

AMSTRAD

MIEUX PROGRAMMER EN ASSEMBLEUR

THOMAS LACHAND-ROBERT



AMSTRAD

MIEUX PROGRAMMER EN ASSEMBLEUR

Thomas LACHAND-ROBERT

AMSTRAD

MIEUX PROGRAMMER EN ASSEMBLEUR



Paris • Berkeley • Düsseldorf

Sybex n'est lié à aucun constructeur.

Tous les efforts ont été faits pour fournir dans ce livre une information complète et exacte. Néanmoins, SYBEX n'assume de responsabilités, ni pour son utilisation, ni pour les contrefaçons de brevets ou atteintes aux droits de tierces personnes qui pourraient résulter de cette utilisation.

Copyright © Sybex 1986.

Tous droits réservés. Toute reproduction, même partielle, par quelque procédé que ce soit, est interdite sans autorisation préalable. Une copie par xérographie, photographie, film, bande magnétique ou autre, constitue une contrefaçon passible des peines prévues par la loi sur la protection des droits d'auteur.

ISBN : 2-7361-0193-6

■ Sommaire

Avertissement	7
1. LA PROGRAMMATION EN ASSEMBLEUR	9
Notation binaire, hexadécimale, et autres rappels	11
Les Assembleurs du commerce	15
2. L'ASSEMBLEUR DU Z80	23
Le Z80. Les registres	24
Les instructions	30
La pile et les sauts	33
Instructions mal aimées et instructions créées	38
3. LES AMSTRAD CPC	41
La mémoire des CPC	42
Les événements temporisés	46
Les indirectes	47
4. COMMUNICATIONS ENTRE BASIC ET ASSEMBLEUR	51
Les appels du BASIC	52
Les variables du BASIC	54
Les extensions d'instruction	57
5. OPTIMISATION ET PROGRAMMATION	61
Optimisation	62
Programmation	68
6. EXEMPLES DE PROGRAMMES	77
Exemple de RSX : PGCD	79
Diagrammes en barres	82
Exemple de RSX graphique et arithmétique : étoiles	87
Une extension d'instruction graphique : cercle rapide	94
Compilateur/décompilateur d'écran	102
Exemple d'interruption : chronomètre permanent	108
Les calculs sur les entiers pour 664 et 6128	113
7. UN GRAND PROGRAMME : ANNUAIRE TÉLÉPHONIQUE	117
Utilisation	118
Fonctionnement	120
Routines	122

8.LES ROUTINES SYSTÈME DES CPC	147
Les restarts	148
Les vecteurs du noyau	152
Les vecteurs du clavier	155
Les vecteurs de texte	162
Les vecteurs graphiques	171
Les vecteurs de l'écran	177
Les vecteurs cassette/disquette	185
Les vecteurs sonores	192
Autres vecteurs du noyau	195
Les vecteurs des périphériques	201
L'initialisation des vecteurs et l'éditeur de texte	204
Les vecteurs spécifiques des CPC 664 et 6128	205
Les vecteurs de l'arithmétique à virgule flottante	209
Les vecteurs de l'arithmétique entière	219
Annexe A : Les instructions du Z80	223
Annexe B : Les rotations et décalages	229
Annexe C : Les adresses intéressantes	234
Bibliographie	238

■ Avertissement

Le livre que vous tenez entre les mains n'est pas un manuel pour commencer la programmation en assembleur, mais pour la poursuivre et l'améliorer. Il cherchera à vous donner les meilleures chances de bien programmer en assembleur, avec efficacité et rigueur.

C'est aussi un mémorandum des possibilités des trois Amstrad CPC, 464, 664 et 6128, sans faire de préférence : vous en serez, j'espère, satisfait quel que soit le type de votre appareil.

Vous y trouverez en effet de nombreux détails sur les Amstrad eux-mêmes, et tout ce qui permet de rendre cette programmation plus efficace, notamment la liste intégrale (publiée pour la première fois à ma connaissance) de toutes les routines des systèmes d'exploitation pour ces trois machines (Chapitre 8), avec tous les détails nécessaires. Cette liste, les tables des annexes, etc., ont été conçues comme si elles n'étaient destinées qu'à moi-même. Je souhaite que vos exigences soient par elles comblées comme les miennes.

D'une façon générale, j'espère que cet ouvrage vous familiarisera assez avec l'assembleur pour que vous puissiez programmer avec lui non pas tout de même si facilement qu'en BASIC, mais du moins beaucoup plus aisément qu'on ne le croit généralement possible, et que vous contribuerez ainsi à réhabiliter un langage trop souvent oublié, quoiqu'essentiel : celui de la machine.

1. LA PROGRAMMATION EN ASSEMBLEUR

Nous ne nous étendrons guère ici sur les avantages (nombreux) de la programmation en assembleur et sur ses inconvénients, qui sont généralement bien connus des personnes qui s'intéressent à l'informatique, même si elles n'ont jamais programmé en assembleur. Rappelons simplement que l'assembleur est nettement plus rapide que le BASIC (en moyenne dix fois plus), et même que les autres langages dits évolués (trois ou quatre fois plus rapide que le PASCAL par exemple), qu'il utilise moins de mémoire que ces derniers ; tout cela résulte d'une évidence : l'assembleur est, à la conversion des mnémoniques aux codes près, le langage même de la machine. En contrepartie, l'assembleur est plus difficile à maîtriser pour un être humain comme vous et moi que les langages construits pour leur simplicité comme le BASIC.

Cela étant, la maîtrise de l'assembleur est, comme bien des choses, une question d'habitude. Lorsque le pli est pris, et à condition, par conséquent, de ne pas se décourager trop vite, il devient très simple de programmer en assembleur, surtout pour des programmes de faible longueur (quelques centaines de codes). Le débogage (suppression des erreurs) paraît plus aisé, les erreurs sont d'ailleurs moins fréquentes : nombre d'entre elles ne résistent pas à quelques minutes de réflexion, et les autres sont détectables souvent par quelques tests très simples.

Mais il faut émettre une réserve importante à cet idyllique tableau. La programmation en assembleur est en effet relativement simple, une fois que l'on a à peu près compris les instructions du Z80 et leur utilisation.

Malheureusement, de là à programmer correctement en assembleur, en limitant les erreurs, en évitant les longueurs, les pertes de temps, les contradictions, il y a du chemin !

C'est ce chemin que le présent ouvrage prétend vous faire suivre. Il souhaite parvenir à vous montrer comment tirer profit de toutes les instructions du Z80 (Chapitre 2), de toutes les routines du système d'exploitation pour les trois CPC (Chapitre 8), et parvenir à une bonne optimisation de vos programmes (Chapitre 5). Cela nécessite évidemment aussi une bonne connaissance des possibilités des Amstrad (Chapitres 3 et 4). Ces chapitres sont complétés par trois annexes : l'Annexe A vous donnera des renseignements intéressants sur les instructions du Z80 ; l'Annexe B détaillera pour vous les rotations et décalages sur huit bits ; l'Annexe C vous donnera quelques adresses utiles de vos Amstrad. Une bibliographie d'ouvrages complémentaires

est donnée, avec quelques commentaires sur chacun, à la fin de ce livre.

Pour le moment, je vous propose de voir comment vous pouvez programmer en assembleur dans les meilleures conditions.

■ NOTATION BINAIRE, HEXADÉCIMALE ET AUTRES RAPPELS

INTÉRÊT DES NOTATIONS BINAIRE ET HEXADÉCIMALE

S'il est une chose fondamentale pour pouvoir vraiment programmer en assembleur, c'est de maîtriser et d'utiliser les notations binaire et hexadécimale des nombres.

Cela ne signifie pas que vous devez être capable de convertir instantanément 9ABFH en décimal (cela n'a aucun intérêt, d'ailleurs), ni même 100101B. Par contre, vous devez pouvoir voir que 9ABFH est plus grand que 93FEH mais plus petit que 9B44H, et aussi que $9ABFH + 3$ donne 9AC2H, et de même pour des nombres binaires (sauf pour les additions, peu intéressantes en binaire) ; pour revoir tout cela, regardez le paragraphe suivant.

Pourquoi est-ce si important ? Pas du tout pour l'amour de l'art. Mais vous devez comprendre qu'un ordinateur est un objet fondamentalement binaire : il compte tout en binaire. Ses registres ont une longueur comptée en bits (un bit est un chiffre binaire, qui vaut donc 0 ou 1). Ses adresses se mesurent en bits, etc.

Il en résulte que, pour savoir si le troisième bit de l'accumulateur est à zéro ou non, savoir que cet accumulateur contient la valeur 141 ne vous avancera guère. Par contre, savoir qu'il contient 8DH, soit 10001101B répondra immédiatement à votre question.

Dans le même ordre d'idées, les adresses internes des Amstrad sont des nombres ronds en hexadécimal. Ainsi la RAM écran, par exemple, va de C000H à FFFFH. Dès lors, si vous écrivez vos adresses 49682, comment savoir s'il s'agit d'une adresse de l'écran ? Alors qu'il est évident que C212H en est une.

Bien sûr, vous pourriez convertir toutes les adresses internes importantes en décimal. Mais vous ne me ferez pas croire que 4000H n'est pas plus simple à retenir que 16384, ni A700H que 42752 !

Le lecteur aura d'ores et déjà remarqué les notations qui seront celles de tout ce livre : la lettre H indique un nombre en hexadéci-

mal, B en binaire et D (ou rien) un nombre en décimal ; ces lettres figurent à la fin des nombres.

BINAIRE ET HEXADÉCIMAL. CONVERSION ET COMPARAISON

Lorsque l'on compte, la base représente le nombre de chiffres différents utilisés. Ainsi, nous avons l'habitude depuis l'enfance de compter en base 10, avec les dix chiffres 0, 1, ..., 9. Cet état de fait précède de quelques millénaires les ordinateurs, nous n'y reviendrons pas ; mais il résulte de la présence de dix doigts sur nos mains.

L'ordinateur, lui, utilise le courant électrique, et mesure la présence ou l'absence de tension. Il ne mesure donc que deux états différents, il compte en binaire. Il n'y a plus que deux chiffres : 0 et 1.

Les règles donnant les valeurs des nombres dans une base donnée sont toujours les mêmes. Si b est la base, b se note (en base b) 10 ; $b*b$ (b^2), 100 ; b^3 , 1000 ; etc. 1 se note toujours 1 et 0, 0. Dès lors, la valeur d'un nombre noté en base b : ...xyz, est égale à la somme des produits des chiffres z , y , x , etc. (donc en partant du dernier chiffre des unités), par les puissances successives de b (y compris $b^0=1$) multipliées par ces chiffres, soit $z*1 + y*b + x*b^2 + \dots$

Ainsi

$$1986D = 6 + 8*10 + 9*100 + 1*1000,$$

et

$$101B = 1 + 0*10B + 1*100B,$$

soit 5 en décimal car $10B=2$ (base) et $100B=4$ ($2*2$).

Pour convertir un nombre décimal en binaire, il faut diviser par 2 autant de fois que nécessaire pour avoir un quotient nul, et noter tous les restes successifs en les écrivant de droite à gauche. Ainsi, 13 vaut 1101B car $13/2=6$, reste 1 ; $6/2=3$, reste 0 ; $3/2=1$, reste 1 ; $1/2=0$ reste 1, et arrêt car le quotient est nul.

On ne peut pas dire que ce soit très simple, aussi évitera-t-on la notation décimale en général.

Le problème, c'est que plus la base est petite, plus le nombre de chiffres pour écrire une valeur donnée est grand. Ainsi, 1986 prend quatre chiffres en décimal, mais onze en binaire (11111000010B).

On comprend alors qu'il vaille mieux trouver une notation moins lourde, et néanmoins permettant des échanges faciles avec le binaire : c'est la notation hexadécimale, en base 16.

Dans cette base, il y a donc seize chiffres, soit six de plus qu'en décimal. Ces six chiffres, qui suivent 0, 1, ..., 9, sont notés par les six premières lettres majuscules, soit A (pour 10D), B (pour 11D), C, D, E, F (pour 15D).

L'intérêt de cette notation est double (outre le fait déjà noté qu'elle utilise peu de chiffres pour une valeur donnée, moins encore que le décimal). D'une part, la conversion binaire-hexadécimal est très simple. Pour le comprendre, notons les valeurs binaires de tous les chiffres hexadécimaux :

0H:	0000B
1H:	0001B
2H:	0010B
3H:	0011B
4H:	0100B
5H:	0101B
6H:	0110B
7H:	0111B
8H:	1000B
9H:	1001B
AH:	1010B
BH:	1011B
CH:	1100B
DH:	1101B
EH:	1110B
FH:	1111B

Il en résulte cette règle simple : à un quartet binaire (4 chiffres binaires) correspond un et un seul chiffre hexadécimal. La règle de transformation est dès lors immédiate. Pour passer du binaire à l'hexadécimal, divisez votre nombre en quartets, en partant de la fin et en ajoutant des zéros devant si nécessaire. Puis transformez chaque quartet en un chiffre hexadécimal en vous aidant du tableau ci-dessus. Ainsi 1986, qui s'écrit 11111000010B, s'écrit encore : 0111 1100 0010B (on a ajouté un zéro devant), soit : 7C2H. CQFD. Inversement, pour passer d'un nombre hexadécimal au binaire, il faut convertir chaque chiffre en un quartet. Ainsi 2FAH s'écrit 0110 1111 1010B, soit encore 101111010B. Tout cela est très simple.

Le second avantage de cette notation hexadécimale est une conséquence de la structure des ordinateurs, qui l'a fait préférer à la notation octale : les registres des ordinateurs et leurs cases mémoire sont groupés par 8 ou 16 bits. Or 8 bits, cela fait exactement 2 quartets, donc deux chiffres en hexadécimal, et 16 bits, quatre chiffres. Dès lors, les adresses du Z80, qui s'écrivent sur 16 bits, sont toutes les combinaisons possibles de quatre chiffres hexadécimaux, donc de 0000H à FFFFH. De même les valeurs pouvant être prises par un registre 8 bits sont toutes les paires de chiffres hexadécimaux, de 00H à FFH (255).

COMPARAISON ENTRE NOMBRES, SIGNES

Pour comparer deux nombres écrits dans une même base, on applique la même règle que pour les nombres décimaux. Tout d'abord on compare le nombre de chiffres de chacun (non compris les zéros en tête éventuels). Celui qui en a le plus est le plus grand. Ainsi 2FD0H > A45H.

S'il y a autant de chiffres, on compare les deux premiers. Celui qui a le plus grand premier chiffre est le plus grand. Ainsi A34FH < C107H car A < C. Si les premiers chiffres sont égaux, on compare les seconds, etc. Si tous les chiffres sont égaux, les deux nombres le sont aussi !

Pour les nombres binaires, la comparaison entre nombres de chiffres est malaisée, du fait du grand nombre de chiffres qu'il y a souvent. De ce fait, on standardise les nombres binaires par octets, en rajoutant éventuellement des zéros devant. Ainsi 1011001B sera plutôt écrit 01011001B (huit bits). On applique alors la comparaison des chiffres.

On aura besoin, pour les nombres binaires, de numéroter les bits. On le fait en partant de la droite (le bit 0 est donc le dernier), de 0 à 7 pour les nombres à 8 bits, de 0 à 15 pour ceux à 16 bits.

Il est important de noter ici la façon dont sont codés les nombres négatifs. On déclare pour cela que le bit le plus fort (bit 7 pour les 8 bits, bit 15 pour les 16 bits) est le bit de signe : il vaut 0 pour les nombres positifs, et 1 pour les négatifs. Pour ces derniers, on en obtient la valeur absolue en complémentant tous les bits (changer les 0 en 1 et inversement), puis en ajoutant 1. Ainsi, FFH = 11111111B est négatif. Sa valeur absolue est obtenue en complémentant tous les bits (cela donne 00000000B), puis en ajoutant 1, ce qui donne 00000001B, soit 1. Donc FFH = -1.

On notera une ambiguïté, car FFH représente aussi la valeur 255. Suivant les cas, on dira que l'on aura affaire à des nombres signés (pour les opérations arithmétiques, ou pour tracer des traits), qui vont de -128 (80H) à +127 (7FH) pour les 8 bits et de -32768 (8000H) à 32767 (7FFFH) pour les 16 bits, ou au contraire non signés (adresses notamment), qui vont de 0 (00H) à 255 (FFH) pour les 8 bits et de 0 (0000H) à 65535 (FFFFH) pour les 16 bits.

Ces problèmes de signes sont parfois source d'erreurs, il convient d'y prêter attention.

Il est important de vous familiariser, si ce n'est pas encore le cas, avec ces notions de bases 2 et 16, afin de pouvoir utiliser et comprendre ces notations, qui sont d'un usage constant dans ce livre ; vous pouvez, pour vous aider, refaire quelques-unes des conversions données en exemple ci-dessus.

INVERSION SEIZE BITS DU Z80

Si vous ne le savez pas encore, notez ici une chose essentielle : lorsque le Z80 stocke en mémoire des adresses ou toute autre valeur sur 16 bits, il inverse les deux octets. Donc, si vous placez par exemple la valeur 3A5FH à l'adresse 1000H, vous trouverez en 1000H l'octet inférieur, soit 5FH, et en 1001H l'octet supérieur, soit 3AH.

De même, si vous souhaitez récupérer dans HL un nombre pointé par IX, il faudra faire

```
LD H, (IX+1) / LD L, (IX+0)
```

et non le contraire.

Cette inversion est une source fréquente d'erreurs, même quand on en a l'habitude. Pensez donc à toujours vérifier ce point en cas de problème.

■ LES ASSEMBLEURS DU COMMERCE

Il n'est pas en théorie absolument nécessaire d'acheter un Assembleur (on appelle ainsi, et l'on met une majuscule pour distinguer du "langage" assembleur, un programme qui se charge de convertir les mnémoniques en codes) pour pouvoir programmer en assembleur,

et plus particulièrement pour mettre en pratique les notions et les programmes de ce livre. Vous pouvez en effet rentrer directement les codes (un chargeur avec des explications est donné en fin de chapitre), ou utiliser les Assembleurs du 8080 fournis avec CP/M.

Toutefois cela n'est pas souhaitable à mon sens. D'abord, la programmation sans un Assembleur est rendue plus complexe et les risques d'erreur sont gros. Quant à utiliser les Assembleurs de 8080, alors que le microprocesseur des Amstrad est un Z80 largement supérieur, revient, me semble-t-il, à n'utiliser que la moitié des possibilités de votre machine, ce qui ne me paraît pas être vraiment une programmation efficace.

Bien qu'il vous soit possible d'agir autrement, nous supposons que vous avez fait l'acquisition d'un des deux Assembleurs du commerce prévus pour Amstrad (pour les trois versions). Ces deux Assembleurs portent les noms de ZEN et DAMS. Afin de faire jouer la concurrence, je vais vous en donner à présent les principaux avantages et inconvénients.

L'ASSEMBLEUR ZEN

Cet Assembleur est commercialisé par la société anglaise Kuma Computers Limited sous une version cassette seulement à ma connaissance, et pour une somme de 250 F.

Le fait que ZEN n'existe qu'en version cassette n'est pas très important, car il est très facile, après l'avoir placé une fois en mémoire avec un magnétophone, de le sauvegarder à nouveau sur disquette par SAVE "ZEN",B, &4000, &1940. Il sera utile de faire un petit chargeur BASIC qui placera le haut de mémoire en &3FFF (3FFFH), chargera le fichier binaire, y passera, voire verrouillera les majuscules, et programmera l'une des touches sur CALL &4000 pour pouvoir y revenir facilement si vous le souhaitez.

Après cette transformation, ZEN, qui utilise les vecteurs système (faites-en autant !) ne fera aucune difficulté pour tourner sur n'importe quel Amstrad, et en particulier pour sauvegarder et lire des programmes sur disquette.

L'un des principaux avantages de ZEN est en effet sa souplesse. Cela est heureux, car on peut ainsi compenser certains de ses défauts.

Le plus important se trouve au niveau de l'édition des lignes. Le concepteur a jugé utile de faire son propre éditeur. Celui-ci est si sommaire que pour changer le premier caractère d'une ligne, par exem-

ple, vous devez effacer la ligne tout entière ! Le programmeur n'a pas voulu utiliser l'éditeur BASIC, et c'est visiblement un tort. Ce problème est facilement réparable : il vous suffit de taper le petit programme assembleur ci-joint, de l'assembler et de sauvegarder la version de ZEN ainsi obtenue. Les éditions de lignes se feront alors avec l'éditeur BASIC. Une place suffisante n'ayant pas été prévue, vous prendrez garde que vos lignes ne dépassent pas 40 caractères, sinon des *artefacts* pourraient se produire.

```

1
2          ;=====
3          ;==      MODIFICATIONS DE ZEN      ==
4          ;(Utilisation de l'editeur BASIC)
5          ;=====
6
7          ORG 4310H
8          LOAD 4310H
9
10         EDIT:   EQU 0BD5EH          ;464:BD3AH,664:BD5BH
11
12         ;CHANGEMENT DE LA ROUTINE "N"
13
14 4310 00          NOP
15 4311 00          NOP
16 4312 37          SCF
17 4313 CDB446      CALL EDITER
18
19
20         ;CHANGEMENT DES ENTREES DE LIGNES
21
22         ORG 46B3H
23         LOAD 46B3H
24
25 46B3 B7          OR   A          ;CARRY A ZERO
26 46B4 215441      EDITER: LD   HL,4154H      ;ENTREES DE PHRASES
27 46B7 010000      LD   BC,0
28 46BA E5          PUSH HL
29 46BB 3007        JR   NC,SUITE
30 46BD 7E          BCLE: LD   A,(HL)          ;CHERCHER CODE CR
31 46BE 23          INC  HL
32 46BF FE0D        CP   13          ;CODE CR?
33 46C1 20FA        JR   NZ,BCLE          ;NON, POURSUIVRE
34 46C3 2B          DEC  HL          ;SE PLACER SUR CODE CR
35 46C4 3600        SUITE: LD  (HL),0          ;METTRE MARQUE FIN
36 46C6 E1          POP  HL
37 46C7 CD5EBD      CALL EDIT          ;EDITER LA PHRASE
38 46CA 7E          CHERCHE: LD  A,(HL)          ;CHERCHER 0 FINAL
39 46CB 0C          INC  C          ;LONGUEUR DANS C
40 46CC 23          INC  HL
41 46CD B7          OR   A
42 46CE 20FA        JR   NZ,CHERCHE
43 46D0 2B          DEC  HL
44 46D1 3A5441      LD  A,(4154H)          ;PREMIER CARAC. (ORDRE)
45 46D4 360D        LD  (HL),13          ;METTRE UN CR SUR LE 0
46 46D6 C9          RET
47
48
49         END

```

Mis à part ce détail, ZEN est un Assembleur tout à fait honorable. Voici ses principales caractéristiques.

ZEN se place en mémoire, comme DAMS, à l'adresse 4000H. Il occupe, avec la table des symboles qui le suit, la place jusqu'à 6000H (8 K), où se trouve le fichier texte. Dans la pratique, on ne pourra donc pas assembler de programme dans la zone 4000H-6800H environ. Cependant la directive ORG permet d'assembler un programme pour une adresse quelconque, à condition de le placer ailleurs au début.

L'assembleur en lui-même est bien conçu, assez rapide. L'assemblage se fait en deux passes. Il n'est pas possible de déplacer la table des symboles, ni de charger au cours d'un assemblage des morceaux de fichier, ce qui limite la longueur du fichier texte à la mémoire disponible (environ 5 K), donc à des programmes moyens. On peut regretter qu'il soit obligatoire de mettre une directive LOAD pour que le programme soit placé en mémoire.

Le fichier texte a l'avantage d'avoir des lignes numérotées. Mais l'absence de mode moniteur rend le déplacement dans ce fichier un peu plus difficile que sous DAMS. Les labels peuvent accepter n'importe quel caractère, mais ils doivent être suivis du caractère (:), et ne différencient pas majuscules et minuscules. La notation des nombres hexadécimaux est un peu lourde : outre le H final, il faut faire précéder ceux qui commencent par une lettre, comme B900H, par un zéro (ce qui donne 0B900H).

Un avantage important de cet Assembleur : il est possible de faire un assemblage à l'écran ou sur l'imprimante, avec à la fois les adresses, les codes et les lignes ; c'est la raison pour laquelle il a été choisi dans ce livre pour lister les programmes.

Les ordres, qui recouvrent de nombreuses possibilités, ne prennent qu'une seule lettre, suivie éventuellement d'un paramètre. Il n'est pas possible de taper plusieurs ordres à la fois comme sous CP/M. Parmi les possibilités offertes que ne possède pas DAMS, on retiendra des modifications très aisées de la mémoire, et la possibilité d'un catalogue de la disquette ou de la cassette comme en BASIC.

ZEN recouvre aussi un désassembleur, comparable à celui de DAMS, et assez bien conçu. On peut ne désassembler que huit lignes à la fois (exploration de la mémoire), ou bien toute une partie de la mémoire.

Le manuel est succinct, mais suffisant.

En résumé, avec la correction donnée par le programme ci-joint, ZEN est un bon Assembleur relativement peu coûteux mais sans pré-

tention, dont je fus surpris de constater qu'il avait parfois d'évidentes supériorités sur DAMS.

L'ASSEMBLEUR DAMS

DAMS (Désassembleur, Assembleur, Moniteur Symboliques) est édité par la société française Micro-application. Il est en vente sous trois versions différentes pour les trois CPC, à des prix semblables (395 F sur 6128). Néanmoins les possibilités sont les mêmes.

DAMS, contrairement à ZEN, qui est globalement moyen, a beaucoup de qualités, malheureusement compensées par un certain nombre de défauts parfois fort gênants.

Tout d'abord, je tiens à exprimer ma surprise et mon indignation sur un point précis. La publicité au dos de la boîte, tout comme le manuel, portent en toutes lettres que DAMS est entièrement relogeable ; le chargeur BASIC inclus est même prévu de telle sorte que vous puissiez lui commander de charger DAMS à une adresse nn quelconque. Or c'est *totalemt faux*, car si vous le chargez, ne serait-ce qu'en 4001H au lieu de 4000H, le logiciel ne tournera pas. En d'autres termes, *DAMS n'est absolument pas relogeable*, pas plus que ZEN, et doit obligatoirement être chargé à l'adresse 4000H (comme ZEN) ; d'ailleurs il est presque impossible de faire un programme relogeable avec un Z80 en l'absence d'instruction d'appel relatif.

Ce point réglé, examinons le reste.

DAMS se présente sous la forme de trois parties différentes, que nous allons examiner tour à tour.

A l'initialisation, DAMS se place en mode 2 et passe au mode moniteur. Dans ce mode, vous pouvez utiliser toutes sortes d'instructions. L'une d'entre elles est l'assemblage, sur lequel nous reviendrons. Les autres sont à peu près les mêmes que sous ZEN, avec des instructions du même type (sauf les réserves faites, sur lesquelles nous reviendrons aussi). Une possibilité supplémentaire assez intéressante est offerte : la commutation des ROM supérieure et inférieure.

L'instruction Llabel vous permet de passer au mode éditeur, à la ligne possédant le label donné comme paramètre (à la ligne courante s'il n'y en a pas). L'éditeur est un éditeur plein écran. Le déplacement entre les lignes est de ce fait plus aisé qu'avec ZEN. Pour l'édition des lignes elles-mêmes, là encore le concepteur a jugé utile de faire son propre éditeur de lignes. Celui-ci est moins mauvais que celui de ZEN, mais bien moins bon que celui du BASIC ; sur ce point,

DAMS est donc inférieur à la version corrigée de ZEN par le programme donné au paragraphe précédent.

La syntaxe des lignes est proche de celle de ZEN. La différence se trouve au niveau des labels : ils ne sont pas suivis de (:), et même n'acceptent aucun caractère non alphanumérique ; ils distinguent par contre majuscules et minuscules (mais cela ne sert à rien). Les lignes sans label doivent être entrées précédées d'un espace. Précisons un bon point ici : les codes opération (première partie des instructions) sont vérifiés et codés ; le fichier texte est donc un peu moins long que sous ZEN, mais les deux types sont du coup incompatibles.

Passons directement à la très grande qualité de ce logiciel : son mode trace. Celui-ci est accessible par la commande T. Grâce à cela, vous pouvez exécuter pas à pas toute une partie de votre programme, avec à chaque instant sous les yeux les valeurs des registres. Pour aller un peu plus vite, dans le cas de routines système par exemple, vous pouvez utiliser l'instruction R (exécuter au ralenti). Dans les deux cas, toutes les instructions sont vérifiées pour éviter tout plantage, et vous pouvez à chaque instant vous arrêter par l'instruction Q, afin par exemple de changer la valeur des registres avec l'instruction (.). Tout cela est fort bien fait. Ce qui est absolument remarquable, c'est que tous les paramètres du programme en cours sont conservés et actualisés, même la position du curseur, les couleurs, etc. (seul oubli : les fenêtres). Cela va même jusqu'à l'excès, car si votre programme ne tourne pas en mode 2, vous verrez apparaître à chaque instruction graphique des moirures dues au changement de mode sans effacement de l'écran, que seule la touche Q peut arrêter.

Cette commande trace est indiscutablement un outil très puissant pour la vérification des programmes.

Malheureusement, on a parfois l'impression qu'on lui a sacrifié certains points. Pas de catalogue de la disquette sans retour au BASIC, des problèmes à la sortie vers l'imprimante (surtout avec la DMP-1), des modifications de la mémoire difficiles (uniquement par un ou deux octets), etc.

Le concepteur a parfois péché par excès. Ainsi, il a trouvé absolument nécessaire de désactiver toutes les touches du pavé numérique, ce qui ne facilite évidemment pas la rentrée des nombres. Les nombres hexadécimaux sont notés précédés d'un caractère # (pourquoi pas la même notation que le BASIC, si l'on ne veut pas du H final ?). Les lignes ne sont pas numérotées. Donc pour accéder à un endroit précis du texte, il faut qu'il s'y trouve un label. Soit. Mais de plus, pour passer au label DEBUT, par exemple, il vous est possible

de taper LDEB au lieu de LDEBUT, ou encore LD (à condition qu'il soit le premier label commençant par D) ; il en résulte un défaut que rien ne corrige : vous ne pourrez pas accéder au label ENTRE par exemple, s'il est précédé dans le programme par un label ENTREE. La commande LENTRE vous placera sur ENTREE. Et comme le déplacement du curseur est assez long...

Au sujet de l'assembleur, il est possible de faire un assemblage par blocs (les morceaux sont stockés sur la disquette et la table des symboles est placée dans la RAM écran pendant l'assemblage), ce qui est heureux car DAMS est très long (quelque 9 K, sans la table des symboles). Par contre, impossible d'avoir un assemblage sur l'imprimante ou à l'écran. Donc, si vous souhaitez connaître l'adresse de telle ou telle instruction (pour tracer à partir de là par exemple), impossible, surtout s'il n'y a plus de table des symboles.

En résumé nous dirons de DAMS que certains points gagneraient à être améliorés. Mais son mode trace et son assemblage par blocs peuvent tout à fait justifier la différence de prix avec ZEN, tout particulièrement pour de longs programmes.

UTILISATION DU CHARGEUR BASIC

Si vous êtes économe et patient, vous pouvez utiliser le chargeur BASIC dont le listing est donné ci-dessous.

```

100 '===== CHARGEUR BASIC
110 MODE 1:MEMORY &9000
120 DEFINT a-z
130 WINDOW #0,1,29,1,25
140 WINDOW #2,30,39,1,25
150 INPUT "Adresse de depart ";ad0
160 PRINT
170 INPUT "Adresse de fin ";adf
180 '== entree des codes hexadecimaux
190 FOR ad=ad0 TO adf
200 PRINT HEX$(ad),:INPUT c$
210 IF LEN(c$)=0 GOTO 280
220 IF LEN(c$)>2 GOTO 590
230 GOSUB 440
240 IF x<0 GOTO 590
250 POKE ad,c
260 PRINT#2,HEX$(ad)+" "+HEX$(c,2)
270 s=s+c:p=p+c*(ad-ad0+1)
280 NEXT
290 '===== verification
300 CLS:PRINT "Somme: "s
310 PRINT"Produit: "p
320 INPUT "D'accord (O/N)";a$
330 IF a$<>"N" GOTO 390
340 CLS
350 FOR ad=ad0 TO adf
360 PRINT HEX$(ad)+" "+HEX$(PEEK(ad),2)
370 WHILE INKEY$="":WEND
380 NEXT
390 '===== sauver le programme

```

```

400 INPUT "Nom du programme ";a$
410 SAVE a$,b,ad0,(adf-ad0+1)
420 NEW:'fin du programme
430 '==== calcul de la valeur a poker
440 l=2:GOSUB 500
450 IF x<0 THEN RETURN ELSE c=18*x
460 l=1:GOSUB 500
470 c=c+x
480 RETURN
490 '== calcul du chiffre hexadecimal
500 IF l<>LEN(c$) THEN x=0:RETURN
510 x=ASC(c$)-48
520 IF x<0 THEN RETURN
530 c$=RIGHT$(c$,1)
540 IF x<10 THEN RETURN
550 x=x-7
560 IF x<10 OR x>15 THEN x=-1
570 RETURN
580 '==== erreur dans les entrees
590 PRINT"Erreur, recommencez..."
600 GOTO 200

```

L'utilisation en est très simple. Vous devez d'abord rentrer les adresses de début, puis de fin du programme (ces adresses sont données en hexadécimal en fin de listing pour les programmes de ce livre, n'oubliez donc pas le symbole &).

Vous pouvez ensuite entrer les codes un par un. Ces codes sont donnés avec les adresses dans la partie gauche du listing. Vous ne devez rentrer, par ligne, que les deux chiffres du code (avec des majuscules pour A à F), ou un seul pour les codes commençant par zéro. Vérifiez aussi que vous les rentrez bien aux bonnes adresses (vous serez aidé par le listing à droite de l'écran).

Si vous rencontrez un saut, dû à une directive *DEFS n*, vous devez passer *n* octets, en entrant une ligne vide en face des adresses correspondantes, jusqu'à ce que vous arriviez au prochain vrai code ou à la fin.

Arrivé à la fin, le programme vous donnera deux valeurs de contrôle "Somme" et "Produit". Comparez-les avec celles données en fin de listing. S'il y a égalité, appuyez alors sur une touche autre que N, et le programme sera sauvé immédiatement sur support magnétique.

En cas de différence, appuyez sur N. Le programme sera alors listé pas à pas pour que vous puissiez vérifier. Si vous trouvez une erreur, arrêtez-le et corrigez par un POKE. Lorsque vous serez au bout du programme, celui-ci sera sauvé normalement.

2. L'ASSEMBLEUR DU Z80

Avant de considérer les particularités intéressantes des Amstrad, il nous faut tout d'abord nous pencher sur le cœur même de la machine : son microprocesseur, le Z80. Nous verrons que ce microprocesseur possède de nombreuses particularités dont, malheureusement, on ne pense pas assez souvent à tirer profit. Naturellement, les possibilités du Z80 doivent être exploitées dans le cadre de la machine, et c'est dans cette optique que ce chapitre est conçu.

Enfin je tiens à rappeler qu'il n'est pas question ici de détailler toutes les instructions du Z80 : cela a déjà été fait par R. Zaks (voir bibliographie). Elles sont supposées déjà connues, en gros, du lecteur. Nous allons voir comment on peut en tirer parti à fond.

■ LE Z80. LES REGISTRES

LE Z80

Vous le savez déjà, le Z80 est un microprocesseur 8 bits assez répandu et dont les possibilités sont plus importantes que celles de son concurrent, présent notamment sur Apple II et Oric, le 6502. C'est probablement la raison pour laquelle il fut choisi par Amstrad comme cœur de ses machines.

Sur tous les Amstrad, il "tourne" à 4 MHz (4 mégahertz), c'est-à-dire que l'horloge interne lui envoie 4 millions d'impulsions (méga signifie million) en une seconde. La durée la plus courte que puisse donc utiliser le Z80 est d'un quart de milliardième de seconde, durée souvent appelée temps-machine, ou période, que nous noterons simplement T.

De ce fait, la durée d'une instruction se mesure commodément en périodes. La liste complète des instructions avec leur durée figure dans l'Annexe A. Nous pouvons y voir que les instructions les plus rapides (comme *NOP*), ont une durée de 4 T, soit un milliardième de seconde, les plus longues (comme *RES b,(IX+n)*), de 23 T soit 5.75 milliardièmes de seconde, ce qui fait évidemment une différence considérable, sur laquelle nous reviendrons plus loin.

L'expression de microprocesseur 8 bits signifie que les registres du Z80 peuvent contenir 8 informations élémentaires indépendantes (bit), soit l'équivalent d'un nombre binaire à huit chiffres. Un tel nombre est donc compris entre 00000000 (0) et 11111111 (255 en décimal, FFH en hexadécimal). Ces bits sont numérotés à partir de la *droite*

de 0 à 7. On les notera souvent b_0 , b_1 , etc. Rappelons que la valeur d'un nombre binaire en décimal est donnée par la formule :

$$b_7b_6b_5b_4b_3b_2b_1b_0 = b_0 * 2^0 + b_1 * 2^1 + b_2 * 2^2 + b_3 * 2^3 + b_4 * 2^4 + b_5 * 2^5 + b_6 * 2^6 + b_7 * 2^7.$$

On se référera sur ce sujet au Chapitre 1.

Toutefois, le Z80 possède une particularité intéressante : l'accouplement de ses registres 8 bits, qui fait que de nombreuses instructions s'exécutent en réalité sur 16 bits (soit un nombre compris entre 0 et FFFFH = 65536). Il existe en outre des registres 16 bits.

Comme la plupart des microprocesseurs 8 bits, le Z80 peut adresser 64 K (64 kilo-octets). Cela signifie que les cases mémoire qu'il peut explorer directement ont un numéro compris entre 0 et FFFFH (nombre sur 16 bits), soit 64 K, car un kilo-octet représente $2^{10} = 1\ 024$ octets, donc 64 K représentent $2^6 * 2^{10} = 2^{16}$ octets. Or il y a effectivement 2^{16} nombres compris entre 0 et FFFFH = $2^{16} - 1$. Un tel nombre est appelé un *mot*, par opposition à un octet.

En fait, nous verrons que sur les Amstrad CPC la mémoire varie entre 96 et 176 K. Mais seuls 64 d'entre eux sont accessibles à la fois.

Au total, le Z80 lui-même possède 18 registres 8 bits, dont 16 sont appariés (en 8 paires donc), plus 4 registres 16 bits. Nous les classerons en groupes ; le groupe 1 contient 4 paires, AF, BC, DE, HL. Le groupe 1 bis contient leurs doubles AF', BC', DE', HL'. Le groupe 2 contient IX et IY. La base est constituée du reste, à savoir R, I, PC et SP. Cette classification commode sera souvent reprise.

Rappelons que le groupe 1 et le groupe 1 bis ne peuvent être utilisés simultanément. On dit souvent qu'ils sont sur une sorte de tableau tournant : lorsque l'une des faces est visible, l'autre est cachée. On fait tourner ce tableau par les instructions *EX AF*, *AF'* et *EXX*.

LES REGISTRES

La base

PC

Le compteur ordinal (*Program Counter*) est un registre 16 bits qui pointe sur l'adresse mémoire de la prochaine instruction à exécuter.

Cette adresse peut être modifiée par un appel (*CALL*), un retour (*RET*) ou un saut (*JUMP*). Ainsi l'instruction *JP nn* consiste simplement à charger *nn* dans le PC. Dès qu'une instruction est exécutée, le processeur va chercher en mémoire l'instruction pointée par le PC, incrémente ce dernier et exécute, et ainsi de suite sauf ordre contraire (*HALT*).

SP

Le pointeur de pile (*Stack Pointer*), comme son nom l'indique, pointe sur la pile. Nous reparlerons de cette dernière dans un sous-paragraphe spécial, car elle est absolument fondamentale. Précisons simplement que le SP ne peut être chargé qu'en mémoire, ou à partir de celle-ci (comme *LD SP,(nn)*), ou à partir de valeurs directes (comme *LD SP,nn*), mais aucunement à partir des registres du Z80 : il n'y a pas d'instructions comme *LD SP,(HL)* ou *LD SP,IX*.

R et I

Deux registres de peu d'intérêt. R est le registre de rafraîchissement de la mémoire vive (il peut servir de générateur de nombres aléatoires, car sa valeur change très rapidement), et I celui des interruptions. Il est peu prudent de chercher à les modifier. Pour plus de précisions, voyez R. Zaks.

Le groupe 1

F

Le registre des flags est très spécialisé. Six de ses bits sont des flags (indicateurs valant 0 ou 1), les deux autres (bits 3 et 5) sont toujours nuls. Nous reviendrons sur ces six flags et leur utilisation. Pour ce qui est de F, sa spécialisation fait qu'il n'est concerné seul par aucune instruction en tant que registre. Il ne peut être globalement modifié que par *POP AF*.

A

L'accumulateur A est le registre 8 bits le plus fréquemment utilisé. De fait, toutes les opérations d'arithmétique 8 bits (*ADD*, *SUB*, *SBC*, ...) et toutes les opérations logiques (*AND*, *OR*, *XOR*, ...) passent par

lui. Ainsi il est impossible d'additionner B et D sans passer par A, qui est toujours l'opérateur. C'est donc le registre qui est concerné par le plus grand nombre d'instructions.

Pour des raisons techniques, il n'est pas possible de placer A seul sur la pile (on n'empile que les registres 16 bits). Il est donc alors associé à F.

AF

Nous avons déjà parlé de A et F séparément. Réunis, ils forment un registre double assez particulier, car seules trois instructions le concernent directement : *POP AF*, *PUSH AF* (dépilement et empilement), et *EX AF,AF'* dont nous avons déjà parlé. Cela s'explique par le fait, entre autres, que F ayant ses bits 3 et 5 toujours nuls, AF ne peut prendre n'importe quelle valeur entre 0 et FFFFH. L'utiliser comme pointeur, par exemple, serait donc vain.

BC

BC est le deuxième des registres doubles du groupe 1. Il se distingue, ainsi que ses deux composantes B et C, par deux points : BC est toujours utilisé pour l'adressage des périphériques (voir instructions *OUT(C),C*, *OUTI*, etc.). D'autre part B et BC sont les compteurs 8 et 16 bits privilégiés (voir instructions *DJNZ n*, *LDIR*, etc.).

DE

DE est le troisième des registres doubles du groupe 1. Peu de particularités. Il sert essentiellement comme deuxième accumulateur 16 bits (après HL), notamment dans les instructions de type *LDI*, etc., et dans les routines (voir celles-ci).

HL

Quatrième des registres doubles du groupe 1, HL n'en est pas pour autant le moins important, bien au contraire. Il est en effet un véritable accumulateur sur 16 bits, permettant des additions, soustractions diverses avec les autres registres 16 bits (mais pas d'opérations logiques). Ce double registre se distingue également comme pointeur : les opérations sur (HL) sont en effet nombreuses (y compris *RLD* et *RRD* qui utilisent (HL) comme opérande). Notons enfin que certaines facilités sont données à ce registre double pour le conserver par les instructions *EX DE,HL* et *EX (SP),HL*.

Le groupe 1 bis

Les registres du groupe 1 bis sont accessibles par deux instructions : *EX AF,AF'* et *EXX*. La deuxième échange les positions de BC et BC', de DE et DE' et de HL et HL' tout à la fois.

Ce groupe 1 bis représente évidemment une tentante table de stock pour les registres, malheureusement les concepteurs de l'Amstrad y ont pensé les premiers. En particulier BC' est utilisé pour stocker l'état des ROM et RAM : *il ne doit jamais être modifié !* De plus, tout appel à une routine du système d'exploitation (appel difficile à éviter...) et toute interruption du système (fréquents) modifieront tous les registres du groupe 1 bis. En foi de quoi, il convient donc de se servir au minimum du groupe 1 bis sur Amstrad.

Le groupe 2

IX et IY

IX et IY sont deux registres doubles qui ne peuvent être partagés en registres simples comme HL, DE, BC ou AF. Ils sont exactement semblables à tous points de vue. Ils servent de pointeurs de variables de la mémoire, car ils possèdent de nombreuses instructions relatives à $(IX+n)$ (où n est un nombre compris entre 0 et 255), c'est-à-dire à l'adresse pointée par IX plus le nombre n . Chacun de ces registres doubles permet donc le contrôle direct de 256 octets de la mémoire, ce qui est évidemment une particularité essentielle. Toutefois il n'existe pas d'opérations du même type sur $(IX+A)$ par exemple.

Les flags

Après une opération, il est souvent essentiel de connaître certains détails sur le résultat, par exemple son signe. Ces résultats sont stockés dans les bits du registre F, qui constituent six flags. Nous allons à présent détailler leur rôle et leur utilité (très inégale). Ces flags sont dans le présent ouvrage notés par une minuscule italique.

Ces flags sont également décrits avec plus de précision par R. Zaks, mais il m'a semblé utile de redonner certains détails, surtout sur les principaux.

s

Le flag de signe *s* est le bit 7 de F. Comme son nom l'indique, ce flag est positionné en fonction du signe du résultat d'une opération (il est mis si ce signe est négatif). Il peut être testé directement (instructions type RET M et RET P, où M signifie "moins" ($s = 1$) et P "plus").

z

Le flag *z* (zéro), bit 6 de F, est mis lorsque le résultat d'une opération est nul. Il peut être testé directement (instructions type RET Z et RET NZ, où Z signifie que $z = 1$ (résultat nul), et NZ l'inverse).

h

Le flag *h*, bit 4 de F, est positionné quand le quartet inférieur de A déborde dans l'autre au cours d'une opération, particularité utilisée par le Z80 lui-même pour l'instruction DAA. En dehors de cela, il est souvent modifié aléatoirement. Son intérêt pour le programmeur étant faible, il ne peut être testé directement.

p/v

Ce flag, bit 2 de F, remplit un double rôle en fonction des instructions. Pour certaines, il indique un dépassement en fonction du signe. Pour d'autres, il indique la parité du résultat (nombre pair ou impair de bits à 1). Il peut être testé directement (instructions de type RET PE [parité impaire, flag mis] ou RET PO).

n

Ce flag, bit 1 de F, est, comme *h*, utilisé par le Z80 pour DAA. Il ne présente aucun intérêt et ne peut être testé directement.

c

La retenue *c* (*carry*), bit 0 de F, est le plus important de tous ces flags. Elle est en principe positionnée, comme son nom l'indique, lorsqu'une opération arithmétique comme une addition provoque une retenue par dépassement de capacité. Mais la retenue sert également de bit supplémentaire aux divers registres, lors d'opérations de décalage, ou de témoin : c'est le cas par exemple dans les com-

paraisons. De tous les flags, la retenue est celui qui est le plus privilégié, car elle possède un jeu propre d'instructions : *SCF* positionne la retenue, *CCF* l'inverse, et *ORA*, par exemple, la met à zéro (sans changer A). Naturellement la retenue peut être testée (*RET C*, *RET NC*). Notons en outre que la retenue sert souvent d'indicateur sur la réussite d'une opération par le système (voyez notamment les routines de scrutation du clavier, ou de cassette/disquette).

■ LES INSTRUCTIONS

Le Z80 possède environ 800 instructions, soit environ trois fois plus que le 8080 avec lequel il avait été conçu pour être compatible à l'origine. Pour parvenir à un tel jeu d'instructions, les concepteurs du Z80 ont dû avoir recours à une astuce pour pouvoir coder ces instructions, car en principe une instruction, sur un microprocesseur 8 bits, est codée sur un seul octet. Pour pouvoir faire mieux, certaines instructions du Z80 sont préfixées, c'est-à-dire précédées d'un octet spécial. Cet octet peut être CBH (préfixe des opérations sur les bis et des rotations), EDH (préfixe des instructions d'entrée-sortie, de l'arithmétique 16 bits, de *LDI*, etc.), DDH (toutes instructions faisant intervenir IX) ou FDH (toutes instructions faisant intervenir IY). En fin de compte, certaines instructions font intervenir quatre octets, opérandes compris.

Les opérandes eux-mêmes, s'ils sont extérieurs à l'instruction, peuvent tenir deux octets (mots). Aucune instruction n'admet plus de deux opérandes. Celles qui le devraient utilisent alors les registres (comme *LDIR* par exemple).

Notez, c'est très important, que c'est essentiellement la longueur en octets d'une instruction et ses opérandes qui en déterminent la durée. C'est absolument fondamental pour l'optimisation en durée et en longueur de vos programmes (voir l'Annexe A et le Chapitre 5).

Nous allons à présent donner quelques détails sur les opérandes et instructions. Vous trouverez plus de détails chez R. Zaks.

LES OPÉRANDES

Les opérandes peuvent être de nombreux types, et nous adopterons une notation pour chacun d'eux.

- Premier type, les valeurs directes : ce sont des nombres sur un ou deux octets qui suivent généralement l'instruction. Nous les noterons n ou nn suivant le nombre d'octets. Par exemple : *JP nn*. Ce type comprend les contenus de valeurs directes. Rappelons que "contenu de " se note dans la syntaxe de l'assembleur du Z80 par des parenthèses. Ainsi *LD A,(nn)* signifie : charger A avec le contenu de l'adresse mémoire n° nn . Notons que ces parenthèses entourent toujours une valeur sur deux octets (puisque'il s'agit d'une adresse mémoire).
- Deuxième type, les registres 8 bits (que l'on notera r). F est exclu, il ne peut s'agir que de A, B, C, D, E, H, L, ou parfois R ou I. Cet opérande est toujours intérieur au code de l'instruction (c'est-à-dire que le modifier exige de modifier le code). Les parenthèses ne se verront jamais autour d'un tel opérande (un seul octet). Notons que A est souvent un opérande implicite. *OR B*, par exemple, admet deux opérandes : A et B ; on le note ainsi plutôt que *OR A,B* par souci de simplicité.
- Troisième type, les registres 16 bits (que l'on notera rs) : AF (rare), BC, DE, HL, IX, IY, SP. Le Z80 permet beaucoup plus d'instructions sur 16 bits qu'il n'est courant chez un microprocesseur 8 bits. Notons que le stockage des valeurs 16 bits se fait en inversant les deux octets (octet faible d'abord). Ainsi, après l'instruction *LD HL,(nn)*, L contiendra l'octet n° nn , et H l'octet n° $nn+1$.
- Quatrième type, les bits : *RES 4,r*, par exemple, met à zéro le quatrième bit du registre r . Ce type comprend aussi les flags (bits de F), et notamment les conditions des sauts et des appels.
- Un peu à part enfin, $(IX+n)$ est en fait comme un registre 8 bits (mêmes instructions), mais c'est un octet de la mémoire vive. Il s'agit là d'une possibilité très intéressante du Z80, malgré sa relative lenteur. De plus, n est un octet qui suit l'instruction.

LES INSTRUCTIONS SANS OPÉRANDE

Il existe peu de véritables instructions sans opérande. Elles sont généralement très rapides. Il s'agit, outre *NOP* et *HALT*, des différents *RET* non conditionnels, de *DI* et de *EI*.

Il existe par contre de nombreuses fausses instructions sans opérande. Elles n'apparaissent ainsi que par leur nom. Ainsi *NEG* aurait

pu aussi bien s'appeler *NEG A*, et de même pour certaines rotations comme *RRA*. Citons aussi *SCF* (*Set Carry Flag* : mettre le flag de retenue) et *CCF*, qui opèrent sur la retenue.

Citons aussi les instructions de type *LDI* qui opèrent en fait sur *BC*, *DE* et *HL*, et *EXX*.

LES INSTRUCTIONS A UN OPÉRANDE

Outre les sauts et appels non conditionnels, les restarts et les retours conditionnels, les rotations, décalages, incréments et décréments (toujours sur des registres 8 bits, ou sur des octets pointés par *HL* ou *IX+n*, sauf pour les incréments et décréments qui peuvent admettre des opérandes 16 bits) sont l'essentiel des instructions n'admettant qu'un opérande. Leur rapidité dépend essentiellement de l'opérande.

Citons également les empilements et dépilements qui admettent un opérande 16 bits obligatoirement (*AF*, *BC*, *DE* ou *HL*).

LES INSTRUCTIONS A DEUX OPÉRANDES

Ce sont les plus nombreuses. Elles comprennent :

- Les appels et sauts conditionnels (voir Chapitre 3).
- Les opérations sur les bits, *BIT*, *RES*, *SET* qui admettent un numéro de bit comme premier opérande et un registre 8 bits ou un octet pointé comme deuxième. Suivant l'opérande, leur durée est très variable (8 T pour *r*, 23 T pour *(IX+n)*). Notez que *BIT* modifie tous les flags, hormis la retenue, mais que *RES* et *SET* n'en modifient aucun.
- Les opérations logiques et arithmétiques 8 bits, dont le premier opérande est toujours *A*, qu'il soit cité ou non ; on peut d'ailleurs se demander pourquoi la syntaxe de l'assembleur demande d'écrire *ADD A,B* mais *SUB B* ? Le deuxième opérande est un registre 8 bits ou un octet pointé. Rappelons qu'il existe quatre opérations arithmétiques (*ADD*, *ADC*, *SUB*, *SBC*) et quatre logiques (*OR*, *AND*, *XOR*, *CP*). Toutes ces opérations modifient tous les flags ; *XOR*, *OR* et *AND* mettent la retenue à zéro (d'où l'utilité de *OR A* et *AND A* qui servent à annuler *c*).

- Les instructions d'arithmétique 16 bits dont le premier opérande est toujours HL, IX ou IY et le second un registre 16 bits quelconque (hormis AF et PC). Les opérations arithmétiques sont les mêmes que pour 8 bits. Notons que ADD ne modifie dans ce cas que la retenue, et pas les autres flags comme le font ces opérations.
- L'instruction LD (LoaD : charger) enfin, qui peut admettre des opérandes très divers. En général, sa syntaxe est : *LD destination, source*. La destination et la source doivent être du même format (8 ou 16 bits) ; seul (nn) peut être, selon les cas, une quantité sur 8 ou 16 bits. Sur 8 bits, l'un au moins des opérandes doit être un registre r (différent de R, I ou F) ; l'autre peut alors être un registre 8 bits (sauf F), ou (HL), (IX+n), ou une valeur directe n ; si l'un des opérandes est A, l'autre peut aussi être (nn), ou (BC), (DE). Sur 16 bits, l'un au moins des opérandes doit être (nn), SP (toujours en premier opérande), ou nn (toujours en deuxième). L'autre peut être un registre 16 bits (sauf PC et AF) ou (nn). Dans ce dernier cas, l'instruction est plus rapide si elle se fait sur HL. Notez une chose *essentielle* sur cette instruction : hormis *LD A,R* et *LD A,I*, *LD ne modifie jamais les flags*. De plus, il existe des instructions "inutiles" comme *LD A,A* qui ne changent rien.
- Enfin, très particulières mais très commodes, les instructions d'échange : outre *EX AF,AF'* et *EXX* qui sont à part, il reste *EX DE,HL* d'usage fréquent, et *EX (SP),rs* où rs est HL, IX ou IY, instruction qui permet d'échanger le contenu d'un registre avec la dernière valeur placée sur la pile. Ces instructions ne modifient pas les flags (sauf *EX AF, AF'* évidemment).

■ LA PILE ET LES SAUTS

LA PILE

Il est fondamental de comprendre comment fonctionne la pile. La comparaison la plus claire, je pense, est celle d'une pile d'assiettes : lorsqu'on place une assiette en haut, c'est elle qui sera retirée la première au prochain retrait ; on dit qu'il s'agit d'une structure LIFO (*Last In, First Out* : dernier entré, premier sorti). Mais il ne faut pas oublier trois choses très importantes, et qui sont souvent source d'erreurs.

1. Le registre SP pointe sur le premier octet (octet de poids faible) du dernier rentré, soit. Mais lorsqu'on fait un empilement (par *PUSH* par exemple), le SP est *décrémenté* de 2. En fait la pile se trouve tête en bas : le sommet est vers le bas de la mémoire, la base vers le haut. Donc si vous voulez l'emplacement du dixième nombre empilé (dixième en partant de la fin), il faut additionner 20 (deux fois 10) au SP.
2. La pile se trouve en RAM. En particulier, elle n'est pas à l'abri de modifications possibles de la RAM. L'erreur classique en la matière consiste à charger un fichier en plein sur la pile, ou pire, à la placer dans la mémoire écran, puis d'effacer ce dernier ! Combinaison de cette erreur et de la précédente : placer le SP à 0, en s'imaginant que la pile va monter dans la mémoire inférieure... alors qu'elle descend ($0 = \text{FFFEH} + 2$) dans la mémoire écran. Inversement, la pile peut aller détruire juste l'endroit où vous avez placé vos paramètres essentiels... D'une façon générale, il faut trouver pour la pile un endroit bien séparé. Le BASIC la place avec raison juste en dessous de l'écran par *LD SP,C000H* (rappelez-vous : elle descend !). Il est conseillé de l'y laisser, surtout pour de petits programmes. Dans le même ordre d'idées, évitez les *PUSH* à répétition, vous risquez de déborder de la zone allouée à la pile. Si vous faites attention, votre pile ne devrait jamais dépasser une cinquantaine d'octets, quelle que soit la longueur de votre programme.
3. Les adresses de retour des appels (par *CALL* et *RST*) sont empilées. C'est là une source d'erreurs insidieuse et fréquente, pas toujours très facile à éviter. De toute façon, la transmission de paramètres à une subroutine ne peut se faire en les plaçant sur la pile, sauf si vous avez prévu de dépiler l'adresse de retour dans la routine.

Pour éviter des erreurs fréquentes (malgré de nombreuses précautions) et surtout graves (car un changement de la pile a toutes les chances de planter le programme au prochain *RET*), rappelons aussi les instructions qui modifient le SP : outre *INC SP*, *DEC SP*, *LD SP,nn*, *LD SP,rs*, *LD SP,(nn)*, il y a aussi tous les *CALL*, tous les *RET*, les *RST*, et évidemment les *POP* et *PUSH*. Par contre, les *JP*, les *JR* et *EX (SP),rs* ne modifient jamais le SP. D'une façon générale, il est important, au moment où vous programmez, que vous ayez en tête la position de la pile et la valeur des derniers octets empilés. Cela évitera une autre erreur également très fréquente, et extrêmement pernicieuse : une

routine où se trouvent plus d'empilements que de dépilements, ou le contraire ; le retour risque alors de donner des effets inattendus...

LES SAUTS

Il existe deux sortes de sauts : les sauts relatifs (*JR*) et les sauts absolus (*JP*).

Ces derniers consistent simplement à charger le PC avec un nombre *nn* sur deux octets qui est l'adresse de saut souhaitée. Une telle instruction prend trois octets en mémoire : un pour l'instruction et deux pour *nn*.

Les sauts relatifs ont l'avantage de ne prendre que deux octets en mémoire, d'où un gain de place qui peut être très utile. Leur paramètre n'est pas un nombre sur deux octets, mais sur un seul qui est additionné au PC en tenant compte de son signe, ce qui permet un saut en arrière de 126 octets au maximum et de 129 en avant. Il en résulte que seules les adresses proches de celle de l'instruction courante peuvent être atteintes. De plus, bien que plus court en octets, ce type de saut est plus long à exécuter (du fait de l'addition nécessaire). Le seul avantage est donc sa place mémoire réduite ; mais cet avantage suffit pour en justifier une utilisation maximale. Notons encore, au sujet des sauts relatifs, leur syntaxe en assembleur. Vous pouvez par exemple écrire : *JR + 12* ; mais cela exige que vous comptiez le nombre d'octets qui va suivre, ce qui n'est guère simple (surtout quand certaines instructions font plusieurs octets) ; de ce fait, on écrit en général *JR adresse*, et c'est l'assembleur qui se charge de calculer le déplacement relatif correspondant.

Avant de passer aux appels de sous-routines, notez que ces sauts et ces appels peuvent être conditionnels, c'est-à-dire exécutables seulement si un flag est positionné correctement. Par exemple, un *JP Z, nn* ne provoquera un saut que si le flag *z* est mis. Dans le cas contraire, l'instruction est ignorée. Les conditions possibles sont pour *JP* : *Z, NZ, C, NC, PO, PE, P, M* (voir description des flags), soit le test de quatre flags. Pour *JR*, seules les quatre premières conditions sont acceptables, soit le test de deux flags.

Enfin, un saut peut se faire à une adresse contenue dans un registre : (*HL*), (*IX*), (*IY*). Ces sauts sont alors plus rapides et prennent moins d'octets en mémoire (mais ce sont toujours des sauts absolus).

En résumé, le saut absolu présente plus de possibilités, mais on ne l'utilisera que lorsqu'on ne pourra pas faire autrement, ou pour une

sévère optimisation en durée. Dans le cas contraire, on cherchera à minimiser l'encombrement du programme en utilisant les sauts relatifs.

LES APPELS DE SOUS-PROGRAMMES

L'instruction *CALL* est l'une des plus souvent utilisées en assembleur, car elle permet d'appeler ce qu'on nomme en général des sous-routines, c'est-à-dire de petits sous-programmes qui sont susceptibles de servir plusieurs fois dans un même programme général. Nous reviendrons sur la manière de programmer ces sous-routines dans le Chapitre 5, mais expliquons tout d'abord leur structure générale.

La séquence usuelle pour un appel de sous-routine est la suivante : *CALL routine*, puis ailleurs dans le programme : *routine:.../RET*. Lorsque le Z80 rencontre l'instruction *CALL nn*, il empile l'adresse de retour (c'est-à-dire l'adresse de l'instruction *CALL + 3*), et place dans le PC le nombre *nn*. Lorsque le Z80 rencontre l'instruction *RET*, il dépile simplement la valeur du haut de la pile dans le PC (comme s'il faisait un *POP PC*). Si vous n'avez pas manipulé la pile inconsidérément dans la sous-routine, la valeur dépilée par le *RET* est justement la valeur empilée par le *CALL*, et le programme reprendra son cours normal après l'exécution de la sous-routine.

Ce scénario classique peut néanmoins être modifié. Il peut l'être tout d'abord par erreur, comme nous l'avons vu précédemment, et cela du fait que la pile contient à la fois des valeurs que vous y avez sauvées et les adresses de retour (il y en a souvent plusieurs, car les appels imbriqués sont très courants). Comme ces dernières ont été empilées en quelque sorte malgré vous, vous risquez de les oublier.

Mais si vous y pensez, il ne faut pas craindre de changer le cours normal des choses. Vous disposez d'instructions qui vont vous aider, s'il est essentiel que vous transmettiez des paramètres à la routine par la pile. Supposons par exemple qu'à l'intérieur de votre routine vous deviez récupérer deux nombres dans HL et DE, empilés avant l'appel. Que faire ? Vous avez trois possibilités :

- Tout d'abord, la première qui vient à l'esprit, et qui est la plus mauvaise : décrémenter deux fois le SP ; il pointe alors sur les valeurs sauvées ; il ne vous reste plus qu'à faire deux *POP*, puis à restituer le SP dans sa valeur initiale. Pourquoi n'est-ce pas une bonne idée ? Pour trois raisons : en premier lieu, les décrémentations et incré-

mentations de SP nécessaires sont nombreuses (huit dans notre exemple), cela finit par prendre beaucoup de temps, ce qui est spécialement mal venu dans une routine utilisée plusieurs fois ; en deuxième lieu, si vous faites d'autres appels ou empilements à l'intérieur de votre sous-routine, vous risquez de détruire l'adresse de retour ; enfin et surtout, après votre *RET*, les deux valeurs seront toujours sur la pile, ce qui n'est généralement pas souhaitable. Cette méthode sera donc utilisée seulement si l'on désire que les deux valeurs soient encore sur la pile après l'appel de la sous-routine, circonstance en fait exceptionnelle (et il existe alors de meilleures méthodes que l'empilement).

- Deuxième méthode, très commode si l'un des registres HL, IX ou IY est inutilisé. C'est souvent le cas de IY. Dans ce cas, dès le début de la sous-routine faites *POP IY*. L'adresse de la sous-routine est alors conservée dans IY. La sous-routine se terminera alors par *JP (IY)*, ordre commode s'il en est. La pile n'étant plus concernée après le dépilement, vous pouvez en faire ce que vous souhaitez.
- Troisième possibilité, si tous vos registres sont modifiables ou utilisés (par exemple si votre sous-routine appelle elle-même une routine de l'arithmétique à virgule flottante), utiliser les instructions d'échange. Dans notre exemple vous ferez : *POP HL/ POP DE/ EX (SP),HL*. La dernière instruction remplace l'adresse de retour sur la pile tout en plaçant dans HL la donnée à récupérer. Si les données étaient sauvegardées dans l'ordre inverse, il faudrait ajouter à cette séquence un *EX DE,HL*. Naturellement, si BC doit intervenir, tout peut se compliquer... à la limite, il vous faudra peut-être avoir recours à une table de stock de deux octets :

PUSH HL/ EX (SP),HL/ LD (stock),HL/ POP HL/ ... dépilements nécessaires .../ PUSH HL/ LD HL,(stock)/ EX (SP),HL

Là encore, l'adresse de retour se retrouve en fin de compte en haut de pile, prête pour un *RET*, et tous les registres sont conservés, même la valeur initiale de HL. On voit sur cet exemple que cette instruction *EX (SP),HL* est extrêmement puissante, car en réalité la valeur de HL n'a nullement été affectée par toutes ces manipulations.

En conclusion, nous voyons que l'appel de sous-routines, s'il peut être problématique, peut aussi réserver des surprises positives, du fait même de sa complexité.

Avant d'en terminer avec les appels de sous-routines, rappelons que les *CALL* et les *RET* peuvent être conditionnels (les conditions sont les mêmes que pour *JP*). D'autre part, il n'existe pas d'appels relatifs : les appels prennent donc tous trois octets de la mémoire.

■ INSTRUCTIONS MAL AIMÉES ET INSTRUCTIONS CRÉÉES

INSTRUCTIONS MAL AIMÉES

On ne pense pas toujours à utiliser toutes les instructions du Z80. Pourtant, toutes ou presque peuvent servir. Je vous propose d'en voir quelques-unes dès à présent. Nous en verrons encore au Chapitre 5.

Tout d'abord les plus connues (mais pas forcément de tous) : les opérations logiques de A sur lui-même. *ORA* et *ANDA* ont le même effet : elles ne modifient pas l'accumulateur, donc semblent inutiles. Rien de plus faux, elles positionnent les flags : la retenue est annulée (c'est pourquoi il n'existe pas de *RCF* : *Reset Carry Flag*, ces ordres conviennent) et les autres flags sont positionnés en fonction de A ; vous pouvez ainsi tester son signe ou sa nullité, même après un *LD A,r*, qui ne positionne pas les flags. *XOR A* met A et tous les flags à zéro. Il est plus rapide et moins encombrant en mémoire que *LD A,0*. Cette dernière instruction ne doit donc être utilisée que si l'on désire annuler A sans modifier les flags.

Dans le même ordre d'idées, les incréments et décréments peuvent rendre des services. Ainsi, pour mettre 1 dans A (ou -1) en positionnant les flags, *XOR A/INC A* est plus rapide et moins encombrant que *LD A,1/ OR A*. Le tout est d'y penser...

Plus subtiles, les additions et soustractions avec retenues. Elles permettent soit des gains de temps, soit même de véritables simplifications. Par exemple, si vous avez dans HL une valeur nn et que vous souhaitez y placer $2*nn + 1$, vous pouvez faire *ADD HL,HL/ INC HL*, mais c'est moins rapide que *SCF/ ADC HL, HL*, qui donne le même résultat. Autre exemple : si vous souhaitez positionner A en fonction de la retenue (c'est fréquent), par exemple en donnant A=0 si la retenue est nulle et A=FFH (= -1) dans le cas contraire, faites *SBC A,A*, et c'est tout ; si vous préférez A=1 si la retenue est nulle et 0 sinon, faites *SBC A,A/ INC A*, et non *CCF/ LD A,0/ RLA* qui est plus long en octets et en temps ; si vous préférez A=1 si la retenue est nulle et -1 sinon (FFH), c'est un soupçon plus compliqué, mais vous avez

le choix entre *SBC A,A/ADD A,A/ INC A* ou *SBC A,A/ SCF/ ADC A,A* dont les durées sont les mêmes (je vous engage à bien vérifier que le bon résultat est ainsi obtenu...). Ce genre de tour est tout de même plus élégant que de lourdes boucles du genre : *LD A,1/ JR NC,suite/ LD A,FFH/ suite:...* plus longues (20 T en moyenne, contre 12 pour la séquence précédente) et plus encombrantes (6 octets contre 3 !). Vous pourrez souvent éviter ce genre de petits sauts en cherchant soigneusement la bonne opération.

Dans le même genre, n'oubliez pas non plus les rotations : il en existe beaucoup dans le Z80. Donc si vous voulez diviser un registre r (B par exemple) par 2, faites *SRL B*, le reste est alors dans la retenue (pour le récupérer dans A, faites *LD A,0/ ADC A,A*). Pour le multiplier par 2, de même, faites *SLA B...*

En combinaison avec les rotations, n'oubliez pas *CCF*. Cette instruction permet de complémenté la retenue, mais aussi certains autres bits. Ainsi, pour complémenté le bit 7 de A, vous pouvez faire *XOR 80H* ; mais pour un autre registre (B par exemple), plutôt que de le faire passer dans A, faites *RL B/ CCF/ RR B*. Un tableau récapitulatif des diverses rotations est donné en Annexe B.

Terminons pour le moment cette liste avec les instructions d'échange. Ces instructions permettent toutes sortes d'échanges. La plus importante est *EX (SP), HL*, car elle permet de prendre n'importe quel mot (16 bits), où qu'il se trouve en mémoire, et de le placer en haut de la pile, sans modifier aucun registre : *PUSH HL/ LD HL,(adresse)/ EX (SP),HL* ; idem pour des valeurs directes nn ; de plus, on peut ainsi en quelque sorte dédoubler le registre HL, en plaçant une variable en haut de pile lorsqu'il n'y a plus de place dans les registres. Les mêmes opérations sont possibles avec IX et IY.

LES INSTRUCTIONS CRÉÉES

Il vous arrivera de souhaiter la présence d'une instruction qui n'existe pas : il faudra alors la simuler, en utilisant plusieurs instructions existantes (cela donne une macro-instruction).

Ainsi *LD DE, BC* n'existe pas. Pour réaliser cela, il faut faire : *LD D,B/ LD E,C*. C'est simple. Tout se complique pour réaliser *LD IX,BC*, car IX n'est pas décomposable en registres 8 bits ; il faut passer par la pile : *PUSH BC/ POP IX*. Et pour *LD HL,SP*, il faut ruser : *LD HL,0/ ADD HL,SP*.

Comment réaliser *SRL HL* (division par deux de HL) : *SRL H/ RR L*, pas très difficile. *ADD IX,HL* n'existe pas non plus : on la remplacera par *EX DE,HL/ ADD IX,DE/ EX DE,HL*, facile aussi. Soit, mais comment faire *ADD BC,IX* ? Comme ceci : *PUSH IX/ ADD IX,BC/ EX (SP),IX/ POP BC*.

Les rotations sont plus compliquées quand il faut les faire sur 16 bits ; pour faire *RL HL*, par exemple, il faut faire *ADC HL,HL*, c'est simple mais subtil. Pour le contraire, *RR HL*, faire *RR H/ RR L*, tout simplement.

Et comment mettre F à une valeur donnée n (par exemple FFH pour mettre tous les flags à 1) sans changer aucun registre ? En faisant *PUSH HL/ LD H,A/ LD L,n/ EX (SP),HL/ POP AF*, par exemple...

Liste non exhaustive, qui prouve qu'il y a toujours moyen de se débrouiller en restant simple : à bon entendeur...

3. LES AMSTRAD CPC

Rappelons qu'il existe actuellement trois Amstrad CPC : le 464, avec lecteur de cassette et 64 K de mémoire vive ; le 664, avec lecteur de disquette et 64 K de mémoire vive ; le 6128, avec lecteur de disquette et 128 K de mémoire vive. Il existe entre eux de nombreux points communs. En particulier, leur microprocesseur est le même (Z80 sous 4 MHz), ce qui permet, aidé par les routines système (voir le Chapitre 8 qui leur est consacré), d'assurer une certaine compatibilité.

Nous allons à présent étudier un peu ces machines : car qui ne connaît pas bien son appareil ne peut espérer en tirer un bon parti.

■ LA MÉMOIRE DES CPC

ORGANISATION DES BLOCS MÉMOIRE

La mémoire des CPC est organisée par blocs de 16 K. Ces blocs sont d'un nombre variant de 6 à 11 suivant le type. Or le Z80 ne peut adresser directement que 64 K de mémoire (adresses 0 à FFFFH). Certains de ces blocs sont donc "parallèles", c'est-à-dire que lorsque l'un se trouve directement adressable, l'autre ne l'est pas. Ces blocs sont représentés de manière symbolique sur la figure de la page 43.

Nous voyons représentée sur ce schéma une échelle à droite représentant les quatre positions possibles des blocs mémoire : bloc inférieur (adresses 0 à 3FFFH), bloc médian inférieur (adresses 4000H à 7FFFH), bloc médian supérieur (adresses 8000H à BFFFH), bloc supérieur enfin (adresses C000H à FFFFH). Pour chacune de ces positions, un ou plusieurs blocs peuvent être connectés. Mais s'ils occupent les mêmes positions, deux blocs ne peuvent être connectés ensemble.

Ainsi, sur le CPC 464, si nous voyons que la partie médiane est formée de deux blocs de RAM seuls, par contre la partie inférieure est occupée simultanément par un bloc de RAM (où se trouvent notamment les programmes BASIC usuels) et un bloc de ROM très important : le système d'exploitation. La partie supérieure est de même occupée par deux blocs parallèles : la RAM écran d'une part, et la ROM BASIC d'autre part.

Sur le 664, un bloc a été ajouté en haut de mémoire : c'est évidemment le DOS (*Disc Operating System*) qui gère les disquettes et leurs lecteurs. En fait, ce bloc comprend aussi une partie de CPM, et une autre de Dr LOGO.

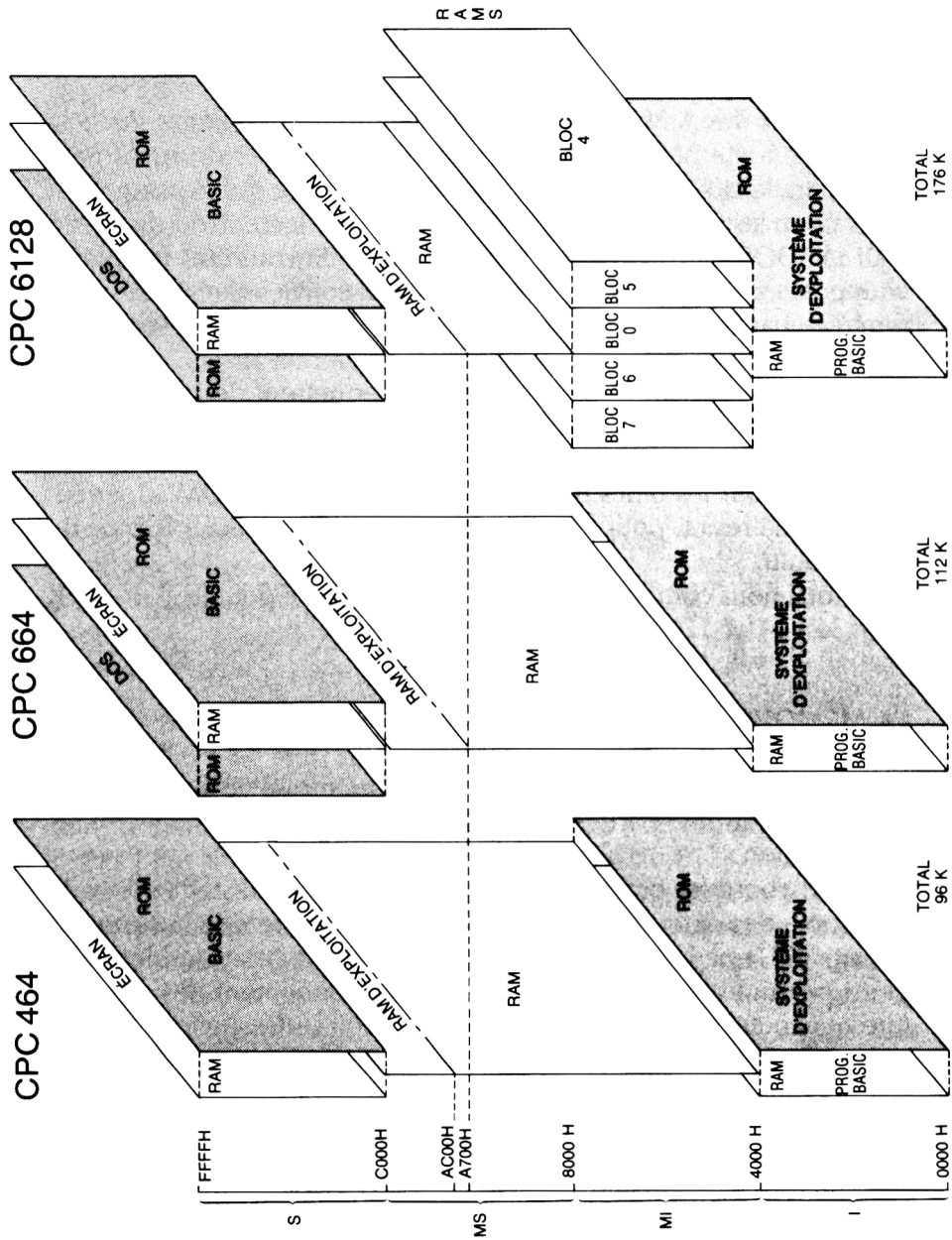


Figure 1 : Organisation mémoire des trois CPC.

Sur le 6128, les 64 K de mémoire supplémentaires ont été ajoutés sous la forme de quatre blocs de RAM (numérotés de 4 à 7) en plus du bloc déjà présent (bloc 0).

Il s'agit là des Amstrad dans leur configuration de vente. Mais la possibilité a été prévue de brancher jusqu'à 251 ROM supplémentaires parallèles au BASIC, comme le DOS. C'est ce qui se passe d'ailleurs chez les possesseurs de 464 munis d'un lecteur de disquette DDI : le DOS se trouve alors dans le lecteur. Si vous êtes bricoleur, vous pouvez ainsi ajouter toutes les ROM qui conviennent. Elles seront repérées par un numéro (0 = BASIC, 7 = DOS) ; ces ROM sont dites externes ou secondaires.

Dans tous les cas, des commutateurs permettent de brancher tel ou tel bloc parallèle en appelant certaines routines (voir Chapitre 7), notamment B900H et B903H pour l'écran et la ROM BASIC, B906H et B909H pour les blocs inférieurs, B90FH pour les ROM supérieures secondaires et, pour le 6128, BD5BH pour connecter le bloc de RAM voulu.

Au total nous comptons en fait 96 K de mémoire pour le 464, 112 K pour le 664 et 176 K pour le 6128.

LA MÉMOIRE VIVE

Outre l'écran qui occupe les 16 K de la mémoire vive supérieure, il existe d'autres parties de la RAM qui sont occupées dès l'initialisation.

Tout d'abord les octets 0 à 3FH sont pris par les restarts : il y a là une copie des octets 0 à 3FH de la ROM inférieure ; de la sorte ces restarts (RST) peuvent fonctionner quel que soit l'état du bloc inférieur, ce qui est essentiel pour le bon fonctionnement des routines (presque toutes introduites par des restarts : voir la description de ceux-ci et des routines au Chapitre 8). Ces octets ne doivent donc jamais être modifiés sans d'extrêmes précautions.

La ligne d'entrée du BASIC (celle que vous tapez après l'affichage du message *Ready*) commence en 40H. Limitée à 256 caractères, cette zone s'achève en 13FH. Le programme BASIC commence en 170H, après une partie transitoire. Ce programme est suivi des variables numériques et alphanumériques. La longueur de cette zone est évidemment très variable ; elle est limitée par l'instruction *memory*. En assembleur, il faut loger quelque part l'assembleur lui-même : tous se placent en 4000H (pour le reste voir la description des assembleurs

du commerce au Chapitre 1). En haut de la RAM médiane, on trouve les vecteurs de B900H à BDC0H (environ), suivis par la pile BASIC et précédés des variables du système dont les valeurs pour les trois CPC sont données en Annexe C. S'il y a un DOS, celui-ci a ses propres variables, qui sont placées en principe de A700H à ABAFH.

En résumé, la zone de RAM libre s'achève en AC00H pour un 464 sans lecteur de disquette, et en A700H pour les autres machines.

Notons encore une particularité curieuse des blocs de RAM du 6128 : ils ne sont pas réinitialisés par un *RST 0* ou assimilé ; seule une mise hors tension de la machine peut les remettre à zéro (façon de parler, car ils contiennent aussi des FFH en initialisation).

LE SYSTÈME D'EXPLOITATION

Une description complète du système d'exploitation serait hors de propos et guère utile ; mais il peut être intéressant de savoir quelques grandes lignes, en plus des routines listées dans le Chapitre 8.

Ce système d'exploitation est le cœur même de l'Amstrad : c'est lui qui s'occupe de tout, hormis de l'unité de disquette. Il a été modifié d'un CPC à l'autre ; du 464 au 664 surtout, il a fallu gagner de nombreux octets pour loger les parties supplémentaires (voir routines spécifiques des 664 et 6128) ; le plus gros rajout est indiscutablement celui provoqué par la routine de remplissage FILL, plus les modifications apportées par MASK au tracé de ligne : au total quelque 520 octets ! Pas de gros rajout en passant du 664 au 6128, mais encore de petites modifications. Dans les deux cas le système a été entièrement relu et on a cherché à gagner le maximum d'octets. Mais il a tout de même fallu détruire l'arithmétique entière, présente seulement sur 464, et dont le BASIC ne se sert pas (il a la sienne propre en haut de mémoire).

Le système d'exploitation se partage en onze morceaux bien distincts. Premier d'entre eux, le noyau (*kernel*) assure la gestion des événements, temporisés ou non, des ROM, des instructions supplémentaires. Une grande partie est recopiée en haut de mémoire (à partir de B900H). Il est suivi du groupe machine (*machine pack*), qui comprend toutes les routines d'entrées-sorties existantes (appelées par les autres groupes). Ensuite, on trouve un groupe qui ne comprend qu'une seule routine, suivie de la liste de tous les vecteurs par défaut : c'est le rétablissement de vecteurs (*jump restore*). Vient alors le groupe écran (*screen pack*), qui se charge de la gestion de l'écran

lui-même : c'est un sous-traitant fréquemment appelé par les deux suivants ; le premier de ces deux suivants assure la gestion du texte (*text screen*) (sortie des caractères à l'écran, fenêtres, etc.) ; le second gère les graphiques (*graphics screen*).

Puis l'on trouve les gestionnaires de périphériques : le clavier d'abord (*keyboard manager*), puis le générateur de son (*sound manager*), puis du lecteur de cassette (*cassette manager*), et ce, même sur les 664 et 6128. On trouve pour finir l'éditeur (*screen editor*), si souvent utilisé par le BASIC, puis la liste des matrices de caractères, enfin l'arithmétique : à virgule flottante, et sur 464, en plus, l'arithmétique sur les entiers.

AUTRES ROM

Il serait complètement hors de propos de parler ici de la ROM BASIC... De fait, les routines qui la composent sont pour la plupart interdites au programmeur en assembleur, car faisant sans cesse référence à des possibilités d'erreur, au programme BASIC, aux variables, etc. : il n'y a pratiquement aucune routine bien distincte à l'intérieur de cette ROM. On notera seulement dans l'Annexe C quelques rares adresses utiles, en particulier les points d'entrée du BASIC (pour les retours de programme).

Pour ce qui est du DOS, on trouvera de nombreux détails dans *le Livre du lecteur de disquette Amstrad* (voir bibliographie). Les communications avec les routines du DOS qui sont des extensions d'instruction sont expliquées dans le Chapitre 4.

■ LES ÉVÉNEMENTS TEMPORISÉS

Les Amstrad possèdent une particularité curieuse mais intéressante pour un bon programmeur en assembleur. Il s'agit des événements temporisés. Il n'est guère possible d'en parler beaucoup ici, car la question est très complexe, et de plus se comprend beaucoup mieux par des exemples et par la pratique.

Tout d'abord, de quoi s'agit-il ?

Le Z80 possède une horloge interne qui régulièrement, tous les 1/300 de seconde, lui envoie ce qu'on appelle une interruption, c'est-à-dire un signal donné. Lorsque le Z80 reçoit ce signal, il passe à une

certaine routine qui va examiner dans une queue d'attente s'il faut faire quelque chose, à défaut va décrémenter des compteurs. De temps à autre, un événement va donc se produire, c'est-à-dire qu'une certaine routine va être exécutée. Selon ce principe, et avec les mêmes conséquences, deux autres interruptions se produisent, elles, tous les 1/50 de seconde : l'une est un multiple de la précédente, l'autre est liée au retour du faisceau du moniteur (phase sombre). A chacune correspondent d'ores et déjà certains événements : pour l'une, l'incrémentement des chronomètres internes, pour l'autre, le rafraîchissement des couleurs et leur clignotement.

Certaines routines système (de BCD7H à BD07H) permettent d'ajouter et d'enlever d'autres routines d'événements. Pour plus de détails, on se référera à ces routines, au Chapitre 8, mais il faut ici encore expliquer quelques notions.

En particulier le bloc d'événement est une partie de la mémoire qui est utilisée par le système pour la gestion de votre événement. Elle doit comprendre huit octets pour les événements rapides (1/300 de seconde) ou liés audit retour de rayon du CRT (contrôleur vidéo), et quatorze pour les "normaux" (1/50 de seconde).

Le compteur, quant à lui, est la valeur qui est décrémentée à chaque interruption ; la routine est exécutée lorsqu'il arrive à zéro. Rien ne se passe si le compteur est négatif.

Pour d'autres détails encore et surtout pour des exemples, bien utiles dans un tel cas, on se référera au Chapitre 6.

■ LES INDIRECTIONS

Les indirections sont elles aussi une particularité d'Amstrad dont un bon programmeur en assembleur peut tirer utilement parti.

Ce sont des adresses (de BDCDH à BDF1H) où se trouvent des sauts à des routines de la ROM inférieure (attention, il s'agit de sauts, donc cette ROM est supposée connectée). L'intérêt de ces adresses est qu'elles sont appelées par certaines autres routines du système d'exploitation. Dès lors, on peut légèrement modifier ces dernières, tout simplement en détournant ces sauts ou en les supprimant (en mettant un code *RET* à la place).

C'est pourquoi ces routines portent le nom d'indirections. A vous de voir ce que leur modification peut donner. Elles peuvent être restaurées par certains RESET (indiqués).

Voici la liste de ces indirections, avec les routines qui les appellent (instruction *CALL*), et celles qui y sautent (instructions *JP*), ainsi que l'adresse de la routine qui les restaure, pour le cas où vous changeriez d'avis sur leur modification.

BDCDH

Saut à l'adresse : 1263H (464) ; 125BH (664) ; 125FH (6128).

Effet : Place un curseur sur l'écran. En sortie, AF est modifié.

Vecteurs sautant : BB5DH (PRINT2), BB75H (LOCATE), BB7BH (TCRON), BB90H (PEN), BB96H (PAPER), BB9CH (INVERS)
+ traitement des codes de contrôle 11 et 12.

Restauré par : BB51H (TRESET).

BDD0H

Saut à l'adresse : 1263H (464) ; 125BH (664) ; 125FH (6128).

Effet : Enlève le curseur de l'écran. En sortie, AF est modifié.

Vecteurs appelant : BB4EH (TINIT), BB6CH (CLS0), BB75H (LOCATE), BB7BH (TCRON), BB7EH (TCROF), BB90H (PEN), BB96H (PAPER), BB9CH (INVERS) et tout déplacement de curseur.

Restauré par : BB51H (TRESET).

BDD3H

Saut à l'adresse : 134AH (464) ; 1347H (664) ; 134BH (6128).

Effet : Écrit un caractère dont A contient le code à l'écran, à la position précisée par HL. En sortie, le groupe 1 est modifié.

Vecteur appelant : BB5DH (PRINT2).

Restauré par : BB51H (TRESET).

BDD6H

Saut à l'adresse : 13C0H (464) ; 13BAH (664) ; 13BEH (6128).

Effet : Lit un caractère à la position précisée par HL sur l'écran. Si un caractère est reconnu, il est placé dans A et la retenue est mise. Le groupe 1 est modifié.

Vecteur appelant : BB60H (READ).

Restauré par : BB51H (TRESET).

BDD9H

Saut à l'adresse : 140CH (464) ; 1406H (664) ; 140AH (6128).

Effet : Écrit le caractère ou exécute le code de contrôle contenu dans A. En sortie, le groupe 1 est modifié.

Vecteur appelant : BB5AH (PRINT).

Restauré par : BB51H (TRESET).

BDDCH

Saut à l'adresse : 1816H (464) ; 1782H (664) ; 1786H (6128).

Effet : Place un point à l'écran, aux coordonnées précisées par DE et HL. En sortie, le groupe 1 est modifié.

Vecteurs sautant : BBEAH (PLOT), BBEDH (PLOTR).

Restauré par : BBBDH (GRESET).

BDDFH

Saut à l'adresse : 182AH (464) ; 1796H (664) ; 179AH (6128).

Effet : Teste le point de coordonnées DE et HL. En sortie, A contient l'encre du point et le groupe 1 est modifié.

Vecteurs sautant : BBF0H (TEST), BBF3H (TESTR).

Restauré par : BBBDH (GRESET).

BDE2H

Saut à l'adresse : 183CH (464) ; 17B0H (664) ; 17B4H (6128).

Effet : Trace un trait à l'écran jusqu'au point de coordonnées DE et HL. En sortie, le groupe 1 est modifié.

Vecteurs sautant : BBF6H (DRAW), BBF9H (DRAWR).

Restauré par : BBBDH (GRESET).

BDE5H

Saut à l'adresse : C82H (464) ; C86H (664) ; C8AH (6128).

Effet : Donne dans A l'encre du point dont l'octet a pour adresse HL et dont le masque se trouve dans C.

Vecteur sautant : Indirection BDDFH.

Restauré par : BC02H (SRESET).

BDE8H

Saut à l'adresse : C68H (464) ; C6DH (664) ; C71H (6128).

Effet : Écrit un point à l'adresse écran HL, ayant pour masque C et pour masque d'encre B. En sortie, AF est modifié.

Vecteurs sautant : Indirections BDDCH et BDE2H.

Restauré par : BC02H (SRESET).

BDEBH

Saut à l'adresse : AF7H (464) ; B13H (664) ; B17H (6128).

Effet : Vide l'écran. En sortie, le groupe 1 est modifié.

Vecteur appelant : BC0EH (MODE).

Restauré par : BC02H (SRESET).

BDEEH

Saut à l'adresse : 1C2FH (464) ; 1DB8H (664 et 6128).

Effet : Teste si la touche ESC est enfoncée.

Vecteur appelant : Test général des touches.

Restauré par : BB03H (CRESET).

BDF1H

Saut à l'adresse : 7F8H (464) ; 825H (664) ; 835H (6128).

Effet : Envoie le caractère contenu dans A vers l'imprimante. En sortie, AF et BC sont modifiés.

Vecteur appelant : BD2BH (PRCAR).

Restauré par : BD2H (PRRESET).

4. COMMUNICATIONS ENTRE BASIC ET ASSEMBLEUR

■ LES APPELS DU BASIC

ROUTINES EN LANGAGE MACHINE

L'un des grands intérêts de la programmation en assembleur, c'est qu'on peut ainsi accélérer ses programmes BASIC en insérant en certains endroits des appels à des programmes assemblés.

Ces appels se font, vous le savez, par l'instruction BASIC CALL. Le premier paramètre de cette instruction est, comme en assembleur, l'adresse où l'appel doit se faire. Lorsque le BASIC rencontre cette instruction, il empile une adresse de retour, puis saute à l'adresse qui suit le CALL. Après exécution de la routine, l'interpréteur BASIC continuera de réaliser le programme en cours ou, si l'appel était direct, reviendra au mode *Ready*.

La ou les routines appelées doivent se trouver en mémoire à une adresse déterminée ; cette adresse (que pour éviter toute erreur on choisira généralement simple comme 8000H, par exemple) doit être supérieure à la valeur HIMEM du BASIC, faute de quoi la routine risque d'être détruite par des variables ou des lignes du programme ; n'oubliez pas que cette limite HIMEM a précisément été conçue dans ce but. De toute façon, l'interpréteur BASIC n'acceptera pas de charger en mémoire une routine en langage machine d'adresse inférieure à HIMEM.

L'utilisation de routines en langage machine est très commode pour toutes sortes d'instructions répétitives qui seraient très longues en BASIC. Par exemple, toutes les opérations sur la mémoire par PEEK et POKE sont très longues, comparativement à l'assembleur. De même, nombreuses sont les opérations graphiques qui gagneront à être réalisées en langage machine. Par contre, certains autres types y gagneront peu, ou rien (en regard de la difficulté relativement plus grande de réalisation) ; calculs arithmétiques, sortie sur l'imprimante de caractères (n'oubliez pas que c'est l'imprimante qui est lente), voire routines sonores...

En revanche, le langage machine peut vous permettre certaines choses difficiles ou impossibles en BASIC : par exemple les opérations basées sur les interruptions (encres clignotant à des vitesses différentes, chronomètres, etc.) ou sur les entrées-sorties (lecture de secteur disque, par exemple). Le gain en vitesse n'est donc pas la seule raison susceptible de motiver un recours à l'assembleur, même si c'est la plus courante.

LES PARAMÈTRES DE CALL

Le BASIC ne permet pas de charger les registres du Z80 avec des valeurs données pour l'appel d'une routine en langage machine. Néanmoins, vous disposez de deux possibilités qui vous permettront de transmettre tous les paramètres voulus à votre routine.

Tout d'abord, l'instruction BASIC CALL est susceptible d'admettre des paramètres. Ces paramètres doivent se trouver derrière l'adresse de l'appel, séparés entre eux par des virgules comme ceci :

CALL &8000, a%, b, 18, 3.75, "Bonjour", c\$

Ces paramètres peuvent être de toutes sortes, mais ils sont convertis par le BASIC en nombres sur deux octets, qui peuvent être de deux types :

- *Si la variable est un entier* compris entre -32768 et $+32767$, soit sous forme littérale (comme la valeur 18 dans l'exemple), soit sous la forme du nom d'une variable entière (comme a%), c'est cette valeur qui est transmise. De même si la variable est réelle (b et 3.75) : c'est l'entier le plus proche qui est transmis (4 pour 3.75), sauf si le réel est trop grand (le BASIC signale alors un *Overflow*) ; dans tous les cas, il s'agit bien d'une valeur sur deux octets.
- *Si la variable est une chaîne* (comme c\$ et "Bonjour") c'est l'adresse du descripteur de chaîne qui est transmise (voir description des variables du BASIC au paragraphe suivant).

Ces valeurs sont empilées quelque part ; au moment de l'accès à la routine, le BASIC place dans A le nombre de paramètres transmis (d'où une détection des erreurs aisée) et dans IX l'adresse du premier de ces mots de deux octets. Mais attention, ce premier est celui qui apparaît le dernier dans la liste (c\$ dans notre exemple), car ces nombres ont été empilés au fur et à mesure par le BASIC. Hormis cette précaution à prendre, vous n'aurez pas de difficulté à récupérer vos paramètres (nous en verrons des exemples).

Connaître l'adresse de la variable est extrêmement commode, pour y stocker le résultat de votre routine, par exemple. Cela est fait automatiquement pour une chaîne. Pour un entier ou un réel, il vous faudra utiliser le pointeur de variable @ (touche à droite de P), ressource étonnante du BASIC Amstrad. En effet ce pointeur permet d'obtenir

l'adresse où sont stockées les variables. Ainsi *PRINT @X* n'affichera pas la valeur de X, mais bien l'adresse de stockage dans la mémoire (voir paragraphe suivant) de cette variable. Connaissant cette adresse et le type de la variable, rien n'est plus simple que de la modifier. Nous allons examiner à présent comment les variables sont stockées en mémoire et le rapport avec le pointeur de variable @. Rappelons simplement que, pour les chaînes X\$, le paramètre de CALL, même écrit X\$, est de toute façon @X\$.

■ LES VARIABLES DU BASIC

Comme nous l'avons vu au paragraphe précédent, il est très utile de savoir comment les variables sont stockées en mémoire par le BASIC, ce afin de pouvoir les modifier ou les lire ; il faut aussi savoir exactement sur quelle valeur pointe le pointeur de variable @.

Les variables BASIC sont stockées les unes après les autres après le programme (en général). Dans tous les cas, on trouve d'abord le nom de la variable (dont le bit 7 du dernier caractère est mis). Ce qui suit est expliqué à présent :

VARIABLE ENTIÈRE

C'est la plus simple : après le nom, on trouve la longueur de cette variable en octets, soit 2, puis les deux octets donnant la valeur de l'entier (inversés comme toujours). Le pointeur de variable pour un entier donne l'adresse des deux octets en question (et non du nom...).

VARIABLE RÉELLE

Plus complexes, les réels ; le début est le même : après le nom, on trouve la longueur de la variable (5), puis les cinq octets de la variable proprement dits (sur lesquels pointe le pointeur @). C'est la valeur de ces cinq octets qui est plus mystérieuse. Supposons que IX pointe sur ce premier octet, comme le pointeur de variable @. Dès lors :

- Si $(IX+4)$ est nul, le réel vaut 0 (les 4 autres octets n'ont pas d'influence dans ce cas).

- Le bit 7 de (IX + 3) donne le signe du réel : 0 = positif, 1 = négatif.
- (IX + 4) moins 80H représente la puissance de 2 qui suit le réel, soit 1 + le logarithme en base 2 du réel. Donc si c'est un nombre positif, c'est le nombre de chiffres précédant la virgule dans la représentation binaire du réel. Si c'est un nombre négatif ou nul, sa valeur absolue est le nombre de zéros compris entre la virgule et le premier 1 dans cette même représentation (voir exemples).
- (IX + 0), puis (IX + 1), puis (IX + 2) et enfin les sept derniers bits de (IX + 3) contiennent tous les bits de la représentation binaire du réel qui suivent le premier 1, celui-là non compris. Ainsi :

10 se code 00 00 00 02 84 car :

10 s'écrit en binaire 1010. Cela fait quatre chiffres, donc (IX + 4) = 80H + 4 = 84H. On ne tient pas compte du premier 1 et l'on obtient alors les 31 bits : 00000000 00000000 00000000 s0000010. s (bit de signe) vaut 0 car 10 est positif. Donc on obtient bien les valeurs 00 00 00 02 en hexadécimal.

-0.5 se code 00 00 00 80 80 car :

0.5 (1/2) s'écrit en binaire 0.1. Cela ne fait aucun chiffre devant la virgule : on compte donc le nombre de zéros qui la suivent immédiatement : il y en a 0. Donc (IX + 4) = 80H - 0 = 80H. On ne tient pas compte du premier 1 : il reste 00000000 00000000 00000000 s0000000, ici s = 1 (nombre négatif). On obtient donc 00 00 00 80 pour les quatre premiers octets.

7188.25 se code 00 00 60 29 8D car :

7188.25 (7188 + 1/4) s'écrit en binaire 1110000001010.01. Il y a 13 chiffres devant la virgule donc (IX + 4) = 80H + 13 = 8DH. Puis on ôte le premier 1, il reste 00000000 00000000 01100000 s0101001. Ici s = 0 (nombre positif). Cela donne bien 00 00 60 29.

LES CHAÎNES ALPHANUMÉRIQUES

Pour une chaîne, on trouve après son nom le nombre 3, puis un descripteur de chaîne de trois octets sur lequel pointe le pointeur de variables @. Ces trois octets sont le nombre de caractères de la chaîne (codé sur un octet) puis l'adresse de la chaîne proprement dite (sur deux octets).

Les chaînes elles-mêmes sont stockées en haut de mémoire, dans le sens descendant inverse des variables.

LES MATRICES

Après le nom de la matrice (ou tableau, ou arrangement) vient le type des variables du tableau (1 = entier, 2 = chaîne, 4 = réel, qui est aussi leur longueur moins 1) sur un octet, puis le nombre qu'il faut additionner à l'adresse courante pour trouver la variable suivante (sur deux octets), puis le nombre d'indices du tableau (sur un octet), puis les nombres de valeurs possibles pour chaque indice, dans l'ordre (et en comptant les valeurs 0), enfin les variables elles-mêmes, les unes après les autres. Pour les chaînes, ce sont les descripteurs de chaînes qui sont à cet endroit.

Le pointeur de variable @ ne peut être utilisé pour une matrice, mais il peut l'être pour n'importe lequel de ses éléments. Ainsi *CALL &8000, @MAT(0,0)* donnera à la routine l'adresse de *MAT(0,0)*, donc des autres éléments qui le suivent. Par exemple : après un *DIM MAT(6,14)*, on trouvera en mémoire, si AD est l'adresse du nom (toutes les valeurs des octets en hexadécimal) :

$AD+0=4D$; $AD+1=48$; $AD+2=C4$, soit "M", "A" et "T" + 80H.

$AD+3=04$: variables réelles (longueur 4 + 1).

$AD+4=6F$; $AD+5=00$: car il faut ajouter 111 (006FH) à $AD+4$ pour trouver la variable suivante en $AD+115$.

$AD+6=02$: deux indices.

$AD+7=07$; $AD+8=0F$: le premier indice a sept valeurs possibles (de 0 à 6), le second quinze (de 0 à 14).

$AD+9$ à $AD+14$: valeur de *MAT(0,0)* (réel sur cinq octets).

$AD+15$ à $AD+20$: valeur de *MAT(0,1)*.

... etc.

$AD+109$ à $AD+114$: valeur de *MAT(6,14)*.

$AD+115$: nom de la variable suivante...

Ici, @*MAT(0,0)* donnerait $AD+9$.

■ LES EXTENSIONS D'INSTRUCTION

Les RSX (*Resident System Extension*), ou extensions d'instruction, sont l'un des meilleurs atouts d'un BASIC bien conçu. Elles permettent en effet au programmeur de BASIC et d'assembleur d'exercer ses talents dans les deux langages, en créant en assembleur des routines qui pourront être utilisées en BASIC sous la forme d'instructions.

Il s'agit en effet d'ajouter des instructions supplémentaires au BASIC. Celui-ci est déjà fort complet, mais rares sont les programmeurs assez sages pour s'en contenter...

De fait, ces RSX peuvent s'avérer très utiles. C'est singulièrement le cas des RSX graphiques, qui accélèrent bien souvent le déroulement des programmes. Ainsi cercles, étoiles, etc., sont généralement très lents en BASIC, d'où l'intérêt du langage machine. Naturellement, il existe d'autres possibilités d'applications.

UTILISATION DES RSX EN BASIC

La RSX se différencie assez peu d'une simple routine appelée par l'instruction BASIC CALL, mais on y aura recours dans certains cas bien définis : si la routine est utilisée fréquemment, si elle est utilisée en mode direct (car il est pénible de taper CALL &8000, ..., et aventureux : risque d'erreur sur l'adresse), et surtout si elle est générale, c'est-à-dire si elle est susceptible d'être utilisée dans différents programmes (cas du cercle, etc.).

Dans une RSX, on remplace donc l'adresse de la routine qui est un nombre (d'où mémorisation difficile, risque d'erreur, etc.) par un nom. Le risque d'erreur est ainsi considérablement réduit : si vous orthographiez mal le nom, il y a de fortes chances pour que vous receviez le message *Unknown command* et, au pire, que vous exécutiez une mauvaise commande. Par contre si vous vous trompez dans l'adresse (&800 au lieu de &8000...), le BASIC ne protestera en général pas et l'adresse appelée fera "planter" l'ordinateur quatre fois sur cinq.

Pour signifier au BASIC qu'il s'agit d'une RSX et éviter le terrible *Syntax error*, on fait précéder le nom de l'instruction par une barre verticale (|) obtenue en pressant simultanément sur (SHIFT) et @ (touche à droite de P). Derrière le nom de l'instruction, séparés de celui-ci et entre eux par des virgules, on place les paramètres de l'instruc-

tion, comme pour un CALL de BASIC normal. Ainsi, par exemple, *CALL &8000, 320, 200, 150* sera remplacé par */CERCLE, 320, 200, 150*.

D'ailleurs, les utilisateurs d'Amstrad dotés d'unités de disquette connaissent déjà bien les RSX : */A, /ERA, "fichier"*, etc. sont autant de RSX dues au DOS.

UTILISATION DES RSX EN ASSEMBLEUR

En assembleur, une RSX est équivalente d'un CALL : après avoir reconnu dans une liste le nom d'une instruction, le système passe à cette routine. L'adresse de retour au BASIC a été préalablement empilée.

Pour ce qui est des paramètres, même chose également : nombre des paramètres dans A, paramètres ou adresses empilés et pointés par IX.

INITIALISATION DES RSX

Naturellement le système ne connaît l'existence que des RSX qu'on lui signale. Il faut donc initialiser les RSX avant de les utiliser.

Cela se fait assez facilement, à l'aide la routine INTRSX (BCD1H). Pour initialiser une ou plusieurs RSX, voici comment faire :

Placer dans HL l'adresse d'une table de stock de 4 octets libres dont le système aura besoin ; placer dans BC l'adresse d'une table de commandes organisée comme suit : adresse des noms/ saut à l'adresse de la première routine/ saut à l'adresse de la deuxième, etc. L'adresse des noms (2 octets) est l'adresse du premier caractère du nom de la première instruction ; les autres caractères doivent suivre, le dernier étant marqué par un bit 7 à 1 (+80H), puis les noms des autres instructions (s'il y a lieu). L'adresse des noms est suivie des sauts aux adresses des routines, dans l'ordre. La table des noms peut contenir autant de RSX que l'on veut. Elle se termine par un octet à zéro.

Après chargement des registres BC et HL avec ces deux adresses, il ne reste plus qu'à passer à la routine INTRSX. Au retour de celle-ci, les RSX sont initialisés.

En pratique il sera commode de placer l'initialisation en tête de la routine et de sauvegarder le fichier assemblé globalement. Dès lors, il suffira, au lieu de charger simplement le fichier par LOAD, comme pour une routine appelée par un CALL, de la charger par RUN : l'ini-

tialisation suivra immédiatement le chargement de la RSX, sans compliquer vos programmes.

RECHERCHE DES RSX

Une autre possibilité très intéressante offerte par le système est la recherche d'une RSX. En effet les noms des RSX ne sont pas stockés dans des tables : plutôt que de les rechercher vous-même, laissez ce soin au système : la routine FINDRSX (BCD4H) vous permettra de la trouver sans difficulté : il suffit de placer dans HL l'adresse du nom de la RSX à trouver (le dernier caractère de ce nom doit avoir son bit 7 mis) ; en retour, vous aurez dans HL l'adresse de la routine de la RSX et dans C l'octet de sélection des ROM : il ne vous reste plus qu'à faire un *JP (HL)* si l'adresse est en RAM, sinon il n'y a qu'à stocker l'adresse puis l'octet de sélection dans une table, puis à faire un *RST 18H* (suivi de *DEFW TABLE*) qui amènera à la routine. S'il y a des paramètres, imitez le BASIC : stockez-les (si ce sont des chaînes ou des variables dont les adresses doivent être données, stockez ces adresses) dans l'ordre (en les empilant, donc par emplacements d'adresses décroissantes), puis donnez l'adresse du dernier stocké dans IX et le nombre de paramètres dans A : c'est tout.

L'intérêt vient de ce qu'il est inutile ainsi de chercher fébrilement la table des adresses des instructions du DOS, par exemple (USER, REN, etc.). Il suffit de savoir leur nom ; en outre, il n'est plus nécessaire de connaître l'octet de sélection de la ROM du DOS. Cela vaut aussi pour toute autre ROM secondaire, évidemment.

Notons qu'il n'est pas nécessaire d'appeler plusieurs fois FINDRSX si vous utilisez la RSX à plusieurs reprises. Il suffit de l'appeler une seule fois, puis de stocker dans une table les trois octets comme il a été dit précédemment. A chaque appel de la routine, un *RST 18H/DEFW TABLE* de la même longueur qu'un *CALL* (contre toute apparence) suffira alors.

5. OPTIMISATION ET PROGRAMMATION

■ OPTIMISATION

L'assembleur a deux avantages principaux : ses programmes sont notablement plus rapides qu'avec le BASIC, et même qu'avec ceux des langages compilés comme le Pascal, et ils prennent généralement moins de place que ces derniers.

Optimiser un programme consiste à tenter de tirer un profit maximal de ces deux avantages. Mais comme il y a deux avantages, il y a aussi deux optimisations possibles : l'optimisation en durée, et l'optimisation en longueur. La première consiste à rendre l'exécution du programme la plus rapide possible, la seconde à limiter au maximum l'encombrement en mémoire de ce programme.

On serait tenté de croire que, si un programme est optimisé en longueur, possédant moins d'instructions à exécuter, il sera aussi plus rapide. Tel n'est, hélas, pas toujours le cas. Nous en verrons de nombreux exemples. Il faudra donc bien souvent faire un choix ou trouver un juste milieu.

OPTIMISATION EN DURÉE

Ce type d'optimisation sera préféré dans les programmes soit naturellement courts (et donc où le gain d'optimisation en longueur serait faible), soit d'utilisation très fréquente (en particulier dans les boucles, où l'on préférera une longueur un peu plus grande pour un peu plus de rapidité), soit évidemment exigeant une grande rapidité (cas par exemple des jeux de réflexes, où il est préférable que la machine réagisse plus vite que le joueur, mais aussi des événements temporisés dont la durée doit obligatoirement être limitée, faute de planter la machine).

L'optimisation en durée sera réalisée en utilisant les instructions les plus rapides : vous serez aidé par l'Annexe A sur ce point. Mais il existe d'autres moyens d'accélérer vos programmes. Citons notamment :

- Calculer sur des entiers, ce qui est toujours plus rapide que sur des réels. Évitez en particulier les calculs de cosinus, tangentes et autres exponentielles ou puissances, calculs de polynômes très longs.

- Limiter l'usage des routines du système, introduites par des restarts dont la durée n'est pas très brève. Si vous avez la possibilité de laisser la ROM inférieure branchée, changez ces restarts en jumps. Vous pouvez également réécrire tout ou partie de ces routines ; certaines, notamment les lectures de variables système, sont de simples chargements comme *LD HL,(nn)*. Notez qu'en contrepartie vous perdez la compatibilité entre les Amstrad.
- Remplacer les JR par des JP. Ces derniers en effet prennent un octet de plus mais leur exécution est moins longue.
- Utiliser le moins de CALL possible. Pour des appels à des routines courtes, préférez recopier plusieurs fois la routine afin d'éviter les CALL.
- Réécrire partiellement ou totalement certaines routines de sorties vers l'écran (ou d'ailleurs vers d'autres périphériques) afin de les adapter à votre cas. Par exemple, la sortie d'un caractère en mode 2 est très simple : il suffit de placer directement dans la RAM écran les huit octets de la matrice caractère (dont l'adresse est donnée par MATADR en BBA5H) à des adresses différant de 800H (différence entre deux lignes). La routine correspondante est très simple, contrairement à celle de la sortie en mode 1 ou 0, bien plus complexe.

Liste évidemment non exhaustive... Voyez notamment les solutions particulières adoptées pour les programmes de ce livre.

OPTIMISATION EN LONGUEUR

Ce type d'optimisation sera préféré dans les programmes longs qui utilisent de surcroît beaucoup de place en mémoire (présence de tableaux, de fichiers complexes) et dont la rapidité est suffisante (jeux de réflexion, programmes de gestion de tableaux, etc.).

Cette optimisation peut être également obtenue de diverses façons, dont malheureusement beaucoup sont la négation de celles du paragraphe précédent permettant une optimisation en longueur. Citons :

- Utiliser des instructions les plus courtes possible en s'aidant de l'Annexe A. Certaines sont d'intéressants raccourcis : *DJNZ*, *LDIR*, etc.

- Préférer un saut relatif à un saut absolu dès que possible. Si cela ne l'est pas, vérifiez que vous ne pouvez pas rapprocher les deux extrémités du saut pour remplacer le JP par un JR. Pour cette même raison, on évitera les tests basés sur les flags *s* et *p/v* qui ne peuvent être testés par JR.
- Utiliser le plus possible de sous-routines. Lisez avec soin votre programme, et transformez toute séquence répétée de quatre octets ou plus en une sous-routine séparée. Utilisez au maximum les routines déjà présentes en mémoire : les routines système, bien entendu, mais aussi celles de votre programme (vérifiez que vous ne pouvez pas transformer en une seule routine deux actes distincts mais peu différents).

On notera la contradiction entre ces deux paragraphes. La synthèse pour les programmes généraux sera faite plus loin. Notez toutefois qu'ici encore la liste n'est pas exhaustive, et aussi que des solutions particulières nombreuses se trouvent dans les programmes de ce livre.

ACTES ÉLÉMENTAIRES OPTIMISÉS

Il est des actes très brefs qui sont très fréquemment utilisés dans tous les programmes, par exemple doubler HL ou l'augmenter d'une valeur *n*. Pour certains de ces actes, il existe plusieurs possibilités, et parmi elles il n'en existe qu'une en général qui soit optimale, soit en longueur, soit en durée. Cette solution optimale est donnée dans la suite pour quelques actes courants. On n'a pas redonné chaque fois la ou les mauvaises solutions. Pour les actes dépendants d'un paramètre, il arrive qu'il y ait plusieurs solutions, selon la valeur du paramètre ; dans ce cas, on les a données toutes, en précisant pour quelles valeurs du paramètre elles étaient optimales en durée ou en longueur.

Les notations sont les suivantes : *dd* désigne un registre double (BC, DE ou HL) et *d_n* son octet fort (B, D ou H), *d_l* l'autre (C, E ou L) ; *nn* est une adresse sur deux octets, *n* un nombre sur un seul octet. IX désigne aussi bien IX que IY (instructions équivalentes). L'acte est l'action à réaliser, la séquence est la suite d'instructions optimale. La durée de celle-ci est calculée en temps machine (valant un quart de microseconde) et sa longueur en octets.

Enfin, on a supposé que tous les registres non concernés par l'acte devaient être conservés (sauf mention : registre libre), à l'exception de AF, librement modifié.

LISTE D'ACTES ELEMENTAIRES OPTIMAUX

Tester si $dd = 0$ (en positionnant le flag z)

Séquence : $LD A, d_r / OR d_r$

Durée : 8

Longueur : 2

Mettre dd à 0

Séquence : $LD dd, 0$.

Durée : 10

Longueur : 3

Mettre dd et A à 0

Séquence : $XOR A / LD d_r, A / LD d_r, A$.

Durée : 12

Longueur : 3

Doubler dd

Séquence : si $dd = HL$: $ADD HL, HL$., sinon : $SLA d_r / RL d_r$.

Durée : 11 pour HL , 16 sinon.

Longueur : 1 pour HL , 2 sinon.

Remarque : La retenue est positionnée en fonction du dépassement, dans les deux cas.

Échanger dd et IX

Séquence : $PUSH dd / EX (SP), IX / POP dd$.

Durée : 44

Longueur : 4

Ajouter la valeur n à HL , le registre dd étant libre

Séquences : (a) : $INC HL$, n fois.

(b) : $LD dd, n / ADD HL, dd$.

Durée : (a) : $6*n$ (meilleure si $n < 4$), (b) : 21 (meilleure si $n > 3$).

Longueur : (a) : n (idem), (b) : 4 (idem).

Ajouter la valeur n à HL, BC et DE étant occupés

Séquences : (a) : comme ci-dessus.

(b) : *PUSH BC/ LD BC,n/ ADD HL,BC/ POP BC.*

Durée : (a) : $6*n$ (meilleure si $n < 7$), (b) : 42 (meilleure si $n > 6$).

Longueur : (a) : n (idem), (b) : 6 (idem).

Mettre à zéro les B octets d'adresses HL et suite

Séquences : (a) : *XOR A/BOUCLE: LD (HL),A/ INC HL/ DJNZ BOUCLE.*

(b) : *PUSH DE/ XOR A/ LD C,B/ LD B,A/ LD D,H/ LD E,L/ INC DE/ LD (HL),A/ LDIR/ POP DE.*

Durée : (a) : $23*B + 28$ (meilleure si $B < 22$), (b) : $21*B + 70$ (meilleure si $B > 21$).

Longueur : (a) : 5, toujours meilleure ((b) : 11).

Remarque : Pour comprendre (b), voir l'acte suivant.

Mettre à zéro les BC registres d'adresses HL et suite (BC grand)

Séquence : *PUSH DE/ LD D,H/ LD E,L/ INC DE/ LD (HL),0/ LDIR/ POP DE.*

Durée : $21*BC + 61$

Longueur : 9

Remarque 1 : Cela ne vous paraît peut-être pas très clair ? De fait, c'est une astuce très artificielle : HL pointe sur le début de la zone à vider, DE juste sur l'octet suivant ($DE = HL + 1$). On met 0 dans (HL). Dès lors, si l'on fait un LDI, le zéro est reporté sur l'octet suivant, et HL et DE sont incrémentés : à ce moment HL pointe sur l'ancienne valeur de DE, mise à zéro, et DE sur l'octet suivant : il ne reste qu'à poursuivre, d'où le LDIR. Cette méthode permet un vidage d'écran très rapide (86 millisecondes pour 16 K !)

Remarque 2 : En faisant *LD (HL),n*, on peut ainsi remplir toute une zone mémoire avec n'importe quel octet.

Charger BC, DE et HL avec (IX + 0), (IX + 1), ..., (IX + 5) (avec deux octets libres en nn et nn + 1)

Séquence : LD (nn),SP/ LD SP,IX/ POP BC/ POP DE/ POP HL/
LD SP,(nn).

Durée : 80

Longueur : 13

Remarque : Méthode bien meilleure que le simple chargement
LD C,(IX+0)/ LD B,(IX+1)/ ... (durée 114 et longueur 18). Très
utile pour récupérer les paramètres d'un CALL du BASIC ou d'une
RSX.

Charger une table avec (IX + 0), (IX + 1), ..., (IX + n)

Séquences : (a) : LD A,(IX+0)/ LD (TABLE+0),A/ .../LD A,(IX+n)/
LD (TABLE+n),A.

(b) : PUSH BC/ PUSH DE/ PUSH IX/ EX (SP),HL/ LD BC,n+1/
LD DE,TABLE/ LDIR/ POP HL/ POP DE/ POP BC.

Durée : (a) : $32*(n+1)$ (meilleure si $n < 9$), (b) : $21*n+122$
(meilleure si $n > 8$).

Longueur : (a) : $6*(n+1)$ (meilleure si $n < 2$), (b) : 16 (meilleure si
 $n > 1$).

Remarque : La deuxième séquence a l'avantage de pouvoir accepter
une longueur variable facilement (en faisant LD BC,(LONGUEUR)
par exemple), d'où possibilité d'usage en sous-routine, etc.

Ici encore, cette liste n'est nullement exhaustive. Je vous incite à
l'utiliser, et même à la compléter le plus largement possible. Notez
que certains de ces actes sont utilisés dans les programmes de cet
ouvrage.

■ PROGRAMMATION

Avant de passer aux exemples pratiques, nous allons définir ici quel-
ques principes simples de programmation, applicables à la quasi-
totalité des programmes en assembleur.

RAPPORTS AVEC L'ASSEMBLEUR

Quel que soit l'assembleur dont vous disposez, il est important d'en utiliser à fond les possibilités, même celles qui ne sont pas d'usage obligatoire.

En premier lieu les labels sont d'une utilité fondamentale échappant parfois au programmeur d'assembleur débutant, qui se limite forcément à de très courts programmes. En fait, l'emploi de labels et d'étiquettes doit être quasi systématique dans tout programme (s'il est destiné à être conservé), même si ce label ou cette étiquette ne sont utilisés qu'une fois. Pour les étiquettes, tout d'abord, la raison en est double : d'abord, cela évite d'avoir à réécrire plusieurs fois le même nombre hexadécimal (ce qui est à la fois pénible et périlleux) soit parce que l'étiquette intervient plusieurs fois, soit parce que dans un développement futur du programme vous pourriez avoir à réutiliser cette étiquette ; ensuite cela rend le programme notablement plus clair. Cette seconde raison est également valable pour les labels. Ajoutons pour ces derniers qu'une instruction comme *JR + 17* est tout à la fois une imprudence et une sottise : imprudence, car si vous vous trompez dans votre compte, tout est perdu ; sottise, car si vous venez à insérer un ordre supplémentaire, comme cela est fréquent, même en admettant que vous y pensiez vous serez obligé de changer le paramètre aussi, ce qui est une perte de temps totalement inutile : n'oubliez pas que l'Assembleur, lui, met seulement quelques centièmes de milliseconde à faire le calcul de cette différence...

Donc, mieux vaut utiliser au maximum les symboles de votre Assembleur. Vous n'y perdrez jamais. La seule petite difficulté susceptible d'intervenir est l'utilisation du même label plusieurs fois. Pour éviter cet inconvénient, je vous recommande *primo* d'utiliser pour les routines système toujours les mêmes étiquettes, par exemple celles que je donne au Chapitre 8 (elles sont prévues pour cet usage et sont toutes distinctes) ; *secundo* d'éviter les étiquettes trop banales comme BOUCLE et SUITE, ou alors de les faire suivre soit d'un numéro (BOUCLE1, ...), soit de la désignation du sous-programme où elles se trouvent (BCLEROUT, ...). Si vous en prenez l'habitude, ces petites opérations sur les labels se feront sans effort, et vous pourrez sans problème étiqueter tout un programme même de grande taille, ce qui, sans méthode, s'apparente à un exercice de haute voltige.

Second usage essentiel de l'assembleur : les commentaires. Rappelons qu'après l'étiquette éventuelle et l'instruction de la ligne vous

pouvez ajouter un commentaire, séparé de l'ordre par un point-virgule. Ce commentaire (c'est-à-dire tous les caractères compris entre le point-virgule et la fin de la ligne) est purement et simplement ignoré au moment de l'assemblage. Contrairement à un REM du BASIC, il n'allonge en rien le programme assemblé et ne modifie nullement son exécution : il n'y a donc pas de raison de s'en priver. Ce serait là un mauvais calcul, car rien ne prouve que vous ne souhaiterez pas, dans quelques semaines, modifier à nouveau votre programme : les commentaires vous permettront alors de vous y retrouver.

Naturellement, il serait excessif de mettre un commentaire à chaque ligne ; on ne l'a fait dans le présent ouvrage que pour faciliter la compréhension des programmes au lecteur, mais celui-ci n'est nullement tenu de les recopier (ce serait inutile, s'il conserve le livre).

Néanmoins, même si vous mettez peu de commentaires, choisissez-les bien. Ceci : `LD B,0 ;METTRE B A ZERO` est inutile, car on ne voit vraiment pas en quoi le commentaire éclaire l'instruction ! En revanche, ceci : `LD B,0 ;INITIALISER LE COMPTEUR` est bien meilleur car le commentaire ici n'est pas redondant avec l'instruction : il lui donne un sens particulier. Tout cela peut paraître évident, mais il n'en est pas moins vrai que nombreux sont les programmes en assembleur dont les commentaires sont si mal faits qu'on ne comprend rien de la volonté du programmeur, même dans ceux qui, par ailleurs, sont excellents.

ORGANISATION DU PROGRAMME SOURCE

A l'intérieur du programme source (version non assemblée), il existe toujours différents groupes dont le classement est important. Chacun a sur ce point ses idées plus ou moins arrêtées. Je vais ici donner les miennes, celles avec lesquelles j'ai écrit les programmes de ce livre, ainsi que les arguments qui les soutiennent.

En premier lieu, il est naturel de mettre une présentation. Celle-ci, plus ou moins sophistiquée, a surtout pour but de vous aider, lors d'une lecture ultérieure du programme, à vous en rappeler les modes d'utilisation et même l'usage, le nom n'y suffisant pas toujours (surtout sur disquette). Cette présentation, constituée de lignes de commentaires, peut indifféremment précéder ou suivre le pseudo-opérateur ORG (en général obligatoire).

J'ai également l'habitude de faire suivre cette présentation de toutes les définitions d'étiquettes nécessaires. Bien sûr, en théorie, on

pourrait semer ces définitions dans le programme, au fur et à mesure qu'on en a l'usage. Mais cela rend le programme très confus et de plus, si vous avez plus tard besoin de cette étiquette plus haut, vous serez obligé d'en réécrire la définition. Donc il vaut mieux les grouper au début. Toutefois, lorsqu'un programme est très nettement scindé en plusieurs parties (cas par exemple des RSX, où l'on distingue l'initialisation de la RSX de son exécution), il peut être plus clair de séparer les étiquettes (si elles ne se recoupent pas).

Il reste encore, hormis le groupe des instructions proprement dites, un groupe fréquemment présent, constitué des données nécessaires au programme, des chaînes alphanumériques (messages divers) et des emplacements libres pour stocker des variables. Ce groupe peut soit précéder, soit suivre les instructions. Dans le second cas, la lisibilité du programme en souffre (obligation de se reporter à la fin) et l'on risque d'avoir des chevauchements inattendus avec d'autres programmes. Dans le premier, le point d'entrée du programme ne coïncidera plus avec son adresse de chargement (et sera de plus inconnu avec DAMS) ; c'est sans importance pour une RSX, mais gênant pour les programmes appelés par un CALL du BASIC ; toutefois, pour remédier à ce dernier inconvénient, on peut placer en tête des données un saut au point d'entrée : c'est la solution choisie pour DAMS et ZEN par leurs concepteurs ainsi que pour l'annuaire du Chapitre 7.

Vous pouvez très bien adopter d'autres points de vue sur ces problèmes : c'est généralement peu important en soi. Ce qui l'est, par contre, c'est de conserver toujours la même organisation ; cela vous permettra de relire facilement vos programmes et surtout d'éviter des attitudes contradictoires dans l'élaboration d'un même programme.

ORGANISATION PAR STRATES DES PROGRAMMES

Nous allons à présent parler de l'organisation du programme tel qu'il sera après assemblage (même si cette organisation s'exécute évidemment sur le programme source).

Il est très important de comprendre l'utilité des strates dans un programme, c'est-à-dire d'une sorte de hiérarchisation des différentes parties qui le composent. Cette hiérarchie est la suivante : on dira qu'une partie (terminée par un *RET*, un *JP*, un *JR* en général) est un sous-programme (ou une subroutine) d'une autre si elle est appelée par cette autre (via l'instruction *CALL* en général). Si deux routines au contraire sont liées par des sauts, elles sont de même niveau.

Ces notions sont absolument fondamentales, car elles sont les premiers pas que vous devez absolument réaliser avant de commencer à écrire un programme. Il vous faut réfléchir à ce qui sera fait : quelle routine sera utilisée plusieurs fois ? Par quelles autres ? Avec quels paramètres ? Quels devront être ses sorties (voir paragraphe suivant) ? Ce découpage fait, vérifiez que ne pouvez encore opérer quelques regroupements en sous-routines. Puis entrez le programme.

Mais pas dans n'importe quel ordre : les routines d'égale importance doivent se trouver voisines, et les routines moins importantes doivent suivre les autres. Rappelez-vous en effet qu'il n'existe pas d'appel à une adresse relative dans le Z80. Donc qu'une sous-routine soit éloignée est sans importance. Par contre, deux routines liées par un saut doivent être les plus proches possible pour permettre d'utiliser un *JR*. On essaiera même de les placer l'une derrière l'autre, afin de supprimer totalement le saut.

Si votre programme risque d'être un peu long, ne le tapez pas tout d'un coup : en cas de plantage (il est rare qu'un programme marche dès le premier essai) vous ne sauriez plus repérer la cause de l'erreur. Il vaut donc beaucoup mieux procéder par étapes : entrer d'abord une routine, avec ses sous-routines éventuelles, la tester puis entrer ses routines de grade supérieur, etc., puis recommencer avec d'autres routines jusqu'à ce que le programme soit achevé. Si vous avez fait des tests chaque fois, il est peu probable que vous éprouviez des surprises désagréables.

Encore une fois ne négligez pas cette organisation : elle vous permettra de développer vos programmes beaucoup plus facilement et par étapes successives, ce qui est assurément la meilleure méthode, surtout si vous n'êtes pas très familier avec la programmation en général, et celle en assembleur en particulier.

UTILISATION DES ROUTINES

Nous en arrivons au niveau inférieur de la programmation, la base en quelque sorte : les routines elles-mêmes.

Avant d'écrire une routine, prenez une feuille de papier et inscrivez les sous-routines appelées par votre routine, puis celles qui l'appellent au contraire (combien y en a-t-il, quelles sont les différences au niveau de l'appel, etc.). Ce travail fait, vous pourrez en déduire les caractéristiques de votre routine.

Ces caractéristiques sont essentiellement les états d'entrée et de sortie de la routine. Consultez ainsi le Chapitre 8, où sont données toutes les routines système : pour chacune d'entre elles il est précisé quels sont les paramètres à lui fournir, quels sont les registres modifiés et quelles sont les valeurs de sortie (états des flags notamment).

Ce qui vaut pour les routines système vaut aussi pour les autres. Donc pour chaque routine vous devrez vous demander : *primo*, quels sont les paramètres à fournir, et dans quel(s) registre(s) ; *secundo*, quels sont les registres qu'il faut conserver, ceux que l'on peut modifier (et dans ce cas quels sont ceux qu'il faudra alors empiler et dépiler) ; *tertio*, quels renseignements la routine doit fournir en sortie et dans quels registres ou flags.

Comment exécuter ce travail ? En réfléchissant !

Ainsi, pour les paramètres. Il est important, dans votre routine, de distinguer des valeurs fixes les valeurs qui seront variables selon la routine appelante. Seules ces dernières devront éventuellement être transmises comme paramètres, c'est évident. Dès lors, une nouvelle question se pose : où transmettre ces paramètres ? Vous avez le choix entre les registres ou la mémoire et, parmi les registres, bien des possibilités se présentent.

Tout d'abord, cela dépend du contexte des routines appelantes. S'il s'agit, par exemple, de contrôler la valeur d'un code qui vient d'être rentré au clavier, votre routine doit admettre ce code dans A ; c'est là en effet qu'il se trouve habituellement après l'appel des routines système de scrutation du clavier.

Ensuite, cela dépend de la façon dont la routine sera exécutée. Ainsi, si votre routine a pour but d'écrire une chaîne alphanumérique dont l'adresse doit être fournie comme paramètre, il est naturel de fournir cette adresse dans HL, car les instructions relatives à (HL) sont globalement plus nombreuses et plus rapides que celles relatives à (DE) ou (BC). Par contre, si la chaîne doit en plus être écrite à un endroit variable, précisé lui aussi comme paramètre, le mieux est sans doute de mettre cette position dans HL (pour l'appel direct de LOCATE en entrée), puis d'écrire la chaîne après un *EX DE,HL*.

Quant à la longueur de la chaîne, bien des choses sont possibles. Si les chaînes sont chacune utilisées une seule fois au cours du programme, on pourra en transmettre la longueur dans un registre (B en général, pour utiliser *DJNZ*) ; mais si l'une au moins est utilisée plusieurs fois à des endroits différents, mieux vaut en stocker la longueur avec, soit de manière directe (le premier octet est la longueur, solution choisie dans l'annuaire du Chapitre 7), soit de manière indi-

recte (bit 7 du dernier code à un, ou encore un 0 pour finir). On pourrait ainsi multiplier les exemples à l'infini (il y en a déjà beaucoup avec les programmes de ce livre).

Pour ce qui est des registres conservés, là encore examinez les possibilités. Vous devez regarder si les routines appelantes ont en commun, tout au moins pour plusieurs d'entre elles, la nécessité de conserver tel ou tel registre lors de l'appel de votre routine. Si ce n'est pas le cas, ne vous cassez pas la tête : si une seule routine appelante doit sauver HL, par exemple, vous ferez

PUSH HL/ CALL ROUTINE/ POP HL

Sinon, si HL, par exemple, doit être sauvé dans deux ou trois routines appelantes au moins, vous devrez vérifier : *primo* si HL est modifié par la routine (sinon, pas de problème !) ; *secundo*, s'il l'est, si vous ne pouvez pas faire autrement ; *tertio* si vous pouvez sauver HL sur la pile sans problème. Ce dernier point n'est en effet pas forcément évident s'il existe plusieurs sorties dans la routine (retours conditionnels, etc.). Un autre problème fréquent se pose lorsqu'il faut conserver A mais changer F (flags mis en sortie pour indiquer des erreurs, etc.). Là encore, vérifiez que vous ne pouvez pas utiliser un autre registre pour stocker A provisoirement.

Les états de sortie sont souvent les plus simples à gérer. Certains sont très usuels. Ainsi l'état de la retenue indique souvent la réussite ou l'échec d'une opération (voir routines système), éventuellement corroboré par le flag z, ou par la valeur de A.

Ces états de sortie, et en particulier les flags, peuvent même justifier l'instauration d'une routine utilisée une seule fois. Par exemple, admettons que, après un INPUT, vous deviez vérifier que le code rentré (qui se trouve dans A) est bien un chiffre (entre 0 et 9) ou une virgule. Pour cela, il sera commode de créer la routine suivante :

CP “,”/ SCF/ RET Z/ CP “9” + 1/ RET NC/ CP “0”/ CCF/ RET

En sortie, la retenue sera mise si le code est celui de la virgule (première comparaison) ou s'il est compris entre celui de 0 et celui de 9, donc si la vérification est positive, sinon elle sera nulle.

Bien sûr, si une telle vérification ne doit être exécutée qu'une fois, il semble superflu d'en faire une routine. Mais si vous remplacez tous les codes *RET Z*, etc., par des *JR Z,ERREUR*, etc., vous allez allonger le programme et aussi son exécution, surtout si cette vérification se

fait dans une boucle. Ne parlons pas de l'allongement résultant si le traitement d'erreur est loin et qu'il faut donc mettre des sauts absolus, ou bien si vous devez vérifier que vous avez affaire à une lettre (quatre comparaisons au moins), etc.

Au total, l'instauration de cette routine, avec son état de sortie très simple, sera bien plus fructueuse.

QUELQUES FAUTES A NE PAS FAIRE. CONCLUSION

Il existe un certain nombre de fautes qui n'empêchent pas un programme de tourner mais qui n'en sont pas moins graves, surtout à répétition. Elles alourdissent les programmes ou les rendent incompréhensibles.

Une hérésie fréquente se produit à la fin des sous-routines, c'est l'obsession du code *RET* final qui mène à trouver ceci : *CALL PRINT/ RET* par exemple. Il faut évidemment mettre *JP PRINT* directement. Vous évitez ainsi un aller-retour totalement inutile sur la pile.

Dans le même ordre d'idées, si dans un programme vous avez trois routines et que la première fait un saut sur la troisième, supprimez le saut et déplacez la troisième pour qu'elle soit à la suite de la première (qui n'aura alors plus aucun code final).

Il est fréquent qu'une même routine soit appelée deux fois de suite (saut de ligne sur l'imprimante, par exemple), et ce à plusieurs endroits du programme. Remplacez alors les *CALL ROUTINE/ CALL ROUTINE* par un *CALL ROUTIN2*, et juste avant l'entrée de votre routine placez la ligne suivante :

```
ROUTIN2: CALL ROUTINE/ ROUTINE:....
```

L'appel de *ROUTIN2* équivaudra à deux appels de *ROUTINE*, et vous gagnerez beaucoup d'octets dans le programme. De même, il est fréquent qu'une même routine, admettant un certain paramètre, soit appelée un grand nombre de fois avec une certaine valeur de ce paramètre. Ainsi, si vous disposez d'une routine *PRCHAIN* qui écrit des chaînes alphanumériques et que l'une d'entre elles (par exemple un message d'erreur d'adresse *ERROR*) est plus courante que les autres, il sera fructueux de faire une sorte de routine-préfixe :

```
PRCHAIN0: LD HL,ERROR/ PRCHAIN:....
```

qui vous évitera d'avoir à faire *LD HL,ERROR* avant chaque appel de *PRCHAIN* en cas d'erreur.

Ce sont là les fautes les plus fréquentes. Il serait évidemment impossible de les répertorier toutes. Les exemples du Chapitre 6 vous indiqueront parfois quelle erreur est à éviter dans leur cas particulier.

Pour finir, rappelons qu'il est bien évident qu'on ne peut pas vraiment parvenir à une programmation ou à une optimisation idéale (il faudrait être sûr qu'elle existe !). Mais ce chapitre tenait simplement à éviter les écueils majeurs et à vous aider à prendre des habitudes qui vous permettront de programmer bien plus facilement et avec beaucoup moins d'erreurs.

6. EXEMPLES DE PROGRAMMES

Nous allons à présent voir quelques exemples simples et courts, en application de ce que nous avons vu dans les chapitres précédents. Les programmes qui suivent ont été choisis soit pour leur intérêt propre, soit pour leur valeur d'exemple d'application, souvent pour les deux. Ils sont extrêmement détaillés par de longues explications dans le texte et par de nombreux commentaires dans leur listing. Il va néanmoins sans dire que la compréhension de chacun sera facilitée si vous avez la curiosité de les essayer.

Il vous faudra pour cela les retaper intégralement, à l'exception des commentaires qu'il est inutile de réécrire ; cela ne vous prendra que quelques minutes avec un Assembleur. Ces programmes sont tous au point, et en principe il n'est pas possible de les planter (sauf indication contraire). Toutefois, il est prudent de les sauver avant de les lancer. Vous risquez en effet de devoir tout recommencer si un plantage se produit, en particulier à cause d'une erreur de copie. Dans un tel cas vérifiez : *primo*, que vous avez rentré toutes les lignes ; *secundo*, que toutes les étiquettes et tous les labels sont exacts (y compris ceux qui suivent les sauts et les appels), ainsi que toutes les données numériques. Si après nouvelle vérification le problème persiste, revoquez toutes les instructions une à une.

Chacun de ces exemples prend un paragraphe à lui seul. Ces paragraphes sont décomposés en trois sous-paragraphes : le premier présente le programme et décrit son utilisation, c'est-à-dire comment s'en servir une fois assemblé ; le deuxième donne le fonctionnement général du programme, en particulier ses structures internes ; le troisième s'attache à la description précise des routines du programme. Ces sous-paragraphes sont précisément classés dans l'ordre naturel de conception d'un programme : d'abord concevoir l'idée du programme, savoir ce que l'on veut en faire ; puis en imaginer la structure interne ; enfin le réaliser, routine par routine.

Tous ces programmes utilisent abondamment les routines système (décrites au Chapitre 8) afin de maintenir une compatibilité maximale entre les trois CPC. Néanmoins, certaines routines ont des adresses différentes selon les machines. Possédant un 6128, ce sont les adresses valables pour ce dernier qui sont dans le listing ; celles pour les 464, et éventuellement pour les 664, sont indiquées sur la même ligne ou à la ligne suivante, dans un commentaire.

De plus, pour les routines de l'arithmétique entière, la mention avec "ENTIERS" en commentaire signifie que vous devez charger le programme "ENTIERS" avant de lancer le vôtre (il est listé en fin de cha-

pitre) ou, si vous préférez, retaper les routines appelées ; seuls les possesseurs de 464 sont dispensés de cet effort : il leur suffira d'appeler la routine système dont l'adresse est indiquée, puisque les 464 possèdent l'arithmétique entière en ROM.

Dans les explications des routines, les numéros entre parenthèses se réfèrent aux lignes correspondantes du listing.

■ EXEMPLE DE RSX : PGCD

PRÉSENTATION ET UTILISATION

Ce très court programme va nous donner un exemple très simple de communication entre BASIC et assembleur. Son but est de calculer le PGCD de deux entiers. Rappelons que le PGCD (Plus Grand Commun Diviseur) de deux nombres est l'entier le plus grand qui soit diviseur et de l'un et de l'autre. Ainsi, le PGCD de 4 et 6 est 2, car 2 divise 4 ($4/2 = 2$, entier) et 6 ($6/2 = 3$, entier), et c'est le plus grand nombre qui soit dans ce cas (3 ne divise pas 4, 4 ne divise pas 6, etc.). Le PGCD de 7 et 9 vaut 1 (7 et 9 sont dits premiers entre eux), etc.

Le calcul de PGCD n'est pas en soi une chose exaltante (quoiqu'il puisse servir dans certaines applications mathématiques, et en particulier la détermination de nombres premiers), mais il a l'avantage de la simplicité. Il se calcule par l'algorithme d'Euclide, exposé au paragraphe suivant. Pour ce qui est de l'utilisation du PGCD, voici comment faire.

Soit A et B par exemple, les deux nombres dont vous voulez connaître le PGCD (ce doit être des entiers, ou des réels inférieurs à 32767). Après avoir initialisé une seule fois la RSX (en faisant CALL &9E00), vous devez choisir la variable où le résultat sera placé. Faites par exemple C% = 0 pour cela (une initialisation de la variable est nécessaire). Puis appelez le programme, comme ceci : |PGCD, @C%, A,B. Le premier paramètre doit en effet être l'adresse du résultat (n'oubliez pas le symbole @), les deux autres, les nombres dont on veut calculer le PGCD (dans n'importe quel ordre). Pour voir s'afficher le résultat, faites PRINT C%.

Le calcul du PGCD en langage machine est instantané, quelles que soient les valeurs d'entrée.

Exemple d'utilisation : après un CALL &9E00 après l'assemblage, faites :

C% = 0: INPUT "Donnez un nombre:";A: |PGCD, @C%, A, 18:
 PRINT C%

Cette ligne vous demandera un nombre, puis affichera le PGCD de ce nombre avec 18.

FONCTIONNEMENT

```

1          ;=====
2          ;==  Extension d'instructions  ==
3          ;==          *** PGCD ***          ==
4          ;=====
5
6
7          ORG 9E00H
8          LOAD 9E00H
9
10
11         ROMS: EQU 0B900H
12         INTRSX: EQU 0BCD1H
13         MOD: EQU 0A56CH          ; AVEC "ENTIERS"
14         ;464: MOD: EQU 0BDBBH
15
16         ; INTEGRATION DE L'EXTENSION.
17
18 9E00 01099E          LD BC,RSX
19 9E03 21139E          LD HL,NOYAU
20 9E06 C3D1BC          JP INTRSX          ; INTEGRER L'EXTENSION
21 9E09 0E9E          RSX: DEFW NOM          ; EMLACEMENT DU NOM
22 9E0B C3179E          JP PGCD          ; ADRESSE D'EXECUTION
23 9E0E 504743          NOM: DEFB "PGC"
24 9E11 C4          DEFB "D"+80H          ; DERN. LETTRE,BIT7=1
25 9E12 00          DEFB 0          ; DERNIER NOM.
26          NOYAU: DEFS 4          ; PLACE POUR NOYAU
27
28
29 9E17 FE03          PGCD: CP 3          ; TROIS PARAMETRES?
30 9E19 2022          JR NZ,ERREUR          ; NON,"SYNTAX ERROR"
31 9E1B DD6E00          LD L,(IX+0)
32 9E1E DD6601          LD H,(IX+1)          ; PREMIER NOMBRE
33 9E21 DD5E02          LD E,(IX+2)
34 9E24 DD5603          LD D,(IX+3)          ; DEUXIEME NOMBRE
35 9E27 DD4E04          LD C,(IX+4)
36 9E2A DD4605          LD B,(IX+5)          ; ADRESSE DU RESULTAT
37 9E2D C5          PUSH BC
38 9E2E EB          BOUCLE: EX DE,HL          ; RESTE DANS HL
39 9E2F D5          PUSH DE
40 9E30 CD6CA5          CALL MOD          ; DIVISION MODULO.
41 9E33 D1          POP DE
42 9E34 7C          LD A,H
43 9E35 B5          OR L          ; RESULTAT NUL?
44 9E36 20F6          JR NZ,BOUCLE          ; RESTE<>0, CONTINUER.
45 9E38 E1          POP HL          ; ADRESSE DU RESULTAT
46 9E39 73          LD (HL),E
47 9E3A 23          INC HL
48 9E3B 72          LD (HL),D          ; SAUVER RESULTAT.
49 9E3C C9          RET          ; FIN.
50
51 9E3D CD00B9          ERREUR: CALL ROMS          ; BRANCHER BASIC
52 9E40 1E02          LD E,2          ; ERREUR 2
53 9E42 7B          LD A,E          ; 6128 ET 664 SEULEMENT
54 9E43 C355CB          JP 0CB55H          ; 464:CA94H
55
56          END

```

DEBUT: 9E00H
FIN: 9E45H

VERIFICATION POUR CHARGEUR BASIC:

SOMME: 7769
PRODUIT: 300838

L'algorithme d'Euclide utilisé dans ce programme est basé sur deux nombres n et p , dont les valeurs initiales sont égales aux deux nombres dont on veut le PGCD : faire la division euclidienne de n par p ($n = p * q + r$, q quotient, r reste) ; si $r = 0$, alors p est le PGCD, sinon, faire $p = r$, $n = p$ et recommencer.

Prenons l'exemple du calcul du PGCD de 4 et 6. On divise 4 par 6 : $4 = 0 * 6 + 4$; comme $4 \neq 0$, on pose $p = 4$ et $n = 6$; à présent, $6 = 1 * 4 + 2$; $2 \neq 0$, donc $p = 2$, $n = 4$; $4 = 2 * 2 + 0$: le reste est enfin nul, le PGCD est donc 2.

ROUTINES

Initialisation de la RSX

Pour bien comprendre cette routine (18 à 26), on se reportera au Chapitre 4. Notons que le nom de la routine est PGCD et que l'addition de 80H à la dernière lettre revient à mettre son bit 7 à 1. Rappelons d'autre part que le pseudo-opérateur DEFW LABEL consiste à placer la valeur du label en mémoire (deux octets, avec l'inversion usuelle du Z80).

Début

Rappelons qu'à l'entrée de la routine IX pointe sur les paramètres et A contient le nombre de paramètres transmis (appel du BASIC). Cette dernière valeur est testée dès le début (29). Comme il faut trois paramètres (adresse du résultat + n et p initiaux), si elle diffère de 3 on passe à la routine d'erreur (voir ci-après) (30). Après ce test on récupère les paramètres : ayant été stockés par empilement, on récupère les nombres d'abord (31-34), puis l'adresse (36), laquelle est sauvée sur la pile (37). Au premier appel de la boucle, les deux nombres se trouvent dans DE et HL.

Boucle et fin

A l'entrée de la boucle (38), n se trouve dans DE, et p dans HL. On commence par en inverser les positions pour que n soit dans HL (38), puis on exécute la division (40). Notez que, le quotient n'ayant aucun intérêt, on a appelé la routine MOD de l'arithmétique. DE, c'est-à-dire p, est sauvé sur la pile (39), puis restitué (41). Après la division, le reste r se trouve dans HL et p dans DE. On teste r (42-43). S'il n'est pas nul, il faut continuer (44) ; dans ce cas, r devient p et p devient n, et l'on se trouve bien dans la situation du début de la boucle (p dans HL et n dans DE).

Si le reste est nul, alors le PGCD est p. Il ne reste qu'à récupérer l'adresse du résultat (45) et à l'y placer (46-48) avant de retourner au BASIC (49).

Erreur

En cas d'erreur (nombre de paramètres incorrect), on branche le BASIC (51), puis on place le numéro de *Syntax error*, soit 2, dans E pour les 464, dans A pour les 664 et 6128 (52-53) avant de passer à la routine de traitement d'erreur (voir Annexe C).

■ DIAGRAMMES EN BARRES

PRÉSENTATION ET UTILISATION

Les diagrammes en barres sont d'utilisation fréquente pour toutes sortes de graphiques d'économie, de gestion, etc. Naturellement, ces diagrammes exigent justement de tracer des barres. Or celles-ci comportent chacune neuf traits (voyez la Figure 2), qui exigent au moins un lever de crayon. Tracer une barre en BASIC exige donc au moins onze instructions graphiques, sans compter celles destinées à assurer à la barre sa forme. Il s'agit donc d'une chose pénible et fort longue si l'on veut dessiner tout un graphique de quelques dizaines de barres.

Le programme suivant est destiné à éviter cette corvée. Il n'a pas été réalisé comme extension d'instruction (mais il ne tient qu'à vous de le faire). Le tracé d'une barre s'obtient par l'ordre

CALL &9F00, x, y, h, l, a,

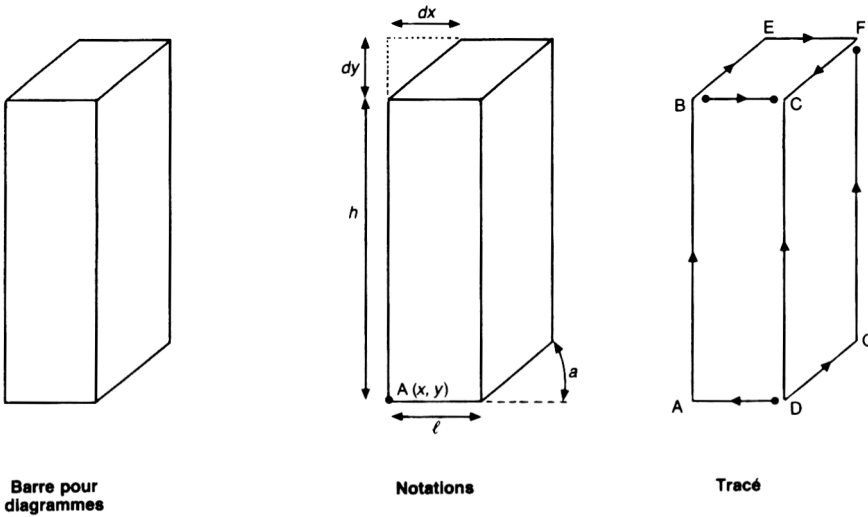


Figure 2.

où x et y sont les coordonnées du coin inférieur droit (point A sur la Figure 2), l la largeur de la barre, h sa hauteur, et a l'angle que font ses fuyantes avec l'horizontale. Toutes ces notations sont précisées sur la Figure 2.

Une seule barre est ainsi tracée, mais beaucoup plus rapidement qu'elle ne le serait en BASIC, et ainsi le tracé d'une vingtaine de barres à partir du BASIC, mais avec ce programme, ne prend qu'une ou deux secondes.

FONCTIONNEMENT

```

1          ;=====
2          ;==  *** TRACE DE BARRES ***  ==
3          ;==  *** POUR DIAGRAMMES ***  ==
4          ;=====
5
6
7          ORG 9F00H
8          LOAD 9F00H
9
10
11         ROMS: EQU 0B900H
12         MOVE: EQU 0BBC0H
13         DRAW: EQU 0BBF9H
14         COPY: EQU 0BD61H      ; 464: BD3DH, 664: BD5EH
15         IR: EQU 0BD64H        ; 464: BD40H, 664: BD61H
16         RI: EQU 0BD6AH        ; 464: BD46H, 664: BD67H
17         PROD: EQU 0BD85H      ; 464: BD61H, 664: BD82H
18         SIN: EQU 0BDACH       ; 464: BD88H, 664: BDA9H
19         COS: EQU 0BDAFH       ; 464: BD8BH, 664: BDACH

```

```

20          ICHSGN: EQU 0A5BFH          ;AVEC "ENTIER".
21          ;464: ICHSGN: EQU 0BDC7H
22
23
24 9F00 FE05   BARRE: CP 5          ;5 PARAMETRES?
25 9F02 C2E09F JP NZ,ERREUR          ;NON,"SYNTAX ERROR".
26 9F05 ED73EB9F LD (TABLE+2),SP      ;SAUVER SP.
27 9F09 DDF9   LD SP,IX          ;PILE SUR PARAMETRES
28 9F0B E1     POP HL          ; ANGLE a
29 9F0C 22ED9F LD (TABLE+4),HL      ;SAUVER PROVISOIIRT.
30 9F0F E1     POP HL          ;LARGEUR 1
31 9F10 22E99F LD (TABLE),HL
32 9F13 7C     LD A,H          ;OCTET FORT DE 1
33 9F14 D1     POP DE          ;HAUTEUR h
34 9F15 E1     POP HL          ;y,COORD Y DU POINT A.
35 9F16 C1     POP BC          ;x,COORD X DU POINT A.
36 9F17 ED7BEB9F LD SP,(TABLE+2)      ;RESTITUER SP.
37 9F1B CB17   RL A          ;LARGEUR POSITIVE?
38 9F1D DADC9F JP C,ERREUR2         ;NON,"IMPROPRER ARG."
39 9F20 C5     PUSH BC          ;SAUVER x
40 9F21 CB7A   BIT 7,D        ;HAUTEUR POSITIVE?
41 9F23 C4D59F CALL NZ,HNEG         ;NON,TENIR COMPTE
42 9F26 ED53EF9F LD (TABLE+6),DE      ;SAUVER h
43 9F2A E5     PUSH HL          ;y
44 9F2B 11F69F LD DE,ACC2
45 9F2E 2AED9F LD HL,(TABLE+4)      ;ANGLE a
46 9F31 CD64BD CALL IR             ;ANGLE EN REEL DANS ACC2
47 9F34 EB     EX DE,HL        ;ACC2 DANS DE
48 9F35 21FB9F LD HL,ACC3
49 9F38 CD61BD CALL COPY           ;ANGLE a DANS ACC3
50 9F3B CDAFBD CALL COS            ;COS(a) DANS ACC3
51 9F3E 11F19F LD DE,ACC1
52 9F41 E5     PUSH HL          ;ACC3
53 9F42 2AE99F LD HL,(TABLE)       ;RECUPERER 1
54 9F45 CD64BD CALL IR             ;LARGEUR EN REEL DANS ACC1
55 9F48 EB     EX DE,HL        ;ACC1 DANS DE
56 9F49 E1     POP HL          ;ACC3
57 9F4A D5     PUSH DE
58 9F4B CD85BD CALL PROD           ;dx=1*COS(a)
59 9F4E CD6ABD CALL RI             ;EN ENTIER
60 9F51 22EB9F LD (TABLE+2),HL      ;SAUVER dx
61 9F54 21F69F LD HL,ACC2
62 9F57 CDACBD CALL SIN            ;SIN(a) DANS ACC2
63 9F5A D1     POP DE          ;ACC1
64 9F5B CD85BD CALL PROD           ;dy=1*SIN(a)
65 9F5E CD6ABD CALL RI             ;EN ENTIER
66 9F61 22ED9F LD (TABLE+4),HL      ;SAUVER dy
67
68 9F64 C1     POP BC          ;VALEUR DE y
69 9F65 D1     POP DE          ;VALEUR DE x
70 9F66 2AE99F LD HL,(TABLE)       ;1
71 9F69 19     ADD HL,DE        ;x+1
72 9F6A E5     PUSH HL
73 9F6B C5     PUSH BC          ;y
74 9F6C 2AEF9F LD HL,(TABLE+6)      ;HAUTEUR h
75 9F6F 09     ADD HL,BC        ;y+h
76 9F70 CDC0BB CALL MOVE           ;POINT (x,y+h),SOIT B.
77 9F73 210000 LD HL,0
78 9F76 ED5BE99F LD DE,(TABLE)       ;1
79 9F7A CDF9BB CALL DRAWR         ;TRACER SEGMENT BC
80 9F7D E1     POP HL          ;y
81 9F7E D1     POP DE          ;x+1
82 9F7F CDC0BB CALL MOVE           ;POINT (x+1,y),SOIT D.
83 9F82 2AE99F LD HL,(TABLE)       ;1
84 9F85 CDBFA5 CALL ICHSGN        ; -1
85 9F88 110000 LD DE,0
86 9F8B D5     PUSH DE
87 9F8C EB     EX DE,HL
88 9F8D CDF9BB CALL DRAWR         ;SEGMENT DA
89 9F90 D1     POP DE          ;0
90 9F91 D5     PUSH DE
91 9F92 2AEF9F LD HL,(TABLE+6)      ;h
92 9F95 CDF9BB CALL DRAWR         ;SEGMENT AB
93 9F98 ED5BEB9F LD DE,(TABLE+2)      ;dx

```

```

94 9F9C 2AED9F      LD  HL, (TABLE+4)  ;dy
95 9F9F CDF9BB      CALL DRAWR         ;SEGMENT BE
96 9FA2 ED5BE99F    LD  DE, (TABLE)   ;1
97 9FA6 E1          POP  HL           ;0
98 9FA7 E5          PUSH HL
99 9FA8 CDF9BB      CALL DRAWR         ;SEGMENT EF
100 9FAB 2AEB9F     LD  HL, (TABLE+2) ;dx
101 9FAE E5          PUSH HL
102 9FAF CDBFA5     CALL ICHSGN        ; -dx
103 9FB2 EB         EX  DE, HL
104 9FB3 2AED9F     LD  HL, (TABLE+4) ;dy
105 9FB6 E5          PUSH HL
106 9FB7 CDBFA5     CALL ICHSGN        ; -dy
107 9FBA CDF9BB     CALL DRAWR         ;SEGMENT FC
108 9FBD 2AEF9F     LD  HL, (TABLE+6) ;h
109 9FC0 CDBFA5     CALL ICHSGN        ; -h
110 9FC3 110000     LD  DE, 0
111 9FC6 CDF9BB     CALL DRAWR         ;SEGMENT CD
112 9FC9 E1          POP  HL           ;dy
113 9FCA D1          POP  DE           ;dx
114 9FCB CDF9BB     CALL DRAWR         ;SEGMENT DG
115 9FCE 2AEF9F     LD  HL, (TABLE+6) ;h
116 9FD1 D1          POP  DE           ;0
117 9FD2 C3F9BB     JP  DRAWR         ;SEGMENT GF ET FIN.
118
119 9FD5 19          HNEG:  ADD  HL, DE      ;x=x-h
120 9FD6 EB         EX  DE, HL
121 9FD7 CDBFA5     CALL ICHSGN        ;h POSITIF
122 9FDA EB         EX  DE, HL
123 9FDB C9          RET
124
125 9FDC 1E05        ERREUR2: LD  E, 5          ;ERREUR 5
126 9FDE 1802        JR  ERREUR+2        ;"IMPROPRER ARGUMENT"
127 9FE0 1E02        ERREUR: LD  E, 2          ;"SYNTAX ERROR"
128 9FE2 CD00B9      CALL ROMS           ;BRANCHER BASIC
129 9FE5 7B          LD  A, E            ;POUR 6128 ET 664
130 9FE6 C355CB      JP  0CB55H         ;464:CA94H
131
132
133          TABLE:  DEFS 8          ;PLACE POUR ENTIERS.
134          ACC1:   DEFS 5          ;ACCUMULATEUR 1
135          ACC2:   DEFS 5          ;ACCUMULATEUR 2
136          ACC3:   DEFS 5          ;ACCUMULATEUR 3
137
138          END

```

```

DEBUT: 9F00H
FIN: 9FFFH

```

VERIFICATION POUR CHARGEUR BASIC:

```

SOMME: 38829
PRODUIT: 4446418

```

Le fonctionnement de ce programme est très simple, presque primaire. Il s'agit simplement de tracer ces neuf traits. Pour cela on est obligé de lever au moins une fois le "crayon". Ici, on a choisi de tracer d'abord le segment du haut BC (voir figure), puis tous les autres à la suite en partant de D et en arrivant à F, dans cet ordre : D, A, B, E, F, C, D, G, F à nouveau.

Les segments sont tracés par la routine DRAWR, tous les calculs sont très simples et basés sur des entiers, une fois que dx et dy sont

connus. Ces deux valeurs (voir figure) sont calculées au début, comme valant $l \cdot \cos(a)$ et $l \cdot \sin(a)$.

Si vous le souhaitez, vous pouvez simplifier ultérieurement le programme pour éviter ces calculs de cosinus et sinus, en donnant à l'angle a une valeur constante ; pour savoir comment faire, reportez-vous à l'explication relative au cercle.

Le programme nécessite une table de huit octets, pour y stocker quatre entiers (l , dx , dy et h) et trois accumulateurs de 5 octets pour le calcul de dx et dy .

ROUTINES

Barre

Dès le début, on vérifie que cinq paramètres ont bien été transmis, faute de quoi on passe à la routine d'erreur (24-25). Ensuite, on récupère ces paramètres, avec l'un des actes décrits au Chapitre 5. Il consiste à sauver le SP (26), ici dans une partie inutilisée au début de la table, puis à le charger avec IX (adresse des paramètres) (27), enfin à dépiler les paramètres les uns après les autres, avant de restituer le SP (36). Comme il y a cinq paramètres, les deux premiers, a et l , sont sauvés dans la table. Pour ce qui est des trois autres (x , y , et h), ils sont placés dans les registres (33-35). Avant de commencer, il est nécessaire de vérifier que l est bien positif. Pour cela, on charge A avec son octet le plus fort (32), et l'on fait passer le bit 7 de cet octet (bit de signe) dans la retenue (37). Si ce bit est non nul, on passe à la routine d'erreur (38).

A présent, on sauve sur la pile x (39). Pour y , il faut d'abord vérifier si h est positif (40). Si ce n'est pas le cas, il faudra prendre pour y la valeur $y+h$ et rendre h positif (41). Ensuite, on sauve y sur la pile à son tour (43) et h dans la table (42).

On passe alors au calcul de dx et dy , qui valent respectivement $l \cdot \cos(a)$ et $l \cdot \sin(a)$. Ces opérations ne pouvant se faire que sur des réels, il faut d'abord transformer a et l en réels, le premier dans l'accumulateur flottant 2 (44-46) et 3, car on l'utilise deux fois (47-49), le second dans l'accumulateur 1 (53-54). On calcule le cosinus de a (50) et on le multiplie par l (55-58) avant de le transformer en un entier dx (59), qui est placé dans la table (60). Notez les opérations de pile et d'échange, destinées à éviter les chargements type *LD HL,ACC1* qui prennent trois octets chacun. Pour le calcul de dy , même

principe, le sinus est calculé sur ACC2, où a se trouve encore à ce moment (61-62), la suite est identique (63-66).

Le dessin commence alors. On calcule, après avoir dépilé x et y (68-69), les coordonnées de D (qui valent $x+l$ et y) et on les empile (70-73), puis celles de B (qui valent x et $y+h$; voir figure pour tout cela) (74-75) où le curseur graphique est positionné (76), avant de tracer le segment BC (77-79). Ensuite on restitue les coordonnées de D empilées (80-81) afin d'y placer le curseur graphique (82) avant de tracer les uns à la suite des autres tous les segments restants par des DRAWR. Des changements de signe sont parfois nécessaires. Notez ici encore les circulations sur la pile, qui ont le même but que précédemment. L'exécution du programme s'achève sur un dernier tracé (117).

Hneg

Cette petite routine est utilisée si h est négatif en entrée. Il faut alors en changer le signe (120-122), et changer aussi y, le coin inférieur droit ayant pour ordonnée $y+h$ si $h < 0$ (119).

Erreur2 et erreur

La routine d'erreur est la même que pour PGCD. Erreur2 permet de sortir l'erreur n° 5 (*Improper argument* ; cas d'une largeur négative).

■ EXEMPLE DE RSX GRAPHIQUE ET ARITHMÉTIQUE : ÉTOILES

PRÉSENTATION ET UTILISATION

Le petit programme BASIC ci-dessous permet d'obtenir le tracé d'une étoile de taille et de nombre de branches variables. On peut difficilement le qualifier de léger ! Beaucoup moins en tout cas que ce simple ordre : |ETOILE, n, r1, où n est le nombre de branches souhaité, et r1 le rayon extérieur des branches (moitié du plus grand diamètre de l'étoile ; voir Figure 3). Par cet ordre, à condition d'avoir assemblé le programme suivant et d'en avoir initialisé au moins une fois la RSX (par CALL &A000), vous obtiendrez un tracé pratiquement instantané d'étoile.

```

100 '=====Trace d'etoiles
110 DEG:CLS
120 INPUT "Combien de branches ";n
130 PRINT:INPUT "Taille (1-200) ";r1
140 CLS:ORIGIN 320,200:MOVE 0,r1
150 r2=r1*COS(360/n)/COS(180/n)
160 a=0
170 FOR i=1 TO 2*n
180 IF i MOD 2=0 THEN r=r1 ELSE r=r2
190 a=a+180/n
200 DRAW r*SIN(a),r*COS(a)
210 NEXT
220 END

```

L'étoile tracée sera centrée sur la position courante du curseur graphique (changée par MOVE en BASIC) qui est restituée en fin de programme. Le nombre de branches doit être compris entre 4 et 127 (inclus). Par exemple, après assemblage et un CALL &A000, faites :

```

MODE 2: MOVE 320,200: FOR R = 10 TO 190 STEP 10: |ETOILE,
5, R: NEXT

```

et appréciez...

FONCTIONNEMENT

```

1          ;=====
2          ;==  Extension d'instructions  ==
3          ;==      ***  ETOILE  ***      ==
4          ;=====
5
6
7          ORG  0A000H
8          LOAD 0A000H
9
10         INTRSX: EQU  0BCD1H
11
12         ; INTEGRATION DE L'EXTENSION.
13
14  A000 0109A0          LD  BC,RSX
15  A003 2115A0          LD  HL,NOYAU
16  A006 C3D1BC          JP   INTRSX          ; INTEGRER L'EXTENSION
17  A009 0EA0          RSX: DEFW NOM          ; EMPLACEMENT DU NOM
18  A00B C337A0          JP   ETOILE          ; EXECUTION
19  A00E 4554F49 NOM:  DEFB  "ETOIL"
19  A012 4C
20  A013 C5          DEFB  "E"+80H          ; DERN. LETTRE: BIT7 =1.
21  A014 00          DEFB  0          ; 0 POUR FINIR
22          NOYAU:  DEFS  4          ; PLACE POUR NOYAU.
23
24
25         ROMS:  EQU  0B900H
26         MOVE:  EQU  0BBC0H
27         POS?:  EQU  0BBC6H
28         ORIG:  EQU  0BBC9H
29         ORIG?: EQU  0BBCCH
30         DRAW:  EQU  0BBF6H
31         COPY:  EQU  0BD61H          ; 464:BD3DH,664:BD5EH
32         IR:    EQU  0BD64H          ; 464:BD40H,664:BD61H
33         RI:    EQU  0BD6AH          ; 464:BD43H,664:BD64H

```

```

34          SOM:      EQU  ØBD7CH          ;464:BD58H,664:BD79H
35          PROD:    EQU  ØBD85H          ;464:BD61H,664:BD83H
36          DIV:     EQU  ØBD88H          ;464:BD64H,664:BD85H
37          DEG:     EQU  ØBD97H          ;464:BD73H,664:BD94H
38          SIN:     EQU  ØBDACH          ;464:BD88H,664:BDA9H
39          COS:     EQU  ØBDAFH          ;464:BD8BH,664:BDACH
40          ICHSGN:  EQU  ØA5BFH          ;AVEC "ENTIERS".
41          ;464: ICHSGN: EQU ØBDC7H
42
43          ACC1:    DEFS 5                ;ACCUMULATEUR 1
44          ACC2:    DEFS 5                ;ACCUMULATEUR 2
45          ANGØ:    DEFS 5                ;CONSTANTE 18Ø/n
46          ANG1:    DEFS 5                ;ANGLE VARIABLE
47          RAY1:    DEFS 5                ;CONSTANTE r1
48          RAY2:    DEFS 5                ;CONSTANTE r2
49
50
51 AØ37 FEØ2      ETOILE: CP 2            ;DEUX PARAMETRES ?
52 AØ39 C231A1   JP NZ,ERREUR           ;NON,"SYNTAX ERROR".
53 AØ3C CD97BD   CALL DEG                ;MODE DEGRES
54          ;RECUPERATION DES PARAMETRES
55 AØ3F 112DAØ   LD DE,RAY1
56 AØ42 DD66Ø1   LD H,(IX+1)
57 AØ45 DD6EØØ   LD L,(IX+Ø)                ;r1,DERNIER PARAMETRE
58 AØ48 CD64BD   CALL IR                ;METTRE r1 DANS RAY1,EN REEL
59 AØ4B DD7EØ2   LD A,(IX+2)            ;n,NOMBRE DE POINTES
60 AØ4E FEØ3     CP 3                    ;AU MOINS 3 POINTES ?
61 AØ5Ø DA2DA1   JP C,ERREUR2           ;NON,"IMPROPRER ARG."
62 AØ53 4F      LD C,A
63 AØ54 37      SCF
64 AØ55 CB11    RL C                    ;C=2*n+1
65 AØ57 3EØØ    LD A,Ø                ;CONSERVER RETENUE
66 AØ59 47      LD B,A                ;COMPTEUR i A ZERO DANS B
67 AØ5A DD8EØ3   ADC A,(IX+3)          ;NON NUL SI n>127
68 AØ5D C22DA1   JP NZ,ERREUR2        ;ALORS,"IMPROPRER..."
69          ;PLACEMENT DE L'ORIGINE AU CENTRE
70 AØ6Ø CDCCBØ   CALL ORIG?
71 AØ63 D5      PUSH DE
72 AØ64 E5      PUSH HL                ;CONSERVER L'ORIG. ACTUELLE
73 AØ65 C5      PUSH BC                ;SAUVER LE COMPTEUR
74 AØ66 E5      PUSH HL
75 AØ67 D5      PUSH DE
76 AØ68 CDC6BØ   CALL POS?            ;POSITION RELATIVE
77 AØ6B C1      POP BC                 ;COORD. X DE L'ORIGINE
78 AØ6C EB      EX DE,HL
79 AØ6D Ø9      ADD HL,BC              ;COORD X ABSOLUE
80 AØ6E EB      EX DE,HL
81 AØ6F C1      POP BC                 ;COORD Y DE L'ORIGINE
82 AØ7Ø Ø9      ADD HL,BC              ;COORD Y ABSOLUE
83 AØ71 CDC9BØ   CALL ORIG            ;ORIGINE AU CENTRE
84          ;CALCUL DE 18Ø/n ET r2
85 AØ74 1119AØ   LD DE,ACC1
86 AØ77 26ØØ    LD H,Ø
87 AØ79 DD6EØ2   LD L,(IX+2)            ;n
88 AØ7C D5      PUSH DE                ;ACC1
89 AØ7D D5      PUSH DE                ;ACC1
90 AØ7E CD64BD   CALL IR                ;n EN REEL DANS ACC1
91 AØ81 21ØØØØ   LD HL,Ø
92 AØ84 1128AØ   LD DE,ANG1
93 AØ87 CD64BD   CALL IR                ;ANNULER a
94 AØ8A 1123AØ   LD DE,ANGØ
95 AØ8D 21B4ØØ   LD HL,18Ø
96 AØ9Ø CD64BD   CALL IR                ;18Ø DANS ANGØ,EN REEL
97 AØ93 D1      POP DE                ;ACC1
98 AØ94 CD88BD   CALL DIV                ;(ANGØ)=18Ø/n
99 AØ97 EB      EX DE,HL              ;ANGØ DANS DE
100 AØ98 E1     POP HL                ;ACC1
101 AØ99 CD61BD   CALL COPY            ;18Ø/n DANS ACC1
102 AØ9C D5      PUSH DE                ;ANGØ
103 AØ9D CD7CBD   CALL SOM                ;(ACC1)=18Ø/n+18Ø/n=36Ø/n
104 AØAØ CDAFBØ   CALL COS                ;(ACC1)=COS(36Ø/n)
105 AØA3 D1     POP DE                ;ANGØ
106 AØA4 E5     PUSH HL                ;ACC1
107 AØA5 211EAØ   LD HL,ACC2

```

```

108 A0A8 CD61BD      CALL COPY                ;(ACC2)=(ANG0)=180/n
109 A0AB CDAFBD      CALL COS                 ;COS(180/n) DANS ACC2
110 A0AE EB          EX DE,HL                ;ACC2 DANS DE
111 A0AF E1          POP HL                  ;ACC1
112 A0B0 CD88BD      CALL DIV                 ;COS(360/n)/COS(180/n)
113 A0B3 112DA0      LD DE,RAY1
114 A0B6 D5          PUSH DE
115 A0B7 CD85BD      CALL PROD               ;OBTENTION DE r2 DANS ACC1
116 A0BA EB          EX DE,HL                ;ACC1 DANS DE
117 A0BB 2132A0      LD HL,RAY2
118 A0BE CD61BD      CALL COPY                ;r2 DANS RAY2
119                  ;PLACEMENT INITIAL
120 A0C1 E1          POP HL                  ;r1
121 A0C2 CD6ABD      CALL RI                  ;r1 EN ENTIER
122 A0C5 110000      LD DE,0
123 A0C8 CDC0BB      CALL MOVE                ;MOVE 0,r1
124
125                  ;TRACE DES SEGMENTS.
126
127 A0CB C1          ROUTINE: POP BC          ;i DS B,2*n+1 DS C
128 A0CC 78          LD A,B                  ;VALEUR DU COMPTEUR i
129 A0CD 3C          INC A                    ;i=i+1
130 A0CE B9          CP C                     ;ARRIVE A 2*i+1 ?
131 A0CF 284E       JR Z,FIN                 ;OUI, TERMINE
132 A0D1 47          LD B,A                   ;RESTOCKER i
133 A0D2 C5          PUSH BC                  ;ET SAUVER LE TOUT.
134 A0D3 CB18       RR B                     ;i PAIR ?
135 A0D5 112DA0      LD DE,RAY1              ;OUI,r=r1
136 A0D8 3003       JR NC,SUITE
137 A0DA 1132A0      LD DE,RAY2              ;NON,r=r2
138 A0DD 211EA0      SUITE: LD HL,ACC2
139 A0E0 E5          PUSH HL
140 A0E1 CD61BD      CALL COPY                ;r DANS ACC2
141 A0E4 2128A0      LD HL,ANG1              ;(ANG1)=ANGLE a
142 A0E7 1123A0      LD DE,ANG0
143 A0EA CD7CBD      CALL SOM                 ;a=a+180/n
144 A0ED 1119A0      LD DE,ACC1
145 A0F0 EB          EX DE,HL
146 A0F1 CD61BD      CALL COPY                ;a DANS ACC1
147 A0F4 CDACBD      CALL SIN                 ;SIN(a) DANS ACC1
148 A0F7 D1          POP DE                  ;ACC2
149 A0F8 CD14A1      CALL PRODSGN             ;x=r*SIN(a)
150 A0FB E5          PUSH HL                 ;COORD x DU POINT COURANT
151 A0FC 2119A0      LD HL,ACC1
152 A0FF 1128A0      LD DE,ANG1
153 A102 CD61BD      CALL COPY                ;ENCORE a DANS ACC1
154 A105 CDAFBD      CALL COS                 ;COS(a) DANS ACC1
155 A108 111EA0      LD DE,ACC2              ;a
156 A10B CD14A1      CALL PRODSGN             ;y=r*COS(a)
157 A10E D1          POP DE                  ;x DANS DE,y DANS HL
158 A10F CDF6BB      CALL DRAW                ;TRACER LA LIGNE
159 A112 18B7       JR ROUTINE              ;A LA SUIVANTE
160
161 A114 CD85BD      PRODSGN: CALL PROD         ;FAIRE LE PRODUIT
162 A117 CD6ABD      CALL RI                  ;CONVERTIR EN ENTIER
163 A11A 17          RLA                      ;SIGNE DANS RETENUE
164 A11B D0          RET NC                  ;POSITIF,OK.
165 A11C C3BFA5      JP ICHSGN               ;NEGATIF,TENIR COMPTE.
166
167 A11F 110000      FIN: LD DE,0
168 A122 210000      LD HL,0
169 A125 CDC0BB      CALL MOVE                ;SE REPLACER AU CENTRE
170 A128 E1          POP HL
171 A129 D1          POP DE                  ;COORD. ANCIENNE ORIGINE
172 A12A C3C9BB      JP ORIG                 ;RESTITUER ANCIENNE ORIG.
173
174 A12D 1E05       ERREUR2: LD E,5          ;ERREUR 5
175 A12F 1802       JR ERREUR+2             ;"IMPROPRER ARGUMENT"
176 A131 1E02       ERREUR: LD E,2          ;"SYNTAX ERROR"
177 A133 CD00B9      CALL ROMS                ;BRANCHER BASIC
178 A136 7B          LD A,E                   ;POUR 6128 ET 664
179 A137 C355CB      JP 0CB55H               ;464:CA94H
180
181                  END

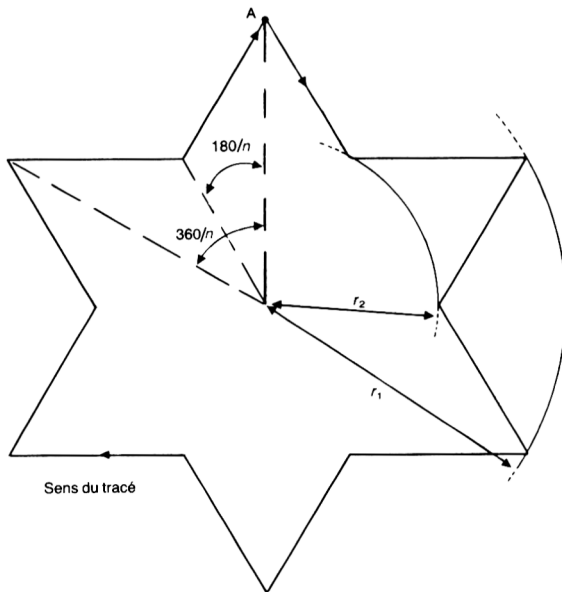
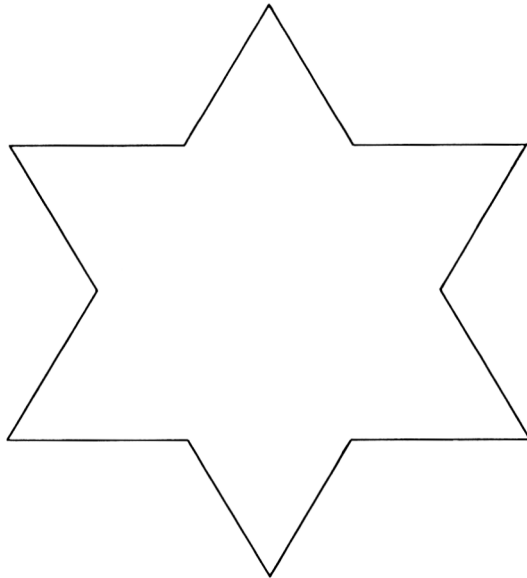
```

DEBUT: A000H
FIN: A139H

VERIFICATION POUR CHARGEUR BASIC:

SOMME: 36431
PRODUIT: 6292731

Étoile (ici à 6 branches)



Notations

Figure 3.

Le fonctionnement de ce programme est assez élémentaire ; il se différencie peu, en fait, du programme BASIC correspondant. Le nombre relativement élevé de lignes s'explique par la présence d'une RSX à initialiser, de nombreuses étiquettes (50 lignes rien qu'avec cela !), mais aussi de calculs arithmétiques dus à la valeur un peu compliquée de r2 (rayon intérieur de l'étoile, voir figure), donné par la formule

$$r2 = r1 * \text{COS}(360/n) / \text{COS}(180/n)$$

Cette formule est calculée pour que les parties rentrantes des branches de l'étoile soient alignées deux à deux.

r2 étant calculé, on trace de point en point. Les angles séparant deux points consécutifs, vus du centre, valent 180/n, et les distances au centre valent alternativement r1 et r2.

Le programme utilise cinq accumulateurs à virgule flottante : ACC1 et ACC2 pour les intermédiaires de calcul, ANG0 pour y placer la constante 180/n, ANG1 où se trouve l'angle a, enfin RAY1 et RAY2 où se trouvent les deux rayons r1 et r2.

ROUTINES

Intégration de l'extension

On se reportera à PGCD et au Chapitre 4 pour cette routine. Le nom de l'instruction est ETOILE.

Récupération des paramètres

Après avoir vérifié que deux paramètres avaient bien été transmis (51-52) et s'être placé en mode degrés (à cause de 180/n) (53), les deux paramètres r1 et n sont récupérés. Tout d'abord r1, qui est aussitôt transformé en réel placé dans RAY1 (55-58), puis le nombre de branches n. Ce nombre est testé : il doit être supérieur à 3 (60-61), sinon l'étoile n'existe pas, et inférieur à 128 afin que l'on puisse stocker la valeur maximale du compteur i, soit $2*n + 1$, dans un seul octet, ici C (62-64). Pour vérifier que c'est bien le cas, on additionne son bit 7, mis dans la retenue (par 64), et les bits 8 à 15, contenus dans (IX+3), à zéro (65 et 67). Si le résultat n'est pas nul, c'est qu'un de ces bits ne l'est pas, donc que n est trop grand (68). Sinon, tout va

bien : n est alors toujours dans $(IX + 2)$, le compteur i , contenu dans B , est initialisé à zéro, et sa valeur maximale est dans C .

Placement de l'origine au centre

Le programme exigeant que l'origine des axes graphiques se trouve au centre de l'étoile, on commence par demander la position de l'origine courante et par la sauver, ainsi que le compteur BC (70-73). Ensuite, on demande la position du curseur graphique, centre de l'étoile (76), et on ajoute ses coordonnées à celles de l'origine, stockées une deuxième fois sur la pile (74-75), afin d'en avoir la position absolue (c'est-à-dire par rapport au coin inférieur droit de l'écran) (77-82). Il ne reste plus qu'à placer l'origine en ce point (83).

Calcul de $180/n$ et $r2$

On récupère n dans HL (86-87) en tenant compte du fait qu'il tient que sur un octet (86), puis on le transforme en un réel stocké dans l'accumulateur 1 (85-90). Ensuite, après avoir initialisé l'angle a à zéro comme en BASIC (91-93), on place 180 en réel dans $ANG0$ (94-96), et l'on exécute la division par n (97-98) afin que $ANG0$ contienne bien $180/n$. Cette valeur est alors recopiée dans $ACC1$ (100-101) et additionnée à elle-même (102-103) afin d'obtenir $360/n$, dont le cosinus est alors calculé (104). Ensuite on recopie $180/n$ dans $ACC2$ (105-108) afin d'en calculer le cosinus (109), puis d'exécuter la division pour obtenir $COS(360/n)/COS(180/n)$ dans $ACC1$ (112). Il ne reste plus qu'à multiplier par $r1$ (113-115) pour avoir $r2$ (formule ci-dessus). Cette valeur est alors copiée dans $RAY2$ (116-118).

Placement initial

On reconvertit ici $r1$ en un entier (120-121), la valeur initiale de IX étant perdue, et l'on place le curseur graphique en haut de l'étoile, en $(0,r1)$ (122-123). Le tracé peut alors commencer.

Tracé des segments

Le compteur est restitué (127) et incrémenté (128). S'il n'atteint pas encore sa valeur maximale de $2*n + 1$ stockée dans C , on continue le tracé, sinon, l'on passe à la fin (130-131). Avant de tracer, on sauve le compteur (132-133).

On détermine d'abord pour le tracé si r vaut $r1$ ou $r2$, en fonction de la parité du compteur (134-137).

Cela fait, on place r dans ACC2 pour les calculs (140). Ensuite, on ajoute $180/n$ à l'angle a (141-143), puis on en calcule le sinus après l'avoir placé dans ACC1 (144-147). On multiplie alors par r et l'on obtient la première coordonnée $r*\sin(a)$ (148-149), sauvee sur la pile (150). Même travail avec $r*\cos(a)$ (151-156), la seconde coordonnée. Il ne reste qu'à restituer la première (157) et à tracer un trait (158) avant de recommencer la boucle (159).

Prodsgn

Cette routine exécute le produit entre les deux réels dont les adresses lui sont fournies (161), puis transforme le résultat en un entier (162). Mais RI donne le signe de l'entier dans A (bit 7) et sa valeur absolue dans HL. Donc si ce signe est positif tout va bien (163-164), sinon il faut changer le signe de HL, positif à tort (165).

Fin

Le curseur graphique est replacé au centre de l'étoile, sa position initiale (167-169). Puis les coordonnées de l'origine initiale sont restituées (170-171) et l'origine est replacée (172). Ici s'arrête le programme.

Erreurs

Comme pour la barre.

■ UNE EXTENSION D'INSTRUCTION GRAPHIQUE : CERCLE RAPIDE

PRÉSENTATION ET UTILISATION

Une des figures géométriques les plus employées est sans aucun doute le cercle, symbole de perfection.

Cette perfection, on en est loin si l'on souhaite en tracer en BASIC ; il faut faire :

```
DEG: MOVE R,0: FOR T = 5 TO 360 STEP 5: DRAW R*COS(T),  
R*SIN(T): NEXT
```

R désignant le rayon du cercle. On a ainsi un dessin suffisamment proche du cercle réel (compte tenu de l'épaisseur des pixels), mais quelle lenteur ! De fait, on a ainsi 72 instructions DRAW et 144 calculs trigonométriques exécutés, et c'est la valeur minimale (un pas de plus de 5 degrés donne un résultat peu circulaire). En réalité, le tracé est même très rapide, vu le travail nécessaire (car le BASIC Amstrad est très performant).

Pour arriver à un tracé presque instantané des cercles, il faut recourir à l'assembleur, bien évidemment sous la forme d'une RSX.

Je dois reconnaître que la présentation d'une RSX de cercle n'est guère originale. Cela étant, sous prétexte que l'assembleur est plus rapide que le BASIC, les auteurs des nombreuses versions de cette RSX que j'ai eu l'occasion de voir (environ une demi-douzaine) se sont contentés plus ou moins de réécrire en assembleur le programme BASIC ci-dessus. Or le gain de temps est alors faible ; en effet, ce qui est le plus long, outre les sorties graphiques (obligatoires), ce sont les calculs de polynômes de cosinus et sinus, qui sont pratiquement les routines les plus lentes du système d'exploitation. Et ce calcul n'est pas plus rapide en assembleur qu'en BASIC, d'où un gain relativement faible.

Or ces calculs, comme il est expliqué au paragraphe suivant, ne sont aucunement nécessaires. De fait, le programme dont le listing est donné ne fait qu'une seule sorte de calculs : des produits sur des entiers, qui sont très rapides, et c'est tout. Mystère expliqué au paragraphe suivant.

De ce fait, le tracé est pratiquement instantané, même pour des cercles de très grand rayon (le programme accepte jusqu'à 511).

Concrètement, l'utilisation est très simple : pour obtenir un cercle de rayon R centré au point de coordonnées (X,Y) (ces coordonnées doivent être calculées par rapport au coin inférieur droit de l'écran, non par rapport à l'origine en cours), faites : |CERCLE, R, X, Y (après avoir, comme toujours, initialisé au moins une fois la RSX par CALL &A200).

FONCTIONNEMENT

```

1          ;=====
2          ;==  Extension d'instructions  ==
3          ;==          CERCLE          ==
4          ;=====
5
6          ORG 0A200H
7

```

```

8                                LOAD 0A200H
9
10
11                               INTRSX: EQU 0BCD1H
12
13                               ; INTEGRATION DE L'EXTENSION.
14
15 A200 0109A2                    LD BC,RSX
16 A203 2115A2                    LD HL,NOYAU
17 A206 C3D1BC                    JP INTRSX ; INTEGRER L'EXTENSION
18 A209 0EA2                      RSX: DEFW NOM
19 A20B C319A2                    JP CERCLE
20 A20E 43455243                 NOM: DEFB "CERCL"
21 A212 4C
22 A213 C5                        DEFB "E"+080H
23 A214 00                        DEFB 0
24
25                               NOYAU: DEFS 4 ; PLACE POUR NOYAU
26
27                               ROMS: EQU 0B900H
28                               MOVE: EQU 0BB00H
29                               ORIG: EQU 0BBC9H
30                               ORIG?: EQU 0BBCCH
31                               DRAW: EQU 0BBF6H
32                               IPRODABS: EQU 0A53BH ; AVEC "ENTIERS"
33                               ;464: IPRODABS: EQU 0BDBEH
34                               ICHSGN: EQU 0A5BFH ; AVEC "ENTIERS"
35                               ;464: ICHSGN: EQU 0BDC7H
36
37                               ; INITIALISATION
38 A219 FE03                      CERCLE: CP 3 ; 3 PARAMETRES?
39 A21B 207A                      JR NZ,ERREUR ; NON, "SYNTAX ERROR"
40 A21D CDC0BB                    CALL ORIG?
41 A220 D5                         PUSH DE
42 A221 E5                         PUSH HL ; COORD ORIGINE ACTUELLE
43 A222 DD6E02                    LD L,(IX+2)
44 A225 DD6603                    LD H,(IX+3)
45 A228 DD5E04                    LD E,(IX+4)
46 A22B DD5605                    LD D,(IX+5) ; COORD. DU CENTRE
47 A22E CDC9BB                    CALL ORIG ; ORIGINE AU CENTRE
48 A231 DD5601                    LD D,(IX+1)
49 A234 DD5E00                    LD E,(IX+0) ; RAYON R DU CERCLE
50 A237 3EFE                      LD A,0FEH
51 A239 A2                        AND D ; RAYON < 512 ?
52 A23A 2057                      JR NZ,ERREUR2 ; NON, "IMPROPRER ARG. "
53 A23C ED53A0A2                 LD (RAYON),DE ; STOCKER
54 A240 210000                    LD HL,0
55 A243 CDC0BB                    CALL MOVE ; SE PLACER EN (R,0).
56 A246 0E00                      LD C,0 ; INITIALISER COMPTEUR.
57
58                               ; QUART PAR QUART
59
60 A248 0612                      TRACER: LD B,18 ; 18 SEGMENTS
61 A24A DD21A2A2                 LD IX,TABLE ; EMLACEMENT DES NOMBRES
62 A24E CD78A2                   QUART: CALL CALCUL ; R*COS(T)
63 A251 CB49                      BIT 1,C ; SIGNE DU COSINUS?
64 A253 C4BFA5                   CALL NZ,ICHSGN ; RENDRE NEGATIF
65 A256 E5                        PUSH HL ; SAUVER
66 A257 CD78A2                   CALL CALCUL ; R*SIN(T)
67 A25A 0C                        INC C
68 A25B CB49                      BIT 1,C ; SIGNE DU SINUS?
69 A25D C4BFA5                   CALL NZ,ICHSGN ; RENDRE NEGATIF
70 A260 0D                        DEC C ; RESTITUER COMPTEUR
71 A261 D1                        POP DE ; RECUPERER COSINUS
72 A262 CB41                      BIT 0,C ; ECHANGE NECESSAIRE?
73 A264 C491A2                   CALL NZ,ECHANGE ; OUI, LE FAIRE
74 A267 C5                        PUSH BC ; SAUVER LES COMPTEURS.
75 A268 CDF6BB                   CALL DRAW ; TRACER
76 A26B C1                        POP BC
77 A26C 10E0                      DJNZ QUART ; TRACER TOUT LE QUART
78 A26E 0C                        INC C ; NUMERO QUART SUIVANT
79 A26F CB51                      BIT 2,C ; INFRIEUR A 4?
80 A271 28D5                      JR Z,TRACER ; OUI, CONTINUER

```

```

81 A273 E1          POP HL
82 A274 D1          POP DE          ;COORD ORIGINE ANCIENNE
83 A275 C3C9BB      JP ORIG          ;RESTITUER ORIGINE ET FIN.
84
85
86 A278 2600        CALCUL: LD H,0
87 A27A DD6E00      LD L,(IX+0)          ;COSINUS OU SINUS
88 A27D DD23        INC IX
89 A27F ED5BA0A2    LD DE,(RAYON)      ;RAYON R.
90 A283 CD3BA5      CALL IPRODABS      ;MULTIPLIER NBRES >0.
91 A286 7C          LD A,H
92 A287 2600      LD H,0
93 A289 CB15        RL L
94 A28B CB17        RL A
95 A28D CB14        RL H
96 A28F 6F          LD L,A          ;DIVISER HL PAR 128
97 A290 C9          RET
98
99 A291 EB          ECHANGE: EX DE,HL
100 A292 C9         RET
101
102
103 A293 1E05        ERREUR2: LD E,5          ;ERREUR 5
104 A295 102        JR ERREUR+2      ;"IMPROPRER ARGUMENT"
105 A297 1E02        ERREUR: LD E,2          ;"SYNTAX ERROR"
106 A299 CD00B9     CALL ROMS          ;BRANCHER BASIC
107 A29C 7B          LD A,E          ;POUR 6128 ET 664
108 A29D C355CB     JP 0CB55H         ;464:CA94H
109
110
111 RAYON:           DEFS 2
DEFB
TABLE:
113 A2A2 800B       DEFB 128,11
114 A2A4 7E16       DEFB 126,22
115 A2A6 7C21       DEFB 124,33
116 A2A8 782C       DEFB 120,44
117 A2AA 7436       DEFB 116,54
118 A2AC 6F40       DEFB 111,64
119 A2AE 6949       DEFB 105,73
120 A2B0 6252       DEFB 98,82
121 A2B2 5B5B       DEFB 91,91
122 A2B4 5262       DEFB 82,98
123 A2B6 4969       DEFB 73,105
124 A2B8 3F6F       DEFB 63,111
125 A2BA 3674       DEFB 54,116
126 A2BC 2C78       DEFB 44,120
127 A2BE 217C       DEFB 33,124
128 A2C0 167E       DEFB 22,126
129 A2C2 0B80       DEFB 11,128
130 A2C4 0080       DEFB 0,128
131
132 END

```

DEBUT: A200H
FIN: A2C5H

VERIFICATION POUR CHARGEUR BASIC:

SOMME: 22528
PRODUIT: 2154475

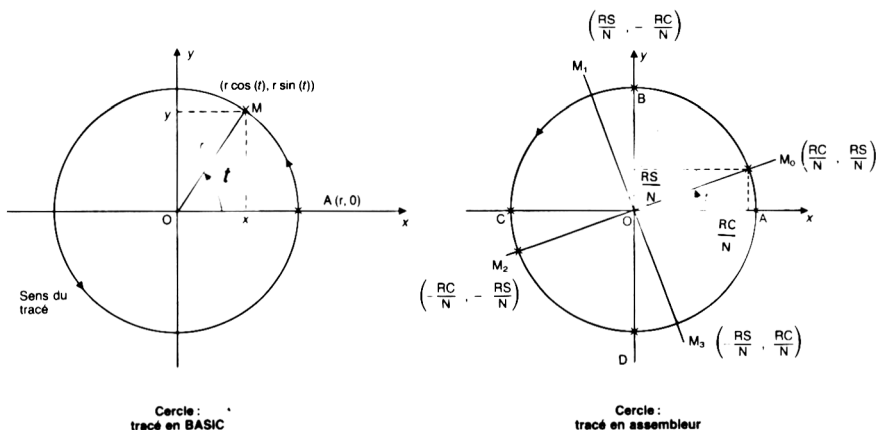


Figure 4.

Donc expliquons comment on peut éviter tous les calculs sur des réels, et particulièrement les calculs de cosinus et sinus.

En réalité c'est fort simple. Regardez le programme BASIC ci-dessus. Vous constatez que les cosinus et sinus calculés sont toujours les mêmes : ce sont ceux des angles de 5 à 360 degrés (de 5 en 5) ; ils ne dépendent nullement des caractéristiques du cercle (ce qui n'était pas le cas pour les étoiles, où les angles dépendaient de n). D'où l'idée qui vient naturellement à l'esprit : les calculer une seule fois et les placer en mémoire.

Toutefois, compte tenu du fait qu'un réel prend cinq octets en mémoire, cela risque de faire beaucoup d'octets utilisés. Dès lors, deuxième idée : les stocker en tant qu'entiers. Cela n'est pas possible directement (leur partie entière est presque toujours nulle), mais l'on peut stocker des valeurs types $N \cdot \text{COS}(T)$, où N est un entier assez grand pour que l'on puisse considérer la différence $N \cdot \text{COS}(T) - \text{ROUND}(N \cdot \text{COS}(T))$ comme négligeable au moins 100. Dès lors, pour calculer $R \cdot \text{COS}(T)$, il suffira de calculer $(R \cdot C) / N$. Les valeurs $C = \text{ROUND}(N \cdot \text{COS}(T))$ étant stockées en mémoire, il ne s'agit plus en effet que de calculs sur des entiers.

Un problème, néanmoins, se pose alors : quelle valeur donner à N ? La réponse est quadruple. *Primo*, on l'a dit, N doit être assez grand (au moins 100), faute de quoi l'imprécision relative au fait d'arrondir $N \cdot \text{COS}(T)$ à l'entier le plus proche risque d'être trop importante. *Secundo*, N doit être inférieur à 256, afin de permettre de stocker les valeurs $C = \text{ROUND}(N \cdot \text{COS}(T))$ et $S = \text{ROUND}(N \cdot \text{SIN}(T))$ sur un

seul octet chacune, pour gagner de la place. *Tertio*, N ne doit pas être grand non plus car $R*N$ doit rester sur 16 bits, donc être inférieur à 65535, ce qui limite les valeurs possibles pour R ; ainsi, pour $N=200$, R ne devrait pas dépasser $65535/200$ soit 327, ce qui est assez limitatif. *Quarto*, comme il faut diviser par N dans le programme, et souvent, il vaut mieux que cette division soit la plus facile possible ; or les divisions les plus simples en binaire sont celles par les puissances de 2 (simples décalages de bits).

D'où la réponse à ce rébus : $N=128$. C'est le seul nombre qui réponde aux quatre critères cités. On a donc placé en fin de programme une table comprenant les valeurs des cosinus et sinus des angles de 5 à 90 degrés, multipliés par 128.

Pourquoi seulement jusqu'à 90 degrés, ce qui ne permet que le tracé du premier quart du cercle ?

Réponse sur la Figure 4, à droite. Vous constatez que les quatre points M_0, M_1, M_2 et M_3 , qui sont à des positions semblables dans des quadrants différents, ont des coordonnées déductibles de celles de M_0 , à savoir $(R*C/N, R*S/N)$. Elles valent en effet dans l'ordre $(-R*S/N, R*C/N)$, $(-R*C/N, -R*S/N)$, et $(R*S/N, -R*C/N)$. Les seules différences résident éventuellement en une inversion et en un changement de signe de l'une ou l'autre des valeurs $R*C/N$ et $R*S/N$. Il n'est donc pas nécessaire de stocker toutes les valeurs C et S jusqu'à 360 degrés ; il suffit de le faire jusqu'à 90 degrés et de tracer ensuite le cercle quart par quart, avec les transformations nécessaires sur les coordonnées, en fonction du numéro du quadrant.

Dressons un tableau où l'on mettra en exergue les cas où des changements sont nécessaires, en fonction du numéro q du quadrant (exprimé en binaire) :

q	q+1	Signe de $R*C/N$	Signe de $R*S/N$	Échange ?
00	001	Plus	Plus	Non
01	010	Plus	Moins	Oui
10	011	Moins	Moins	Non
11	100	Moins	Plus	Oui

Il n'est pas difficile de tirer des lois de ce tableau, où l'on a volontairement fait paraître les valeurs binaires de q+1. On constate qu'il faut changer le signe de $R*C/N$ si le bit 1 de q est non nul, qu'il faut en faire autant de $R*S/N$ si le bit 1 de q+1 est non nul, enfin qu'un échange des coordonnées est nécessaire lorsque le bit 0 de q est non nul.

Voilà la structure du programme révélée : c'est à la fois très simple et très subtil. C'est par ces astuces successives que l'on parvient à optimiser au maximum le tracé de cercles.

Notons toutefois qu'on pourrait même diviser le cercle en huit. Mais le gain de temps est alors inexistant.

Le programme, outre la table des cosinus et sinus (qui prend 36 octets), utilise un emplacement de deux octets pour stocker le rayon.

ROUTINES

Intégration de l'extension

On se référera à l'explication du PGCD et au Chapitre 4 pour l'explication de cette partie. Le nom de l'instruction est CERCLE.

Initialisation

On vérifie d'abord que trois paramètres ont bien été transmis, faute de quoi un message d'erreur est fourni (38-39). Le tracé exigeant que l'origine soit placée au centre du cercle, les coordonnées de l'origine courante sont demandées (40) et sauvées sur la pile (41-42). Ensuite on demande les coordonnées du centre (43-46) où l'origine est placée (47) ; notez que, de ce fait, les coordonnées du centre doivent être calculées par rapport au coin inférieur droit de l'écran, et non par rapport à l'origine courante ; si vous souhaitez l'inverse, faites comme dans le listing de l'étoile, lignes 70 à 83.

Ensuite, le rayon R est chargé dans DE (48-49). Il faut alors vérifier qu'il est inférieur à 512, c'est-à-dire que ses bits 9 à 15 sont nuls (donc les bits 1 à 7 de D). Pour cela on charge D dans A en annulant son bit 0 par un AND FEH (FEH = 11111110B) (50-51) ; si le résultat n'est pas nul, c'est que R est trop grand, on passe alors à la routine d'erreur (52).

Le rayon est maintenu stocké dans l'emplacement qui lui est réservé (53) et le curseur graphique est placé au point A de la Figure 4, soit le point le plus à droite du cercle, en (R,0) (54-55).

Le registre C, qui contiendra constamment le compteur de quadrant q, est initialisé à zéro (56).

Tracer

Cette boucle est destinée à tracer un quart de cercle dont le numéro est placé dans C. Elle contient elle-même une autre boucle, chaque quart comprenant dix-huit traits. Le compteur de cette deuxième boucle QUART est placé dans B. Il est initialisé à 18 (60). Le registre IX pointant sur les valeurs de C et S courantes est initialisé à chaque quart sur le début de la table (61).

La boucle QUART est très simple. Elle calcule d'abord $R \cdot C / N$ (62). Elle teste alors le bit 1 du compteur q, qui se trouve dans C (63) ; s'il n'est pas nul, le signe de $R \cdot C / N$ est inversé, pour avoir $-R \cdot C / N$ (64) ; de toute façon, le résultat est stocké sur la pile (65). On recommence alors avec $R \cdot S / N$ (66) ; ici c'est le bit 1 de q + 1 qu'il faut tester, d'où une incrémentation (67) et une restitution (70). On récupère la première valeur dans DE (71). Un échange est nécessaire si le bit 0 de q est non nul (72-73). Il ne reste plus alors qu'à tracer, en prenant soin de sauver le compteur BC (74-76), puis à recommencer ainsi dix-huit fois (77).

Le quart étant fini, il faut passer au suivant. On incrémente donc C (78) et l'on vérifie qu'il n'atteint pas la valeur 4 (en testant son bit 2) (79). Si ce n'est pas le cas, le tracé est poursuivi, quart par quart.

Si C atteint 4, le tracé est terminé. Il ne reste qu'à restituer l'origine de départ (81-83), et à finir.

Calcul

Cette subroutine est destinée à calculer $R \cdot C / N$ et $R \cdot S / N$. En entrée, IX pointe sur C ou S, qui est alors chargé dans HL (86-87) (en se rappelant que ces valeurs sont stockées sur un octet). IX est incrémenté pour pointer sur la valeur suivante (88). On charge dans DE le rayon R (89) et on multiplie (90) ; remarquez que c'est la routine IPRODABS qui a été choisie, car les nombres ici sont tous positifs : il est donc inutile de s'embarrasser de problèmes de signes.

Le résultat obtenu doit être divisé par $N = 128 = 80H$. Cela consiste simplement à en supprimer les bits 0 à 6 (décalage de 7 bits). Pour cela, on place les bits 8 à 15 dans A (91), on vide H (92), on place le bit 7 dans la retenue (93), puis dans le bit 0 de A, dont le bit 7 est placé dans la retenue (ex-bit 15, nouveau bit 8) (94), puis dans le bit 0 de H (puisque c'est le nouveau bit 8) (95). Les nouveaux bits 0 à 7 (ex-bits 7 à 14) se trouvant dans A, il faut les remettre dans L (96), CQFD, HL a été divisé par 128 en quelques microsecondes seulement.

Echange et erreur

La routine échange se passe de commentaires. Pour erreur, voyez la barre.

■ COMPILATEUR/DÉCOMPILATEUR D'ÉCRAN

PRÉSENTATION ET UTILISATION

L'écran graphique, sur Amstrad, a une très haute résolution pour un ordinateur de ce type. C'est évidemment un avantage du point de vue de la finesse des dessins, mais cela occupe 16 K de mémoire. Dès lors, lorsqu'il s'agit de sauver une image écran sur support magnétique, cela devient un inconvénient : sur cassette, le chargement est très long ; sur disquette, une face de disque peut juste stocker une dizaine d'images...

Pour remédier à cet état de choses, le programme suivant donne un compilateur/décompilateur d'écran. Par une méthode exposée au paragraphe suivant, le compilateur transforme l'écran en un fichier binaire spécial en général moins long, parfois beaucoup moins long. Ce fichier peut ensuite être sauvé sur support magnétique. Pour restituer l'image, il suffit d'appeler le décompilateur.

Ce double programme est présenté sous la forme de deux RSX qui admettent chacune un paramètre. Celui de COMPIL est l'adresse où le fichier compilé sera placé. Celui de DECOMPIL est au contraire l'adresse du fichier à décompiler.

Notez que ces adresses ne sont pas testées. Donc si vous choisissez une adresse trop proche du programme lors de la compilation, celui-ci va se recouvrir lui-même et se planter (une marge de 4000H est prudente). Si vous vous trompez d'adresse au moment de la décompilation, là encore vous risquez le plantage.

L'utilisation est donc très simple : faites |COMPIL, adresse pour compiler, et |DECOMPIL, adresse pour restituer l'image. Après une compilation, la longueur du fichier compilé (nécessaire pour le sauver) se trouve en ses deux premiers octets ; vous l'aurez donc par la formule

$$L = \text{PEEK}(\text{adresse} + 1) + 256 * \text{PEEK}(\text{adresse})$$

Le gain en % par rapport à l'image non compilée sera donné par la formule

$$G = 100 * (1 - L / 3FD0)$$

ce qui vous permettra d'apprécier les performances du compilateur (couramment 30 à 50 %). Si le gain est négatif (c'est théoriquement possible, voir paragraphe suivant), l'image a été rallongée : il faut la sauver non compilée.

Encore un détail : lors de la compilation, ne placez pas le curseur sur la dernière ligne de l'écran : cela modifierait l'OFFSET écran et l'image serait décalée lors de la décompilation.

FONCTIONNEMENT

```

1          ;=====
2          ;== *** COMPILATEUR D'ECRAN *** ==
3          ;=====
4
5
6          ORG 0A300H
7          LOAD 0A300H
8
9
10         INTRSX: EQU 0BCD1H
11
12 A300 0109A3      LD  BC,RSX
13 A303 2120A3      LD  HL,NOYAU
14 A306 C3D1BC      JP   INTRSX          ;INTEGRER 2 EXTENSIONS
15 A309 11A3        RSX:  DEFW NOM
16 A30B C324A3      JP   COMPIL          ;COMPILATEUR
17 A30E C379A3      JP   DECOMPIL       ;DECOMPILATEUR
18 A311 434F4D50    NOM:  DEFB "COMPI"
19 A315 49
20 A316 CC          DEFB "L"+80H
21 A317 4445434F    DEFB "DECOMPI"
22 A31B 4D5049
23 A31E CC          DEFB "L"+80H
24 A31F 00          DEFB 0
25 NOYAU:  DEFS 4
26
27 ROMS:  EQU 0B900H
28 ECRAN: EQU 0C000H
29 LONGUEUR: EQU 3FD0H          ;LONGUEUR ECRAN
30
31 A324 3D          COMPIL: DEC  A          ;1 PARAMETRE?
32 A325 2049        JR   NZ,ERREUR       ;NON, "SYNTAX ERROR"
33 A327 DD5601      LD  D,(IX+1)
34 A32A DD5E00      LD  E,(IX+0)        ;ADRESSE DU RESULTAT
35 A32D D5          PUSH DE
36 A32E 13          INC  DE
37 A32F 13          INC  DE          ;SAUTER DEUX OCTETS.
38 A330 2100C0      LD  HL,ECRAN
39 A333 01D03F      LD  BC,LONGUEUR
40 A336 7E          LD  A,(HL)         ;PRENDRE OCTET
41 A337 23          INC  HL
42 A338 BE          CP   (HL)         ;EGAL AU SUIVANT?
43 A339 CC53A3      CALL Z,EGAUX      ;OUI, TRAITER
44 A33C 12          LD  (DE),A

```

```

45 A33D 13          INC  DE
46 A33E 0B          DEC  BC
47 A33F CB78        BIT   7, B          ;BC <0 ?
48 A341 28F3        JR    Z, RUN          ;NON, CONTINUER
49 A343 AF          XOR   A
50 A344 12          LD   (DE), A          ;ZERO
51 A345 13          INC  DE
52 A346 12          LD   (DE), A          ;ENCORE
53 A347 13          INC  DE
54 A348 3C          INC  A                ;REPETITION 1 FOIS ...
55 A349 12          LD   (DE), A          ;...=FIN DU FICHIER
56 A34A E1          POP  HL                ;ADRESSE DEBUT
57 A34B EB          EX   DE, HL
58 A34C ED52        SBC  HL, DE            ;LONGUEUR
59 A34E EB          EX   DE, HL
60 A34F 73          LD   (HL), E
61 A350 23          INC  HL
62 A351 72          LD   (HL), D          ;SAUVER LONGUEUR
63 A352 C9          RET
64
65 A353 D5          EGAUX: PUSH DE
66 A354 1601        LD   D, 1
67 A356 5F          LD   E, A              ;SAUVER L'OCTET
68 A357 14          BCLE: INC D
69 A358 2813        JR    Z, DEPASS        ;D=256, DEPASSEMENT
70 A35A 23          INC  HL
71 A35B 0B          DEC  BC
72 A35C CB78        BIT   7, B          ;BC <0 ?
73 A35E 2004        JR    NZ, FINBCLE     ;OUI, TERMINER.
74 A360 7B          LD   A, E              ;RECUPERER OCTET
75 A361 BE          CP   (HL)              ;MEME OCTET?
76 A362 28F3        JR    Z, BCLE          ;OUI, CONTINUER.
77 A364 E3          FINBCLE: EX (SP), HL
78 A365 73          LD   (HL), E          ;SAUVER OCTET
79 A366 23          INC  HL
80 A367 73          LD   (HL), E          ;DEUX FOIS
81 A368 23          INC  HL
82 A369 7A          LD   A, D              ;ET NBRE DE REPETITIONS
83 A36A EB          EX   DE, HL            ;REMETTRE DANS DE.
84 A36B E1          POP  HL                ;ET RESTITUER HL.
85 A36C C9          RET                    ;SAUVER NBRE AU RETOUR
86
87 A36D 15          DEPASS: DEC D          ;BONNE VALEUR
88 A36E 18F4        JR    FINBCLE
89
90 A370 CD00B9       ERREUR: CALL ROMS
91 A373 1E02        LD   E, 2              ;"SYNTAX ERROR"
92 A375 7B          LD   A, E              ;6128 ET 664 SEULEMENT.
93 A376 C355CB       JP   0CB55H            ;464:CA94H
94
95
96                  ;DECOMPILATEUR
97
98 A379 3D          DECOMPIL:DEC A        ;1 PARAMETRE?
99 A37A 20F4        JR    NZ, ERREUR
100 A37C DD6601      LD   H, (IX+1)
101 A37F DD6E00      LD   L, (IX+0)          ;RECUPERER ADRESSE
102 A382 23          INC  HL
103 A383 23          INC  HL
104 A384 1100C0      LD   DE, ECRAN
105 A387 7E          RUN2: LD A, (HL)        ;PRENDRE OCTET
106 A388 23          INC  HL
107 A389 BE          CP   (HL)              ;EGAL AU SUIVANT?
108 A38A CC92A3      CALL Z, OUTMULT        ;OUI, ALORS SORTIR
109 A38D C8          RET   Z                ;TROUVE LA FIN, TERMINE.
110 A38E 12          LD   (DE), A          ;SORTIR OCTET
111 A38F 13          INC  DE
112 A390 18F5        JR    RUN2              ;CONTINUER
113
114 A392 23          OUTMULT: INC HL        ;SUIVANT
115 A393 46          LD   B, (HL)          ;NOMBRE DE REPETITIONS
116 A394 05          DEC  B                ;EGAL A 1?
117 A395 C8          RET   Z                ;OUI, ALORS FIN.
118 A396 12          OUT: LD (DE), A

```

```

119 A397 13      INC DE      ;FLAG z NON MODIFIE PAR
120 A398 10FC    DJNZ OUT    ;...CES 3 INSTRUCTIONS.
121 A39A 23      INC HL
122 A39B C9      RET        ;RETOUR ET SAUVER 1 FOIS ENC.
123
124
125              END

```

```

DEBUT: A300H
FIN: A39BH

```

VERIFICATION POUR CHARGEUR BASIC:

```

SOMME: 15671
PRODUIT: 1258988

```

Le fonctionnement de ce compilateur est basé sur la simplicité relatives des images écran. Si vous avez la curiosité d'explorer la mémoire écran, vous verrez qu'il s'y trouve fréquemment de longues séries d'octets tous semblables (des zéros en particulier) dus au fait que le fond graphique n'est jamais totalement recouvert.

L'idée est alors de remplacer une série de N octets de valeur X par les deux octets X et N. Les octets isolés seront simplement sauvés seuls.

Mais un problème se posera à la décompilation : comment le décompilateur distinguera-t-il les octets de type N (nombre de répétitions) des octets de type X ? Ces derniers prenant toutes les valeurs possibles pour un octet, il faut trouver un moyen de les distinguer.

La solution choisie est simple. Lorsqu'un octet est isolé (différent de celui qui le suit), il est sauvé tel quel. Lorsqu'une série de N octets à X est rencontrée, on sauve trois octets : X, X, N. En effet, les octets isolés sont nécessairement suivis, dans le fichier compilé, d'une valeur différente. Dès lors, si deux octets se suivant sont égaux, le décompilateur saura qu'il s'agit d'une série et il ira chercher le troisième pour savoir combien il y a d'octets semblables.

Ainsi la série de nombres suivante :

```
01 FE AB 00 00 00 00 45 79 79 79 79 88 34 10 10 42
```

sera compilée ainsi :

```
01 FE AB 00 00 05 45 79 79 04 88 34 10 10 02 42
```

soit un gain de 2 octets sur 18 (11 %) ; les nombres N ont été soulignés.

On notera une chose : les séries de deux octets sont compilées sur trois octets. Ce sont donc les seules qui représentent un allongement du fichier. De ce fait, le fichier compilé peut être plus long que le fichier de départ (échec de la compilation). En fait, cela n'arrive jamais

sur une image écran, sauf fait exprès. Par contre, cela interdit de compiler ainsi un fichier ASCII ou un programme.

ROUTINES

Intégration des extensions

Pour cette partie (10-24), on se reportera à PGCD et au Chapitre 4. Notons que deux extensions sont ici intégrées à la fois, de noms COMPIL et DECOMPIL.

Compil

Dès le début du programme, on vérifie qu'un paramètre a bien été transmis (31-32). Ce paramètre (adresse du fichier compilé) est placé dans DE (33-34) et empilé (35). DE est incrémenté deux fois, afin de laisser la place pour stocker la longueur du fichier compilé (36-37). HL est chargé avec l'adresse de l'écran, BC avec le nombre d'octets à traiter (38-39) (les 30H derniers octets, à zéro, ne sont pas comptés). Ces allocations de registres resteront les mêmes dans le programme principal : HL contient l'octet suivant à traiter. DE celle de l'emplacement suivant dans le fichier compilé, et BC le nombre d'octets restant à traiter.

La boucle RUN est très simple. L'octet à traiter est chargé dans A (40) puis comparé à celui qui le suit (41-42). En cas d'égalité, on a une série, un traitement spécial s'impose (43). Sinon, l'octet est simplement sauvé (44-45) puis BC est décrémenté. Si ce compteur n'est pas encore négatif (on teste son bit de signe pour cela (47)), on recommence la boucle (48).

Lorsque la compilation est finie, on place en fin de fichier compilé la série 00 00 01 (49-55) caractérisant la fin du fichier (en effet il ne peut y avoir de série type X X 01 dans le fichier compilé, cela signifierait une répétition une seule fois !). L'adresse du début du fichier compilé est alors récupérée (56) ; celle de la fin se trouvant dans DE, il suffit de faire la différence pour connaître la longueur (57-59, en notant que la retenue n'est pas mise en 58, à cause de 49 et 55). Cette longueur est, comme on l'a dit, stockée en début de fichier compilé (60-62). Le programme s'arrête en 63.

Égax

Cette routine traite le cas des séries d'un même octet X (qui se trouve dans A en entrée). Cet octet est placé dans E (67), après que DE a été sauvé sur la pile (65). D est alors initialisé à 1 : c'est le compteur de répétitions N. La boucle BCLE est destinée à calculer la valeur de N. Des comparaisons successives sont faites (74-75) avec les incréments nécessaires (68, 70, 71). La boucle s'interrompt dans trois cas : si le compteur D dépasse un octet (69) (il est alors restitué par *Depass*), si l'on arrive à la fin de l'écran (72-73), et naturellement si l'on atteint la fin de la série (76). Dans ces trois cas on passe à Finbcle. A cet instant, on place dans HL l'adresse courante du fichier compilé qui était sur la pile (77) et l'on place dans ce fichier deux fois la valeur X (78-81) ; puis le nombre de répétitions N est placé dans A (82) avant de redonner à DE (83) puis HL (84) leurs valeurs de pointeurs et de revenir au programme principal (85) ; ce n'est qu'au retour que N sera sauvé (44).

Depass et erreur

En cas de dépassement de D (arrive à 256), on décrémente D pour lui redonner la valeur maximale 255 et l'on passe à Finbcle. De ce fait, une série de plus de 255 octets semblables sera compilée sous la forme de plusieurs triades.

Pour erreur, voyez PGCD.

Décompilation

On vérifie d'abord qu'un paramètre a bien été transmis (98-99). Ce paramètre (adresse du fichier à décompiler) est placé dans HL (100-101). On saute alors les deux octets de longueur (102-103) et l'on place dans DE la valeur initiale de l'emplacement du décompilé, à savoir le début de l'écran (104).

La boucle RUN2 est très simple. On charge un octet et on le compare au suivant (105-107). S'il ne lui est pas égal, on le sort tout simplement sur l'écran et l'on recommence (110-112). Sinon, on passe à Outmult (voir ci-après). Lorsque cette routine donne un flag z nul en sortie, c'est que l'on est arrivé au bout du fichier : la décompilation cesse alors (109).

Outmult

Cette routine a pour but de sortir à l'écran l'équivalent des triades X X N, soit des séries de N fois l'octet X (qui se trouve dans A en entrée).

Pour cela, on saute le second X (114), puis l'on charge dans B la valeur de N (115). Celle-ci est aussitôt décrémentée (116). Si elle s'avère égale à 1, alors la fin de fichier est rencontrée (117, puis 109). Sinon, notez que le flag z restera à la même valeur (non mis) jusqu'au retour. La petite boucle Out ne sort que N - 1 fois l'octet X à l'écran, B n'ayant pas été incrémenté à nouveau (118-120), et ce parce que la dernière sortie de l'octet X se fera au retour (en 110). Avant ce retour, HL est pointé sur l'octet suivant à décompiler (121).

■ EXEMPLE D'INTERRUPTION : CHRONOMÈTRE PERMANENT

PRÉSENTATION ET UTILISATION

Les Amstrad présentent un système d'interruption très intéressant et très complexe qui permet de nombreuses choses. A titre d'exemples, c'est par elles que sont gérées les boucles EVERY du BASIC, que les chronomètres sont incrémentés, que les couleurs clignotantes sont gérées, que les touches du clavier sont testées, etc. Sur ce sujet, voyez aussi quelques détails au Chapitre 3.

Les routines système permettent trois types d'interruptions. Les plus intéressantes, à mon sens, sont celles liées au retour de spot du CRT (contrôleur vidéo), qui se produisent tous les 1/50 de seconde. En effet, elles permettent des sorties sur l'écran tout en évitant les effets désagréables que cela peut parfois produire. Toutefois les interruptions normales (1/50 de seconde aussi) peuvent être aussi utiles.

Nous allons voir ici un exemple d'interruption. Il s'agit d'un chronomètre permanent, c'est-à-dire qu'il reste en haut et à droite de l'écran obstinément tant que vous n'appellez pas la routine qui le débranche. Il est possible de le régler comme une horloge. Il donne les heures, les minutes et les secondes. Seules les lectures sur support magnétique l'interrompent un court instant, rien d'autre ne l'empêche de continuer, et en particulier aucune instruction BASIC, ce qui est éminemment précieux. Si l'écran est effacé, le chronomètre réapparaît à la seconde suivante.

L'utilisation est très simple. Un appel CALL &A400 met le chronomètre en route. Par contre l'appel CALL &A4A4 arrête le chronomètre (jusqu'à le faire reprendre par la précédente). Pour mettre le chronomètre à une heure précise, c'est à peine plus compliqué : il faut *poker* les codes ASCII des chiffres dans les cases mémoire d'adresses A00EH à A015H, en commençant par les secondes. Ainsi, pour mettre le chronomètre à 12 h, 45 mn, 10 s, il faut placer dans ces cases mémoire, dans l'ordre : 01:54:21, sans oublier les deux (:).

Dans le même genre, vous pouvez tirer grand parti des interruptions : objets mobiles, couleurs clignotant à des vitesses différentes, curseur clignotant, etc.

FONCTIONNEMENT

```

1          ;=====
2          ;==      *** CHRONOMETRE ***      ==
3          ;==      *** PERMANENT ***        ==
4          ;=====
5
6
7          ORG 0A400H
8          LOAD 0A400H
9
10
11         PRINT: EQU 0BB5AH
12         LOCATE: EQU 0BB75H
13         LOCATE?: EQU 0BB78H
14         TCRON: EQU 0BB7BH
15         TCROF: EQU 0BB7EH
16         CRVIS: EQU 0BB8AH
17         PEN: EQU 0BB90H
18         PEN?: EQU 0BB93H
19         SLIMITS: EQU 0BC17H
20         CRTEVIN: EQU 0BCD7H
21         CRTEVDEL: EQU 0BCDDH
22
23
24         ;INITALISATION DE L'EVENEMENT
25
26 A400 219CA4      LD HL, BLOC      ;ADRESSE DU BLOC
27 A403 0681        LD B, 81H      ;ASYNCHRONE, PRIORITAIRE.
28 A405 0E00        LD C, 0      ;EN RAM.
29 A407 1116A4      LD DE, CHRONO    ;ADRESSE D'EXECUTION
30 A40A C3D7BC      JP CRTEVIN     ;INITIALISER
31
32
33 A40D 00          TABLE: DEFB 0      ;EMPLACEMENT DU COMPTEUR
34 A40E 30303A30    DEFB "00:00:00"    ;CHRONO A ZERO
35 A412 303A3030
36 A416 F3          CHRONO: DI          ;PAS D'INTERRUPTIONS
37 A417 F5          PUSH AF        ;SAUVER CONTEXTE
38 A418 3A0DA4      LD A, (TABLE)    ;COMPTEUR
39 A41B 3C          INC A          ;INCREMENTER
40 A41C FE32        CP 50           ;ARRIVE A CINQUANTE?
41 A41E DA96A4      JP C, FIN        ;NON, TERMINER
42 A421 C5          PUSH BC
43 A422 D5          PUSH DE
44 A423 E5          PUSH HL        ;SAUVER CONTEXTE
45 A424 210EA4      LD HL, TABLE+1 ;PREMIER CARACTERE.
46 A427 0602        LD B, 2          ;EXECUTER 2 FOIS (SEC+MIN)
47 A429 34          COMP60: INC (HL) ;INCREMENTER UNITES

```

```

48 A42A 3E39 LD A, "9" ;VALEUR MAXIMALE
49 A42C BE CP (HL) ;DEPASSEE?
50 A42D 302A JR NC, SORTIE ;NON, OK.
51 A42F 3E30 LD (HL), "0" ;REMETTRE UNITES A 0.
52 A431 23 INC HL ;DIZAINES
53 A432 34 INC (HL) ;INCREMENTER AUSSI
54 A433 3E35 LD A, "5" ;VALEUR MAXIMALE
55 A435 BE CP (HL) ;DEPASSEE?
56 A436 3021 JR NC, SORTIE ;NON, OK.
57 A438 3E30 LD (HL), "0" ;REMETTRE DIZAINES A 0.
58 A43A 23 INC HL ;SAUTER LE (:)
59 A43B 23 INC HL ;CHIFFRE SUIVANT
60 A43C 10EB DJNZ COMP60 ;ENCORE POUR MINUTES
61 A43E 34 INC (HL) ;INCREMENTER UNITES HEURES
62 A43F 7E LD A, (HL)
63 A440 FE34 CP "4" ;ATTEINT QUATRE?
64 A442 2B0A JR Z, VERIF24 ;OUI, VERIFIER SI 24.
65 A444 FE39 CP "9" ;VALEUR MAXIMALE ATTEINTE?
66 A446 3811 JR C, SORTIE ;NON, ALORS OK.
67 A448 3E30 LD (HL), "0" ;UNITES HEURES A ZERO
68 A44A 23 INC HL ;DIZAINES D'HEURES
69 A44B 34 INC (HL) ;INCREMENTER
70 A44C 180B JR SORTIE ;ET SORTIR.
71 A44E 23 VERIF24: INC HL ;DIZAINES D'HEURES
72 A44F 7E LD A, (HL)
73 A450 FE32 CP "2" ;ATTEINT 2 (DONC 24)?
74 A452 2005 JR NZ, SORTIE ;NON, OK.
75 A454 3E30 LD A, "0" ;REMETTRE A ZERO
76 A456 77 LD (HL), A ;DIZAINES
77 A457 2B DEC HL ;ET UNITES
78 A458 77 LD (HL), A
79
80 A459 CD93BB SORTIE: CALL PEN? ;ENCRE ACTUELLE?
81 A45C F5 PUSH AF ;SAUVER
82 A45D CD8ABB CALL CRVIS ;PLACER UN CURSEUR.
83 A460 3E03 LD A, 3 ;COULEUR 3
84 A462 CD90BB CALL PEN ;... POUR ECRIRE CHRONO.
85 A465 CD78BB CALL LOCATE? ;POSITION COURANTE
86 A468 E5 PUSH HL ;SAUVER
87 A469 CD7EBB CALL TCROF ;PAS AFFICHER CURS
88 A46C CD17BC CALL SLIMITS ;TAILLE ECRAN
89 A46F 78 LD A, B ;DERNIERE COLONNE
90 A470 D608 SUB 8 ;HUIT CARACTERES
91 A472 67 LD H, A ;COTE DROIT
92 A473 2E01 LD L, 1 ;PREMIERE LIGNE
93 A475 CD75BB CALL LOCATE ;COIN SUPERIEUR DROIT
94 A478 0608 LD B, 8 ;HUIT CARACTERES
95 A47A 2115A4 LD HL, TABLE+8 ;DIZAINES D'HEURES
96 A47D 7E BCLC: LD A, (HL) ;PRENDRE CARAC.
97 A47E CD5ABB CALL PRINT ;ET L'ECRIRE
98 A481 2B DEC HL ;AU SUIVANT
99 A482 10F9 DJNZ BCLC
100 A484 E1 POP HL ;ANCIENNE POSITION CURSEUR
101 A485 CD75BB CALL LOCATE ;RESTITUER
102 A488 F1 POP AF ;ANCIENNE COULEUR ECRITURE
103 A489 CD90BB CALL PEN ;RESTITUER
104 A48C CD7BBB CALL TCROF ;REAUTORISER CURSEUR
105 A48F CD8ABB CALL CRVIS ;METTRE LE CURSEUR
106 A492 E1 POP HL
107 A493 D1 POP DE
108 A494 C1 POP BC ;RESTITUER CONTEXTE
109 A495 AF XOR A ;COMPTEUR A ZERO
110 A496 320DA4 FIN: LD (TABLE), A ;SAUVER COMPTEUR
111 A499 F1 POP AF ;RESTITUER
112 A49A FB EI ;REAUTORISER INTERRUPTIONS
113 A49B C9 RET
114
115 BLOC: DEFS 8 ;PLAS POUR BLOC EVENT
116
117 A4A4 219CA4 DESARM: LD HL, BLOC ;ADRESSE BLOC
118 A4A7 C3DDBC JP CRTEVDEL ;ENLEVE LE BLOC
119
120
121 END

```

DEBUT: A400H

FIN: A4A9H

VERIFICATION POUR CHARGEUR BASIC:

SOMME: 18642

PRODUIT: 1724106

Comme on l'a dit, tout est basé sur les interruptions. A chaque appel de la routine principale, tous les 1/50 de seconde, un compteur, placé en A00DH (premier octet de la table), est incrémenté. Tant qu'il n'atteint pas la valeur 50, rien ne se passe.

S'il l'atteint, il est remis à zéro et l'horloge est incrémentée. Pour cela on incrémente le chiffre des unités des secondes. S'il dépasse 9, on le remet à zéro et on incrémente celui des dizaines ; si celui-ci dépasse 5, on le remet aussi à zéro et on incrémente les minutes, etc.

Une fois l'horloge incrémentée, on l'écrit en haut et à droite de l'écran, dans la couleur 3. Cela exige de placer un pavé curseur à la position courante, car on est obligé de déplacer le curseur pour écrire l'horloge. De même, il faut en sauver puis en restituer la position courante et aussi la couleur d'écriture.

Comme pour toute interruption, il est nécessaire de ne modifier aucun registre ("sauver le contexte"). Le groupe 1 est donc empilé, puis dépilé.

ROUTINES

Initialisation de l'événement

On se reportera à la description de CRTEVIN au Chapitre 8 pour plus de détails. L'événement a été classé asynchrone (c'est-à-dire non soumis à une file d'attente) et prioritaire, afin qu'aucun décalage ne vienne troubler l'heure courante (27). Le bloc est en fin de programme et comprend 16 octets (26 et 115).

Incrémentation de l'horloge

Dès le début, les autres interruptions sont interdites (36) pour éviter tout problème, puis l'accumulateur est empilé (37). On y charge alors le compteur (38), qui est incrémenté (39). On le compare alors à cinquante (40). S'il n'atteint pas cette valeur, on passe à la fin (41), où le compteur est sauvé, l'accumulateur restitué et les interruptions réautorisées avant la sortie (110-113).

Si le compteur atteint 50, les autres registres sont empilés (42-44) et HL est placé sur le premier code de l'horloge, à savoir les unités des secondes (ces codes sont classés à l'envers). Ce code est incrémenté (46) ; s'il n'atteint pas sa valeur maximale de "9", on passe à la sortie à l'écran de l'horloge (48-50). Sinon, il faut remettre ce chiffre à zéro (51) et incrémenter le chiffre suivant, à savoir les dizaines de secondes (52-53). Là encore, si la valeur maximale de ce chiffre (qui est "5", car le nombre de secondes ne doit pas dépasser 59) n'est pas atteinte, on sort (54-56), sinon il faut recommencer avec les minutes ; la même routine est donc effectuée deux fois (46 et 60). Si les minutes dépassent 59 aussi, il faut incrémenter les heures.

Pour les heures, même principe, mais il faut en plus vérifier que la valeur 24 n'est pas atteinte. De ce fait, si le chiffre des unités atteint 4, on passe à Verif24 (63 – 64) après incrémentation (61), où l'on teste le chiffre des dizaines (71-73). S'il n'atteint pas "2", tout va bien (74), sinon il faut mettre les heures à zéro (75 – 78) avant de passer à la sortie.

Sortie

Cette routine écrit l'horloge à l'écran. La couleur d'écriture est d'abord sauvée (80-81), puis un curseur est placé à l'écran (pour qu'on ne voie pas que le curseur va être déplacé) (82), la couleur d'écriture est mise à trois (83-84). La position courante du curseur est empilée (85-86) et l'affichage du curseur est interdit (afin qu'on ne le voie pas pendant l'écriture) (87). Pour se placer dans le coin supérieur droit, il faut connaître le numéro de la dernière colonne (en fonction du mode) (88-89) et lui soustraire les huit caractères de l'horloge (90) avant de positionner le curseur (91-93). Il ne reste qu'à sortir, par une petite boucle, les huit caractères (94-99) en commençant par la fin, pour afficher les heures d'abord (95) et en descendant (98).

Cela fait, il faut remettre le curseur à sa place (100-101), en réautoriser l'affichage (104-105) et replacer la bonne couleur d'écriture (102-103). Enfin les registres sont restitués (106-108) et le compteur remis à zéro (109-110) avant la sortie (111-113).

Desarm

Cette routine supprime le chronomètre. Voir la description de CRTEVDEL. Le chronomètre n'est pas effacé.

■ LES CALCULS SUR LES ENTIERS POUR 664 ET 6128

Comme vous le verrez en fin de Chapitre 8, les CPC 464 sont dotés de quelques routines permettant des calculs sur des entiers, en fin de ROM inférieure. Ces routines n'existent plus sur les CPC 664 et 6128. Pour compenser cette absence, en voici le listing complet et commenté.

Il serait ici hors de propos d'expliquer les algorithmes utilisés, d'ailleurs fort simples. Je pense que les commentaires suffiront à vous les faire comprendre.

Ce listing clôt ce chapitre.

```

1          ;=====
2          ;==          ARITHMETIQUE          ==
3          ;==          AVEC ENTIERS          ==
4          ;=====
5
6
7          ORG 0A500H
8          LOAD 0A500H
9
10
11         ;METTRE LE SIGNE DANS B
12
13 A500 7C      BSIENE: LD   A, H
14 A501 B7          OR   A
15 A502 FA0BA5     JP   M, SUITE0
16 A505 B0          OR   B
17 A506 FABFA5     JP   M, ICHSGN
18 A509 37          SCF
19 A50A C9          RET
20 A50B EE80     SUITE0: XOR 080H
21 A50D B5          OR   L
22 A50E C0          RET NZ
23 A50F 78          LD   A, B
24 A510 37          SCF
25 A511 8F          ADC  A, A
26 A512 C9          RET
27
28         ;ADDITION HL=HL+DE
29
30 A513 B7          OR   A
31 A514 ED5A       ADC  HL, DE
32 A516 37          SCF
33 A517 E0          RET PO
34 A518 F6FF       OR   0FFH          ;POSITIONNER A ET c
35 A51A C9          RET
36
37         ;SOUSTRACTION HL=DE-HL
38
39 A51B EB          EX   DE, HL
40
41         ;SOUSTRACTION HL=HL-DE
42
43 A51C B7          OR   A
44 A51D ED52       SBC  HL, DE
45 A51F 37          SCF
46 A520 E0          RET PO
47 A521 F6FF       OR   0FFH
48 A523 C9          RET
49
50         ;MULTIPLICATION (AVEC SIGNE)
51

```

```

52 A524 CD30A5          CALL SIGNERES
53 A527 CD3BA5          CALL IPRODABS
54 A52A D200A5          JP NC,BSIGNE
55 A52D F6FF           OR ØFFH
56 A52F C9             RET
57
58                       ;CALCULER SIGNE DU RESULTAT
59
60 A530 7C             SIGNERES:LD A,H
61 A531 AA             XOR D
62 A532 47             LD B,A ;STOCKER SIGNE DANS B
63 A533 EB             EX DE,HL
64 A534 CDBCA5          CALL ABS
65 A537 EB             EX DE,HL
66 A538 C3BCA5          JP ABS
67
68                       ;MULTIPLICATION (SANS SIGNE)
69
70 A53B 7C             IPRODABS:LD A,H
71 A53C B7             OR A ;HL< 256? (H=Ø)
72 A53D 2805          JR Z,SUITE1 ;OK, CONTINUER
73 A53F 7A             LD A,D
74 A540 B7             OR A ;DE <256?
75 A541 37             SCF
76 A542 C0             RET NZ ;NON, DEPASSEMENT
77 A543 EB             EX DE,HL ;PLUS PETIT DANS HL
78 A544 B5             SUITE1: OR L ;HL=Ø?
79 A545 C8             RET Z ;OUI, RESULTAT Ø, FIN.
80 A546 7A             LD A,D
81 A547 B3             OR E ;DE =Ø ?
82 A548 7D             LD A,L ;MULTIPLICATEUR DANS A
83 A549 6B             LD L,E
84 A54A 62             LD H,D ;MULTIPLICANDE DANS HL
85 A54B C8             RET Z ;NUL, TERMINE
86 A54C FE03          CP 3 ;INFÉRIEUR A 3?
87 A54E 3810          JR C,CASPART ;OUI, CAS PARTICULIER
88 A550 37             SCF
89 A551 8F             BCLE0: ADC A,A ;DECALER A
90 A552 30FD          JR NC,BCLE0 ;CONTINUER JUSQ DEPASST
91 A554 29             BCLE1: ADD HL,HL ;DOUBLER HL
92 A555 D8             RET C ;DEPASSEMENT, TERMINE
93 A556 87             ADD A,A ;DECALER
94 A557 3002          JR NC,SUITE2 ;BIT NUL, BOUCLER
95 A559 19             ADD HL,DE ;AJOUTER ENCORE
96 A55A D8             RET C ;DEPASSEMENT, TERMINE
97 A55B FE80          SUITE2: CP Ø80H ;FINI?
98 A55D 20F2          JR NZ,BCLE0 ;NON, CONTINUER
99 A55F C9             RET
100 A560 FE01          CASPART: CP 1 ;MULTIPL. EGAL A 1?
101 A562 C8             RET Z ;OUI, TERMINE
102 A563 29             ADD HL,HL ;EGAL A DEUX, DONC DOUBLER
103 A564 C9             RET
104
105                       ;DIVISION (AVEC SIGNE)
106
107 A565 CD74A5          CALL DIVIS
108 A568 DA00A5          FINDIV: JP C,BSIGNE
109 A56B C9             RET
110
111                       ;DIVISION MODULO
112
113 A56C 4C             LD C,H
114 A56D CD74A5          CALL DIVIS
115 A570 EB             EX DE,HL ;RESTE DANS HL
116 A571 41             LD B,C
117 A572 18F4          JR FINDIV
118 A574 CD30A5          DIVIS: CALL SIGNERES
119
120                       ;DIVISION EUCLIDIENNE
121
122 A577 7A             LD A,D
123 A578 B3             OR E
124 A579 C8             RET Z ;QUOTIENT NUL, RETOUR
125 A57A C5             PUSH BC ;SAUVER SIGNE RESULTAT

```

```

126 A57B EB          EX  DE, HL
127 A57C 0601       LD  B, 1
128 A57E 7C         LD  A, H
129 A57F B7         OR   A
130 A580 2009       JR   NZ, SUITE3
131 A582 7A         LD  A, D
132 A583 BD         CP   L
133 A584 3805       JR   C, SUITE3
134 A586 65         LD  H, L
135 A587 2E00       LD  L, 0
136 A589 0609       LD  B, 9          ;DECALE HUIT FOIS
137 A58B 7B         SUITE3: LD  A, E
138 A58C 95         SUB  L          ;POSITIONNER RETENUE
139 A58D 7A         LD  A, D
140 A58E 9C         SBC  A, H          ;DE<HL?
141 A58F 3805       JR   C, SUITE4    ;OUI, CONTINUER
142 A591 04         INC  B          ;RETENIR UN DECALAGE
143 A592 29         ADD  HL, HL       ;NON, DECALER HL
144 A593 30F6       JR   NC, SUITE3   ;POURSUIVRE
145 A595 3F         CCF
146 A596 3F         SUITE4: CCF
147 A597 78         LD  A, B
148 A598 44         LD  B, H
149 A599 4D         LD  C, L
150 A59A 210000     LD  HL, 0
151 A59D 3D         DEC  A          ;EGAL A UN?
152 A59E 2003       JR   NZ, SUITE5   ;NON, POURSUIVRE
153 A5A0 1817       JR   FINDVEUC    ; OUI, TERMINE
154 A5A2 29         BCLE2: ADD  HL, HL  ;DECALER
155 A5A3 F5         SUITE5: PUSH AF
156 A5A4 78         LD  A, B
157 A5A5 1F         RRA
158 A5A6 47         LD  B, A
159 A5A7 79         LD  A, C
160 A5A8 1F         RRA
161 A5A9 4F         LD  C, A
162 A5AA 7B         LD  A, E
163 A5AB 91         SUB  C
164 A5AC 7A         LD  A, D
165 A5AD 98         SBC  A, B
166 A5AE 3805       JR   C, SUITE6    ;PLUS GRAND, CONTINUER
167 A5B0 57         LD  D, A
168 A5B1 7B         LD  A, E
169 A5B2 91         SUB  C
170 A5B3 5F         LD  E, A          ;RETIRER DIVISEUR
171 A5B4 2C         INC  L          ;INCREMENTER QUOTIENT
172 A5B5 F1         SUITE6: POP  AF
173 A5B6 3D         DEC  A
174 A5B7 20E9       JR   NZ, BCLE2    ;PAS FINI, POURSUIVRE
175 A5B9 37         FINDVEUC: SCF
176 A5BA C1         POP  BC
177 A5BB C9         RET
178
179          ;CALCULER VALEUR ABSOLUE
180
181 A5BC 7C         ABS:  LD  A, H
182 A5BD B7         OR   A          ; POSITIF?
183 A5BE F0         RET  P          ;OUI, TERMINE
184
185          ;CHANGEMENT DE SIGNE DE HL
186
187 A5BF AF         ICHSGN: XOR  A
188 A5C0 95         SUB  L
189 A5C1 6F         LD  L, A          ;L=-L
190 A5C2 9C         SBC  A, H
191 A5C3 95         SUB  L
192 A5C4 BC         CP   H
193 A5C5 67         LD  H, A
194 A5C6 37         SCF
195 A5C7 C0         RET  NZ
196 A5C8 FE01       CP   1          ;NUL?
197 A5CA C9         RET
198
199          ;TESTER SIGNE DE HL

```

```

200
201 A5CB 7C          LD  A,H
202 A5CC 87          ADD  A,A          ;DECALER
203 A5CD 9F          SBC  A,A
204 A5CE D8          RET  C          ;NEGATIF, TERMINE
205 A5CF B5          OR   L          ;NUL?
206 A5D0 C8          RET  Z          ; OUI, TERMINE
207 A5D1 AF          XOR  A
208 A5D2 3C          INC  A          ;METTRE 1 DANS A AVEC FLAGS
209 A5D3 C9          RET
210
211                  ;COMPARER HL ET DE
212
213 A5D4 7C          LD  A,H
214 A5D5 AA          XOR  D
215 A5D6 7C          LD  A,H
216 A5D7 F2DFA5      JP   P,SUITE7    ;MEME SIGNE, TESTER
217 A5DA 87          ADD  A,A          ;RETENUE A 1
218 A5DB 9F          FINCOMP: SBC  A,A ;POSITIONNER
219 A5DC D8          RET  C          ;NEGATIF, TERMINE
220 A5DD 3C          INC  A          ;METTRE A UN
221 A5DE C9          RET
222 A5DF BA          SUITE7: CP   D
223 A5E0 20F9        JR   NZ,FINCOMP ;DIFFERENTS, POSITIONNER
224 A5E2 7D          LD  A,L
225 A5E3 93          SUB  E
226 A5E4 20F5        JR   NZ,FINCOMP ;IDEM
227 A5E6 C9          RET          ;EGAUX.
228
229
230                  END

```

7. UN GRAND PROGRAMME : ANNUAIRE TÉLÉPHONIQUE

Voici maintenant le listing d'un grand programme : un annuaire téléphonique.

Ce programme ne peut évidemment recevoir ici des explications aussi détaillées que pour les précédents. Seules l'utilisation et quelques grandes lignes de programmation seront données. Les commentaires des lignes suffiront, j'espère, à vous faire comprendre le reste.

La version donnée ici est prévue pour disquette. Les possesseurs de lecteur de cassette pourront faire quelques adaptations (messages, etc.).

Vu la longueur du fichier, il doit être placé sous ZEN, en 3000H avant d'être sauvé mais exécuté en 9000H, et sous DAMS, il faut le diviser en plusieurs parties.

Pour ZEN, la table des symboles a été réduite afin d'éviter un dépassement.

■ UTILISATION

Ce programme gère un annuaire téléphonique formé de petits groupes (appelés références) constitués d'un nom, d'un prénom, d'un numéro de téléphone et éventuellement de cinq lignes de renseignements supplémentaires.

Chargez le programme assemblé et stocké sur une disquette, puis lancez-le. Un en-tête est affiché. Dès le début, le programme va rechercher sur la disquette un annuaire (du nom de ANNFICh). S'il ne le trouve pas, il vous proposera de changer de disquette ou d'initialiser un nouvel annuaire.

On arrive ensuite au menu principal. Vous disposez de cinq choix. Le choix que vous pouvez sélectionner directement en appuyant sur la touche (RETURN) clignote. Pour choisir une autre possibilité, appuyez sur une touche quelconque, sauf (ESC) et (RETURN). Le choix suivant se mettra alors à clignoter, et ainsi de suite si vous continuez jusqu'à ce que le choix que vous désirez clignote ; il ne vous reste plus qu'à le sélectionner en appuyant sur (RETURN).

Donc cinq choix s'offrent à vous, qui sont :

- CONSULTER, c'est-à-dire lire l'annuaire.
- ENREGISTRER, c'est-à-dire placer une nouvelle référence dans l'annuaire.
- MODIFIER, c'est-à-dire changer tout ou partie d'une référence.

- SUPPRIMER, c'est-à-dire retirer une référence de l'annuaire.
- FIN, c'est-à-dire sauver l'annuaire sur disquette puis revenir au BASIC.

Si vous choisissez de consulter, l'annuaire vous propose de chercher un nom ou un numéro de téléphone (selon ce que vous connaissez). Une fois votre choix fait, vous devez rentrer en bas ce nom ou ce numéro que vous cherchez ; pour le nom, vous pouvez n'en rentrer que les trois premières lettres. Cette entrée faite, le programme vous dit combien il a trouvé de références lui correspondant et les affiche s'il y en a.

Ensuite, une fin qui est la même pour presque tous les choix du menu principal : le programme vous propose de recommencer l'opération (donc de chercher une autre référence) ou de revenir au menu principal.

Si vous choisissez (dans le menu principal) d'enregistrer, le programme va vous demander successivement :

- Le *nom* (obligatoire) : pas plus de 32 caractères, obligatoirement des lettres ou l'un des caractères suivants : espace, &, ', virgule, point ou tiret (—). Le nom est transformé en majuscules.
- Le *prénom* (facultatif) : pas plus de 16 caractères, obligatoirement des lettres ou des tirets. Il est transformé en minuscules, sauf l'initiale.
- Le *numéro de téléphone* (obligatoire) : pas plus de 16 caractères, obligatoirement des chiffres ou des espaces.
- Cinq lignes d'*adresse* (facultatives). Ce champ est libre, mais chaque ligne ne doit pas contenir plus de 40 caractères (une ligne) ; vous pouvez y mettre les renseignements complémentaires que vous souhaitez. Si vous rentrez une ligne vide, l'enregistrement s'arrête.

Le choix qui vous est alors proposé est semblable à celui qui suit "Consulter", mais vous pouvez en outre sauver l'annuaire.

La suppression a un déroulement très simple : le nom à supprimer vous est demandé ; puis, une fois trouvé et affiché, une confirmation est demandée. Si vous répondez positivement, la référence est effacée.

La modification commence de même, puis les lignes de la référence à modifier sont éditées les unes après les autres. Pour le reste, tout se déroule comme à l'enregistrement.

■ FONCTIONNEMENT

STRUCTURE DE L'ANNUAIRE

Le fonctionnement général est assez simple, une fois choisies les structures d'enregistrement de l'annuaire.

Cette structure est relativement complexe. Elle se décompose en deux listes, appelées liste 1 et liste 2 dans la suite.

La liste 1, première présente en mémoire en 2006H, se compose, si l'annuaire contient N références, de N pointeurs de 9 octets chacun. Ces pointeurs contiennent : les trois premières lettres du nom (octets 0 à 2), puis quatre nombres qui sont les derniers nombres du numéro de téléphone (chaque nombre représente deux caractères du numéro, donc est compris entre 0 et 99) (octets 3 à 6), enfin l'adresse du bloc correspondant (octets 7 et 8).

Ce bloc est composé de toutes les lignes de la référence, chacune précédée de sa longueur, et se termine par un zéro. L'ensemble de tous les blocs constitue la liste 2.

On voit donc la différence entre les deux listes : la liste 1 contient les renseignements qui permettront aisément la recherche, d'autant que ses éléments ont une longueur constante de 9 octets et sont classés par ordre alphabétique des noms (ou plutôt des triades, trois premières lettres du nom). La liste 2, qui suit immédiatement la liste 1 en mémoire, est constituée de blocs disparates, de longueurs variables, sans classement aucun.

L'ensemble de cette structure, avec un annuaire s'étendant jusqu'à TAMPON, permet de stocker environ de deux à trois cents références.

La liste 1 est précédée de trois nombres sur 2 octets, de 2000H à 2005H : d'abord l'adresse du début de la liste 2, puis celle de fin de cette liste (et donc de l'annuaire), enfin le nombre N de références de l'annuaire.

Précisons un détail sur les pointeurs de la liste 1 : les trois premiers octets, étant les codes ASCII du début du nom, ont leur bit 7 à zéro. Les deux premiers de ces bits 7 sont donc en fait utilisés comme ceci : si le bit 7 du premier octet est non nul, c'est que cette triade se ren-

contre à plusieurs reprises dans la liste 1 (on est en présence d'une série) ; si le bit 7 du deuxième octet est non nul aussi, c'est que l'on a la dernière triade de la série.

Prenons un exemple pour éclaircir tout cela. Vous avez enregistré le nom suivant : dupont/ anatole/ 16 33 44 55 66/ 2 rue des fosses/ PARIS/ N'appeler qu'après 19H. Le programme a affiché ceci :

DUPONT

Anatole

16 33 44 55 66

2 rue des fosses

PARIS

N'appeler qu'après 19H.

Au moment de l'enregistrement, il y avait déjà un DUPUIS dans l'annuaire, donc une triade DUP. Vous trouverez comme pointeur les codes hexadécimaux suivants :

C4 : code ASCII de D, plus bit 7 mis (déjà une triade).

55 50 : codes ASCII de U et P.

21 2C 37 42 : valeurs hexadécimales de 33, 44, 55, 66.

56 21 : adresse du bloc (2156H).

En 2156H, vous trouverez d'abord 6 (longueur du nom), puis les six lettres de DUPONT, puis 7 (longueur du prénom), etc.

Si vous consultez l'annuaire en donnant DUP comme nom, le programme vous signalera la présence de deux noms, DUPUIS et DUPONT. Si vous donnez DUPO ou plus de lettres encore, le programme ne vous signalera qu'un seul nom, DUPONT.

AUTRES POINTS DE FONCTIONNEMENT

On notera encore quelques détails importants de fonctionnement. Le programme s'exécute en mode 1. Le seul registre à valeur constante est IY, qui pointe sur la table des paramètres (qui commence à FLAGS).

Lors d'une entrée au clavier, quelle qu'elle soit (choix ou demande de chaîne), un BREAK conduit à un retour en arrière (éventuellement jusqu'au menu principal) et à l'arrêt des opérations en cours.

■ ROUTINES

Voici quelques routines importantes de ce programme, sommairement expliquées :

ENTREE

RETOUR

ENTREE est le point d'entrée du programme à l'initialisation : il se charge de régler les deux fenêtres (l'une comprend les 22 dernières lignes, l'autre les 23 dernières), de mettre les couleurs et l'en-tête, puis de lire le fichier Annfich (annuaire proprement dit) avec LIRFICH ; si ce fichier n'est pas trouvé, il propose de changer de disquette ou d'en créer un. Il passe ensuite à RETOUR.

RETOUR est le point d'entrée à chaud : il affiche le menu principal.

CHOIX

Cette routine affiche une série de choix (dont le nombre doit être précisé dans B) qui se trouvent à l'adresse placée sur la pile (c'est pourquoi elle est appelée par un CALL, qui est en fait un JP). A cette adresse doivent se trouver les chaînes à afficher pour les choix, suivies des adresses de saut correspondantes. La routine attend qu'un choix soit fait, puis saute à l'adresse correspondante.

PRCHAIN

Cette routine, très souvent appelée, affiche une chaîne alphanumérique. L'adresse de la chaîne (dont le premier octet est la longueur) doit être placée dans HL.

Si la chaîne contient un code n compris entre 1 et 31 (non alphanumérique), la n^{ième} chaîne de la table CODORDR est écrite à la place du code. Si la chaîne contient un zéro, il est remplacé par le code contenu dans (IY + 0).

SEEKNOM

Cette routine recherche dans la liste 1 la triade (trois premières lettres du nom) pointée par DE. En sortie, la retenue est mise si le nom a été trouvé, sinon elle est nulle. Dans les deux cas HL pointe sur le pointeur du nom ou, s'il n'a pas été trouvé, sur l'endroit où il serait dans l'ordre alphabétique. Si plusieurs noms ont été trouvés, leur nom-

bre moins un se trouve dans A (le flag z est donc mis s'il n'y en a qu'un).

La recherche est une recherche dichotomique, dont le principe est expliqué chez R. Zaks (voir bibliographie).

SEEKNUM

Cette routine cherche un numéro pointé par DE en partant du début de la liste 1. Si le numéro est trouvé, la retenue est mise. Le cas de plusieurs numéros semblables, très improbable, n'est pas prévu.

CONSULT CONSNUM CONSNOM

Cette routine est celle de consultation. Les deux cas (consulter un nom ou un numéro) sont semblables, mais le premier est plus complexe (possibilité de noms multiples). La routine demande le nom, le vérifie, puis appelle SEEKNUM. Les noms trouvés sont ensuite listés.

ENREG

Cette routine assez compliquée permet l'enregistrement du nom dans l'annuaire. Après les demandes de toutes les lignes et leur vérification éventuelle, le bloc et le pointeur sont construits à l'adresse TAMPON, l'un derrière l'autre.

Ensuite SEEKNUM est appelée pour savoir où placer le pointeur. Celui-ci est inséré, le bloc est placé en fin de liste 2, les trois paramètres indiquant le début et la fin de cette liste et le nombre d'éléments sont modifiés, puis toutes les adresses des pointeurs sont augmentées de 9 (un pointeur de plus, donc liste 2 décalée).

SUPPRIM DETRUIRE

Cette routine est l'inverse de la précédente mais, ses entrées étant limitées, elle est plus simple. Le nom à supprimer est demandé puis, après l'avoir cherché et trouvé, une confirmation est demandée. En cas de réponse positive, le bloc, puis le pointeur sont supprimés.

La principale difficulté résulte de ce que le bloc ayant une longueur quelconque et se trouvant n'importe où dans la liste 2 il faut réactualiser tous les pointeurs de la liste 1 qui pointent sur des blocs qui suivaient le bloc supprimé (en plus de la décrémentation de 9 qui résulte de la suppression du pointeur).

MODIF

Cette routine ressemble un peu à un mixage de SUPPRIM et de ENREG. Il faut en effet d'abord demander le nom à modifier, le chercher et éditer ses lignes une à une. Ensuite, le bloc ayant été construit et le nouveau pointeur calculé, il faut supprimer l'ancien pointeur et l'ancien bloc avant de replacer les nouveaux.

```
1 ;=====
2 ;=====
3 ;==
4 ;== ANNUAIRE TELEPHONIQUE ==
5 ;==
6 ;=====
7 ;== T. LACHAND-ROBERT, 1986 ==
8 ;=====
9 ;=====
10
11
12 ORG 9000H
13 LOAD 3000H
14
15
16 LISTES: EQU 2000H
17 TAMPON: EQU 8E00H
18 ZONE: EQU 8F00H
19 ROMS: EQU 0B900H
20 CWAIT: EQU 0BB06H
21 CSCRUT: EQU 0BB09H
22 PRINT: EQU 0BB5AH
23 WIND0: EQU 0BB66H
24 CLS0: EQU 0BB6CH
25 HPOS: EQU 0BB6FH
26 LOCATE: EQU 0BB75H
27 TCRON: EQU 0BB7BH
28 SYSCRON: EQU 0BB81H
29 SYSCROF: EQU 0BB84H
30 PEN: EQU 0BB90H
31 PAPER: EQU 0BB96H
32 OPAQ: EQU 0BB9FH
33 WINDOW: EQU 0BBB4H
34 SWSTREAM: EQU 0BBB7H
35 MOVE: EQU 0BBC0H
36 MOVER: EQU 0BBC3H
37 GPEN: EQU 0BBDEH
38 GPAPER: EQU 0BBE4H
39 DRAW: EQU 0BBF6H
40 TAGWR: EQU 0BBFCH
41 SINIT: EQU 0BBFFH
42 INK: EQU 0BC32H
43 BORDER: EQU 0BC38H
44 SPDINK: EQU 0BC3EH
45 OPEN: EQU 0BC77H
46 CLOSE: EQU 0BC7AH
47 KINFICH: EQU 0BC83H
48 OPENOUT: EQU 0BC8CH
49 CLOSOUT: EQU 0BC8FH
50 KOUTFICH: EQU 0BC96H
51 EDIT: EQU 0BD5EH ;464: BD3AH; 664: BD5BH
52
53 9000 C30792 JP ENTREE ;Saut au point d'entree
54
55 ;DONNEES ET CHAINES ALPHANUMERIQUES
56 ;-----
57
58
59 9003 1A00 ENCREs: DW 1AH ;3= blanc/noir clignotant
60 9005 0404 DW 404H ;2= magenta
61 9007 1A1A DW 1A1AH ;1= blanc
```

```

62 9009 0A0A          DW 0A0AH          ;0= turquoise
63
64          ;CHAINES DES ORDRES
65
66 900B 09434F4E CCONSULT:DB 9,"CONSULTER"
66 900F 53554C54
66 9013 4552
67 9015 0B454E52 CENREG: DB 11,"ENREGISTRER"
67 9019 45474953
67 901D 54524552
68 9021 084D4F44 CMODIF: DB 8,"MODIFIER"
68 9025 49464945
68 9029 52
69 902A 09535550 CSUPPRIM:DB 9,"SUPPRIMER"
69 902E 5052494D
69 9032 4552
70 9034 0346494E CFIN: DB 3,"FIN"
71 9038 0A435245 CCREER: DB 10,"CREER UN ",1
71 903C 45522055
71 9040 4E2001
72 9043 14434841 CCHGDSC: DB 20,"CHANGER DE"
72 9047 4E474552
72 904B 204445
73 904E 20444953 DB " DISQUETTE"
73 9052 51554554
73 9056 5445
74 9058 08434845 CCHERCHE:DB 8,"CHERCHER"
74 905C 52434845
74 9060 52
75 9061 08002045 CENC: DB 8,0," ENCORE"
75 9065 4E434F52
75 9069 45
76 906A 06524554 CRET: DB 6,"RETOUR"
76 906E 4F5552
77 9071 0A534155 CSAUV: DB 10,"SAUVER L'",1
77 9075 56455220
77 9079 4C2701
78 907C 08002055 CUNNOM: DB 8,0," UN NOM"
78 9080 4E204E4F
78 9084 4D
79 9085 0B00 CUNNUM: DB 11,0
80 9087 20554E20 DB " UN NUMERO"
80 908B 4E554D45
80 908F 524F
81 9091 026F6D CNOM: DB 2,"om"
82 9094 05756D65 CNUM: DB 5,"umero"
82 9098 726F
83
84          ;MESSAGES
85
86 909A 20416E6E MESS0: DB 32,"Annuaire "
86 909E 75616972
86 90A2 6520
87 90A4 61627365 DB "absent de la"
87 90A8 6E742064
87 90AC 65206C61
88 90B0 20646973 DB " disquette."
88 90B4 71756574
88 90B8 74652E
89 90BB 14457272 MESS1: DB 20,"Erreur. "
89 90BF 6575722E
89 90C3 20
90 90C4 5265636F DB "Recommencez:"
90 90C8 6D6D656E
90 90CC 63657A3A
91 90D0 034E043A MESS2: DB 3,"N",4,"";"Nom:"
92 90D4 06507265 MESS3: DB 6,"Pren",4,"";"Prenom:"
92 90D8 6E043A
93 90DB 11416472 MESS4: DB 17,"Adresse, "
93 90DF 65737365
93 90E3 2C20
94 90E5 6C69676E DB "ligne 1:"
94 90E9 6520313A
95 90ED 104E0520 MESS5: DB 16,"N",5,32 ;"Numero "
96 90F1 64652074 DB "de telephone:"
96 90F5 656C6570

```

```

96 90F9 686F6E65
96 90FD 3A
97 90FE 1254726F MESS6: DB 18,"Trouve un"
97 9102 75766520
97 9106 756E
98 9108 20736575 DB " seul n",0,"."
98 910C 6C206E00
98 9110 2E
99 9111 174E0420 MESS7: DB 23,"N",4," pas "
99 9115 70617320
100 9119 64616E73 DB "dans l'annuaire."
100 911D 206C2761
100 9121 6E6E7561
100 9125 6972652E
101 9129 0D54726F MESS8: DB 13,"Trouve "
101 912D 75766520
102 9131 32206E04 NBRNOM: DB "2 n",4,"s."
102 9135 732E
103 9137 08 DB 8
104 9138 414E4E55 MESS9: DB "ANNUAIRE "
104 913C 41495245
104 9140 20
105 9141 54454C45 DB "TELEPHONIQUE"
105 9145 50484F4E
105 9149 49515545
106 914D 092A2A2A MESS10: DB 9,"*** ",0," ***"
106 9151 2000202A
106 9155 2A2A
107 9157 094E4F4D MESS11: DB 9,"NOM A ",0," ?"
107 915B 20412000
107 915F 203F
108 9161 1643686F MESS12: DB 22,"Choisissez"
108 9165 69736973
108 9169 73657A
109 916C 20756E65 DB " une lettre:"
109 9170 206C6574
109 9174 7472653A
110 9178 08002028 MESS13: DB 8,0," (O/N)?"
110 917C 4F2F4E29
110 9180 3F
111 9181 16537570 MESS14: DB 22,"Suppression"
111 9185 70726573
111 9189 73696F6E
112 918D 20656666 DB " effectuee."
112 9191 65637475
112 9195 65652E
113 9198 26417070 MESS15: DB 38,"Appuyez sur "
113 919C 7579657A
113 91A0 20737572
113 91A4 20
114 91A5 756E6520 DB "une touche "
114 91A9 746F7563
114 91AD 686520
115 91B0 706F7572 DB "pour continuer."
115 91B4 20636F6E
115 91B8 74696E75
115 91BC 65722E
116 91BF 19496E74 MESS16: DB 25,"Introduisez "
116 91C3 726F6475
116 91C7 6973657A
116 91CB 20
117 91CC 6C612064 DB "la disquette."
117 91D0 69737175
117 91D4 65747465
117 91D8 2E
118
119 ;ADRESSE DES ORDRES CODES.
120
121 91D9 3791 CODORD: DW MESS9-1 ;Chaine 1:"ANNUAIRE"
122 91DB 5890 DW CCHERCHE ;Chaine 2:"CHERCHER"
123 91DD 1590 DW CENREG ;Chaine 3:"ENREGISTRER"
124 91DF 9190 DW CNOM ;Chaine 4:"UN NOM"
125 91E1 9490 DW CNUM ;Chaine 5:"UN NUMERO"
126 91E3 0B90 DW CCONSULT ;Chaine 6:"CONSULTER"
127 91E5 2190 DW CMODIF ;Chaine 7:"MODIFIER"
128 91E7 2A90 DW CSUPPRIM ;Chaine 8:"SUPPRIMER"

```

```

129
130 91E9 414E4E46 NOMFICH: DB "ANNFICH.BIN" ;Nom d'enregistrement
130 91ED 49434E2E
130 91F1 42494E
131
132 FLAGS: DS 4 ;Divers flags pointes par IY
133 ACC: DS 4 ;Deux accumulateurs entiers
134 POS: DS 2 ;Position curseur
135 ADORD: DS 2 ;Adresse de l'ordre courant
136
137 ;CARACTERES POUR LE NOM
138 9200 2620272C TABCARAC:DB "& ',.-",0 ;Caracteres autorises
138 9204 2E2D00
139
140
141 ; POINT D'ENTREE A FROID.
142 ;FENETRES, COULEURS ET TITRE.
143 ;-----
144
145
146 9207 CDFBFB ENTREE: CALL SINIT ;Mode 1,etc.
147 920A 010A0A LD BC,0A0AH
148 920D CD38BC CALL BORDER
149 9210 210390 LD HL,ENCRS ;Adresse des encres
150 9213 3E04 LD A,4 ;Quatre encres
151 9215 46 BCLENCR: LD B,(HL)
152 9216 23 INC HL
153 9217 4E LD C,(HL) ;Valeur de l'encre
154 9218 23 INC HL
155 9219 3D DEC A
156 921A F5 PUSH AF
157 921B E5 PUSH HL
158 921C CD32BC CALL INK ;Positionner
159 921F E1 POP HL
160 9220 F1 POP AF
161 9221 20F2 JR NZ,BCLENCR
162 9223 3E02 LD A,2
163 9225 CD96BB CALL PAPER ;Fond magenta pour le titre
164 9228 3E02 LD A,2
165 922A CDE4BB CALL GPAPER ;Idem graphique
166 922D CD6CBB CALL CLS0
167 9230 3E01 LD A,1 ;Titre en blanc
168 9232 CDDEBB CALL GPEN
169 9235 212020 LD HL,2020H ;Clignotement assez lent
170 9238 CD3EBC CALL SPDINK ;Positionner
171 923B 114C00 LD DE,76
172 923E 218801 LD HL,392
173 9241 CDC0BB CALL MOVE ;Coin premiere lettre
174 9244 213891 LD HL,MESS9 ;Lettres a afficher
175 9247 0615 LD B,21 ;Vingt et une
176 9249 C5 BCLTITRE: PUSH BC
177 924A 7E LD A,(HL) ;Donner la lettre
178 924B 23 INC HL
179 924C E5 PUSH HL
180 924D CDFCBB CALL TAGWR ;Et l'ecrire
181 9250 210000 LD HL,0
182 9253 110800 LD DE,8
183 9256 CDC3BB CALL MOVER ;Se deplacer de 8 points
184 9259 E1 POP HL
185 925A C1 POP BC
186 925B 10EC DJNZ BCLTITRE ;Et continuer
187 925D 110000 LD DE,0
188 9260 218E01 LD HL,398
189 9263 E5 PUSH HL
190 9264 D5 PUSH DE
191 9265 E5 PUSH HL
192 9266 CDC0BB CALL MOVE ;Coin superieur du cadre
193 9269 E1 POP HL
194 926A 117F02 LD DE,639
195 926D D5 PUSH DE
196 926E CDF6BB CALL DRAW ;Premier bord
197 9271 D1 POP DE
198 9272 217001 LD HL,368
199 9275 E5 PUSH HL
200 9276 CDF6BB CALL DRAW ;Deuxieme bord
201 9279 E1 POP HL

```

```

202 927A D1          POP DE
203 927B D5          PUSH DE
204 927C CDF6BB     CALL DRAW          ;Troisieme bord
205 927F D1          POP DE
206 9280 E1          POP HL
207 9281 CDF6BB     CALL DRAW          ;Quatrieme et dernier
208 9284 3E02       LD A,2
209 9286 2600       BCLFEN: LD H,0
210 9288 6F         LD L,A
211 9289 111928     LD DE,02819H     ;Ligne 2 ou 3
212 928C F5         PUSH AF          ;Coin inf. droit de l'ecran
213 928D CD66BB     CALL WIND0
214 9290 F1         POP AF          ;Fenetre 2 ou 3
215 9291 47         LD B,A
216 9292 0E00       LD C,0
217 9294 F5         PUSH AF
218 9295 CDB7BB     CALL SWSTREAM    ;Positionner
219 9298 F1         POP AF
220 9299 3C         INC A
221 929A FE04       CP 4
222 929C 20E8       JR NZ,BCLFEN    ;Definir aussi fen. 3
223 929E CDEB9A     CALL FEN1
224 92A1 CD7499     CALL LIRFICH     ;Lire l'annuaire
225
226
227                 ;MENU PRINCIPAL
228                 ;-----
229
230
231 92A4 FD21F491    RETOUR: LD IY,FLAGS ;Valeur constante de IY
232 92A8 CDEB9A     CALL FEN1        ;Effacer plus grande fenetre
233 92AB 21A492     LD HL,RETOUR
234 92AE 221A93     LD (BREAKAD),HL ;Adresse en cas de Break
235 92B1 0605       LD B,5           ;5 choix dans la table
236 92B3 CDCA92     CALL CHOIX       ;Saut a la routine de choix
237 92B6 0B90       DW CCONSULT     ;Chaine alphanumerique,...
238 92B8 0394       DW CONSULT      ;...et adresse correspondante
239 92BA 1590       DW CENREG
240 92BC 5995       DW ENREG
241 92BE 2190       DW CMODIF
242 92C0 4A97       DW MODIFIER
243 92C2 2A90       DW CSUPPRIM
244 92C4 0F98       DW SUPPRIME
245 92C6 3490       DW CFIN
246 92C8 3295       DW FIN
247
248                 ;AFFICHAGE DES CHOIX
249
250 92CA 48          CHOIX: LD C,B     ;Stocker nombre de choix
251 92CB CD6B9A     CALL LOC+10      ;Placer curseur
252 92CE E1         POP HL          ;Recuperer adresse de la table
253 92CF 54         LD D,H
254 92D0 5D         LD E,L
255 92D1 CD5C9B     CALL ECR0
256 92D4 CD0A9A     BCLCHX1: CALL SUIVANT ;Ecrire en blanc
257 92D7 23        INC HL          ;Ecrire une chaine et passer.
258 92D8 23        INC HL
259 92D9 23        INC HL
260 92DA 10F8       DJNZ BCLCHX1    ;Adresse chaine suivante
261 92DC CD6B9A     BCLCHX2: CALL LOC+10 ;Ecrire toutes les chaines
262 92DF 41        LD B,C          ;Replacer le curseur
263 92E0 62        LD H,D
264 92E1 6B        LD L,E          ;Recuperer nombre de chaines
265 92E2 CD569B     BCLCHX3: CALL ECR1    ;Et adresse de la table
266 92E5 CD339B     CALL PRCH1      ;Ecrire en clignotant
267 92E8 CD5C9B     CALL ECR0
268 92EB CD06BB     CALL CWAIT      ;La chaine pointee par HL
269 92EE FEFC       CP 0FCH         ;Ecrire normalement
270 92F0 2824       JR Z,BREAK      ;Attendre une touche
271 92F2 FE0D       CP 13           ;Break?
272 92F4 280F       JR Z,SUITCHX    ;RETURN ?
273 92F6 2B        DEC HL          ;Oui, choix fait, continuer
274 92F7 CD0A9A     CALL SUIVANT    ;Non, choix non fait...
275 92FA 05        DEC B           ;Reecrire en blanc et passer
276 92FB 28DF       JR Z,BCLCHX2   ;Fin de la liste ?
277 92FD 23        INC HL          ;Oui, repasser au debut

```

```

278 92FE 23          INC HL
279 92FF 23          INC HL ;Adresse chaine suivante
280 9300 CD4E9B      CALL HORPOS8 ;Positionnement horizontal
281 9303 18DD        JR BCLCHX3 ;Et boucler
282 9305 23          SUITCHX: INC HL ;Choix fait
283 9306 5E          LD E,(HL)
284 9307 23          INC HL
285 9308 56          LD D,(HL) ;Adresse de l'instruction
286 9309 7A          LD A,D
287 930A B3          OR E ;Nulle ?
288 930B EB          EX DE,HL
289 930C 2804        JR Z,VERSOP ;Oui, saut a ADORD
290 930E 22FE91      LD (ADORD),HL ;Non, sauver
291 9311 E9          JP (HL) ;Et sauter
292 9312 2AFE91      VERSOP: LD HL,(ADORD) ;Adresse de l'op. precedente
293 9315 E9          JP (HL)
294 9316 CD6CBB      BREAK: CALL CLS0
295 9319 C3          DB 0C3H ;Code de l'instruction JP
296                BREAKAD: DS 2 ;Adresse en cas de Break.
297
298                ;RECHERCHE D'UN NOM
299
300 931C FD360003     SEEKNOM: LD (IY+0),3 ;3 caracteres a comparer
301 9320 FD3603FF     LD (IY+3),255 ;Flag 3 initialise
302 9324 ED53F891     LD (ACC),DE ;Stocker adr. nom a chercher
303 9328 2A0420       LD HL,(LISTES+4) ;Nombre de ref.
304 932B 7C          LD A,H
305 932C B5          OR L ;Zero ?
306 932D 2850        JR Z,PREMIER ;Oui, annuaire vide, termine
307 932F 54          LD D,H
308 9330 5D          LD E,L ;Nombre dans DE
309 9331 29          ADD HL,HL ;Initialiser HL
310 9332 FD360200     LD (IY+2),0 ;Et flag 2 a zero
311 9336 CDC293       BCLESEEK: CALL ADDSUB ;Additionner ou soustraire
312 9339 CB3A        SRL D
313 933B CB1B        RR E ;Diviser DE par deux
314 933D E5          PUSH HL
315 933E 210000       LD HL,0
316 9341 ED5A        ADC HL,DE ;Et ajouter le reste
317 9343 EB          EX DE,HL
318 9344 E1          POP HL
319 9345 D5          PUSH DE
320 9346 ED5B0420     LD DE,(LISTES+4) ;Valeur maximale
321 934A E5          PUSH HL
322 934B 37          SCF
323 934C ED52        SBC HL,DE ;Depassement de HL ?
324 934E E1          POP HL
325 934F D1          POP DE
326 9350 3029        JR NC,DERNIER ;Oui, trop loin, termine
327 9352 E5          PUSH HL
328 9353 D5          PUSH DE
329 9354 CD7293       CALL DONADR ;Donner adresse du pointeur
330 9357 ED5BF891     LD DE,(ACC) ;Et celle du nom cherche
331 935B CDDC9A       CALL COMPARE ;Faire la comparaison
332 935E D1          POP DE
333 935F E1          POP HL
334 9360 2821        JR Z,TROUVE ;Egal, alors trouve
335 9362 9F          SBC A,A ;Selon inegalite...
336 9363 FD7702       LD (IY+2),A ;...flag 2= 0 ou FFH
337 9366 CDB493       CALL TESTFG2 ;Tester si DE=1 deux fois
338 9369 20CB        JR NC,BCLESEEK ;Non, continuer la recherche
339 936B FDCB0206     RLC (IY+2) ;Recherche non aboutie
340 936F 3001        JR NC,DONADR ;Prendre valeur par excès
341 9371 23          INC HL
342
343 9372 2B          DONADR: DEC HL ;Donner adresse correspondante
344 9373 CDC199       CALL FOIS9 ;Multiplier par 9
345 9376 110620       LD DE,LISTES+6
346 9379 19          ADD HL,DE ;Ajouter LISTES+6
347 937A C9          RET ;Retenue nulle en sortie
348 937B 2A0020       DERNIER: LD HL,(LISTES) ;Adresse fin liste 1
349 937E C9          RET
350 937F 210620       PREMIER: LD HL,LISTES+6 ;Adresse debut liste 1
351 9382 C9          RET
352
353 9383 CD7293       TROUVE: CALL DONADR ;Aller chercher adr. pointeur

```

```

354 9386 010900 LD BC,9
355 9389 CB7E BIT 7,(HL) ;Appartient a une serie ?
356 938B 280E JR Z,FINTRV ;Non, OK
357 938D B7 BCLTRV1: OR A ;Oui, chercher debut de serie
358 938E ED42 SBC HL,BC ;-9: pointeur precedent
359 9390 7C LD A,H
360 9391 FE20 CP 20H ;Depassement par le bas?
361 9393 3805 JR C,FINTRV-1 ;Oui, termine
362 9395 CDBA99 CALL ELSERIE ;Prec. appartient a la serie ?
363 9398 38F3 JR C,BCLTRV1 ;Non, continuer
364 939A 09 ADD HL,BC ;Repasser au premier de serie
365 939B E5 FINTRV: PUSH HL
366 939C 110700 LD DE,7
367 939F 19 ADD HL,DE
368 93A0 5E LD E,(HL)
369 93A1 23 INC HL
370 93A2 56 LD D,(HL) ;Adresse de bloc dans DE
371 93A3 E1 POP HL
372 93A4 AF XOR A
373 93A5 CB7E BIT 7,(HL) ;Appartient a une serie ?
374 93A7 37 SCF ;retenue= trouve
375 93A8 C8 RET Z ;Non, A=0, termine
376 93A9 E5 PUSH HL
377 93AA 23 INC HL
378 93AB 09 BCLTROUV: ADD HL,BC ;Compter la serie
379 93AC 3C INC A
380 93AD CB7E BIT 7,(HL) ;Dernier de la serie ?
381 93AF 28FA JR Z,BCLTROUV ;Non, continuer
382 93B1 E1 POP HL ;Restituer premier
383 93B2 37 SCF ;Trouve.
384 93B3 C9 RET ;Fin.
385 93B4 1B TESTFG2: DEC DE
386 93B5 7A LD A,D
387 93B6 B3 OR E ;DE= 1?
388 93B7 13 INC DE
389 93B8 C0 RET NZ ;Non, ok.
390 93B9 FDBE03 CP (IY+3) ;DE deja egal a 1 avant ?
391 93BC C8 RET Z ;Oui, retour
392 93BD FD7703 LD (IY+3),A ;Non, mettre flag 3 a zero
393 93C0 3C INC A ;Flag z non mis
394 93C1 C9 RET
395 93C2 FDCB0206 ADDSUB: RLC (IY+2) ;Flag 2 a 0 ou FFH
396 93C6 3002 JR NC,ADDSUB2 ;0, soustraire
397 93C8 19 ADD HL,DE ;Monter dans la liste
398 93C9 C9 RET
399 93CA 7D ADDSUB2: LD A,L ;Retenir parite de HL
400 93CB ED52 SBC HL,DE ;Descendre dans la liste
401 93CD 1F RRA ;HL pair avant ?
402 93CE D0 RET NC ;Oui, ok.
403 93CF 7B LD A,E
404 93D0 3D DEC A
405 93D1 B2 OR D ;DE= 1?
406 93D2 C8 RET Z ;Oui, termine.
407 93D3 23 INC HL ;Sinon, augmenter HL
408 93D4 C9 RET
409
410 ;RECHERCHE D'UN NUMERO
411
412 93D5 210920 SEEKNUM: LD HL,LISTES+9 ;Premier numero
413 93D8 FD360004 LD (IY+0),4 ;4 chiffres a comparer
414 93DC ED4B0420 LD BC,(LISTES+4) ;Nombre de ref.
415 93E0 78 LD A,B
416 93E1 B1 OR C ;Nul ?
417 93E2 C8 RET Z ;Oui, termine
418 93E3 C5 BCLSKNUM: PUSH BC
419 93E4 D5 PUSH DE
420 93E5 E5 PUSH HL
421 93E6 CDDC9A CALL COMPARE ;Comparer avec num. ds (DE)
422 93E9 E1 POP HL
423 93EA D1 POP DE
424 93EB 280B JR Z,TROUVNUM ;Trouve, ok.
425 93ED 010900 LD BC,9
426 93F0 09 ADD HL,BC ;Non, pointeur suivant
427 93F1 C1 POP BC
428 93F2 0B DEC BC
429 93F3 78 LD A,B

```

```

430 93F4 B1 OR C ;Arrive au dernier ?
431 93F5 20EC JR NZ,BCLSKNUM ;Non, comparer encore
432 93F7 C9 RET ;Oui, pas trouve, termine
433
434 93F8 C1 TROUVNUM: POP BC
435 93F9 010400 LD BC, 4
436 93FC 00 ADD HL, BC ;Fin du pointeur
437 93FD 7E LD A, (HL)
438 93FE 23 INC HL
439 93FF 66 LD H, (HL)
440 9400 6F LD L, A ;Adresse du bloc dans HL
441 9401 37 SCF ;retenue= trouve
442 9402 C9 RET
443
444
445 ;CONSULTER L'ANNUAIRE
446 ;-----
447
448
449 9403 FD360106 CONSULT: LD (IY+1), 6 ;Chaine 6= "CONSULTER"
450 9407 260C LD H, 12 ;Centrage
451 9409 CDF89A CALL FEN2 ;Ecrire titre et nouvelle fen.
452 940C 0602 LD B, 2 ;Deux choix
453 940E FD7001 LD (IY+1), B ;Chaine 2= "CHERCHER"
454 9411 CDCA92 CALL CHOIX ;Vers le choix
455 9414 7C90 DW CUNNOM ;Table
456 9416 5B94 DW CONSNOM
457 9418 8590 DW CUNNUM
458 941A 2C94 DW CONSNUM
459
460 941C FD7701 DEBCONS: LD (IY+1), A ;Partie commune
461 941F 321391 LD (MESS7+2), A ;Stocker code pour messages
462 9422 CD5A9A CALL POSHAUT ;Placer en haut
463 9425 CD6CBB CALL CLS0 ;Effacement
464 9428 110A8E LD DE, TAMPON+10 ;Emplacement de recherche
465 942B C9 RET
466
467
468 942C 3E05 CONSNUM: LD A, 5 ;Chaine 5= (N)umero
469 942E CD1C94 CALL DEBCONS ;Partie commune
470 9431 CDDA96 CALL ENRNUM ;Enregistrer numero demande
471 9434 11038E LD DE, TAMPON+3 ;Emplacement des codes
472 9437 CDD593 CALL SEEKNUM ;Rechercher
473 943A D2EF94 JP NC, NONTRV0 ;Pas trouve, ecrire message
474 943D E5 UNSEUL: PUSH HL
475 943E CD769A CALL RELOCATE ;Se replacer
476 9441 21FE90 LD HL, MESS6
477 9444 CD3E9B CALL PRCHAIN ;Ecrire message
478 9447 CD709A CALL LOC+15
479 944A E1 POP HL
480 944B CDED99 CALL PRNOM ;Ecrire la reference trouvee
481 944E CDC999 CALL WAIT ;Attendre
482 9451 CD6CBB CALL CLS0
483 9454 FD360102 LD (IY+1), 2 ;Code de "CHERCHER"
484 9458 C30695 JP VERSCHX2 ;Choix2: Encore/Retour
485
486
487 945B 3E04 CONSNUM: LD A, 4 ;Chaine 4: (N)om
488 945D CD1C94 CALL DEBCONS ;Partie commune
489 9460 CD8996 CALL ENRNOM ;Demander nom cherche
490 9463 11008E LD DE, TAMPON ;Emplacement des codes
491 9466 CD1C93 CALL SEEKNUM ;Recherche dichotomique
492 9469 D2EF94 JP NC, NONTRV0 ;Pas trouve, ecrire message
493 946C 010700 LD BC, 7
494 946F 09 ADD HL, BC ;Partie adresse des pointeurs
495 9470 47 LD B, A
496 9471 0E00 LD C, 0 ;Initialiser C
497 9473 04 INC B ;Nombres de pointeurs ds serie
498 9474 E5 PUSH HL
499 9475 210A8E LD HL, TAMPON+10 ;Emplacement du nom cherche
500 9478 DD21008F LD IX, ZONE ;Zone de stock
501 947C C5 BCLCONS: PUSH BC ;Verifier concordance reelle
502 947D DD7201 LD (IX+1), D
503 9480 DD7300 LD (IX+0), E ;Sauver adresse de stock
504 9483 7E LD A, (HL) ;Longueur nom cherche
505 9484 E5 PUSH HL

```

```

506 9485 23          INC HL
507 9486 13          INC DE
508 9487 CDDF9A     CALL COMPARE+3      ;Pas comparer longueurs !
509 948A E1          POP HL              ;Comparer
510 948B C1          POP BC
511 948C CCE994     CALL Z,TROUVEUN    ;Egal, incrementer C et IX
512 948F E3          EX (SF),HL        ;Recuperer pointeur
513 9490 110A00     LD DE,10
514 9493 19          ADD HL,DE           ;Pointeur suivant
515 9494 56          LD D,(HL)
516 9495 2B          DEC HL
517 9496 5E          LD E,(HL)         ;Prendre adresse de bloc
518 9497 E3          EX (SF),HL        ;Recuperer TAMPON+10
519 9498 10E2       DJNZ BCLCONS       ;Poursuivre
520 949A F1          POP AF             ;Detruire dernier pointeur
521 949B 79          LD A,C             ;Nombre de noms trouves
522 949C B7          OR A              ;Nul ?
523 949D 2850       JR Z,NONTRV0      ;Oui, ecrire non trouve.
524 949F DD21008F   LD IX,ZONE         ;Place pour adr. des blocs
525 94A3 CDE294     CALL GETADR        ;Aller chercher premier
526 94A6 3D          DEC A              ;Un seul ?
527 94A7 2894       JR Z,UNSEUL       ;Oui, ecrire un seul
528 94A9 DD21008F   LD IX,ZONE
529 94AD 41          LD B,C
530 94AE 0E31       LD C,"1"
531 94B0 81          ADD A,C            ;Nombre de ref. trouvees
532 94B1 323191     LD (NERNOM),A     ;Stocker dans message
533 94B4 CD769A     CALL RELOCATE     ;Remplacer curseur
534 94B7 212991     LD HL,MESS8
535 94BA CD3E9B     CALL PRCHAIN      ;Ecrire combien trouver
536 94BD FD360102   LD (IY+1),2       ;Code de "CHERCHER"
537 94C1 CD709A     BCLCONS2:CALL LOC+15 ;Placer curseur
538 94C4 79          LD A,C            ;Numero
539 94C5 0C          INC C
540 94C6 CD5ABB     CALL PRINT        ;Ecrire numero
541 94C9 CD159A     CALL ALALIGN2    ;Descendre de deux lignes
542 94CC CDE294     CALL GETADR      ;Aller chercher adr. du bloc
543 94CF CDED99     CALL PRNOM       ;Ecrire la reference
544 94D2 CDC999     CALL WAIT        ;Attendre
545 94D5 C5          PUSH BC
546 94D6 F5          PUSH AF
547 94D7 CD6CBB     CALL CLS0        ;Effacer l'ecran
548 94DA F1          POP AF
549 94DB C1          POP BC
550 94DC 2828       JR Z,VERSCHX2    ;Si Break ds WAIT, termine
551 94DE 10E1       DJNZ BCLCONS2    ;Continuer
552 94E0 1824       JR VERSCHX2      ;Vers choix 2: Encore/Retour
553
554 94E2 DD6601     GETADR: LD H,(IX+1) ;Aller chercher adr. du bloc
555 94E5 DD6E00     LD L,(IX+0)
556 94E8 0D          DEC C
557 94E9 0C          TROUVEUN:INC C   ;Incrementer compteur
558 94EA DD23       INC IX           ;Passer au suivant
559 94EC DD23       INC IX
560 94EE C9          RET
561
562 94EF FD7E01     NONTRV0: LD A,(IY+1) ;Code de Nom ou Numero
563 94F2 FD360102   LD (IY+1),2      ;2= code de "CHERCHER"
564 94F6 1802       JR NONTROUV+2
565 94F8 3E04       NONTROUV:LD A,4  ;4= code de (N)om
566 94FA 321391     LD (MESS7+2),A  ;Stocker dans message
567 94FD CD769A     CALL RELOCATE   ;Remplacer curseur
568 9500 211191     LD HL,MESS7
569 9503 CD3E9B     CALL PRCHAIN    ;Ecrire message
570 9506 0602       VERSCHX2:LD B,2  ;Deux choix
571 9508 C3F595     JP CHOIX3+2
572
573
574 ;SAUVER L'ANNUAIRE SUR DISQUETTE
575 950B CD1195     SAUVER: CALL SAUVER2 ;Sauver l'annuaire
576 950E C3A492     JP RETOUR       ;Et revenir
577 9511 21E991     SAUVER2: LD HL,NOMFICH ;Nom du fichier
578 9514 060B     LD B,11         ;11 caracteres
579 9516 11008E     LD DE,TAMPON    ;Tampou pour sauver
580 9519 CDBCB3     CALL OPENOUT    ;Ouvrir le fichier
581 951C 30F3       JR NC,SAUVER2   ;Marche pas, reessayer

```

```

582 951E 110020 LD DE,LISTES ;Adresse debut
583 9521 2A0220 LD HL,(LISTES+2) ;Adresse fin
584 9524 B7 OR A
585 9525 ED52 SBC HL,DE ;Longueur
586 9527 EB EX DE,HL
587 9528 3E02 LD A,2 ;Fichier binaire
588 952A 44 LD B,H
589 952B 4D LD C,L ;Point d'entree= debut
590 952C CD98BC CALL KOUTFICH ;Sauver le tout
591 952F C38FBC JP CLOSOUT ;Et fermer le fichier
592
593
594 ;SORTIE DU PROGRAMME
595 ;-----
596
597
598 9532 CD6CBB FIN: CALL CLS0
599 9535 CD619A CALL LOC ;Placer curseur
600 9538 21BF91 LD HL,MESS16
601 953B CD3E9B CALL PRCHAIN ;Demander la disquette
602 953E CDC999 CALL WAIT ;...et attendre
603 9541 CAA492 JP Z,RETOUR ;Break? Alors revenir
604 9544 CD1195 CALL SAUVER2 ;Sauver le fichier
605 9547 AF XOR A
606 9548 CDB4BB CALL WINDOW ;Fenetre plein ecran
607 954B CD6CBB CALL CLS0
608 954E 3E01 LD A,1
609 9550 CD90BB CALL PEN ;Stylos normaux
610 9553 CD00B9 CALL ROMS ;Connecter BASIC
611 9556 C358C0 JP 0C058H ;(464: C064H) Au mode Ready
612
613
614 ;ENREGISTRER UN NOM DANS L'ANNUAIRE
615 ;-----
616
617
618 9559 FD360103 ENREG: LD (IY+1),3 ;Chaine 3= "ENREGISTRER"
619 955D 260A LD H,10 ;Centrage du sous-titre
620 955F CDF89A CALL FEN2 ;Fenetre inferieure
621 9562 21F395 LD HL,CHOIX3 ;Adresse de retour...
622 9565 221A93 LD (BREAKAD),HL ;...en cas de Break
623 9568 CD5A9A CALL POSHAUT ;Regler emplacement du curseur
624 956B ED70BE LD HL,TAMPON+7 ;Emplacement adresse de bloc
625 956E ED5B0220 LD DE,(LISTES+2) ;Fin du fichier
626 9572 73 LD (HL),E ;Sera l'adr. du nouveau bloc
627 9573 23 INC HL
628 9574 72 LD (HL),D ;...donc la stocker.
629 9575 23 INC HL ;Tampou +9...
630 9576 EB EX DE,HL ;...dans DE
631 9577 CD8996 CALL ENRNOM ;Enregistrer le nom
632 957A CD4496 CALL ENRPRE ;Puis le prenom
633 957D CDDA96 CALL ENRNUM ;Puis le numero
634 9580 DD219D9A LD IX,INPUTVID ;Demander des lignes vides
635 9584 CD1696 CALL ENRADR ;...en enregistrant l'adresse
636 9587 21098E FINMODIF: LD HL,TAMPON+9 ;Adresse de debut du bloc
637 958A B7 OR A ;(dans DE, adresse de fin)
638 958B EB EX DE,HL
639 958C ED52 SBC HL,DE ;Difference= longueur du bloc
640 958E 22FA91 LD (ACC+2),HL ;Conserver
641 9591 1100BE LD DE,TAMPON ;Debut du pointeur
642 9594 D5 PUSH DE
643 9595 CD1C93 CALL SEEKNUM ;Chercher le nom
644 9598 DC0496 CALL C,DEJAUN ;Deja un? Placer les flags
645 959B E5 PUSH HL ;Adresse pointeur suivant
646 959C ED5B0220 LD DE,(LISTES+2) ;Fin des listes
647 95A0 D5 PUSH DE
648 95A1 EB EX DE,HL
649 95A2 B7 OR A
650 95A3 ED52 SBC HL,DE ;Calculer longueur a decaler
651 95A5 23 INC HL
652 95A6 D1 POP DE ;Fin des listes
653 95A7 44 LD B,H
654 95A8 4D LD C,L ;Longueur dans BC
655 95A9 210900 LD HL,9
656 95AC 19 ADD HL,DE ;Fin listes +9
657 95AD E5 PUSH HL

```

```

658 95AE EB          EX DE,HL
659 95AF 78         LD A,B
660 95B0 B1         OR C
661 95B1 2802       JR Z,SUITENRG
662 95B3 EDB8       LDDR
663 95B5 D1         SUITENRG:POP DE
664 95B6 21098E     LD HL,TAMPON+9
665 95B9 ED4BFA91  LD BC,(ACC+2)
666 95BD EDB0       LDIR
667 95BF ED530220   LD (LISTES+2),DE
668 95C3 010900     LD BC,9
669 95C6 2A0020     LD HL,(LISTES)
670 95C9 09         ADD HL,BC
671 95CA 220020     LD (LISTES),HL
672 95CD D1         POP DE
673 95CE E1         POP HL
674 95CF EDB0       LDIR
675 95D1 2A0420     LD HL,(LISTES+4)
676 95D4 23         INC HL
677 95D5 220420     LD (LISTES+4),HL
678 95D8 EB         EX DE,HL
679 95D9 210D20     LD HL,LISTES+13
680 95DC 010900     LD BC,9
681 95DF D5         BCLENRG: PUSH DE
682 95E0 5E         LD E,(HL)
683 95E1 23         INC HL
684 95E2 56         LD D,(HL)
685 95E3 EB         EX DE,HL
686 95E4 09         ADD HL,BC
687 95E5 EB         EX DE,HL
688 95E6 72         LD (HL),D
689 95E7 2B         DEC HL
690 95E8 73         LD (HL),E
691 95E9 D1         POP DE
692 95EA 09         ADD HL,BC
693 95EB 1B         DEC DE
694 95EC 7A         LD A,D
695 95ED B3         OR E
696 95EE 20EF       JR NZ,BCLENRG
697 95F0 CD6CBB     CALL CLS0
698
699 95F3 0603       CHOIX3: LD B,3
700 95F5 CDCA92     CALL CHOIX
701 95F8 6190       DW CENC
702 95FA 0000       DW 0
703 95FC 6A90       DW CRET
704 95FE A492       DW RETOUR
705 9600 7190       DW CSAUV
706 9602 0B95       DW SAUVER
707
708 9604 3A008E     DEJAUN: LD A,(TAMPON)
709 9607 CBFF       SET 7,A
710 9609 32008E     LD (TAMPON),A
711 960C CB7E       BIT 7,(HL)
712 960E C0         RET NZ
713 960F CBFE       SET 7,(HL)
714 9611 23         INC HL
715 9612 CBFE       SET 7,(HL)
716 9614 2B         DEC HL
717 9615 C9         RET
718
719
720 ;ENTREE DES DIVERSES LIGNES
721 ;-----
722
723
724 9616 21DB90     ENRADR: LD HL,MESS4
725 9619 CD4296     CALL VERSIX
726 961C F5         PUSH AF
727 961D 0628       LD B,40
728 961F CD879A     CALL COPIE
729 9622 21EB90     LD HL,MESS5-2
730 9625 F1         POP AF
731 9626 2817       JR Z,FINADR
732 9628 E5         PUSH HL
733 9629 0628       LD B,40

```

734	962B	CD839A	CALL	MINAB	;MIN (longueur,40) dans B
735	962E	21FF8E	LD	HL,ZONE-1	;Debut ligne entree -1
736	9631	70	LD	(HL),B	;Stocker longueur tronconnee
737	9632	CD769A	CALL	RELOCATE	;Replacer curseur
738	9635	CD3E9B	CALL	PRCHAIN	;Ecrire ligne entree
739	9638	E1	POP	HL	;Numero de ligne
740	9639	34	INC	(HL)	;Incrementer
741	963A	3E36	LD	A,"6"	;Depasse 5?
742	963C	BE	CP	(HL)	
743	963D	20D7	JR	NZ,ENRADR	;Non, enreg. ligne suivante
744	963F	3631	FINADR:	LD (HL),"1"	;Remettre chiffre 1 dans mess.
745	9641	C9	RET		
746	9642	DDE9	VERSIX:	JF (IX)	
747					
748					
749	9644	21D490	ENRPRE:	LD HL,MESS3	; "Prenom:"
750	9647	CD9D9A	CALL	INPUTVID	;Editer ligne vide en ZONE
751	964A	282F	VRFPRE:	JR Z,NOPRE	;Longueur nulle? Pas de prenom
752	964C	0610	LD	B,16	
753	964E	E5	PUSH	HL	
754	964F	F5	PUSH	AF	
755	9650	CD839A	CALL	MINAB	;Tronconner a 16
756	9653	2B	DEC	HL	;Debut ligne entree -1
757	9654	77	LD	(HL),A	;Stocker longueur tronconnee
758	9655	23	PRENOM2:	INC HL	;Premiere lettre
759	9656	CD359A	CALL	MAJUS	;Transformer en majuscule
760	9659	77	LD	(HL),A	;Resauver
761	965A	380D	JR	C,SUITPRE	;Bien lettre? Alors continuer
762	965C	CDA69A	ERRPRE:	CALL INPUTERR	;Envoyer message d'erreur
763	965F	18E9	JR	VRFPRE	;Et recommencer
764	9661	23	BCLPRE:	INC HL	;Lettre suivante
765	9662	7E	LD	A,(HL)	
766	9663	CD489A	CALL	MINUS	;Transformer en minuscule
767	9666	77	LD	(HL),A	;Resauver
768	9667	3016	JR	NC,VERTIRET	;Pas une lettre? Alors tiret?
769	9669	10F6	SUITPRE:	DJNZ BCLPRE	;Continuer jusqu'a la fin
770	966B	F1	POP	AF	
771	966C	E1	POP	HL	
772	966D	E5	PUSH	HL	
773	966E	0610	LD	B,16	
774	9670	CD8A9A	CALL	COPIE2	;Copier prenom dans le bloc
775	9673	E1	POP	HL	
776	9674	CD769A	CALL	RELOCATE	;Replacer le curseur
777	9677	2B	DEC	HL	
778	9678	C33E9B	JF	PRCHAIN	;Et ecrire le prenom, et fin
779	967B	AF	NOPRE:	XOR A	;Pas de prenom, alors
780	967C	12	LD	(DE),A	;Mettre un zero dans le bloc
781	967D	13	INC	DE	
782	967E	C9	RET		
783	967F	FE2D	VERTIRET:	CP "-"	;Pas une lettre, alors tiret?
784	9681	20D9	JR	NZ,ERRPRE	;Non, alors erreur
785	9683	05	DEC	B	;Un caractere de moins
786	9684	28D6	JR	Z,ERRPRE	;Tiret dernier, alors erreur
787	9686	37	SCF		
788	9687	18CC	JR	PRENOM2	;Traiter deuxieme prenom
789					
790					
791	9689	21D090	ENRNOM:	LD HL,MESS2	; "Nom:"
792	968C	CD9D9A	CALL	INPUTVID	;Enregistrer le nom
793	968F	B7	VRFNOM:	OR A	;Longueur nulle ?
794	9690	2843	JR	Z,ERRNOM+2	;Oui, alors erreur
795	9692	E5	PUSH	HL	
796	9693	F5	PUSH	AF	
797	9694	0620	LD	B,32	
798	9696	CD839A	CALL	MINAB	;Tronconner a 32
799	9699	78	LD	A,B	
800	969A	2B	DEC	HL	
801	969B	77	LD	(HL),A	;Stocker longueur
802	969C	23	BCLNOM:	INC HL	
803	969D	CD359A	CALL	MAJUS	;Transformer en majuscule
804	96A0	77	LD	(HL),A	
805	96A1	D4C496	CALL	NC,AUTCARAC	;Pas une lettre,alors verifier
806	96A4	10F6	DJNZ	BCLNOM	;Toutes les lettres
807	96A6	F1	POP	AF	
808	96A7	E1	POP	HL	
809	96A8	E5	PUSH	HL	

```

810 96A9 0620 LD B,32
811 96AB CDBA9A CALL COPIE2 ;Copier nom obtenu
812 96AE E1 POP HL
813 96AF CD769A CALL RELOCATE ;Curseur
814 96B2 2B DEC HL
815 96B3 CD3E9B CALL PRCHAIN ;Ecrire le nom
816 96B6 D5 PUSH DE
817 96B7 21008F LD HL,ZONE ;Et copier...
818 96BA 11008E LD DE,TAMPON ;...en debut de pointeur
819 96BD 010300 LD BC,3 ;...les 3 premieres lettres
820 96C0 EDB0 LDIR
821 96C2 D1 POP DE ;Restituer adr.-bloc courante
822 96C3 C9 RET
823 96C4 E5 AUTCARAC: PUSH HL
824 96C5 210092 LD HL,TABCARAC ;Table des carac. autorises
825 96C8 4F LD C,A ;Caractere a verifier
826 96C9 7E BCLCARAC: LD A,(HL) ;Caractere de la table
827 96CA B7 OR A ;Atteint fin de la table (0)?
828 96CB 2806 JR Z,ERRNOM ;Oui, alors erreur
829 96CD 23 INC HL
830 96CE B9 CP C ;Meme carac.?
831 96CF 20F8 JR NZ,BCLCARAC ;Non, chercher encore
832 96D1 E1 POP HL ;Oui, ok.
833 96D2 C9 RET
834 96D3 E1 ERRNOM: POP HL ;Vidanger la pile
835 96D4 E1 POP HL
836 96D5 CDA69A CALL INPUTERR ;Demander nouveau nom
837 96D8 18B5 JR VRFNOM ;Et recommencer
838
839
840 96DA 21ED90 ENRNUM: LD HL,MESS5 ;"Numero:"
841 96DD CD9D9A CALL INPUTVID ;Demander
842 96E0 FE10 VRFNUM: CP 16 ;Plus de 16 caracteres ?
843 96E2 3061 JR NC,ERRNUM ;Oui, erreur
844 96E4 47 LD B,A
845 96E5 E5 PUSH HL
846 96E6 F5 PUSH AF
847 96E7 2B DEC HL
848 96E8 77 LD (HL),A ;Sauver longueur du numero
849 96E9 23 BCLNUM: INC HL
850 96EA CD2D9A CALL CHIFFRE ;Bien un chiffre ?
851 96ED D44197 CALL NC,VERSPACE ;Non, alors un espace ?
852 96F0 10F7 DJNZ BCLNUM
853 96F2 0604 LD B,4 ;Faire 4 nombres pour pointeur
854 96F4 23 INC HL ;Commencer par la fin
855 96F5 F1 POP AF
856 96F6 F5 PUSH AF ;Recuperer longueur
857 96F7 4F LD C,A ;Dans C
858 96F8 0C INC C
859 96F9 D5 PUSH DE
860 96FA 11068E LD DE,TAMPON+6 ;Fin emplacement de stock
861 96FD 0D BCLNUM2: DEC C ;Caractere suivant
862 96FE 2B DEC HL
863 96FF CD2D9A CALL CHIFFRE ;Est un chiffre ?
864 9702 30F9 JR NC,BCLNUM2 ;Non, suivant
865 9704 D5 PUSH DE
866 9705 D630 SUB "0" ;Transformer en nombre 0 a 9
867 9707 5F LD E,A ;Dans E
868 9708 2B DEC HL ;Caractere suivant
869 9709 CD2D9A CALL CHIFFRE ;Est un chiffre ?
870 970C 300A JR NC,SUITNUM2 ;Non, passer
871 970E 0D DEC C ;Compteur de carac.
872 970F D630 SUB "0" ;Transformer en nombre 0 a 9
873 9711 87 ADD A,A
874 9712 57 LD D,A
875 9713 87 ADD A,A
876 9714 87 ADD A,A
877 9715 82 ADD A,D ;Multiplier par 10
878 9716 83 ADD A,E ;Rajouter l'autre
879 9717 5F LD E,A ;Total: nombre de 0 a 99
880 9718 7B SUITNUM2: LD A,E
881 9719 D1 POP DE
882 971A 12 LD (DE),A ;Stocker nombre dans pointeur
883 971B 1B DEC DE
884 971C 2B BCLNUM3: DEC HL
885 971D 0D DEC C ;Suivant

```

```

886 971E FA3D97      JP  M,NODXCHIF      ;Plus de caracteres, terminer
887 9721 7E         LD  A,(HL)
888 9722 FE20        CP  020H
889 9724 20F6        JR  NZ,BCLNUM3     ;Jusqu'au prochain espace
890 9726 10D5        DJNZ BCLNUM2       ;Continuer, jusqu'a 4 nombres
891 9728 D1          SUIITNUM3:POP DE
892 9729 F1          POP AF
893 972A 0610        LD  B,16
894 972C E1          POP HL
895 972D E5          PUSH HL
896 972E CD879A      CALL COPIE         ;Copier le numero (chaine)
897 9731 E1          POP HL
898 9732 D0          RET NC            ;Termine si pas de numero
899 9733 CD769A      CALL RELOCATE     ;Curseur
900 9736 2B          DEC HL
901 9737 C33E9B      JP  PRCHAIN       ;Ecrire le numero et fin
902 973A AF          XOR  A
903 973B 12          LD  (DE),A
904 973C 1B          DEC  DE
905 973D 10FB        NODXCHIF:DJNZ $-3 ;Remplir pointeur avec des 0
906 973F 18E7        JR  SUIITNUM3     ;Et continuer
907 9741 FE20        VERSPACE:CP 32    ;Est un espace (ASCII 20H)?
908 9743 C8          RET Z            ;Oui, ok
909 9744 F1          POP AF           ;Non, vider pile et erreur
910 9745 CDA69A      ERRNUM:CALL INPUTERR ;Correction
911 9748 1896        JR  VRFNUM       ;Et recommencer
912
913
914 ;MODIFICATION D'UNE REFERENCE
915 ;-----
916
917
918 974A 3E07        MODIFIER:LD  A,7   ;Chaine 7= "MODIFIER"
919 974C CDD497      CALL SUPMOD       ;Partie commune avec supprimer
920 974F 22F891      LD  (ACC),HL     ;Adresse ancien pointeur
921 9752 D5          PUSH DE          ;Adresse bloc a modifier
922 9753 D5          PUSH DE
923 9754 CD6CBB      CALL CLS0
924 9757 E1          POP HL          ;Dans HL
925 9758 CDB797      CALL LGNCOPY     ;Copier le nom en ZONE
926 975B E5          PUSH HL
927 975C 11098E      LD  DE,TAMPON+9 ;Place pour faire nouveau bloc
928 975F 21D090      LD  HL,MESS2    ;"Nom:"
929 9762 CDA99A      CALL INPUT      ;Editer l'ancien nom
930 9765 CDBF96      CALL VRFNOM     ;Verifier le nouveau nom
931 9768 E1          POP HL
932 9769 FD360001    LD  (IY+0),1    ;Flag a 1 pour prenom
933 976D CDB797      CALL LGNCOPY     ;Idem pour le prenom
934 9770 E5          PUSH HL
935 9771 21D490      LD  HL,MESS3
936 9774 CD0E00      CALL 0EH        ;Call (BC)
937 9777 CD4A96      CALL VRFPRE
938 977A E1          POP HL
939 977B FD360064    LD  (IY+0),100 ;Flag <>1 pour la suite
940 977F CDB797      CALL LGNCOPY     ;Idem pour le numero
941 9782 E5          PUSH HL
942 9783 21ED90      LD  HL,MESS5
943 9786 CDA99A      CALL INPUT
944 9789 CDE096      CALL VRFNUM
945 978C E1          POP HL
946 978D 22FA91      LD  (ACC+2),HL  ;Stocker
947 9790 DD21AA97    LD  IX,MODADR   ;Sub-routine pour ENRADR
948 9794 CD1696      CALL ENRADR     ;Editer l'adresse
949 9797 E1          POP HL
950 9798 D5          PUSH DE
951 9799 EB          EX  DE,HL
952 979A 2AF891      LD  HL,(ACC)    ;Adresse ancien pointeur
953 979D CD2698      CALL DETRUIRE   ;Enlever ancien
954 97A0 D1          POP DE
955 97A1 2A0220      LD  HL,(LISTES+2) ;Fin listes
956 97A4 22078E      LD  (TAMPON+7),HL ;...comme adresse de bloc
957 97A7 C38795      JP  FINMODIF    ;Integrer nouv. bloc et point.
958
959 97AA E5          MODADR: PUSH HL
960 97AB 2AFA91      LD  HL,(ACC+2)  ;Adresse dans ancien bloc
961 97AE CDB797      CALL LGNCOPY    ;Copier une ligne

```

```

962 97B1 22FA91      LD  (ACC+2),HL
963 97B4 E1          POP  HL
964 97B5 C5          PUSH BC                ;JP (BC)
965 97B6 C9          RET
966
967 97B7 7E          LGNCOPY: LD  A,(HL)      ;Longueur de la ligne
968 97B8 23          INC  HL
969 97B9 B7          OR   A                ;Nulle ?
970 97BA 280F        JR   Z,LGNVID         ;Oui, aviser
971 97BC D5          PUSH DE
972 97BD 4F          LD  C,A
973 97BE AF          XOR  A
974 97BF 47          LD  B,A                ;Longueur dans BC
975 97C0 11008F      LD  DE,ZONE
976 97C3 EDB0        LDIR                    ;Recopier dans ZONE
977 97C5 12          LD  (DE),A            ;Mettre un zero a la fin
978 97C6 D1          POP  DE
979 97C7 01A99A      LD  BC,INPUT          ;Ligne sera editee
980 97CA C9          RET
981 97CB 019D9A      LGNVID: LD  BC,INPUTVID ;Ligne vide sera editee
982 97CE FD3500      DEC  (IY+0)           ;Est au prenom ?
983 97D1 C8          RET  Z                ;Oui, ok.
984 97D2 2B          DEC  HL                ;Non,ne plus avancer dans bloc
985 97D3 C9          RET                    ;(car derniere ligne atteinte)
986
987
988 97DA FD7701      SUPMOD: LD  (IY+1),A    ;Stocker code operation
989 97DB 21F395      LD  HL,CHOIX3        ;Adresse de retour
990 97DA 221A93      LD  (BREAKAD),HL    ;... en cas de Break
991 97DD 260C        LD  H,12              ;Centrage
992 97DF CDF89A      CALL FENZ            ;Fenetre inferieure
993 97E2 CD5A9A      CALL POSHAUT         ;Preparer place
994 97E5 2C          INC  L
995 97E6 CD75BB      CALL LOCATE          ;Et positionner curseur
996 97E9 CD669B      CALL PEN0
997 97EC 215791      LD  HL,MESS11
998 97EF CD3E9B      CALL PRCHAIN         ;"NOM A "(traiter)?"
999 97F2 11048E      LD  DE,TAMPON+4     ;Emplacement
1000 97F5 CD8996      CALL ENRNOM          ;Demander nom a traiter
1001 97FB CD6CBB      CALL CLS0
1002 97FB CD709A      CALL LOC+15
1003 97FE 11008E      LD  DE,TAMPON       ;Emplacement 3 prem. lettres
1004 9801 CD1C93      CALL SEEKNM          ;Rechercher
1005 9804 D2F894      JP   NC,NONTROUV    ;Pas trouve, message et fin
1006 9807 CD5199      CALL CONFIRM         ;Demander confirmation
1007 980A D8          RET  C                ;Confirmation, ok.
1008 980B F1          POP  AF               ;Vider pile
1009 980C C3A492      JP   RETOUR          ;Retourner au menu principal
1010
1011
1012 ;SUPPRIMER UNE REFERENCE DE L'ANN.
1013 ;-----
1014
1015
1016 980F 3E08          SUPPRIME:LD  A,8      ;Chaine 8= "SUPPRIMER"
1017 9811 CDD497      CALL SUPMOD          ;Partie commune
1018 9814 CD2698      CALL DETRUIRE        ;Detruire pointeur et bloc
1019 9817 CD6CBB      CALL CLS0
1020 981A CD619A      CALL LOC
1021 981D 218191      LD  HL,MESS14        ;"Suppression effectuee."
1022 9820 CD3E9B      CALL PRCHAIN
1023 9823 C3F395      JP   CHOIX3          ;Choix encore/ retour/ sauver
1024
1025 ;RETIRER UN BLOC ET UN POINTEUR
1026
1027 9826 D5          DETRUIRE:PUSH DE      ;Adresse du bloc
1028 9827 CB7E        BIT  7,(HL)          ;Appartient a une serie ?
1029 9829 C4BD98      CALL NZ,SERIE        ;oui, aviser
1030 982C 22F891      LD  (ACC),HL
1031 982F 62          LD  H,D
1032 9830 6B          LD  L,E                ;Debut du bloc
1033 9831 0600        LD  B,0
1034 9833 CDB798      CALL SAUTLGN         ;Sauter le nom
1035 9836 CDB798      CALL SAUTLGN         ;Sauter le prenom
1036 9839 CDB798      CALL SAUTLGN         ;Sauter autres lignes
1037 983C 20FB        JR   NZ,BCLSAUT     ;Jusqu'a ligne vide = fin

```

1038	983E	E5	PUSH	HL	;Fin du bloc	
1039	983F	44	LD	B,H		
1040	9840	4D	LD	C,L		
1041	9841	B7	OR	A		
1042	9842	ED52	SBC	HL,DE	;Longueur du bloc	
1043	9844	E5	PUSH	HL		
1044	9845	DDE1	POP	IX	;Dans IX	
1045	9847	2A0220	LD	HL,(LISTES+2)	;Fin listes	
1046	984A	23	INC	HL		
1047	984B	B7	OR	A		
1048	984C	ED42	SBC	HL,BC	;Moins fin du bloc	
1049	984E	44	LD	B,H		
1050	984F	4D	LD	C,L	;=longueur blocs derriere	
1051	9850	E1	POP	HL	;Debut de bloc	
1052	9851	EDB0	LDIR		;Bloc detruit	
1053	9853	2AF891	LD	HL,(ACC)	;Debut du pointeur	
1054	9856	E5	PUSH	HL		
1055	9857	010900	LD	BC,9		
1056	985A	09	ADD	HL,BC	;Fin du pointeur	
1057	985B	E5	PUSH	HL		
1058	985C	EB	EX	DE,HL	;Dans DE	
1059	985D	B7	OR	A	;Dans HL:fin listes provisoire	
1060	985E	ED52	SBC	HL,DE	;Longueur du reste	
1061	9860	44	LD	B,H		
1062	9861	4D	LD	C,L		
1063	9862	E1	POP	HL	;Fin du pointeur	
1064	9863	D1	POP	DE	;Debut du pointeur	
1065	9864	EDB0	LDIR		;Detruire le pointeur	
1066	9866	1B	DEC	DE		
1067	9867	ED530220	LD	(LISTES+2),DE	;Nouvelle fin listes	
1068	986B	ED4B0420	LD	BC,(LISTES+4)	;Nombre d'elements	
1069	986F	0B	DEC	BC	;Un de moins	
1070	9870	ED430420	LD	(LISTES+4),BC		
1071	9874	78	LD	A,B		
1072	9875	B1	OR	C	;Plus d'elements?	
1073	9876	CAA899	JP	Z,CREER	;Alors recreer un annuaire	
1074	9879	210D20	LD	HL,LISTES+13	;lere adresse, 1er pointeur	
1075	987C	FDE3	EX	(SP),IY	;Fin bloc detruit dans IY	
1076	987E	C5	BCLEDETR:	PUSH	BC	;Modifier adr. des pointeurs
1077	987F	010900	LD	BC,9		
1078	9882	5E	LD	E,(HL)	;Prendre adresse de bloc	
1079	9883	23	INC	HL		
1080	9884	56	LD	D,(HL)		
1081	9885	FDE5	PUSH	IY		
1082	9887	E3	EX	(SP),HL	;Fin bloc detruit dans HL	
1083	9888	CD229A	CALL	ICOMP	;Comparer	
1084	988B	E1	POP	HL	;Bloc detruit avant ou apres?	
1085	988C	DCAD98	CALL	C,DIFF	;Si avant, soustraire longueur	
1086	988F	EB	EX	DE,HL		
1087	9890	B7	OR	A		
1088	9891	ED42	SBC	HL,BC	;Soustraire 9 en plus	
1089	9893	EB	EX	DE,HL		
1090	9894	72	LD	(HL),D	;Et restocker le tout	
1091	9895	2B	DEC	HL		
1092	9896	73	LD	(HL),E		
1093	9897	09	ADD	HL,BC	;Pointeur suivant	
1094	9898	C1	POP	BC	;Restituer compteur	
1095	9899	0B	DEC	BC		
1096	989A	78	LD	A,B		
1097	989B	B1	OR	C	;Atteint zero ?	
1098	989C	20E0	JR	NZ,BCLEDETR	;Non, continuer	
1099	989E	FDE1	POP	IY	;Restituer IY	
1100	98A0	2A0020	LD	HL,(LISTES)	;Debut liste 2	
1101	98A3	010900	LD	BC,9		
1102	98A6	B7	OR	A		
1103	98A7	ED42	SBC	HL,BC	;Retirer 9	
1104	98A9	220020	LD	(LISTES),HL		
1105	98AC	C9	RET			
1106						
1107	98AD	DDE5	DIFF:	PUSH	IX	;Longueur du bloc detruit
1108	98AF	E3	EX	(SP),HL	;Dans HL	
1109	98B0	EB	EX	DE,HL		
1110	98B1	B7	OR	A		
1111	98B2	ED52	SBC	HL,DE	;Soustraire de DE	
1112	98B4	EB	EX	DE,HL		
1113	98B5	E1	POP	HL	;Restituer HL	

```

1114 98B6 C9          RET
1115 98B7 7E          SAUTLGN: LD  A,(HL)      ;Longueur de la ligne
1116 98B8 3C          INC  A                ;Compter octet de longueur
1117 98B9 4F          LD   C,A              ;B vaut 0
1118 98BA 09          ADD  HL,BC            ;Ajouter a HL
1119 98BB 3D          DEC  A                ;Remplacer longueur et flag z
1120 98BC C9          RET
1121
1122 98BD 010900       SERIE:  LD  BC,9        ;Nom appartient a une serie
1123 98C0 E5          PUSH HL
1124 98C1 23          INC  HL
1125 98C2 CB7E       BIT  7,(HL)          ;Dernier de la serie ?
1126 98C4 2013       JR   NZ,DESERIE     ;Oui, faire en fonction
1127 98C6 09          ADD  HL,BC            ;Pointeur suivant, +1
1128 98C7 CB7E       BIT  7,(HL)          ;Dernier de la serie ?
1129 98C9 280C       JR   Z,FINSER2      ;Non serie de 3 au moins, ok.
1130 98CB CDBB99     CALL  ELSERIE+1      ;Appartient a la meme serie ?
1131 98CE E1          POP  HL
1132 98CF D8          RET  C                ;Oui, ok.
1133 98D0 E5          PUSH HL
1134 98D1 09          FINSER1: ADD HL,BC    ;Suivant sera seul...
1135 98D2 CBBE       RES  7,(HL)          ;Annuler bits 7
1136 98D4 23          INC  HL
1137 98D5 CBBE       RES  7,(HL)
1138 98D7 E1          FINSER2: POP HL      ;Restituer pointeur initial
1139 98D8 C9          RET
1140 98D9 B7          DESERIE: OR  A
1141 98DA ED42       SBC  HL,BC            ;Pointeur precedent
1142 98DC CBF8       SET  7,(HL)          ;Sera dernier de la serie
1143 98DE ED42       SBC  HL,BC            ;Precedent encore
1144 98E0 CDBB99     CALL  ELSERIE+1      ;Appartient a la meme serie ?
1145 98E3 38F2       JR   C,FINSER2      ;Oui, ok.
1146 98E5 18EA       JR   FINSER1        ;Non serie de 2.
1147
1148
1149
1150 ;-----
1151 ;SUB-ROUTINES IMPORTANTES
1152 ;-----
1153
1154
1155 ;CHOIX ENTRE REFERENCES SEMBLABLES
1156 ;ET CONFIRMATION.
1157
1158 98E7 E5          PLUSIEUR: PUSH HL     ;Choix dans une serie
1159 98E8 E5          PUSH HL              ;Premier de la serie
1160 98E9 3C          INC  A                ;Nombre d'elements
1161 98EA 4F          LD   C,A              ;
1162 98EB 47          LD   B,A              ;
1163 98EC 216191     LD   HL,MESS12        ;"Choisissez une lettre:"
1164 98EF CD3E9B     CALL  PRCHAIN         ;
1165 98F2 CD159A     CALL  ALALIGN2        ;Sauter 1 ligne et a la ligne
1166 98F5 E1          POP  HL              ;Premier pointeur
1167 98F6 110700     LD   DE,7             ;
1168 98F9 19          ADD  HL,DE            ;Lieu de l'adresse du bloc
1169 98FA 13          INC  DE               ;
1170 98FB 13          INC  DE               ;DE= 9
1171 98FC 3E61       LD   A,"a"           ;Premiere lettre
1172 98FE F5          BCLPLUS: PUSH AF      ;Ecrire les noms et prenom
1173 98FF CD5ABB     CALL  PRINT           ;
1174 9902 3E29       LD   A,")"           ;
1175 9904 CD5ABB     CALL  PRINT           ;Ecrire a), b), etc.
1176 9907 E5          PUSH HL
1177 9908 7E          LD   A,(HL)          ;
1178 9909 23          INC  HL              ;
1179 990A 66          LD   H,(HL)          ;
1180 990B 6F          LD   L,A              ;LD HL,(HL): adresse du bloc
1181 990C CD3E9B     CALL  PRCHAIN         ;Ecrire le nom
1182 990F 23          INC  HL              ;
1183 9910 3E20       LD   A,32            ;
1184 9912 CD5ABB     CALL  PRINT           ;Puis un espace
1185 9915 7E          LD   A,(HL)          ;Longueur du prenom
1186 9916 B7          OR   A                ;Nulle ?
1187 9917 C43E9B     CALL  NZ,PRCHAIN     ;Non, alors ecrire le prenom
1188 991A CD189A     CALL  ALALIGN2        ;Ligne suivante
1189 991D CD09BB     CALL  CSCRUT         ;Touche enfoncee

```

1190	9920	DC06BB	CALL C, WAIT	;Oui, attendre et laisser lire
1191	9923	E1	POP HL	
1192	9924	19	ADD HL, DE	;Pointeur suivant
1193	9925	F1	POP AF	;Lettre
1194	9926	3C	INC A	;Lettre suivante
1195	9927	10D5	DJNZ BCLPLUS	;Ecrire tous
1196	9929	CD06BB	BCLLETT: CALL WAIT	;Attendre une lettre
1197	992C	FEFC	CP 0FCH	;Break ?
1198	992E	C8	RET Z	;Oui, pas de choix, termine
1199	992F	CD489A	CALL MINUS	;Transformer en minuscule
1200	9932	30F5	JR NC, BCLLETT	;Pas une lettre, ignorer
1201	9934	D661	SUB "a"	;Numero correspondant
1202	9936	B9	CP C	;Depasse le nombre de ref. ?
1203	9937	30F0	JR NC, BCLLETT	;Oui, alors erreur, ignorer
1204	9939	4F	LD C, A	
1205	993A	87	ADD A, A	
1206	993B	81	ADD A, C	
1207	993C	4F	LD C, A	
1208	993D	87	ADD A, A	
1209	993E	81	ADD A, C	;Multiplier par 9
1210	993F	4F	LD C, A	;Noter que B=0 (DJNZ en 1195)
1211	9940	C5	PUSH BC	
1212	9941	CD6CBB	CALL CLS0	
1213	9944	C1	POP BC	
1214	9945	E1	POP HL	;Premier pointeur
1215	9946	09	ADD HL, BC	;Pointeur demande
1216	9947	E5	PUSH HL	
1217	9948	010700	LD BC, 7	
1218	994B	09	ADD HL, BC	;Sur adresse de bloc
1219	994C	5E	LD E, (HL)	
1220	994D	23	INC HL	
1221	994E	56	LD D, (HL)	;Placer dans DE
1222	994F	E1	POP HL	;Pointeur demande
1223	9950	AF	XOR A	;Demander confirmation
1224				
1225	9951	C2E798	CONFIRM: JP NZ, PLUSIEUR	;Plusieurs ref., alors quelle ?
1226	9954	D5	PUSH DE	;Adresse de bloc
1227	9955	EB	EX DE, HL	;Aussi dans HL
1228	9956	CDED99	CALL PRNOM	;Ecrire la ref. en entier
1229	9959	CD1D9A	CALL DESCEND	;Ligne suivante
1230	995C	217891	LD HL, MESS13	; (traitement) " (O/N)?"
1231	995F	CD3E9B	CALL PRCHAIN	
1232	9962	EB	EX DE, HL	;Dans HL adresse pointeur
1233	9963	D1	POP DE	;Restituer adresse du bloc
1234	9964	CD06BB	CALL WAIT	;Attendre une touche
1235	9967	FEFC	CP 0FCH	;Break ?
1236	9969	C8	RET Z	;Oui, pas de confirmation, fin
1237	996A	CD369A	CALL MAJUS+1	;Transformer en majuscule
1238	996D	3F	CCF	;N'est pas une lettre?
1239	996E	D8	RET C	;Oui, alors comme Oui
1240	996F	FE4E	CP "N"	;Touche N?
1241	9971	C8	RET Z	;Oui, pas de confirmation: c=0
1242	9972	37	SCF	;Sinon, c=1
1243	9973	C9	RET	;Confirmation faite
1244				
1245			;LECTURE DE L'ANNUAIRE	
1246				
1247	9974	21E991	LIRFICH: LD HL, NOMFICH	;Nom du fichier
1248	9977	11008E	LD DE, TAMPON	;Tampon de lecture
1249	997A	060B	LD B, 11	;Onze lettres dans le nom
1250	997C	CD77BC	CALL OPEN	;Ouvrir le fichier
1251	997F	300A	JR NC, LECTERR	;Pas trouve, aviser
1252	9981	210020	LD HL, LISTES	;Emplacement du fichier
1253	9984	CD83BC	CALL KINFICH	;Lire le fichier
1254	9987	CD7ABC	CALL CLOSE	;Et le fermer
1255	998A	D8	RET C	;Retour si tout va bien
1256	998B	F1	LECTERR: POP AF	;Detruire adresse de retour
1257	998C	CD6CBB	CALL CLS0	
1258	998F	CD619A	CALL LOC	;Curseur
1259	9992	CD669B	CALL PEN0	
1260	9995	219A90	LD HL, MESS0	; "Annuaire absent..."
1261	9998	CD3E9B	CALL PRCHAIN	
1262	999B	0602	LD B, 2	;Deux possibilites
1263	999D	CDCA92	CALL CHOIX	
1264	99A0	4390	DW CCHGDISC	;Changer de disquette
1265	99A2	9E92	DW RETOUR-6	

```

1266 99A4 3890      DW  CCREER          ;Ou creer un annuaire vide
1267 99A6 A899      DW  CREER
1268
1269 99A8 210620     CREER: LD  HL,LISTES+6 ;Debut des listes (vides)
1270 99AB 220020     LD  (LISTES),HL      ;Est aussi debut de la liste 2
1271 99AE 220220     LD  (LISTES+2),HL   ;Et sa fin
1272 99B1 210000     LD  HL,0             ;Zero elements
1273 99B4 220420     LD  (LISTES+4),HL
1274 99B7 C3A492     JP  RETOUR
1275
1276
1277                ;-----
1278                ;PETITES ROUTINES COMMUNES
1279                ;-----
1280
1281
1282 99BA 23          ELSERIE: INC  HL      ;Appartient a la meme serie ?
1283 99BB 7E         LD  A,(HL)          ;Deuxieme caractere
1284 99BC 2F         CPL                    ;Bit 7=0 si dern. autre serie
1285 99BD 2B         DEC  HL              ;Premier caractere
1286 99BE A6         AND  (HL)           ;Bit 7=0 si pas dans une serie
1287 99BF 17        RLA                    ;Donc c=1 si dans meme serie
1288 99C0 C9        RET
1289
1290 99C1 CDC499     FOIS9:  CALL  FOIS9+3 ;Multiplier par 9 (3*3)
1291 99C4 54         LD  D,H              ;Multiplier par 3
1292 99C5 5D        LD  E,L
1293 99C6 29        ADD  HL,HL
1294 99C7 19        ADD  HL,DE
1295 99C8 C9        RET
1296
1297                ;MESSAGE ET ATTENDRE UNE TOUCHE
1298
1299 99C9 CD629B     WAIT:  CALL  PEN1      ;Ecrire en clignotant
1300 99CC 211602     LD  HL,0216H        ;Bas de l'ecran
1301 99CF CD75BB     CALL  LOCATE
1302 99D2 219891     LD  HL,MESS15      ;"Appuyez sur une touche..."
1303 99D5 CD3E9B     CALL  PRCHAIN
1304 99D8 3E0D      LD  A,13
1305 99DA CD5ABB     CALL  PRINT         ;Se replacer en debut de ligne
1306 99DD CD06BB     CALL  CWAIT         ;Attendre une touche
1307 99E0 FEFC      CP   0FCH           ;z =1 si Break
1308 99E2 F5        PUSH AF
1309 99E3 CD669B     CALL  PEN0          ;Ecrire normalement
1310 99E6 3E12      LD  A,18
1311 99E8 CD5ABB     CALL  PRINT         ;Supprimer le message
1312 99EB F1        POP  AF             ;Restituer touche appuee
1313 99EC C9        RET
1314
1315 99ED FD360000    PRNOM: LD  (IY+0),0      ;Ecrire une reference entiere
1316 99F1 CD3E9B     CALL  PRCHAIN      ;Ecrire une ligne
1317 99F4 CD189A     CALL  ALALIGN      ;Curseur, ligne suivante
1318 99F7 23        INC  HL              ;Ligne suivante du bloc
1319 99F8 FD3400     INC  (IY+0)         ;Incrementer flag
1320 99FB 7E         LD  A,(HL)          ;Longueur de la ligne
1321 99FC B7        OR   A              ;Nulle ?
1322 99FD 20F2      JR   NZ,PRNOM+4    ;Non, ecrire lignes suivantes
1323 99FF FD3500     DEC  (IY+0)         ;Prenom de longueur nulle ?
1324 9A02 2019      JR   NZ,DESCEND    ;Non, fin du bloc, terminer
1325 9A04 FD3400     INC  (IY+0)         ;Oui, continuer
1326 9A07 23        INC  HL              ;...avec lignes suivantes
1327 9A08 18E7      JR   PRNOM+4
1328
1329 9A0A CD4E9B     SUIVANT: CALL  HORPOS8 ;Curseur en colonn e 8
1330 9A0D CD339B     CALL  PRCH1        ;Ecrire chaine dans (HL)
1331 9A10 CD1D9A     CALL  DESCEND      ;Descendre de 2 lignes
1332 9A13 1806      JR   DESCEND
1333
1334 9A15 CD1D9A     ALALIGN2:CALL  DESCEND
1335 9A18 3E0D      ALALIGN:LD  A,13   ;Passer a la ligne
1336 9A1A CD5ABB     CALL  PRINT
1337 9A1D 3E0A      DESCEND:LD  A,10   ;Descendre d'une ligne
1338 9A1F C35ABB     JP   PRINT
1339
1340 9A22 7C        ICOMP:LD  A,H      ;Comparer DE et HL
1341 9A23 92        SUB  D              ;Octets forts differents ?

```

```

1342 9A24 2003      JR    NZ,$+5      ;Oui, vers 1346
1343 9A26 7D        LD    A,L          ;
1344 9A27 93        SUB   E            ;Comparer octets faibles
1345 9A28 C8        RET   Z            ;HL = DE, retour (A=0)
1346 9A29 9F        SBC  A,A          ;A= FFH si DE > HL
1347 9A2A D8        RET   C            ;
1348 9A2B 3C        INC  A            ;A=1 sinon
1349 9A2C C9        RET               ;Flags positionnes
1350
1351                ;VERIFICATIONS DE CODES ASCII
1352
1353 9A2D 7E        CHIFFRE: LD    A,(HL)      ;Est un chiffre entre 0 et 9 ?
1354 9A2E FE3A      CP    "9"+1
1355 9A30 D0        RET   NC          ;Non, trop grand
1356 9A31 FE30      CP    "0"
1357 9A33 3F        CCF              ;c=1 si chiffre
1358 9A34 C9        RET
1359
1360 9A35 7E        MAJUS:  LD    A,(HL)      ;Transformer en majuscule
1361 9A36 FE41      CP    "A"
1362 9A38 3F        CCF
1363 9A39 D0        RET   NC          ;Trop petit, pas une lettre
1364 9A3A FE5B      CP    "Z"+1
1365 9A3C D8        RET   C            ;Est deja une majuscule, ok.
1366 9A3D FE61      CP    "a"
1367 9A3F 3F        CCF
1368 9A40 D0        RET   NC          ;Entre Z et a, pas une lettre
1369 9A41 FE7B      CP    "z"+1
1370 9A43 D0        RET   NC          ;Trop grand, pas une lettre
1371 9A44 D620      SUB   32
1372 9A46 37        SCF              ;Transformer en majuscule
1373 9A47 C9        RET              ;Retenue = est bien une lettre
1374
1375 9A48 FE7B      MINUS:  CP    "z"+1      ;Transformer en minuscule
1376 9A4A D0        RET   NC
1377 9A4B FE60      CP    "a"-1
1378 9A4D 3F        CCF
1379 9A4E D8        RET   C
1380 9A4F FE5B      CP    "Z"+1
1381 9A51 D0        RET   NC
1382 9A52 FE41      CP    "A"
1383 9A54 3F        CCF
1384 9A55 D0        RET   NC
1385 9A56 C620      ADD  A,32
1386 9A58 37        SCF
1387 9A59 C9        RET
1388
1389                ;POSITIONNEMENTS DU CURSEUR
1390
1391 9A5A 210101     POSHAUT: LD    HL,101H      ;Initialiser POS,...
1392 9A5D 22FC91   LD    (POS),HL      ;...emplacement du curseur
1393 9A60 C9        RET
1394
1395 9A61 210201     LOC:    LD    HL,102H      ;Placements divers
1396 9A64 180D      JR    LOC2          ;Debut de deuxieme ligne
1397 9A66 211501   LD    HL,115H
1398 9A69 1808      JR    LOC2          ;Debut d'avant-derniere ligne
1399 9A6B 210608   LD    HL,806H
1400 9A6E 1803      JR    LOC2          ;Colonne 8, ligne 6
1401 9A70 210501   LD    HL,0105H      ;Debut de la ligne 5
1402 9A73 C375BB   LOC2:   JP    LOCATE
1403
1404 9A76 E5        RELOCATE: PUSH HL
1405 9A77 2AFC91   LD    HL,(POS)      ;Emplacement du curseur
1406 9A7A 2C        INC  L              ;Ligne suivante
1407 9A7E 22FC91   LD    (POS),HL
1408 9A7E CD75BB   CALL LOCATE         ;Remplacer curseur
1409 9A81 E1        POP  HL
1410 9A82 C9        RET
1411
1412 9A83 B8        MINAB:  CP    B            ;MIN (A,B) dans B
1413 9A84 D0        RET   NC          ;A > B ok.
1414 9A85 47        LD    B,A          ;Sinon B= A
1415 9A86 C9        RET
1416
1417 9A87 B7        COPIE:  OR    A            ;Copier ligne entree dans bloc

```

```

1418 9A88 280F      JR      Z,COPVID      ;Si vide, mettre un zero
1419 9A8A CD839A    COPIE2: CALL MINAB      ;Tronconner
1420 9A8D 78        LD      A,B
1421 9A8E 0600      LD      B,0
1422 9A90 E5        PUSH   HL
1423 9A91 12        LD      (DE),A        ;Longueur de la ligne
1424 9A92 13        INC    DE
1425 9A93 4F        LD      C,A           ;BC= longueur
1426 9A94 EDB0      LDIR                   ;Copier
1427 9A96 E1        POP    HL
1428 9A97 37        SCF                               ;Retenue= ligne non vide
1429 9A98 C9        RET
1430 9A99 12        COPVID: LD      (DE),A      ;Mettre un zero
1431 9A9A 13        INC    DE
1432 9A9B AF        XOR    A               ;c= 0
1433 9A9C C9        RET
1434
1435                ;ENTREES DE LIGNES
1436
1437 9A9D E5        INPUTVID: PUSH HL      ;Editer une ligne vide
1438 9A9E 210000     LD      HL,0
1439 9AA1 22008F     LD      (ZONE),HL    ;Mettre deux zeros
1440 9AA4 1804      JR      INPUT+1
1441 9AA6 21BB90     INPUTERR: LD HL,MESS1  ;Editer avec message d'erreur
1442 9AA9 E5        INPUT:  PUSH HL       ;Editer avec message dans (HL)
1443 9AAA CD629B     CALL   PEN1          ;Ecrire en clignotant
1444 9AAD CD669A     CALL   LOC+5        ;Placer curseur
1445 9AB0 E1        POP    HL           ;Message a ecrire
1446 9AB1 CD3E9B     CALL   PRCHAIN
1447 9AB4 CD669B     CALL   PEN0         ;Ecrire normalement
1448 9AB7 CD189A     CALL   ALALIGNE     ;Ligne suivante (derniere)
1449 9ABA 21008F     LD      HL,ZONE     ;Emplacement de la chaine
1450 9ABD CD5EBD     CALL   EDIT         ;... a editer
1451 9AC0 CA1693     JF     Z,BREAK      ;Break, sauter a BREAKAD
1452 9AC3 E5        PUSH   HL
1453 9AC4 C5        PUSH   BC
1454 9AC5 0600      LD      B,0
1455 9AC7 7E        BCLINPUT: LD A,(HL)    ;Calculer longueur de la ligne
1456 9AC8 23        INC    HL
1457 9AC9 04        INC    B
1458 9ACA B7        OR     A             ;Zero final ?
1459 9ACB 20FA      JR     NZ,BCLINPUT  ;Non, continuer
1460 9ACD 78        LD      A,B         ;Dans A
1461 9ACE 3D        DEC    A            ;Pas compter le zero final
1462 9ACF F5        PUSH   AF
1463 9AD0 CD669A     CALL   LOC+5        ;Replacer curseur
1464 9AD3 3E14      LD      A,20
1465 9AD5 CD5ABB     CALL   PRINT        ;Et effacer le bas de l'ecran
1466 9AD8 F1        POP    AF
1467 9AD9 C1        POP    BC
1468 9ADA E1        POP    HL
1469 9ADB C9        RET
1470
1471 9ADC FD7E00     COMPARE: LD A,(IY+0) ;Comparer 3 ou 4 caracteres
1472 9ADF 47        LD      B,A         ;Nombre dans B
1473 9AE0 EB        EX     DE,HL
1474 9AE1 1A        BCLCOMP: LD A,(DE)
1475 9AE2 E67F     AND    7FH          ;Pas tenir compte des bits 7
1476 9AE4 BE        CP     (HL)
1477 9AE5 C0        RET    NZ           ;Différents, termine
1478 9AE6 23        INC    HL
1479 9AE7 13        INC    DE
1480 9AE8 10F7     DJNZ  BCLCOMP      ;Continuer
1481 9AEA C9        RET                ;Ici flag z mis encore
1482
1483                ;FENETRES
1484
1485 9AEB 3E02      FEN1:  LD      A,2    ;Plus grande fenetre
1486 9AED CDB4BB    CALL   WINDOW      ;Positionner
1487 9AF0 3E00      LD      A,0
1488 9AF2 CD96BB    CALL   PAPER       ;Fond turquoise
1489 9AF5 C36CBB    JP     CLS0        ;Effacer et retour
1490 9AF8 E5        FEN2:  PUSH   HL       ;Plus petite fenetre
1491 9AF9 3E02      LD      A,2
1492 9AFB CDB4BB    CALL   WINDOW      ;Grande fenetre d'abord
1493 9AFE 3E02      LD      A,2

```

```

1494 9B00 CD96BB      CALL PAPER           ;Fond magenta
1495 9B03 CD6CBB      CALL CLS0           ;Vider
1496 9B06 E1          POP HL              ;Dans H colonne
1497 9B07 2E01        LD L,1              ;Et dans L ligne 1
1498 9B09 CD75BB      CALL LOCATE         ;Pour le titre
1499 9B0C 214D91      LD HL,MESS10
1500 9B0F CD3E9B      CALL PRCHAIN        ;Ecrire le titre
1501 9B12 3E03        LD A,3              ;Petite fenetre
1502 9B14 18D7        JR FEN1+2
1503
1504                   ; ECRITURES DE CHAINES ET DE CODES
1505
1506 9B16 C5          PRCODCH: PUSH BC    ;Ecrire chaine codee
1507 9B17 E5          PUSH HL
1508 9B18 21D991      LD HL,CODORD       ;Debut table d'adresse
1509 9B1B B7          OR A                ;Code de la chaine a ecrire
1510 9B1C CC2F9B      CALL Z,DONNECOD    ;Si nul, alors (IY+1)
1511 9B1F 3D          DEC A               ;Nombre entre 0 et 7
1512 9B20 87          ADD A,A             ;Multiplier par deux
1513 9B21 4F          LD C,A
1514 9B22 0600        LD B,0
1515 9B24 09          ADD HL,BC           ;Ajouter a HL
1516 9B25 7E          LD A,(HL)
1517 9B26 23          INC HL
1518 9B27 66          LD H,(HL)
1519 9B28 6F          LD L,A              ;LD HL,(HL):donner adr. chaine
1520 9B29 CD3E9B      CALL PRCHAIN        ;Ecrire cette chaine
1521 9B2C E1          POP HL
1522 9B2D C1          POP BC              ;Et poursuivre ecriture
1523 9B2E C9          RET
1524 9B2F FD7E01      DONNECOD:LD A,(IY+1)
1525 9B32 C9          RET
1526
1527 9B33 D5          PRCH1:  PUSH DE     ;Ecr. chaine pointee par (HL)
1528 9B34 5E          LD E,(HL)
1529 9B35 23          INC HL
1530 9B36 56          LD D,(HL)           ;(HL)
1531 9B37 EB          EX DE,HL            ;Dans HL
1532 9B38 CD3E9B      CALL PRCHAIN        ;Ecrire la chaine
1533 9B3B EB          EX DE,HL
1534 9B3C D1          POP DE
1535 9B3D C9          RET
1536
1537 9B3E C5          PRCHAIN: PUSH BC    ;Ecrire chaine alphanumerique
1538 9B3F 46          LD B,(HL)           ;Longueur de la chaine
1539 9B40 23          INC HL
1540 9B41 7E          LD A,(HL)           ;Caractere suivant
1541 9B42 FE20        CP 20H              ;Inferieur a 20H ?
1542 9B44 D45ABB      CALL NC,PRINT       ;Non, ok.
1543 9B47 DC169B      CALL C,PRCODCH     ;Oui, alors ecr. chaine codee
1544 9B4A 10F4        DJNZ PRCHAIN+2
1545 9B4C C1          POP BC
1546 9B4D C9          RET
1547
1548 9B4E 3E08        HORPOS8: LD A,8     ;Placer curseur sur colonne 8
1549 9B50 E5          PUSH HL
1550 9B51 CD6FBB      CALL HPOS           ;Positionnement horizontal
1551 9B54 E1          POP HL
1552 9B55 C9          RET
1553
1554                   ; ENCREs
1555
1556 9B56 E5          ECR1:  PUSH HL     ;Ecrire en clignotant
1557 9B57 CD629B      CALL PEN1           ;...et en sauvant HL
1558 9B5A E1          POP HL
1559 9B5B C9          RET
1560 9B5C E5          ECR0:  PUSH HL     ;Ecrire en blanc
1561 9B5D CD669B      CALL PEN0           ;...et en sauvant HL
1562 9B60 E1          POP HL
1563 9B61 C9          RET
1564 9B62 3E03        PEN1:  LD A,3     ;Ecrire en clignotant
1565 9B64 1802        JR PEN0+2
1566 9B66 3E01        PEN0:  LD A,1     ;Ecrire en blanc
1567 9B68 C390BB      JP PEN
1568
1569                   ;----- FIN -----

```

1570
1571
1572

END

8. LES ROUTINES SYSTÈME DES CPC

Voici à présent la liste détaillée des routines des trois CPC. L'essentiel de ces routines est commun aux trois, avec des vecteurs aux mêmes adresses. Toutefois, les vecteurs arithmétiques font exception (voir chapitre précédent), d'où d'ailleurs quelques problèmes de compatibilité. De plus, il existe une douzaine de vecteurs des 664 et 6128 qui n'existent pas sur le 464, et un vecteur du 6128 qui n'existe pas sur le 664.

Pour chaque vecteur on trouvera :

- Son adresse ou ses adresses si elles sont distinctes sur les trois CPC.
- Son étiquette (valable pour ZEN), qui est utilisée dans les programmes de cet ouvrage. (Si vous utilisez DAMS, remplacez les ponctuations comme (?) par d'autres caractères.)
- Sa description détaillée. Pour plus de précisions éventuelles, reportez-vous au texte du livre et aux exemples.
- Éventuellement des notes, indiquant les registres modifiés, etc.
- *Instruction* désigne la manière dont les adresses suivantes sont appelées : par un jump (C3H), un restart (on se référera alors à sa description), mais ne figure pas lorsque l'adresse d'exécution coïncide avec l'adresse d'appel.
- Enfin les adresses auxquelles les vecteurs sautent, après le restart qui les lance.

Les vecteurs sont classés par ordre d'adresses croissantes et groupés par types précédés d'une courte description.

■ LES RESTARTS

Les restarts sont des instructions spéciales du Z80 qui permettent de sauter directement aux adresses 0, 8H, 10H, 18H, 20H, 28H, 30H, 38H. Le saut s'effectue comme pour un CALL : le PC est en effet empilé ; mais l'exécution est nettement plus rapide. De plus, ces instructions ne prennent qu'un octet dans la mémoire, contre trois pour un CALL normal.

Outre les routines exécutées par les restarts, les adresses 0 à 40H de la RAM contiennent quelques petites routines supplémentaires qui peuvent être utiles. Ces routines doivent être appelées par un CALL

normal. On ne leur a pas donné d'étiquettes spéciales, leurs adresses ne comprenant que deux chiffres.

Il est peu prudent de modifier les routines des restarts, car ils sont fréquemment utilisés par le système. En particulier, RST 8, RST 18, RST 28 sont les instructions de lancement des vecteurs principaux.

Toutefois, le RST 30 est laissé au bon usage de l'utilisateur.

RST 0

Étiquette : aucune (instruction).

Description : Réinitialisation complète de la machine. Toute la RAM est effacée, sauf la RAM supplémentaire du 6128. Produit en BASIC par l'appui des trois touches (CTRL) (SHIFT) (ESC).

Notes : A éviter !

Adresse d'exécution : 0.

RST 8

Étiquette : aucune (instruction).

Description : Exécution d'une routine d'adresse inférieure à 4000H.

Les deux octets qui suivent le RST 8 dans le programme déterminent à la fois l'adresse de la routine (par leurs bits 0 à 13) et l'état des ROM : bit 15 pour la ROM supérieure et bit 14 pour la ROM inférieure, le bit étant mis si l'on veut que la ROM soit déconnectée.

Notes : Tous les registres sont conservés, sauf le groupe 1 bis (voir à ce sujet le Chapitre 2).

Instruction : Jump.

Adresses d'exécution : B982H (464), B98AH (664 et 6128).

000BH

Étiquette : **0BH**.

Description : Comme le RST 8, mais l'adresse se trouve dans HL.

Instruction : Jump.

Adresses d'exécution : B97CH (464), B984H (664 et 6128).

000EH

Étiquette : **0EH**.

Description : Saute à l'adresse contenue dans BC, réalisant l'instruction imaginaire : JP (BC).

Notes : Pour l'intérêt, voir aussi 001EH.

Adresse d'exécution : 0EH.

RST 10

Étiquette : aucune (instruction).

Description : Exécution d'une routine d'une ROM secondaire, dont l'adresse est codée dans les deux octets qui suivent le RST 10 dans le programme (voir RST 8).

Notes : Tous les registres sont conservés, sauf IY et le groupe 1 bis.

Instruction : Jump.

Adresses d'exécution : BA16H (464), BA1DH (664 et 6128).

0013H

Étiquette : **13H**.

Description : Comme RST 10, mais les deux octets sont dans HL.

Notes : Voir RST 10.

Instruction : Jump.

Adresses d'exécution : BA10H (464), BA17H (664 et 6128).

0016H

Étiquette : **16H**

Description : Comme 0EH, mais avec DE.

Notes : Voir 0EH.

Adresse d'exécution : 16H.

RST 18

Étiquette : aucune (instruction).

Description : Permet d'appeler une routine en activant une ROM secondaire : les deux octets qui suivent le restart dans le programme donnent l'adresse d'une table où se trouve l'adresse de la routine à appeler, suivie du numéro de la ROM secondaire à activer (0 pour le BASIC, 7 pour le DOS), soit trois octets en tout.

Notes : Voir RST 10. Est utilisé dans les 664 et 6128 seulement, pour appeler le DOS (voyez les vecteurs correspondants, de BC77H à BC9BH).

Instruction : Jump.

Adresses d'exécution : B9BFH (464), B9C7H (664 et 6128).

001BH

Étiquette : **1BH**.

Description : Exécution d'une routine d'adresse contenue dans HL n'importe où dans la mémoire, C contenant l'état des ROM : bit 2 pour la ROM inférieure, bit 3 pour la supérieure, le bit mis indiquant que la ROM est déconnectée.

Notes : Voir RST 10.

Instruction : Jump.

Adresses d'exécution : B9B1H (464), B9B9H (664 et 6128).

001EH

Étiquette : **1EH**.

Description : Un seul octet : JP (HL).

Notes : Permet de réaliser des instructions supplémentaires. Par exemple, *CALL Z,(HL)*, qui n'existe pas, est réalisé par *CALL Z,1EH*.

Adresse d'exécution : 1EH.

0020H

Étiquette : aucune (instruction).

Description : Charge A avec (HL) en RAM, et ce quel que soit l'état de sélection des ROM.

Notes : Voir RST 8.

Instruction : Jump.

Adresses d'exécution : BACBH (646), BAC6H (664 et 6128).

0023H

Étiquette : **23H**.

Description : Comme RST 18, mais les deux octets sont contenus dans HL.

Notes : Voir RST 18.

Instruction : Jump.

Adresses d'exécution : B9B9H (464), B9C1H (664 et 6128).

RST 28

Étiquette : aucune (instruction).

Description : Saut à une adresse de la ROM inférieure, dont l'adresse suit le RST 28.

Notes : Voir RST 8. En sortie, la ROM inférieure est déconnectée.

Instruction : Jump.

Adresses d'exécution : BA2EH (464), BA35H (664 et 6128).

RST 30

Étiquette : aucune (instruction).

Description : Restart libre pour l'utilisateur (8 octets libres).

Notes : Initialement, contient une routine qui déconnecte la ROM inférieure en conservant son état précédent dans (002BH), sauf sur le 664, où l'on trouve un *RST 0*.

Adresse d'exécution : 30H.

RST 38

Étiquette : aucune (instruction).

Description : Point d'entrée des interruptions des périphériques.

Instruction : Jump.

Adresses d'exécution : B939H (464), B941H (664 et 6128).

■ LES VECTEURS DU NOYAU

Le noyau (*kernel*) du système d'exploitation est la base même du système. Il peut être appelé indirectement par les restarts ou par les vecteurs suivants, d'adresses B900H et suite. Ces vecteurs appellent comme les restarts des routines situées de B921H à BAE3H, qui sont les copies d'une partie du noyau. Il est très imprudent de tenter de les modifier, car elles gèrent les ROMS, les RAMS, et sont utilisées par le noyau lui-même et donc par tout le système, sans parler du BASIC.

B900H

Étiquette : **ROMS**.

Description : Connecte la ROM supérieure.

Notes : En sortie, A contient l'indicateur précédent de l'état des ROM : bit 2 pour l'inférieure, bit 3 pour la supérieure, le bit étant mis quand la ROM est déconnectée.

Instruction : Jump.

Adresses d'exécution : BA5EH (464), BA5FH (664 et 6128).

B903H

Étiquette : **RAMS**.

Description : Déconnecte la ROM supérieure.

Notes : Voir B900H.

Instruction : Jump.

Adresses d'exécution : BA68H (464), BA66H (664 et 6128).

B906H

Étiquette : **ROMI**.

Description : Connecte la ROM inférieure.

Notes : Voir B900H.

Instruction : Jump.

Adresses d'exécution : BA4AH (464), BA51H (664 et 6128).

B909H

Étiquette : **RAMI**.

Description : Déconnecte la ROM inférieure.

Notes : Voir B900H.

Instruction : Jump.

Adresses d'exécution : BA54H (464), BA58H (664 et 6128).

B90CH

Étiquette : **ROMANT**.

Description : Restaure l'état antérieur des ROMS. En entrée, A contient l'état à restaurer (donné en sortie dans les routines précédentes, voir B900H).

Notes : Voir B900H.

Instruction : Jump.

Adresses d'exécution : BA72H (464), BA70H (664 et 6128).

B90FH

Étiquette : **ROMSUP**.

Description : Connecte la ROM supérieure dont le numéro se trouve dans C (7 pour le DOS).

Notes : En sortie, A et B contiennent l'état des ROMS (voir B900H), C le numéro de la ROM supérieure précédemment connectée.

Instruction : Jump.

Adresses d'exécution : BA7EH (464) BA79H (664 et 6128).

B912H

Étiquette : **ROM?**.

Description : Cherche quelle ROM supérieure est connectée. Le numéro en est placé dans A.

Instruction : Jump.

Adresses d'exécution : BAA2H (464), BA9DH (664 et 6128).

B915H

Étiquette : **ROMCV?**.

Description : Donne dans A la classe (80H = principale, 1 = secondaire), et dans HL le numéro de version et le numéro de marque d'une ROM supérieure dont le numéro a été donné dans C.

Notes : Respectivement pour le BASIC et le DOS : version = 0 et 5 ; marque = 1 et 0.

Instruction : Jump.

Adresses d'exécution : BA83H (464), BA7EH (664 et 6128).

B918H

Étiquette : **ROMSANT**.

Description : Opération inverse de ROMSUP (B90FH). Les registres B et C donnés en sortie de ROMSUP doivent être retransmis en entrée.

Notes : BC est modifié.

Instruction : Jump.

Adresses d'exécution : BA8CH (464), BA87H (664 et 6128).

B91BH

Étiquette : **LDIRRAM**.

Description : Exécute un LDIR dans les RAMS, même si les ROMS ont été connectées. En sortie, les registres sont comme après un LDIR classique et les ROMS sont dans leurs états antérieurs.

Instruction : Jump.

Adresses d'exécution : BAA6H (464), BAA1H (664 et 6128).

B91EH

Étiquette : **LDDRRAM**.

Description : Comme ci-dessus, mais avec un LDDR.

Instruction : Jump.

Adresses d'exécution : BAACH (464), BAA7H (664 et 6128).

B921H

Étiquette : **TESTEV**.

Description : Utilisé pour le traitement des événements.

Adresse d'exécution : B921H.

■ LES VECTEURS DU CLAVIER

Ici commence la liste des vecteurs du système. Elle comprend 190 vecteurs d'adresses constantes sur les trois CPC, de BB00H à BD37H, répartis en douze groupes. Le premier d'entre eux est constitué par les vecteurs du clavier.

Notez que tous les vecteurs du clavier sont appelés par des RST 8. Il va donc sans dire que le groupe 1 bis est modifié. Cette remarque est vraie pour la majorité des vecteurs système.

BB00H

Étiquette : **CINIT**.

Description : Initialisation du clavier. Outre le RESET (voir la suivante), les états et les valeurs des touches (y compris (CAPS LOCK)), les délais d'attente et de répétition sont remis à leurs valeurs par défaut.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 19E0H (464), 1B5CH (664 et 6128).

BB03H

Étiquette : **CRESET**.

Description : RESET du clavier. Le BREAK reprend sa valeur par défaut, les touches de fonction sont toutes réinitialisées et l'indirection BDEEH est restituée.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1A1EH (464), 1B98H (664 et 6128).

BB06H

Étiquette : **CWAIT**.

Description : Attend qu'un caractère soit entré au clavier, ou lit le dernier caractère du buffer de clavier (qui peut en contenir jusqu'à 20) s'il n'est pas vide. En sortie, la retenue est mise et A contient le caractère.

Notes : Le caractère n'est pas imprimé.

Instruction : RST 8.

Adresses d'exécution : 1A3CH (464), 1BBFH (664 et 6128).

BB09H

Étiquette : **CSCRUT**.

Description : Lit un éventuel caractère au clavier ou dans le buffer du clavier. En sortie, $c = 1$ si un caractère a été lu (il se trouve alors dans A), sinon $c = 0$ et A est modifié.

Notes : Cette routine est celle appelée répétitivement par la précédente.

Instruction : RST 8.

Adresses d'exécution : 1A42H (464), 1BC5H (664 et 6128).

BBOCH

Étiquette : **CARINBUF**.

Description : Place le caractère entré dans A dans le buffer de clavier.

Notes : Voir les deux routines précédentes.

Instruction : RST 8.

Adresses d'exécution : 1A77H (464), 1BFAH (664 et 6128).

BB0FH

Étiquette : **STRCOD**.

Description : Associe un code numérique (communiqué dans B) à une chaîne alphanumérique dont l'adresse est dans HL et la longueur dans C.

Notes : La retenue est mise si tout va bien. Dans tous les cas le groupe 1 est modifié. Cette routine ne doit être exécutée qu'après l'allocation d'un tampon par STRTAMP (BB15H).

Instruction : RST 8.

Adresses d'exécution : 1ABDH (464), 1C46H (664 et 6128).

BB12H

Étiquette : **CARINSTR**.

Description : Donne dans A un des caractères d'une chaîne d'expansion positionnée par le vecteur précédent. En entrée, A contient le code de la chaîne et L le numéro du caractère à lire (à partir de zéro).

Notes : La retenue est mise si tout va bien. DE est modifié.

Instruction : RST 8.

Adresses d'exécution : 1B2EH (464), 1CB3H (664 et 6128).

BB15H

Étiquette : **STRTAMP**.

Description : Donne un tampon (place mémoire) au système pour y placer les chaînes d'expansion (voir BB0FH). En entrée, DE contient l'adresse du tampon et HL sa longueur.

Notes : La retenue est mise si tout va bien. Dans tous les cas, le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1A7BH (464), 1C04H (664 et 6128).

BB18H

Étiquette : **CWAIT2**.

Description : Exactement comme CWAIT (BB06H), mais ne tient compte ni du buffer de clavier ni des chaînes d'expansion.

Notes : Le buffer de clavier n'est pas affecté par cette routine.

Instruction : RST 8.

Adresses d'exécution : 1B56H (464), 1CDBH (664 et 6128).

BB1BH

Étiquette : **CSCRUT2**.

Description : Comme CSCRUT (BB09H), avec la même réserve que pour la précédente.

Notes : Voir la précédente.

Instruction : RST 8.

Adresses d'exécution : 1B5CH (464), 1CE1H (664 et 6128).

BB1EH

Étiquette : **INKEY**.

Description : Teste si la touche dont le numéro est contenu dans A est pressée. Si ce n'est pas le cas, A vaut zéro en sortie et le flag z est mis, sinon A est non nul mais modifié. Dans les deux cas, C contient l'état des touches (SHIFT) et (CTRL), par ses bits 7 (pour (CTRL)) et 5 (pour (SHIFT)), ces bits étant mis lorsque les touches correspondantes sont enfoncées.

Notes : De toute façon, HL est modifié.

Instruction : RST 8.

Adresses d'exécution : 1CBDH (464), 1E45H (664 et 6128).

BB21H

Étiquette : **CAPS?**.

Description : Donne l'état de la touche (CAPS LOCK) dans H et de (CTRL) (CAPS LOCK) dans H (sur 664 et 6128, sinon de la tou-

che (SHIFT)) sous forme de flags : FFH si elles sont enfoncées, 0 sinon.

Notes : Cet état peut être modifié sur 664 et 6128 par CAPS (BD3AH) et sur 464 par LD (B4E7H),HL.

Instruction : RST 8.

Adresses d'exécution : 1BB3H (464), 1D38H (664 et 6128).

BB24H

Étiquette : JOY.

Description : Réalise la fonction BASIC JOY. En sortie, A et H contiennent JOY(0) et L contient JOY(1).

Notes : F est modifié.

Instruction : RST 8.

Adresses d'exécution : 1C5CH (464), 1DE5H (664 et 6128).

BB27H

Étiquette : KEY.

Description : Associe un code fourni dans B à une touche dont le numéro est fourni dans A, dont la pression sans (SHIFT) et sans (CTRL) donnera la chaîne correspondant au code, réalisant ainsi une partie de la fonction BASIC KEY DEF.

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 1D52H (464), 1ED8H (664 et 6128).

BB2AH

Étiquette : KEY?.

Description : Donne dans A le code de la chaîne qui sera produite par la pression sans (CTRL) ni (SHIFT) de la touche dont le numéro a été communiqué en entrée dans A.

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 1D3EH (464), 1EC4H (664 et 6128).

BB2DH

Étiquette : SKEY.

Description : Comme KEY (BB27H), mais pour la pression de la touche avec (SHIFT) et sans (CTRL).

Instruction : RST 8.

Adresses d'exécution : 1D57H (464), 1EDDH (664 et 6128).

BB30H

Étiquette : **SKEY?**.

Description : Comme KEY? (BB2AH), mais pour la pression de la touche avec (SHIFT) et sans (CTRL).

Instruction : RST 8.

Adresses d'exécution : 1D43H (464), 1EC9H (664 et 6128).

BB33H

Étiquette : **CKEY**.

Description : Comme KEY (BB27H), mais pour la pression de la touche avec (CTRL).

Instruction : RST 8.

Adresses d'exécution : 1D5CH (464), 1EE2H (664 et 6128).

BB36H

Étiquette : **CKEY?**.

Description : Comme KEY? (BB2AH), mais pour la pression de la touche avec (CTRL).

Instruction : RST 8.

Adresses d'exécution : 1D48H (464), 1ECEH (664 et 6128).

BB39H

Étiquette : **REPEAT**.

Description : Donne à la touche dont le numéro est fourni dans A la possibilité de se répéter (si B vaut FFH en entrée), ou au contraire l'impossibilité de se répéter (si B vaut 0).

Notes : Le groupe 1 est modifié, sauf DE.

Instruction : RST 8.

Adresses d'exécution : 1CABH (464), 1E34H (664 et 6128).

BB3CH

Étiquette : **REPEAT?**.

Description : Examine si la touche dont le numéro est fourni dans A peut ou non se répéter. Si oui, $z=0$, sinon $z=1$.

Notes : AF et HL sont modifiés. La retenue est mise.

Instruction : RST 8.

Adresses d'exécution : 1CA6H (464), 1E2FH (664 et 6128).

BB3FH

Étiquette : **SPEEDK**.

Description : Réalise la fonction BASIC SPEED KEY x,y. En entrée, x est placé dans H, et y dans L.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 1C6DH (464), 1DF6H (664 et 6128).

BB42H

Étiquette : **SPEEDK?**.

Description : Lit dans H et L les deux paramètres de la routine précédente.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 1C69H (464), 1DF2H (664 et 6128).

BB45H

Étiquette : **POSBRK**.

Description : Associe la routine dont l'adresse est contenue dans DE et dont l'octet de sélection des ROMS (voir RST 18) se trouve dans C, au BREAK.

Instruction : RST 8.

Adresses d'exécution : 1C71H (464), 1DFAH (664 et 6128).

BB48H

Étiquette : **NOBRK**.

Description : Désarme le BREAK.

Notes : Inverse de la précédente. AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 1C82H (464), 1E0BH (664 et 6128).

BB4BH

Étiquette : **BREAK**.

Description : Provoque un BREAK.

Notes : Même si le BREAK n'est pas actif, AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 1C90H (464), 1E19H (664 et 6128).

■ LES VECTEURS DE TEXTE

BB4EH

Étiquette : **TINIT**.

Description : Initialisation du texte. Outre un RESET (voir la suivante), l'encre du fond est remise à 0, celle d'écriture à 1 ; la fenêtre couvre tout l'écran ; le mode opaque est restitué ; les caractères graphiques interdits, ceux de texte autorisés ; tous les caractères reprennent leurs valeurs par défaut et les fenêtres sont toutes réinitialisées sur la fenêtre courante.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1078H (464), 1070H (664), 1074H 6128).

BB51H

Étiquette : **TRESET**.

Description : RESET du texte. Les cinq indirections de texte (BDCDH à BDD9H) sont réinitialisées.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1088H (464), 1080H (664), 1084H (6128).

BB54H

Étiquette : **CANWR**.

Description : Autorise l'affichage des caractères.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 1451H (464), 1455H (664), 1459H (6128).

BB57H

Étiquette : **NOWR**.

Description : Interdit l'affichage des caractères.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 144BH (464), 144EH (664), 1452H (6128).

BB5AH

Étiquette : **PRINT**.

Description : Affichage d'un caractère dont le code est contenu dans A ; ou, si ce code est inférieur à 1FH, exécution de la commande correspondante.

Notes : Pour les codes de commandes, voir COMTAB (BBB1H).

Pour les caractères usuels, cette routine est meilleure que la suivante, car elle conserve les registres.

Instruction : RST 8.

Adresses d'exécution : 1400H (464), 13FAH (664), 13FEH (6128).

BB5DH

Étiquette : **PRINT2**.

Description : Comme la précédente, mais si le code est inférieur à 1FH, le caractère graphique correspondant est imprimé.

Notes : Le groupe 1 est modifié, d'où l'avantage de PRINT pour les caractères usuels.

Instruction : RST 8.

Adresses d'exécution : 1334H (464), 1331H (664), 1335H (6128).

BB60H

Étiquette : **READ.**

Description : Lit sur l'écran un caractère à la position courante du curseur. Si un caractère est effectivement reconnu, il est placé dans A et la retenue est mise. Sinon A et la retenue valent zéro.

Instruction : RST 8.

Adresses d'exécution : 13ABH (464), 13A8H (664), 13ACH (6128).

BB63H

Étiquette : **CARGR.**

Description : Interdit si A vaut zéro, ou autorise, dans le cas contraire, l'écriture des caractères graphiques correspondant aux codes de commandes.

Instruction : RST 8.

Adresses d'exécution : 13A7H (464), 13A4H (664), 13A8H (6128).

BB66H

Étiquette : **WIND0.**

Description : Exécute l'instruction BASIC WINDOW #0,u,v,w,x. Les numéros des lignes limites sont placés dans L et E, ceux des colonnes dans H et D. L'ordre est sans importance. Les paramètres sont ajustés pour tenir à l'écran. Le curseur est placé en haut et à gauche de la fenêtre si son affichage est autorisé (voir de BB7BH à BB84H).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 120CH (464), 1204H (664), 1208H (6128).

BB69H

Étiquette : **WIND0?.**

Description : Lit les paramètres de la routine précédente. HL contient les coordonnées du coin supérieur gauche de la fenêtre, et DE du coin inférieur droit. La retenue est nulle si la fenêtre couvre tout l'écran, et A vaut FFH, sinon la retenue est mise et A est modifié.

Instruction : RST 8.

Adresses d'exécution : 1256H (464), 124EH (664), 1252H (6128).

BB6CH

Étiquette : **CLS0**.

Description : Efface la fenêtre courante, exécutant ainsi la fonction BASIC CLS#0.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1540H (464), 154BH (664), 154FH (6128).

BB6FH

Étiquette : **HPOS**.

Description : Déplace le curseur dans la colonne dont le numéro est contenu dans A.

Notes : AF et HL sont modifiés. La position du curseur est considérée par rapport à la fenêtre courante.

Instruction : RST 8.

Adresses d'exécution : 115EH (464), 1156H (664), 115AH (6128).

BB72H

Étiquette : **VPOS**.

Description : Comme la précédente, mais sur la ligne.

Notes : Voir la précédente.

Instruction : RST 8.

Adresses d'exécution : 1169H (464), 1161H (664), 1165H (6128).

BB75H

Étiquette : **LOCATE**.

Description : Exécute l'instruction BASIC LOCATE x,y. En entrée, x se trouve dans L et y dans H.

Notes : Voir HPOS (BB6FH).

Instruction : RST 8.

Adresses d'exécution : 1174H (464), 116CH (664), 1170H (6128).

BB78H

Étiquette : **LOCATE?**.

Description : Lit les paramètres de la précédente dans HL et place dans A le compteur de défilement (ROLL COUNT).

Instruction : RST 8.

Adresses d'exécution : 1180H (464), 1178H (664), 117CH (6128).

BB7BH

Étiquette : **TCRON**.

Description : Autorise l'affichage du curseur pour l'utilisateur.

Notes : AF est modifié. Le curseur est placé à l'écran si l'affichage pour le système est autorisé (voir SYSCRON et SYSCROF ci-après en BB81H et BB84H).

Instruction : RST 8.

Adresses d'exécution : 1289H (464), 1282H (664), 1286H (6128).

BB7EH

Étiquette : **TCROF**.

Description : Interdit l'affichage du curseur pour l'utilisateur.

Notes : AF est modifié. Le curseur est retiré de l'écran.

Instruction : RST 8.

Adresses d'exécution : 129AH (464), 1293H (664), 1297H (6128).

BB81H

Étiquette : **SYSCRON**.

Description : Autorise l'affichage du curseur par le système.

Notes : Le curseur est placé à l'écran si son affichage pour l'utilisateur est permis (voir les deux précédentes).

Instruction : RST 8.

Adresses d'exécution : 1279H (464), 1272H (664), 1276H (6128).

BB84H

Étiquette : **SYSCROF**.

Description : Interdit l'affichage du curseur pour le système.

Notes : Le curseur est retiré de l'écran.

Instruction : RST 8.

Adresses d'exécution : 1281H (464), 127AH (664), 127EH (6128).

BB87H

Étiquette : **INWIND?**.

Description : Examine si la position de curseur fournie dans HL (comme pour LOCATE en BB75H) se trouve dans la fenêtre courante (la retenue est alors mise et B modifié). Si ce n'est pas le cas, HL contient l'endroit où le curseur serait placé et, si la fenêtre doit défiler dans le sens normal, B vaut FFH ; dans le sens contraire, B vaut 0.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 11CEH (464), 11C6H (664), 11CAH (6128).

BB8AH

Étiquette : **CRVIS**.

Description : Rend le curseur visible.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 1268H (464), 1261H (664), 1265H (6128).

BB8DH

Étiquette : **CRINV**.

Description : Rend le curseur invisible.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 1268H (464), 1261H (664), 1265H (6128).

BB90H

Étiquette : **PEN**.

Description : Positionne sur la valeur d'encre contenue dans A l'encre d'écriture des caractères (PEN).

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 12A9H (464), 12A2H (664), 12A6H (6128).

BB93H

Étiquette : **PEN?**.

Description : Donne dans A l'encre d'écriture des caractères (PEN).

Instruction : RST 8.

Adresses d'exécution : 12BDH (464), 12B6H (664), 12BAH (6128).

BB96H

Étiquette : **PAPER**.

Description : Positionne sur la valeur d'encre contenue dans A l'encre du fond (PAPER).

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 12AEH (464), 12A7H (664), 12ABH (6128).

BB99H

Étiquette : **PAPER?**.

Description : Donne dans A l'encre d'écriture du fond (PAPER).

Instruction : RST 8.

Adresses d'exécution : 12C3H (464), 12BCH (664), 12COH (6128).

BB9CH

Étiquette : **INVERS**.

Description : Inverse l'encre du fond et celle des caractères (PEN et PAPER).

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 12C9H (464), 12C2H (664), 12C6H (6128).

BB9FH

Étiquette : **OPAQ**.

Description : Se place en mode opaque si A vaut 0 en entrée (le fond est alors affiché lors des écritures), en mode transparent sinon.

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 137AH (464), 1377H (664), 137BH (6128).

BBA2H

Étiquette : **OPAQ?**.

Description : Donne en sortie dans A le paramètre de la routine précédente.

Notes : DE, AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 1387H (464), 1384H (664), 1388H (6128).

BBA5H

Étiquette : **MATADR**.

Description : Donne dans HL l'adresse de la matrice du caractère dont le code ASCII figurait en entrée dans A. Si la retenue est mise, la matrice se trouve en RAM (matrice redéfinissable), sinon la matrice est en ROM (caractère standard).

Notes : De toute façon, AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 12D3H (464), 12D0H (664), 12D4H (6128).

BBA8H

Étiquette : **DEFSYMB**.

Description : Redéfinition d'une matrice de caractère. En entrée, A contient le numéro du caractère à modifier et HL l'adresse de la nouvelle matrice de ce caractère. En sortie, la retenue est mise si tout va bien ; sinon, c'est que le caractère n'est pas redéfinissable (voir la routine suivante).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 12F1H (464), 12EEH (664), 12F2H (6128).

BBABH

Étiquette : **POSMTAB**.

Description : Positionnement de la table des caractères redéfinissables. En entrée, E contient le premier caractère redéfinissable souhaité, si D est nul. Dans ce cas, toutes les matrices de caractères

tères de E à 255 sont recopiées en RAM à l'adresse fournie dans HL (attention à la place : jusqu'à 2K !). Si D n'est pas nul en entrée, la table est supprimée, les caractères redeviennent standard. Dans tous les cas, en sortie, AF et HL sont positionnés comme pour la routine suivante avec les valeurs relatives à l'ancienne table ; BC est modifié et DE contient la valeur d'entrée de HL.

Notes : Si une table redéfinie existait déjà, les caractères communs aux deux tables sont recopiés sans modification : il n'y a donc pas de réinitialisation des caractères dont le code est supérieur à E.

Instruction : RST 8.

Adresses d'exécution : 12FDH (464), 12FAH (664), 12FEH (6128).

BBAEH

Étiquette : **GETMTAB**.

Description : Donne en sortie dans HL l'adresse de la table des caractères redéfinissables et dans A le numéro du premier de ces caractères, si la retenue est mise. Si elle ne l'est pas, c'est qu'une telle table n'existe pas (aucun caractère redéfinissable).

Instruction : RST 8.

Adresses d'exécution : 132AH (464), 1327H (664), 132BH (6128).

BBB1H

Étiquette : **COMTAB**.

Description : Donne dans HL l'adresse de la table des codes de commandes. Cette table contient 32 triplets d'octets correspondant aux 32 caractères de commande (0 à 1FH). Chaque triplet est constitué de l'adresse de la routine à exécuter lors de la rencontre du code, précédée du nombre de paramètres à transmettre à cette routine (maximum 15) codé sur un octet. (Attention : sur 664 et 6128, le bit 7 de cet octet doit en plus être mis !) Les paramètres en question doivent avoir été mis dans le buffer (par CARINBUF en BB0CH, par exemple). Leur emplacement est transmis dans HL.

Notes : Les codes de commandes peuvent être exécutés par PRINT (BB5AH). L'adresse de cette table permet de modifier les routines exécutées. Les commandes initiales figurent dans votre manuel de BASIC.

Instruction : RST 8.

Adresses d'exécution : 14CBH (464), 14D0H (664), 14D4H (6128).

BBB4H

Étiquette : **WINDOW.**

Description : La fenêtre dont le numéro (1 à 7) est donné dans A devient la fenêtre courante. En sortie, A contient le numéro de l'ancienne fenêtre et HL est modifié.

Notes : En fait le STREAM n° A est recopié dans le STREAM courant. Un STREAM est l'ensemble des caractéristiques d'une fenêtre (voir liste des paramètres du système pour plus de précision). Il y a 8 STREAMS en mémoire, correspondant aux 8 fenêtres, plus le STREAM courant.

Instruction : RST 8.

Adresses d'exécution : 10E8H (464), 10E0H (664), 10E4H (6128).

BBB7H

Étiquette : **SWSTREAM.**

Description : Échange des STREAMS dont les numéros figurent en entrée dans B et C (voir la routine précédente, *notes*).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1107H (464), 10FFH (664), 1103H (6128).

■ LES VECTEURS GRAPHIQUES

Les vecteurs suivants sont évidemment essentiels, puisqu'ils permettent de réaliser les nombreux graphiques qui sont si agréables sur Amstrad. Pour plus de détails sur leur utilisation, voyez le Chapitre 6, avec les exemples.

BBBAH

Étiquette : **GINIT.**

Description : Initialisation des graphiques. Outre un RESET (voir la suivante), l'encre graphique est mise à 1, celle de fond graphique à 0 ; l'origine est remise à (0,0), la fenêtre graphique est réduite au vide.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 15B0H (464), 15A4H (664), 15A8H (6128).

BBBDH

Étiquette : **GRESET**.

Description : RESET des graphiques.

Notes : Le groupe 1 est modifié. Les trois indirectes graphiques (BDDCH à BDE2H) sont réinitialisées.

Instruction : RST 8.

Adresses d'exécution : 15DFH (464), 15D3H (664), 15D7H (6128).

BBC0H

Étiquette : **MOVE**.

Description : Réalise la fonction BASIC MOVE x,y. En entrée, x est dans DE et y dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 15F4H (464), 15FAH (664), 15FEH (6128).

BBC3H

Étiquette : **MOVER**.

Description : Réalise la fonction BASIC MOVER x,y. En entrée, x est dans DE et y dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 15F1H (464), 15F7H (664), 15FBH (6128).

BBC6H

Étiquette : **POS?**.

Description : Donne les coordonnées (XPOS, YPOS) du curseur graphique dans DE et HL.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 15FCH (464), 1602H (664), 1606H (6128).

BBC9H

Étiquette : **ORIG.**

Description : Réalise la fonction BASIC ORIGIN x,y. En entrée, x est dans DE et y dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1604H (464), 160AH (664), 160EH (6128).

BBCCH

Étiquette : **ORIG?**

Description : Donne les coordonnées de l'origine graphique dans DE et HL.

Instruction : RST 8.

Adresses d'exécution : 1612H (464), 1618H (664), 161CH (6128).

BBCFH

Étiquette : **WGRAV.**

Description : Fixe les bords droit et gauche de la fenêtre graphique. En entrée, les coordonnées de ces bords sont données dans DE et HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1734H (464), 16A1H (664), 16A5H (6128).

BBD2H

Étiquette : **WGRAH.**

Description : Comme la précédente, mais avec les bords supérieur et inférieur.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1779H (464), 16E6H (664), 16EAH (6128).

BBD5H

Étiquette : **WGRAV?.**

Description : Donne dans DE et HL les bords droit et gauche de la fenêtre graphique.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 17A6H (464), 1713H (664), 1717H (6128).

BBD8H

Étiquette : **WGRAH?**.

Description : Comme la précédente, mais avec les bords supérieur et inférieur.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 17BCH (464), 1729H (664), 172DH (6128).

BBDBH

Étiquettes : **CLG**.

Description : Efface la fenêtre graphique, réalisant la fonction BASIC CLG.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 17C5H (464), 1732H (664), 1736H (6128).

BBDEH

Étiquette : **GPEN**.

Description : Fixe la couleur d'écriture graphique à la valeur donnée en entrée dans A, réalisant la fonction BASIC GRAPHICS PEN a.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 17F6H (464), 1763H (664), 1767H (6128).

BBE1H

Étiquette : **GPEN?**.

Description : Donne dans A le paramètre de la routine précédente.

Instruction : RST 8.

Adresses d'exécution : 1804H (464), 1771H (664), 1775H (6128).

BBE4H

Étiquette : **GPAPER**.

Description : Fixe la couleur du fond graphique à la valeur donnée en entrée dans A, réalisant la fonction BASIC GRAPHICS PAPER a.

Notes : A est modifié.

Instruction : RST 8.

Adresses d'exécution : 17FDH (464), 176AH (664), 176EH (6128).

BBE7H

Étiquette : **GPAPER?**.

Description : Donne dans A le paramètre de la routine précédente.

Instruction : RST 8.

Adresses d'exécution : 180AH (464), 1776H (664), 177AH (6128).

BBEAH

Étiquette : **PLOT**.

Description : Réalise la fonction BASIC PLOT x,y. En entrée, x est dans DE et y dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1813H (464), 177FH (664), 1783H (6128).

BBEDH

Étiquette : **PLOTR**.

Description : Réalise la fonction BASIC PLOTR x,y. En entrée, x est dans DE et y dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1810H (464), 177CH (664), 1780H (6128).

BBFOH

Étiquette : **TEST**.

Description : Donne dans A la couleur du point de coordonnées absolues (x,y) (avec en entrée x dans DE et y dans HL), réalisant la fonction BASIC TEST (x,y).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1827H (464), 1793H (664), 1797H (6128).

BBF3H

Étiquette : **TESTR**.

Description : Comme la routine précédente, mais avec des coordonnées relatives, réalisant la fonction BASIC TESTR (x,y).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1824H (464), 1790H (664), 1794H (6128).

BBF6H

Étiquette : **DRAW**.

Description : Réalise la fonction BASIC DRAW x,y. En entrée, x est dans DE et y dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1839H (464), 17A5H (664), 17A9H (6128).

BBF9H

Étiquette : **DRAWR**.

Description : Réalise la fonction BASIC DRAWR x,y. En entrée, x est dans DE et y dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1836H (464), 17A2H (664), 17A6H (6128).

BBFCH

Étiquette : **TAGWR**.

Description : Écrit le caractère dont le code se trouve dans A en entrée, à l'emplacement du curseur graphique, comme après un TAG en BASIC.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1945H (464), 193CH (664), 1940H (6128).

■ LES VECTEURS DE L'ÉCRAN

BBFFH

Étiquette : **SINIT**.

Description : Initialisation du système écran. Outre un RESET (voir la suivante), la base écran est remise en C000H, le mode à 1 avec les couleurs par défaut.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : AA0H (464), ABBH (664), ABFH (6128).

BC02H

Étiquette : **SRESET**.

Description : Reset du système écran. Le mode forçage est restitué et les trois indirections d'écran (BDE5H à BDEBH) sont réinitialisées.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : AB1H (464), ACCH (664), AD0H (6128).

BC05H

Étiquette : **OFFSET**.

Description : Positionne l'OFFSET d'écran à la valeur contenue dans HL en entrée. Permet ainsi un défilement de l'écran.

Notes : AF et HL sont modifiés. L'OFFSET est un nombre pair compris entre 0 et 7FFH qui correspond à l'octet supérieur d'un caractère qui devient le coin supérieur gauche de l'écran. Les caractères qui dépassent d'un côté réapparaissent de l'autre, mais sans oublier le décalage dû aux octets non utilisés. Ainsi, pour décaler l'écran d'une colonne vers la gauche, il faut augmenter l'OFFSET de 2 ; pour le décaler d'une ligne vers le haut, de 80H. Mais si l'OFFSET vaut 0 par exemple, pour décaler l'écran d'une ligne vers le bas il faut un OFFSET de $7B0H = 800H - 80H + 30H$, ce 30H étant dû aux 30H octets inutilisés entre C7D0H et C7FFH. → 50H

Instruction : RST 8.

Adresses d'exécution : B3CH (464), B33H (664), B37H (6128).

BC08H

Étiquette : **SSTART**.

Description : Fixe l'adresse de départ en RAM de la mémoire écran, à partir de A, dont les bits 6 et 7 donnent les bits les plus significatifs de cette adresse, d'où quatre possibilités : 0, 4000H, 8000H, C000H.

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : B45H (464), B38H (664), B3CH (6128).

BC0BH

Étiquette : **SSTART?**.

Description : Donne dans A le premier octet de l'adresse de la mémoire écran (voir la routine précédente), et dans HL l'OFFSET (voir OFFSET en BC05H).

Instruction : RST 8.

Adresses d'exécution : B50H(464), B52H (664), B56H (6128).

BC0EH

Étiquette : **MODE**.

Description : Fixe l'écran dans le mode contenu dans A en entrée (0, 1 ou 2).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : ACAH (464), AE5H (664), AE9H (6128).

BC11H

Étiquette : **MODE?**.

Description : Donne dans A le mode écran actuel et positionne en conséquence la retenue et le zéro : Mode 2 : $c=0, z=0$; Mode 1 : $c=0, z=1$; Mode 0 : $c=1, z=0$.

Instruction : RST 8.

Adresses d'exécution : AECH (464), B08H (664), B0CH (6128).

BC14H

Étiquette : **CLS.**

Description : Efface l'écran tout entier.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : AF7H (464), B13H (664), B17H (6128).

BC17H

Étiquette : **SLIMITS.**

Description : Donne les limites de l'écran : dans B la dernière colonne, dans C la dernière ligne, en fonction du mode.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : B57H (464), B59H (664), B5DH (6128).

BC1AH

Étiquette : **CARADR.**

Description : Donne dans HL l'adresse dans la mémoire écran d'un caractère situé à la position donnée dans HL en entrée (H colonne, L ligne). De plus, en sortie, B contient en fonction du mode actuel le nombre d'octets occupés par un caractère dans la mémoire écran.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : B64H (464), B66H (664), B6AH (6128).

BC1DH

Étiquette : **PTADR.**

Description : Donne dans HL l'adresse dans la mémoire écran de l'octet où se trouve un point dont les coordonnées ont été fournies en entrée dans DE et HL. En sortie C contient le masque correspondant au point, et B le nombre de points restant sur l'octet, en fonction du mode.

Notes : AF et DE sont modifiés.

Instruction : RST 8.

Adresses d'exécution : BA9H (464), BABH (664), BAFH (6128).

BC20H

Étiquette : **RBYTE**.

Description : Donne dans HL l'adresse en mémoire écran de l'octet se trouvant à la droite sur l'écran de l'octet dont l'adresse en mémoire écran a été communiquée en entrée dans HL.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : BF9H (464), C01H (664), C05H (6128).

BC23H

Étiquette : **LBYTE**.

Description : Comme la routine précédente, mais pour l'octet de gauche.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : C05H (464), C0DH (664), C11H (6128).

BC26H

Étiquette : **NBYTE**.

Description : Comme RBYTE (BC20H), mais pour l'octet du dessous.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : C13H (464), C1BH (664), C1FH (6128).

BC29H

Étiquette : **PBYTE**.

Description : Comme RBYTE (BC20H), mais pour l'octet du dessus.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : C2DH (464), C35H (664), C39H (6128).

BC2CH

Étiquette : **INKCOD**.

Description : Fournit dans A le masque correspondant à l'encre dont le numéro a été communiqué en entrée.

Instruction : RST 8.

Adresses d'exécution : C86H (464), C8AH (664), C8EH (6128).

BC2FH

Étiquette : **INKDECOD**.

Description : Fait la conversion contraire de celle de la routine précédente : le masque est en entrée, l'encre en sortie. Seule compte en fait l'encre du point le plus à gauche dans l'octet.

Instruction : RST 8.

Adresses d'exécution : CA0H (464), CA3H (664), CA7H (6128).

BC32H

Étiquette : **INK**.

Description : Exécute l'instruction BASIC INK i, u, v. En entrée, i est dans A, u et v dans B et C.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : CECH (464), CEEH (664), CF2H (6128).

BC35H

Étiquette : **INK?**.

Description : Donne dans B et C les couleurs associées à l'encre dont le numéro est fourni en entrée dans A.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : D14H (464), D16H (664), D1AH (6128).

BC38H

Étiquette : **BORDER**.

Description : Réalise la fonction BASIC BORDER u, v. En entrée, u est dans B et v dans C.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : CF1H (464), CF3H (664), CF7H (6128).

BC3BH

Étiquette : **BORDER?**.

Description : Donne dans B et C les paramètres de la routine précédente.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : D19H (464), D1BH (664), D1FH (6128).

BC3EH

Étiquette : **SPINK**.

Description : Réalise la fonction BASIC SPEED INK u, v. En entrée, u est dans H et v dans L.

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : CE4H (464), CE6H (664), CEAH (6128).

BC41H

Étiquette : **SPDINK?**.

Description : Donne dans H et L les paramètres de la routine précédente.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : CE8H (464), CEAH (664), CEEH (6128).

BC44H

Étiquette : **FILLBOX**.

Description : Remplit un rectangle, considéré comme un ensemble de caractères (comme les fenêtres), avec l'octet donné dans A. En entrée, H et D contiennent les colonnes gauche et droite du rectangle, et E et L les lignes supérieure et inférieure.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : DB3H (464), DB5H (664), DB9H (6128).

BC47H

Étiquette : **FILLBOX2**.

Description : Comme la routine précédente, mais le rectangle est considéré comme un ensemble d'octets dans la mémoire écran.

HL contient l'octet du coin supérieur gauche, D le nombre d'octets par ligne, et E le nombre de lignes.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : DB7H (464), DB9H (664), DBDH (6128).

BC4AH

Étiquette : **CARINVER**.

Description : Inverse deux couleurs (dont les masques sont donnés dans B et C) dans le caractère dont la position est donnée dans HL, (H colonne, L ligne).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : DDFH (464), DE1H (664), DE5H (6128).

BC4DH

Étiquette : **SROLL**.

Description : Décale l'écran d'une ligne vers le haut si B diffère de zéro, vers le bas sinon.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : DFAH (464), DFCH (664), E00H (6128).

BC50H

Étiquette : **WROLL**.

Description : Comme la routine précédente, mais pour un rectangle de l'écran seulement. Ce rectangle est défini par HL et DE comme dans FILLBOX (BC44H). La partie vidée est remplie avec l'octet contenu dans A en entrée.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : E3EH (464), E40H (664), E44H (6128).

BC53H

Étiquette : **CARMAT**.

Description : Convertit la matrice de caractère dont l'adresse est fournie dans HL en une suite de masques situés à l'endroit pointé par DE en entrée.

Notes : Le groupe 1 est modifié. L'adresse de la matrice de caractère peut être obtenue par la routine MATADR (BBA5H).

Instruction : RST 8.

Adresses d'exécution : EF3H (464), EF5H (664), EF9H (6128).

BC56H

Étiquette : **MATCAR**.

Description : Conversion inverse de celle de la routine précédente, mais à partir d'un caractère de l'écran dont la position est fournie dans HL (H colonne, L ligne) et pour l'encre dont le masque est fourni dans A. La matrice sera construite à l'adresse fournie dans DE.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : F49H (464), F26H (664), F2AH (6128).

BC59H

Étiquette : **GMOD**.

Description : Fixe le mode graphique (dernier paramètre de DRAW sur les 664 et 6128) fourni dans A : 0=forçage ; 1=XOR ; 2=AND ; 3=OR.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : C49H (464), C51H (664), C55H (6128).

BC5CH

Étiquette : **POINT**.

Description : Place un point dont le masque est donné dans C et le masque d'encre dans B à l'écran, à l'adresse fournie dans HL.

Notes : AF est modifié. Attention, contrairement à PLOT (BBEAH), cette routine ne tient pas compte du mode graphique défini par la routine précédente.

Instruction : RST 8.

Adresses d'exécution : C6BH (464), C70H (664), C74H (6128).

BC5FH

Étiquette : **HORIZ**.

Description : Trace une ligne horizontale dans l'encre dont le masque est fourni dans A, entre les points d'ordonnée fournie dans HL et d'abscisses fournies dans DE (départ) et BC (arrivée).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : FC4H (464), F8FH (664), F93H (6128).

BC62H

Étiquette : **VERTIC**.

Description : Comme la routine précédente avec une verticale. L'abscisse des extrémités est fournie dans HL, l'ordonnée de départ dans DE, celle d'arrivée dans BC.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 102FH (464), F97H (664), F9BH (6128).

■ LES VECTEURS CASSETTE/DISQUETTE

Les vecteurs suivants concernent le lecteur de cassette. Mais sur les 664, 6128 ou sur le 464 avec drive, si l'on est en mode DISC les vecteurs BC77H à BC9BH marqués d'un astérisque (*) concernent le lecteur de disquette. Dans ce cas, ils se reportent par un RST 18 à l'adresse A88BH, qui repasse le contrôle du DOS aux adresses indiquées dans le champ *adresses d'exécution*. Dans ce cas toujours, le RST 18 modifie IY. *On ne l'a pas précisé chaque fois.*

Pour ce qui est du passage du mode disque au mode cassette, il faut utiliser les instructions du DOS (comme DISC, etc.), dont les adresses peuvent être obtenues par FINDRSX (BCD4H).

BC65H

Étiquette : **KINIT**.

Description : Initialisation du MOS. Tous les fichiers sont abandonnés, les messages sont autorisés, puis l'on passe à la routine suivante avec les valeurs par défaut A=19H et HL=14DH.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 2370H (464), 24BCH (664 et 6128).

BC68H

Étiquette : **KSPEED**.

Description : Fixe la vitesse d'écriture sur cassette : en entrée, HL contient la moitié de la longueur d'un bit 0, et A la compensation à faire.

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 237FH (464), 24CEH (664 et 6128).

BC6BH

Étiquette : **KMESSG**.

Description : Autorise (si A vaut 0 en entrée) ou interdit (dans le cas contraire) l'affichage des messages du MOS.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 238EH (464), 24E1H (664 et 6128).

BC6EH

Étiquette : **MOTORON**.

Description : Met le moteur du magnétophone en marche et donne dans A son état précédent (0 = arrêté ; 10H = en marche). La retenue est mise, sauf si on a appuyé sur (ESC) (mais le moteur n'est pas arrêté).

Instruction : RST 8.

Adresses d'exécution : 2A4BH (464), 2BBBH (664 et 6128).

BC71H

Étiquette : **MOTOROF.**

Description : Arrêt du moteur et comme la routine précédente en sortie.

Instruction : RST 8.

Adresses d'exécution : 2A4FH (464), 2BBFH (664 et 6128).

BC74H

Étiquette : **MOTOR.**

Description : Remplace le moteur dans son état précédent, communiqué en entrée dans A. En sortie, comme les deux précédentes.

Instruction : RST 8.

Adresses d'exécution : 2A51H (464), 2BC1H (664 et 6128).

BC77H *

Étiquette : **OPEN.**

Description : Ouvre un fichier en lecture. En entrée, HL contient l'adresse de son nom et B sa longueur, et DE contient l'adresse du tampon de 2K nécessaire. En sortie, si tout va bien la retenue est mise, A contient le type du fichier (bit 0 mis = protégé ; bits 1 et 2 : 00 = BASIC, 01 = Binaire, 10 = image écran, 11 = ASCII ; bit 4 mis pour les ASCII ; autres nuls), BC sa longueur, DE l'adresse où les données doivent être écrites, et HL l'adresse de l'en-tête, qui est placé au début de chaque bloc ou enregistrement. Si quelque chose ne va pas, la retenue n'est pas mise et le groupe 1 est modifié. Le flag z indique alors le problème : s'il est mis, on a appuyé sur (ESC) ; sinon, c'est que le fichier était déjà ouvert, ou le système occupé.

Notes : Dans tous les cas IX est modifié.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 2392H (464), 24E5H (664 et 6128) ;
DOS : CEAFH.

BC7AH *

Étiquette : **CLOSE.**

Description : Ferme le fichier après la fin de la lecture. La retenue est mise si tout va bien.

Notes : Le groupe 1 est modifié.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 23FCH (464), 2550H (664 et 6128) ;
DOS : D1B6H.

BC7DH *

Étiquette : **CLOSE!**.

Description : Arrête la lecture et ferme le fichier.

Notes : Le groupe 1 est modifié.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 2401H (464), 2557H (664 et 6128) ;
DOS : D1BCH.

BC80H *

Étiquette : **KINCAR.**

Description : Lit un octet dans le fichier et le place dans A. La retenue est mise si tout va bien. Sinon, A est modifié et le flag z mis indique la pression de (ESC), non mis indique qu'on est à la fin du fichier.

Notes : IX est aussi modifié.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 2435H (464), 25A0H (664 et 6128) ;
DOS : CF64H.

BC83H *

Étiquette : **KINFICH.**

Description : Lit un fichier et le place en mémoire à l'adresse fournie dans HL. Si tout va bien, la retenue est mise et HL contient le point d'entrée du fichier, sinon voir la routine précédente.

Notes : Le groupe 1 et IX sont modifiés.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 24ABH (464), 2618H (664 et 6128) ;
DOS : CFF5H.

BC86H *

Étiquette : **KRET.**

Description : Effectue l'inverse de KINCAR (BC80H) en remplaçant le caractère lu dans le tampon.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 249AH (464), 2607H (664 et 6128) ;
DOS : D069H.

BC89H *

Étiquette : **EOF?**.

Description : Regarde si la fin du fichier est atteinte. Sinon, la retenue est mise. Si elle ne l'est pas, on a atteint la fin du fichier si $z=0$, on a appuyé sur (ESC) si $z=1$.

Notes : AF et IX sont modifiés.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 2496H (464), 2603H (664 et 6128) ;
DOS : D065H.

BC8CH *

Étiquette : **OPENOUT**.

Description : Ouvre un fichier en sortie. Les paramètres d'entrée et les états d'erreurs sont les mêmes que pour OPEN (BC77H).
En sortie, si tout va bien (retenue mise) HL contient l'adresse du tampon.

Notes : Dans tous les cas, le groupe 1 et IX sont modifiés.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 23ABH (464), 24FEH (664 et 6128) ;
DOS : CF37H.

BC8FH *

Étiquette : **CLOSOUT**.

Description : Comme CLOSE, mais pour le fichier de sortie.

Notes : Le groupe 1 et IX sont modifiés.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 2415H (464), 257FH (664 et 6128) ;
DOS : D1D8H.

BC92H *

Étiquette : **CLOSOUT!**.

Description : Arrête l'écriture et ferme le fichier.

Notes : Le groupe 1 est modifié.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 242EH (464), 2599H (664 et 6128) ;
DOS : D1C2H.

BC95H *

Étiquette : **KOUTCAR**.

Description : Écrit sur le fichier le caractère fourni dans A. En sortie, la retenue est mise si tout va bien, sinon c'est que l'on a appuyé sur (ESC) (z=1) ou que le fichier n'était pas ouvert.

Notes : AF et IX sont modifiés.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 245BH (464), 25C6H (664 et 6128) ;
DOS : D08FH.

BC98H *

Étiquette : **KOUTFICH**.

Description : Écrit dans le fichier en sortie une partie de la mémoire dont l'adresse est fournie dans HL, la longueur dans DE, le point d'entrée dans BC et le type dans A. Pour les erreurs, comme la routine précédente.

Notes : AF et IX sont modifiés. Cette routine ne peut être combinée avec la précédente.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 24EAH (464), 2653H (664 et 6128) ;
DOS : D0D8H.

BC9BH *

Étiquette : **CAT**.

Description : Donne le catalogue de la cassette ou de la disquette, comme la fonction BASIC CAT. En entrée, DE contient l'adresse du tampon de 2K nécessaire. Pour les erreurs, voir OPENOUT (BC8CH).

Notes : Le groupe 1 et IX sont modifiés.

Instruction : RST 8 (MOS) ; RST 18 en A88BH (DOS).

Adresses d'exécution : MOS : 2528H (464) ; 2692H (664 et 6128) ;
DOS : D513H.

BC9EH

Étiquette : **KWRITE**.

Description : Écrit un bloc de données dont l'adresse est fournie dans HL et la longueur dans DE sur cassette. A contient l'octet de synchronisation nécessaire. En sortie, si tout va bien la retenue est mise, sinon le code de l'erreur produite est placé dans A.

Notes : Le groupe 1 et IX sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 283FH (464), 29AFH (664 et 6128).

BCA1H

Étiquette : **KREAD**.

Description : Comme la précédente, mais en lisant. Les paramètres d'entrée sont les mêmes.

Notes : Le groupe 1 et IX sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 2836H (464), 29A6H (664 et 6128).

BCA4H

Étiquette : **CHECK**.

Description : Compare la mémoire dont l'adresse est donnée dans HL et la longueur dans DE avec le contenu du fichier en lecture. La retenue est mise s'il y a similitude, sinon A contient le code de l'erreur.

Notes : Le groupe 1 et IX sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 2851H (464), 29C1H (664 et 6128).

■ LES VECTEURS SONORES

Les vecteurs suivants gèrent les sons. Dans ce domaine, vous savez que les possibilités des Amstrad sont très étendues, grâce à leur PSG (générateur de sons programmable). Néanmoins, la programmation en langage machine est ici très proche du BASIC ; les vecteurs suivants y sont en effet très adaptés, et les différences avec les fonctions BASIC sonores sont généralement très faibles. C'est la raison pour laquelle la plupart de ces vecteurs portent le nom de fonctions BASIC. Pour les détails, on se reportera donc au manuel d'utilisation. Précisons pour finir que la routine OUTPSG (BD34H) permet de programmer directement le PSG : avis aux amateurs !

BCA7H

Étiquette : **SDRESET**.

Description : RESET des sons. Tous les sons de tous les canaux sont éliminés, les files d'attente vidées. Mais les enveloppes sont préservées.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1E68H (464), 1FE9H (664 et 6128).

BCAAH

Étiquette : **SOUND**.

Description : Exécute la fonction BASIC SOUND. En entrée, HL contient l'adresse de la table des paramètres de cette fonction, dans l'ordre suivant : état du canal (1 octet), numéro d'enveloppe de volume (1 octet), numéro d'enveloppe de ton (1 octet), période (2 octets), période de bruit (1 octet), volume (1 octet), durée (2 octets), soit 9 octets en tout. En sortie, si tout va bien la retenue est mise et HL modifié, sinon HL est conservé (plus de place dans les queues sonores).

Notes : Dans tous les cas, AF, BC, DE et IX sont modifiés. D'autre part, les paramètres doivent se trouver dans la RAM médiane, sinon le programme (qui est en ROM) ne peut les lire.

Instruction : RST 8.

Adresses d'exécution : 1F9FH (464), 2114H (664 et 6128).

BCADH

Étiquette : **SQ**.

Description : Donne (dans A) l'état du canal sonore dont le numéro est fourni dans A, réalisant la fonction BASIC SQ(a).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 206CH (464), 21CEH (664 et 6128).

BCB0H

Étiquette : **ON SQ**.

Description : Arme une interruption dont l'adresse est fournie dans HL pour qu'elle se produise dès que la queue sonore dont le numéro a été fourni dans A contiendra une place, réalisant en quelque sorte la fonction BASIC ON SQ(a) GOSUB...

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 2089H (464), 21EBH' (664 et 6128).

BCB3H

Étiquette : **RELEASE**.

Description : Réalise la fonction BASIC RELEASE ; le paramètre doit être fourni dans A.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 204AH (464), 21ACH (664 et 6128).

BCB6H

Étiquette : **SDHALT**.

Description : Suspension des sons sur tous les canaux. En sortie, si un son était en attente, la retenue est mise et BC modifié.

Notes : Dans tous les cas, AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 1ECBH (464), 2050H (664 et 6128).

BCB9H

Étiquette : **SDRST**.

Description : Réactive les sons suspendus par la routine précédente.

Notes : AF, BC, DE et IX sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 1EE6H (464), 206BH (664 et 6128).

BCBCH

Étiquette : **ENV**.

Description : Définit l'enveloppe de volume dont le numéro est fourni dans A, réalisant la fonction BASIC ENV. En entrée, HL contient l'adresse des paramètres (16 au total). En sortie, si tout va bien la retenue est mise, BC vaut 0, HL a augmenté de 16, A est modifié. Sinon, c'est que le numéro n'était pas valable (non compris entre 1 et 15) et ces registres sont conservés.

Notes : Dans tous les cas, F et DE sont modifiés. Au sujet de l'emplacement des paramètres, voyez la note pour SOUND (BCAAH).

Instruction : RST 8.

Adresses d'exécution : 2338H (464), 2495H (664 et 6128).

BCBFH

Étiquette : **ENT**.

Description : Comme la routine précédente, avec les enveloppes de ton.

Instruction : RST 8.

Adresses d'exécution : 233DH (464), 249AH (664 et 6128).

BCC2H

Étiquette : **ENV?**.

Description : Donne dans HL l'adresse des paramètres de l'enveloppe de volume dont le numéro a été fourni dans A, sauf si ce numéro n'est pas valable (dans ce cas, la retenue est mise et F et HL sont seuls modifiés). Si tout va bien, la retenue est mise, AF est modifié et BC vaut 10H (longueur d'une enveloppe).

Instruction : RST 8.

Adresses d'exécution : 2349H (464), 24A6H (664 et 6128).

BCC5H

Étiquette : **ENT?**.

Description : Comme la routine précédente, avec les enveloppes de ton.

Instruction : RST 8.

Adresses d'exécution : 234EH (464), 24ABH (664 et 6128).

■ AUTRES VECTEURS DU NOYAU

Pour tout ce qui est afférent, dans cette partie, aux événements et interruptions, on se référera avec profit au Chapitre 3.

BCC8H

Étiquette : **RESET**.

Description : Reset du système. En sortie, B contient l'adresse de sélection d'une ROM éventuelle, C l'octet de sélection et DE le point d'entrée de cette ROM.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 5CH (464, 664 et 6128).

BCCBH

Étiquette : **ROMSINIT**.

Description : Initialise toutes les ROM externes. DE et HL contiennent en entrée les adresses des premiers et derniers octets des ROM ; en sortie, ils contiennent les nouvelles valeurs de ces adresses.

Notes : AF et BC sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 329H (464), 326H (664 et 6128).

BCCEH

Étiquette : **ROMINIT**.

Description : Initialise une ROM externe dont l'octet de sélection est fourni dans C. DE et HL comme pour la précédente.

Notes : AF et B sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 332H (464), 330H (664 et 6128).

BCD1H

Étiquette : **INTRSX**.

Description : Initialise une ou plusieurs RSX (extension résidente du système). En entrée, HL contient l'adresse d'une table de 4 octets libre, et BC l'adresse de la table contenant les adresses des noms de commandes (voir le Chapitre 4).

Notes : DE est modifié.

Instruction : RST 8.

Adresses d'exécution : 2A1H (464), 2A0H (664 et 6128).

BCD4H

Étiquette : **FINDRSX**.

Description : Recherche d'une RSX dont l'adresse du nom a été fournie dans HL. En sortie, la retenue est mise si la RSX existe, et dans ce cas C contient l'octet de sélection des ROM et HL l'adresse de la routine.

Notes : AF, BC et DE sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 2B2H (464), 2B1H (664 et 6128).

BCD7H

Étiquette : **CRTEVIN**.

Description : Initialise un événement lié aux interruptions du CRT (contrôleur vidéo). En entrée, HL contient l'adresse du bloc d'événement (voir Chapitre 3), B sa classe, C l'octet de sélection des ROM et DE l'adresse de la routine à exécuter.

Notes : AF, DE et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 163H (464, 664 et 6128).

BCDAH

Étiquette : **CRTEV**.

Description : Repositionne un événement du type précédent (dont l'adresse du bloc est fournie dans HL), annulant les effets de la routine suivante.

Notes : AF, DE et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 16AH (464, 664 et 6128).

BCDDH

Étiquette : **CRTEVDL**.

Description : Suspend un événement du type CRT (dont l'adresse est fournie dans HL). Cette suspension dure jusqu'à l'appel de la routine précédente.

Notes : AF, DE et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 170 H (464, 664 et 6128).

BCE0H

Étiquette : **FEVIN**.

Description : Comme CRTEVIN (BCD7H), mais pour les événements rapides (1/300 de seconde).

Notes : AF, DE et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 176H (464, 664 et 6128).

BCE3H

Étiquette : **FASTEV**.

Description : Comme CRTEV (BCDAH), pour les événements rapides.

Notes : AF, DE et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 17DH (464, 664 et 6128).

BCE6H

Étiquette : **FEVEDEL**.

Description : Comme CRTEVDL (BCDDH), pour les événements rapides.

Notes : AF, DE et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 183H (464, 664 et 6128).

BCE9HH

Étiquette : **EVENT**.

Description : Initialisation ou redémarrage d'un événement normal (1/50 de seconde). En entrée, HL contient l'adresse du bloc, E la valeur initiale du compteur, D la classe de l'événement et BC l'adresse de la routine de traitement.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1B3H (464, 664 et 6128).

BCECHH

Étiquette : **DELEVENT**.

Description : Comme CRTEVDL (BCDDH), avec les événements normaux (voir la précédente). En sortie, la retenue est mise si l'événement n'avait pas été déjà suspendu et DE contient la valeur du compteur.

Notes : AF, DE et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 1C5H (464, 664 et 6128).

BCEF2H

Étiquette : **EVENTIN**.

Description : Comme CRTEVIN, mais pour un événement quelconque.

Instruction : RST 8.

Adresses d'exécution : 1D2H (464, 664 et 6128).

BCF2H

Étiquette : **EVBLOC**.

Description : Détruit un bloc d'événement dont l'adresse est fournie dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 1E2H (464, 664 et 6128).

BCF5H

Étiquette : **EVRESET.**

Description : Vidange des queues d'événement.

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 228H (464), 227H (664 et 6128).

BCF8H

Étiquette : **CLREV.**

Description : Enlève l'événement dont l'adresse de bloc est fournie dans HL de la queue d'événement.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 285H (464), 284H (664 et 6128).

BCFBH

Étiquette : **NEXTEV.**

Description : Passe à l'événement suivant, sauf s'il a une priorité inférieure à la priorité courante ; dans ce dernier cas, la retenue n'est pas mise, et de même s'il n'y a pas d'événement ensuite (dans ce cas le flag z est mis). Par contre, si tout va bien la retenue est mise.

Notes : AF, DE et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 256H (464), 255H (664 et 6128).

BCFEH

Étiquette : **EVENT!.**

Description : Exécution immédiate d'un événement dont l'adresse de bloc est fournie dans HL.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 21AH (464), 219H (664 et 6128).

BD01H

Étiquette : **DONEEV**.

Description : Tient prêt un événement (dont l'adresse de bloc est fournie dans HL) pour qu'il suive un événement dont le code de priorité est fourni dans A.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 277H (464), 276H (664 et 6128).

BD04H

Étiquette : **DIEVENT**.

Description : Suspension de tous les événements normaux.

Notes : HL est modifié.

Instruction : RST 8.

Adresses d'exécution : 295H (464), 294H (664 et 6128).

BD07H

Étiquette : **ENEVENT**.

Description : Autorise de nouveau les événements normaux.

Notes : HL est modifié.

Instruction : RST 8.

Adresses d'exécution : 29BH (464), 29AH (664 et 6128).

BD0AH

Étiquette : **DISEV**.

Description : Suspend l'événement dont l'adresse de bloc est fournie dans HL.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 28EH (464), 28DH (664 et 6128).

BD0DH

Étiquette : **TIME?**.

Description : Donne dans DEHL (sur 32 bits) le temps écoulé au chronomètre, en 1/300 de seconde.

Instruction : RST 8.

Adresses d'exécution : 99H (464, 664 et 6128).

BD10H

Étiquette : **TIME**.

Description : Positionne le chronomètre à la valeur fournie dans DEHL.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : A3H (464, 664 et 6128).

■ LES VECTEURS DES PÉRIPHÉRIQUES

Les vecteurs suivants gèrent directement les périphériques et sont de ce fait d'un intérêt très relatif. Retenons toutefois les vecteurs de gestion de l'imprimante, dont l'utilité est évidente.

BD13H

Étiquette : **BOOT**.

Description : Charge un programme et l'exécute directement. En entrée, HL doit contenir l'adresse de la routine de chargement du programme. Cette routine est exécutée, puis le programme est lancé par START (routine suivante).

Notes : Le SP est remplacé à C000H ; diverses réinitialisations sont effectuées avant de lancer le programme.

Instruction : RST 8.

Adresses d'exécution : 5DCH (464), 5D7H (664), 5EDH (6128).

BD16H

Étiquette : **START**.

Description : Exécute un programme dont l'adresse est donnée dans HL, et l'octet de sélection des ROM dans C.

Notes : Comme la précédente.

Instruction : RST 8.

Adresses d'exécution : 60BH (464), 606H (664), 61CH (6128).

BD19H

Étiquette : **FRAME**.

Description : Attend le début du balayage vertical du CRT (contrôleur vidéo), réalisant ainsi la fonction BASIC FRAME.

Notes : Pour les possesseurs de 464, voir page 8.5 du manuel, en bas ; pour les autres, voir explication de FRAME.

Instruction : RST 8.

Adresses d'exécution : 7BAH (464), 7A4H (664), 7B4H (6128).

BD1CH

Étiquette : **MMODE**.

Description : Positionne le mode écran sur la valeur fournie dans A. Diffère de MODE (BC0EH) par l'absence totale de réinitialisation. L'écran n'est même pas effacé.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 776H (464), 766H (664), 776H (6128).

BD1FH

Étiquette : **MOFFSET**.

Description : Fixe à partir de A l'adresse de départ de l'écran et l'offset à la valeur contenue dans HL comme décrit pour OFFSET et SSTART (BC05H et BC08H). La différence vient de ce que ces modifications ne sont pas enregistrées dans les indicateurs de RAM. Le système d'exploitation agit donc comme s'il n'y avait pas eu changement, d'où des effets parfois curieux.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 7C6H (464), 7B0H (664), 7C0H (6128).

BD22H

Étiquette : **NOIR**.

Description : Positionne toutes les couleurs, y compris celles du bord à la valeur de (DE), donnant l'impression que l'écran s'est effacé.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 786H (464), 776H (664), 786H (6128).

BD25H

Étiquette : **COLORS**.

Description : Positionne les couleurs aux valeurs contenues dans les octets pointés par DE. La couleur du bord vient en premier.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 799H (464), 77CH (664), 78CH (6128).

BD28H

Étiquette : **PRRESET**.

Description : Reset de la sortie parallèle (vers l'imprimante).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 7E6H (464), 7D0H (664), 7E0H (6128).

BD2BH

Étiquette : **PRCAR**.

Description : Fait imprimer le caractère fourni dans A, sauf si l'imprimante est hors tension ou trop longuement occupée (dans ce cas, la retenue n'est pas mise en sortie).

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 7F2H (464), 80BH (664), 81BH (6128).

BD2EH

Étiquette : **BUSY?**.

Description : Vérifie si l'imprimante est opérationnelle. Dans ce cas, la retenue est mise en sortie, sinon c'est que l'imprimante est occupée ou hors tension.

Notes : La retenue est seule modifiée.

Instruction : RST 8.

Adresses d'exécution : 81BH (464), 848H (664), 858H (6128).

BD31H

Étiquette : **SEND CAR.**

Description : Envoie un caractère à l'imprimante sans attendre qu'elle devienne opérationnelle, si elle ne l'était pas. La retenue est mise en sortie.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 807H (464), 834H (664), 844H (6128).

BD34H

Étiquette : **OUTPSG.**

Description : Envoie l'octet fourni dans C dans le registre du PSG (générateur de sons programmable) dont le numéro est fourni dans A, évitant ainsi de complexes manipulations d'entrées/sorties.

Notes : AF et BC sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 826H (464), 853H (664), 863H (6128).

■ L'INITIALISATION DES VECTEURS ET L'ÉDITEUR DE TEXTE

Les deux vecteurs suivants sont particuliers et n'ont pu trouver place ni dans le paragraphe précédent, ni dans le suivant. EDIT possède des adresses différentes suivant les Amstrad, car il est destiné au BASIC (mais vous pouvez tout à fait l'utiliser). Voyez à ce sujet le début du paragraphe "Vecteurs de l'arithmétique à virgule flottante".

BD37H

Étiquette : **JPREST.**

Description : Réinitialise tous les vecteurs (mais pas les indirections).
Sur 664 et 6128, réalise au passage la fonction BASIC TAPE.

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 888H (464), 8BBH (664), 8BDH (6128).

BD3AH (464), **BD5BH** (664), **BD5EH** (6128)

Étiquette : **EDIT**.

Description : Édite la phrase dont l'adresse est fournie dans HL.

Cette phrase doit contenir moins de 255 caractères et se finir par un zéro. Il s'agit de l'éditeur puissant utilisé par le BASIC pour ses lignes et ses commandes, avec usage des touches (CLR), (DEL), (COPY), ...

Notes : Tous les registres sont conservés, sauf A qui contient le caractère de sortie (ODH = (RETURN) ou FCH = (ESC)) et F, car la dernière instruction est : *CP FCH*.

Instruction : RST 8.

Adresses d'exécution : 2A98H (464), 2C02H (664 et 6128).

■ LES VECTEURS SPÉCIFIQUES DES CPC 664 ET 6128

Les vecteurs de ce paragraphe n'existent pas sur CPC 464. La plupart correspondent aux nouvelles fonctions BASIC du 664. Le dernier d'entre eux ne se trouve que sur le 6128 : il permet l'accès aux blocs de RAM supplémentaires de cet appareil, ce qui en fait évidemment un vecteur précieux.

BD3AH

Étiquette : **CAPS**.

Description : Fixe par HL l'état de (CAPS LOCK) et de (CTRL) (CAPS LOCK). Ces états sont ceux indiqués pour la sortie de CAPS? (BB21H).

Instruction : RST 8.

Adresses d'exécution : 1D3CH (664 et 6128).

BD3DH

Étiquette : **CCLEAR**.

Description : Vide le buffer clavier, réalisant ainsi la fonction BASIC CLEAR INPUT.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 1BFEH (664 et 6128).

BD40H

Étiquette : **CRFG?**.

Description : Donne dans A la valeur des flags VDU/curseur. Les flags curseur sont donnés par le bit 0 (texte : voir TCRON (BB7BH)) et le bit 1 (système : voir SYSCRON (BB81H)). Le flag VDU est le bit 7. Pour ces trois flags, 0 signifie autorisé et 1 interdit.

Instruction : RST 8.

Adresses d'exécution : 145CH (664), 1460H (6128).

BD43H

Étiquette : **MASK0**.

Description : Réinitialisation des paramètres de tracé de ligne : le mode forçage est positionné, les paramètres de masque sont remplacés à leurs valeurs par défaut (voir les trois routines suivantes).

Notes : AF et HL sont modifiés.

Instruction : RST 8.

Adresses d'exécution : 15E8H (664), 15ECH (6128).

BD46H

Étiquette : **MASK3**.

Description : Fixe à la valeur fournie dans A le troisième paramètre de masque pour le tracé de ligne. Ce paramètre, comme le deuxième (voir la routine suivante), règle le tracé du premier point, mais sans provoquer de décalage dans la ligne à tracer : si A diffère de zéro, ce premier point est forcé à l'allumage.

Notes : La valeur par défaut de ce paramètre est 0.

Instruction : RST 8.

Adresses d'exécution : 19D1H (664), 19D5H (6128).

BD49H

Étiquette : **MASK2**.

Description : Fixe le deuxième paramètre de la fonction BASIC MASK à la valeur fournie dans A. On se référera au manuel pour plus de détails.

Notes : La valeur par défaut de ce paramètre est FFH.

Instruction : RST 8.

Adresses d'exécution : 17ACH (664), 17B0H (6128).

BD4CH

Étiquette : **MASK1**.

Description : Comme la précédente, mais avec le premier paramètre.

Notes : La valeur par défaut de ce paramètre est FFH.

Instruction : RST 8.

Adresses d'exécution : 17A8H (664), 17ACH (6128).

BD4FH

Étiquette : **PHYSPOS**.

Description : Donne la position physique réelle d'un point dont les coordonnées ont été fournies en entrée dans DE et HL, en fonction du mode courant. Cette position physique est constituée du numéro de ligne (0 à 199) donnée dans DE et du numéro de colonne (0 à 639) dans HL.

Notes : AF est modifié.

Instruction : RST 8.

Adresses d'exécution : 1626H (664), 162AH (6128).

BD52H

Étiquette : **FILL**.

Description : Remplit une surface avec l'encre dont le numéro est fourni dans A, réalisant ainsi la fonction BASIC FILL. En entrée, HL contient l'adresse d'une table de stock libre, et DE sa longueur (plus elle est grande, plus le remplissage est précis).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 19D5H (664), 19D9H (6128).

BD55H

Étiquette : **SCREEN**.

Description : Modifie l'enregistrement en mémoire de l'offset écran, fixé à la valeur fournie dans HL, et de la base écran, fixée à la valeur fournie dans A, mais sans sortir ces valeurs sur les périphériques. Pour une véritable modification, il faut donc appeler ensuite MOFFSET (BD1FH).

Notes : A et HL sont standardisés pour permettre l'appel de MOFFSET juste après.

Instruction : RST 8.

Adresses d'exécution : B41H (664), B45H (6128).

BD58H

Étiquette : **PRCARTAB**.

Description : Modification de la table des caractères d'imprimante. Cette table est positionnée aux valeurs fournies dans HL et suite, de la manière suivante : en premier, vient sur 1 octet le nombre de caractères de la table (maximum : 20), puis des paires d'octets composées du code à transformer lors de l'impression, puis du code à mettre à sa place. La valeur par défaut de cette table se trouve aux adresses 7E7H à 7FBH (664), 7F7H à 80BH (6128).

Notes : Le groupe 1 est modifié.

Instruction : RST 8.

Adresses d'exécution : 7FCH (664), 80CH (6128).

BD5BH (6128)

Étiquette : **PERMUT**.

Description : Permutation de blocs de RAM médiane inférieure (adresses 4000H à 7FFFH), numérotés 0, 4, 5, 6 ou 7, en fonction de la valeur de A modulo 8 en entrée : 0, 4, 5, 6 ou 7 = connecter le bloc de RAM correspondant ; 1 = l'écran 0 est recopié dans le 7, qui est placé dans la mémoire MI, ainsi que les sorties vers l'écran (plus rien ne s'affiche sur l'écran visible qui reste sur la RAM supérieure) ; 2 = idem, mais dans la mémoire MS (d'où une destruction des vecteurs et des paramètres !) ; 3 = comme 1, mais l'écran 7 n'est pas modifié.

Notes : En sortie, A contient le numéro de l'ancien bloc présent en RAM MI. Pour plus de détails, voir Chapitre 3.

Instruction : RST 8.

Adresse d'exécution : 397H (6128).

■ LES VECTEURS DE L'ARITHMÉTIQUE À VIRGULE FLOTTANTE

Voici maintenant la liste des vecteurs arithmétiques des Amstrad. Ces vecteurs sont en principe destinés au BASIC (quoiqu'ils puissent être très utiles au programmeur d'assembleur), c'est pourquoi ils se trouvent à des adresses variables suivant les CPC. De plus, parmi les vecteurs de l'arithmétique à virgule flottante donnés dans ce paragraphe, deux n'existent plus sur les CPC 664 et 6128.

Rappelons que les entiers sont codés sur 2 octets, et les réels (nombres à virgule flottante), sur 5 octets. Dans la suite, on désigne par FAC (accumulateur à virgule flottante) l'opérande des vecteurs, qui est toujours le réel pointé par HL. C'est toujours dans le FAC que les résultats des opérations sont stockés. On ne l'a donc pas précisé chaque fois.

Signalons une différence importante entre ces routines et les précédentes : pour des raisons de commodité, IX, et éventuellement IY sont modifiés de façon à pointer sur le ou les réels opérands.

Enfin un rappel évident, mais qui peut s'avérer justifié : lors des calculs, c'est la ROM qui est branchée aux adresses inférieures. Pas question, par conséquent, d'y placer le FAC ou vos opérands. Par contre, vous pouvez profiter des réels stockés en ROM .

BD3DH (464), **BD5EH** (664), **BD61H** (6128)

Étiquette : **COPY**.

Description : Copie le réel (c'est-à-dire 5 octets) pointé par DE dans le FAC. Sur 464, donne en plus dans A le type de la variable (2 = entier, 3 = chaîne, 5 = réel).

Notes : La retenue est mise en sortie.

Instruction : RST 28.

Adresses d'exécution : 2E18H (464), 2F91H (664 et 6128).

BD40H (464), **BD61H** (664), **BD64H** (6128)

Étiquette : **IR**.

Description : Convertit l'entier fourni dans HL en un réel placé dans le FAC, dont le signe est fourni par le signe de A (bit 7 mis = réel négatif). En entrée, l'adresse du FAC doit être fournie dans DE.

Notes : DE est modifié.

Instruction : RST 28.

Adresses d'exécution : 2E29H (464), 2F9FH (664 et 6128).

LISTE DES REELS DE LA ROM ET DE LEURS ADRESSES

Constantes	Adresses (464)	Adresses (664 et 6128)
10	2F53H	30F5H
100	2F58H	30FAH
1000	2F5DH	30FFH
10000	2F62H	3104H
10 ⁵	2F67H	3109H
10 ⁶	2F6CH	310EH
10 ⁷	2F71H	3113H
10 ⁸	2F76H	3118H
10 ⁹	2F7BH	311DH
10 ¹⁰	2F80H	3122H
10 ¹¹	2F85H	3127H
10 ¹²	2F8AH	312CH
10 ¹³	2F8FH	3131H
1/SQR(2)	3081H	3220H
LOG(2)	3086H	3225H
LOG ₁₀ (2)	308BH	322AH
1/2	30CCH	326BH
1/LOG(2)	30FBH	329DH
M1 *	3100H	32A2H
M2 *	3105H	32A7H
PI	31A9H	2F73H
PI/2	3205H	339CH
1/PI	321DH	33B4H
1/180	3222H	33B9H
PI/180	3227H	33BEH
180/PI	322CH	33C3H
1	3332H	2F82H

* M1 et M2 sont les logarithmes respectifs du plus grand nombre autorisé ($1.7014 E38 = 2^{127}$) et du plus petit ($2.938735 E-39 = 2^{-128}$) ; ils valent donc 88.0296919 et -88.7228391.

BD43H (464), **BD64H** (664), **BD67H** (6128)

Étiquette : **I4R**.

Description : Convertit un entier codé sur 4 octets pointés en entrée par HL (de l'octet de poids le plus faible à celui de poids le plus fort) en un réel dont le signe est en outre celui de A en entrée (comme pour la précédente). Ce réel (compris entre -2^{32} et $2^{32}-1$) est stocké à l'adresse pointée par HL, à la place de l'entier.

Notes : AF et IX sont modifiés : IX pointe sur le FAC en sortie.

Instruction : RST 28.

Adresses d'exécution : 2E55H (464), 2FC8H (664 et 6128).

BD46H (464), **BD67H** (664), **BD6AH** (6128)

Étiquette : **RI**.

Description : Convertit le FAC en un entier placé en sortie dans HL. Pour la valeur de AF en sortie, voir SOM (BD7CH, etc.).

Notes : Comme la précédente.

Instruction : RST 28.

Adresses d'exécution : 2E66H (464), 2FD9H (664 et 6128).

BD49H (464), **BD6AH** (664), **BD6DH** (6128)

Étiquette : **RI2**.

Description : Comme la précédente, mais le résultat est dans les deux premiers octets du FAC initial.

Notes : Comme I4R (BD43H, etc.).

Instruction : RST 28.

Adresses d'exécution : 2E8EH (464), 3001H (664 et 6128).

BD4CH (464), **BD6DH** (664), **BD70H** (6128)

Étiquette : **FIX**.

Description : Exécute sur le FAC la fonction BASIC FIX. Le résultat est un entier sur 4 octets mis dans le FAC si la retenue est mise. Si elle ne l'est pas, c'est que le réel initial était trop grand (supé-

rieur à 2⁶⁴). D'autre part, le signe du résultat est donné par celui de B (bit 7 mis = négatif).

Notes : Comme I4R (BD43H, etc.), et BC est aussi modifié.

Instruction : RST 28.

Adresses d'exécution : 2EA1H (464), 3014H (664 et 6128).

BD4FH (464), **BD70H** (664), **BD73H** (6128)

Étiquette : INT.

Description : Comme la précédente, avec INT.

Notes : Comme la précédente.

Instruction : RST 28.

Adresses d'exécution : 2EACH (464), 3055H (664 et 6128).

BD52H (464), **BD73H** (664), **BD76H** (6128)

Étiquette : aucune.

Description : Cette routine est utilisée par le BASIC pour l'impression des nombres ; en elle-même, elle ne présente pas d'intérêt.

Instruction : RST 28.

Adresses d'exécution : 2EB6H (464), 305FH (664 et 6128).

BD55H (464), **BD76H** (664), **BD79H** (6128)

Étiquette : M10.

Description : Multiplie le FAC par 10 à la puissance A. En sortie, AF est positionné comme après une somme (voir la routine suivante).

Notes : BC, DE et IX sont modifiés : IX pointe sur le FAC.

Instruction : RST 28.

Adresses d'exécution : 2F1DH (464), 30C6H (664 et 6128).

BD58H (464), **BD79H** (664), **BD7CH** (6128)

Étiquette : SOM.

Description : Ajoute au FAC le réel pointé par DE. AF est positionné en fonction du résultat : si la retenue est mise, tout est OK, et A contient le quatrième octet du résultat (dont le bit 7 donne le signe). Si la retenue n'est pas mise, une erreur s'est produite selon la valeur de A : si A = FFH il y a eu dépassement par le haut (nom-

bre trop grand). Dans ce cas le FAC contient 1,70141 E38 (nombre maximal autorisé). D'autres erreurs sont possibles pour d'autres opérations (voir celles-ci).

Notes : BC, DE, IX et IY sont modifiés : IX pointe sur le FAC et IY sur la valeur d'entrée de DE (deuxième opérande). Notez que si le résultat est inférieur à 2^{-128} , il est considéré comme nul.

Instruction : RST 28.

Adresses d'exécution : 333FH (464), 34A2H (664 et 6128).

BD5BH (464)

Étiquette : **DIFF1**.

Description : Comme la précédente, mais en soustrayant le réel pointé par DE du FAC.

Notes : Comme la précédente.

Instruction : RST 28.

Adresses d'exécution : 3337H (464).

BD5EH (464), **BD7FH** (664), **BD82H** (6128)

Étiquette : **DIFF2**.

Description : Comme SOM (BD58H, etc.), mais en soustrayant le FAC du réel pointé par DE.

Notes : Comme SOM (BD58H, etc.).

Instruction : RST 28.

Adresses d'exécution : 333BH (464), 349EH (664 et 6128).

BD61H (464), **BD82H** (664), **BD85H** (6128)

Étiquette : **PROD**.

Description : Comme SOM (BD58H, etc.), mais en faisant le produit.

Notes : Comme SOM (BD58H, etc.).

Instruction : RST 28.

Adresses d'exécution : 3315H (464), 3577H (664 et 6128).

BD64H (464), **BD85H** (664), **BD88H** (6128)

Étiquette : **DIV**.

Description : Comme SOM (BD58H, etc.), mais en divisant le FAC par le réel pointé par DE. Autre état d'erreur possible : si la rete-

nue n'est pas mise et que $A=0$, alors il y a eu tentative de division par zéro. Dans ce cas le FAC n'est pas modifié.

Notes : Comme SOM (BD58H, etc.).

Instruction : RST 28.

Adresses d'exécution : 349EH (464), 3604H (664 et 6128).

BD67H (464)

Étiquette : **M2**.

Description : Multiplie le FAC par 2 à la puissance A. En sortie, comme pour M10 (BD79H, etc.).

Notes : Comme RI (BD6AH, etc.). BC est modifié en cas de dépassement.

Instruction : RST 28.

Adresses d'exécution : 3578H (464).

BD6AH (464), **BD8BH** (664), **BD8EH** (6128)

Étiquette : **RCOMP**.

Description : Compare le réel pointé par DE avec le FAC. S'il lui est égal, A vaut zéro et F est positionné en conséquence ; s'il lui est supérieur, A vaut FFH et la retenue est mise ; enfin s'il lui est inférieur, A vaut 1 et la retenue n'est pas mise.

Notes : IX et IY sont modifiés, comme pour SOM (BD58H, etc.).

Instruction : RST 28.

Adresses d'exécution : 359AH (464), 36DFH (664 et 6128).

BD6DH (464), **BD8EH** (664), **BD91H** (6128)

Étiquette : **RCHSGN**.

Description : Change le signe du FAC. En sortie, A contient la nouvelle mantisse.

Notes : Comme RI (BD6AH, etc.).

Instruction : RST 28.

Adresses d'exécution : 35F8H (464), 3731H (664 et 6128).

BD70H (464), **BD91H** (664), **BD94H** (6128)

Étiquette : **SGN**.

Description : Teste le signe du FAC. En sortie, A vaut 0 si le FAC est nul, 1 s'il est positif, FFH sinon. F est positionné en conséquence.

Notes : IX pointe sur le FAC en sortie.

Instruction : RST 28.

Adresses d'exécution : 35E8H (464), 3727H (664 et 6128).

BD73H (464), **BD94H** (664), **BD97H** (6128)

Étiquette : **DEG**.

Description : Change le mode trigonométrique : si A vaut 0 en entrée, on passe en radians, sinon en degrés.

Notes : Tous les registres sont conservés.

Instruction : RST 28.

Adresses d'exécution : 31AEH (464), 3345H (664 et 6128).

BD76H (464), **BD97H** (664), **BD9AH** (6128)

Étiquette : **PI**.

Description : Place le réel PI (3,141592...) dans le FAC.

Notes : AF et DE sont modifiés.

Instruction : RST 28.

Adresses d'exécution : 31A3H (464), 2F73H (664 et 6128).

BD79H (464), **BD9AH** (664), **BD9DH** (6128)

Étiquette : **SQR**.

Description : Calcule la racine carrée du FAC, réalisant ainsi la fonction BASIC SQR. En sortie, AF est positionné comme pour la suivante.

Notes : AF, BC, DE, IX et IY sont modifiés : IX pointe sur le FAC en sortie.

Instruction : RST 28.

Adresses d'exécution : 310AH (464), 32ACH (664 et 6128).

BD7CH (464), **BD9DH** (664), **BDA0H** (6128)

Étiquette : **PUISS**.

Description : Comme SOM (BD7CH, etc.), mais en réalisant la fonction puissance (le FAC est porté à la puissance du réel pointé par

DE). En sortie, AF est comme pour SOM, avec les erreurs supplémentaires possibles si la retenue n'est pas mise : A=0 signifie division par zéro, A=1 signifie opération impossible (FAC négatif et exposant non entier).

Notes : Comme SOM (BD7CH, etc.).

Instruction : RST 28.

Adresses d'exécution : 310DH (464), 32AFH (664 et 6128).

BD7FH (464), **BDA0H** (664), **BDA3H** (6128)

Étiquette : **LOG**.

Description : Comme SQR (BD9DH, etc.), mais avec le logarithme naturel (LOG). États d'erreurs comme ci-dessus ; avec en plus, si A=FEH, logarithme d'un nombre négatif.

Notes : Comme SQR (BD9DH, etc.).

Instruction : RST 28.

Adresses d'exécution : 3014H (464), 31B6H (664 et 6128).

BD82H (464), **BDA3H** (664), **BDA6H** (6128)

Étiquette : **LOG10**.

Description : Comme la précédente, mais avec le logarithme décimal (LOG10).

Notes : Comme SQR (BD9DH, etc.).

Instruction : RST 28.

Adresses d'exécution : 300FH (464), 31B1H (664 et 6128).

BD85H (464), **BDA6H** (664), **BDA9H** (6128)

Étiquette : **EXP**.

Description : Comme SQR (BD9DH, etc.), mais avec l'exponentielle (EXP).

Notes : Comme SQR (BD9DH, etc.).

Instruction : RST 28.

Adresses d'exécution : 3090H (464), 322FH (664 et 6128).

BD88H (464), **BDA9H** (664), **BDACH** (6128)

Étiquette : **SIN**.

Description : Comme SQR (BD9DH, etc.), mais avec le sinus (SIN).

Notes : Comme SQR (BD9DH, etc.).

Instruction : RST 28.

Adresses d'exécution : 31BCH (464), 3353H (664 et 6128).

BD8BH (464), **BDACH** (664), **BDAFH** (6128)

Étiquette : **COS**.

Description : Comme SQR (BD9DH, etc.), mais avec le cosinus (COS).

Notes : Comme SQR (BD9DH, etc.).

Instruction : RST 28.

Adresses d'exécution : 31B2H (464), 3349H (664 et 6128).

BD8EH (464), **BDAFH** (664), **BDB2H** (6128)

Étiquette : **TAN**.

Description : Comme SQR (BD9DH, etc.), mais avec la tangente (TAN). État d'erreur (retenue non mise) où $A=0$ équivaut à division par zéro.

Notes : Comme SQR (BD9DH, etc.).

Instruction : RST 28.

Adresses d'exécution : 3231H (464), 33C8H (664 et 6128).

BD91H (464), **BDB2H** (664), **BDB5H** (6128)

Étiquette : **ATN**.

Description : Comme SQR (BD9DH, etc.), mais avec l'arc-tangente (ATN).

Notes : Comme SQR (BD9DH, etc.).

Instruction : RST 28.

Adresses d'exécution : 3241H (464), 33D8H (664 et 6128).

BD94H (464), **BDB5H** (664), **BDB8H** (6128)

Étiquette : **I5R**.

Description : Transforme un nombre entier de 5 octets en un réel placé dans le FAC. Les 4 octets de poids fort doivent se trouver dans le FAC en entrée, rangés par poids croissants. Le cinquième octet est nul, mais la routine additionne à l'entier un nombre cal-

culé à partir des deux bits supérieurs de C : 00=ajoute 0 ; 01 et 10=ajoute 80H ; 11=ajoute 100H. En outre, le réel créé aura le signe de A (bit 7 mis=réel négatif).

Notes : Comme I4R (BD43H, etc.).

Instruction : RST 28.

Adresses d'exécution : 2E5EH (464), 2FD1H (664 et 6128).

BD97H (464), **BDB8H** (664), **BDBBH** (6128)

Étiquette : **RNDINIT**.

Description : Initialisation du générateur de nombres aléatoires.

Notes : HL est modifié.

Instruction : RST 28.

Adresses d'exécution : 2F94H (464), 3136H (664 et 6128).

BD9AH (464), **BDBBH** (664), **BDBEH** (6128)

Étiquette : **RANDOM**.

Description : Prend le FAC comme générateur des nombres aléatoires, réalisant ainsi la fonction BASIC RANDOMIZE. Le FAC ne doit pas être nul en entrée.

Notes : En sortie, IX pointe sur le FAC qui n'est pas modifié, et le groupe 1 est modifié.

Instruction : RST 28.

Adresses d'exécution : 2FA1H (464), 3143H (664 et 6128).

BD9DH (464), **BD7CH** (664), **BD7FH** (6128)

Étiquette : **RND**.

Description : Génère un nombre aléatoire et le place dans le FAC, réalisant la fonction BASIC RND.

Notes : En sortie, IX pointe sur le FAC, et AF, BC et DE sont modifiés. La retenue est mise.

Instruction : RST 28.

Adresses d'exécution : 2FB7H (464), 3159H (664 et 6128).

BDA0H (464), **BD7FH** (664), **BD8BH** (6128)

Étiquettes : **RND0**.

Description : Donne le dernier nombre généré par la précédente, réalisant la fonction BASIC RND(0).

Notes : Comme la précédente.

Instruction : RST 28.

Adresses d'exécution : 2FE6H (464), 3188H (664 et 6128).

■ LES VECTEURS DE L'ARITHMÉTIQUE ENTIÈRE

Les vecteurs suivants n'existent que sur le 464, car sur le 664 et le 6128 ils ont été retirés de la ROM inférieure pour des raisons de place mémoire. Toutefois, le listing du programme *ENTIERS* vous permettra de disposer également de ces vecteurs qui peuvent être très commodes, à l'exception des deux premiers, de peu d'intérêt.

BDA3H (464)

Étiquette : aucune.

Description : Place dans B la valeur de H (bit 7 donne le signe), 0 dans E, 2 dans C, et remplace HL par sa valeur absolue.

Notes : AF est modifié.

Instruction : RST 28.

Adresses d'exécution : 3708H (464).

BDA6H (464)

Étiquette : aucune.

Description : Place 0 dans B et E, et 2 dans C.

Instruction : RST 28.

Adresses d'exécution : 370EH (464).

BDA9H (464) **A500H** (programme *ENTIERS*)

Étiquette : aucune.

Description : Routine utilisée par les suivantes : positionne B en fonction du signe de HL.

Notes : AF est modifié.

Instruction : RST 28.

Adresses d'exécution : 3715H (464).

BDACH (464) **A513H** (programme *ENTIERS*)

Étiquette : **ISOM**

Description : Additionne HL et DE et positionne AF en fonction du résultat : si OK, la retenue est mise et A conservé, sinon (débordement) la retenue est nulle et A vaut FFH. Le flag z est mis si le résultat est nul.

Instruction : RST 28.

Adresses d'exécution : 3728H (464).

BDAFH (464) **A51CH** (programmes *ENTIERS*)

Étiquette : **IDIFF1**.

Description : Soustrait DE de HL et positionne AF comme la précédente.

Instruction : RST 28.

Adresses d'exécution : 3731H (464).

BDB2H (464) **A51BH** (programme *ENTIERS*)

Étiquette : **IDIFF2**.

Description : Comme la précédente, mais HL vaut DE – HL en sortie.

Notes : DE vaut en sortie la valeur initiale de HL.

Instruction : RST 28.

Adresses d'exécution : 3730H (464).

BDB5H (464) **A524H** (programme *ENTIERS*)

Étiquette : **IPROD**.

Description : Multiplie DE et HL en tenant compte de leurs signes. Le résultat est placé dans HL et AF positionné comme pour ISOM (BDACH).

Notes : A est de toute façon modifié.

Instruction : RST 28.

Adresses d'exécution : 3739H (464).

BDB8H (464) **A565H** (programme *ENTIERS*)

Étiquette : **DIVEUC**.

Description : Division euclidienne de HL par DE, en tenant compte

des signes : en sortie, HL contient le quotient et DE le reste de la division.

Notes : AF est modifié.

Instruction : RST 28.

Adresses d'exécution : 377AH (464).

BDBBH (464) **A56CH** (programme *ENTIERS*)

Étiquette : **MOD**.

Description : Place dans HL la valeur de HL modulo DE, réalisant la fonction BASIC MOD.

Notes : AF, BC et DE sont modifiés.

Instruction : RST 28.

Adresses d'exécution : 3781H (464).

BDBEH (464) **A53BH** (programme *ENTIERS*)

Étiquette : **IPRODABS**.

Description : Comme IPROD (BDB5H), mais sans tenir compte des signes éventuels de DE et HL. La retenue est nulle en cas de dépassement.

Notes : AF est modifié.

Instruction : RST 28.

Adresses d'exécution : 3750H (464).

BDC1H (464) **A577H** (programme *ENTIERS*)

Étiquette : **IDIVABS**.

Description : Comme DIVEUC (BDB8H), avec les mêmes réserves que pour la précédente.

Notes : AF est modifié.

Instruction : RST 28.

Adresses d'exécution : 378CH (464).

BDC4H (464) **A5D4H** (programme *ENTIERS*)

Étiquette : **ICOMP**.

Description : Compare DE et HL. Si $DE > HL$, A vaut FFH ; si $DE < HL$, A vaut 1 ; si $DE = HL$, A vaut 0. Dans tous les cas, F est positionné en conséquence.

Instruction : RST 28.

Adresses d'exécution : 37E9H (464).

BDC7H (464) **A5BFH** (programme *ENTIERS*)

Étiquette : **ICHSGN**.

Description : Change HL en son opposé. La retenue est mise si OK.

Notes : A est modifié. Son bit 7 donne le nouveau signe de HL.

Instruction : RST 28.

Adresses d'exécution : 37D4H (464).

BDCAH (464) **A5CBH** (programme *ENTIERS*)

Étiquette : **ISGN**.

Description : Teste le signe de HL et positionne AF en conséquence (voir ICOMP (BDC4H), en remplaçant DE par 0).

Instruction : RST 28.

Adresse d'exécution : 37E0H (464).

A5BCH (programme *ENTIERS*)

Étiquette : **ABS**.

Description : Remplace HL par sa valeur absolue.

Notes : AF est modifié.

Annexe A

Les instructions du Z80

Voici à présent la liste alphabétique des instructions du Z80. Elles sont classées par genre. On a précisé pour chacune leur durée (en temps machine d'un quart de millionième de seconde), leur longueur (en octets) et les flags modifiés. Les codes machine par contre ne sont pas précisés (classement par genre). Vous les trouverez chez R. Zaks, entre autres.

Les notations sont les suivantes :

b désigne un numéro de bit, soit un nombre de 0 à 7. Pour les restarts, on a mis *RST b*, mais la syntaxe varie suivant les Assembleurs.

cond désigne une condition, soit M, P, PO, PE, Z, NZ, C ou NC.

conds désigne une condition spéciale soit Z, NZ, C ou NC.

n désigne une valeur sur un octet, comprise entre 0 et 255.

k désigne une valeur sur un octet, comprise entre -128 et 127.

nn désigne une valeur sur deux octets.

r désigne un registre 8 bits parmi A, B, C, D, E, H, L.

dd désigne un double registre, soit BC, DE ou HL.

IX désigne indifféremment IX ou IY.

Un prime a été mis quand le même symbole intervient deux fois, comme dans LD *r,r'*.

Lorsque deux durées ont été données, c'est que l'instruction dépend soit d'une condition (dans ce cas la première durée est celle où la condition n'est pas remplie), soit d'une instruction répétitive (dans ce cas la première durée est celle qui correspond à un arrêt du bouclage).

Pour les flags, on a utilisé les notations suivantes :

- 0 flag mis à zéro.
- 1 flag mis à un.
- X flag modifié.
- flag conservé.

Instruction	Durée	Longueur	Flags			
			c	z	s	p/v
ADC A,n	7	2	X	X	X	X
ADC A,r	4	1	X	X	X	X
ADC A, (HL)	7	1	X	X	X	X
ADC A, (IX+n)	19	3	X	X	X	X
ADC HL,dd	15	2	X	X	X	X
ADC HL,SP	15	2	X	X	X	X
ADD A,n	7	2	X	X	X	X
ADD A,r	4	1	X	X	X	X
ADD A,(HL)	7	1	X	X	X	X
ADD A,(IX+n)	19	3	X	X	X	X
ADD HL,dd	11	1	X	—	—	—
ADD HL,SP	11	1	X	—	—	—
ADD IX,BC	15	2	X	—	—	—
ADD IX,DE	15	2	X	—	—	—
ADD IX,IX	15	2	X	—	—	—
ADD IX,SP	15	2	X	—	—	—
AND n	7	2	0	X	X	X
AND r	4	1	0	X	X	X
AND (HL)	7	1	0	X	X	X
AND (IX+n)	19	3	0	X	X	X
BIT b,r	8	2	—	X	X	X
BIT b,(HL)	12	2	—	X	X	X
BIT b,(IX+n)	20	4	—	X	X	X
CALL nn	17	3	—	—	—	—
CALL cond,nn	10/17	3	—	—	—	—
CCF	4	1	X	—	—	—
CP n	7	2	X	X	X	X
CP r	4	1	X	X	X	X
CP (HL)	7	1	X	X	X	X
CP (IX+n)	19	3	X	X	X	X
CPD	16	2	—	X	X	X
CPDR	16/21	2	—	X	X	X
CPI	16	2	—	X	X	X
CPIR	16/21	2	—	X	X	X
CPL	4	1	—	—	—	—
DAA	4	1	X	X	X	X
DEC r	4	1	—	X	X	X
DEC (HL)	11	1	—	X	X	X
DEC (IX+n)	23	3	—	X	X	X
DEC dd	6	1	—	—	—	—
DEC IX	10	2	—	—	—	—
DEC SP	6	1	—	—	—	—

Instruction	Durée	Longueur	Flags			
			c	z	s	p/v
DI	4	1	—	—	—	—
DJNZ k	8/13	2	—	—	—	—
EI	4	1	—	—	—	—
EX AF,AF'	4	1	échangés...			
EX DE,HL	4	1	—	—	—	—
EXX	4	1	—	—	—	—
EX (SP),HL	19	1	—	—	—	—
EX (SP),IX	23	2	—	—	—	—
HALT	4	1	—	—	—	—
IM 0,1 ou 2	8	2	—	—	—	—
IN r,(C)	12	2	—	X	X	X
IN A,(n)	11	2	—	—	—	—
INC r	4	1	—	X	X	X
INC (HL)	11	1	—	X	X	X
INC (IX+n)	23	2	—	X	X	X
INC dd	6	1	—	—	—	—
INC IX	10	2	—	—	—	—
INC SP	6	1	—	—	—	—
IND	16	2	—	X	X	X
INDR	16/21	2	—	1	X	X
INI	16	2	—	X	X	X
INIR	16/21	2	—	1	X	X
JP nn	10	3	—	—	—	—
JP cond,nn	10/10	3	—	—	—	—
JP (HL)	4	1	—	—	—	—
JP (IX)	8	2	—	—	—	—
JR k	12	2	—	—	—	—
JR conds,k	7/12	2	—	—	—	—
LD r,r'	4	1	—	—	—	—
LD (HL),r	7	1	—	—	—	—
LD (IX+n),r	19	3	—	—	—	—
LD r,n	7	2	—	—	—	—
LD (HL),n	10	2	—	—	—	—
LD (IX+n),n'	19	4	—	—	—	—
LD r,(HL)	7	1	—	—	—	—
LD r,(IX+n)	19	3	—	—	—	—
LD A,I	9	2	—	X	X	X
LD I,A	9	2	—	—	—	—
LD A,R	9	2	—	X	X	X
LD R,A	9	2	—	—	—	—
LD A,(dd)	7	1	—	—	—	—
LD (dd),A	7	1	—	—	—	—

Instruction	Durée	Longueur	Flags			
			c	z	s	p/v
LD (nn),A	13	3	—	—	—	—
LD A,(nn)	13	3	—	—	—	—
LD dd,nn	10	3	—	—	—	—
LD IX,nn	14	4	—	—	—	—
LD SP,nn	10	3	—	—	—	—
LD SP,HL	6	1	—	—	—	—
LD SP,IX	10	2	—	—	—	—
LD dd,(nn)	20,HL:16	3	—	—	—	—
LD (nn),dd	20,HL:16	3	—	—	—	—
LD IX,(nn)	20	4	—	—	—	—
LD (nn),IX	20	4	—	—	—	—
LD SP,(nn)	20	4	—	—	—	—
LD (nn),SP	20	4	—	—	—	—
LDD	16	2	—	—	—	X
LDDR	16/21	2	—	—	—	0
LDI	16	2	—	—	—	X
LDIR	16/21	2	—	—	—	0
NEG	8	2	X	X	X	X
NOP	4	1	—	—	—	—
OR n	7	2	0	X	X	X
OR r	4	1	0	X	X	X
OR (HL)	7	1	0	X	X	X
OR (IX+n)	19	3	0	X	X	X
OUT (C),r	12	2	—	—	—	—
OUT (n),A	11	2	—	—	—	—
OUTD	16	2	—	X	X	X
OTDR	16/21	2	—	1	X	X
OUTI	16	2	—	X	X	X
OTDI	16/21	2	—	1	X	X
POP AF	10	1	X	X	X	X
POP dd	10	1	—	—	—	—
POP IX	14	1	—	—	—	—
PUSH AF	11	1	—	—	—	—
PUSH dd	11	1	—	—	—	—
PUSH IX	14	1	—	—	—	—
RES b,r	8	2	—	—	—	—
RES b,(HL)	15	2	—	—	—	—
RES b,(IX+n)	23	4	—	—	—	—
RET	10	1	—	—	—	—
RET cond	5/11	1	—	—	—	—
RETI	14	2	—	—	—	—
RETN	14	2	—	—	—	—

Instruction	Durée	Longueur	Flags			
			c	z	s	p/v
RL r	8	2	X	X	X	X
RL (HL)	15	2	X	X	X	X
RL (IX+n)	23	4	X	X	X	X
RLA	4	1	X	—	—	—
RLC r	8	2	X	X	X	X
RLC (HL)	15	2	X	X	X	X
RLC (IX+n)	23	4	X	X	X	X
RLCA	4	1	X	—	—	—
RLD	18	2	—	X	X	X
RR r	8	2	X	X	X	X
RR (HL)	15	2	X	X	X	X
RR (IX+n)	23	4	X	X	X	X
RRA	4	1	X	—	—	—
RRC r	8	2	X	X	X	X
RRC (HL)	15	2	X	X	X	X
RRC (IX+n)	23	4	X	X	X	X
RRCA	4	1	X	—	—	—
RRD	18	2	—	X	X	X
RST b	11	1	—	—	—	—
SBC A,n	7	1	X	X	X	X
SBC A,r	4	1	X	X	X	X
SBC A,(HL)	7	1	X	X	X	X
SBC A,(IX+n)	19	3	X	X	X	X
SBC HL,dd	15	2	X	X	X	X
SBC HL,SP	15	2	X	X	X	X
SCF	4	1	1	—	—	—
SET b,r	8	2	—	—	—	—
SET b,(HL)	15	2	—	—	—	—
SET b,(IX+n)	23	4	—	—	—	—
SLA r	8	2	X	X	X	X
SLA (HL)	15	2	X	X	X	X
SLA (IX+n)	23	4	X	X	X	X
SRA r	8	2	X	X	X	X
SRA (HL)	15	2	X	X	X	X
SRA (IX+n)	23	4	X	X	X	X
SRL r	8	2	X	X	X	X
SRL (HL)	15	2	X	X	X	X
SRL (IX+n)	23	4	X	X	X	X
SUB n	7	2	X	X	X	X
SUB r	4	1	X	X	X	X
SUB (HL)	7	1	X	X	X	X
SUB (IX+n)	19	3	X	X	X	X

Instruction	Durée	Longueur	Flags			
			c	z	s	p/v
XOR n	7	2	0	X	X	X
XOR r	4	1	0	X	X	X
XOR (HL)	7	1	0	X	X	X
XOR (IX+n)	19	3	0	X	X	X

Annexe B

Les rotations et décalages

La liste des rotations et décalages, accompagnée de schémas, est donnée ci-dessous. On a précisé pour chacun les flags modifiés, la longueur en octets et la durée (bien que ces renseignements soient aussi précisés par l'annexe précédente).

On notera surtout des différences peu évidentes à priori, entre les instructions type *RR A* et *RRA*. Dans les deux cas, la rotation effectuée sur l'accumulateur est la même. Mais la première est plus longue (2 octets contre 1), deux fois plus lente et de plus modifie tous les flags, alors que *RRA* ne modifie que la retenue. La même remarque s'applique aux autres rotations binaires (*RLA*, *RLCA*, *RRCA*).

Les flags modifiés ne comprennent pas *h* et *n*, sans intérêt. De plus, *r* désigne ici un registre 8 bits au sens étendu, soit : A, B, C, D, E, H, L, (HL), (IX + n), (IY + n) (ce dernier n'a pas été distingué du précédent pour le calcul des longueurs et durées).

ROTATIONS

RL r

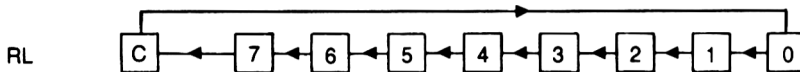
Rotation à gauche du registre, en passant par la retenue.

Les bits sont décalés, bit 0 vers bit 1, etc. Le bit 7 passe dans la retenue et la retenue passe dans le bit 0.

Flags modifiés : Tous.

Longueur : 2, sauf (IX + n) : 4.

Durée : 8, sauf (HL) : 15 et (IX + n) : 23.



RLA

Comme la précédente, mais sur l'accumulateur A.

Flags modifiés : Aucun, sauf la retenue.

Longueur : 1.

Durée : 4.

RLC r

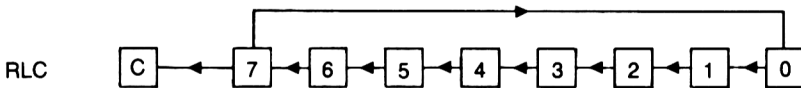
Rotation à gauche du registre, sans passer par la retenue.

Les bits sont décalés, bit 0 vers bit 1, etc., et bit 7 dans le bit 0. Le bit 7 est en outre aussi copié dans la retenue.

Flags modifiés : Tous.

Longueur : 2, sauf (IX+n) : 4.

Durée : 8, sauf (HL) : 15 et (IX+n) : 23.



RLCA

Comme la précédente, mais sur l'accumulateur A.

Flags modifiés : Aucun, sauf la retenue.

Longueur : 1.

Durée : 4.

RR r

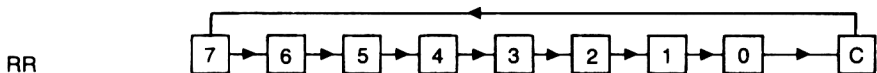
Rotation à droite du registre, en passant par la retenue.

Les bits sont décalés, bit 7 vers bit 6, etc. Le bit 0 passe dans la retenue et la retenue passe dans le bit 7.

Flags modifiés : Tous.

Longueur : 2, sauf (IX+n) : 4.

Durée : 8, sauf (HL) : 15 et (IX+n) : 23.



RRA

Comme la précédente, mais sur l'accumulateur A.

Flags modifiés : Aucun, sauf la retenue.

Longueur : 1.

Durée : 4.

RRC r

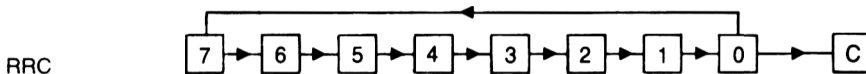
Rotation à droite du registre, sans passer par la retenue.

Les bits sont décalés, bit 7 vers bit 6, etc., et bit 0 dans le bit 7. Le bit 0 est en outre aussi copié dans la retenue.

Flags modifiés : Tous.

Longueur : 2, sauf (IX+n) : 4.

Durée : 8, sauf (HL) : 15 et (IX+n) : 23.



RRCA

Comme la précédente, mais sur l'accumulateur A.

Flags modifiés : Aucun, sauf la retenue.

Longueur : 1.

Durée : 4.

DÉCALAGES

SLA r

Décalage arithmétique à gauche du registre (multiplication par 2).

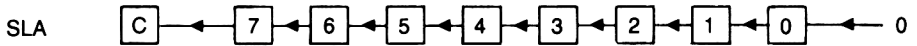
Les bits sont recopiés, bit 0 dans bit 1, etc. Le bit 7 est copié dans la retenue et le bit 0 est annulé.

Flags modifiés : Tous.

Longueur : 2, sauf (IX+n) : 4.

Durée : 8, sauf (HL) : 15 et (IX+n) : 23.

Note : S'obtient plus simplement sur A par *ADD A,A*.



SRA r

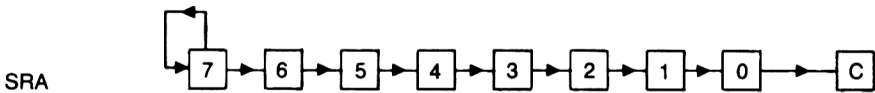
Décalage arithmétique à droite du registre.

Les bits sont recopiés, bit 7 dans bit 6, etc. Le bit 0 est copié dans la retenue et le bit 7 est inchangé.

Flags modifiés : Tous.

Longueur : 2, sauf (IX+n) : 4.

Durée : 8, sauf (HL) : 15 et (IX+n) : 23.



SRL r

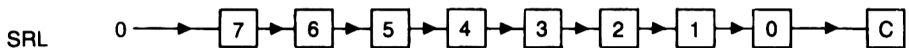
Décalage logique à droite du registre (division par 2).

Les bits sont recopiés, bit 7 dans bit 6, etc. Le bit 0 est copié dans la retenue et le bit 7 est annulé.

Flags modifiés : Tous.

Longueur : 2, sauf (IX+n) : 4.

Durée : 8, sauf (HL) : 15 et (IX+n) : 23.



ROTATIONS DÉCIMALES

Les deux rotations suivantes s'effectuent par quartets (groupes de 4 bits) sur A et l'octet pointé par HL, soit (HL), exclusivement.

RLD

Rotation décimale à gauche.

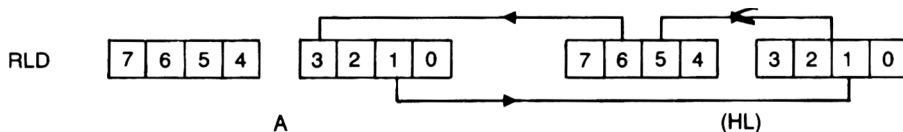
Rotation sur le quartet inférieur de A et ceux de (HL), dans cet ordre :

quartet inférieur de A vers quartet inférieur de (HL) vers quartet supérieur de (HL) (ce dernier repassant dans le quartet inférieur de A).

Flags modifiés : Tous, sauf la retenue.

Longueur : 2 (codes ED 6F).

Durée : 18.



RRD

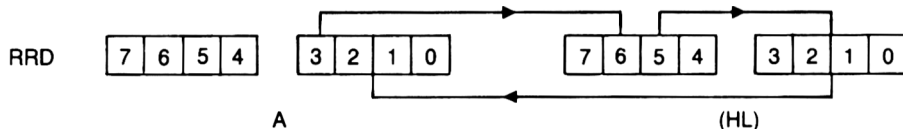
Rotation décimale à droite.

Rotation sur le quartet inférieur de A et ceux de (HL), dans cet ordre : quartet inférieur de A vers quartet supérieur de (HL) vers quartet inférieur de (HL) (ce dernier repassant dans le quartet inférieur de A).

Flags modifiés : Tous, sauf la retenue.

Longueur : 2 (codes ED 67).

Durée : 18.



Annexe C

Les adresses intéressantes

Voici un recueil d'adresses qui peuvent servir. Il s'agit essentiellement de paramètres du système ou du BASIC que vous pouvez modifier ou lire, quelquefois de points d'entrée (mais ceux des routines système, listées au Chapitre 8, n'ont pas été redonnés).

En général ces adresses ne sont pas les mêmes pour les trois CPC. De ce fait, il arrive que deux adresses soient données. La première est celle des CPC 464, la seconde des CPC 664 et 6128.

Lorsque la mention 464, 664 ou 6128 figure entre parenthèses, c'est que l'adresse ne concerne que le ou les CPC correspondants.

Les adresses sont données en hexadécimal, par ordre croissant (pour le 464).

Un D entre parenthèses signifie : version avec lecteur de disquette seulement.

Paramètres du lecteur de disquette

A700 (D)	Lecteur courant (0=A, 1=B).
A701 (D)	Numéro d'utilisateur (<i>USER</i>) courant.
A702 (D)	Lecteur actif.
A708 (D)	Indicateur pour fichiers en lecture : FFH=pas de fichier ouvert, sinon numéro du lecteur.
A72C (D)	Indicateur pour fichiers en écriture : FFH=pas de fichier ouvert, sinon numéro du lecteur.
A751 (D)	Adresse du tampon de lecture (2 octets).
A755 (D)	En-tête du fichier en lecture (64 octets).
A79B (D)	Adresse du tampon d'écriture (2 octets).
A79F (D)	En-tête du fichier en écriture (64 octets).
A89F (D)	Numéro du premier secteur de chaque piste (41H, C1H ou 1 suivant le format de la disquette).
A8A0 (D)	Nombre de secteurs par piste (9 ou 8 suivant le format).
A8A3 (D)	Octet de remplissage des pistes (E5H en principe).

Paramètres BASIC

ADAA/AD90	Numéro d'erreur.
ADAB/AD91	Adresse de la dernière instruction exécutée (2 octets).
AE7B/AE5E	Adresse du haut de mémoire (<i>HIMEM</i>) (2 octets).
AE81/AE64	Adresse du début du programme (2 octets).
AE83/AE66	Adresse de la fin du programme (2 octets)
AE85/AE68	Adresse du début de la table des variables (2 octets).
AE87/AE6A	Adresse du début des variables en matrices (2 octets).
AE89/AE6C	Adresse de fin des variables en matrices (2 octets).
B08C/B06F	Adresse de la pile BASIC (2 octets).
B0C1/B09F	Type de la variable contenue dans l'accumulateur.
B0C2/B0A0	Accumulateur pour variables (5 octets).

Les paramètres système

B1C8/B7C3	Mode d'écran courant.
B1C9/B7C4	Offset écran.
B1CB/B7C5	Adresse de l'écran (2 octets).
B1D7/B7D2	Durées des encres clignotantes (2 octets).
B1DA/B7D6	Couleurs courantes (y compris du bord) (32 octets).
B20C/B6B5	Numéro de la fenêtre courante.
B20D/B6B6	Streams des huit fenêtres (par stream : 14 octets sur 464, 13 octets sur 664 et 6128) (112/104 octets).
B285/B726	Stream courant (14/13 octets), soit :
B285/B726	Position du curseur : ligne, colonne (2 octets).
B287/B728	Flag fenêtre : 0 = écran entier.
B288/B729	Coins de la fenêtre (haut puis bas) (4 octets).
B28C/B72D	Compteur de défilement (<i>ROLL COUNT</i>).
B72D	(664 et 6128) Bit 7 = flag VDU (0 = débranché) ; bits 0 et 1 comme ci-dessous.
B28D	(464) Flags curseur. Bit 0 : curseur utilisateur, bit 1 : curseur système (0 = affichage interdit).
B28E	(464) Flag VDU (0 = débranché).
B28F/B72F	Masque de la couleur d'écriture.
B290/B730	Masque de la couleur du fond.

B291/BB731	Mode d'affichage du fond (0 = forçage, 1 = ou exclusif, 2 = et, 3 = ou).
B292/B732	Usage inconnu ?
B293/B733	Flag écriture des caractères graphiques (0 = interdit).
B294/B734	Flag non nul s'il y a des caractères redéfinis.
B295/B735	Numéro du premier caractère redéfini.
B296/B736	Adresse de la table des matrices de caractères redéfinissables (2 octets).
B2C3/B763	Table des sauts aux caractères de contrôle (96 octets).
B328/B693	Coordonnées de l'origine graphique (x puis y) (4 octets).
B32C/B697	Coordonnées courantes du curseur graphique (4 octets).
B330/B69B	Limites de la fenêtre graphique (abscisses, puis ordonnées) (8 octets).
B338/B6A3	Masque de l'encre graphique d'écriture.
B339/B6A4	Masque de l'encre graphique du fond.
B6B2	(664 et 6128) Paramètres de MASK pour les tracés de lignes (3 octets).
B4E7	(464) État de la touche (SHIFT) (FFH si enfoncée).
B631	(664 et 6128) État de verrouillage de (CONTROL) (CAPS LOCK) (FFH si elle est verrouillée).
B4E8/B632	État de verrouillage de (CAPS LOCK) (<i>idem</i>).
B4E9/B633	Vitesse de répétition des touches.
B4EA/B634	Délai avant la première répétition des touches.
B451/B68B	Adresse de la table des touches sans (SHIFT) ni (CONTROL) (2 octets).
B543/B68D	Adresse de la table des touches avec (SHIFT) seul (2 octets).
B545/B68F	Adresse de la table des touches avec (CONTROL) (2 octets).
B547/B691	Adresse de la table des touches avec répétition (2 octets).
B60A/B2A6	15 enveloppes sonores de volume (ENV) de 16 octets chacune (240 octets).

B6FA/B396	15 enveloppes sonores de ton (<i>ENV</i>) (240 octets).
B800/B118	Flag messages cassette (0 = autorisés).
B802/B11A	Flag fichier ouvert sur cassette (0 = pas de fichier).
B803/B11B	Adresse du tampon pour le catalogue (2 octets).
B805/B11D	Adresse du tampon pour la lecture sur cassette (2 octets).
B84A/B162	Adresse du tampon pour l'écriture sur cassette (2 octets).
B8CD/B1E5	Octet de synchronisation.
B8D1/B1E9	Vitesse d'écriture sur cassette.

Adresses de la ROM BASIC

C002/C006	Initialisation. Envoi de "BASIC 1.0" (ou 1.1) et passage au mode Ready.
C064/C058	Entrée du mode Ready.
CA94/CB55	Entrée des messages d'erreur. Le numéro de l'erreur doit être fourni dans E sur 464, et dans A sur 664 et 6128.
FFAA	(464) Compare l'accumulateur à la table pointée par HL, jusqu'à ce qu'il y ait égalité (retenue mise) ou que (HL)=0 (retenue nulle) ; HL et F sont seuls modifiés.
FFB8	(464) Tester si HL=DE (flag z positionné et A modifié).
FFBE	(464) Tester si HL=BC (flag z positionné et A modifié).
FFC4	(464) Soustraction : DE=DE-HL. F et DE seuls modifiés.
FFCF	(464) Soustraction : HL=HL-DE. F et HL seuls modifiés.
FFDA	(464) Soustraction : BC=HL-DE. F et BC seuls modifiés.
FFE7	(464) Soustraction : HL=HL-BC. F et HL seuls modifiés.

Bibliographie

Voici une bibliographie classée par ordre de préférence. Il va sans dire que les commentaires n'engagent que l'auteur.

La programmation du Z80 par Rodney Zaks (éditions Sybex).

Pour tout savoir sur le Z80, ses instructions, sa vitesse, ses périphériques, etc., et même sur des structures de listes ! Un gros pavé qui devient indispensable à partir d'un certain niveau de programmation.

Le livre du lecteur de disquette Amstrad par Bruckman et Schieb (éditions Micro-application).

Tout, absolument tout, et même plus sur le lecteur de disquette. Indispensable si les routines système ne vous suffisent plus. Un peu brouillon parfois, mais un excellent listing du DOS.

Clefs pour Amstrad par Daniel Martin (éditions du PSI).

Beaucoup d'erreurs, d'oublis ou d'inexactitudes, même dans la seconde version (pour 664 et 6128). Mais le format du livre et une très bonne conception logique en font un bon mémorandum à garder sous la main.

Programmation en assembleur par Georges Fagot-Barraly (éditions Sybex).

Un petit livre pour débiter en assembleur. Des exemples simples, une présentation claire.

La bible de l'Amstrad CPC par Bruckman, Lothar, English et Gerits (éditions Micro-application).

Pour avoir le listing des ROM du 464, avec quelques commentaires (moins que dans le livre du lecteur de disquette). Un gros pavé nullement essentiel, pour les curieux. Est doté d'une suite pour 664 et 6128, dans le même ton.

Le livre de l'Amstrad CPC 464 - CPC 664 par Daniel Martin et Philippe Jadoul (éditions BCM, diffusé par PSI).

Un livre bien fait, mais dont je n'ai pas encore compris l'utilité. Copie un peu développée des *Clefs pour Amstrad*. On aurait préféré que ces dernières soient revues et corrigées.

***POUR UN CATALOGUE COMPLET
DE NOS PUBLICATIONS***

FRANCE
6-8, Impasse du Curé
75881 PARIS CEDEX 18
Tél. : (1) 42.03.95.95
Télex : 211801

U.S.A.
2344 Sixth Street
Berkeley, CA 94710
Tel. : (415) 848.8233
Telex : 336311

ALLEMAGNE
Vogelsanger. WEG 111
4000 Düsseldorf 30
Postfach N° 30.09.61
Tel. : (0211) 61 80 2-0
Telex : 08588163



Paris • Berkeley • Düsseldorf

Cet ouvrage est destiné à tous ceux qui ont déjà acquis les bases de la programmation en assembleur ou en langage machine. Il présente des méthodes de programmation en assembleur Z80 accompagnées de nombreux exemples de programmes d'application fonctionnant sur les Amstrad CPC 464, 664, et 6128.

L'AUTEUR

THOMAS LACHAND ROBERT est né en décembre 1966. De formation essentiellement scientifique, il prépare l'école Polytechnique.





MAXIMUM PROGRAMMER ASSISTANCE
FOR THE SYBEX

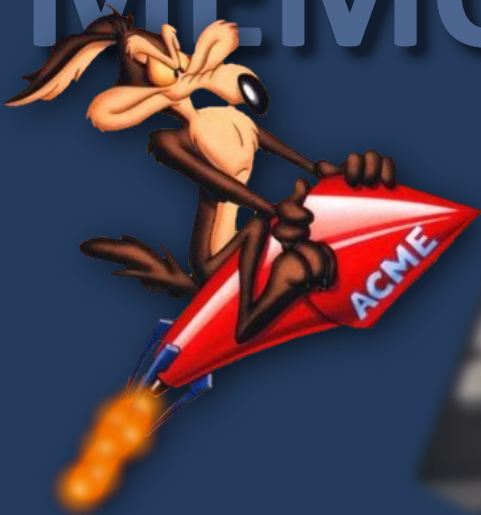


Document **numérisé**
avec amour par :

AMSTRAD

CPC 

MÉMOIRE ÉCRITE



<https://acpc.me/>