Z, C, H, N

Equivalent Z80 : XOR r

XRI nn

OU exclusif entre l'accumulateur et la constante nn

$A \leftarrow A \text{ XOR } nn$

Indicateurs modifiés : P, S, Z, C, H, N

Equivalent Z80 : XOR nn

XTHL

Echange d'une part le contenu de SP et le registre L, et d'autre part le contenu de SP + 1 et le registre H;

$(Temp1) \leftarrow (SP), (Temp2) \leftarrow (SP + 1),$

$(SP) \leftarrow L, (SP + 1) \leftarrow H,$

$L \leftarrow (Temp1), H \leftarrow (Temp2)$

Indicateurs modifiés : Aucun

Equivalent Z80 : EX (SP),HL