Jean-François Gallet Alain Pierrot

Les Astuces d'Alice 32 et 90

MATRA ET HACHETTE

Les auteurs tiennent à remercier les membres de l'équipe de MATRA-TANDY ÉLECTRONIQUE, qui ont aimablement communiqué de précieux renseignements techniques permettant la réalisation de ce livre.

Préambule

Les notations utilisées

Ce livre ne concerne qu'ALICE 90 et la nouvelle version d'ALICE, qui contient l'éditeur-assembleur. Comme cette dernière ne se distingue pas vraiment de la première version plus simple, dont il ne sera jamais question ici, on désignera systématiquement cette version qui contient l'éditeur-assembleur sous le nom d'ALICE 32, pour rappeler sa taille-mémoire (16 K de mémoire morte + 16 K de mémoire vive).

Les notations hexadécimale et binaire seront respectivement marquées par les préfixes $ et %, pour les distinguer de la notation décimale. Par exemple :

143 = $8F = %1ØØØ1111.

Le chiffre zéro est noté Ø, par opposition à la lettre O.

A l'intérieur d'un octet, les bits sont numérotés dans l'ordre suivant :

B7 B6 B5 B4 B3 B2 B1 BØ
1 Ø Ø Ø Ø Ø Ø Ø

La valeur de l'octet codé ci-dessus est $8Ø.

Informations communiquées par l'auteur le 1er mai 2011

Je lance ce sujet car j'ai retrouvé la 1ère épreuve du livre les Astuces d'Alice sur laquelle j'avais noté les erreurs que j'avais détectées. Sur l'autre exemplaire que je possède, elles ne sont pas corrigées, donc j'imagine que Matra Hachette a édité le livre sans en tenir compte. Donc voilà la liste des erreurs que j'avais vues à l'époque:

page 13 : c'est $FBD4 et non $F8D4
page 13 : c'est $FBD6 et non $F8D6
page 36 : c'est $DF90 et non $DFA0 (soit 53232 au lieu de 57248)
page 79 : c'est Alice 90 et non Alice 9 (OK, tout le monde avait deviné...)
page 159 : c'est un connecteur 36 points (et non 34), avec 2 précisions supplémentaires : broche 35: non utilisée, broche 36: Entrée son
page 160 : c'est le schéma d'un Alice 32, et page 162, celui d'un Alice 90 (ce n'était pas indiqué)
page 165 : le programme K7WRT se trouve en $DF90 et non DFA0

Bon, j'espère que cela ne vous avait pas trop perturbés...

Informations communiquées par un lecteur le 1er mai 2011

Il y en a d'autres !

- p 7 : M4"A"C0 : 1C : 2A+
- p 12 : POLCA à remplacer par OUTCA
- p 37 : K7REA, en $DFBC devient $DFAC
- p 84 : mode choisi 40 ou 80 caractères et cette définition n'est possible qu'en mode 40 caractères;
- p 105 : bas de la page après R1 en dessous c'est R3 et non R2
- p 149 : en haut de la page Basic 16 Ko de mémoire morte la place de 18 Ko, en bas de page $3745 à remplacer 2 fois par $3345
- p 150 : en bas de page "qui se trouve en $3250 et $3251 et non en $3280 et $3281
- p 159 : manque les barres de complémentation sur W, SEL, RESET, NMI
- Schéma du Alice 90 manque des n°s de connexion EF 9345 (Z12) (cf page hardware dans le wiki d'Alice carte mère Alice 90 rubrique sortie vidéo - schéma)
- p 165 : bas de page KSINP à remplacer par KDINP
- p 172 : bas de page Un registre d'état à 8 bits "F" et non "P" et manque pour un compteur programme à 16 bits "PC"

1

Les bases

Introduction

Cet ouvrage est destiné à compléter la documentation succincte qui accompagne votre micro-ordinateur ALICE 90 ou ALICE 32. Elle suppose de la part du lecteur un niveau déjà honnête pour la programmation du BASIC, ainsi qu'une connaissance minimale du langage assembleur : son but n'est pas l'initiation, mais plutôt la fourniture d'outils permettant la réalisation de programmes très élaborés, à la limite des programmes professionnels. Vous ne trouverez donc pas la liste des instructions du microprocesseur MC 68Ø3 (qui se trouve dans la brochure fournie avec ALICE), ni d'indications sur les bases de la programmation en assembleur (que vous pouvez trouver dans les livres cités dans la bibliographie, Annexe 1).

Les particularités du MC 68Ø3

Certains aspects du MC 68Ø3 sont originaux par rapport aux microprocesseurs que vous avez pu utiliser. Ce chapitre leur est consacré.

Le microprocesseur MC 68Ø3 représente l'évolution de la famille Motorola MC 68ØØ. Bien que compatible avec le MC 68ØØ, il présente plusieurs caractéristiques propres :
— la liste des instructions permet d'utiliser un nouveau registre, le registre D, constitué par la réunion des registres A et B. Ce registre permet la réalisation d'opérations sur 16 bits, comme MUL, qui évite de nombreuses boucles, donc une perte de temps. Les instructions LDD, STD, ADD, SUBD, ASLD, LSLD permettent des calculs pratiques sur des adresses ou de grands nombres.
— trois instructions nouvelles concernent le registre d'indexation, PSHX et PULX, qui opèrent sur la pile, et ABX, qui additionne les registres B et X.

— l'instruction CPX, trop souvent méconnue, voit sa puissance accrue; elle permet de positionner tous les registres de condition du 68Ø3, alors que le 68ØØ ne permet que les tests d'égalité.

Outre ces instructions particulièrement notables, il faut attirer l'attention sur un point que la lecture du guide d'ALICE ne permet pas de découvrir : l'aspect *"hardware"* du 68Ø3. Il possède en effet quelques particularités matérielles :
— une mémoire vive interne de 128 octets est accessible. Elle est très utilisée par le BASIC de Microsoft, et présente donc peu d'intérêt. sauf pour les programmes écrits en assembleur qui ne profitent d'aucune des ressources du BASIC. Dans ce dernier cas, cette zone mémoire à accès rapide permet de concevoir des applications plus performantes.
— deux ports d'entrées-sorties servent à gérer le clavier d'ALICE. On peut les utiliser dans un programme assembleur pour réaliser sa propre gestion du clavier, mais nous verrons qu'il y a d'autres méthodes, plus simples, pour redéfinir le clavier.
— un port d'entrées-sorties série est utilisé principalement pour l'impression. Le faible nombre de signaux disponibles restreint son domaine d'application : piloter un modem bi-directionnel simultané dans ces conditions est pratiquement impossible, mais on donnera un exemple d'utilisation dans un logiciel spécialement adapté pour faire communiquer deux ALICE
— une horloge programmable, bien que peu employée par le moniteur BASIC, offre des perspectives intéressantes pour la programmation en assembleur. En pratique, cette horloge peut servir à déclencher une fonction après un délai prédéterminé. Durant ce délai, le microprocesseur peut exécuter un autre programme.

Le programme moniteur

Un programme en assembleur est très rapide. bien adapté aux possibilités de la machine sur laquelle il s'exécute, et il occupe (en général) moins de place qu'un programme en langage évolué. Ce serait donc un outil de programmation idéal, n'étaient quelques inconvénients :
— comme toutes les opérations doivent être détaillées à l'aide des instructions élémentaires du processeur, certaines fonctions, élémentaires en BASIC, demandent des prodiges d'imagination au programmeur en assembleur.
— la mise au point des programmes est difficile, car une éventuelle erreur dans un programme assembleur vous oblige dans le meilleur des cas à reprendre le contrôle d'ALICE en faisant RESET. En cas de «beau plantage», il ne vous reste que la ressource d'éteindre. d'où l'intérêt de faire de fréquentes sauvegardes.
— il est très difficile de contrôler l'exécution d'un programme assembleur étape par étape.

L'ensemble de ces difficultés nous a amenés à écrire le programme MONITEUR qui se trouve en annexe à la fin de cet ouvrage. Avant de programmer les exemples fournis dans ce livre, nous vous engageons à entrer le programme MONITEUR, même si vous ne comprenez pas encore son fonctionnement; il vous aidera à la mise au point des exemples.

Le programme MONITEUR vous offre les facilités suivantes :
— examiner n'importe quel emplacement mémoire, et en modifier le contenu, si c'est un emplacement de la mémoire vive;
— examiner et modifier si nécessaire les registres du 68Ø3, y compris le pointeur de pile S;
— lancer l'exécution d'un segment de programme, avec un point d'arrêt à une adresse pré-déterminée.

Ce programme présente l'avantage d'être relogeable, il peut fonctionner à n'importe quelle adresse de la mémoire vive. En fait, il est en général pratique de l'installer à la fin de la mémoire vive disponible, c'est-à-dire en $ADØØ pour ALICE 9Ø, $4DØØ pour ALICE 32 sans extension mémoire, et $8DØØ pour ALICE 32 avec l'extension mémoire.

Les commandes du moniteur

M suivi d'une adresse en hexadécimal sur quatre chiffres (sans le signe dollar) permet de visualiser le contenu d'un emplacement. Une fois la valeur affichée, vous pouvez :
— modifier le contenu de l'emplacement en entrant la nouvelle valeur, en hexadécimal;
— visualiser l'emplacement mémoire suivant en appuyant sur +;
— visualiser l'emplacement mémoire précédent en appuyant sur −;
— arrêter cette commande en appuyant sur une autre touche, comme BREAK, par exemple.
Quand vous passez d'un emplacement mémoire à un autre sans le modifier, et quand vous arrêtez la commande, un signe ✻ signale que le contenu est inchangé.

Voici un exemple de l'utilisation de la commande M :

M4CØ : 1C : 2A+	modification + visualisation de l'adresse suivante
4AC1 : Ø3 : + ✻	visualisation de l'adresse suivante sans modification
4AC2 : 14 : Ø2 −	modification + visualisation de l'adresse précédente
4AC1 : Ø3 : − ✻	visualisation de l'adresse précédente sans modification
4ACØ : 2A : <BREAK> ✻	fin de la commande sans modification.

<ENTER> visualise le contenu de chaque registre du microprocesseur; sur la première ligne se trouve le nom du registre, au-dessus de sa valeur. Les registres A et B sont des registres sur 8 bits, qui sont réunis côte à côte pour donner la valeur du registre D. Les registres X (index), PC (valeur courante du compteur ordinal, c'est-à-dire l'adresse à laquelle est arrêté le programme) et SP (registre S, pointeur de la pile) sont sur 16 bits. Le registre contenant les drapeaux *(flags)*, appelé ici F, est détaillé bit par bit; en fait seuls six bits sont utilisés, les deux bits de poids fort valant généralement 1.

Exemple de l'effet de la commande <ENTER> :

```
HINZVC   A   B   X     PC    SP
Ø11ØØ1  1A  2Ø  C7B3  43ØD  3681
```

A, B et F suivis de 2 chiffres modifient le contenu des registres correspondants, puis visualisent à nouveau la totalité des registres.

X, P et S suivis de 4 chiffres agissent de même pour les registres sur 16 bits.

```
A4Ø
HINZVC   A   B   X     PC    SP
Ø11ØØ1  4Ø  2Ø  C7B3  43ØD  3681
F22
HINZVC   A   B   X     PC    SP
1ØØØ1Ø  4Ø  2Ø  C7B3  43ØD  3681
XB8C9
HINZVC   A   B   X     PC    SP
1ØØØ1Ø  4Ø  2Ø  B8C9  43ØD  3681
```

H suivi d'une adresse sur quatre chiffres permet de reprendre l'exécution du programme à partir de l'adresse contenue dans le registre PC, jusqu'à l'adresse spécifiée dans la commande. A cette adresse le programme rendra le contrôle au programme MONITEUR, qui affichera alors les registres, comme avec la fonction ENTER. Ainsi

```
H465E
HINZVC   A   B   X     PC    SP
111ØØ1  1A  43  C2FØ  465E  3688
```

fait exécuter le programme entre les instructions situées entre 43ØD et 465E. Attention, l'adresse d'arrivée doit être le premier octet, lorsque l'instruction est codée sur plusieurs.
C lance l'exécution du programme à partir de l'adresse courante, sans mettre de point d'arrêt.

Pour commencer à contrôler votre programme TOTO, implanté à l'adresse $5ØØØ (2Ø48Ø), il faut suivre la procédure suivante :
1. réserver la place nécessaire pour le programme et le MONITEUR, par CLEAR 1ØØ, 2Ø48Ø;

2. charger les deux programmes dans la mémoire d'ALICE :

```
CLOADM «MONITEUR», 3ØØØØ
CLOADM «TOTO»
```

(il faut préciser l'adresse d'implantation du MONITEUR, sinon il se charge à l'adresse ØØØØ, ce qui produit quelques dégâts!);

3. entrer dans le MONITEUR :

```
EXEC 3ØØØØ
```

(le MONITEUR vous souhaite alors la bienvenue);

4. indiquer au moniteur l'adresse d'exécution de votre programme en utilisant la fonction H :

```
H 5ØØØ
```

Ensuite peut commencer la chasse aux erreurs, en utilisant les fonctions vues précédemment.

Règles de programmation en assembleur

Étant donné la difficulté de comprendre un programme assembleur, à la syntaxe très éloignée du langage courant, même pour un anglicisté, il faut se plier à certaines règles.

— Découper le programme en modules cohérents. Tester un programme module par module est plus facile que pour un programme d'une seule pièce.

— Bien préciser, module par module, les paramètres d'entrée et de sortie, les registres modifiés, et, bien sûr, la fonction du module.

— Commenter le programme module par module plutôt qu'avec une ligne de commentaire à chaque instruction.

— Paramétrer autant que possible le programme : écrire
ETAT1 = $1 en début de programme, puis
LDAA #ETAT1
...
ORAB #ETAT1 dans le corps du programme, plutôt que
LDAA #$1
...
ORAB # $1.
Cela évitera de longues modifications le jour où ETAT1 ne vaudra plus 1, mais 2...

— Commencer le programme par un en-tête, comprenant le titre du programme, sa date de création ou de modification, un numéro de version si nécessaire. Rappeler aussi le but du programme, l'amnésie vient vite!

Particularités de l'éditeur-assembleur d'ALICE

La syntaxe de l'instruction CLEAR, nécessaire à la réservation de place pour le fichier source et le fichier objet n'est pas évidente; un coup d'œil à l'Annexe Carte mémoire d'ALICE permet de comprendre les effets de cette instruction.

Pour éviter des erreurs dues à l'oubli d'un signe dollar, comme l'éditeur-assembleur ne reconnaît que l'hexadécimal, il est conseillé de commencer le programme par des définitions des valeurs fréquentes comme :

```
Ø = $Ø
1 = $1
```

Souvenez-vous que la directive d'assemblage BLC réserve de la place en mémoire, mais qu'elle ne l'initialise pas : le contenu de la zone réservée doit être donné par programme.

2

Les entrées-sorties standard (clavier, écran & imprimante)

Ce chapitre propose un certain nombre d'astuces destinées à faciliter l'écriture de programmes de gestion d'écran et de clavier. Ces techniques n'utilisent pas pleinement les richesses graphiques d'ALICE, mais elles préparent à leur exposé systématique, auquel sont consacrés les chapitres 10, 11 et 12.

Écrire un caractère sur l'écran ou sur l'imprimante

Le livret d'accompagnement d'ALICE mentionne l'existence de quelques routines de sortie, après l'exposé sur les instructions du 68Ø3 :

POLCA	$F883	scrutation du clavier
OUTCA	$F9C6	émission d'un caractère
OUTTX	$E7A8	émission d'une chaîne de caractères
GRNCH	$FBD4	effacement de l'écran

Nous allons examiner plus en détail leur fonctionnement.

Les programmes d'écriture sur l'écran et sur l'imprimante sont communs : la sortie se fait sur l'écran ou sur l'imprimante suivant l'état d'un registre appelé DEVNU (numéro du périphérique), dont l'adresse est $E8.

— Si le contenu de DEVNU est Ø, l'affichage se fait sur l'écran.

— Si le contenu de DEVNU est différent de Ø (le BASIC écrit $FE, mais toute autre valeur a le même effet), l'affichage se fait sur l'imprimante.

L'affichage sur l'écran peut être effectué à l'endroit que l'on veut en utilisant les registres CURAD et CURAD + 1 (adresse du curseur $328Ø). CURAD contient le numéro de la rangée (numérotée à partir de zéro, du haut de l'écran vers le bas), et CURAD + 1 contient le numéro de la colonne (la colonne la plus à gauche est la colonne Ø). Écrire dans ces deux registres amènera donc le curseur à la position choisie.

Muni de ces renseignements, vous avez alors deux possibilités :
— écrire un caractère, à l'aide du programme OUTCA ($F9C6). Dans ce cas, il faut fournir à POLCA le code ASCII du caractère à afficher, dans le registre A du 68Ø3. Par exemple, le programme qui suit affiche le caractère 'E'sur la deuxième colonne de la cinquième rangée de l'écran :

```
DEVNU = $E8
CURAD = $328Ø
OUTCA = $F9C6
        CLR   DEVNU     ; SORTIE SUR ÉCRAN
        LDD   #$Ø4Ø1    ; RANGÉE 5  COLONNE 2
        STD   CURAD
        LDAA  #$45      ; CODE ASCII DE 'E'
        JSR   OUTCA
BOUCL   BRA   BOUCL
```

— écrire toute une chaîne de caractères : pour éviter d'écrire le texte caractère par caractère, utilisez le programme OUTTX ($E7A8). Il faut fournir à OUTTX dans le registre X du 68Ø3 l'adresse − 1 du premier caractère de la chaîne à afficher. La fin du texte doit être marquée par un octet mis à Ø. Par exemple, pour écrire un texte sur l'imprimante, vous pouvez utiliser le petit programme suivant :

```
DEVNU           =       $EB
OUTTX           =       $E7AB

        LDAA    #$1             ; SORTIE SUR IMPRIMANTE

        STAA    DEVNU
        LDX     #TEXTE − $1     ; ADRESSE − 1

        JSR     OUTTX
BOUCL   BRA     BOUCL
TEXTE   'CECI EST UN TEXTE
        DFD     $ØDØØ
```

On définit deux octets après les caractères de la chaîne à l'aide de DFD : ØD est le code ASCII du retour chariot, qui permet en l'occurrence d'envoyer à la ligne après l'affichage de la chaîne, mais ce retour chariot n'est pas obligatoire ; par contre l'octet suivant à ØØ est obligatoire pour signaler à OUTTX la fin de la chaîne.
Attention : le programme OUTTX est utilisé par le BASIC d'ALICE pour écrire des textes qui se trouvent entre guillemets (le signe ",

appelé par les anglo-saxons 'quotes'). Or OUTTX arrête l'impression lorsqu'il rencontre le code des guillemets dans la chaîne de caractères à afficher, ce qui a pour inconvénient de vous interdire l'usage du caractère " dans une chaîne.

L'effacement de l'écran

Pour effacer l'écran, inutile d'écrire des espaces un par un sur tout l'écran : appelez plutôt le programme GRNCH ($F8D4) qui est consacré à cette fonction. Notez l'astuce suivante : si vous voulez écrire sur tout l'écran un caractère graphique, mettez le code de ce caractère dans le registre B du 68Ø3, et appelez le programme en $F8D6. Souvenez-vous que les caractères graphiques sont codés entre $8Ø et $FF, soit 128 et 255 en décimal, comme c'est rappelé dans le manuel d'ALICE.

Choisir la vitesse de son imprimante

La vitesse de communication avec l'imprimante est normalement de 6ØØ bauds. Si vous avez le livret «ALLEZ PLUS LOIN AVEC ALICE», livré avec l'extension mémoire de 16 K-octets, vous pourrez y trouver les valeurs à écrire en $4224 (décimal 16932) pour obtenir des vitesses différentes. Malheureusement, ces vitesses ne sont valables que pour la première version d'ALICE, qui était contrôlée par une horloge dont le quartz était plus lent; d'autre part l'adresse des variables-système a changé.

Voici donc les valeurs correspondant aux nouvelles versions d'ALICE. Il faut les écrire à l'adresse $3223 (déc. 12835), sur deux octets.

Vitesse	Valeurs hexadécimales			Valeurs décimales		
	$3223	et	$3224	12835	et	12836
3ØØ bauds	$Ø1		ØF	1		15
6ØØ bauds	$ØØ		81	Ø		129
12ØØ bauds	$ØØ		41	Ø		65
24ØØ bauds	$ØØ		1E	Ø		3Ø
48ØØ bauds	$ØØ		ØC	Ø		12

N.B. N'oubliez pas que le 68Ø3 permet d'écrire sur deux octets adjacents grâce à l'instruction STD; l'affectation s'écrit en deux lignes d'assembleur :

```
LDD  #$Ø1ØF
STD  $3223
```

Écrire un nombre

Comme vous le verrez en détail dans le programme du chapitre suivant, il est assez compliqué d'écrire un nombre sur l'écran ou l'imprimante en assembleur. On est loin de l'instruction BASIC :

```
A = 128 + 6
PRINT A
```

En assembleur, il faut décomposer le nombre en chiffres que l'on va afficher un par un ensuite. Cette décomposition se fait par divisions successives (par les puissances de 1Ø si on affiche un nombre décimal).

Il existe heureusement un programme dans la mémoire morte d'ALICE dont la raison d'être est l'affichage des numéros de lignes des programmes BASIC. On peut bien entendu s'en servir aussi dans un programme assembleur. Le nom de cette routine est LIPRT, et elle se trouve à l'adresse $F419. Il faut lui fournir dans le registre D (c'est-à-dire les registres A et B concaténés) une valeur hexadécimale comprise entre $Ø et $FFFF. Sur l'écran s'affiche la valeur décimale correspondante (entre Ø et 65535).

Pour ce qui est des nombres en format flottant, c'est-à-dire des nombres comportant soit des virgules (1Ø,258 par exemple), soit des exposants (4E + 1Ø par exemple), cela devient très compliqué. La meilleure solution est sûrement de désassembler la ROM d'ALICE, pour trouver le programme adapté ; bon courage...

La gestion du clavier

Contrairement à certains autres micro-ordinateurs, ALICE ne gère pas son clavier grâce à des interruptions. Simplement, chaque fois qu'un programme a besoin d'une entrée au clavier, dans les instructions INPUT ou INKEY$ par exemple, le BASIC fait appel à un programme qui va scruter un port du 68Ø3 et en déduit la ou les touches enfoncées.

Nous allons voir deux méthodes pour gérer le clavier :
— Tout d'abord nous verrons une méthode utilisant les ressources du MONITEUR BASIC. Elle permet d'acquérir très facilement la touche enfoncée par l'utilisateur.
— La deuxième méthode est destinée aux jeux d'action où il est nécessaire de scruter très vite le clavier. Dans ce cas, on utilisera sans doute seulement quelques touches du clavier, mais on gérera entièrement le clavier par nous-mêmes.

Gérer le clavier grâce au moniteur BASIC

Pour interroger le clavier, en assembleur comme en BASIC, il existe deux possibilités : on peut attendre qu'une touche soit enfoncée, ou simplement scruter l'état du clavier.

— L'attente d'une touche est gérée par le programme KDINP, dont l'adresse est $F868. Ce programme se comporte un peu comme l'instruction INPUT du BASIC : un curseur clignote en attente de la frappe. Lorsqu'une touche a été enfoncée, le programme KDINP renvoie son code ASCII dans le registre A.

Attention : ce programme ne fait pas l'écho de la touche enfoncée sur le clavier à l'écran. Il faut, si on le désire, utiliser un programme pour le faire — par exemple OUTCA, comme dans le programme ci-dessous :

```
JSR     KDINP     ;  acquisition d'une
                     touche
JSR     OUTCA     ;  écho sur l'écran
```

Gérer le clavier pour les jeux d'action

— Un autre programme permet de tester fugitivement le clavier, comme l'instruction INKEY$ du BASIC. En assembleur, il faut utiliser le programme POLCA (adresse $F883). POLCA redonne immédiatement le contrôle au programme qui l'appelle. Si une touche est enfoncée, son code ASCII se trouve dans le registre A, et le bit Z du registre d'état est à Ø. Si aucune touche n'est enfoncée, le registre A est à Ø et le bit Z du registre d'état est à 1. On peut écrire le programme comme suit :

```
        JSR     POLCA
        BEQ     RIEN      ;  saut en RIEN si aucune touche
                          ;  n'est enfoncée
traitement de la touche
```

Nota : il existe deux exceptions; aucun de ces programmes ne vous fournira de code pour l'appui simultané sur les touches SHIFT et Ø, ainsi que pour les touches CONTROL et Ø. Ces deux combinaisons servent à mettre à jour des indicateurs qui influenceront ensuite les codes ASCII qui vous seront rendus : comme en BASIC, SHIFT+Ø fait passer du mode majuscules au mode minuscules et inversement, et CONTROL+Ø fait changer de couleur les caractères graphiques, et change donc le code des touches comme SHIFT+B, SHIFT+Q, etc.

La réaffectation des touches du clavier

Il est possible de changer l'affectation des touches du clavier, par exemple pour avoir un clavier QWERTY à la place d'un clavier AZERTY.

Aux adresses $3ØØ9 et $3ØØA se trouve l'adresse d'un tableau qui contient les adresses de quatre tables; la première table correspond aux touches normales, la seconde aux touches associées à CONTROL, la troisième à celles associées à SHIFT; la quatrième table répète les touches associées à CONTROL.

Dans le cas usuel, on trouve les valeurs suivantes :
contenu de

TABAD	($3ØØ9 – 3ØØA) = $D421
TABNOR	($D421 – D422) = $F986
TABCON	($D423 – D424) = $D838
TABSHF	($D425 – D426) = $D8ØØ
TABCON	($D427 – D428) = $D838

Le tableau qui suit donne la correspondance entre les touches du clavier et les codes trouvés dans les tables.

Touches	TABNOR	TABSHF	TABCON	Début de table +
–	$Ø	$Ø	$Ø	$ØØ
@	$4Ø	$13	$88	$Ø1
Q	$51	$8F	$Ø8	$Ø2
B	$42	$8Ø	$B3	$Ø3
C	$43	$82	$B2	$Ø4
D	$44	$87	$82	$Ø5
E	$45	$8D	$9B	$Ø6
F	$46	$86	$9Ø	$Ø7
G	$47	$85	$84	$Ø8
H	$48	$48	$A3	$Ø9
I	$49	$49	$8A	$ØA
J	$4A	$4A	$81	$ØB
K	$4B	$4B	$9E	$ØC
L	$4C	$4C	$BC	$ØD
/	$2F	$3F	$BA	$ØE
N	$4E	$4E	$B9	$ØF
O	$4F	$4F	$A5	$1Ø
P	$5Ø	$5Ø	$C7	$11
A	$41	$89	$15	$12
R	$52	$8C	$9C	$13
S	$53	$88	$Ø9	$14
T	$54	$8B	$8C	$15
U	$55	$8F	$8Ø	$16
V	$56	$81	$B5	$17
Z	$5A	$84	$5E	$18
X	$58	$83	$B1	$19
Y	$59	$8A	$8F	$1A
W	$57	$8E	$ØA	$1B
—	$Ø	$Ø	$Ø	$1C
—	$Ø	$Ø	$Ø	$1D
—	$Ø	$Ø	$Ø	$1E
ENTER	$ØD	$ØD	$ØD	$1F
espace	$2Ø	$Ø6	$Ø7	$2Ø
Ø	$3Ø	$3Ø	$3Ø	$21

Touches	TABNOR	TABSHF	TABCON	Début de table +
1	$31	$21	$8E	$22
2	$32	$22	$93	$23
3	$33	$23	$98	$24
4	$34	$24	$97	$25
5	$35	$25	$96	$26
6	$36	$26	$94	$27
7	$37	$27	$95	$28
8	$38	$28	$9D	$29
9	$39	$29	$86	$2A
:	$3A	$2A	$89	$2B
M	$4D	$4D	$92	$2C
'	$2C	$3C	$BB	$2D
—	$2D	$3D	$91	$2E
.	$2E	$3E	$B7	$2F
;	$3B	$2B	$B6	$3Ø
—	$Ø	$Ø	$Ø	$31
—	$Ø	$Ø	$Ø	$32
BREAK	$Ø3	$Ø3	$Ø3	$33
flèche ←	$Ø8	$Ø8	$Ø8	$34
flèche →	$ØA	$ØA	$ØA	$35
flèche ↓	$Ø9	$Ø9	$Ø9	$36
flèche ↑	$ØB	$ØB	$ØB	$37

Ces tables se trouvent dans la mémoire morte d'ALICE. Par conséquent, pour modifier l'affectation des touches, il vous faut les redéfinir entièrement dans une table implantée en mémoire vive, et donner l'adresse de cette table en $3ØØ9-$3ØØA.

Gestion du clavier par le port d'entrée d'ALICE

Si la méthode que nous venons de voir pour gérer le clavier est trop lente pour un jeu d'action, et si quelques touches suffisent (les flèches par exemple), il est avantageux d'utiliser directement le port d'entrées-sorties d'ALICE. Voici ce qu'il faut en connaître : la gestion du clavier utilise les deux ports du 68Ø3, ainsi qu'une adresse réservée aux entrées-sorties ($BFFF). Les deux ports d'entrées-sorties du 68Ø3 sont programmés en sortie pour le port 1 et en entrée pour le port 2 à l'initialisation. Le clavier se lit par colonne : il y a huit colonnes. Pour lire le contenu d'une colonne, il faut écrire dans le port 1, dont l'adresse est $Ø2, un mot dont on met un bit à Ø : le bit Ø correspond à la colonne Ø, le bit 1 à la colonne 1 et ainsi de suite.

Le contenu de la colonne se trouve alors dans $BFFF. Un bit à Ø indique que la touche correspondante est enfoncée. Voici la table de correspondance entre les bits et les touches :

	Colonne Ø	1	2	3	4	5	6	7	Code de l'octet
bit Ø	@	Q	B	C	D	E	F	G	FE
bit 1	H	I	J	K	L	/?	N	O	FD
bit 2	P	A	R	S	T	U	V	Z	FB
bit 3	—	Y	W	—	—	—	ENTER	espace	F7
bit 4	Ø	1	2	3	4	5	6	7	EF
bit 5	8	9	:	M	,<	-=	.>	+;	DF
bit 6	CONTROL	X	BREAK	←	→	↓	↑	SHIFT	BF
bit 7	—	—	—	—	—	—	—	—	

Le port 2 (port d'entrée) est utilisé pour les touches CONTROL et SHIFT. Pour la touche SHIFT, il faut avoir dans le port 1 ($Ø2) la valeur $7F, puis lire le port 2 (adresse $Ø3). Le bit 1 du port 2 est à Ø si la touche SHIFT est enfoncée.
Pour la touche CONTROL, il faut écrire dans le port 1 la valeur $FE, puis lire le port 2. Le bit 1 du port 2 est à Ø si la touche CONTROL est enfoncée.

Ces renseignements peuvent être intéressants pour des jeux rapides : quelques instructions du type STAA $Ø2 et LDAA $Ø3 suffisent pour lire l'état de ces touches. Mais cette rapidité n'est pas possible si vous voulez scruter tout le clavier.

Attention : si vous gérez vous-même le clavier, il est prudent de relire pour savoir si la touche est toujours enfoncée quelques millisecondes après une première lecture, ceci pour éviter les rebonds après l'appui sur une touche. On peut utiliser une boucle contrôlée par le registre X à cet effet, en décrémentant X de $ 1ØØØ à Ø par exemple, avant de recommencer la lecture.

3

Premier programme : la fonction RENUM

Nous vous proposons, dans ce chapitre, un programme qui complète le BASIC d'ALICE. En effet, le BASIC d'ALICE ne permet pas de renuméroter les lignes d'un programme en mémoire. Lors de la mise au point d'un programme, il vous est sûrement arrivé de vous trouver à court de numéros de lignes ; faute d'une fonction de renumérotation, vous êtes dans l'obligation de retaper une partie du programme pour intercaler les lignes supplémentaires.

Grâce à l'assembleur contenu dans ALICE, nous allons donc écrire cette fonction qui lui faisait défaut jusqu'ici. Ce programme ne se contente pas de renuméroter les lignes. Il recalcule aussi les paramètres suivant les instructions GOTO, GOSUB, ON...GOTO, ON...GOSUB et THEN, comme il se doit pour que la renumérotation soit effective, mais il vérifie aussi les branchements du programme BASIC : si un GOTO n'est pas suivi d'un numéro de ligne correspondant à une ligne existante, l'erreur sera signalée, et la ligne erronée sera dénoncée.

Attention : ce programme est assez long (environ 62Ø lignes de programme source) et il nécessite l'utilisation de l'extension mémoire de 16 K-octets pour un ALICE 32. Mais il est possible de le réduire en supprimant les commentaires. Le programme objet (environ 12ØØ octets plus un tableau qui exige quatre octets pour chaque ligne de BASIC) laisse de la place, même pour un ALICE 32 sans extension mémoire.

Structure d'un programme BASIC en mémoire

Avant d'entamer l'explication du programme proprement dit, il est bon de voir comment est rangé un programme BASIC dans la mémoire d'ALICE.

— L'adresse du début du programme BASIC se trouve dans le mot TXTAB, à l'adresse $93-$94. Cette adresse est fixée par le moniteur BASIC, et elle est toujours égale à $3346 (= 13126).

— L'adresse de la fin du programme BASIC est en fait l'adresse de la première variable utilisée par le programme BASIC. Elle se trouve dans le mot VARTA, à l'adresse $95-$96.

— Chaque ligne de programme se compose des éléments suivants :
— deux octets qui donnent l'adresse de la ligne BASIC suivante. Ces deux octets valent Ø pour la dernière ligne.
— deux octets qui forment le numéro de ligne en hexadécimal. Par exemple, la ligne 158 sera codée $ØØ et $9E.
— la ligne de programme proprement dite. Les mots clés du BASIC sont codés sur un octet, tout le reste étant représenté par la traduction en ASCII. Par exemple, GOTO 1ØØØ devient

$81	$2Ø	$31, $3Ø, $3Ø, $3Ø
code de GOTO	espace	1ØØØ en ASCII

— la fin de la ligne est marquée par un octet à $ØØ.

— La fin du programme est marquée par deux octets à $ØØ.

L'exemple ci-dessus (l'instruction GOTO 1ØØØ) met en évidence l'une des difficultés que doit régler le programme : les numéros de lignes sont codés différemment en début de ligne et dans les instructions du type GOTO. Il faut convertir les nombres d'un code à l'autre.

Exemple pratique avec le programme MONITEUR

Nous allons examiner le contenu de la mémoire avec le programme MONITEUR. Vous l'avez entré grâce à l'éditeur, puis assemblé et sauvegardé sous forme d'un fichier objet sur une cassette. Pour l'utiliser, entrez les instructions suivantes (réservation de la place pour le programme) :

```
CLEAR 1Ø, 4 * 16 ↑ 3 + 13 * 16 ↑ 2
CLOADM "MONITEUR", 4 * 16 ↑ 3 + 13 * 16 ↑ 2
```

Mettez en route le magnétophone ; lorsque le fichier est chargé, arrêtez le magnétophone.

Pour cet exemple, nous allons examiner le programme suivant :

```
1Ø A = 1
2Ø PRINT A
3Ø A = A + 1 : GOTO 2Ø
```

Tapez donc ce programme, puis lancez l'exécution du MONITEUR en tapant :

```
EXEC 4 * 16 ↑ 3 + 13 * 16 ↑ 2
```

Après les présentations d'usage, le MONITEUR est prêt à prendre en compte vos commandes. Pour l'instant, nous n'en utiliserons qu'une, la visualisation de mémoire, c'est-à-dire la commande M.
Examinons le contenu de TXTAB :

```
MØØ93 : 33 :
```

Cela confirme ce que nous avons vu plus haut ; pour examiner le contenu de $ØØ94, il suffit de faire +. Comme le contenu de $ØØ93 n'a pas été modifié, un astérisque s'affiche.

```
MØØ93 : 33 : + *
 ØØ94 : 46 : + *
 ØØ95 : 33 : + *
 ØØ96 : 68 : *
```

Un appui sur BREAK a demandé l'arrêt de la commande M.
Le programme BASIC réside donc entre l'adresse $3346 et l'adresse $3367. Examinons le :

```
M3346 : 33 : + *          poids fort de l'adresse de la ligne suivante
  3347 : 4E : + *         poids faible de l'adresse de la ligne suivante
  3348 : ØØ : + *  1Ø     poids fort du numéro de la ligne
  3349 : ØA : + *         poids faible du numéro de la ligne
  334A : 41 : + * A       code ASCII de 'A'
  334B : AF : + =         code de la fonction =
  334C : 31 : + 1         code ASCII de '1'
  334D : ØØ : +           fin de la ligne BASIC

  334E : 33 : + *         adresse de la ligne suivante
  334F : 56 : + *
  335Ø : ØØ : + *  2Ø     numéro de la ligne sur deux octets
  3351 : 14 : + *
  3352 : 86 : + * PRINT  code de la fonction PRINT
  3353 : 2Ø : + *         espace
  3354 : 41 : + * A       code ASCII de 'A'
  3355 : ØØ : + *         fin de la ligne

  3356 : 33 : + *
  3357 : 66 : + *
  3358 : ØØ : + *  3Ø     numéro de la ligne sur deux octets
  3359 : 1E : + *
  335A : 41 : + * A       code ASCII de 'A'
  335B : AF : + * =       code de la fonction =
```

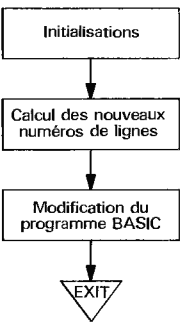
```
335C : 41 : + * A        code ASCII de 'A'
335D : A7 : + * +        code de la fonction +
335E : 31 : + * 1        code ASCII de '1'
335F : 2Ø : + *          espace
336Ø : 3A : + * :        code ASCII de ' :'
3361 : 81 : + * GOTO     code de la fonction GOTO
3362 : 2Ø : + *          espace
3363 : 32 : + * 2        code ASCII de '2'
3364 : 3Ø : + * Ø        code ASCII de 'Ø'
3365 : ØØ : + *          fin de la ligne
3366 : ØØ : + *
3367 : ØØ : + *          fin du programme
```

Après avoir interrompu la visualisation du programme par BREAK, vous pouvez le modifier; par exemple, si vous faites
M3349 : ØA : 11 (BREAK)

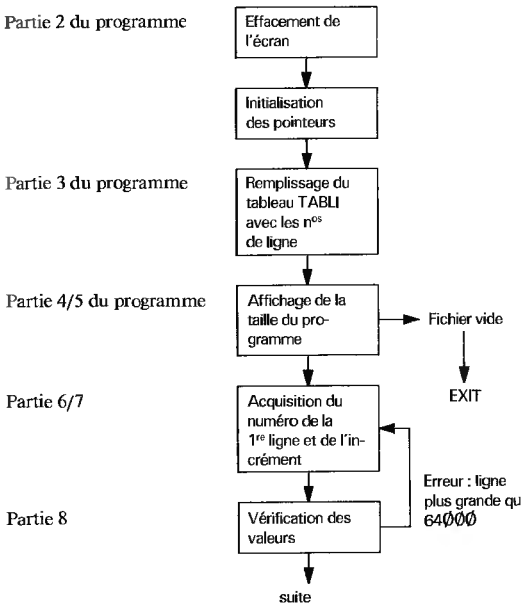
la première ligne devient la ligne 17 (= $11). Faites RESET puis LIST pour le vérifier.

Organigramme du programme RENUM

Nous allons découper le programme en modules assez importants que nous détaillerons ensuite :



Initialisations :

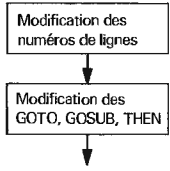


Le tableau TABLI comprend quatre octets par ligne BASIC. Les deux premiers octets (que nous remplissons ici), contiennent l'ancien numéro de la ligne. Les deux suivants contiendront le nouveau numéro de la ligne.

Calcul des nouveaux numéros de lignes :

C'est la partie 9 du programme. La deuxième partie du tableau TABLI est remplie grâce à PRLIG (numéro de la nouvelle première ligne) et à INCRM (valeur de l'incrément). Le nombre de lignes du programme BASIC se trouve dans NBLIG.

Modification du programme BASIC :



La partie de modification des numéros de lignes est simple, il suffit de prendre la deuxième partie du tableau pour la mettre dans le programme.

Par contre, la seconde partie est plus compliquée :

— il faut d'abord retrouver les GOTO, GOSUB et THEN dans les lignes BASIC; c'est le rôle dévolu au programme ANLIG en 12.6;

— puis il faut retrouver le numéro de la ligne qui suit le GOTO (ou le GOSUB ou le THEN). C'est le rôle de TRTGO en 12.8;

— ensuite, il faut vérifier que cette ligne existait bien dans le programme BASIC avec les anciens numéros de lignes. Sinon, on affiche un texte d'erreur (programme SNBLI en 12.11);

— puis l'on remplace l'ancien numéro de ligne par le nouveau. Ce n'est pas toujours simple : si l'on remplace un GOTO 1Ø par un GOTO 1ØØ. il faut agrandir le fichier et en décaler la fin. L'inverse est aussi possible. C'est le programme ASREP en 12.9 qui fait cela, avec l'aide du programme DECAL en 12.12.

Listing commenté du programme :

Ce programme, tel que vous le trouvez ici, est écrit pour un ALICE 32 avec l'extension mémoire de 16 K-octets. Pour l'adapter aux autres versions, il suffit de changer la valeur suivant la directive ORG :

— pour ALICE 32, mettre ORG $44ØØ

— pour ALICE 90, mettre ORG $AØØØ

Ces valeurs sont aussi fonction de la taille du programme BASIC (le tableau TABLI se trouve à la fin du programme. Si il n'a pas assez de place, il risque de déborder en dehors de la mémoire d'ALICE et de détruire le programme BASIC.

```
;-----------------------------------+
;     Programme RENUM               !
;     --------------                !
;  Ce programme permet de renumeroter !
;   automatiquement les lignes du pro- !
;   gramme BASIC qui se trouve en me - !
;   moire.Il modifie les instructions  !
;   SOTO et GOSUB .                    !
;-----------------------------------+
;     Auteur : J-F Gallet           !
;     Date : 23 Novembre 1984       !
;-----------------------------------+

        ORG   $8000
        EXC   RENUM

;1.    CONSTANTES UTILISEES :
;--
0       = $0
1       = $1
2       = $2
4       = $4
5       = $5

TXTAB = $93         ;DEBUT DU PROGRAMME BAGIC
VARTA = $95         ;FIN OU PROGRAMME BASIC
OUTTX = $E7A8       ;AFFICHAGE D'UN TEXTE
KDINP = $F86e       ;ACQUISITION D'UN CARACTERE
OUTCA = $F9C6       ;AFFICHAGE D'UN CARACTERE
GRNCH = $FBD4       ;EFFACEMENT DE L'ECRAN

;2.    Initialisation des variables :
;--
RENUM
        JSR   GRNCH ;EFFACEMENT DE L'ECRAN
        LDX   #TITRE-1
        JSR   OUTTX ;AFFICHAGE DU TITRE
        LDX   #0
        STX   NBLIG         ;INIT DU NOMBRE DE LIGNES A 0
        LDX   #TABLI
        STX   ATDLI
        STX   ATCLI         ;INIT DES POINTEURS
        LDX   TXTAB
        STX   ADFIC         ;ADRESSE DANS LE FICHIER

;3.    Remplissage de la 1ere partie du tableau :
;--          (anciens numeros de lignes)
REM1P
        LDD   0,X
        BEQ   FICVI         ;FIN DU FICHIER (POINTEUR = 0)
        LDD   2,X
        LDX   ATCLI
        STD   0,X           ;STOCKAGE DU NUMERO DE LIGNE COURANT
        LDAB  #4
        ABX
        STX   ATCLI         ;ARTICLE SUIVANT -> TABLEAU DE LIGNES
        LDX   NBLIG
        INX
        STX   NBLIG         ;NOMBRE DE LIGNES + 1
        LDX   ADFIC
        LDX   0,X           ;LIGNE DE PROGRAMME SUIVANTE
        STX   ADFIC
        BRA   REM1P
```

```
64
65 ;4.   Le fichier BASIC est-il vide ?
66 ;--
67 FICVI
68       LDX   ATCLI
69       STX   ATFLI
70       LDX   NBLIG
71       BNE   FICOK        ;IL Y A UN FICHIER BASIC EN MEMOIRE
72       LDX   #TEFVD-1
73       JSR   OUTTX        ;AFFICHAGE DU TEXTE D'ERREUR
74       RTS
75
76 ;5.   Affichage du nombre de lignes du fichier BASIC :
77 ;--
78 FICOK
79       LDX   #TNBL1-1
80       JSR   OUTTX        ;AFFICHAGE DE LA 1ERE PARTIE DU TEXTE
81       LDX   NBLIG
82       JSR   AFFNB        ;AFFICHAGE DU NOMBRE DE LIGNES
83       LDX   #TNBL2-1
84       JSR   OUTTX        ;AFFICHAGE DE LA FIN DU TEXTE
85
86 ;6.   Acquisition du numero de la 1ere ligne :
87 ;--
88 Q1LIG
89       LDX   #T1LIG-1
90       JSR   ACREP        ;ACQUIGITION D'UNE REPONSE NUMERIQUE
91       STX   PRLIG
92       CPX   #0
93       BEQ   Q1LIG        ;LIGNE 0 INTERDITE
94       CPX   #$FA00
95       BHS   Q1LIG        ;LIGNE SUPERIEURE A 64000 INTERDITE
96
97 ;7.   Acquisition de la valeur du pas d'incrementation :
98 ;--
99 QINCR
100      LDX   #TINCR-1
101      JSR   ACREP        ;ACQUISITION D'UNE REPONSE NUMERIQUE
102      STX   INCRM
103      CPX   #$100
104      BHS   QINCR        ;LIMITEE A 255
105
106 ;8.   Verification des numeros de lignes :
107 ;--
108      LDX   PRLIS
109      LDD   NBLIG        ;BOUCLE SUR TOUTES LES LIGNES
110 VRNLI
111      SUBD  #$1
112      BEQ   REM2P
113      PSHB
114      LDAB  INCRM+1
115      ABX
116      PULB
117      CPX   #$FA00
118      BHS   Q1LIG        ;AUCUNE LIGNE SUPERIEURE A 64000
119      BRA   VRNLI
120
121 ;9.   Remplissage de la 2eme partie du tableau :
122 ;--        (nouveaux numeros de lignes)
123 REM2P
124      LDX   ATDLI
125      STX   ATCLI        ;POINTEUR COURANT AU DEBUT DU TABLEAU
126      LDX   NBLIS        ;NOMBRE DE LIGNES
127      LDD   PRLIG        ;NUMERO DE LA 1ERE LIGNE
128 REM21
```

```
129         PSHX
130         LDX   ATCLI
131         STD   2,X           ;NOUVEAU NUMERO DE LIBNE
132         ADDD  INCRM
133         INX
134         INX
135         INX
136         INX                 ;+4 = LONGUEUR D'UN ARTICLE DU TABLEAU
137         STX   ATCLI         ;NOUVEAU POINTEUR COURANT
138         PULX
139         DEX
140         BNE   REM21         ;BOUCLE SI PAS LA DERNIERE LIGNE
141
142 ;10.  Change les numeros de lignes dans le programme BASIC :
143 ;---
144         LDX   TXTAB
145         STX   ADFIC         ;ADRESSE DU DEBUT DU PROGRAMME BAGIC
146         LDX   ATDLI
147         BTX   ATCLI         ;POINTEUR DANS LE TABLEAU DES LIGNES
148 CHLIG
149         LDX   ATCLI
150         LDD   2,X
151         INX
152         INX
153         INX
154         INX
155         STX   ATCLI         ;NOUVEAU POINTEUR -) TABLEAU DE LIGNES
156         LDX   ADFIC
157         LDX   0,X
158         BEQ   RGOTO         ;FIN DU FICHIER BASIC
159         LDX   ADFIC
160         STD   2,X           ;NOUVEAU NUMERO DE LIGNE -) FICHIER
161         LDX   0,X
162         STX   ADFIC         ;NOUVEAU POINTEUR -) FICHIER BASIC
163         BRA   CHLIS
164
165 ;11.  A la recherche des GOTO et GOSUB :
166 ;---
167 RGOTO
168         LDX   TXTAB
169 RG010
170         LDD   0,X
171         BEQ   RNFIN         ;FIN DU PROGRAMME BASIC
172         JSR   ANLIG         ;ANALYSE DE LA LIGNE BASIC COURANTE
173         LDX   0,X           ;LIGNE SUIVANTE
174         BRA   RG010
175 RNFIN
176         LDX   #TXFIN-1
177         JGR   OUTTX         ;ECRITURE DU TEXTE DE FIN
178         RTS
179
180 ;12.  Sous-programmes :
181 ;---
182
183  ;12.1      Affichage d'un nombre :
184  ;ENTREE : X = NOMBRE A AFFICHER SUR L'ECRAN
185  ;SORTIE : NEANT
186 AFFNB
187         STX   NBASC
188         BSR   CBNAS         ;CONVERSION ASCII
189         LDX   #ASBUF
190         LDAB  #5            ;NOMBRE DE CHIFFRES DANS LE NOMBRE
191 AFF10                       ;ELIMINE LES ZEROS NON SIGNIFICATIFS
192         LDAA  0,X
193         CMPA  #$30          ;0 ASCII
```

```
194         BNE   AFF20         ;CHIFFRE SIGNIFICATIF
195         DECB
196         INX
197         BRA   AFF10
198 AFF20
199         LDAA  0,X
200         JSR   OUTCA         ;AFFICHE LES CHIFFRES 1 PAR 1
201         INX
202         DECB
203         BNE   AFF20
204         RTS
205
206   ;12.2       Conversion hexadecimal -) ASCII :
207   ;ENTREE : NBASC = NOMBRE A CONVERTIR
208   ;SORTIE : NOMBRE EN ASCII SUR 5 CHIFFRES DANS ASBUF
209 CBNAS
210         CLRB                ;INIT DU NOMBRE DE CHIFFRES
211         LDX   #ASBUF        ;INIT ADRESSE DANS LE BUFFER
212 CBN10
213         PSHB
214         PSHX
215         BSR   DVDIX         ;DIVISION PAR 10 PUISSANCE (5-B)
216         PULX
217         PULB
218         ADDA  #$30          ;POUR CONVERSION ASCII
219         STAA  0,X           ;STOCKAGE DANS LE BUFFER ASBUF
220         INX                 ;NOUVEAU POINTEUR DANS ASBUF
221         INCB
222         CMPB  #5
223         BNE   CBN10
224         RTS
225
226   ;12.3       Division par les puissances de 10 :
227   ;ENTREE : NOMBRE A DIVISER DANS NBASC
228   ;           B = 5 - PUISSANCE DE 10
229   ;             SI B=0 ON DIVISE PAR 10 PUISSANCE 5 ...
230   ;SORTIE : RESTE DANS NBASC
231   ;           A = QUOTIENT
232 DVDIX
233         LDX   #TAB10        ;TABLE DES PUISSANCES DE 10
234         ABX
235         ABX
236         LDX   0,X           ;VALEUR DE 10 PUISSANCE 5-B
237         STX   PUI10
238         CLR   RESUL         ;INITIALISATION DU QUOTIENT
239         LDD   NBASC         ;NOMBRE A DIVISER
240 DVD10
241         SUBD  PUI10
242         BLO   DVD20         ;DIVISION TERMINEE
243         INC   RESUL         ;INCREMENTATION DU QUOTIENT
244         STD   NBASC
245         BRA   DVD10
246 DVD20
247         LDAA  RESUL         ;PARAMETRE DE SORTIE DANS A
248         RTS
249
250   ;12.4       Acquisition d'une reponse numerique :
251   ;ENTREE : X = ADRESSE DU TEXTE DE LA QUESTION
252   ;SORTIE : X = NOMBRE ENTRE PAR L'OPERATEUR
253 ACREP
254         PSHX                ;SAUVEGARDE EN CAS D'ERREUR
255 ACR10
256         PULX                ;RECUPERATION DE LA QUEGTION
257         PSHX
258         JSR   OUTTX         ;AFFICHAGE DE LA QUESTION
```

```
259         LDX   #0
260         STX   NBASC         ;INIT DE LA VALEUR
261 ACR20
262         JSR   KDINP         ;ACQUISITION D'UN DIGIT
263         JSR   OUTCA
264         CMPA  #$0D
265         BEQ   ACR30         ;EXIT SI ENTER
266         SUBA  #$30          ;CARACTERE ENTRE 0 ET 9
267         BLO   ACR10         ;ERREUR : ON REPOSE LA QUESTION
268         CMPA  #$A
269         BHS   ACR10         ;ERREUR : ON REPOSE LA QUESTION
270         PSHA
271         BSR   MLDIX         ;MULTIPLIE PAR 10 LA VALEUR COURANTE
272         LDX   NBASC
273         PULB
274         ABX                 ;PLUS LE DERNIER CHIFFRE ENTRE
275         STX   NBASC         ;= NOUVEAU RESULTAT
276         BRA   ACR20
277 ACR30
278         PULX                ;A CAUSE DU PSHX D'ENTREE
279         LDX   NBASC         ;PARAMETRE DE SORTIE
280         RTS
281
282  ;12.5      Multiplication par 10 du contenu de NBASC :
283  ;ENTREE : NOMBRE A MULTIPLIER DANS NBASC
284  ;SORTIE : RESULTAT DANS NBASC
285 MLDIX
286         LDD   NBASC
287         LDX   #$09          ;ON BOUCLE 10 FOIS
288 MLD10
289         ADDD  NBASC
290         DEX
291         BNE   MLD10
292         STD   NBASC         ;NOUVEAU RESULTAT
293         RTS
294
295  ;12.6      Analyse d'une ligne :
296  ;ENTREE : X = ADRESSE DU DEBUT DE LA LIGNE BASIC
297  ;SORTIE : X = IDEM
298 ANLIB
299         STX   ADFIC
300         LDAB  #4            ;1ER OCTET DE LA LIGNE
301         ABX
302 ANL00
303         LDAA  0,X
304         BEQ   ANFIN         ;FIN DE LA LIGNE
305         CMPA  #$83          ;REM
306         BEQ   ANFIN
307         CMPA  #$81          ;GOTO
308         BEQ   ANL10
309         CMPA  #$82          ;GOSUB
310         BEQ   ANL10
311         CMPA  #$A3          ;THEN
312         BNE   ANL20
313 ANL10                       ;TRAITEMENT DE GOTO ET DE GOSUB
314         BSR   TRTGO
315         BRA   ANL30         ;SUITE DE LA LIGNE
316 ANL20
317         CMPA  #$22          ;QUOTES
318         BNE   ANL30
319         BSR   TRTTX
320 ANL30
321         INX
322         BRA   ANL00
323 ANFIN                       ;FIN DE LA LIGNE COURANTE
```

```
324         LDX   ADFIC
325         LDAB  #4
326         ABX
327 ANF10                       ;CALCUL DU DEBUT LIGNE SUIVANTE
328         LDAA  0,X
329         BEQ   ANF20         ;0 INDIQUE LA FIN DE LA LIGNE BASIC
330         INX
331         BRA   ANF10
332 ANF20
333         INX
334         STX   RESUL
335         LDD   RESUL
336         LDX   ADFIC
337         STD   0,X           ;ADRESSE DE LA PROCHAINE LIGNE BASIC
338         RTS
339
340  ;12.7      Texte : pas de traitement :
341  ;ENTREE : X = ADRESSE DU DEBUT DE LA CHAINE DE CARACTERES
342  ;SORTIE : X = ADRESSE DE LA FIN DE LA CHAINE DE CARACTERES
343 TRTTX
344         INX
345         LDAA  0,X
346         BEQ   TTX10         ;FIN DE LIGNE
347         CMPA  #$22          ;FIN DU TEXTE
348         BNE   TRTTX
349         BRA   TTX20
350 TTX10
351         DEX
352 TTX20
353         RTS
354
355  ;12.8      Traitement de GOTO et GOSUB :
356  ;ENTREE : X = ADRESSE DANS LE FICHIER BAGIC
357  ;                (DEBUT DE L'INSTRUCTION)
358  ;SORTIE : X = ADRESSE DANS LE FICHIER BASIC
359  ;                (APRES L'INSTRUCTION)
360 TRTGO
361         INX
362         LDAA  0,X
363         BEQ   TGO10         ;FIN DE LA LIGNE
364         CMPA  #$20          ;ELIMINE LES ESPACES
365         BEQ   TRTGO
366         CMPA  #$2C          ; LES VIRGULES
367         BEQ   TRTGO
368         CMPA  #$3A          ;LE PROGRAMME S'ARRETE SUR 2 POINTS
369         BHS   TGO10
370         CMPA  #$30          ; ENTRE 0 ET 9
371         BHS   TGO20         ;PROCHAIN CARACTERE
372 TGO10                       ;EXIT
373         DEX
374         RTS
375 TGO20
376         BSR   AGREP         ;REMPLACEMENT D'UNE CHAINE ASCII
377         BRA   TRTGO
378
379  ;12.9      Remplacement d'une chaine ASCII :
380  ;ENTREE : X = POINTEUR SUR LA CHAINE A REMPLACER
381  ;SORTIE : X = POINTEUR APRES LA NOUVELLE CHAINE
382 ASREP
383         STX   ADSTR         ;ADRESSE DE LA CHAINE
384         BSR   CASBN         ;CONVERGION ASCII -) BINAIRE
385         BSR   SNBLI         ;CALCUL DU NOUVEAU NO DE LIGNE
386         LDX   NBASC
387         BEQ   ASRFI         ;ERREUR : NUMERO DE LIGNE INEXIGTANT
388         JGR   CBNAS         ;CONVERSION EN ASCII
```

```
389
390                         ;CALCUL DE LA LONGUEUR DE LA CHAINE :
391         LDAA   #5
392         STAA   NBCR2
393         LDX    #ASBUF
394 ASR10
395         LDAA   0,X
396         INX
397         CMPA   #$30         ;ELIMINE LES ZEROS
398         BNE    ASR20
399         DEC    NBCR2
400         BRA    ASR10
401 ASR20
402         JSR    DECAL        ;DECALE LA FIN DU FICHIER
403
404                         ;INSERE LE NOUVEAU NUMERO DE LIGNE :
405         LDX    #ASBUF
406         LDAB   #5
407         SUBB   NBCR2
408         ABX                 ;ADRESSE DU 1ER CARACTERE
409         LDAB   NBCR2
410 ASR30
411         LDAA   0,X
412         PSHX
413         LDX    ADSTR
414         STAA   0,X
415         INX
416         STX    ADSTR
417         PULX
418         INX
419         DECB
420         BNE    ASR30
421 ASRFI
422         LDX    ADSTR        ;POINTEUR SUR LE CARACTERE SUIVANT
423         DEX
424         RTS
425
426  ;12.10     Conversion ASCII -) hexadecimal :
427  ;ENTREE : ADSTR : ADRESSE DU DEBUT DE LA CHAINE ASCII
428  ;SORTIE : NBBIN : VALEUR EN HEXADECIMAL SUR 2 OCTETS
429  ;         NBCR1 : LONGUEUR DE LA CHAINE ASCII A TRADUIRE
430 CASBN
431         CLR    NBCR1        ;NOMBRE DE CARACTERES
432         LDX    #0
433         STX    NBASC        ;INIT DE LA VALEUR HEXA
434         LDX    ADSTR        ;ADRESSE DE LA CHAINE
435 CAS10
436         LDAA   0,X
437         SUBA   #$30
438         BLO    CASFI        ;CE N'EST PAS UN CHIFFRE : EXIT
439         CMPA   #$0A
440         BHS    CASFI        ;CE N'EST PAS UN CHIFFRE : EXIT
441         INC    NBCR1
442         PSHX
443         PSHA
444         JSR    MLDIX        ;MULTIPLICATION PAR 10
445         LDX    NBASC
446         PULB
447         ABX
448         STX    NBASC        ;NOUVELLE VALEUR HEXA
449         PULX
450         INX
451         BRA    CAS10
452 CASFI
453         LDX    NBASC
```

```
454         STX   NSBIN         ;ZONE DE STOCKAGE
455         RTS
456
457   ;12.11      Calcul du nouveau numero de ligne :
458   ;ENTREE : NBBIN : ANCIEN NUMERO DE LIGNE
459   ;SORTIE : NBASC : NOUVEAU NUMERO DE LIGNE
460   ;         SI NBASC = 0 : LA LIGNE N'EXISTE PAS
461 SNBLI
462         LDX   ATDLI
463 SNB10
464         PSHX
465         LDX   0,X
466         CPX   NBBIN         ;NUMERO RECHERCHE
467         BEQ   SNB20         ; ET TROUVE
468         PULX
469         LDAB  #4
470         ABX
471         CPX   ATFLI         ;FIN DE LA TABLE ?
472         BHS   SNB30         ; ET OUI ...
473         BRA   SNB10
474 SNB20
475         PULX
476         LDX   2,X           ;NOUVEAU NUMERO DE LIGNE
477         STX   NBASC
478         RTS
479 SNB30                       ;LIGNE INEXISTANTE
480         LDX   #TENL1-1
481         JSR   OUTTX         ;AFFICHE LE DEBUT DU TEXTE D'ERREUR
482         LDX   ADFIC
483         LDX   2,X
484         JSR   AFFNB         ;AFFICHAGE NO DE LA LIGNE ERRONEE
485         LDX   #TENL2-1
486         JSR   OUTTX         ;AFFICHE LA 2EME PARTIE DU TEXTE
487         LDX   NBBIN         ;LE COUPABLE
488         JSR   AFFNB         ;SUR L'ECRAN
489         LDX   #TENL3-1
490         JSR   OUTTX         ;AFFICHAGE DE LA FIN DU TEXTE D'ERREUR
491         LDX   #0            ;FLAG D'ERREUR
492         STX   NBASC
493         LDX   ADSTR
494         LDAB  NBCR1         ;CALCUL DU NOUVEAU POINTEUR
495         ABX
496         STX   ADSTR
497         RTS
498
499   ;12.12      Decalage de la fin du fichier :
500   ;ENTREE : NBCR1 : LONGUEUR DE L'ANCIEN NUMERO DE LIGNE
501   ;         NBCR2 : LONGUEUR DU NOUVEAU NUMERO DE LIGNE
502   ;SORTIE : VARTA : NOUVELLE ADRESSE DE LA FIN DU FICHIER BASIC
503 DECAL
504         LDAB  NBCR2
505         SUBB  NBCR1         ;B = NOMBRE D'OCTETS A DECALER
506         BEQ   DECFI         ;PAS DE DECALAGE
507         BHS   DEC30         ;AGRANDISSEMENT DU FICHIER
508
509                   ;DECALAGE : FICHIER PLUS PETIT :
510         NEGB
511         LDX   ADSTR
512 DEC10
513         PSHX
514         ABX
515         CPX   VARTA
516         BHS   DEC20         ;TERMINE
517         LDAA  0,X
518         PULX
```

```
519         STAA  0,X
520         INX
521         BRA   DEC10
522 DEC20
523         PULX
524         STX   VARTA         ;FIN DU FICHIER BASIC
525         BRA   DECFI
526
527               ;DECALAGE : FICHIER PLUS GRAND :
528 DEC30
529         LDX   VARTA
530 DEC40
531         CPX   ADSTR         ;TERMINE
532         BLS   DEC50
533         LDAA  0,X
534         PSHX
535         ABX
536         STAA  0,X
537         PULX
538         DEX
539         BRA   DEC40
540 DEC50
541         LDX   VARTA
542         ABX
543         STX   VARTA         ;FIN DU FICHIER
544
545               ;EXIT
546 DECFI
547         RTS
548
549 ;13.  Donnees du programme :
550 ;--
551
552 NBLIG BLC 2          ;LIGNE COURANTE
553 ATDLI BLC 2          ;ADRESSE DE DEBUT DU TABLEAU DE LIGNE
554 ATCLI BLC 2          ;ADRESSE COURANTE DANS LE TABLEAU DE LIGNE
555 ATFLI BLC 2          ;ADRESSE DE FIN DU TABLEAU DE LIGNE
556 ADFIC BLC 2          ;ADRESSE DANS LE FICHIER BASIC
557 ASBUF BLC 5          ;BUFFER POUR UN NOMBRE ASCII
558 NBASC BLC 2          ;NOMBRE A CONVERTIR EN ASCII
559 NBBIN BLC 2          ;NOMBRE CONVERTI DE L'ASCII
560 PRLIG BLC 2          ;NUMERO DE LA NOUVELLE 1ERE LIGNE
561 INCRM BLC 2          ;INCREMENT
562 PUI10 BLC 2          ;PUISSANCE DE 10 COURANTE
563 RESUL BLC 2          ;VARIABLE INTERMEDIAIRE
564 ADSTR BLC 2          ;ADRESSE DE LA CHAINE DE CARACTERES
565 NBCR1 BLC 2          ;LONGUEUR DE L'ANCIEN NUMERO DE LIGNE
566 NBCR2 BLC 2          ;   "          NOUVEAU      "
567
568  ;TABLEAU DES PUISSANCES DE 10 :
569
570 TAB10
571         DFD   $2710         ;10000
572         DFD   $03E8         ;1000
573         DFD   $64           ;100
574         DFD   $A            ;10
575         DFD   1             ;1
576
577  ;TEXTEG :
578
579 TITRE ' RRRR  EEEEE N   N U   U M    M
580         DFD   $0D
581         ' R   R E     NN  N U   U M M MM M
582         DFD   $0D
583         ' RRRR  EEE   N N N U   U M    M
```

```
584         DFD   $0D
585         ' R  R  E     N  NN U   U M    M
586         DFD   $0D
587         ' R   R EEEEE N   N  UUU  M    M
588         DFD   $0D0D
589         DFD   0
590
591 TEFVD '*PAS DE FICHIER BASIC EN MEMOIRE !!!
592         DFD   $0D00
593
594 TNBL1 'IL Y A :
595         DFD   $0D00
596 TNBL2 ' LIGNES DANS VOTRE PROGRAMME BASIC .
597         DFD   $0D00
598
599 TILIG DFD   $0D
600         'NUMERO DE LA PREMIERE LIGNE ?
601         DFD   $2000
602 TINCR DFD   $0D
603         'VALEUR DE L'INCREMENT ?
604         DFD   $2000
605
606 TENL1 'ERREUR DANS LA LIGNE
607         DFD   $2000
608 TENL2 ' :
609         DFD   $0D
610         'LA LIGNE
611         DFD   $2000
612 TENL3 ' N'EXISTAIT PAS !!!
613         DFD   $0D00
614
615 TXFIN DFD   $0D
616         'RENUMEROTATION TERMINEE ...
617         DFD   $0D00
618
619  ;TABLEAU DES NUMEROS DE LIBNES :
620 TABLI                          ;TOUTE LA FIN DE LA MEMOIRE
621
622 ;14   Fin
623 ;--
```

4

Les entrées-sorties cassettes

Le BASIC d'ALICE vous permet de sauvegarder des programmes BASIC par la commande CSAVE, et des tableaux numériques par la commande CSAVE*. L'éditeur-assembleur vous permet de sauver des fichiers de texte ainsi que des fichiers en langage machine, qui sont ensuite chargés grâce à la commande CLOADM du BASIC. Ce chapitre va donner des outils pour sauvegarder vous-même une zone mémoire, puis la relire. Vous pourrez même créer votre propre type de fichier pour des applications spécifiques (traitement de texte ou tableur, si vous en avez le courage).

Sauvegarde d'une zone mémoire

Il est très facile de sauvegarder une zone mémoire avec les nouvelles versions d'ALICE (ALICE 32 ou ALICE 9Ø). En effet, ces versions ont été enrichies de programmes qui permettent de sauvegarder très facilement des zones mémoires sur cassettes.

Attention : le programme CSAVEM qui est mentionné dans le guide «ALLER PLUS LOIN AVEC ALICE» ne fonctionne pas avec les nouvelles versions.

Le programme de sauvegarde peut être appelé aussi bien d'un programme BASIC que d'un programme assembleur. Mais il faut tout d'abord fournir au programme un certain nombre de valeurs, qui se répartissent en deux catégories :
— les paramètres nécessaires à l'enregistrement, par exemple nom, adresses de début et de fin...
— les paramètres utilisés lors de la lecture et de l'utilisation : adresse de chargement, et éventuellement d'exécution.

Voici le détail des valeurs à fournir :

NBNAM adresse $3ØFØ = décimal 12528
NBNAM contient la longueur du nom du fichier à enregistrer. Le maximum est 8.

VARFI adresse $3ØF1 = décimal 12529
VARFI est une table de huit mots qui contient le nom du fichier en ASCII. Si ce nom fait moins de huit caractères, il faut compléter avec des espaces ($2Ø ASCII).

FILTY adresse $3ØF9 = décimal 12537
FILTY contient le type du fichier. Certains types sont déjà définis. Si votre enregistrement ne suit pas leurs règles (qui seront expliquées en fin de chapitre), il faut utiliser votre propre type.

ASCFL adresse $3ØFA = décimal 12538
ASCFL indique s'il s'agit d'un fichier en ASCII ; l'utilité de ce drapeau reste à démontrer : il est toujours à zéro.

GAPFL adresse $3ØFB = décimal 12539
GAPFL indique s'il s'agit d'un fichier contigu (en un seul bloc) ou non. On le met en général à 1.

ADSTR adresse $3ØFC = décimal 1254Ø
ADSTR donne sur deux octets l'adresse d'exécution lorsqu'il s'agit d'un programme en langage machine. Ce paramètre s'ajoute au paramètre éventuellement fourni après la commande CLOADM utilisée pour le chargement du programme. Si ADSTR n'est pas mis à zéro, on peut se passer du paramètre d'EXEC, l'adresse d'exécution est fournie lors de la lecture sur cassette.

ADLOA adresse $3ØFE = décimal 12542
ADLOA donne sur deux octets l'adresse à laquelle il faudra charger le programme lors de la lecture. Pour un fichier en langage machine par exemple, ce sera l'adresse où il faudra écrire le premier octet lu sur la cassette. Ce paramètre s'ajoute au paramètre éventuellement fourni après CLOADM.

ADDEB adresse $31ØØ = décimal 12544
ADDEB donne sur deux octets l'adresse du premier octet à enregistrer.

ADFIN adresse = $31Ø2 = décimal 12546
ADFIN donne sur deux octets l'adresse du dernier octet à enregistrer.

Le programme d'écriture s'appelle K7WRT ; il se trouve à l'adresse $DFAØ (décimal 57248).

Lecture d'un fichier sur cassette

Pour lire un programme BASIC, il existe bien entendu la fonction CLOAD (ainsi que CLOAD* pour un tableau numérique). Pour un programme en langage machine, vous pouvez utiliser la fonction CLOADM. Sa syntaxe est la même que pour les autres variantes, c'est-à-dire :
CLOADM "TEST" pour charger le fichier TEST.
Il existe toutefois une possibilité supplémentaire : nous venons de voir que, lors de l'enregistrement d'un fichier en langage machine, on doit fournir une adresse de chargement. C'est cette adresse qui est utilisée par CLOADM.

Mais supposons que vous ayez donné comme adresse de chargement 16ØØØ et que vous vouliez mettre votre programme en 17ØØØ. Il suffira alors d'écrire :
CLOADM "TEST", 1ØØØ
Ce second paramètre fournit donc un *offset*, un déplacement, pour le chargement du programme. Ce déplacement s'applique aussi à l'adresse d'exécution : si, à l'enregistrement, vous avez demandé une exécution en 165ØØ, notre exemple fera que l'exécution débutera en 175ØØ.

Nota : cette facilité est utilisée par le programme MONITEUR. Lors de l'enregistrement, on précise que l'adresse de chargement et celle d'exécution sont Ø (c'est le rôle des directives d'assemblage ORG $Ø et EXC INIT). Il faut donc impérativement donner l'adresse de chargement lors de la lecture. Ce programme est dit relogeable car on peut le mettre n'importe où en mémoire (dans un endroit acceptable pour ALICE : si vous ne précisez pas l'adresse, le programme s'écrira à l'adresse Ø. La suite est imprévisible...).

En assembleur, il existe deux possibilités : vous pouvez utiliser les mêmes paramètres que pour l'écriture. Appelez dans ce cas le programme K7REA, en $DFBC.
Vous pouvez utiliser une autre possibilité, plus fine : nous la détaillerons dans le chapitre qui suit.

Mais auparavant, il faut voir comment est enregistré un fichier sur la cassette : bien entendu, c'est une suite d'informations binaires, de Ø et de 1, arrangées octet par octet.

Pour les enregistrer, on les fait précéder d'un en-tête pour permettre la synchronisation du micro-ordinateur (128 octets égaux à $55), puis d'un espace blanc d'environ Ø,5 seconde, puis d'un second en-tête (128 octets à $55), puis des blocs contenant les données proprement dites, et enfin d'un bloc de fin.

Le format général d'un bloc est le suivant :
— un octet à $55
— un octet de synchronisation ($3C)
— un octet qui indique le type du bloc =
  — $ØØ = bloc qui contient le nom du fichier
  — $Ø1 = bloc de données
  — $FF = bloc de fin
— un octet qui donne la longueur des données (de $ØØ à $FF)
Cet octet vaut $ØF pour un bloc "nom" et $ØØ pour un bloc fin.
— les données (jusqu'à 255 octets dans un bloc);
— un octet de vérification. Il est égal à la somme des octets de données, plus l'octet de type et l'octet de longueur;
— un octet de fin ($55).

Le bloc "nom" est un peu particulier : à la place des données, il contient les quinze octets suivants :
— 8 octets pour le nom du programme.
— 1 octet pour le type de fichier. Voici les types qui sont définis actuellement :
  — $ØØ = BASIC
  — $Ø1 = tableau numérique BASIC
  — $Ø2 = langage machine
  — $Ø5 = fichier texte de l'éditeur
— 2 octets de type : ASCII = $FF binaire = $ØØ
  contigu = $Ø1 segmenté = $FF
— 2 octets pour l'adresse de démarrage de l'exécution, lorsque le type est langage machine.
— 2 octets pour l'adresse de chargement.

Les programmes que nous allons voir exigent que les variables suivantes soient actualisées :

— FICNM	$3257	nom du fichier
— FICTY	$3267	type du fichier
— BLKTY	$3275	type du bloc
— K7ACC	$3278	adresse des données.

De plus, l'adresse $3272 (REALL) doit être à zéro pour effectuer réellement la lecture. Sinon les données ne sont pas chargées, comme dans la commande SKIPF du BASIC. Voici donc les deux programmes utilisables :
— CMPNA ($FE37) lit les en-têtes de fichier qui se trouvent sur la cassette. On ne se sert de CMPNA que dans deux cas :
  — lorsque le fichier est trouvé; dans ce cas le bit Z du registre d'état est mis à 1 ;
  — lorsqu'une erreur se produit; dans ce cas le bit Z est à Ø.
— K7LEC lit ensuite les données et les écrit en mémoire, bloc par bloc. De même, en cas d'erreur, le bit Z est à Ø. Il faut alors contrô-

ler la fin du fichier pour vérifier si la lecture est achevée (BLKTY est alors à $FF). Le registre X contient alors l'adresse du dernier octet écrit en mémoire.

Format du fichier texte de l'éditeur

Voici le format du fichier texte tel qu'il est construit par l'éditeur d'ALICE. Vous pouvez l'utiliser pour faire votre propre langage, ou pour toute autre application.

Un fichier texte est divisé en rangées. Chaque rangée correspond effectivement à une rangée sur l'écran. Comme la taille de l'écran est limitée à 80 caractères par rangée, la rangée du fichier contiendra au maximum 80 caractères. Au maximum, car pour gagner de la place les caractères espace de fin de ligne sont éliminés.

L'indication de la fin de la rangée est inscrite avant les caractères qui la composent, sous forme d'un octet notant le compte effectif de caractères. Ce nombre varie de Ø (rangée vide) à 8Ø (rangée pleine).

La fin de fichier est indiquée par un code 255 ($FF) inscrit à la place du nombre de caractères.

Ainsi, si vous avez un texte de trois lignes, dont la deuxième est vide, comme celui-ci :

CECI EST UN TEXTE — 1re ligne
— 2e ligne vide
— 3e ligne

FIN

cela donnera :

```
 $11  $43,$45,$43,$49,  $2Ø,  $45,$53,$54,  $2Ø,  $55,$4E, $2Ø,  $54
longueurC   E   C   I  espace E   S   T espace U   N espace T
$45, $58, $54, $45,  $ØØ      $Ø3    $46, $49, $4E,   $FF
 E   X   T   E  longueur longueur  F   I    N  fin du texte
```

L'adresse du début du fichier est égale à la valeur du deuxième paramètre de l'instruction de réservation de la place CLEAR x, ad. Par exemple, pour CLEAR 6Ø,15ØØØ, votre fichier texte commence en 15ØØØ ($3A98). Il est enregistré sur cassette par blocs de 255 octets, sauf le dernier bloc, qui peut éventuellement en faire moins. Le type inscrit en FILTY ($3ØF9) a pour code 5.

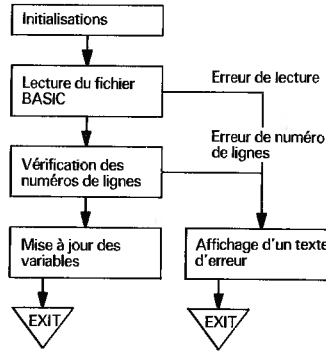
5

Programme : la fonction MERGE

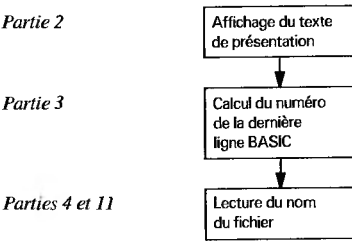
Nous allons encore combler une lacune du BASIC d'ALICE : ce chapitre donne un programme permettant la fusion de deux fichiers BASIC. Le programme que nous allons écrire lit un programme BASIC sur la cassette et l'ajoute après le programme BASIC déjà installé en mémoire. En outre, ce programme vérifie que les numéros de lignes sont bien consécutifs.

Ce programme utilise des notions que nous avons déjà vues dans les deux programmes qui précèdent : la structure d'un programme BASIC, l'utilisation du clavier et de l'écran, et les possibilités de lecture sur cassette.

Organigramme :



Initialisations :



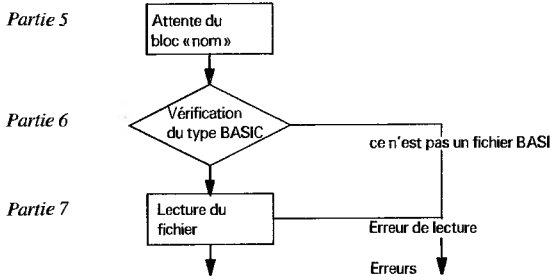
Le calcul du numéro de la dernière ligne BASIC utilise la structure que nous avons vue dans le chapitre 3. De plus, nous conservons au passage l'adresse de la fin du programme BASIC. Ce sera l'adresse d'implantation du programme que nous allons ajouter.

Le programme de lecture du nom du fichier utilise les programmes KDINP et OUTCA. De plus, il reconnaît l'utilisation de la touche BREAK, qui provoque la répétition de la question.

En outre, ce programme commence par initialiser la zone de travail avec des espaces. Ainsi, s'il y a moins de 8 caractères entrés au clavier pour le nom du fichier, ils seront automatiquement complétés par des espaces.

Inversement, à partir du huitième caractère, le curseur arrête sa progression et tout nouveau caractère entré efface le précédent.

Lecture du programme BASIC :



Dans la partie 5, le programme CMPNA ne redonne le contrôle que lorsqu'il a trouvé un fichier dont le nom correspond à celui qui est demandé, ou lorsqu'il détecte une erreur. En particulier, CMPNA signale une erreur lorsque la lecture commence au milieu d'un autre fichier. C'est pourquoi, dans ce cas-là, on le fait boucler sur lui-même, jusqu'à ce qu'il arrive au début du fichier qui suit.

La partie 7 utilise les programmes que nous avons vus dans le chapitre qui précède. Il s'arrête lorsqu'il trouve un bloc fin (type de bloc = $FF).

La vérification des numéros de lignes est très simple. Le numéro de la première ligne que nous venons de lire se trouve aisément, car nous avons l'adresse de cette première ligne. Le numéro de ligne se trouve dans les troisième et quatrième octets.

Pour la réinitialisation des variables, nous utilisons le programme BASIC qui est appelé après chaque modification, insertion ou destruction de ligne BASIC. Il recalcule tous les liens entre les instructions BASIC.

Enfin, les programmes d'erreur se contentent d'afficher un texte (en français) correspondant à l'erreur rencontrée.

Attention : il faut adapter la directive ORG à la taille mémoire dont vous disposez. Ici, elle est calculée pour un ALICE 32 avec l'extension mémoire. Mettez ORG $4DØØ pour ALICE 32 et ORG $ADØØ pour ALICE 9Ø.

```
 1 ;-------------------------------------+
 2 ;     Programme MERGE                 !
 3 ;     --------------                  !
 4 ;  Ce programme permet la fusion de 2 !
 5 ;   programmes ecrits en BASIC .      !
 6 ;  ATTENTION : les numeros des lignes !
 7 ;   doivent etre consecutifs , sinon  !
 8 ;   le programme n'est pas charge .   !
 9 ;-------------------------------------+
10 ;     Auteur : J-F Gallet             !
11 ;     Date : 23 Novembre 1984         !
12 ;-------------------------------------+
13
14       ORG   $8000
15       EXC   MERGE
16
17 ;1.   Constantes utilisees :
18 ;---
19
20 TXTAB = $93          ;DEBUT DU PROGRAMME BASIC
21 VARTA = $95          ;FIN DU PROGRAMME BASIC
22 FICNM = $3257        ;ADRESSE DU NOM DU FICHIER
23 FICTY = $3267        ;TYPE DU FICHIER
24 REALL = $3273        ;INDIQUE QUE LA LECTURE DOIT ETRE FAITE
25 BLKTY = $3275        ;TYPE DU BLOC
```

```
26 K7ALC = $3278      ;ADRESSE DES DONNEES A LIRE
27 CURAD = $3280      ;ADRESSE DU CURSEUR
28 CHEAD = $E2F3      ;CALCUL DES LIENS DES LIGNES BASIC
29 OUTTX = $E7A8      ;AFFICHAGE D'UN TEXTE
30 KDINP = $F868      ;ACQUISITION D'UN CARACTERE
31 OUTCA = $F9C6      ;AFFICHAGE D'UN CARACTERE
32 GRNCH = $FBD4      ;EFFACEMENT DE L'ECRAN
33 CMPNA = $FE37      ;LECTURE DU BLOC TYPE ET NOM DU FICHIER
34 K7LEC = $FE89      ;PROGRAMME DE LECTURE CASSETTE
35
36 ;2.   Initialisations :
37 ;--
38 MERGE
39        JSR   GRNCH
40        LDX   #TXMER        ;TEXTE DE PRESENTATION
41        JSR   OUTTX
42
43 ;3.   Calcul du numero de la derniere ligne :
44 ;--
45        LDX   TXTAB
46        LDD   $0,X
47        BNE   PRLGN         ;AU MOINS UNE LIGNE DANS LE PROGRAMME
48        STD   NDRNL
49        STX   ADLEC         ;ADRESSE DE LECTURE DU PROGRAMME
50        BRA   LECNM
51 PRLGN
52        LDD   $2,X          ;POUR LE CAS D'UN PROGRAMME
53        STD   NDRNL         ; D'UNE SEULE LIGNE
54        LDX   $0,X
55        STX   ADLEC
56 CDRNL
57        LDD   $2,X
58        LDX   $0,X          ;LIGNE SUIVANTE
59        BEQ   LECNM         ;DERNIERE LIGNE ?
60        STD   NDRNL
61        STX   ADLEC
62        BRA   CDRNL
63
64 ;4.   Lecture du nom du fichier a lire :
65 ;--
66 LECNM
67        BSR   TITRE
68
69 ;5.   Lecture du bloc 'nom de fichier' :
70 ;--
71 LCBNM
72        JSR   CMPNA
73        BNE   LCBNM         ;ERREUR PENDANT LA LECTURE DU TITRE
74
75 ;6.   Verification du type du fichier :
76 ;--
77        LDAA  FICTY
78        BNE   ERTYP         ;CE N'EST PAS UN FICHIER BASIC
79
80 ;7.   Lecture du fichier
81 ;--
82        LDX   ADLEC
83        STX   K7ALC
84 LECFI
85        LDAA  #$FF
86        CLR   REALL         ;LECTURE REE..LE
87        CLRB
88        STX   K7ALC
```

```
 89          JSR   K7LEC
 90          BNE   ERLEC       ;ERREUR EN LECTURE
 91          LDAA  BLKTY       ;TYPE DU BLOC
 92          CMPA  #$FF
 93          BNE   LECFI       ;CE N'EST PAS LA FIN DU FICHIER
 94          PSHX              ;ADRESSE DE LA FIN DU FICHIER
 95
 96 ;8.     VERIFICATION DU NUMERO DE LIGNE :
 97 ;--
 98          LDX   ADLEC
 99          LDX   $2,X        ;NOUVELLE PREMIERE LIGNE
100          CPX   NDRNL       ; ET L'ANCIENNE DERNIERE LIGNE
101          BLS   ERLIG       ;ERREUR
102          PULX
103          STX   VARTA
104
105 ;9.     Reinitialisation des variables :
106 ;--
107          JSR   CHEAD       ;CALCUL DES LIENS DU BASIC
108          LDX   #TXFIN-$1
109          JSR   OUTTX
110          RTS
111
112 ;10.    Erreurs :
113 ;--
114 ERTYP
115          LDX   #TXTYP-$1
116          BRA   EREUR
117 ERLEC
118          LDX   #TXLEC-$1
119          BRA   EREUR
120 ERLIG
121          PULX
122          LDX   #TXLIG-$1
123 EREUR
124          JSR   OUTTX       ;AFFICHAGE DU TEXTE D'ERREUR
125          CLRA
126          CLRB
127          LDX   VARTA
128          DEX
129          DEX
130          STD   $0,X        ;FIN DU FICHIER DEJA EN MEMOIRE
131          STD   $2,X
132          RTS
133
134 ;11.    Lecture du nom du fichier a lire :
135 ;--
136 ;ENTREE : NEANT
137 ;SORTIE : FICNM = NOM DU FICHIER A LIRE (8 OCTETS)
138 TITRE
139          LDX   #TXTIT-$1
140          JSR   OUTTX
141          LDX   #FICNM
142          LDAA  #$20
143          LDAB  #$8
144 TIT10
145          STAA  $0,X
146          INX
147          DECB
148          BNE   TIT10
149          LDX   #FICNM
150 TIT20
151          JSR   KDINP
```

```
152         JSR   OUTCA
153         CMPA  ##3           ;BREAK
154         BEQ   TITRE
155         CMPA  ##$D
156         BEQ   TIT40
157         STAA  $0,X
158         CPX   #FICNM+$8
159         BEQ   TIT30
160         INX
161         BRA   TIT20
162 TIT30
163         LDAB  CURAD+$1
164         DECB
165         STAB  CURAD+$1
166         BRA   TIT20
167 TIT40
168         RTS
169
170 ;12.  Donnees du programme :
171 ;--
172     ;Textes :
173 TXMER DFC   $0D
174         'PROGRAMME DE FUSION DE PROGRAMMES BASIC
175         DFD   $0D00
176
177 TXTIT DFC   $0D
178         'NOM DU FICHIER A LIRE ?
179         DFD   $2000
180
181 TXFIN DFC   $0D
182         'LECTURE TERMINEE
183         DFD   $0D00
184
185 TXTYP DFC   $0D
186         'ERREUR : CE N'EST PAS UN FICHIER BASIC
187         DFD   $0D00
188
189 TXLEC DFC   $0D
190         'ERREUR DURANT LA LECTURE DU FICHIER
191         DFD   $0D00
192
193 TXLIG DFC   $0D
194         'ERREUR : NUMEROS DE LIGNE NON CONSECUTIFS
195         DFD   $0D00
196
197  ;Variables :
198 NDRNL BLC $2       ;NUMERO DE LA DERNIERE LIGNE
199
200 ADLEC BLC $2       ;ADRESSE DE LA DERNIERE LIGNE
201
202 ;13.  Fin
```

La technique de l'overlay

Cette méthode employée sur des ordinateurs plus puissants serait très utile pour des micro-ordinateurs dont la taille mémoire est (relativement) limitée. Malheureusement, peu de moniteurs BASIC proposent cette formule.

De quoi s'agit-il? Lorsqu'un programme BASIC est trop grand pour la mémoire du micro-ordinateur, la seule solution qui existe est de le raccourcir en enlevant des fonctions, ou des commentaires. Une autre solution serait de couper le programme en fonctions indépendantes les unes des autres et de charger la fonction au moment où l'on doit s'en servir. Par exemple, un programme de statistiques serait partagé en deux fonctions : une fonction d'acquisition et une fonction de calcul et d'affichage.

Dans le cadre d'ALICE, cela pourrait se faire en partageant la mémoire utilisateur en quatre parties :

1re partie : programme RÉSIDENT
2e partie : fonctions TEMPORAIRES
3e partie : données BASIC
4e partie : programmes langage machine

Première partie : le programme RÉSIDENT

Ce programme BASIC est chargé constamment dans la mémoire. C'est lui qui gère le passage d'une fonction à une autre. Pour cela, il appelle le programme en langage machine qui se trouve dans la quatrième partie. De plus, ce programme peut contenir des sous-programmes utilisés par plusieurs fonctions parmi celles qui se trouvent dans la deuxième partie.

Deuxième partie : les fonctions temporaires

Ces fonctions sont chargées au fur et à mesure des besoins du programme résident. Chacune de ces fonctions doit être indépendante des autres (une fonction ne peut pas appeler un sous-programme qui se trouve dans une autre fonction). Par contre, toutes les fonctions partagent les mêmes données, qui se trouvent dans la troisième partie.

Troisième partie : les données BASIC

Pour que les données BASIC soient partagées par toutes les fonctions et la partie résidente, il faut que cette troisième partie ait une adresse fixe. Cette adresse est calculée en additionnant la longueur du programme résident et celle de la fonction qui occupe le plus de place.
VARTA (adresse $95) est le pointeur sur la première donnée BASIC. C'est ce pointeur que vous devez calculer une fois pour toutes avant de charger la première fonction.
Attention aussi aux variables alphanumériques : lorsque vous déclarez une variable de la manière suivante :

```
A$="TEST",
```

le pointeur du BASIC n'utilise pas la zone réservée aux variables alphanumériques (qui se trouve juste avant la zone réservée aux programmes en langage machine, comme c'est indiqué dans l'Annexe 2). Ce pointeur se contente d'indiquer l'adresse de la variable dans le programme lui-même. Par conséquent, si cette déclaration se fait dans une fonction temporaire, la variable A$ sera perdue lorsque vous chargerez une autre fonction à sa place.
Pour forcer le BASIC à sauver la variable dans l'espace réservé aux variables alphanumériques, il faut exécuter une fonction sur cette variable, par exemple A$="TEST" + " ".

Quatrième partie : les programmes en langage machine

Cette quatrième partie contient tous les programmes en langage machine, et, en particulier, un programme du type du programme MERGE qui vous permettra de charger les fonctions les unes après les autres (après avoir demandé à l'utilisateur de mettre en route le magnétophone à cassettes. Vivement les disquettes!).

6

Les interruptions

Les notions d'interruption et de gestion des interruptions sont pratiquement inconnues des programmeurs familiers du seul BASIC. Elles sont pourtant extrêmement importantes dans la programmation d'un micro-ordinateur.

Voyons tout d'abord ce qu'est une interruption. En général, une interruption est un signal qui avertit le microprocesseur qu'un périphérique est en attente d'un traitement. Le microprocesseur est programmé de manière à décider si le périphérique est prioritaire par rapport aux tâches qui sont déjà en cours ; si c'est le cas, le microprocesseur suspend temporairement l'exécution de ces tâches pour gérer le périphérique, et lorsque ce traitement est terminé, il reprend les tâches suspendues à l'endroit où il les a abandonnées.

Pour être efficace, un programme de gestion d'interruption doit être court (sans quoi les tâches normales du microprocesseur ne s'effectueront jamais). Par exemple, pour la gestion d'une liaison série, le programme d'interruption doit se contenter de stocker l'octet reçu, le traitement et l'affichage étant pris en charge par le programme principal. De plus, un programme d'interruption trop long pourrait perturber la réception du caractère suivant.

Enfin, le traitement d'interruption ne doit pas perturber le fonctionnement des tâches en cours de traitement. Par exemple, il doit sauvegarder les registres pour les restituer à la fin du programme d'interruption. De même, la pile en cours d'utilisation ne doit pas être détruite.

Les interruptions du MC68Ø3

Nous allons d'abord examiner le traitement que fait subir le MC68Ø3 à une interruption qui apparaît. Le microprocesseur commence par

vérifier qu'il peut autoriser le traitement de cette interruption ; l'autorisation est donnée si deux conditions sont remplies :
— certaines interruptions sont masquables, c'est-à-dire que le programme principal peut interdire le traitement de ces interruptions. Pour cela, il faut utiliser le bit i du registre d'état du MC68Ø3. Lorsque ce bit est à 1, ces interruptions sont masquées ; lorsqu'il est à zéro, elles sont autorisées.
— si l'autorisation est donnée, le MC68Ø3 vérifie aussi qu'une interruption plus prioritaire n'est pas en cours de traitement. Dans ce cas, le traitement de la nouvelle interruption est différé jusqu'à ce que la première soit entièrement traitée.
Lorsque ces deux conditions sont satisfaites, le MC68Ø3 sauvegarde son état sur la pile opérationnelle. Il sauvegarde successivement :
— le compteur ordinal (PC), c'est-à-dire l'adresse de l'instruction en cours,
— les registres X, A et B,
— le registre d'état.
De plus, le bit i du registre d'état est mis à 1, pour masquer toute nouvelle interruption tant que celle-ci n'est pas complètement traitée.
Ensuite, suivant le type de l'interruption, le microprocesseur va chercher dans une table qui se trouve à la fin de la mémoire l'adresse du programme d'interruption.
Le programme dont l'adresse a été trouvée grâce à cette table s'exécute alors. Il doit, en principe, se terminer par l'instruction RTI (retour d'interruption), qui restaure le contexte du microprocesseur avant l'interruption : celui-ci reprend sur la pile les valeurs du registre d'état et des registres A, B et X, et l'exécution du programme principal se poursuit à partir de l'endroit où l'interruption s'était produite.

Voici la liste des interruptions qui existent dans le 68Ø3. Elles sont données selon leur ordre de priorité (la première étant la plus prioritaire). L'adresse qui se trouve en face du nom de l'interruption est l'adresse qui contient l'adresse du programme d'interruption. (Cette indirection permet au programmeur de dérouter vers son propre programme de gestion d'interruption certaines d'entre elles.) Nous verrons plus loin le traitement qu'effectue ALICE pour chacune des interruptions.

Interruption		Adresse du vecteur
RESET	=Reset	$FFFE-$FFFF
NMI	=Non Maskable Interrupt Interruption non masquable	$FFFC-$FFFD
SWI	=Software Interrupt Interruption par programme	$FFFA-$FFFB

IRQI	=Maskable Interrupt on Request 1 Interruption masquable sur demande 1	$FFF8-$FFF9
ICF	=Input Capture Interrupt Flag Interruption de saisie d'entrée	$FFF6-$FFF7
OCF	=Output Compare Interrupt Flag Interruption de comparaison de sortie	$FFF4-$FFF5
TOF	=Timer Overflow Interrupt Interruption de dépassement d'horloge	$FFF2-$FFF3
SCI	=Serial Communication Interrupt Interruption sur la sortie série	$FFFØ-$FFF1

Les interruptions ICF, OCF, TOF et SCI sont regroupées sous l'étiquette commune d'IRQ2. Elles sont provoquées par des équipements internes du MC68Ø3, l'horloge et l'interface de communication série.

Seules les interruptions IRQ1 et IRQ2 sont masquables, c'est-à-dire que toutes les autres s'exécutent, sauf si une interruption prioritaire est déjà en cours de traitement.

RESET :

L'interruption RESET est très spéciale. Elle est provoquée soit par la mise sous tension de votre ALICE (bouton marche-arrêt), soit par l'appui sur le bouton de RESET qui se trouve à l'arrière du boîtier. Le vecteur d'interruption de RESET contient l'adresse $F72E. A cette adresse commence le programme général d'initialisation d'ALICE. Il efface la page d'écran, initialise les ports d'entrées-sorties et recalcule certains pointeurs utilisés par le BASIC. Ce programme peut prendre deux directions suivant le contenu de l'adresse $EA : si cet octet contient $55, le système n'est pas entièrement réinitialisé, et le contenu de la mémoire vive n'est pas détruit, ce qui sauvegarde vos programmes BASIC, vos données, vos programmes assembleur. Cela correspond à l'usage normal du bouton RESET (par exemple, si vous avez fait LPRINT sans imprimante connectée, il vous faudra avoir recours à RESET, mais vous retrouverez vos données intactes). Si le contenu de $EA est différent de $55, par exemple à la mise sous tension, ou si vous avez modifié le contenu de cette adresse, volontairement ou non, ALICE effectue en plus quelques opérations d'initialisation, en particulier une recherche de la taille de la mémoire disponible. Il n'y a qu'une solution pour cela, écrire dans toute la mémoire à partir de $3ØØØ, et ne s'arrêter que lorsque ce qui est écrit est différent de ce qui est relu (tentative d'écriture sur la mémoire morte). Cette opération a pour conséquence la destruction de toute information inscrite en mémoire vive. Si vous voulez remettre à zéro toute la mémoire vive après le déroulement de votre programme, vous pouvez donc faire

```
CLR     $EA
JMP     $F72E
```

A la fin de l'exécution du programme d'initialisation, $EA est mis à $55, pour éviter une réinitialisation complète en cas d'appui sur RESET.

L'interruption RESET est entièrement gérée par ALICE, et vous ne pouvez pas la dérouter pour vos propres programmes. En revanche, toutes les autres interruptions peuvent être utilisées pour vos applications : les vecteurs inscrits en mémoire morte, en $FFFØ-$FFFC...$FFF1-$FFFD, pointent sur des adresses en mémoire vive, que vous pouvez donc modifier à votre gré. Lors de l'initialisation, ALICE inscrit à ces adresses des instructions RTI. Ainsi, si une interruption apparaît, aucun programme ne se déroulera. Ces RTI sont astucieusement espacés de trois octets : on peut donc les remplacer par des JMP suivis de l'adresse de votre programme d'interruption.

Voici le tableau des adresses de la mémoire vive d'ALICE correspondant aux interruptions :

Interruption	Vecteur	Adresse en mémoire vive
NMI	$FFFC-$FFFD	$3212 (12818)
SWI	$FFFA-$FFFB	$320F (12815)
IRQ1	$FFF8-$FFF9	$320C (12812)
ICF	$FFF6-$FFF7	$3209 (12809)
OCF	$FFF4-$FFF5	$3206 (12806)
TOF	$FFF2-$FFF3	$3203 (12803)
SCI	$FFFØ-$FFF1	$3200 (12800)

NMI et IRQ1

Les interruptions NMI et IRQ1 sont provoquées par des transitions de l'état 1 à l'état Ø sur les 'pattes' du MC68Ø3 dénommées $\overline{\text{NMI}}$ et $\overline{\text{IRQ1}}$. Bien entendu, vous ne pouvez pas relier vos propres montages directement sur le MC68Ø3 (ou alors il faut ouvrir ALICE, et je ne réponds plus de rien!). L'un des défauts d'ALICE est que vous n'avez aucun moyen de contrôler la demande d'interruption IRQ1 : alors que l'interruption NMI peut être provoquée par le connecteur d'extension d'ALICE, aucun branchement externe n'est prévu pour IRQ1. C'est d'autant plus dommage qu'aucun des programmes intégrés dans ALICE (BASIC et éditeur-assembleur) n'utilise cette interruption. En revanche, comme je viens de le souligner, l'interruption NMI est disponible sur la broche n° 31 du connecteur d'extension situé à l'arrière d'ALICE (cf. Annexe 4, Le connecteur BUS d'ALICE).

Si vous voulez utiliser plusieurs périphériques, vous devez tous les relier à la broche NMI du connecteur d'extension d'ALICE. La première tâche du programme d'interruption sera de reconnaître le périphérique (ou les périphériques) qui a (qui ont) provoqué l'interruption.

SWI

L'interruption SWI est d'un emploi assez marginal. En fait c'est une instruction qui permet d'effectuer un branchement inconditionnel à un sous-programme en n'utilisant qu'un seul octet (l'instruction SWI), alors que les instructions BSR et JSR en utilisent respectivement 2 et 3. Par exemple, ceci est très précieux dans le programme MONITEUR : lorsque vous demandez l'implantation d'un point d'arrêt, le programme va écrire un SWI à l'adresse spécifiée. Par conséquent, lorsque le programme à tester arrive sur l'instruction SWI, il donne le contrôle à un programme d'interruption qui vous permet d'examiner et de modifier la mémoire et les registres. Lorsqu'il faut relancer l'exécution, il suffit de remettre à la place de SWI l'instruction d'origine et de relancer le programme à cet endroit.

En étudiant le listing du MONITEUR, vous verrez comment on peut modifier les registres, en utilisant la pile pour les stocker puis les restituer. C'est une des utilisations principales de SWI. SWI est le plus souvent utilisée par les systèmes d'exploitation, car elle ne peut être interrompue que par RESET et NMI. Cela permet la gestion de variables système sans grand danger.

ICF, OCF et TOF

Les interruptions ICF, OCF et TOF sont toutes trois relatives à l'un des périphériques intégrés au microprocesseur MC68Ø3, son horloge, ou timer, programmable.

A quoi cela peut-il servir? En premier lieu, à provoquer une interruption au bout d'un temps prédéterminé. Le programme d'interruption peut alors entreprendre des actions de vérification... On sait que le temps demandé est écoulé. C'est plus facile et plus précis que de compter les instructions d'un programme pour savoir que l'on en sortira au bout de *n* millisecondes.

Voyons comment fonctionne cette horloge. Le programmeur peut y accéder par quatre registres, dont l'adresse est située dans les seize premiers octets de la mémoire du 68Ø3. En voici le détail :
— le compteur (adresse $Ø9-ØA = 9-1Ø). Ce compteur sur seize bits est incrémenté à chaque impulsion de l'horloge, qui correspond au signal E (Enable) du 68Ø3. Ce compteur est remis à zéro lors d'un RESET. En principe, on ne doit que le lire; si on tente d'y écrire, il prend la valeur $FFF8. Lorsque le compteur arrive à la valeur $FFFF, il continue en passant par $Ø.

— le registre de comparaison en sortie (adresse $ØB-$ØC = 11-12). Ce registre sur seize bits peut être lu ou écrit. Sa valeur est constamment comparée à celle du compteur. En cas d'égalité, un signal peut être envoyé sur le port 2, bit 1 du 68Ø3, si ce port est configuré en sortie. (*Nota :* sur ALICE, les ports 1 et 2 sont utilisés pour le clavier

et ne sont pas disponibles sur le connecteur d'extension). Ce registre est mis à la valeur $FFFF lors du RESET.

— le registre de capture en entrée (adresse $ØD-$ØE = 13-14). Ce registre de seize bits est uniquement accessible à la lecture. Il est utilisé pour stocker («capturer») la valeur du compteur lorsqu'une transition est appliquée au bit Ø du port 2.

— le registre de contrôle et d'état de l'horloge (adresse $Ø8 = 8). C'est ce registre qui vous permet de décider de l'utilisation de l'horloge. Il comporte 8 bits, qui peuvent tous être lus mais dont cinq seulement peuvent être écrits. Les trois bits accessibles à la seule lecture indiquent qu'un événement (qui peut éventuellement déclencher une interruption) est apparu. Voici la description de ce registre.

B7	B6	B5	B4	B3	B2	B1	BØ
ICF	OCF	TOF	EICI	EOCI	ETOI	EIDG	OLVL

OLVL : Output Level (niveau de sortie). Ce bit indique la valeur qu'il faut envoyer sur le bit 1 du port 2 lorsque le registre de comparaison en sortie est égal au compteur. On peut le lire ou l'écrire, et il est mis à Ø au RESET.

EIDG : Enable Input Digit (autorisation d'entrée de nombres). Ce bit indique quelle transition doit provoquer l'écriture du compteur dans le registre de capture en entrée. Si EIDG = Ø, c'est une transition sur un front descendant, sinon c'est une transition sur un front montant. Ce bit peut être lu ou écrit; il est mis à Ø au RESET.

ETOI : Enable Timer Overflow Interrupt (autorisation d'interruption de dépassement d'horloge). Ce bit peut être lu et écrit. Lorsqu'il est à 1, l'interruption TOF est autorisée. Le débordement du compteur (c'est-à-dire son passage de $FFFF à $Ø provoquera cette interruption. Ce bit est remis à Ø par RESET.

EOCI : Enable Output Compare Interrupt (autorisation d'interruption de comparaison en sortie). Ce bit peut être lu et écrit. Il autorise l'interruption OCF lorsqu'il est à 1. Cette interruption apparaîtra lorsque le registre de comparaison en sortie sera égal au compteur. Ce bit est mis à Ø par RESET.

EICI : Enable Interrupt Capture Input (autorisation d'interruption de saisie en entrée). Ce bit peut être lu et écrit. S'il est à 1, il autorise l'interruption ICF. Cette interruption apparaîtra lorsque la transition programmée par le bit EIDG apparaîtra sur le bit Ø du port 2. Il est mis à Ø par RESET.

TOF : Timer Overflow Flag (drapeau de dépassement d'horloge). Ce bit n'est accessible qu'à la lecture. Il est mis à 1 lorsque le

compteur arrive à $FFFF. Si ETOI est à 1, une interruption est envoyée au 68Ø3. Ce bit n'est remis à Ø qu'au RESET ou *après les lectures successives du registre de contrôle et d'état (adresse 8) et du compteur (adresse 9).*

OCF : Output Compare Flag (drapeau de comparaison en sortie). Ce bit n'est accessible qu'à la lecture. Il est mis à 1 lorsque le registre de comparaison en sortie est égal au compteur. Il provoque une interruption si EOCI vaut 1. Il n'est remis à Ø que par RESET ou *par une lecture du registre de contrôle et d'état (adresse 8) suivie d'une écriture dans le registre de comparaison en sortie (adresse 11 ou 12).*

ICF : Input Capture Flag (drapeau de saisie en entrée). Ce bit n'est accessible qu'à la lecture. Il passe à 1 lorsque la transition programmée par EIDG apparaît sur le bit Ø du port 2. Une interruption apparaîtra si EICI vaut 1. Ce bit est remis à Ø par RESET ou *par la lecture successive du registre de contrôle et d'état (adresse 8) puis du registre de capture en entrée (adresse 13).*

Voilà toutes les possibilités de l'horloge du 68Ø3. Sur ALICE, les ports d'entrées-sorties sont monopolisés par le clavier, ce qui réduit l'utilité de l'interruption OCF et annule complètement toute utilisation d'ICF.

OCF et TOF peuvent être utilisées toutes les deux pour générer des interruptions au bout d'un temps déterminé. Par exemple, si le compteur n'est pas réinitialisé, l'interruption TOF arrivera régulièrement. Calculons sa période : le quartz des nouvelles versions d'ALICE a une période de 4 Méga-Hertz (la première version avait un quartz un peu plus lent). Cette période est divisée de manière interne par quatre pour générer les signaux du 68Ø3. Il y a donc un intervalle d'une micro-seconde entre deux tops envoyés à l'horloge. Chaque interruption est produite lorsque le compteur arrive à $FFFF, c'est-à-dire tous les $(16)^4 = 65\,536$ tops, ce qui fait un intervalle de 65.5 millisecondes entre chaque interruption.

Si l'on veut effectuer certaines opérations chaque seconde, on peut l'utiliser en laissant passer 14 périodes sans rien faire, les opérations étant effectuées lors de la quinzième interruption (65.5 millisecondes multipliées par 15, cela fait Ø.984 seconde). On peut aussi affiner ce compteur en utilisant OCF pour attendre les 16 millisecondes qui manquent.

Le chapitre suivant présente une utilisation de l'horloge.
(*N.B. :* une autre utilisation, marginale, de l'horloge peut être la génération d'un nombre aléatoire. Par exemple, si l'on cherche un chiffre entre Ø et 7, on peut prendre le contenu de l'horloge, et grâce à un

AND n'en conserver que les trois bits de poids faible. Bien entendu cela ne donne pas des nombres parfaitement aléatoires).

SCI

L'interruption SCI est provoquée par un autre équipement interne au 68Ø3, l'interface de communication série. Il permet au microprocesseur de communiquer en utilisant le protocole de la liaison série. Cette interruption ne peut pas être utilisée dans ALICE : elle sert déjà pour la liaison avec l'imprimante et pour l'écriture et la lecture sur un magnétophone à cassettes. Ce sont les mêmes circuits qui sont utilisés pour ces deux fonctions. C'est pourquoi il faut arrêter l'imprimante lorsqu'on enregistre un programme. Voici rapidement les possibilités du 68Ø3 lorsque l'on peut utiliser l'interface de communication série.

— Plusieurs formats sont disponibles : standard ou bi-phasé.
— La communication peut être gouvernée par l'horloge interne ou une horloge externe. La vitesse de cette horloge peut alors être divisée par 16, 128, 1Ø24 ou 4Ø96. Pour un quartz à 4 MégaHertz comme celui d'ALICE, cela donne des vitesses de transmission de 625ØØ bauds, 7812 bauds, 976 bauds ou 244 bauds (ces valeurs n'ont rien de standard).
— Les interruptions peuvent être autorisées pour l'émission et/ou la réception individuellement.
— L'interface de communication série utilise un registre de contrôle (adresse $1Ø) qui permet de choisir l'horloge et sa vitesse. Un autre registre de contrôle indique au 68Ø3 quelles interruptions sont autorisées (adresse $11). Ce même registre indique au programme si le registre d'émission est vide, si le registre de réception est plein ou si une erreur est apparue.
— Les données transmises sont accueillies sur un registre d'émission (adresse $13) et un registre de réception (adresse $12).

Les signaux de cette liaison série sont appliqués sur le port 2 (qui n'est pas accessible sur ALICE) :

— port 2 bit 3 = réception des données
2 = sortie de l'horloge
1 = émission des données.

Pour plus de détails sur ce périphérique qui ne concerne pas directement ALICE, voir la notice MOTOROLA relative au microprocesseur MC68Ø3.

ALICE utilise très peu les interruptions ; en fait, seul RESET est programmé. L'utilisateur peut donc faire ce que bon lui semble des interruptions qui restent disponibles, c'est-à-dire NMI, les interruptions relatives à l'horloge et SWI. Le seul inconvénient des interruptions de l'horloge est qu'elles sont masquables, et que le programme peut en interdire la réalisation. C'est ce que fait

ALICE à chaque fois que vous utilisez la liaison avec un magnétophone à cassette : lorsque ALICE initialise le transfert, le BASIC masque les interruptions, mais le programme présente un défaut, il ne les réinitialise pas. Par conséquent, si vous voulez utiliser les interruptions, il faut penser à les autoriser en début de programme ou après toute liaison avec le magnétophone.

7

Programme : l'horloge en temps réel

Le programme que nous allons voir exploite l'horloge intégrée au microprocesseur MC68Ø3. Il va vous permettre d'utiliser votre ALICE comme une (encombrante) horloge. En effet, après vous avoir demandé d'entrer l'heure, il va afficher l'heure courante (mise à jour chaque seconde) en haut à droite de l'écran. Cet affichage se poursuivra même si vous êtes en train de travailler en BASIC, ou si vous imprimez un listing.

L'affichage peut être suspendu en modifiant le contenu d'une adresse de la mémoire, avec un POKE par exemple. De même, on peut utiliser l'heure dans des programmes assembleur ou BASIC : l'heure, la minute et la seconde courantes sont disponibles dans trois octets de la mémoire, que l'affichage soit demandé ou non.

Attention : cette horloge ne prétend pas rivaliser avec un chronomètre ; sa précision est assez approximative (en particulier, pour les raisons que nous avons vues au chapitre précédent, la durée exacte des " secondes " est de Ø.98 secondes). De plus, le BASIC d'ALICE masque parfois les interruptions, pendant l'impression par exemple, ce qui empêche le décompte des secondes de se faire au rythme voulu.

Organigramme du programme

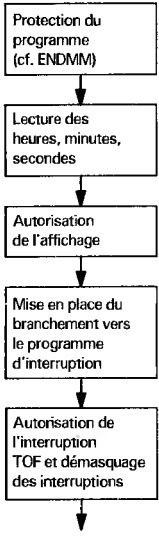
Le programme HORLOG est composé de deux parties :
— la partie d'initialisation, qui demande à l'utilisateur d'entrer l'heure courante, en tapant dans l'ordre l'heure, la minute et la seconde, en deux chiffres à chaque fois. L'interruption est ensuite autorisée ;
— la partie d'interruption proprement dite. L'interruption est provoquée par l'horloge (on utilise ici l'interruption TOF, qui apparaît toutes les 65.5 millisecondes).

Les autres programmes affichent des textes ou lisent des données venant du clavier en utilisant les programmes de la mémoire d'ALICE (OUTTX, OUTCA, KDINP).

Ce programme comporte en plus deux astuces :
— il utilise NBCOL, qui contient le nombre de colonnes affichées sur l'écran (32, 4Ø ou 8Ø). Cela permet dans tous les cas de cadrer le texte (heure, minute, seconde) en haut à droite de l'écran ;
— il modifie ENDMM. Ce mot contient l'adresse du dernier octet de la mémoire d'ALICE. En modifiant son contenu, on interdit au programme BASIC ou assembleur d'aller écraser les données qui se trouvent au-delà. Cela permet de protéger le programme, en particulier contre l'assembleur, qui utilise la fin de la mémoire pendant l'assemblage d'un fichier source.

Programme de mise à l'heure

Partie 1



Partie 2

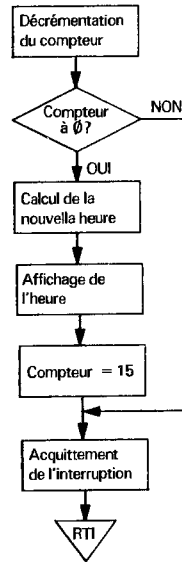
Le programme de lecture de deux chiffres (partie 3) vérifie que les deux caractères entrés sont compris entre Ø et 9. Si c'est le cas, il calcule la valeur hexadécimale correspondante.

Programme d'interruption

Partie 4

Partie 5

Partie 6



Comme on le voit, le programme d'interruption n'est vraiment utilisé à fond qu'une fois sur quinze, afin de calculer la nouvelle heure une fois par seconde. Dans les autres cas, seules six instructions sont exécutées.

Lorsque le compteur passe à Ø, on calcule la nouvelle heure simplement en ajoutant une seconde à l'heure précédente. Il faut bien sûr gérer les changements de minute, d'heure aussi bien que ceux de seconde.

Les programmes BASIC ou assembleur pourront utiliser les trois mots qui donnent l'heure courante : il suffit en BASIC de faire trois PEEK en donnant l'adresse des mots HEURE, MINUT et SECON.

Le dernier programme affiche l'heure sur l'écran. En mettant le mot AFFIC à Ø, on peut interdire cet affichage tout en conservant le calcul de l'heure courante.

Par ailleurs, le programme sauvegarde le support en cours d'utilisation (écran ou imprimante) ainsi que la position courante du curseur (à l'aide du mot CURAD), pour aller écrire son texte en haut à gauche de l'écran. Ces variables du BASIC sont restituées à la sortie du programme.

L'adresse du programme ($8EØØ) convient pour un ALICE 32 avec l'extension mémoire. Pour un ALICE 32 sans extension ou un ALICE 9Ø, il faut changer le contenu de la directive ORG, respectivement en ORG $4EØØ et ORG $AEØØ.

```
 1 ;+-----------------------------------+
 2 ;       Programme HORLOG             !
 3 ;       ---------------              !
 4 ;       Ce programme utilise les inter-  !
 5 ;       ruptions generees par le timer   !
 6 ;       du 6803 .                    !
 7 ;+-----------------------------------+
 8 ;       Auteur : J-F Gallet          !
 9 ;       Date : 23 Novembre 1984      !
10 ;+-----------------------------------+
11
12        ORG   $8E00
13        EXC   HORLO
14
15 NBCOL = $3011       ;NOMBRE DE CARACTERE PAR RANGEE
16 TOFAD = $3203       ;PROGRAMME D'INTERRUPTION TIMER OVERFLOW
17 ENDMM = $3250       ;ADRESSE DE LA FIN DE LA MEMOIRE
18 TCSR  = $0008       ;TIMER CONTROL AND STATUS REGISTER
19 COUNT = $0009       ;COMPTEUR
20 DEVNU = $00E8       ;SUPPORT D'AFFICHAGE (0=ECRAN, 1=IMPRIMANTE,
21 OUTTX = $E7A8       ;AFFICHAGE D'UN TEXTE
22 OUTCA = $F9C6       ;AFFICHAGE D'UN CARACTERE
23 KDINP = $F868       ;ATTENTE D'UN CARACTERE
24 CURAD = $3280       ;ADRESSE DU CARACTERE COURANT SUR L'ECRAN
25 NBC32 = $20         ;32 EN DECIMAL
26
27 ;1.    Programme de mise a l'heure :
28 ;--
29 HORLO
30        LDX   #HORLO
31        GTX   ENDMM          ;A CAUSE DE L'ASSEMBLEUR
32        CLR   AFFIC
33        LDX   #MHTX-$1
34        JSR   OUTTX
35        BSR   LECDI          ;LECTURE DES HEURES
36        BCC   HORLO          ;ERREUR
37        CMPA  #$18
38        BHS   HORLO          ;ERREUR
39        STAA  HEURE
40        LDAA  #$3A
41        JSR   OUTCA
42
43        BSR   LECDI          ;LECTURE DES MINUTES
44        BCC   HORLO          ;ERREUR
45        CMPA  #$3C
46        BHS   HORLO          ;ERREUR
47        STAA  MINUT
48        LDAA  #$3A
49        JSR   OUTCA
```

```
50
51          BSR   LECDI          ;LECTURE DES SECONDES
52          BCC   HORLO          ;ERREUR
53          CMPA  #$3C
54          BHS   HORLO          ;ERREUR
55          STAA  SECON
56
57 ;2.   Initialisation de l'interruption :
58 ;--
59          LDAA  #$1
60          STAA  AFFIC          ;DRAPEAU D'AFFICHAGE
61          LDAA  #$0F
62          STAA  COMPT          ;POUR UNE SECONDE
63          LDX   #ITPRG
64          STX   TOFAD+$1       ;ECRITURE D'UN JMP
65          LDAA  #$7E
66          STAA  TOFAD          ;VERS LE PROGRAMME D'INTERRUPTION
67          LDAA  #$4
68          STAA  TCSR           ;AUTORISATION DE L'INTERRUPTION TIMER
69          CLI                  ;DEMASQUAGE DES INTERRUPTIONS
70          RTS
71
72 ;3.   Lecture de 2 chiffres :
73 ;--
74 ;ENTREE : NEANT
75 ;SORTIE : CARRY = 0 SI CARACTERE ERRONE , 1 SI OK
76 ;        A = VALEUR EN HEXADECIMAL
77 LECDI JSR   KDINP          ;ATTENTE D'UN CARACTERE
78          JSR   OUTCA          ;ECHO
79          BSR   VERIF
80          BCC   LEC10
81          LDAB  #$A
82          MUL                  ;C'ETAIT LA DIZAINE
83          JSR   KDINP          ;ATTENTE DE L'UNITE
84          JSR   OUTCA          ;ECHO
85          BSR   VERIF
86          BCC   LEC10
87          ABA                  ;RESULTAT FINAL
88          SEC                  ;RESULTAT OK
89 LEC10 RTS
90
91 VERIF SUBA  #$30          ;TEST SI CHIFFRE ENTRE 0 ET 9
92          BLO   VER20
93          CMPA  #$A
94          BHS   VER20
95 VER10 SEC                  ;OK : CARRY A 1 , VALEUR DANS A
96          RTS
97 VER20 CLC                  ;ERREUR : CARRY A 0
98          RTS
99
100 ;4.   Decompte pour 1 seconde :
101 ;--
102 ;CE PROGRAMME D'INTERRUPTION EST ACTIVE PAR LE TIMER
103 ;INTERNE DU 6803 , ENVIRON 15 FOIS PAR SECONDE .
104 ;IL UTILISE DONC UN COMPTEUR , ET N'AFFICHE LA NOUVELLE HEURE
105 ;QUE LORSQUE LE COMPTEUR PASSE A 0 .
106 ITPRG
107         SEI                  ;MASQUAGE DES INTERRUPTIONS
108         DEC   COMPT          ;CALCUL DE L'HEURE ???
109         BNE   ITFIN          ;CE N'EST PAS POUR CETTE FOIS-CI
110         BSR   CALHE          ;CALCUL DE LA NOUVELLE HEURE
111         BSR   AFFHE          ;AFFICHAGE DE L'HEURE
112         LDAA  #$0F           ;REINITIALISATION DU COMPTEUR
113         STAA  COMPT
114 ITFIN
```

```
115         LDAA  (TCSR         ;INDISPENSABLES POUR REAUTORISER
116         LDAA  (COUNT        ; LES INTERRUPTIONS
117         RTI
118
119 ;5.    Calcul de l'heure :
120 ;---
121 ;ENTREE : HEURE , MINUTE , SECONDE PRECEDENTES
122 ;SORTIE : HEURE , MINUTE , SECONDE NOUVELLES
123 CALHE LDX   #SECON
124         LDAA  $0,X
125         INCA
126         CMPA  #$3C          ;60EME SECONDE ?
127         BEQ   CAL10         ;OUI : NOUVELLE MINUTE
128         STAA  $0,X
129         RTS
130 CAL10 CLR   $0,X            ;SECONDE 0
131         LDAA  $1,X
132         INCA
133         CMPA  #$3C          ;60EME MINUTE ?
134         BEQ   CAL20         ;OUI : NOUVELLE HEURE
135         STAA  $1,X
136         RTS
137 CAL20 CLR   $1,X            ;MINUTE 0
138         LDAA  $2,X
139         INCA
140         CMPA  #$18          ;24EME HEURE ?
141         BEQ   CAL30         ;OUI : HEURE 0
142         STAA  $2,X
143         RTS
144 CAL30 CLR   $2,X
145         RTS
146
147 ;6.    Affichage de l'heure :
148 ;---
149 ;ENTREE : AFFIC = 0 PAS D'AFFICHAGE , 1 AFFICHAGE
150 ;SORTIE : NEANT
151 AFFHE TST   AFFIC
152         BEQ   AFFIN         ;PAS D'AFFICHAGE
153         LDAA  (DEVNU        ;SAUVEGARDE DU SUPPORT POUR LE CAS
154         PSHA                ; OU L'ON SERAIT EN TRAIN D'IMPRIMER
155         CLR   DEVNU         ;AFFICHAGE SUR L'ECRAN
156         LDX   CURAD
157         PSHX                ;SAUVEGARDE DE LA POSITION DU CURSEUR
158         LDAA  NBCOL
159         CMPA  #$50          ;CALCUL DE LA POSITION D'AFFICHAGE
160         BEQ   AFF10         ; SUIVANT LE MODE COURANT
161         CMPA  #$28
162         BEQ   AFF10
163         LDAA  #NBC32
164 AFF10
165         SUBA  #$A
166         STAA  CURAD+$1
167         CLR   CURAD
168         LDAB  HEURE
169         BSR   IMPRI         ;AFFICHAGE DE L'HEURE
170         LDAB  MINUT
171         BSR   IMPRI         ;            DES MINUTES
172         LDAB  SECON
173         BSR   IMPRI         ;            DES SECONDES
174         PULX
175         STX   CURAD         ;ON RECUPERE LA POSITION DU CURSEUR
176         PULA
177         STAA  (DEVNU        ;ET LE SUPPORT D'AFFICHAGE
178 AFFIN RTS
179
```

```
180 IMPRI LDAA  #$20        ;AFFICHAGE D'UN ESPACE
181       JSR   OUTCA
182       CMPB  #$A
183       BHS   IMP10
184       LDAA  #$20        ;SI LE CHIFFRE A ECRIRE < 10
185       JSR   OUTCA       ;ON AFFICHE UN ESPACE EN 1ERE POSITION
186       BRA   IMP50
187 IMP10 CLRA
188 IMP20 CMPB  #$A         ;CALCUL DES DIZAINES
189       BLO   IMP30
190       INCA
191       SUBB  #$A
192       BRA   IMP20
193 IMP30
194       BSR   IMP40       ;AFFICHAGE DE LA DIZAINE
195 IMP50 TBA               ;AFFICHAGE DE L'UNITE
196 IMP40 ADDA  #$30        ;AFFICHAGE DU CHIFFRE CONTENU DANS A
197       JSR   OUTCA
198       RTS
199
200 ;7.   Donnees du programme :
201 ;--
202
203 COMPT DFO   $0           ;COMPTEUR POUR ARRIVER A UNE SECONDE
204
205 SECON DFO   $0           ;SECONDE COURANTE
206 MINUT DFO   $0           ;MINUTE
207 HEURE DFO   $0           ;HEURE
208
209 AFFIC DFO   $0           ;1 SI AFFICHAGE SUR L'ECRAN
210
211 MHTX  DFO   $D           ;TEXTE DE PRESENTATION
212       'Entrez l'heure SVP :
213       DFO   $2000
214
215 ;8.   Fin
```

8

La liaison série RS 232C

Nous avons déjà vu qu'ALICE pouvait communiquer avec l'extérieur grâce à une liaison au standard RS 232C. Le mot 'standard' peut paraître exagéré : d'une part, la prise utilisée sur ALICE (une prise DIN à quatre broches) n'est pas très commune ; d'autre part, il n'y a pas de véritable norme en ce qui concerne les données, on peut au mieux parler de plusieurs standards.
Il faut toutefois reconnaître que le standard choisi pour ALICE est le plus répandu.

Nota : une erreur s'est glissée dans la documentation fournie avec ALICE ; dans les caractéristiques logicielles d'impression, il faut lire 8 bits de données au lieu de 7 bits. Si votre imprimante est configurée pour 7 bits, cela n'a aucune importance, le huitième bit est ignoré.

La liaison physique : le câble

Le premier problème que l'on rencontre lorsqu'on utilise une liaison série est la compatibilité entre les connecteurs utilisés aux deux extrémités du câble de liaison. J'ai relié plusieurs périphériques avec le micro-ordinateur ALICE que je possède : à chaque fois, il m'a fallu construire un câble adéquat. Cela ne demande que peu de matériel :
— un fer à souder,
— du fil électrique à 4 conducteurs (du genre fil téléphonique),
— les fiches qui s'adaptent aux deux extrémités. La fiche quatre broches n'est pas très facile à trouver, mais les détaillants en matériel électronique pourront vous la fournir.
Voici donc quelques exemples de branchement. Les numéros que je donne sont indiqués sur les fiches.

Pour relier ALICE à une imprimante Microline OKI 82 A

L'imprimante OKI 82 A utilise une fiche CANNON 25 broches mâle.
— Le signal 1 de la fiche ALICE n'est pas utilisé.
— Le signal 2 de la fiche ALICE est relié au signal 11 de la prise CANNON.
— Le signal 3 de la fiche ALICE est relié au signal 7 de la prise CANNON.
— Le signal 4 de la fiche ALICE est relié au signal 3 de la prise CANNON.
— Il faut relier les signaux suivants sur la prise CANNON : 4 et 5 ensemble, 6, 8 et 2Ø ensemble.

Pour relier ALICE au modem DIGITELEC

Le modem DIGITELEC utilise une fiche CANNON 25 broches femelle.
— Le signal 1 de la fiche ALICE est relié au signal 8 de la prise CANNON.
— Le signal 2 de la fiche ALICE est relié au signal 3 de la prise CANNON.
— Le signal 3 de la fiche ALICE est relié au signal 7 de la prise CANNON.
— Le signal 4 de la fiche ALICE est relié au signal 2 de la prise CANNON.

(En fait, cette prise peut n'être utilisée que pour les premiers essais. Ensuite il vaut mieux concevoir une carte qui s'enfiche sur le connecteur d'extension d'ALICE et qui gère tous les signaux du modem grâce à l'interruption NMI.)

Pour relier ALICE à un autre ALICE

Pour réaliser la jonction entre deux ALICE, afin d'échanger des données ou des programmes, il faut deux fiches DIN 4 broches.
— Le signal 1 n'est pas utilisé.
— Le signal 2 du premier ALICE est relié au signal 4 du deuxième.
— Les signaux 3 des deux ALICE sont reliés.
— Le signal 4 du premier ALICE est relié au signal 2 du deuxième.

Comme vous le voyez, il existe plusieurs sortes de branchements. Notez bien qu'il y a peu de risques d'endommager ALICE ou le périphérique si vous inversez deux signaux, car les tensions utilisées en RS 232 varient entre – 12 Volts et + 12 Volts. Mais votre liaison fonctionnera mal dans le meilleur des cas (pertes de caractères) et ne fonctionnera pas du tout dans le pire. Lisez toujours attentivement les notices.

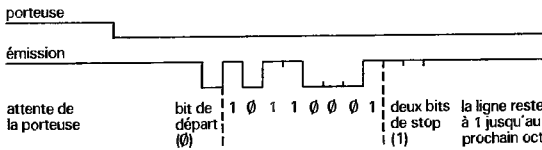
Le principe d'une liaison série

La liaison série permet d'échanger des données en les envoyant bit par bit sur le même fil de transmission, contrairement à la liaison parallèle où les bits sont envoyés en même temps sur 8 fils de transmission.

Pour transmettre un octet sur la liaison série, il faut respecter les étapes suivantes :
— envoyer tout d'abord un bit de départ, qui correspond à un bit à Ø ;
— envoyer ensuite un par un les huit bits de l'octet à transmettre, en commençant par le bit de poids faible (BØ dans notre notation) ;
— envoyer deux bits à 1 pour indiquer la fin de la transmission de l'octet.

Il est alors possible d'envoyer le prochain octet. Lorsque la ligne est inutilisée, il faut envoyer continûment des bits à 1, car le premier Ø est pris comme bit de départ. Ceci est valable pour les fils d'émission et de réception.

De plus, ALICE utilise un signal appelé détection de porteuse qui lui indique si le périphérique est prêt à recevoir les données. Par exemple, dans le cas d'une imprimante, ALICE envoie environ 6Ø caractères par seconde. Si l'imprimante est plus lente, elle utilisera ce signal pour ralentir l'émission des caractères. Voici donc un schéma qui donne l'ordre des opérations à effectuer pour envoyer un octet sur la liaison série. Dans l'exemple, l'octet envoyé est $8D = %1ØØØ11Ø1.



Chaque bit doit être maintenu à Ø ou à 1 pendant un temps rigoureusement constant (à 6ØØ bauds, ce temps est d'un six centième de seconde, c'est-à-dire 1.67 milliseconde). Ceci est nécessaire pour que le périphérique en réception puisse se synchroniser et recevoir correctement les données.

Dans l'exemple ci-dessus, nous avons envoyé huit bits ; c'est le plus commode dans la mesure où cela correspond à un octet. Certains périphériques travaillent sur 7 bits, suivant en cela le standard ASCII qui définit les caractères sur 7 bits.

En outre, pour augmenter la sécurité des échanges, certains protocoles utilisent des bits de parité. Pour cela, il faut envoyer un bit supplémen-

taire, qui s'intercale entre le dernier bit 'utile' de l'octet à transmettre et le premier bit de stop. Il existe deux types de bits de parité :
— la parité paire, où la somme des bits 'utiles' et du bit supplémentaire doit donner un nombre pair. Par exemple, l'octet $8D contient quatre bits à 1 et quatre bits à Ø. Le bit de parité devra donc être à Ø;
— la parité impaire, où la somme des bits 'utiles' et du bit supplémentaire doit donner un nombre impair. Pour $8D. le bit de parité impaire sera alors 1.

Le registre d'émission-réception du MC 68Ø3

Pour gérer la liaison RS 232, ALICE utilise le port 2 du MC 68Ø3. Ce port aurait pu être utilisé en liaison avec les interruptions IRQ2, mais (malheureusement?) ce n'est pas le cas.

ALICE utilise le registre de données du port 2, à l'adresse $3. En fait seuls deux bits de ce registre sont utilisés :
— le bit de poids faible (BØ) contient le bit à émettre;
— le bit B2 reflète la valeur du signal réception de données.

Émission d'un octet sur ALICE

Nous allons voir, dans ce paragraphe, comment émettre un octet à partir d'ALICE. Évidemment, la solution de facilité serait d'utiliser le programme OUTCA ($F9C6) que nous avons vu au chapitre 2 : il suffit de mettre le contenu de DEVNU ($E8) à 1 pour émettre sur la liaison série. Plus facile encore, on pourrait utiliser LPRINT en BASIC.
Il vaut pourtant la peine d'écrire le programme en assembleur, car le problème de l'émission d'un caractère sur la liaison série est analogue à celui de la réception, pour lequel nous n'avons pas les mêmes facilités. Les deux opérations sont liées, et le programme a au moins des vertus pédagogiques.

D'autre part, nous avons vu qu'il existe plusieurs protocoles d'échange d'informations (parités, nombre de bits stop sont des paramètres qui peuvent être modifiées). Les programmes inclus dans ALICE ne traitent pas tous les cas. Le programme fourni met donc à même de pallier cette limitation en donnant les éléments pour adapter ALICE à un protocole particulier.

Détaillons les étapes nécessaires à l'émission d'un octet :
— attente de la porteuse;
— émission du bit de début;
— émission de l'octet (et éventuellement du bit de parité);
— émission du ou des bit(s) de fin.

Voici, étape par étape, les sous-programmes correspondants.

L'attente de la porteuse

Il suffit de lire le registre de données du port 2 (bit 2) du MC 68Ø3 ; dès que ce bit passe à Ø, l'émission est possible. Cela s'écrit en une simple boucle :

```
ATTEN   LDAB <$3     ; LECTURE DU REGISTRE
                     ; DU PORT 2
        BITB #$4     ; TEST DU BIT DE RÉCEPTION
        BNE ATTEN    ; BOUCLE SI SIGNAL A 1
        RTS          ; EXIT
```

Ce programme est donc très simple. On peut toutefois le compléter un peu pour éditer une erreur si l'attente se prolonge trop longtemps ; ce serait utile, par exemple, pour indiquer que l'imprimante n'est pas connectée.

L'émission du bit de début

Nous allons ici écrire le programme d'émission d'un bit. En entrée, ce programme trouvera dans le bit BØ du registre $3 le bit à émettre (Ø ou 1). Pour le bit de début, il faudra donc mettre ce bit à Ø, par exemple grâce à CLRB. L'émission d'un bit est un programme très simple : on recopie le registre B dans le registre de données du port 2, puis on attend le temps nécessaire pour maintenir la même valeur sur la ligne, de manière à ce que le signal soit reconnu par le système de réception.

```
BITEM
                STAB    <$3
                BSR     EMDEL   ; PROGRAMME D'ATTENTE
                RTS
EMDEL
                BSR     EMD1Ø
EMD1Ø
                LDX     EMDET
EMD2Ø
                DEX
                BNE     EMD2Ø
                RTS
EMDET
                DFD     $ØØB5   ; TEMPORISATION
```

Le programme d'attente peut paraître bizarrement écrit, à cause de son branchement à la sous-routine EMD1Ø ; on aurait pu se contenter de doubler la valeur d'EMDET pour obtenir la même temporisation. En fait, tel qu'il est écrit, ce programme permet deux appels différents, suivant la temporisation qu'on veut : en EMDEL, on attend une période entière (c'est l'utilisation que nous en faisons dans BITEM) ; en EMD1Ø,

on attend seulement une demi-période, ce qui est utile dans le programme de réception. Le contenu d'EMDET permet de choisir la vitesse de l'émission. Ici, cela correspond à 600 bauds. les valeurs correspondant aux autres vitesses vous sont données dans le chapitre 2.
Pour émettre le bit de début, il faut donc faire simplement

```
CLRB
BBR BITEM
```

L'émission de l'octet

Dans cet exemple, nous émettons 8 bits sans parité. En entrée, nous supposerons que l'octet à émettre se trouve dans le registre A. Ce programme est donc simplement une boucle de décalage du registre A (8 fois). Le bit à émettre sera copié dans le registre B pour utiliser le programme BITEM que nous venons de voir.

```
EMBB
        LDAB   #$8      ; COMPTEUR DE BITB
EMB1Ø
        PSHB            ; SAUVEGARDE
                        ; DU COMPTEUR
        CLRB
        LSRA            ; BIT A ÉMETTRE
                        ; → CARRY
        RDLB            ; CARRY → B
        BBR    BITEM    ; ÉMISSIDN DU BIT
        PULB
        DECB            ; DÉCRÉMENTATION DU
                        ; COMPTEUR
        BNE    EMB1Ø    ; PRDCHAIN BIT
        RTS             ; ÉMISSION DE L'DCTET
                        ; FINIE
```

L'émission des bits stop

Il suffit d'envoyer deux bits à 1.

```
LDAB    #$1
BSR     BITEM
BSR     BITEM
RTS
```

Réception d'un octet

L'utilité de ce programme est moins contestable : la réception sur la liaison RS 232 n'est pas prévue dans la version de base d'ALICE. La seule solution est donc d'écrire le programme.

La prise fournie sur ALICE limite les possibilités du programme de réception : en dehors du fil de masse, il ne reste que trois signaux : un fil pour l'émission, un fil pour la réception et un fil pour la détection de la porteuse. Le fil d'émission d'un ALICE arrive sur le fil de réception de l'autre ALICE. Comme le signal de détection de porteuse ne peut être

utilisé qu'en réception et qu'il n'existe qu'un signal en émission, il reste donc un signal que nous n'utiliserons pas. ALICE signalera qu'il est prêt à recevoir en envoyant un Ø sur le fil d'émission (ce sera la porteuse).

Un autre obstacle pénalise la réception : il n'est pas possible d'utiliser les interruptions. Le programme devra donc être constamment en réception.

Le programme se découpe aussi en quatre parties :
— émission de la 'porteuse' ;
— attente du bit de début ;
— réception de l'octet ;
— vérification du bit ou des bits stop.

L'émission de la 'porteuse'

Ce petit programme se contente d'envoyer un Ø sur la ligne d'émission.

```
EMPOR
        LOAB <$3
        ANDB #$FE
        STAB <$3
        RTS
```

L'attente du bit de début

Le programme attend que le fil d'émission passe à Ø. Ensuite, il attend pour vérifier que le signal reste bien à Ø. Cette boucle d'attente dure une demi-période. Après cette vérification, le programme attend encore une période pour se synchroniser sur le premier bit à recevoir.

```
ATDEB
        LDAB   <$3
        BITB   #$4      ;  TEST DU SIGNAL REÇU
        BNE    ATDEB    ;  SIGNAL TOUJOURS A 1
        BSR    EMD1Ø    ;  ATTENTE 1/2 PÉRIODE
        LDAB   <$3
        BITB   #$4
        BNE    ATDEB    ;  FAUSSE ALERTE
        BSR    EMDEL    ;  ATTENTE 1 PÉRIODE
        RTS
```

La réception de l'octet proprement dit

Le programme reçoit l'octet bit par bit. Il va lire l'état de la ligne de réception et enregistre son état. Puis il attend une période pour recevoir le bit suivant. Le résultat se trouvera dans le registre A.

```
RCOCT
        CLRA
        LDAB #$B        ;  COMPTEUR DE BITS
RCD1Ø
        PSHB            ;  SAUVEGARDE DU COMPTEUR
        CLC             ;  VALEUR PAR DÉFAUT = Ø
        LDAB <$3
```

```
        BITB #$4        ; TEST SI BIT A Ø DU 1
        BEQ RCO2Ø       ; SAUT SI BIT A Ø
        SEC             ; CARRY = 1
RCD2Ø
        RORA            ; BIT REÇU DANS A
        BSR EMOEL       ; ATTENTE POUR PROCHAIN BIT
        PULB            ; COMPTEUR DE BITS
        DECB            ; DERNIER BIT?
        BNE RCO1Ø
        RTS
```

La vérification du bit stop

Ce programme vérifie seulement que le bit suivant est bien à 1. Sinon il faut signaler une erreur, et ignorer l'octet reçu.

```
VERST
        LDAB  <$3
        BITB  #$4
        BEQ   ERREUR    ; PROGRAMME DE GESTION
                        ;  D'ERREUR
        RTS
```

9

Liaison entre deux ALICE

Ce chapitre utilise les programmes que nous avons vus dans le chapitre précédent. Il permet de transmettre un texte d'un ALICE à un autre par la liaison RS 232C. Pour fonctionner correctement, ce programme exige le câble décrit lui aussi dans le chapitre précédent.

Le mode d'emploi du programme est très simple : il faut le charger dans les deux ALICE, puis lancer l'exécution des deux programmes. Il faut alors répondre à la question.

ÉMISSION OU RÉCEPTION?

Trois réponses sont acceptées, les autres provoquent la répétition de la question :

— un appui sur la touche BREAK indique la fin de l'utilisation du programme. On revient alors sous le contrôle du BASIC d'ALICE.

— la touche R indique que l'on veut se mettre en réception. Chaque octet reçu par la liaison RS 232C est copié sur l'écran, grâce au programme maintenant bien connu, OUTCA ;

— la touche E indique que l'on veut se mettre en émission.

ALICE est alors en attente : chaque touche entrée au clavier est envoyée sur la liaison série.

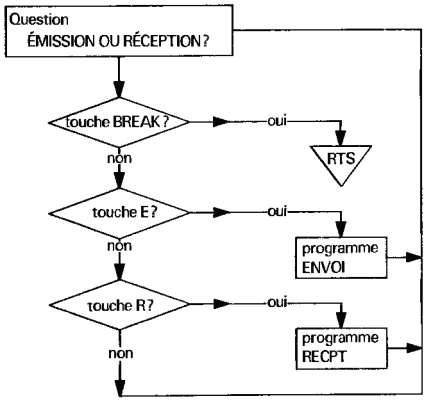
Lorsque vous appuierez sur la touche BREAK, ce caractère sera envoyé sur la liaison série pour débloquer le programme de réception. Ensuite le programme revient lui aussi à la question

ÉMISSION OU RÉCEPTION?

Attention : A chaque caractère envoyé, le programme attend la présence du signal de porteuse. Si ce signal est absent, le programme se bloque. Il est donc indispensable que ce signal soit fourni par l'autre ALICE, déjà en réception. Pour tester le programme d'émission, vous pouvez relier ALICE à une imprimante.

Organigramme du programme

Le programme est très simple, puisque le plus gros a été fait dans les routines du chapitre précédent.



Le programme ENVOI lit la touche entrée par l'utilisateur, fait l'écho sur l'écran et envoie l'octet sur la liaison série. Si la touche n'était pas BREAK, il recommence. Les programmes utilisés sont déjà connus : KDINP pour la scrutation du clavier (avec clignotement du curseur), OUTCA pour l'écriture sur l'écran.

Le programme RECPT est encore plus simple : il reçoit un octet, l'écrit sur l'écran (OUTCA encore) et recommence si le code reçu n'était pas celui de BREAK.

```
 1 ;-------------------------------------+
 2 ;      Programme RS232C               !
 3 ;      ---------------                !
 4 ;      Ce programme permet d'echanger !
 5 ;      un texte entre 2 ALICE . Il u- !
 6 ;      tilise la prise DIN situee a   !
 7 ;      arriere d'ALICE .              !
 8 ;-------------------------------------+
 9 ;      Auteur : J-F Gallet            !
10 ;      Date : 29 Decembre 1984        !
11 ;-------------------------------------+
12
13       ORG    $8C00
14       EXC    RS232
15
```

```
;1.    Donnees externes utilisees par le programme :
;--
OUTTX = $E7A8      ;IMPRESSION D'UN TEXTE
KDINP = $F868      ;ACQUISITION D'UN CARACTERE
OUTCA = $F9C6      ;IMPRESSION D'UN CARACTERE

;2.    Choix : emission ou reception ?
;--
RS232
        LDX   #TXQUE-$1
        JSR   OUTTX
        JSR   KDINP
        JSR   OUTCA
        CMPA  #$3
        BNE   RS010
        RTS
RS010
        CMPA  #$45          ;E COMME EMISSION
        BNE   RS020
        LDX   #TXEMI-$1
        JSR   OUTTX
        BSR   ENVOI
        BRA   RS232
RS020
        CMPA  #$52          ;R COMME RECEPTION
        BNE   RS232
        LDX   #TXREC-$1
        JSR   OUTTX
        BSR   RECPT
        BRA   RS232

;3.    Emission des caracteres entres au clavier :
;--
ENVOI
        JSR   KDINP
        JSR   OUTCA
        BSR   EMISS
        CMPA  #$3
        BNE   ENVOI
        RTS

;4.    Reception des caracteres envoyes par l'autre ALICE :
;--
RECPT
        BSR   RECEP
        JSR   OUTCA
        CMPA  #$3
        BNE   RECPT
        RTS

;5./   Programme d'emission :
;--
;Entree : A = Octet a emettre
;Sortie : A = Inchange
EMISS
        BSR   ATTEN               ;ATTENTE DE LA PORTEUSE
        CLRB                      ;EMISSION DU BIT START
        BSR   BITEM
        BSR   EM8B                ;EMISSION DE L'OCTET
        LDAB  #$1                 ;ENVDI DU BIT STOP
        BSR   BITEM
        BSR   BITEM
        RTS
```

```
 79
 80 ;6.   Sous-programme d'attente de la porteuse :
 81 ;---
 82 ATTEN
 83       LDAB  ($3
 84       BITB  #$4
 85       BNE   ATTEN
 86       RTS
 87
 88 ;7.   Sous-programme d'emission d'un octet :
 89 ;---
 90 EM8B
 91       PSHA                          ;SAUVEGARDE DU CARACTERE
 92       LDAB  #$8
 93 EM810
 94       PSHB
 95       CLRB
 96       LSRA
 97       ROLB
 98       BSR   BITEM
 99       PULB
100       DECB
101       BNE   EM810
102       PULA
103       RTS
104
105 ;8.   Sous-programme d'emission d'un bit :
106 ;---
107 BITEM
108       STAB  ($3
109       BSR   EMDEL
110       RTS
111
112 ;9.   Sous-programme de temporisation :
113 ;---
114 EMDEL
115       BSR   EMD10
116 EMD10
117       LDX   EMDET
118 EMD20
119       DEX
120       BNE   EMD20
121       RTS
122
123 EMDET DFD   $0085
124
125 ;10.  Programme de reception d'un octet sur la ligne RS232 :
126 ;---
127 ;Entree : neant
128 ;Sortie : A = caractere recu
129 RECEP
130       BSR   EMPOR
131       BSR   ATDEB
132       BSR   RCOCT
133       BSR   VERST
134       BSR   STPOR
135       RTS
136
137 ;11.  Sous-programme d'emission et d'arret de la porteuse :
138 ;---
139 EMPOR
140       LDAB  ($3
141       ANDB  #$FE
```

```
142         STAB   ($3
143         RTS
144 STPDR
145         LDAB   #$1
146         STAB   ($3
147         RTS
148
149 ;12.    Sous-programme d'attente du bit de debut :
150 ;---
151 ATDEB
152         LDAB   ($3
153         BITB   #$4
154         BNE    ATDEB
155         BSR    EMD10
156         LDAB   ($3
157         BITB   #$4
158         BNE    ATDEB
159         BSR    EMDEL
160         RTS
161
162 ;13.    Sous-programme de reception d'un octet :
163 ;---
164 RCOCT
165         CLRA
166         LDAB   #$8
167 RCO10
168         PSHB
169         CLC
170         LDAB   ($3
171         BITB   #$4
172         BEQ    RCO20
173         SEC
174 RCO20
175         RORA
176         BSR    EMDEL
177         PULB
178         DECB
179         BNE    RCO10
180         RTS
181
182 ;14.    Sous-programme de verification du bit stop :
183 ;---
184 VERST
185         LDAB   ($3
186         BITB   #$4
187         BEQ    EREUR
188         RTS
189 EREUR
190         PSHA
191         LDX    #ERTX-$1
192         JSR    OUTTX
193         PULA
194         RTS
195
196 ;15.    Textes divers :
197 ;---
198 TXQUE   DFD    $0D
199         'EMISSION OU RECEPTION ?
200         DFD    $2000
201
202 TXEM1   'MISSION : TAPEZ VOTRE TEXTE .
203         DFD    $0D00
204
```

```
205 TXREC 'ECEPTION : PRET A RECEVOIR .
206       DFD   $0D00
207
208 ERTX  DFO   $OD
209       'ERREUR EN RECEPTION. CARACTERE RECU :
210       DFD   $2000
211
212 ;16.  Fin
```

10

Le processeur de visualisation, les bases

Le circuit intégré EF 9345

La principale différence entre la première version d'ALICE et les nouvelles, ALICE 32 et ALICE 9, réside dans l'utilisation d'un circuit intégré EF 9345 pour contrôler la visualisation des données. Ce circuit intégré reçoit du microprocesseur des informations comme les caractères à afficher, l'emplacement d'affichage, les données de couleur... Il traduit ces informations en signaux électriques transmis à la prise Péritel.

Le circuit EF 9345 conserve les possibilités offertes par la première version d'ALICE : les jeux de caractères alphanumériques et semi-graphiques sont restés compatibles. De plus, l'affichage de 16 lignes de 32 caractères a été conservé, ce qui permet d'utiliser sur les nouveaux modèles les programmes conçus pour l'ancien.

Les premières différences sont illustrées par l'effet des commandes CLS 4, CLS 8 et CLS 81, qui donnent un premier aperçu des capacités du EF 9345. Il est aussi possible de changer la couleur du fond, celle des caractères, dans un programme BASIC à l'aide de la fonction SET*. Mais ce circuit est si puissant qu'il mérite d'être étudié ici, au long de trois chapitres.

Ce premier chapitre est destiné à expliquer les caractéristiques techniques du EF 9345 qui vous permettront de suivre le détail des deux suivants, consacrés respectivement aux modes 40 et 80 caractères par rangée.

Les registres du EF 9345

Dans la première version d'ALICE, il était très simple d'aller écrire sur l'écran sans passer par le BASIC : il suffisait d'aller stocker un octet dans une partie de la mémoire du micro-ordinateur, qui était le reflet de ce qui s'affichait sur l'écran. Les ALICE 32 et 9Ø fonctionnent différemment : pour afficher quelque chose, il faut envoyer une série d'ordres à un processeur différent, le 9345. Il y a toujours une zone de la mémoire correspondant au contenu de l'écran, mais elle est désormais mise à jour par le 9345, au fur et à mesure qu'il reçoit des ordres du microprocesseur 68Ø3. Cette partie de la mémoire *n'est pas disponible* pour le microprocesseur, qui ne peut ni la lire, ni y écrire directement.

La communication avec l'écran passe donc par les registres du 9345, qui sont au nombre de 13 :
— 8 sont directement adressables par le microprocesseur (c'est-à-dire, par exemple, avec des instructions comme LDAA ou STAA);
— 5 ne sont adressables que par l'intermédiaire des 8 premiers.

Lorsque plusieurs ordres composant une commande sont envoyés au 9345, rien ne se passe tant que le programme n'a pas demandé l'exécution de la séquence entière. C'est pour cela que chaque registre a deux adresses : la première est utilisée pour passer une commande simple, la seconde pour passer une commande et débuter l'exécution de la séquence.

Le premier registre, appelé RØ, se trouve à l'adresse $BF2Ø pour une commande, et $BF28 pour l'exécution. Le registre RØ sert de registre de commande lorsqu'on y écrit, et de registre d'état lorsqu'on y lit. En particulier, ce registre sélectionne les modes 4Ø ou 8Ø caractères. En lecture, il donne l'état de disponibilité du 9345 (voir plus loin le traitement du signal BUSY).

Les trois registres R1, R2 et R3 se trouvent aux adresses $BF21, $BF22 et $BF23 pour une commande, et $BF29, $BF2A et $BF2B pour une exécution. Ce sont des registres de données qui indiquent au 9345 quel caractère doit être écrit, avec sa couleur et ses attributs.

Les quatre registres R4, R5, R6 et R7 se trouvent aux adresses $BF24, $BF25, $BF26 et $BF27 pour une commande, et $BF2C, $BF2D, $BF2E et $BF2F pour une exécution. Ce sont des pointeurs dans la mémoire privée du 9345.

Les cinq derniers registres, comme nous l'avons vu, ne sont pas adressables par le microprocesseur. Il faut y accéder indirectement, en particulier grâce au registre de commande RØ. Leurs rôles sont les suivants :

— les deux premiers, appelés ROR et DOR, servent d'adresses de base dans la mémoire privée du 9345;
— les trois autres, appelés PAT, MAT et TGS fournissent les caractéristiques générales des signaux vidéo (nombre de lignes, entrelaçage, synchronisation) qui assurent un bon fonctionnement avec un téléviseur. Ils contiennent aussi les caractéristiques de visualisation valables pour toute la page : couleur de marge, type de curseur.

Le signal BUSY

Ce signal occupe un seul bit du registre RØ, en lecture, mais il est très important : si vous négligez ce signal, vous obtiendrez n'importe quoi sur votre écran en essayant d'y écrire par l'intermédiaire des autres registres.
Le remède est simple : en principe, avant d'écrire dans un registre du 9345, il faut attendre son autorisation, c'est-à-dire que le bit B7 du registre RØ soit à Ø. Toutefois, comme ce bit ne passe à 1 que lorsque le microprocesseur demande au 9345 d'exécuter une séquence, il est plus sûr et plus commode de tester ce bit BUSY juste après avoir demandé une exécution. En général, on utilise ce sous-programme très simple :

```
BUSY    TST     RØ
        BMI     BUSY
        RTS
```

Ne l'oubliez jamais! C'est le genre d'erreur qui donne des résultats bizarres, apparemment aléatoires, et dont la source est très difficile à trouver, car le microprocesseur peut dans ce cas envoyer des séquences parfaitement correctes au 9345 sans que celui-ci les exécute conformément à votre attente.

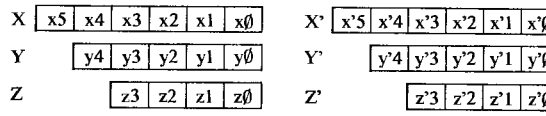
La mémoire privée du 9345

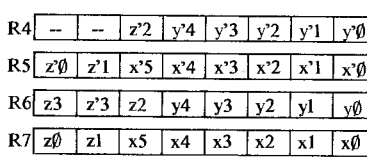
Le 9345 peut gérer jusqu'à 16 K-octets de mémoire privée. ALICE lui alloue 8 K-octets. Cette mémoire est divisée en 8 blocs de 1 K-octet. Chaque bloc est divisé lui-même en 25 tampons *(buffers)* de 4 octets. Pour donner l'adresse d'un octet, il faut fournir les paramètres suivants :
— X, numéro d'ordre de l'octet dans le buffer (Ø<= X <=39);
— Y, numéro d'ordre du buffer dans un bloc d'1 Ko; (c'est là que réside une bizarrerie du 9345 : la valeur de Y peut être Ø, 1 ou entre 8 et 31, bornes comprises. De plus, 1 Ko vaut 1Ø24 octets; donc on a 25 tampons de 4Ø octets plus un reste de 24 octets. Ces octets sont récupérés de la manière suivante : le tampon Y=1 n'existe que dans les blocs pairs et il est incomplet dans les blocs impairs (comme sa gestion est compliquée, on ne l'utilise jamais);
— Z, numéro du bloc de 1 Ko (Ø <= Z <=7).

Les blocs sont regroupés en districts, formés de quatre blocs successifs (1er district : Z=Ø, 1, 2, 3; 2e district : Z=4, 5, 6, 7).

Les registres utilisés pour accéder à cette mémoire sont au nombre de quatre :
— R4 et R5, qui forment un pointeur auxiliaire;
— R6 et R7, qui forment le pointeur principal.

Par exemple, si vous voulez utiliser un pointeur sur une zone définie par les paramètres X, Y et Z et un pointeur auxiliaire sur une zone définie par les paramètres X', Y' et Z', voici le format des registres R4, R5, R6 et R7 :





Comme vous le voyez, le format de ces registres est lui aussi assez complexe. Il faut un programme du genre de celui proposé ici pour calculer les valeurs de R4, R5, R6 et R7.
(*N.B.* : la plupart du temps, seuls les registres principaux R6 et R7 sont utilisés.)

Programme de calcul de R7 et R6

```
                    ZØ=$1
                    Z1=$2
                    Z2=$4
                    Z3=$8

                ; 1. CALCUL DE R7
                LDAB    Z
                LDAA    X
                BITB    #ZØ
                BEQ     ETØØ
                ORAA    #$8Ø
ETØØ            BITB    #Z1
                BEQ     ETØ1
                ORAA    #$4Ø
ETØ1            STAA    R7
```

```
                ;2. CALCUL DE R8
                LDAA    Y
                BITB    #Z2
                BEQ     ETØ2
                ORAA    #$2Ø
ETØ2            BITB    #Z3
                BEQ     ETØ3
                ORAA    #$BØ
ETØ3            STAA    R6
```

Nota : en général ce programme est très simplifié, car on utilise le premier bloc pour la visualisation; dans ce cas on a Z=Ø, ce qui donne :

```
        LDAA  X
        STAA  R7
        LDAA  Y
        STAA  R6
```

ou

```
        LDAA  Y
        LOAB  X
        STO   R6
```

Partition de la mémoire privée du 9345

Une page, telle qu'elle est affichée sur l'écran, représente 25 rangées de caractères (en fait le mode 16 rangées de 32 caractères est obtenu en inscrivant des caractères noirs sur les rangées et caractères superflus!).
Chaque rangée est associée à 2 ou 3 tampons de 4 octets. Cet ensemble constitue un buffer de ligne. Ces tampons suivent la règle suivante :
— ils ont le même numéro de buffer Y;
— ils se trouvent dans 2 ou 3 blocs successifs du même district;
— le premier bloc de cet ensemble doit être un bloc pair.
Cela laisse peu de choix : lorsqu'il faut deux blocs, le premier peut être le bloc Ø, le bloc 2, le 4 ou le 6; lorsqu'il en faut trois, le premier bloc peut être le bloc Ø ou le bloc 4.

Par conséquent on peut avoir jusqu'à quatre pages de visualisation en mémoire. En général, pour utiliser les modes d'adressage de la mémoire les plus simples, on se limite à une ou deux pages en mémoire. Nous nous limiterons à ces cas dans ce livre.

On définit donc trois blocs pour une page sur l'écran :
— Le premier bloc définit les caractères qui se trouvent sur l'écran. Ce sont soit des caractères alphanumériques (lettres, chiffres, ponctuation), soit des caractères semi-graphiques, soit des caractères graphiques redéfinis par le programmeur. (Nous verrons plus loin comment définir ses propres caractères).
— Les deux blocs suivants définissent les caractéristiques du caractère affiché : couleur, clignotement, inversion vidéo... Ces

caractéristiques diffèrent suivant le mode choisi, 4 ou 8 caractères par rangée.

Le reste de la mémoire est consacré à la définition des caractères par l'utilisateur. Cette définition n'est possible qu'en mode 4 caractères; elle sera étudiée dans le chapitre qui lui est consacré.

Les commandes usuelles du 9345

Toutes les commandes sont spécifiées au 9345 par le registre RØ.

Le type de la commande est spécifié par les quatre bits de poids fort de RØ, (B7, B6, B5, B4). Les quatre bits de poids faible sont des paramètres nécessaires à l'exécution de la commande. Les arguments sont transmis grâce aux autres registres. Suivant le nombre d'octets d'arguments, on aura des commandes courtes ou longues. Si le processeur reste maître du nombre d'arguments à actualiser, la commande est dite variable.

Voici le détail des commandes utilisées de manière courante.

1. Indirection. **IND**

RØ =	1	Ø	Ø	Ø	R/$\overline{W}$	— r —

Cette commande permet de lire ou d'écrire un des registres indirects : on lit le contenu du registre si R/$\overline{W}$ = 1 ; l'octet est transféré du registre indirect vers le registre R1. Si R/$\overline{W}$ = Ø, on est en position d'écriture, l'octet est transféré du registre R1 vers le registre indirect.

r indique quel est le registre concerné :

— r = 1 registre TGS
— r = 2 MAT
— r = 3 PAT
— r = 4 DOR
— r = 7 ROR (les valeurs Ø, 5 et 6 sont inutilisées).

L'argument est fourni par le registre R1.

Cette commande est très importante : les registres indirects précisent des paramètres comme la norme de l'écran utilisé, les paramètres généraux de l'écran (marge, type du curseur...), le mode d'affichage (4Ø ou 8Ø colonnes), l'adresse des blocs contenant les caractères définis par le programmeur. Ces registres sont en général initialisés en début de programme, et ne sont modifiés que pour la définition de caractères graphiques.

2. Affichage ou lecture d'un caractère en mode 4Ø colonnes.
Commande longue (24 bits) **KRF**

RØ =	Ø	Ø	Ø	Ø	R/$\overline{W}$	ā	Ø	i

Cette commande permet d'afficher un caractère défini par R1, R2, R3 à l'adresse définie par R6 et R7, lorsque le bit R/$\overline{W}$ est à Ø. Si R/$\overline{W}$ est à 1, c'est l'inverse : le caractère et ses caractéristiques qui se trouvent à l'adresse définie par R6 et R7 se retrouvent dans R1, R2 et R3.
$\overline{a}$ indique la condition d'arrêt de la commande :
a = Ø, on n'affiche qu'un octet;
a = 1, l'affichage se poursuit jusqu'à ce qu'une nouvelle commande soit envoyée.
i, lorsqu'il est à 1, indique que le pointeur dans R6 et R7 sera incrémenté après exécution de la commande.
(Cette commande, et la commande OCT, que nous verrons un peu plus loin, seront plus largement analysées dans le chapitre 12).

3. Affichage ou lecture d'un caractère en mode 8Ø colonnes.
Commande longue (12 bits) **KRL**

RØ =	Ø	1	Ø	1	R/$\overline{W}$	$\overline{a}$	Ø	i

R/$\overline{W}$, $\overline{a}$ et i ont la même signification que dans la commande KRF. Cette commande est analogue à KRF, ses paramètres seront détaillés dans le chapitre 11.

4. Écriture ou lecture d'un octet en mémoire privée.
OCT

RØ =	Ø	Ø	1	1	R/$\overline{W}$	p	Ø	i

R/$\overline{W}$ et i ont la même signification que pour les commandes précédentes. p indique quels registres sont utilisés comme pointeurs :
Ø = utilisation de R6 et R7
1 = utilisation de R4 et R5.
Nous reverrons cette commande dans la description du mode 4Ø colonnes.

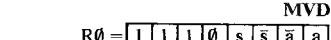
5. Recopie simple d'un tampon sur un autre.
MVB

RØ =	1	1	Ø	1	s	$\overline{s}$	$\overline{a}$	a

s$\overline{s}$ indique le sens de la copie :
s$\overline{s}$ = Ø1, la source est pointée par les registres R6 et R7, la destination par les registres R4 et R5;
s$\overline{s}$ = 1Ø, la source est pointée par les registres R4 et R5, la destination par les registres R6 et R7.
a$\overline{a}$ indique la condition de fin de recopie :
a$\overline{a}$ = Ø1, la copie s'arrête à la fin du tampon;
a$\overline{a}$ = 1Ø, il n'y a pas d'arrêt tant que le 6803 n'envoie pas d'autre commande.

6. Recopie double.

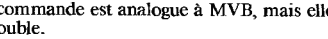
MVD



Cette commande est analogue à MVB, mais elle travaille sur un tampon double.

7. Recopie triple.

MVT

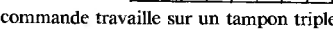


Cette commande travaille sur un tampon triple.

Ces trois commandes, MVB, MVD et MVT, sont utilisées pour recopier une rangée de caractères sur une autre rangée. Les principales applications sont les suivantes :
— le défilement de l'écran vers le haut *(roll up)*;
— le défilement de l'écran vers le bas *(roll down)*;
— l'effacement d'écran (on recopie une rangée vide sur toutes les autres).

8. Incrémentation de Y.

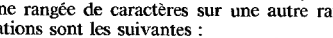
INY



Cette commande incrémente le numéro de buffer Y. L'incrémentation s'effectue dans le registre R6.

9. Pas d'opération.

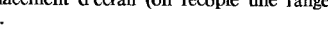
NOP



Cette commande est utilisée pour initialiser le 9345, et pour arrêter une commande qui utilise l'incrémentation automatique ($\bar{a} = 1$).

Les commandes du 9345 rarement utilisées

1. Affichage ou lecture d'un caractère en mode 4Ø colonnes.
Commande courte (16 bits) KRG



Cette commande est similaire à la commande KRF. Elle n'utilise que les registres R1 et R2 comme paramètres (mais R3 est un registre de travail modifié lors de l'exécution). L'avantage de cette commande est sa compatibilité avec un autre circuit intégré, le EF 934Ø/9341. Elle fait perdre certains avantages du 9345 : le jeu des caractères accen-

tuées et l'un des jeux de caractères semi-graphiques ne sont plus accessibles.
Cette commande n'est jamais utilisée dans ALICE.

2. Affichage ou lecture d'un caractère en mode 4Ø colonnes.
Commande variable (24 ou 8 bits)

KRV

	Ø	Ø	1	Ø	R/$\overline{W}$	ā	Ø	i
RØ =								

Cette commande est similaire à la commande KRF. Elle n'utilise que le registre R1 lorsque les attributs du caractère courant sont les mêmes que pour le caractère précédent. Lorsque l'on veut définir de nouveaux attributs dans les registres R2 et R3, il faut simplement mettre le bit de poids fort de R1 à 1. De plus, KRV utilise une astuce : considérant qu'un caractère accentué est ordinairement suivi par un caractère non accentué, ce mode repasse automatiquement dans le jeu de caractères non accentués après un caractère accentué.
Attention : le premier caractère d'une rangée doit être complètement défini par R1, R2 et R3.
Ce mode n'est jamais utilisé dans ALICE : dans ce cas, la structure des buffers contenant les attributs est beaucoup moins prévisible que pour les codes longs.

3. Affichage ou lecture d'un caractère en mode 8Ø colonnes.
Commande courte (8 bits)

KRC

	Ø	1	Ø	Ø	R/$\overline{W}$	ā	Ø	i
RØ =								

Cette commande est similaire au mode KRL, avec les limitations suivantes : il n'y a pas de caractères graphiques, et les caractères alphanumériques ne possèdent aucun attribut.

4. Expansion. **EXP**

	Ø	1	1	Ø	Ø	Ø	Ø	Ø
RØ=								

Cette commande transforme une rangée définie en code variable en rangée définie en code étendu (similaire au code long). La commande EXP n'est utilisable qu'en mode 4Ø caractères par rangée.

5. Compression. **CMP**

	Ø	1	1	1	Ø	Ø	Ø	Ø
RØ =								

Cette commande est l'inverse de la commande EXP.

6. Mode caractères étendus. **KRE**

	Ø	Ø	Ø	1	R/$\overline{W}$	Ø	Ø	i
RØ =								

Cette commande, utilisée en conjonction avec la commande EXP, permet de lire ou d'écrire un code étendu et de la transformer en code long.

7. Commandes de synchronisation verticale. **VSM et VRM**

RØ =	1	Ø	Ø	1	1	Ø	Ø	1	VSM
RØ =	1	Ø	Ø	1	Ø	1	Ø	1	VRM

Ces commandes affectent un bit du registre d'état RØ. Ce bit (qui donne l'état de la synchronisation) reste toujours à Ø après la commande VSM, et donne réellement l'état après la commande VRM.

Le registre d'état du 9345

Nous venons de voir que pour envoyer des commandes au 9345 il suffit de placer l'octet de commande dans le registre RØ et de commander l'exécution.

Contrairement aux autres registres, RØ a un rôle différent lorsqu'il est lu : en lecture, RØ donne des renseignements sur l'état du 9345. Voici sa structure :

RØ =	Busy	AL	LXm	LXa	MSB_{R1}	Sync	–	–

Nous avons déjà vu le rôle de BUSY : tant que BUSY est à 1, cela signifie que la commande précédente n'est pas encore complètement exécutée. Pendant ce temps, vous pouvez lire le registre RØ, et vous pouvez envoyer une commande en demandant son exécution. La commande en cours est alors achevée autoritairement. Cette facilité est utilisée en particulier lorsque le bit ā est mis à 1 dans les commandes KRF, KRL et OCT, car ces commandes ne possèdent pas de condition d'arrêt.

LXm est mis à 1 lorsque le pointeur X dans un tampon atteint la valeur 39. LXm concerne les registres principaux R6 et R7.

LXa a le même rôle, mais concerne les registres auxiliaires R4 et R5.

AL est le bit d'alarme : il est mis à 1 lorsque soit LXm soit LXa sont mis à 1 et qu'une incrémentation est demandée (bit i dans une commande).

MSB_{R1} reflète la valeur du bit de poids fort du registre R1. En général, il est à 1 lorsque l'on utilise des caractères définis par l'utilisateur.

Sync : ce bit donne l'état du signal de synchronisation verticale. Il est toujours à Ø si l'on envoie la commande VRM.

Lorsqu'une commande débute son exécution, les bits AL, LXm, LXa et MSB_{R1} sont remis à Ø.

Les registres du 9345

1. Les registres à accès direct

RØ : nous avons déjà vu son rôle ; il sert à envoyer les commandes au 9345, ainsi qu'à lire l'état du 9345.
R1 : le registre R1 est utilisé pour les transferts vers ou en provenance du 9345.
R2 : le registre R2 a un rôle similaire à celui de R1.
R3 : le registre R3 contient les attributs concernant les caractères contenus dans R1 et/ou R2.
R4, R5, R6 et R7 forment des pointeurs dans la mémoire privée du 9345 ; nous avons déjà vu leur structure.
L'utilisation des registres R1, R2 et R3 dépend fortement du mode utilisé, 4Ø ou 8Ø caractères par rangée. Nous étudierons donc leurs rôles plus précisément dans les chapitres 11 et 12.

2. Les registres à accès indirect

Pour lire ou écrire ces registres, il faut obligatoirement utiliser la commande IND. On utilise alors le bit R/$\overline{W}$ pour définir l'opération : lorsque R/$\overline{W}$ est mis à Ø, le contenu de R1 est transféré dans le registre indirect indiqué par la commande, alors que si R/$\overline{W}$ est à 1, c'est le contenu du registre indirect qui est transféré dans R1.

Voici le contenu des registres indirects :

ROR et DOR contiennent les adresses de base de la page à visualiser (ROR) et du générateur de caractères définis par l'utilisateur (DOR) en mode 4Ø caractères par rangée.

PAT, MAT et TGS sélectionnent des caractéristiques générales de l'écran.

ROR =	Z3	Z1	Z2	--	YOR

Z3, Z2 et Z1 définissent le numéro du premier bloc qui doit être visualisé à l'écran. Ce premier bloc doit toujours être un bloc pair, c'est pour cela que Z0 n'est pas programmable (il est toujours égal à Ø).

YOR définit l'adresse de la rangée d'origine. C'est une valeur entre 8 et 31. En effet, pour le 9345, l'écran est divisé en deux parties :
— la rangée de service, qui est la première rangée de l'écran, en haut,
— les rangées de travail (les 24 autres).

Les rangées sont visualisées dans l'ordre, à partir de la rangée précisée par YOR. Pour garder l'ordre contenu dans la mémoire privée du 9345, on affecte à YOR la valeur 8.

DOR =	Z3Q	Z3G'1	Z2G'1	Z1G'1	Z3G'Ø	Z2G'Ø	Z1G'Ø	ZØG'Ø

en 4Ø colonnes

Le registre DOR est utilisé différemment suivant le mode choisi (4Ø ou 8Ø colonnes). En 4Ø colonnes, il définit les adresses dans la mémoire privée du 9345 où se trouvent les générateurs de caractères graphiques programmés par l'utilisateur. Voici rapidement leur signification (tout cela sera développé plus longuement dans le chapitre 12).

Z3Q : bit de poids fort du numéro du premier bloc qui contiendra la définition des caractères en mode quadrichrome.

Z3G'1, Z2G'1 et Z1G'1 : bits de poids fort du numéro du premier bloc qui contiendra des caractères semi-graphiques en mode bichrome.

Z3G'Ø Z2G'Ø, Z1G'Ø et ZØG'Ø : numéro du bloc qui contiendra des caractères alphanumériques définis par l'utilisateur, en fait en mode bichrome là aussi.

DOR =	il	B1	G1	R1	iØ	BØ	GØ	RØ

en 80 colonnes.

Dans le mode 8Ø caractères par rangée, DOR définit les deux couleurs actives.

iØ autorise (ou interdit) l'incrustation du fond. BØ, GØ et RØ sont les composantes de la première couleur active; BØ=1 si il y a du bleu (Ø sinon), GØ=1 définit du vert et RØ du rouge. Il est possible de combiner ces couleurs. Par exemple si BØ=GØ=RØ=1, on obtient la couleur blanche. La signification des bits il, B1, G1 et R1 est identique, mais s'applique à la deuxième couleur active. Les deux couleurs actives sont des couleurs de caractères. La couleur de fond est celle de la marge, elle est définie dans le registre indirect MAT.

TGS =	CHAR/RG	YS	VIDEO CARAC

CHAR/RG : ces deux bits sont utilisés en conjonction avec le bit de poids fort du registre PAT. Ils permettent de choisir le mode 4Ø ou 8Ø caractères par rangée. Voici les différentes combinaisons possibles :

PAT 7	TGS 7	TGS 6	Mode
Ø	Ø	Ø	4Ø caractères, commande longue
Ø	Ø	1	4Ø caractères, commande variable
1	Ø	Ø	4Ø caractères, commande courte
Ø	1	1	8Ø caractères, commande longue
Ø	1	Ø	8Ø caractères, commande courte

YS définit la rangée de service : ce peut être soit Ø soit 1 (rappel : le tampon 1 n'existe que dans les blocs de rang pair).

VIDEO CARAC définit les caractéristiques de l'écran utilisé :

TGS 4 : type de synchronisation (1 : signal composite et comparateur de phase, Ø : synchronisation horizontale et verticale).
TGS 3 et TGS 2 : resynchronisation verticale ou horizontale si 1.
TGS 1 : vidéo entrelacée (1) ou non entrelacée (Ø).
TGS Ø : 525 lignes (1) ou 625 (Ø).

PAT = | C/R | F | INSERT | M | LB | UB | S |

C/R : bit utilisé avec les bits de poids fort du registre TGS, comme nous venons de le voir.
F : autorisation de clignotement (1) ou inhibition (Ø).
INSERT : deux bits définissant le type d'insertion vidéo. Nous verrons plus loin ce qu'on entend par là, et comment ALICE utilise l'insertion vidéo. Voici les possibilités de ce mode :

PAT 5	PAT 4	**Mode d'insertion**
Ø	Ø	incrustation
Ø	1	écart
1	Ø	forçage
1	1	inhibition

M : le bit M n'est pas utilisé en mode 8Ø caractères par rangée. Il autorise le masquage lorsqu'il est à 1 : chaque caractère dont l'attribut personnel «masquage» est à 1 est représenté à l'écran par un espace.
LB : si LB est à 1, l'affichage du bas de l'écran est autorisé, si il est à Ø, le bas de l'écran prend la couleur de la marge.
UB : même rôle, pour le haut de l'écran.
S : mis à 1, S autorise la rangée de service, à Ø, la rangée de service prend la couleur de la marge.

MAT = | DH | C | CM | MI | MC |

DH : double hauteur. Lorsque ce bit est à 1, la hauteur des caractères est doublée. Évidemment, seule la moitié des caractères apparaît sur l'écran (la taille de l'écran n'est pas changée!).
C : autorise l'utilisation du curseur (1) ou non (Ø).
CM : définit le type du curseur sur deux bits; voici le tableau :

MAT 5	MAT 4	**Type du curseur**
Ø	Ø	curseur fixe, complémente le caractère
1	Ø	curseur clignotant, complémente le caractère
Ø	1	curseur fixe, souligne le caractère
1	1	curseur clignotant, souligne le caractère

MI : incrustation de la marge.
MC : définit la couleur de la marge sur trois bits, par combinaison des trois couleurs de base, Bleu (MAT 2), Vert (MAT 1) et Rouge (MAT Ø). La couleur est présente si le bit est à 1 ; voici les combinaisons :

Codage des couleurs	MAT 2	MAT 1	MAT Ø
noir	Ø	Ø	Ø
rouge	Ø	Ø	1
vert	Ø	1	Ø
jaune	Ø	1	1
bleu	1	Ø	Ø
magenta	1	Ø	1
cyan	1	1	Ø
blanc	1	1	1

Les caractères alphanumériques

Le jeu de caractères alphanumériques est commun aux deux modes, 4Ø et 8Ø caractères par rangée. Il comprend bien entendu les lettres majuscules et les signes de ponctuation.

Vous savez déjà que les caractères minuscules étaient disponibles en mode 8Ø colonnes; ils le sont aussi en mode 4Ø colonnes, mais le BASIC d'ALICE n'utilise pas cette possibilité. Toutefois, on peut facilement y accéder en programmant soi-même les entrées-sorties sur l'écran.

De même, de nombreux caractères accentués, intéressant en particulier les francophones, sont directement affichables, ainsi que quelques caractères spéciaux. En mode 4Ø colonnes, l'ensemble des jeux de caractères forme un ensemble compatible avec les normes Vidéotex en usage en France. Si vous disposez d'un modem, il vous suffit de le raccorder à ALICE et d'écrire son programme de gestion pour pouvoir profiter d'un Minitel intelligent (en ce qui me concerne, c'est déjà fait). Il est probable que MATRA profitera de cette facilité offerte par le 9345.

Le curseur

Si vous avez bien lu les possibilités de la commande MAT, peut-être avez-vous vu une incohérence entre la gestion du curseur par ALICE et les quatre gestions standard du 9345. En effet, pour autoriser les caractères semi-graphiques d'ALICE dans toutes les couleurs disponibles, le curseur affiche alternativement le caractère et un carré dans la couleur courante du curseur puis un carré noir. On peut changer la couleur courante par un appui simultané sur CONTROL et Ø. Pour gérer le curseur ainsi, ALICE n'a pas autorisé le mode curseur. Si ce mode vous intéresse, il suffit de mettre le bit C du registre MAT à 1, puis de choisir le type de curseur qui vous convient le mieux.

Voici un petit programme en BASIC qui vous permettra de comparer les quatre versions existantes (il faut être en mode 40 ou 80 caractères par rangée) :

```
40 A0 = 11*16↓3 + 15*16↓2 + 32 : REM ADRESSE DE R0
50 A1 = A0+1 : REM ADRESSE DE R1
60 R0 = 130 : REM COMMANDE IND POUR LE REGISTRE MAT
70 R1 (0) = 72 : R1(1)=104 : R1(2)=88 : R1(3)=120 : REM
LES 4 TYPES DE CURSEURS DISPONIBLES
80 FOR A = 0 TO 3
90 POKE A1, R1(A) : POKE A0+8, R0 : REM LE +8 COM-
MANDE L'EXECUTION
100 IF INKEY$ = " " THEN 100
110 NEXT A
120 GOTO 80
```

Attention : si vous utilisez ce curseur géré par le 9345, vous ne pouvez plus utilisez le programme d'acquisition de caractères géré par ALICE qui s'appelle KDINP, et qui se trouve à l'adresse $F868. KDINP attend la frappe d'une touche et retourne son code ASCII dans le registre A du 6803. Le problème est qu'il fait clignoter le curseur suivant la technique d'ALICE, ce qui fera apparaître deux curseurs sur les deux emplacements contigus. Utilisez plutôt le programme POLCAT qui se trouve à l'adresse $F883. POLCAT fournit dans le registre A le code ASCII de la touche enfoncée, ou 0 s'il n'y a pas de touche enfoncée. L'attente d'une touche se fait donc grâce à la séquence suivante :

```
SCRUT   JSR   POLCA      ; POLCA = $F883
        BEQ   SCRUT       ; PAS DE TOUCHE
```

De même, en BASIC il est alors préférable d'utiliser INKEY$ plutôt que INPUT.

La gestion du 9345 dans ALICE

Après cet exposé de l'étendue des possibilités du 9345, vous pouvez voir que ce circuit intégré est nettement sous-employé par le BASIC d'ALICE, par souci de simplicité. Cette précaution permettait de respecter deux impératifs :

— tout d'abord conserver une compatibilité maximale avec la première version d'ALICE, pour pouvoir transférer les programmes d'un modèle à l'autre. Or la première version d'ALICE utilisait un processeur de visualisation moins évolué, le MOTOROLA MC 6847;

— ensuite conserver une facilité d'emploi. ALICE se présente tout d'abord comme un micro-ordinateur d'initiation ; l'utilisation de tous les modes du 9345 aurait conduit à une multitude d'instructions spécialisées, alors que le mode 80 colonnes n'est déjà pas d'utilisation immédiate...

La première condition a provoqué l'élimination des caractères minuscules en mode 4Ø colonnes. De même, les caractères accentués ne sont pas utilisables par le BASIC.
La deuxième condition a limité les couleurs de fond : en BASIC le fond est vert, sous éditeur assembleur il est bleu; cela permet d'éviter une confusion en cas de passage accidentel d'un mode à l'autre.

Comme l'éditeur assembleur permet de largement dépasser le stade de l'initiation et d'aborder le domaine des logiciels professionnels, le 9345 permet la création de logiciels de traitement de texte avec ses 8Ø caractères par rangée, ses caractères accentués, de logiciels tableurs ou graphiques. Il permet aussi de créer des jeux rapides et graphiquement évolués.

Pour en revenir à ALICE, voici les valeurs utilisées à l'initialisation du 9345 :

— en mode 4Ø caractères par rangée

R1	RØ+exéc.	Rôle de la commande
$1Ø	$81	Initialisation de TGS : caractéristiques de l'écran + 4Ø caractères
$28	$82	Initialisation de MAT : marge noire + pas de curseur
$67	$83	Initialisation de PAT : autorisation de clignotement sans masquage, affichage sur tout l'écran
$13	$84	Initialisation de DOR : réservation de la mémoire semi-graphique
$Ø8	$87	Initialisation de ROR : visualisation des lignes de caractères dans l'ordre à partir du bloc Ø

— en mode 8Ø caractères par rangée

R1	RØ+exéc.	Rôle de la commande
$DØ	$81	Initialisation de TGS : caractéristiques de l'écran + 8Ø colonnes
$28	$82	Initialisation de MAT : marge noire + pas de curseur
$67	$83	Initialisation de PAT : autorisation de clignotement sans masquage, affichage sur tout l'écran

$FA $84	Initialisation de DOR : caractères noirs sur fond vert
$Ø8 $87	Initialisation de ROR : visualisation des lignes de caractères dans l'ordre à partir du bloc Ø

Cela vous donne une base pour effectuer des essais. En particulier, faites attention au registre TGS, qui peut donner des résultats inattendus s'il est manipulé sans précautions.

Le tableau fournit aussi une autre indication : ALICE exige que le bit d'incrustation i soit mis à 1. Notez aussi que les caractères semi-graphiques que propose ALICE sont différents de ceux du 9345, pour des raisons de compatibilité avec la version précédente. Le jeu semi-graphique d'ALICE est donc défini à l'initialisation dans la mémoire utilisateur du 9345, dans le mode bichrome.

Le partage de la mémoire du 9345 en 2 écrans

Nous avons vu qu'il était possible de partager la mémoire privée du 9345 pour y préparer plusieurs images. Il est alors possible de visualiser l'une ou l'autre de ces images en changeant simplement le contenu du registre indirect ROR. En pratique, dans ALICE, nous allons utiliser cette facilité pour gérer deux écrans différents. En effet les modes que nous utilisons (commandes KRF ou KRL suivant le nombre de caractères par rangée) nécessitent 3 Ko de la mémoire privée du 9345. Comme celle-ci fait 8 Ko en tout, je vous laisse en tirer la conclusion.
Dans cet exemple, nous allons écrire des caractères 'A' dans le premier écran et des caractères 'B' dans le second. Un appui sur une touche du clavier fera passer alternativement d'un écran à l'autre.

```
 1 ;-------------------------------------+
 2 ;     Programmes d'affichage de       !
 3 ;     deux pages:                     !
 4 ;     en page 0 affichage de A        !
 5 ;     en page 1 affichage de B        !
 6 ;-------------------------------------+
 7 ;     Auteur : C. MUSET               !
 8 ;     Date : 2 Decembre 1984          !
 9 ;-------------------------------------+
10
11        ORG   $4C00
12        EXC   DEBUT
13
14 ;1. DEFINITION DES REGISTRES
15 ;--
16 R0     =     $BF20
17 R1     =     $BF21
18 R2     =     $BF22
```

```
 19 R3     =      $BF23
 20 R4     =      $BF24
 21 R5     =      $BF25
 22 R6     =      $BF26
 23 R7     =      $BF27
 24 EXEC   =      $8
 25
 26 TBS    =      $8100
 27 MAT    =      $8200
 28 PAT    =      $8300
 29 DDR    =      $8400
 30 RDR    =      $8700
 31
 32 POLCA =      $F883
 33
 34 ;2. MODE 40 CARACTERES PAR RANGEE
 35 ;--
 36 DEBUT
 37         LDX    #INI40
 38         BSR    INIRG
 39
 40 ;3. AFFICHAGE DE A EN PAGE 0
 41 ;--
 42         LDAA   #$41
 43         STAA   AFF
 44         LDD    #$0
 45         STD    PAGE
 46         BSR    AFFPG
 47
 48 ;4. AFFICHAGE DE B EN PAGE 1
 49 ;--
 50         LDAA   #$42
 51         STAA   AFF
 52         LDD    #$2000
 53         STD    PAGE
 54         BSR    AFFPG
 55
 56 ;5. PASSAGE EN PAGE 1
 57 ;--
 58 BOUCL
 59         BSR    WAIT
 60         LDX    #PASS1
 61         BSR    INIRG
 62         LDD    #$2800
 63         STD    R6
 64
 65 ;6. PASSAGE EN PAGE 0
 66 ;--
 67         BSR    WAIT
 68         LDX    #PASS0
 69         BSR    INIRG
 70         LDD    #$0800
 71         STD    R6
 72
 73 ;7. ON RECOMMENCE
 74 ;--
 75         BRA    BOUCL
 76
 77 ;8. SP INITIALISATION DES REGISTRES IND.
 78 ;--
 79 ;entree X ADRESSE DU TABLEAU
 80 INIRG
 81         LDAB   $0,X   ;LONGUEUR
```

```
 82 INIR1
 83        INX
 84        LDAA   $1,X
 85        STAA   R1
 86        LDAA   $0,X
 87        STAA   R0+EXEC
 88        BSR    BUSY
 89        INX
 90        DECB
 91        BNE    INIR1
 92        RTS
 93
 94 ;9. SP AFFICHAGE DE AFF SUR UNE PAGE
 95 ;--
 96 ;entree AFF VALEUR A AFFICHER
 97 AFFPG
 98        LDD    PAGE   ;NUMERO DE PAGE
 99        STD    R6
100        LDD    #$0107 ;COULEUR ET JEU
101        STD    R2
102        LDAA   #$01   ;40 CAR AVEC INC.
103        STAA   R0
104        LDAA   AFF
105        BSR    WLIG   ;RANGEE 0
106        LDD    PAGE
107        ADDD   #$0800 ;RANGEE 1
108        STD    R6
109        LDAA   AFF
110        BSR    WPG    ;RANGEE 1 A 24
111        RTS
112
113 ;10. SP AFFICHAGE DE A SUR UNE RANGEE
114 ;--
115 ;entree A VALEUR A AFFICHER
116 WLIG
117        LDAB   #$28   ;40 CAR PAR RANGEE
118 WLIG1
119        STAA   R1+EXEC
120        BSR    BUSY
121        DECB
122        BNE    WLIG1
123        RTS
124
125 ;11. SP AFFICHAGE DE A SUR RANGEE 1 A 24
126 ;entree A VALEUR A AFFICHER
127 WPG
128        LDAB   #$18   ;24 RANGEES
129 WPG1
130        PSHB
131        BSR    WLIG
132        INC    R6
133        PULB
134        DECB
135        BNE    WPG1
136        RTS
137
138 ;12. SP BUSY
139 ;--
140 BUSY
141        TST    R0
142        BMI    BUSY
143        RTS
144
```

```
145 ;13. ATTENTE D'UNE TOUCHE
146 ;--
147 WAIT
148         JSR    PDLCA
149         TSTA
150         BEQ    WAIT
151         RTS
152
153 ;14. TABLEAU INIT REGISTRES INDIRECTS
154 ;--
155 INI40
156         DFD    $5
157         DFD    TGS+$10
158         DFD    MAT+$68
159         DFD    PAT+$67
160         DFD    DOR+$13
161         DFD    ROR+$08
162
163 PASS0
164         DFD    $1
165         DFD    ROR+$08
166
167 PASS1
168         DFD    $1
169         DFD    ROR+$28
170
171 ;15. DONNEES DU PROGRAMMES
172 ;--
173 AFF     BLC    $1     ;VALEUR DE R1
174 PAGE    BLC    $2     ;VALEUR DE R6 R7
```

11

Le circuit EF 9345 en mode 8Ø caractères

Vous avez choisi d'afficher 8Ø caractères par rangée : c'est vrai, cela fait plus professionnel. En revanche, la taille des caractères exige l'utilisation d'un téléviseur de bonne qualité ou d'un moniteur. Le jeu des caractères disponibles comprend tous les caractères (lettres, chiffres et signes divers) habituels, ainsi que les minuscules. De plus les caractères accentués usuels sont eux aussi disponibles. Bien sûr, on ne pourra visualiser que trois couleurs simultanément sur l'écran, mais cela suffit bien souvent.

Comment accéder au mode 8Ø caractères par rangée ?

Le moyen le plus naturel d'accéder au mode 8Ø caractères par rangée est d'utiliser la commande BASIC CLS 8Ø (ou CLS 81). Malheureusement, cette commande ne fonctionne qu'en mode direct ; si vous l'utilisez dans un programme, l'exécution de ce programme est interrompue après l'exécution de CLS. Ne vous désolez pas déjà, la limitation est contournable en lançant l'exécution directe du programme qui se trouve dans la mémoire morte d'ALICE.

Pour cela, il faut :
— mettre à Ø le contenu de TECRAN (adresse $3Ø1A = 12314)
— faire appel au programme INASS (adresse $D42C = 54316).
Par exemple, le petit programme BASIC :

```
1Ø POKE 12314,Ø : EXEC 54316
2Ø PRINT "VOUS POUVEZ ECRIRE 8Ø CARACTERES SUR 1 RANGEE"
```

devrait tenir ses promesses. Il vous permet de changer de mode de visualisation en cours de programme.

Attention : lorsque l'on change de mode d'affichage, on ne peut pas éviter l'effacement de la page affichée.

Si vous voulez gérer vous-même le passage en mode 8Ø caractères, par exemple parce que vous voulez choisir la couleur de l'écran et des caractères, alors il va falloir utiliser les registres du boîtier EF 9345.

Sélection du mode 8Ø caractères

Pour sélectionner le mode 8Ø caractères, il faut écrire dans les registres TGS et PAT. Le mode choisi se trouve sélectionné par les bits B7 et B6 de TGS, et B7 de PAT. En mode 8Ø caractères (commandes longues), les bits B7 et B6 de TGS doivent être à 1 et le bit B7 de PAT doit être à Ø. La valeur des autres bits (pour PAT en particulier) varie suivant l'application (bits d'insertion et de masquage). Dans le cas général, on utilise le programme suivant :

```
LDAA #$81       ; INDIRECTION : REGISTRE TGS
LDAB #$DØ       ; MODE 8Ø – LONG
STAB  R1        ; R1 = $BF21
STAA  RØ+EXEC   ; RØ = $BF2Ø, EXEC = $8
BSR   BUSY      ; OBLIGATOIRE APRÈS EXÉCUTION
LDAA #$83       ; INDIRECTION : REGISTRE PAT
LDAB #$67       ; MODE 8Ø – LONG
STAB  R1
STAA  RØ+EXEC
BSR   BUSY
```

Il ne faut pas oublier de demander l'exécution lors de la dernière écriture d'une séquence d'instructions, ni surtout d'appeler le programme BUSY pour se mettre à l'abri des facéties du 9345.
On peut simplifier ce programme en omettant toute la partie relative au registre PAT : dans ALICE, le bit 7 de PAT n'est jamais mis à 1 (ce cas correspond seulement au mode 4Ø caractères par rangée, avec des commandes courtes).

Écriture des caractères dans la mémoire privée du 9345

Il faut ensuite écrire les caractères dans la mémoire privée du 9345. Pour cela, on utilise la commande KRL. Dans le registre RØ, il faut copier l'octet de commande

Ø	1	Ø	1	Ø	ā	Ø	

ā ne doit être mis à 1 que lorsque l'on veut effacer une page entière. Dans ce cas, il ne faut pas tester le bit BUSY, il faut seulement attendre un moment, à l'aide d'une boucle dans le programme, puis arrêter la commande en envoyant une autre commande, comme NOP par exemple.
i est très pratique en mode 8Ø caractères : lorsque ce bit est à 1, l'adresse dans la mémoire privée du EF 9345 est augmentée de 1 après

l'exécution de la commande. Cela permet d'écrire des caractères successifs sans recalculer l'adresse du caractère qui suit. Attention cependant, ce bit n'autorise pas les changements de rangée. Il faut donc écrire $5Ø ou $51 dans le registre RØ si l'on n'utilise pas le bit ā.

On doit fournir l'adresse du caractère à écrire au 9345 par l'intermédiaire des registres R6 et R7. Voici les règles à suivre :
— si l'on veut accéder à la rangée Ø, il faut écrire Ø dans le registre R6;
— si l'on veut accéder à la rangée n ($1 \leqslant n \leqslant 24$), il faut écrire n+7 dans le registre R6. Cela correspond à la règle générale vue au chapitre précédent; par exemple, pour écrire sur la dixième rangée, il faut écrire 17 dans R6;
— si l'on veut accéder à la colonne c de la rangée choisie, deux cas se présentent :
  — pour les colonnes paires (c=Ø, 2, 4,..., 78), il faut écrire c/2 dans R7; pour la quatrième colonne, il faut écrire 2 dans R7.
  — pour les colonnes impaires (c=1, 3, 5,..., 79), il faut écrire (c−1)/2+128 dans R7, c'est-à-dire 128, 129, 13Ø,..., 167. Par exemple, pour la 5e colonne, il faut écrire (5−1)/2+128=13Ø.

Comme vous le voyez, le bit i de la commande KRL est vraiment précieux : il se charge lui-même de calculer R7 après chaque écriture dans la mémoire du EF 9345. Voici toutefois un petit programme qui calcule R6 et R7 à partir du numéro de rangée et du numéro de colonne.

```
; entrée : A = numéro de rangée (de Ø à 24)
;          B = numéro de colonne (de Ø à 79)
         TSTA
         BEQ   CAL1Ø  ;  TEST RANGÉE Ø
         ADDA #$7
CAL1Ø
         STAA  R6
         LSRB          ;  DIVISÉ PAR 2 + BØ → CARRY
         6CC   CAL2Ø   ;  SAUT SI COLONNE PAIRE
         ADDB #$8Ø     ;  POUR COLONNE IMPAIRE
CAL2Ø
         STAB  R7
```

Le caractère proprement dit et ses caractéristiques se trouvent dans les registres R1 et R3. De même que pour les adresses, le contenu du registre R3 varie suivant la position du caractère. En effet, seuls 4 bits sont utilisés. Pour les caractères de rang pair (colonnes Ø, 2, 4, ..., 78), les 4 bits utilisés sont les bits de poids fort (B7, B6, B5 et B4), alors que pour les colonnes impaires (1, 3, 5, ..., 79), il faut écrire dans les 4 bits de poids faible (B3, B2, B1 et BØ).

Pour illustrer rapidement le début de ce chapitre, je vous propose le programme suivant : il affiche sur l'écran le jeu de caractères standard disponibles en mode 8Ø. Ce programme utilise le bit i, donc il

dispense de calculer l'adresse du caractère à afficher. Le numéro de colonne est initialisé à Ø, puis la suite est calculée par le 9345.

```
 1 ;-----------------------------------------+
 2 ;     Programme d'affichage des           !
 3 ;    caracteres disponibles dans          !
 4 ;  le mode 80 caracteres par rangee        !
 5 ;-----------------------------------------+
 6 ;     Auteur : C. MUSET                   !
 7 ;     Date :   23 Novembre 1984           !
 8 ;-----------------------------------------+
 9
10        ORG   $8000
11        EXC   AF80
12
13 ;1.    Definition des registres du 9345 :
14 ;--
15 R0     = $BF20
16 R1     = R0+$1
17 R2     = R1+$1
18 R3     = R2+$1
19 R4     = R3+$1
20 R5     = R4+$1
21 R6     = R5+$1
22 R7     = R6+$1
23 EXEC   = $0
24
25 ;2.    Effacement de l'ecran :
26 ;--
27 AF80
28        BSR   BUSY
29        LDD   #$81D0
30        STAB  R1
31        STAA  R0+EXEC
32        BSR   BUSY
33        LDD   #$84FF
34        STAB  R1
35        STAA  R0+EXEC
36        BSR   BUSY
37        LDD   #$5120
38        STD   R0
39        LDD   #$2000
40        STD   R2
41        CLRA                  ;1ERE RANGEE
42        BSR   EFLIS
43        LDAA  #$08
44 EF10
45        BSR   EFLI0
46        INCA
47        CMPA  #$20
48        BNE   EF10
49
50 ;3.    Affichage de tous les caracteres du jeu G0
51 ;--
52 JEUG0
53        LDAA  #$0B            ;COULEUR : BLANC ET NOIR
54        STAA  R3
55        LDAA  #$51            ;80 C/R + AUTO-INCREMENTATION
56        BTAA  R0
57
58        CLR   CARAC
59        LDAA  #$10            ;NOMBRE DE RANGEES
```

```
60        LDAB  #$08          ;1ERE RANGEE UTILISEE
61 WSP10
62        STAB  R6
63        PSHB
64        PSHA
65        CLRA
66        STAA  R7
67        LDAB  CARAC
68        LDAA  #$08          ;NOMBRE DE COLONNES
69 WSP20
70        STAB  R1+EXEC
71        BSR   BUSY
72        ADDB  #$10          ;CARACTERE SUIVANT
73        DECA
74        BNE   WSP20
75        INC   CARAC
76        PULA                ;NOMBRE DE RANGEES
77        PULB
78        INCB                ;1ER CARACTERE
79        DECA
80        BNE   WSP10
81 FIN
82        BRA   FIN
83
84 ;4.   Programme d'attente du bit BUSY :
85 ;--
86 BUSY
87        TST   R0
88        BMI   BUSY
89        RTS
90
91 ;5.   Effacement d'une ligne :
92 ;--
93 ;ENTREE : A = NUMERO DE RANSEE
94 ;SORTIE : A INCHANGE
95 EFLIG
96        CLR   R7
97        LDAB  #$50
98 EFL10
99        STAA  R6+EXEC
100       BBR   BUSY
101       DECB
102       BNE   EFL10
103       RTS
104
105 ;6.   Donnee du programme :
106 CARAC BLC   $1    ;1ER CARACTERE DE LA RANGEE
107
108 ;7.   Fin
```

Couleurs et attributs vidéo en 8Ø caractères

On peut afficher trois couleurs simultanément sur l'écran en mode 8Ø caractères par rangée. Ces trois couleurs se divisent en deux groupes.

— **La couleur de marge,** qui peut être choisie parmi les huit couleurs suivantes : noir, rouge, vert, jaune, bleu, magenta, cyan et blanc.
On choisit la couleur de marge en utilisant le registre MAT ; les trois bits de poids faible contiennent la couleur, comme nous l'avons vu au chapitre 1Ø.

— **Les deux couleurs actives,** c'est-à-dire les couleurs utilisées pour le fond et l'écriture sur l'écran. En principe, ces deux couleurs devraient être choisies parmi les huit couleurs que nous venons de citer. En fait, lorsqu'on utilise ALICE 32 ou ALICE 9Ø avec son connecteur cerclé de rouge branché sur le téléviseur (pour inhiber les possibilités d'incrustation), on peut obtenir huit nouvelles couleurs. On obtient ces couleurs en jouant sur la valeur du bit d'incrustation. Ce bit peut être ajouté aux trois bits de couleur : lorsqu'il vaut 1, on obtient les couleurs habituelles, énumérées ci-dessus ; lorsque le bit d'incrustation vaut Ø, on obtient de nouvelles nuances. On utilise par exemple cette possibilité pour obtenir de l'orange sur ALICE. Toutefois, ces couleurs ne sont pas toujours bien discernables, cela dépend du réglage de votre téléviseur. Chacune de ces couleurs est donc codée sur 4 bits. Les 8 bits nécessaires pour les deux couleurs sont écrits dans le registre DOR, suivant le format décrit au chapitre 1Ø.

Vous trouverez dans le chapitre relatif au mode 4Ø caractères par rangée un petit programme qui vous permettra d'afficher les 16 couleurs simultanément sur votre écran.
Souvenez-vous que le mode 8Ø caractères ne permet d'afficher que trois couleurs simultanément sur l'écran.

Nota : on peut obtenir les mêmes seize couleurs pour la couleur de marge, mais on obtient des couleurs composites, inexpliquées dans le manuel Thomson relatif au 9345. Pour tester les effets, vous devez aussi utiliser le bit d'insertion, mais cette fois dans le registre MAT qui définit la couleur de marge.

Comment afficher un caractère alphanumérique dans une couleur donnée ?

Pour afficher un caractère dans une couleur donnée :

— il faut choisir son code (compris entre $Ø et $7F) dans la table du jeu de caractères, et copier ce code dans le registre R1 ;

— il faut ensuite choisir la couleur du caractère et du fond sur lequel on va l'écrire parmi les trois couleurs disponibles (couleur de marge, première couleur active et deuxième couleur active). Quatre possibilités sont offertes, ce qui correspond à la sélection de deux bits, appelés N et D.

Couleur du caractère	Couleur du fond	Valeur de D	Valeur de N
1re couleur active	couleur de marge	Ø	Ø
couleur de marge	1re couleur active	Ø	1
2e couleur active	couleur de marge	1	Ø
couleur de marge	2e couleur active	1	1

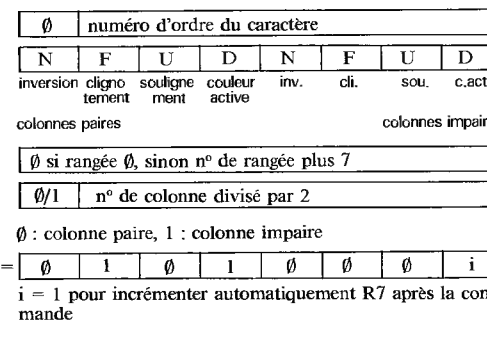
En fait, le bit D choisit la couleur active, alors que le bit N produit l'inversion vidéo.
Deux attributs sont disponibles pour le caractère : le clignotement (bit F) et le soulignement (bit U).

Cela donne un ensemble de quatre bits, que l'on groupe dans l'ordre NFUD. Ces quatre bits doivent être écrits dans le registre R3 :

— sur les quatre bits de poids fort pour les caractères de rang pair (colonnes Ø, 2, 4, ... 78);

— sur les quatre bits de poids faible pour les caractères de rang impair (colonnes 1, 3, 5, ... 79).

En résumé, pour un caractère alphanumérique, on utilisera toutes les possibilités du 9345 en positionnant les registres de la manière suivante :



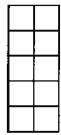
i = 1 pour incrémenter automatiquement R7 après la commande

Rappelez-vous qu'il ne faut pas écraser les caractéristiques de la colonne impaire lorsque vous écrivez en colonne paire et inversement.

Le mode semi-graphique en 8Ø caractères par rangée

On peut aussi utiliser des caractères semi-graphiques dans le mode 8Ø caractères par rangée. Pour cela, on définit l'emplacement sur l'écran selon la même procédure que pour les caractères alphanumériques. Le calcul des registres R6 et R7 reste donc le même.

Il faut ensuite définir le caractère. La taille est la même que celle d'un caractère ordinaire, c'est-à-dire 1Ø lignes de 4 colonnes. Mais vous ne pouvez pas définir chaque point : votre définition couvre 5 lignes de 2 colonnes. Cela donne le quadrillage suivant :



Chacun des points du quadrillage peut prendre soit la couleur de la marge, soit l'une des deux couleurs actives (mais un même caractère ne peut utiliser qu'une couleur active, le mélange n'est pas possible). La couleur active est indiquée par le bit D (le même que pour un caractère alphanumérique). Si D est Ø, vous avez la première couleur active, s'il est à 1, la seconde.

Dessinez votre caractère : les points dans la couleur de marge auront la valeur Ø, et les points dans la couleur active auront la valeur 1. Ainsi, par exemple, le caractère

aura comme image

1	Ø
Ø	1
Ø	1
1	Ø
1	1

Ces valeurs doivent ensuite être inscrites dans les registres du 9345. Pour cela on utilise l'équivalence suivante entre les points du caractère et les bits des registres R1 et R3 :

C0	C1
C2	C3
C4	C5
C6	A1
A2	A3

Les bits CØ, C1,..., C6 vont s'écrire dans le registre R1. De plus, pour indiquer que l'on a affaire à un caractère semi-graphique, on met le bit de poids fort de R1 à 1. Cela donne :

R1 =	1	C6	C5	C4	C3	C2	C1	CØ

Dans notre exemple, on aura donc R1 = %111Ø1ØØ1 = $E9.

Les trois autres bits (A1, A2, A3) sont groupés avec le bit D (la couleur active). Ces quatre bits sont écrits dans le registre R3 suivant la même règle que pour les caractères alphanumériques :

— dans le quartet de poids fort de R3 pour les caractères de rang pair (l'autre quartet ne doit pas être modifié);

— dans le quartet de poids faible pour les caractères de rang impair (l'autre quartet ne doit pas être modifié).

Colonnes Ø, 2, 4, ...78	R3 =	A3	A2	A1	D	—	—	—	—

Colonnes 1, 3, 5. ...79	R3 =	—	—	—	—	A3	A2	A1	D

Ce mode vous donne la même définition (16Ø pixels dans la largeur de l'écran, 125 pixels dans la hauteur) que les modes CLS 8Ø et CLS 81 du BASIC, comme il est normal : le mode CLS 81 représente l'utilisation normale du 9345 en 8Ø caractères par rangée; le mode CLS 8Ø utilise systématiquement l'inversion vidéo, en mettant toujours le bit N de R3 à 1 pour les caractères alphanumériques. La commande SET* va changer les couleurs utilisables dans les registres MAT et DOR.

Comme vous le voyez, la justification de ces deux instructions du BASIC, dont la syntaxe pouvait paraître compliquée, réside dans les règles d'utilisation du 9345.

La technique du 'bit-map' (mémoire d'écran 1 bit/1 pixel)

L'un des avantages du mode 80 caractères en mode semi-graphique est que chacun des 20000 points de l'écran est défini par un bit de la mémoire privée du 9345 : c'est ce qu'on appelle la technique *"bit map"*.

Si l'on dispose d'un programme capable de mettre ce bit à 1 ou 0 à partir de l'adresse du point sur l'écran, il sera facile d'exécuter n'importe quel dessin point par point. Bien entendu, il faudra tenir compte du fait que les deux couleurs actives ne peuvent pas être mélangées dans le même caractère.

Dans le programme qui suit, vous trouverez un sous-programme qui effectue le calcul de l'adresse du point sur l'écran, et qui calcule aussi les valeurs à inscrire dans les registres R1 et R3.

Voyons d'abord le mode d'emploi du programme :
— le programme commence par effacer l'écran ;
— il permet ensuite de dessiner sur l'écran point par point :
pour cela, vous pouvez choisir la couleur en appuyant sur la touche 0 pour la couleur de marge, ou sur les touches 1 ou 2 pour les deux couleurs actives ;
— pour déplacer le curseur sur l'écran, vous pouvez utiliser les touches W, Z, S et Q (on peut obtenir la répétition automatique en appuyant sur SHIFT + espace, et l'arrêter en appuyant sur CONTROL + espace).

Seules les parties 7 et 8 méritent d'être détaillées. Elles calculent les valeurs à écrire dans les registres R1, R3, R6 et R7.

• Le programme CALCU commence par calculer la position du caractère sur l'écran. C'est relativement facile : il suffit de diviser l'abscisse du point par 2 pour obtenir le numéro de la colonne. Le numéro de la rangée est obtenu en divisant l'ordonnée du point par 5 (l'abscisse varie de 0 à 159, l'ordonnée de 0 à 124, l'origine 0, 0 se trouve en haut à gauche de l'écran). On applique ensuite le calcul que nous avons vu précédemment dans ce chapitre pour calculer R6 et R7.
Le reste des deux divisions est ensuite utilisé pour calculer les masques que nous appliquerons à R1 et R3. Ces masques peuvent prendre deux aspects :
— Si la couleur de marge est utilisée, le programme met à zéro le bit correspondant, alors que tous les autres bits sont mis à 1.
— Si une couleur active est choisie, le programme met à 1 le bit correspondant, met le bit de couleur à 1 si c'est la deuxième couleur active qui est choisie, et met tous les autres bits à 0.

• **Le programme AFFIC** utilise ensuite ces données. Il lit le contenu des registres R1 et R3. Si l'on n'a pas affaire à un caractère semi-graphique, on force le contenu pour obtenir un semi-graphique de la couleur de la marge, en distinguant le cas des caractères qui se trouvent sur les colonnes paires de celui des caractères des colonnes impaires, à cause du registre R3.
Ensuite, suivant que l'on se trouve en couleur de marge (auquel cas on doit mettre un bit à Ø) ou en couleur active (auquel cas il faut mettre un bit à 1 et écrire la couleur du caractère), il faut utiliser soit l'instruction AND, soit l'instruction ORA pour calculer les valeurs des registres R1 et R3. Il suffit ensuite d'exécuter la commande KRL pour afficher le nouveau caractère.

```
 1 ;-----------------------------------------+
 2 ;     Programme d'affichage en mode       !
 3 ;semi-graphique , bit-map 80 caracteres!
 4 ;          par rangee .                   !
 5 ;-----------------------------------------+
 6 ;     Auteur : C. Muset   J-F Sallet      !
 7 ;     Date :   3 Janvier 1985             !
 8 ;-----------------------------------------+
 9
10        ORG    $4E00
11        EXC    BITMP
12
13 ;1.    Definition des registres du 9345 :
14 ;--
15 TECRA = $301A       ;TYPE DE L'ECRAN
16 ROLTB = $3231       ;TABLEAU MEMORISANT L'ETAT DES TOUCHES
17 INASS = $D42C       ;INITIALISATION DU TYPE DE L'ECRAN
18 POLCA = $F0B3       ;SCRUTATION DU CLAVIER
19 R0    = $BF20       ;REGISTRES DU 9345
20 R1    = R0+$1
21 R2    = R1+$1
22 R3    = R2+$1
23 R4    = R3+$1
24 R5    = R4+$1
25 R6    = R5+$1
26 R7    = R6+$1
27 EXEC  = $8
28
29 ;2.    Effacement de l'ecran :
30 ;--
31 BITMP
32        CLR    TECRA
33        JSR    INASS
34
35 ;3.    Acquisition d'une touche du clavier :
36 ;--
37 ACCLV
38        TST    REPET          ;REPETITION ?
39        BEQ    RPFIN
40        LDX    #ROLTB
41        LDAA   #$FF
42 RP010
43        STAA   $0,X
44        INX
45        CPX    #ROLTB+$8
```

```
46          BNE    RPo10
47  RPFIN
48          JSR    POLCA
49          BEQ    ACCLV
50          CMPA   ##5A          ;TOUCHE 'Z'
51          BNE    ACC10
52          BSR    DECPY
53          BRA    ACCLV
54  ACC10
55          CMPA   ##57          ;TOUCHE 'W'
56          BNE    ACC20
57          BSR    INCPY
58          BRA    ACCLV
59  ACC20
60          CMPA   ##51          ;TOUCHE 'Q'
61          BNE    ACC30
62          BSR    DECPX
63          BRA    ACCLV
64  ACC30
65          CMPA   ##53          ;TOUCHE 'S'
66          BNE    ACC40
67          BSR    INCPX
68          BRA    ACCLV
69  ACC40
70          CMPA   ##07          ;TOUCHE 'CTRL ESP'
71          BNE    ACC50
72          BSR    REPFI
73          BRA    ACCLV
74  ACC50
75          CMPA   ##06          ;TOUCHE 'SHIFT ESP'
76          BNE    ACC60
77          BSR    REPDB
78          BRA    ACCLV
79  ACC60
80          CMPA   ##30          ;TOUCHE '0'
81          BNE    ACC70
82          BSR    COMAR
83          BRA    ACCLV
84  ACC70
85          CMPA   ##31          ;TOUCHE '1'
86          BNE    ACC80
87          BSR    COUL1
88          BRA    ACCLV
89  ACC80
90          CMPA   ##32          ;TOUCHE '2'
91          BNE    ACC90
92          BSR    COUL2
93  ACC90
94          BRA    ACCLV
95
96  ;4.     Touches de repetition :
97  ;--
98  REPFI
99          CLR    REPET
100         RTS
101 REPDB
102         LDAA   ##1
103         STAA   REPET
104         RTS
105
106 ;5.     Changement de couleur :
107 ;--
108 COMAR
```

```
109         LDAA  #$01
110         STAA  MARGE
111         CLR   COLOR
112         RTS
113 COUL1
114         CLR   MARGE
115         CLR   COLOR
116         RTS
117 COUL2
118         CLR   MARGE
119         LDAA  #$11
120         STAA  COLOR
121         RTS
122
123 ;6.    Deplacement du curseur :
124 ;--
125 DECPY                           ;DEPLACEMENT VERS LE HAUT
126         TST   PY
127         BNE   DPY10
128         RTS
129 DPY10
130         DEC   PY
131         BRA   TRACE
132 INCPY                           ;DEPLACEMENT VERS LE BAS
133         LDAA  PY
134         CMPA  #$7C              ;PY < 125
135         BNE   IPY10
136         RTS
137 IPY10
138         INC   PY
139         BRA   TRACE
140 DECPX                           ;DEPLACEMENT VERS LA GAUCHE
141         TST   PX
142         BNE   DPX10
143         RTS
144 DPX10
145         DEC   PX
146         BRA   TRACE
147 INCPX                           ;DEPLACEMENT VERS LA DROITE
148         LDAA  PX
149         CMPA  #$9F              ;PX < 160
150         BNE   IPX10
151         RTS
152 IPX10
153         INC   PX
154 TRACE
155         BSR   CALCU             ;CALCUL DE LA POSITION DU CARACTERE
156         BSR   AFFIC             ;AFFICHAGE DU POINT
157         RTS
158
159 ;7.    Calcul de la position du caractere :
160 ;--
161 ;Entree : PX = Abscisse entre 0 et 159
162 ;         PY = Ordonnee entre 0 et 124
163 ;Sortie : Valeurs dans les registres R6 et R7 du 9345
164 ;         Masque dans les mots MEMOA et MEMOB
165   ;7.1 Calcul de R7 :
166 CALCU
167         LDAA  PX
168         CLRB
169         LSRA
170         ROLB                    ;CONSERVATION DU RESTE
171         PSHB
```

```
172         LSRA
173         BCC    CAL10        ;CALCUL DU BIT B7 DE R7
174         ADDA   #$80
175 CAL10
176         STAA   R7
177    ;7.2 Calcul de R6 :
178         LDAB   PY
179         CLRA                ;INIT DU DIVIDENDE
180 CAL20
181         CMPB   #$5          ;DIVISION PAR 5
182         BLO    CAL30
183         SUBB   #$5
184         INCA
185         BRA    CAL20
186 CAL30
187         TSTA                ;TEST DE LA RANGEE 0
188         BEQ    CAL40
189         ADDA   #$7
190 CAL40
191         STAA   R6
192    ;7.3 Calcul des masques pour R1 et R3 :
193         PULA
194         ASLB
195         ABA
196         TAB
197         LDX    #TBMSQ
198         ABX
199         ABX                 ;2 OCTETS PAR MASQUE
200         LDD    $0,X
201         ORAB   COLOR
202         TST    MARGE
203         BEQ    CAL60        ;PAS LA COULEUR DE MARGE
204         COMA
205         COMB
206         TST    R7           ;COLONNE PAIRE OU IMPAIRE ?
207         BMI    CAL50        ;COLONNE IMPAIRE
208         ORAB   #$0F
209         BRA    CAL80
210 CAL50
211         ORAB   #$F0
212         BRA    CAL80
213 CAL60
214         TST    R7           ;COLONNE PAIRE OU IMPAIRE ?
215         BMI    CAL70        ;COLONNE IMPAIRE
216         ANDB   #$F0
217         BRA    CAL80
218 CAL70
219         ANDB   #$0F
220 CAL80
221         STD    MEMDA
222         RTS
223
224 ;8.   Ecriture dans la memoire du 9345 :
225 ;--
226 AFFIC
227         LDAA   #$58         ;COMMANDE KRL - LECTURE
228         STAA   RO+EXEC
229         JSR    BUSY
230         LDAA   R1
231         LDAB   R3
232         TST    R7           ;COLONNE PAIRE ?
233         BMI    IMPAI        ;TRAITEMENT DES COLONNES IMPAIRES
234         TSTA
```

```
235         BMI   GRAIM        ;GRAPHISME
236         LDAA  ##80
237         ANDB  ##0F
238         BRA   GRAIM
239 IMPAI
240         TSTA
241         BMI   GRAIM
242         LDAA  ##80
243         ANDB  ##F0
244 GRAIM
245         TST   MARGE
246         BEQ   IMACT
247         ANDA  MEMOA
248         ANDB  MEMOB
249         BRA   AFFRT
250 IMACT
251         ANDB  ##EE         ;SUPPRIME LA COULEUR PRECEDENTE
252         ORAA  MEMOA
253         ORAB  MEMOB
254 AFFRT
255         STAA  R1
256         STAB  R3
257         LDAA  ##50         ;COMMANDE KRL - ECRITURE
258         STAA  R0+EXEC
259         BSR   BUSY
260         RTS
261
262 ;9.     Programme de test du bit busy :
263 ;---
264 BUSY
265         TST   R0
266         BMI   BUSY
267         RTS
268
269 ;10.    Donnees du programme :
270 ;----
271 TBMSQ
272         DFD   $0100        ;BIT C0
273         DFD   $0200        ;     C1
274         DFD   $0400        ;     C2
275         DFD   $0800        ;     C3
276         DFD   $1000        ;     C4
277         DFD   $2000        ;     C5
278         DFD   $4000        ;     C6
279         DFD   $0022        ;     A1
280         DFD   $0044        ;     A2
281         DFD   $0088        ;     A3
282
283 MEMOA DFO   $0             ;MASQUES POUR R1 ET R3
284 MEMOB DFO   $0
285
286 PX    DFO   $0             ;POSITION SUR L'ECRAN
287 PY    DFO   $0
288 COLOR DFO   $0             ;COULEUR ACTIVE
289 MARGE DFO   $0             ;1 SI COULEUR DE MARGE
290
291 REPET DFO   $0             ;INDICATEUR DE REPETITION
292
293 ;11.    Fin
```

12

Le EF 9345 en mode 4Ø caractères

L'utilisation du EF 9345 en mode 4Ø caractères offre le plus large éventail de possibilités : son jeu de caractères alphanumériques est plus étendu, il possède des caractères semi-graphiques prédéfinis, et laisse à l'utilisateur la possibilité de définir ses propres caractères. Bien entendu, ces avantages sont contrebalancés par un inconvénient : ces possibilités accrues exigent des programmes plus compliqués lorsque l'on veut en profiter pleinement.

Accès au mode 4Ø caractères par rangée

De même que pour le mode 8Ø caractères, CLS 4Ø ne peut être utilisé qu'en mode commande, pas en mode programme. Il faut donc, en mode programme, utiliser une solution analogue à celle vue dans le chapitre précédent. Voici donc le moyen de changer le mode du 9345 en BASIC.

1. Mettre à 1 (pour 4Ø colonnes par rangée) le contenu de TECRAN (adresse $3Ø1A= 12314 décimal).

2. Faire appel au programme INASS (adresse $D42C = 54316 décimal).

Vous pouvez écrire en BASIC :

```
1Ø INPUT "MODE CHOISI (Ø = 8Ø CARACTÈRES,
1 = 4Ø CARACTÈRES)"; A
2Ø POKE 12314, A : EXEC 54316
3Ø GOTO 1Ø
```

Ces adresses sont évidemment exploitables dans un programme en assembleur. Mais vous pouvez aussi gérer vous-même directement le passage en programmant les registres du 9345.

1. Sélection du mode 4Ø caractères :
Comme dans le mode 8Ø caractères, il faut écrire dans les registres TGS et PAT. Les bits B7 et B6 de TGS doivent être mis à Ø, ainsi que le bit B7 du registre PAT.
Voici un exemple de programme pour utiliser le mode 4Ø caractères dans le cas général :

```
LDAA   #$81      ; INDIRECTION : REGISTRE TGS
LDAB   #$1Ø      ; MODE 4Ø LONG
STAB   R1        ; R1 = $8F21
STAA   RØ+EXEC   ; RØ = $8F2Ø, EXEC = $8
BSR    BUSY      ; EXÉCUTION COMPLÈTE !
LDAA   #$83      ; INDIRECTION : REGISTRE PAT
LDAB   #$67      ; MODE 8Ø LONG
STAB   R1
STAA   RØ+EXEC
BSR    BUSY
```

Rappel : le registre PAT n'a pas à être réécrit dans les cas normaux. ALICE ne met jamais le bit B7 à 1, qui correspondrait aux commandes courtes du mode 4Ø caractères, qui ne sont pas utilisées.

2. Écriture des caractères dans la mémoire privée du 9345 :
Nous utiliserons ici la commande la plus complète, KRF, pour écrire les caractères dans la mémoire privée du 9345. Comme toutes les commandes, KRF sera écrit dans le registre RØ. En voici le contenu :

Ø Ø Ø Ø Ø $\bar{a}$ Ø i

L'utilisation des bits $\bar{a}$ et i est identique au mode 8Ø caractères : $\bar{a}$ est utilisé pour répéter la commande sur la page entière, par exemple pour l'effacer, si il est mis à 1. L'arrêt de la commande se fait en envoyant une autre commande au 9345 (NOP par exemple) après avoir attendu suffisamment longtemps pour l'opération voulue. On peut utiliser une petite boucle dans le programme par exemple.
i, lorsqu'il est à 1, provoque l'incrémentation du pointeur de gestion de la mémoire privée du 9345. Cette incrémentation n'est effective que sur une rangée : il faut calculer les pointeurs à chaque changement de ligne.

3. Le calcul de l'adresse du caractère à écrire :
Le calcul de l'adresse du caractère à écrire est plus simple que dans le mode 8Ø caractères. Il suit la règle générale, c'est-à-dire :
— R6 vaut Ø pour la rangée Ø, ou n+7 pour la rangée n ($1 \leqslant n \leqslant 24$).
— R7 est égal au numéro de la colonne, donc entre Ø et 39.

Le petit programme suivant calcule les valeurs à inscrire dans les registres R6 et R7 à partir des numéros de rangée et de colonne :

```
        LDAA  RANGE       ; NUMÉRO DE RANGÉE
        BEQ   CAL1Ø
        ADDA  #$7         ; POUR RANGÉES 1 A 24
CAL 1Ø
        STAA  R6
        LDAA  COLON       ; NUMÉRO DE COLONNE
        STAA  R7
```

4. Le code du caractère et ses attributs :
Il faut aussi écrire le code du caractère choisi et les attributs qui lui sont relatifs dans les registres R1, R2 et R3. Nous verrons plus loin toutes les possibilités disponibles pour ces registres.
Nous vous proposons ici comme exemple un programme qui affiche lui aussi sur l'écran le jeu de caractères standard appelé jeu GØ, disponible en mode 4Ø caractères.

Il faut adapter la directive ORG à la taille mémoire de votre ALICE : ORG $4DØØ pour ALICE 32, et ORG $ADØØ pour ALICE 9Ø.

```
 1 ;-----------------------------------------+
 2 ;     Programme d'affichage des           !
 3 ;   caracteres disponibles dans           !
 4 ;  le mode 40 caracteres par rangee       !
 5 ;-----------------------------------------+
 6 ;     Auteur : C. MUSET                   !
 7 ;     Date :   23 Novembre 1984           !
 8 ;-----------------------------------------+
 9
10       ORG   $8000
11       EXC   AF40
12
13 ;1.   Definition des registres du 9345 :
14 ;--
15 R0    = $BF20
16 R1    = R0+$1
17 R2    = R1+$1
18 R3    = R2+$1
19 R4    = R3+$1
20 R5    = R4+$1
21 R6    = R5+$1
22 R7    = R6+$1
23 EXEC  = $8
24
25 ;2.   Effacement de l'ecran :
26 ;--
27 AF40
28       BSR   BUSY
29       LDD   #$8110      ;REGISTRE TGS
30       STAB  R1
31       STAA  R0+EXEC
32       BSR   BUSY
33       LDD   #$8413      ;REGISTRE DOR
34       STAB  R1
35       STAA  R0+EXEC
```

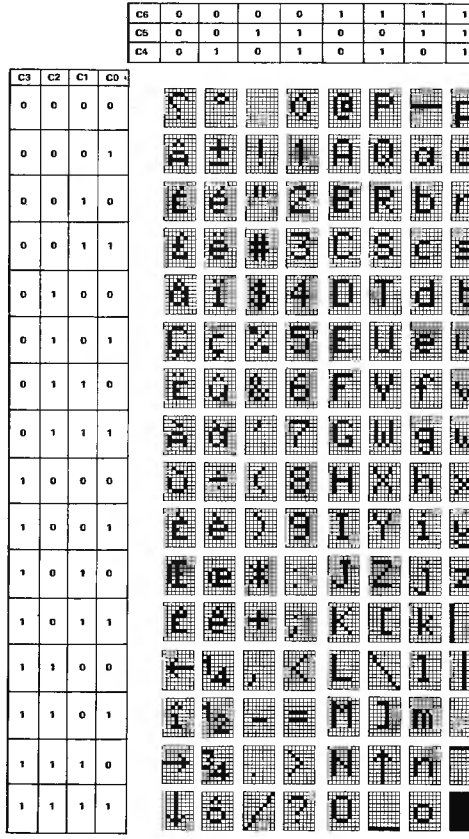
```
36          BSR   BUSY
37          LDD   #$0120        ;ON ECRIT DES ESPACES
38          STD   R0
39          LDD   #$0100        ;ATTRIBUTS+COULEUR
40          STD   R2
41          CLRA                ;1ERE RANGEE
42          BSR   EFLIG         ;EFFACEMENT D'UNE LIGNE
43          LDAA  #$8           ;NUMERO DE LA 2EME RANGEE
44 EF10
45          BSR   EFLIG
46          INCA
47          CMPA  #$20          ;DERNIERE LIGNE ??
48          BNE   EF10
49
50 ;3.     Affichage de tous les caracteres du jeu G0
51 ;--
52 JEUG0
53          LDAA  #$01          ;JEU DE CARACTERES G0
54          STAA  R2
55          LDAA  #$07          ;COULEUR : BLANC ET NOIR
56          STAA  R3
57          LDAA  #$01          ;40 C/R + AUTO-INCREMENTATION
58          STAA  R0
59
60          CLR   CARAC
61          LDAA  #$10          ;NOMBRE DE RANGEES
62          LDAB  #$08          ;1ERE RANGEE UTILISEE
63 WSP10
64          STAB  R6
65          PSHB
66          PSHA
67          CLRA
68          STAA  R7
69          LDAB  CARAC
70          LDAA  #$08          ;NOMBRE DE COLONNES
71 WSP20
72          STAB  R1+EXEC
73          BSR   BUSY
74          ADDB  #$10          ;CARACTERE SUIVANT
75          DECA
76          BNE   WSP20
77          INC   CARAC
78          PULA                ;NOMBRE DE RANGEES
79          PULB
80          INCB                ;1ER CARACTERE
81          DECA
82          BNE   WSP10
83 FIN
84          BRA   FIN
85 ;4.     Programme d'attente du bit BUSY :
86 ;--
87 BUSY
88          TST   R0
89          BMI   BUSY
90          RTS
91
92 ;5.     Effacement d'une ligne :
93 ;--
94 ;ENTREE : NUMERO DE LA RANGEE (SUIVANT L'ORDRE DU 9345)
95 ;SORTIE : A INCHANGE
96 EFLIG
97          CLR   R7
98          LDAB  #$28          ;NOMBRE DE CARACTERES PAR RANGEE
```

```
 99 EFL10
100         STAA   R6+EXEC
101         BSR    BUSY
102         DECB
103         BNE    EFL10
104         RTS
105
106 ;6.     Donnee du programme :
107 CARAC BLC    $1      ;1ER CARACTERE DE LA RANGEE
108
109 ;7.     Fin
```

Les jeux de caractères en mode 4Ø caractères

Les caractères que vous venez de visualiser grâce au programme précédent constituent le jeu de caractères de base contenu dans le 9345. Ce jeu s'appelle GØ. Il comprend tous les signes alphanumériques usuels (en tout 128 signes) :
— lettres majuscules
— lettres minuscules
— chiffres
— ponctuation
— signes divers tels $, ↕. %, etc.
Quelques signes moins courants sont aussi disponibles :
— minuscules accentuées, et quelques majuscules accentuées (ce circuit est de conception française!)
— le c cédille majuscule et minuscule, Œ et œ
— les fractions 1/4, 1/2, 3/4.
Cet ensemble de caractères n'est utilisé qu'en partie par le BASIC d'ALICE, toujours pour des raisons de compatibilité avec la première version d'ALICE.

On peut aussi accéder à un autre jeu de caractères du 9345, le jeu de caractères semi-graphiques G1Ø. Ce jeu contient lui-aussi 128 caractères. Chaque caractère comprend deux parties, le fond et le premier plan. Nous allons voir une méthode pour calculer le code d'un caractère semi- graphique. Pour cela, il faut distinguer deux types de caractères semi-graphiques (chaque type contient 64 caractères) :
— les mosaïques, qui se dessinent suivant le schéma suivant :



Les jeux de caractères en mode 4Ø caractères

Les mosaïques sont formées de six cases jointives ; à chaque case est associé un bit :

Numéro du bit :

Ø	1
2	3
4	5

Un bit à Ø indique que la case prend la couleur du fond, un bit à 1 que la case fait partie du premier plan, par exemple :

=

1	1
Ø	1
1	Ø

= Ø11Ø11

Pour indiquer qu'il s'agit d'une mosaïque, le bit 6 est mis à 1. Le bit de poids fort reste indéfini ; on le met en général à Ø. On obtient donc le caractère %Ø1Ø11Ø11 = $5B.

— les caractères dits «séparés». Ces caractères s'inscrivent dans le dessin suivant :

Les caractères séparés sont formés sur six cases séparées par des zones que vous ne contrôlez pas : ces zones gardent toujours la couleur du fond. Par ailleurs la définition du caractère est identique : chaque case peut prendre soit la couleur du fond. soit celle du premier plan. Le codage du caractère est le même. La seule différence est que le bit B6 doit être à Ø pour indiquer qu'il s'agit d'un caractère séparé. Par exemple

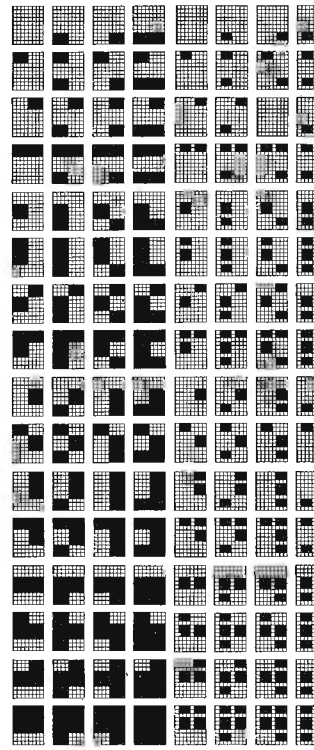
sera codé %ØØØ11Ø11 = $1B.
Voici ci-contre la liste des caractères semi-graphiques du 9345.

Il existe un troisième jeu de caractères semi-graphiques sur le 9345, appelé G11. Il ne contient que 32 caractères, qui forment les diagonales et les médianes du rectangle de base du caractère.

Semi-graphique MOSAÏQUE — Semi-graphique SÉPARÉ

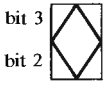
C6	1	1	1	1	0	0	0	0
C5	0	0	1	1	0	0	1	1
C4	0	1	0	1	0	1	0	1

C3	C2	C1	C0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1



Caractères semi-graphiques

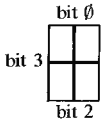
Pour les diagonales, chaque bit est associé à une diagonale donnée :



Les bits 4 et 5 doivent être à Ø, les bits 6 et 7 sont indéfinis (en général, on préfère les mettre à Ø).
Ainsi, par exemple :

donnera %ØØØØ1ØØ1 = $Ø9.

Pour les médianes, le principe est le même, en utilisant la figure suivante :

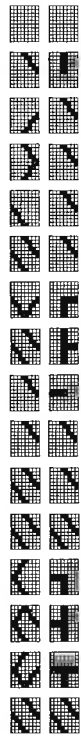


Dans ce cas, le bit 4 est à 1, le bit 5 à Ø et les bits 6 et 7 indéfinis (donc à Ø en général). Par exemple :

donnera le code %ØØ11ØØ1 = $19.

Voici la liste de ces 32 caractères semi- graphiques :

				C5	0	0
				C4	0	1
C3	C2	C1	C0			
0	0	0	0			
0	0	0	1			
0	0	1	0			
0	0	1	1			
0	1	0	0			
0	1	0	1			
0	1	1	0			
0	1	1	1			
1	0	0	0			
1	0	0	1			
1	0	1	0			
1	0	1	1			
1	1	0	0			
1	1	0	1			
1	1	1	0			
1	1	1	1			



Enfin il existe deux jeux de caractères alphanumériques appelés G2Ø et G21. Ce sont deux jeux composés de deux parties :

	C5	0	0	1	1
	C5	0	1	0	1
B5					
0					
1					

Exemple

	7	6	5	4	3	2	1	0
Octet C	X	0	1	0	0	0	0	1
Octet B	0	1	0	0	X	X	X	X
Octet A	X	X	X	X	X	X	X	X

X = bits définis par l'utilisateur

C3	C2	C1	C0	C4 = 0	C4 = 1
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

— la première partie est formée par l'ensemble des minuscules du jeu G0 ainsi que quelques caractères spéciaux (32 caractères);
— la deuxième partie est composée d'un jeu d'accents (aigu, tréma, grave...) ainsi que du signe cédille (huit caractères).

La deuxième partie se superpose à la première, pour former un caractère complexe : on peut ainsi former e accent aigu (é) ou c cédille (ç), mais aussi bien x tréma (ẍ), ou n'importe quelle combinaison. Le code du caractère utilise les 5 bits de poids faible (bits 0 à 4). Ce code est complété par les deux bits de définition du complément, le bit de poids fort étant ici aussi indéfini.
Par exemple, pour obtenir ü (u tréma), on donnera le code

%0 10 10101 = $55, dans le jeu G21.

indéfini tréma u

Le jeu G20 correspond à B5=0, G21 à B5=1

Comment indiquer au 9345 quel jeu de caractères utiliser?

L'ensemble des caractères que nous venons de décrire est défini de manière permanente dans le 9345. Pour écrire un caractère sur l'écran, il faut fournir au 9345 le code du caractère ainsi que le jeu de caractères à utiliser.

Le jeu de caractères est codé sur 4 bits dans le registre R2. Ce sont les quatre bits de poids fort qui sont utilisés (B7, B6, B5 et B4). Voici les valeurs que doivent prendre ces bits :
— Le bit B7 est toujours à 0 pour les caractères standards du 9345. Il sera à 1 pour les caractères définis par l'utilisateur (nous verrons comment plus loin).

Bit 6	Bit 5	Bit 4	Jeu correspondant
0	0	1/0	Jeu G0 ; le bit B4 est à 1 pour souligner le caractère
0	1	0	Jeu G10
0	1	1	Jeu G11
1	0	1/0	Jeu G20 ; le bit B4 est à 1 pour souligner le caractère
1	1	1/0	Jeu G21 ; le bit B4 est à 1 pour souligner le caractère

Nota : L'ensemble du jeu de caractères du 9345 est compatible avec la norme Vidéotex, c'est-à-dire qu'il contient tous les caractères utilisés par le Minitel. C'est une caractéristique intéressante pour les applications télématiques.

Vous pouvez vérifier le tableau ci-dessus en modifiant le programme

d'exemple du début de ce chapitre. On peut afficher le jeu G1Ø au lieu du jeu GØ en modifiant la ligne 53 : il suffit de charger $21 au lieu de $Ø1 dans le registre R2. De même, la valeur $31 vous donnera le jeu G11, ou la valeur $11 vous soulignera tous les caractères.

Les attributs vidéo en mode 4Ø caractères

Le registre R2 est complété par les différents attributs vidéo disponibles. Ceux-ci sont codés sur les quatre bits de poids faible de R2, ainsi que sur deux bits de R3 (les bits 7 et 3). En voici la liste :

R2 = jeu de caractères L M H i
R3 = N F

Le bit i est le bit d'incrustation (état normal = 1, incrustation autorisée = Ø). Comme dans le mode 8Ø caractères, ce bit peut être utilisé pour générer des couleurs supplémentaires. Nous verrons cela dans le paragraphe qui suit, relatif aux couleurs des caractères.

Le bit H, lorsqu'il est à 1, provoque l'affichage du caractère en double hauteur. Le caractère est affiché sur deux rangées successives et sur la même colonne. Pour que l'affichage soit correct, le caractère doit être affiché deux fois en donnant des numéros de rangée successifs. Ce bit H doit être mis à 1 lors des deux affichages.

Le bit M est le bit de masquage. Lorsque le masquage est autorisé (pour cela le bit 3 du registre PAT doit être à 1), le fait de mettre le bit M à 1 provoque le «non-affichage» du caractère (un carré de la couleur du fond est affiché à la place du caractère). Cela peut être utilisé pour produire des effets : par exemple. alors que le masquage est demandé (bit 3 du registre PAT à 1), on envoie des caractères avec le bit M à 1, et rien ne s'affiche à l'écran. Il suffit de mettre le bit 3 du registre PAT à Ø pour que tous ces caractères apparaissent brusquement sur l'écran. On utilise ce système pour la protection de certaines zones lors de l'introduction d'un mot de passe ou d'un code confidentiel dans les logiciels professionnels.

Le bit L, lorsqu'il est à 1, provoque l'affichage en double largeur. Le caractère est affiché sur deux colonnes successives et sur la même rangée. Pour que l'affichage soit correct, le caractère doit être affiché deux fois en donnant des numéros de colonnes successifs. Le bit L doit être mis à 1 lors des deux affichages. Il est possible de combiner les bits H et L pour obtenir un caractère en double taille (double hauteur et double largeur). Dans ce cas, il faut écrire quatre fois le caractère sur deux rangées et deux colonnes successives, avec à chaque fois les bits L et H à 1.

Le bit F, lorsqu'il est à 1, provoque le clignotement du caractère. Dans ce cas, on affiche successivement le caractère sur le fond, puis le fond seul. La fréquence du clignotement est d'environ Ø.5 Hz (le caractère apparaît puis disparaît une fois toutes les deux secondes).

Attention : le clignotement n'est effectif que s'il est autorisé.
Pour cela, le bit 6 du registre PAT doit être à 1. S'il est à Ø, aucun caractère ne pourra clignoter.

Le bit N, lorsqu'il est à 1, provoque l'inversion vidéo du caractère. Il provoque l'inversion vidéo de la couleur du fond et de la couleur du caractère.

Tous ces attributs peuvent être utilisés simultanément. On peut demander l'inversion et le clignotement, par exemple. Il ne faut pas oublier non plus que le soulignement peut être utilisé pour les caractères alphanumériques.

Exemple de programme démontrant les possibilités du 9345

Ce programme met en évidence à l'écran les caractères compris entre A et Z. Pour modifier leurs caractéristiques, il faut utiliser les touches suivantes :

1 = soulignement
2 = double hauteur
3 = double largeur
4 = inversion vidéo
5 = clignotement

Ces attributs peuvent être cumulés. La touche Ø efface l'écran.

```
10 R0=11*16+3+15*16+2+2*16 : REM ADRESSE DES REGISTRES DU 9345
20 POKE 12314,1 : EXEC 54316
30 PRINT "   EXEMPLE D'UTILISATION DES ATTRIBUTS"
40 PRINT "         DU 9345 EN MODE 40 CARACTERES" :PRINT:PRINT
50 S=0 :L=0 :H=0 :N=0 :F=0
60 R2=1:R3=116
70 POKE R0+2,R2:POKE R0+3,R3
100 REM LECTURE DU CARACTERE SUR LE CLAVIER :
110 A$=INKEY$ : IF A$="" THEN 110
120 A=ASC(A$)
200 REM CAS DU CARACTERE NORMAL :
210 IF A < ASC("A") OR A > ASC("Z") THEN 300
220 R6 = PEEK (R0+6) :R7 = PEEK (R0+7)
230 IF H = 0 THEN 260
240 POKE R0+6,R6-1 :GOSUB 290 :IF L=1 THEN GOSUB 290
250 POKE R0+6,R6 : POKE R0+7,R7
260 IF L = 1 THEN GOSUB 290
270 GOSUB 290
280 GOTO 100
290 POKE R0+1+8,A :RETURN
300 REM CAS DES CARACTERES DE CHANSEMENT D'ATTRIBUTS
310 ON A-ASC("0")+1 GOSUB 10,400,500,600,700,800
320 GOTO 100
400 REM SOULIGNEMENT
410 S=1-S
440 R2=R2-(16)*((-1)+(S))
450 POKE R0+2,R2
460 RETURN
500 REM DOUBLE LARGEUR
510 L=1-L
```

```
540 R2=R2-(8)*((-1)*(L))
550 POKE R0+2,R2
560 RETURN
600 REM DOUBLE HAUTEUR
610 H=1-H
640 R2=R2-(2)*((-1)*(H))
650 POKE R0+2,R2
660 RETURN
700 REM INVERSION VIDEO
710 N=1-N
740 R3=R3-(128)*((-1)*(N))
750 POKE R0+3,R3
760 RETURN
800 REM CLIGNOTEMENT
810 F=1-F
840 R3=R3-(8)*((-1)*(F))
850 POKE R0+3,R3
860 RETURN
```

Les couleurs en mode 40 caractères

Jusqu'à présent, seules deux possibilités ont été exploitées dans le mode 40 caractères par rangée :
— le BASIC utilise des caractères noirs sur fond vert ;
— l'éditeur-assembleur utilise des caractères blancs sur fond bleu.
Mais il est possible d'utiliser n'importe laquelle des couleurs, aussi bien pour le fond que pour les caractères.

Pour cela, on utilise les bits qui nous restent dans le registre R3 :
— les bits de poids faible (bits 0, 1 et 2) sont utilisés pour coder la couleur de fond ;
— les bits 4, 5 et 6 sont utilisés pour coder la couleur du caractère.

Le code utilisé est le suivant : chacun des bits représente une couleur (Bleu, Vert, Rouge), et l'on peut mélanger ces trois couleurs pour obtenir les huit nuances classiques :

Bleu	Vert	Rouge	Couleur
0	0	0	Noir
0	0	1	Rouge
0	1	0	Vert
0	1	1	Jaune
1	0	0	Bleu
1	0	1	Magenta
1	1	0	Cyan
1	1	1	Blanc

Exemple de programme de modification des couleurs des caractères et du fond

Ce petit programme en BASIC vous permet de modifier la couleur du fond et la couleur du caractère : les touches A à Z sont visualisées sur l'écran. Pour changer de couleur de caractère, il faut appuyer sur

la touche < puis sur un chiffre entre 0 et 7 qui représente le code de la couleur. De même pour la couleur du fond, il faut appuyer sur > puis sur le chiffre qui code la couleur.

Nota : En plus des couleurs de fond et de caractère, il est aussi possible de modifier la couleur de la marge, qui correspond ici réellement au pourtour de l'écran. Pour cela, il faut écrire dans les bits 0, 1 et 2 du registre MAT, en suivant les mêmes règles que pour les couleurs de fond et de caractères (Noir = 0, Rouge = 1,...).

```
10 R0=11*16↑3+15*16↑2+2*16
20 POKE 12314,1:EXEC 54316
30 PRINT"MODIFICATION DE COULEURS"
40 CC=7:REM COULEUR CARACTERE
50 CF=0:REM COULEUR DE FOND
60 PRINT
70 GOSUB 700
80 PRINT
90 I=1:R2=0
100 REM LECTURE DU CLAVIER
110 A$=INKEY$:IF A$=""GOTO110
120 A=ASC(A$)
200 REM AFFICHAGE
205 IF A=ASC(" ") GOTO 220
210 IF A<ASC("A")OR A>ASC("Z")THEN 300
220 POKE R0+3,CF+16*CC
230 POKE R0+2,R2+I
240 POKE R0+1+8,A
250 GOTO 100
300 IF A=ASC("<") GOTO 400
310 IF A=ASC(">") GOTO 500
320 IF A=ASC(" ") GOTO 220
340 GOTO 100
400 REM MODIFICATION COULEUR CARACTERE
410 GOSUB 600
420 CC=A
430 GOTO 100
440 GOTO 100
500 REM MODIFICATION COULEUR DE FOND
510 GOSUB 600
520 CF=A
530 GOTO 100
600 REM SAISIE DE LA COULEUR
620 A$=INKEY$:IF A$="" GOTO 620
630 A=ASC(A$)-ASC("0")
640 IF A<0 OR A>7 GOTO 620
650 RETURN
700 REM MESSAGE DE COULEUR
710 PRINT"< COULEUR DE CARACTERE"
720 PRINT"> COULEUR DE FOND"
730 PRINT"0 NOIR"
740 PRINT"1 ROUGE"
750 PRINT"2 VERT"
760 PRINT"3 JAUNE"
770 PRINT"4 BLEU"
780 PRINT"5 VIOLET"
790 PRINT"6 CYAN"
800 PRINT"7 BLANC"
810 PRINT
820 RETURN
```

Comme dans le mode 80 caractères par rangée, il est possible d'utiliser le bit d'incrustation pour obtenir de nouvelles couleurs. 16 couleurs peuvent alors être affichées mais elles ne sont pas toujours discernables.

Nota : Suivant la programmation du registre PAT (voir le chapitre 10), on peut modifier soit la couleur du fond, soit la couleur du caractère, soit les deux. Dans le cas présent (celui utilisé par ALICE pour générer la couleur orange), ce sont les deux couleurs (fond et caractère) qui sont modifiées.
D'autre part, en utilisant le bit d'incrustation de la marge (bit 3 du registre MAT), il est possible de modifier sa couleur de la même façon que pour les couleurs de fond et de caractère.

Pour obtenir les seize couleurs dans l'exemple BASIC (couleurs de fond et de caractère), il suffit d'ajouter la ligne suivante :

```
330 IF A=ASC("O") THEN I=1-I
```

Chaque pression sur la touche Ø inversera la valeur du bit d'incrustation.

Programmation d'un caractère par l'utilisateur

Nous allons aborder pour la première fois la programmation d'un caractère par l'utilisateur. Voici le principe de l'opération :

- Il faut d'abord choisir le type de caractère qu'on veut employer. Il en existe trois :
  — les caractères alphanumériques
  — les caractères semi-graphiques bichromes
  — les caractères semi-graphiques quadrichromes.

- Ensuite, il faut allouer une partie de la mémoire privée du 9345 à ce type de caractères. Pour cela :
  — il faut lui donner un code (un numéro qui permette ensuite de le réutiliser) ;
  — il faut définir la forme du caractère en positionnant des bits à Ø ou à 1 dans la mémoire privée du 9345.

- Ensuite il sera possible d'utiliser le caractère aussi souvent que voulu : il suffit de donner le numéro et le type du caractère.

Cela permet de créer de nouveaux caractères qui n'appartiennent pas au jeu standard du 9345, par exemple les symboles utilisés pour les jeux de cartes (pique, cœur, carreau et trèfle) ou des caractères utilisés dans d'autres pays (alphabets grec ou cyrillique, tilde espagnol,...).

Les modes bichromes : réservation de la zone de définition d'un jeu de caractères

Il existe trois modes bichromes définis par l'utilisateur. On les appelle respectivement G'0 (c'est le mode alphanumérique) et G'10 et G'11 (ce sont les modes semi-graphiques). La définition de chacun de ces jeux occupe un bloc de 1 K-octet en mémoire privée du 9345.
— Pour le jeu G'0, l'adresse du bloc réservé est donnée sur 4 bits dans le registre DOR (ce sont les bits de poids faible B0, B1, B2 et B3).
Attention : Un code sur 4 bits permet de définir un numéro de bloc compris entre 0 et 15. Or il n'y a que 8 blocs disponibles dans ALICE. L'utilisation de G'0 n'est donc possible que si le bit B3 de DOR est mis à 0.
— Les jeux G'10 et G'11 sont définis de manière consécutive sur 2 K-octets de mémoire privée du 9345. On ne donne donc que l'adresse du premier bloc (celui qui définit G'10). Ce premier bloc doit obligatoirement être un bloc pair (blocs 0, 2, 4 ou 6 puisqu'ALICE n'a que 8 K-octets de mémoire privée pour le 9345). Il faut donc trois bits pour la définition de l'adresse de ce premier bloc. Ces trois bits se trouvent dans les bits B4, B5, et B6 du registre DOR.
Attention : De même que pour G'0, il faut mettre le bit B6 à zéro pour utiliser ce mode sur ALICE.

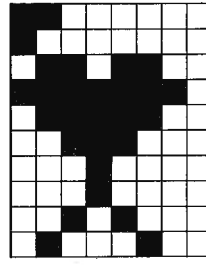
Par exemple, si vous voulez utiliser ces trois jeux de caractères simultanément sur ALICE, vous pouvez utiliser les adresses suivantes :
— La mémoire d'écran d'ALICE occupe 3 K-octets de mémoire (1 K-octet pour les 25*40 = 1000 caractères, et 2 K-octets pour les attributs et les couleurs de ces caractères).
— Donnons-lui les trois premiers K-octets de la mémoire privée du 9345; rappelez-vous que cette adresse peut être modifiée grâce au registre ROR. Elle occupe donc les blocs 0, 1 et 2.
— Nous pouvons utiliser le bloc 3 pour le jeu de caractères G'0.
— Les deux blocs 4 et 5 sont disponibles pour les jeux de caractères G'10 et G'11. C'est possible, puisque le premier bloc demandé est un bloc pair.
La valeur de DOR sera donc
DOR = %00010011=$23
Le bit B7 est mis à 0, nous verrons son utilisation plus loin. Au total nous occupons 6 K-octets de la mémoire privée du 9345, sur les 8 disponibles.

Les modes bichromes : définition d'un caractère

Un caractère bichrome est défini par un rectangle de 10 tranches de 8 points (comme on a 25 rangées de 40 caractères, on obtient la résolution maximale disponible sur ALICE : 320*250 points sur l'écran).

Il faut 1Ø octets pour définir un caractère bichrome. On peut donc définir 4 caractères dans un tampon de 4Ø octets. La définition de ces tampons se trouve dans le chapitre 1Ø. Comme il y a 25 tampons dans un bloc d'un K-octet, on peut définir 1ØØ caractères par jeu de caractères. ce qui fait un maximum de 3ØØ caractères semi-graphiques bichromes.

Voyons comment est codé un caractère, d'après l'exemple suivant :



sera codé sur les 8 octets suivants :

1re tranche : %ØØØØØØ11 = $Ø3
2e tranche : %ØØØØØØØ1 = $Ø1
3e tranche : %ØØ11Ø11Ø = $36
4e tranche : %Ø1111111 = $7F
5e tranche : %ØØ11111Ø = $3E
6e tranche : %ØØØ111ØØ = $1C
7e tranche : %ØØØØ1ØØØ = $Ø8
8e tranche : %ØØØØ1ØØØ = $Ø8
9e tranche : %ØØØ1Ø1ØØ = $14
10e tranche : %ØØ1ØØØ1Ø = $22

Comme vous pouvez le constater, les bits de la couleur de fond restent à Ø, tandis que les bits de la couleur du caractère sont à 1. Ce qui est un peu bizarre, c'est que le bit de gauche représente le pavé de droite, et inversement.

Ces dix tranches ne sont pas rangées successivement dans la mémoire du 9345. Il faut utiliser l'ordre suivant :
début du tampon :

1er octet = 1re tranche du 1er caractère
2e octet = 1re tranche du 2e caractère
3e octet = 1re tranche du 3e caractère
4e octet = 1re tranche du 4e caractère

5e octet = 2e tranche du 1er caractère
6e octet = 2e tranche du 2e caractère
...
37e octet = 1Øe tranche du 1er caractère
38e octet = 1Øe tranche du 2e caractère
39e octet = 1Øe tranche du 3e caractère
4Øe octet = 1Øe tranche du 4e caractère

Comme vous le voyez, les caractères sont définis par groupes de tranches et non caractère par caractère.

Pour un caractère donné, nous connaissons donc maintenant :
— le numéro du bloc où nous allons écrire tous les caractères qui composent notre jeu ;
— le numéro du tampon (attention, comme nous l'avons vu, les tampons sont numérotés Ø, 8, 9, ..., 31. Cf. chapitre 1Ø) ;
— les 1Ø octets qui composent la définition de notre caractère ;
— l'adresse de chacun de ces octets à l'intérieur du tampon.

Muni de toutes ces informations, nous allons utiliser l'instruction OCTet (commande OCT du 9345). Voici son format :

OCT =	Ø	Ø	1	1	R/$\overline{W}$	p	Ø	i

R/$\overline{W}$ Indique si l'on doit écrire dans la mémoire du 9345 (il faut alors mettre Ø), ou lire l'octet de la mémoire du 9345 (il faut alors mettre 1).

p Indique quels registres vont être utilisés comme pointeurs dans la mémoire du 9345 : si p=Ø, on utilise R6 et R7, si p=1, on utilise R4 et R5. En général, il vaut mieux utiliser R4 et R5 (p=1) en sauvegardant ainsi l'état de R6 et R7, utilisés comme pointeurs pour l'affichage des caractères.

i Provoque l'incrémentation du pointeur dans la mémoire du 9345 lorsqu'il est mis à 1 ; cela ne présente guère d'intérêt dans notre cas, puisqu'un même caractère n'est pas codé sur des octets successifs.

Comme d'habitude, la commande OCT doit être écrite dans RØ, et l'octet à écrire doit être mis dans R1.

Voyons maintenant un bref exemple d'application de cette commande. Si nous voulons écrire le caractère que nous avons défini tout à l'heure dans le 3e bloc (jeu G'Ø), dans le 3e tampon (tampon nº 9), comme 3e caractère du tampon, il faut donner les valeurs suivantes :
— 1re tranche
RØ = $34 (commande OCT, R/$\overline{W}$=Ø, p=1, i=Ø)
R1 = $Ø3 (valeur de l'octet à écrire)

R4 = $09 (voir le calcul de R4 dans le chap. 10)
R5 = $C2 (voir le calcul de R5 dans le chap. 10)

Il faut bien sûr demander ensuite l'exécution et vérifier son achèvement à l'aide de la routine BUSY.

Pour chaque tranche ensuite, il faut ajouter 4 à R5 et modifier R1 suivant le code de la tranche. R0 et R4 restent inchangés. La dernière tranche sera donc :

R0 = $34
R1 = $22
R4 = $09
R5 = $E6

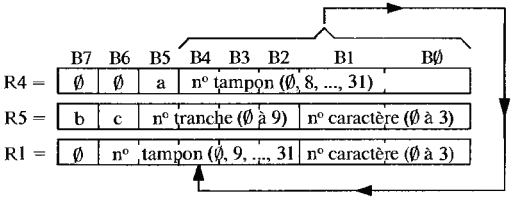
Les modes bichromes : utilisation d'un caractère

Une fois que le caractère est défini, il faut voir comment l'utiliser. Pour cela on utilise la même commande que pour un caractère alphanumérique ou semi-graphique standard, KRF. Seuls les paramètres changent. Il faut tout d'abord préciser le jeu dans lequel va être choisi notre caractère. Le numéro du jeu est codé dans R2 (bits B7, B6, B5 et B4). Le bit B7 doit être mis à 1, pour indiquer un caractère non standard. Le bit B6 doit être à 0 pour indiquer que l'on utilise le mode monochrome. Les deux bits restants sont codés de la manière suivante :

B5	B4	Jeu de caractères
0	1/0	Jeu alphanumérique G'0; B4 provoque le soulignement lorsqu'il est à 1
1	0	Jeu semi-graphique G'10
1	1	Jeu semi-graphique G'11

A l'intérieur du jeu, il faut ensuite donner le numéro du caractère. Ceux-ci sont numérotés de la façon suivante :
— les 4 caractères du tampon 0 ont respectivement comme numéros 0, 1, 2 et 3;
— les 4 caractères du tampon 8 (le deuxième tampon utilisable) sont numérotés 32, 33, 34 et 35;
— ensuite les numéros suivent, dans les tampons 9 à 31. Le dernier caractère est numéroté 127.

Le caractère que nous avons défini tout à l'heure (3e caractère du tampon 9) a donc le numéro 38 = $26.
Ce numéro doit être écrit dans le registre R1.
Voici un tableau qui donne la correspondance entre les registres R4 et R5 lors de la définition du caractère et la valeur de R1 lors de son utilisation :



a, c, b donnent les trois bits de poids faible du numéro de bloc (le bit de poids fort se trouve dans R6, mais vaut toujours Ø pour ALICE, puisqu'on ne dispose que de 8 Ko).
Nous pouvons ensuite définir les attributs de notre caractère (clignotement, insertion, double hauteur et/ou largeur, inversion vidéo, masquage, incrustation et couleur de fond et de caractère) en utilisant les mêmes paramètres que pour un caractère standard du 9345.

Programme de définition et d'utilisation d'un caractère bichrome

Nous donnons comme exemple un programme qui va écrire le caractère défini dans notre exemple, puis l'afficher en haut à gauche de l'écran.
Le programme commence par initialiser l'écran en mode 4Ø caractères par rangée (partie 2). Il n'initialise pas le registre DOR, car ALICE définit à l'initialisation le bloc 3 pour le jeu G'Ø.

Ensuite il écrit les dix tranches qui forment le caractère dans la mémoire du 9345. A chaque tranche R5 est incrémenté quatre fois (on aurait pu se contenter d'incrémenter R5 trois fois en mettant à 1 le bit i de la commande). C'est la partie 3 du programme.
Enfin le programme affiche le caractère en haut de l'écran. Il boucle ensuite pour que vous ayez le temps de voir l'affichage (partie 4).

```
 1 ;-----------------------------------------+
 2 ;      Programme BICHROME                 !
 3 ;                                         !
 4 ;Ce programme permet de creer un          !
 5 ; caractere bichrome et de le             !
 6 ; visualiser                              !
 7 ;-----------------------------------------+
 8 ;      Auteur :C MUSET                    !
 9 ;      Date : 9 Janvier 1985              !
10 ;-----------------------------------------+
11
12       ORG   $8000
13       EXC   BICHR
14
```

```
;1. CONSTANTES
;--
R0     =      $BF20
R1     =      $BF21
R2     =      $BF22
R3     =      $BF23
R4     =      $BF24
R5     =      $BF25
R6     =      $BF26
R7     =      $BF27
EXEC   =      $8
TECRA  =      $301A
INASS  =      $D42C

;2.    INITIALISATION
;--
BICHR
       LDAA   #$01
       STAA   TECRA
       JSR    INASS

;3. STOCKAGE DU CARACTERE
;--
;   +--+--+--+--+--+--+--+--+
;   !B0!B1!0 0 0 0 1 0! -> R5
;   +--+--+--+--+--+--+--+--+

       LDD    #$09C2
       STD    R4
       LDX    #TABIC
       LDAB   #$0A   ;10 TRANCHES
BI010
       LDAA   $0,X
       STAA   R1
       LDAA   #$34   ;ECRITURE OCTET
       STAA   R0+EXEC
       BSR    BUSY
       INC    R5
       INC    R5
       INC    R5
       INC    R5
       INX
       DECB
       BNE    BI010

;4. AFFICHAGE DU CARACTERE
;--
       LDD    #$0
       STD    R6
       LDD    #$2681 ;JEU G'0
       STD    R1
       LDAA   #$70
       STAA   R3
       LDAA   #$01
       STAA   R0+EXEC
       BSR    BUSY

FIN    BRA    FIN

;5. BUSY
;--
BUSY
       TST    R0
```

```
78          BMI    BUSY
79          RTS
80
81 ;6. TABLEAU BICHROME
82 ;--
83 TABIC
84          DFD    $03
85          DFD    $01
86          DFD    $36
87          DFD    $7F
88          DFD    $3E
89          DFD    $1C
90          DFD    $08
91          DFD    $08
92          DFD    $14
93          DFD    $22
94
```

Définition d'un caractère bichrome à partir du BASIC

Comme le BASIC d'ALICE utilise le jeu G'Ø pour ses caractères semi-graphiques, qui ne font pas partie des caractères standard du 9345, on peut utiliser une petite astuce : nous allons modifier un de ces caractères et l'utiliser en BASIC.

Pour cela, il suffit de modifier le programme en supprimant la partie 4 : un retour de sous-programme (RTS) fera l'affaire pour assurer la liaison avec le BASIC.

Il faut ensuite exécuter le programme. Si vous demandez l'exécution immédiate, vous n'obtiendrez pas votre nouveau caractère : à chaque fois que l'on fait RESET, ou que l'on sort de l'éditeur-assembleur, le jeu de caractères d'ALICE est recopié dans la mémoire du 9345. Il faut donc utiliser la commande EXEC à partir d'un programme BASIC :

EXEC 8*16↑3 (si vous avez demandé ORG $8ØØØ).

Ensuite, l'appui sur les touches SHIFT et F vous affichera le caractère nouvellement défini à la place du caractère semi-graphique usuel.

Les modes quadrichromes : réservation de la zone de définition des caractères

Il existe huit jeux de caractères quadrichromes définissables par l'utilisateur. On les appelle respectivement QØ, Q1, Q2,..., Q7.

Comme dans le mode bichrome, chaque jeu de caractères occupe un bloc d'un K-octet dans la mémoire du 9345. L'ensemble occupe donc 8 K-octets, qui doivent répondre à deux conditions :
— les huit blocs d'un Ko doivent être consécutifs;
— l'adresse du premier bloc ne peut être que le bloc Ø ou le bloc 8.

Pour indiquer quelle est l'adresse choisie pour le premier bloc, on utilise le bit B7 du registre DOR : il sera à Ø lorsqu'on utilise les huit premiers blocs, et à 1 si l'on utilise les huit derniers.
Rappel : ALICE n'alloue que huit blocs à la mémoire vidéo, donc il faudra mettre le bit B7 de DOR à Ø.

Si l'on fait par exemple la partition suivante de la mémoire du 9345, on obtiendra :
— 3 blocs pour la mémorisation de la page affichée (il faut en effet 3 octets pour chacun des 4Ø*25 = 1ØØØ caractères visualisés) ;
— 1 bloc pour les caractères alphanumériques bichromes définis par l'utilisateur (jeu G'Ø) ;
— 2 blocs pour les caractères semi-graphiques bichromes définis par l'utilisateur (jeux G'1Ø et G'11) ;
— 2 blocs pour définir les caractères quadrichromes (il n'en reste pas plus !).
La valeur du registre DOR sera donc la suivante :

DOR =	Ø	Ø	1	Ø	Ø	Ø	1	1	= $23

La configuration de la mémoire du 9345 sera par conséquent :

Bloc Ø	mémoire d'écran	Pointeur premier bloc quadrichrome
Bloc 1		
Bloc 2		
Bloc 3	G'Ø	
Bloc 4	G'1Ø	
Bloc 5	G'11	
Bloc 6	Q6	
Bloc 7	Q7	

Attention : comme vous le voyez, les jeux QØ à Q5 sont inutilisables, car ils détruiraient la mémoire d'écran et les jeux de caractères bichromes. Lorsqu'on n'utilise pas la totalité des jeux de caractères bichromes, il est possible de récupérer des blocs pour le mode quadrichrome. Le maximum est toutefois de 5 blocs (Q3 à Q7, en général).

Les modes quadrichromes : définition d'un caractère en haute résolution

Il existe deux types de modes quadrichromes : il y a des caractères quadrichromes «haute résolution» et «basse résolution».

Les caractères quadrichromes haute résolution sont divisés en 1Ø tranches de 4 points. (On atteint ici la résolution de 4Ø*4 = 16Ø points en largeur sur 25*1Ø = 25Ø points en hauteur).

De même qu'en mode bichrome, chaque tranche occupe 1 octet de la mémoire du 9345. Chacun des points est donc défini par deux bits :

ces deux bits, qui permettent d'attribuer une valeur entre Ø et 3, soit quatre valeurs possibles, indiqueront lors de l'affichage quelles couleurs devront être utilisées.

Supposons par exemple la tranche suivante :
couleur Ø couleur 2 couleur 1 couleur Ø
Il faudra alors définir l'octet suivant (la couleur de droite se retrouve dans les bits de gauche et inversement, comme dans les modes bichromes) :

% ØØ11ØØØ = $18

Les dix tranches d'un caractère devront être écrites octet par octet dans la mémoire du 9345 en utilisant la même règle qu'en mode bichrome : à l'intérieur d'un tampon, on inscrit tout d'abord la première tranche des quatre caractères, puis la seconde tranche des quatre caractères et ainsi de suite jusqu'à la dixième tranche.

Prenons un exemple, l'enregistrement du caractère suivant :

couleur Ø	couleur 2	couleur 1	couleur Ø	= %ØØ Ø1 1Ø ØØ	= $18
couleur 3	couleur 3	couleur 2	couleur 1	= %Ø1 1Ø 11 11	= $6F
couleur 1	couleur 3	couleur 2	couleur 1	= %Ø1 1Ø 11 Ø1	= $6D
couleur Ø	couleur 1	couleur 3	couleur 2	= %1Ø 11 Ø1 ØØ	= $B4
couleur 1	couleur Ø	couleur 1	couleur 3	= %11 Ø1 ØØ Ø1	= $D1
couleur 3	couleur 1	couleur Ø	couleur 1	= %Ø1 ØØ Ø1 11	= $47
couleur 2	couleur 3	couleur 1	couleur Ø	= %ØØ Ø1 11 1Ø	= $1E
couleur 1	couleur 2	couleur 3	couleur 1	= %Ø1 11 1Ø Ø1	= $79
couleur 1	couleur 2	couleur 3	couleur 3	= %11 11 1Ø Ø1	= $F9
couleur Ø	couleur 1	couleur 2	couleur Ø	= %ØØ 1Ø Ø1 ØØ	= $24

Une fois définis les 1Ø octets du caractère quadrichrome ci-dessus, il faut décider de l'endroit où nous allons l'enregistrer. Utilisons par exemple le bloc Q6, le 24e tampon, c'est-à-dire que le numéro du tampon sera 24 + 7 = 31 = $1F, et le 2e caractère de ce tampon (numéro 1).

On utilise la commande OCT. Pour la première tranche, les valeurs des registres seront les suivantes :

RØ = $34 (commande OCT, R/$\overline{W}$ = Ø, p = 1, i = Ø)
R1 = $18 (valeur de la 1re tranche)
R4 = $3F (= % ØØ1 11111 : R4 contient le bit 3 du numéro de bloc, –bloc 6 = 1– dans B5 ; les autres bits donnent le n° du tampon)
R5 = $41 (= % Ø1 ØØØØØ1 : R5 contient les bits Ø et 1 du numéro de bloc –bloc 6 : Ø et 1– dans B7 et B6 ; les autres bits contiennent le n° de l'octet)

Pour écrire les autres tranches, il faut seulement changer R1, qui contient la description de la tranche, et R5 qu'il faut augmenter de 4 à

chaque tranche, si le bit i d'OCT est à Ø. Pour la dernière tranche, on aura les valeurs suivantes :

RØ = $34
R1 = $24
R4 = $3F
R5 = $65

Le mode quadrichrome haute résolution : utilisation d'un caractère

On utilise la commande KRF pour afficher un caractère quadrichrome. Les paramètres sont assez différents de ceux des caractères en mode bichrome (qu'ils soient standard ou définis par l'utilisateur).

Il faut fournir les renseignements suivants au 9345 :
— le jeu de caractères choisi (QØ à Q7);
— le numéro du caractère à l'intérieur du jeu;
— les quatre couleurs à visualiser;
— les attributs, réduits ici à la seule incrustation.

Le numéro du caractère : il se trouve dans R1 et se calcule comme pour un caractère bichrome. Les caractères du tampon Ø ont les numéros Ø, 1, 2 et 3, puis à partir du tampon 8 (le deuxième disponible) les numéros vont de 32 à 127. Le caractère de l'exemple est le deuxième du bloc 31, il aura donc le numéro 125 = $7D.

Le numéro du jeu et le bit d'incrustation : le registre R2 contient le numéro du jeu quadrichrome et le bit d'incrustation suivant le format

R2 =	1	1	jeu	—	R = Ø	i

Les deux bits de poids fort indiquent que l'on se trouve en mode quadrichrome; ils sont tous les deux à 1.
Le numéro du jeu vient ensuite; dans l'exemple, c'est le jeu 6 = 11Ø.
Il y a une difficulté, car les bits ne doivent pas être dans l'ordre : les deux bits de poids faible doivent être inversés (ce qui donne dans l'exemple 1Ø1).
Le bit R à Ø indique que l'on est en haute résolution, alors que i joue le même rôle que pour les caractères bichromes. Dans notre exemple, nous aurons donc
R2 = % 111Ø1ØØ1 = $E9

Nota : l'inversion des deux bits de poids faible du jeu est identique au contenu du registre R5, où ils sont aussi inversés.

La définition des 4 couleurs du caractère : on utilise R3. Le codage des couleurs est assez compliqué. Chacun des bits de R3 représente la validation d'une couleur, dans l'ordre suivant :
blanc cyan magenta bleu jaune vert rouge noir

Il faut mettre 4 bits à 1 pour indiquer les quatre couleurs désirées. Par exemple, si l'on veut du rouge, du jaune, du bleu et du cyan, la valeur de R3 sera :
R3 = % 01011010 = $5A

Le bit mis à 1 le plus à droite définit la couleur 0 (ici le rouge), le deuxième bit à 1 à partir de la droite désigne la couleur 1 (le jaune), le troisième bit à 1 définit la couleur 2 (le bleu) et le quatrième bit à 1 définit la couleur 3 (le cyan).
S'il y a plus de quatre bits à 1, les derniers bits sont ignorés. S'il y a moins de quatre bits, les couleurs non définies sont attribuées à la couleur blanche.

Dans notre exemple, la première tranche du caractère aura les couleurs suivantes :

couleur 0	couleur 2	couleur 1	couleur 0 =
rouge	bleu	jaune	rouge

Le caractère entier sera donc :

R	B	J	R
C	C	B	J
J	C	B	J
R	J	C	B
J	R	J	C
C	J	R	J
B	C	J	R
J	B	C	J
J	B	C	C
R	J	B	R

En modifiant la valeur de R3, on pourrait obtenir le même caractère dans des couleurs différentes. Par exemple, avec R3 = %10011010 = $9A, le cyan sera remplacé par du blanc. En revanche il est impossible d'obtenir un caractère dont la première tranche serait magenta-bleu-jaune-magenta : la couleur magenta ne peut pas être la couleur n° 0 si le jaune est employé, puisque le bit "magenta" se trouve plus à gauche que le bit "jaune" dans R3. Dans ce cas, il faut définir un nouveau caractère dans le jeu Q6.

Programme de définition et d'affichage d'un caractère quadrichrome haute définition

Le programme définit le caractère que nous avons utilisé comme exemple, et l'affiche sur l'écran. Il faut de bons yeux, et un téléviseur bien réglé pour distinguer les couleurs des points qui composent le caractère.

```
;-----------------------------------------+
;      Programme QUADRICHROME             !
;                                         !
;Ce programme permet de creer un          !
; caractere quadrichome haute             !
; resolution et de le visualiser          !
;-----------------------------------------+
;      Auteur :C MUSET                    !
;      Date : 9 Janvier 1985              !
;-----------------------------------------+

        ORG    $8000
        EXC    QUADR

;1. CONSTANTES
;--
R0      =      $BF20
R1      =      $BF21
R2      =      $BF22
R3      =      $BF23
R4      =      $BF24
R5      =      $BF25
R6      =      $BF26
R7      =      $BF27
EXEC    =      $8
TECRA   =      $301A
INASS   =      $D42C

;2.    INITIALISATION
;--
QUADR
        LDAA   #$01
        STAA   TECRA
        JSR    INASS

;3. STOCKAGE DU CARACTERE
;--
;    +--+--+--+--+--+--+--+--+
;    !B0!B1!0 0 0 0 0 1!  -> R5 : 2EME CARACTERE
;    +--+--+--+--+--+--+--+--+
;
;    +-+-+-+-+-+-+-+-+
;    !0 0!B2!1 1 1 1 1!  -> R4 : BUFFER 31 = $1F
;    +-+-+-+-+-+-+-+-+
;
; BLOC 6 => B0=0 B1=1 B2=1

        LDD    #$3F41
        STD    R4
        LDX    #TAQUA
        LDAB   #$0A          ;10 TRANCHES
BI010
        LDAA   $0,X
        STAA   R1
        LDAA   #$34          ;ECRITURE OCTET
        STAA   R0+EXEC
        BSR    BUSY
        INC    R5
        INC    R5
        INC    R5
        INC    R5
        INX
```

```
63        DECB
64        BNE   B1010
65
66 ;4. AFFICHAGE DU CARACTERE
67 ;--
68        LDD   #$800
69        STD   R6
70        LDD   #$7DE9        ;JEU 06 (B0=0 B1=B2=1)
71        STD   R1
72        LDAA  #$5A          ;ROUGE JAUNE BLEU CYAN
73        STAA  R3
74        LDAA  #$01
75        STAA  R0+EXEC
76        BSR   BUSY
77
78 FIN    BRA   FIN
79
80 ;5. BUSY
81 ;--
82 BUSY
83        TST   R0
84        BMI   BUSY
85        RTS
86
87 ;6. TABLEAU QUADRICHROME
88 ;--
89 TABLA
90        DFO   $18
91        DFO   $6F
92        DFO   $6D
93        DFO   $B4
94        DFO   $D1
95        DFO   $47
96        DFO   $1E
97        DFO   $79
98        DFO   $F9
99        DFO   $24
100
101 ;7.   FIN
```

Définition d'un caractère quadrichrome basse résolution

On peut aussi définir des caractères quadrichromes en basse résolution. Les caractères en basse résolution sont divisés en 5 tranches de 4 points. La résolution est donc deux fois plus faible qu'en haute résolution. Un caractère n'occupe que 5 octets dans la mémoire du 9345, ce qui permet de définir 200 caractères par bloc d'un Ko, au lieu des 100 caractères en haute résolution.

L'écriture dans la mémoire du 9345 se fait de la même façon que pour la haute définition. La seule différence est que le 1er caractère est codé sur les 5 octets n° 0, 4, 8, 12 et 16 (1 octet par tranche). Le 2e caractère utilise les octets n° 1, 5, 9, 13 et 17, et ainsi de suite jusqu'au 8e caractère (octets 23, 27, 31. 35 et 39).

Nota : on peut mêler dans un même bloc des caractères haute et basse résolution, mais ce n'est pas conseillé, si vous voulez éviter des problèmes d'adressage lors de la définition et de l'affichage.

Affichage d'un caractère basse résolution

On utilise la même procédure d'affichage que pour la haute résolution, excepté ce qui concerne le registre R2. La valeur de R2 sera la suivante :

R2 =	1	1	jeu	K	R = 1	i

Le bit R à 1 indique la basse résolution.
Le bit K permet d'atteindre les caractères supplémentaires :
quand K = Ø, on atteint les 4 premiers caractères d'un tampon ;
quand K = 1, on atteint les 4 derniers caractères d'un tampon.

Programme de définition et d'affichage d'un caractère en basse résolution

Le programme définit deux caractères (qui représentent en fait chacun une moitié du caractère haute définition du programme précédent). Il utilise le jeu Q7 et inscrit ces caractères respectivement comme 3e et 7e caractères du 24e tampon (= tampon nº 31 = $1F).

Les deux caractères sont ensuite affichés sur l'écran. Ils ont tous les deux le même numéro de caractère, c'est le bit K dans R2 qui les distingue.

```
 1 ;-----------------------------------------+
 2 ;     Programme QUADRICHROME               !
 3 ;                                          !
 4 ;Ce programme permet de creer deux         !
 5 ; caracteres quadrichomes basse            !
 6 ; resolution et de les visualiser          !
 7 ;-----------------------------------------+
 8 ;     Auteur :C MUSET                      !
 9 ;     Date : 9 Janvier 1985                !
10 ;-----------------------------------------+
11
12       ORG   $8000
13       EXC   QUADR
14
15 ;1. CONSTANTES
16 ;--
17 R0    =     $BF20
18 R1    =     $BF21
19 R2    =     $BF22
20 R3    =     $BF23
21 R4    =     $BF24
22 R5    =     $BF25
23 R6    =     $BF26
24 R7    =     $BF27
25 EXEC  =     $8
26 TECRA =     $301A
```

```
27 INASS =      $D42C
28
29 ;2.   INITIALISATION
30 ;--
31 QUADR
32       LDAA  #$01
33       STAA  TECRA
34       JSR   INASS
35
36 ;3. STOCKAGE DES CARACTERES
37 ;--
38 ;   +--+--+--+--+--+--+--+--+
39 ;   !B0!B1!0 0 0 0 1 0!  -) R5 : 3EME CARACTERE
40 ;   +--+--+--+--+--+--+--+--+
41 ;
42 ;   +--+--+--+--+--+--+--+--+
43 ;   !0 0!B2!1 1 1 1 1!  -) R4 : BUFFER 31 = $1F
44 ;   +--+--+--+--+--+--+--+--+
45 ;
46 ;BLOC 7 =) B0=1 B1=1 B2=1
47
48       LDD   #$3FC2       ;1ER CARACTERE (NUMERO 2)
49       STD   R4
50       LDX   #TABU1
51       BSR   ECR5
52       LDD   #$3FD6       ;2EME CARACTERE  (NUMERO 6)
53       STD   R4
54       LDX   #TABU2
55       BSR   ECR5
56       BRA   AFFIC        ;AFFICHAGE
57
58 ;4. ECRITURE 5 TRANCHES
59 ;--
60 ECR5
61       LDAB  #$05         ;5 TRANCHES
62 ECR50
63       LDAA  $0,X
64       STAA  R1
65       LDAA  #$34         ;ECRITURE OCTET
66       STAA  R0+EXEC
67       BSR   BUSY
68       INC   R5
69       INC   R5
70       INC   R5
71       INC   R5
72       INX
73       DECB
74       BNE   ECR50
75       RTS
76
77 ;5. AFFICHAGE DES CARACTERES
78 ;--
79 AFFIC
80       LDD   #$800
81       STD   R6
82       LDD   #$7EFB       ;JEU 07 K=0 R=1
83       STD   R1
84       LDAA  #$5A         ;ROUGE JAUNE BLEU CYAN
85       STAA  R3
86       LDAA  #$01
87       STAA  R0+EXEC
88       BSR   BUSY
89       LDD   #$901
```

```
 90         STD    R6
 91         LDD    ##$7EFF        ;JEU 05 K=1 R=1
 92         STD    R1
 93         LDAA   ##$01
 94         STAA   R0+EXEC
 95         ESR    BUSY
 96
 97 FIN     BRA    FIN
 98
 99 ;6. BUSY
100 ;---
101 BUSY
102         TST    R0
103         BMI    BUSY
104         RTS
105
106 ;7. TABLEAUX QUADRICHROMES
107 ;---
108 TAQU1
109         DFD    $18
110         DFD    $6F
111         DFD    $6D
112         DFD    $E4
113         DFD    $D1
114 TAQU2
115         DFD    $47
116         DFD    $1E
117         DFD    $79
118         DFD    $F9
119         DFD    $24
120
121 ;8.     FIN
```

Exemple d'application : les lutins

Le 9345 n'est pas un processeur de visualisation adapté à la définition de lutins. On utilise d'ordinaire plutôt des processeurs spéciaux capables de gérer plusieurs niveaux (un fond et un avant plan au moins) pour les lutins.

Toutefois, comme il est possible de définir ses propres caractères, il est possible de définir un personnage par exemple vu dans différentes attitudes, de face, de dos, en profil gauche ou droit... En mode bichrome, on peut jouer sur l'attribut de double hauteur et celui de double largeur pour obtenir un personnage plus grand. On peut aussi combiner plusieurs caractères.

La technique peut être affinée : au lieu de déplacer le lutin caractère par caractère (ce qui donne un mouvement un peu saccadé), on peut définir un ensemble de caractères pour simuler un déplacement point par point. Cela exige donc un grand nombre de caractères (songez qu'on peut se déplacer dans les quatre directions sur l'écran, ajoutez

les différentes attitudes...), mais l'animation est bien plus souple et naturelle.

Enfin, il faut tenir compte du fond pour atteindre la perfection dans l'animation du lutin... Bon courage!

Nota : l'animation de lutins est plus facile à réaliser en mode bichrome, mais il n'y a que 3ØØ caractères disponibles.

Annexes

Annexe 1 : Bibliographie

Pour apprendre l'assembleur 68ØØ

Programmation du 68ØØ, Daniel DAVID et Rodney ZAKS, Sybex
68ØØ : Programmation en langage assembleur, L.-A. LEVENTHAL, Éditions Radio
Les Mystères d'ALICE, Alain BONNEAUD, SORACOM Informatique

Documentation technique

Motorola Semiconductors Advance Information, MC 68Ø1C MC 68Ø3C
Thomson EFCIS Integrated Circuits. Advance Information, EF 9345

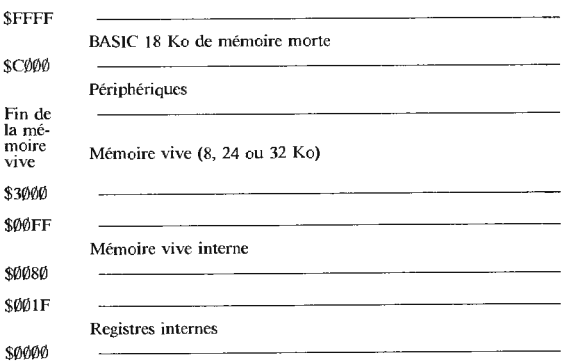
Ouvrages généraux

Pratique du micro-ordinateur ALICE, Henri LILEN, Éd. Radio
Dessiner, peindre et jouer avec ALICE, Louis GROS, Eyrolles
20 programmes astucieux pour ALICE, Ian CREASEY et Alain STEMMER, Hachette Informatique
Les écrans d'ALICE 90; Volume 1 : Premiers pas en programmation; Volume 2 : Pour en savoir plus en programmation, Denis PELLERIN, Hachette Informatique
ALICE et ALICE 90, Collection MICROMONDE, Cedic/Nathan
La découverte d'ALICE, Maurice CHARBIT, P. S. I.
102 programmes pour ALICE, Jacques DECONCHAT, P. S. I.
Exercices en BASIC pour ALICE et ALICE 90. P. S. I.
L'Assembleur d'ALICE et ALICE 90, P. S. I.
Jeux en BASIC pour ALICE, Pierre MONSAUT, Sybex
56 programmes sur ALICE et ALICE 90, Stanley R. TROST, Sybex
Premiers programmes sur ALICE et ALICE 90, Rodnay ZAKS, Sybex

Annexe 2

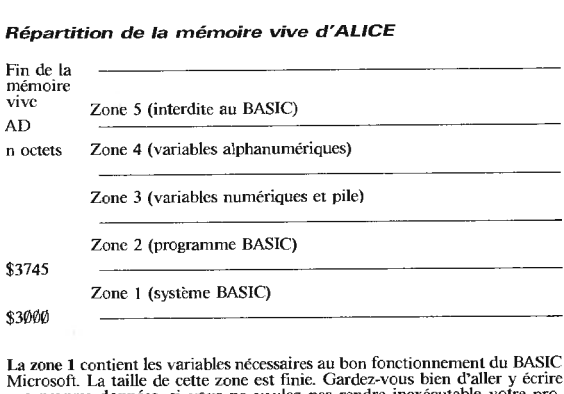
Carte mémoire d'ALICE 32 et d'ALICE 9Ø

Les cartes mémoire des versions 32 et 9Ø d'ALICE ne diffèrent que par l'adresse de la fin de la mémoire vive utilisable :
$4FFF pour ALICE 32;
$8FFF pour ALICE 32 avec l'extension mémoire;
$AFFF pour ALICE 9Ø.



(Voir aussi le manuel "Découvrez le BASIC" livré avec ALICE)

Répartition de la mémoire vive d'ALICE



La zone 1 contient les variables nécessaires au bon fonctionnement du BASIC Microsoft. La taille de cette zone est finie. Gardez-vous bien d'aller y écrire vos propres données, si vous ne voulez pas rendre inexécutable votre programme BASIC. Cette zone s'étend de $3000 à $3745.

La zone 2 contient le programme BASIC. Elle s'étend au fur et à mesure que vous ajoutez des instructions à votre programme.

La zone 3 contient les variables numériques, les tableaux numériques et la pile du système BASIC. Cette pile permet de gérer les boucles FOR... NEXT, et les GOSUB... RETURN.

La zone 4 contient les variables alphanumériques, c'est-à-dire les chaînes de caractères.

La zone 5 est une zone inaccessible au BASIC, sauf par les instructions PEEK, POKE. EXEC.

Lorsque vous mettez ALICE sous tension, la zone 5 n'existe pas, et la zone 4 fait 1Ø0 octets. L'instruction CLEAR permet de modifier cette partition de la mémoire :

— CLEAR N donne la nouvelle dimension de la zone 4. Elle ne modifie pas la zone 5.

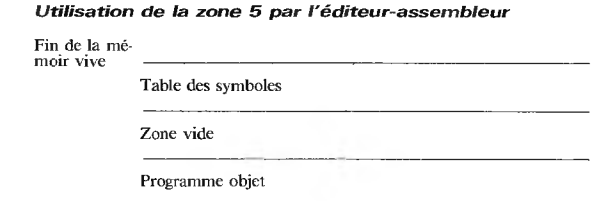
— CLEAR N, AD définit tout d'abord l'adresse de début de la zone 5, puis réserve N octets pour la zone 4.

Lorsque vous utilisez l'éditeur-assembleur inclus dans ALICE, il faut réserver de la place dans la zone 4 et dans la zone 5, pour les raisons suivantes :

— dans la zone 5 l'éditeur-assembleur doit loger le programme source (le texte que vous entrez), le programme objet (produit par l'opération d'assemblage) et une table contenant tous les symboles utilisés par votre programme, avec leur valeur ;

— la zone 4 est utilisée lors de l'impression du listing sur l'écran ou sur une imprimante, pour la table des symboles. C'est pourquoi il faut lui réserver au moins 6Ø octets. En revanche, l'éditeur-assembleur ne modifie pas la zone 2, il n'est donc pas nécessaire de faire NEW avant de l'utiliser.

Utilisation de la zone 5 par l'éditeur-assembleur



Une astuce : pour l'éditeur-assembleur, l'adresse de la fin de la mémoire vive est contenue dans les deux octets qui se trouvent en $328Ø-$3281. Il vous suffit donc de modifier cette adresse pour réserver de la place à un programme en assembleur, par exemple le programme MONITEUR.
Attention tout de même, cette zone pourra être détruite par les commandes CLOAD et MERGE qui utilisent la fin réelle de la mémoire vive.

Annexe 3

Programme MONITEUR

```
 1 ;-----------------------------------------+
 2 ;     Programme MONITEUR                  !
 3 ;     -----------------                   !
 4 ; Ce programme permet de visualiser le    !
 5 ;  contenu de la memoire d'ALICE , de     !
 6 ;  le modifier , de placer un point       !
 7 ;  d'arret,de visualiser les registres     !
 8 ;  et de les modifier .                    !
 9 ;-----------------------------------------+
10 ;     Auteur : J-F Gallet                 !
11 ;     Date : 23 Novembre 1984             !
12 ;-----------------------------------------+
13
14       ORG   $0     ;POUR ETRE RELOGEABLE
15       EXC   INIT
16
17 ;1.   Donnees externes :
18 ;---
19 DEVNU =$E8          ;SUPPORT D'IMPRESSION
20 SWINT =$FFFA        ;SWI VECTEUR D'INTERRUPTION
21 KDINP =$F868        ;ATTENTE D'UN CARACTERE
22 OUTCA =$F9C6        ;AFFICHAGE D'UN CARACTERE
23 OUTTX =$E7A8        ;AFFICHAGE D'UN TEXTE
24
25 ;2.   Programme d'adaptation du programme de translation :
26 ;---
27 INIT  BSR   IN010
28 IN010 TSX
29       LDD   $0,X
30       SUBD  #$2
31       PSHB
32       LDAB  #OFFST-$2
33       LDX   $0,X
34       ABX
35       PULB
36       STD   $0,X
37       PSHX
38       PULA
39       PULB
40       TSX
41       LDX   $0,X
42       PSHB
43       LDAB  #TLET2+$1-$2
44       ABX
45       PULB
46       STD   $0,X
47       PBHB
48       LDAB  #TLET3-TLET2
49       ABX
50       PULB
51       STD   $0,X
52       TSX
53       LDD   $0,X
54       ADDD  #TRTAB-$2
55       PSHB
56       LDX   $0,X
57       LDAB  #TLET1+$1-$2
```

```
 58          ABX
 59          PULB
 60          STD   $0,X
 61          PULX
 62
 63 ;3.      Programme de translation :
 64 ;--
 65 TLET1 LDX   #TRTAB
 66 TLDEB LDD   $0,X
 67          BEQ   TLFIN
 68          PSHX
 69 TLET2 ADDD  OFFST
 70          PSHB
 71          PSHA
 72          PULX
 73          LDD   $0,X
 74 TLET3 ADDD  OFFGT
 75          STD   $0,X
 76          PULX
 77          INX
 78          INX
 79          BRA   TLDEB
 80 TLFIN LDX   #INIT
 81          LDAA  #$7E
 82          STAA  $0,X
 83 TLFI1 LDD   #BEGIN
 84          STD   $1,X
 85          BRA   INIT
 86
 87 ;4.      Donnees du programme de translation :
 88 ;--
 89 OFFST DFD   $0
 90
 91 TRTAB DFD   TLFIN+$1     ;ADRESSE
 92          DFD   TLFI1+$1
 93          DFD   TL005+$1
 94          DFD   TL010+$1
 95          DFD   TL015+$1
 96          DFD   TL020+$1
 97          DFD   TL025+$1
 98          DFD   TL030+$1
 99          DFD   TL035+$1
100          DFD   TL040+$1
101          DFD   TL042+$1
102          DFD   TL043+$1
103          DFD   TL045+$1
104          DFD   TL050+$1
105          DFD   TL055+$1
106          DFD   TL060+$1
107          DFD   TL065+$1
108          DFD   TRABF+$1
109          DFD   TL070+$1
110          DFD   BEGIN+$1
111          DFD   TL072+$1
112          DFD   TL075+$1
113          DFD   TL085+$1
114          DFD   ATBRK+$1
115          DFD   DB010+$1
116          DFD   DB020+$1
117          DFD   TL090+$1
118          DFD   COMMH+$1
119          DFD   TL095+$1
120          DFD   TL100+$1
```

```
121         DFD   TL105+$1
122         DFD   REG10+$1
123         DFD   RGD10+$1
124         DFD   TL110+$1
125         DFD   TL120+$1
126         DFD   TL130+$1
127         DFD   COMCD+$1
128         DFD   TL135+$1
129         DFD   TL140+$1
130         DFD   TL145+$1
131         DFD   SPAD+$1
132         DFD   $0
133
134 ;5.    Routines d'ecriture sur l'ecran :
135 ;--
136 ; CONVO    CONVERSION OCTET -> ASCII
137 ;          ENTREE B 0,...,FF
138 ;          SORTIE A 1ER DIGIT
139 ;                B 2EME DIGIT
140 CONVO
141         PSHB
142         ANDB  #$F0
143         LSRB
144         LSRB
145         LSRB
146         LSRB
147         BSR   CONV1
148         TBA
149         PULB
150         ANDB  #$F
151
152 ;CONV1       CONVERSION DIGIT
153 ;       ENTREE B 0,...,F  SORTIE -> B
154 CONV1 CMPB  #$A
155         BLO   CONV2
156         ADDB  #$7
157 CONV2 ADDB  #$30
158         RTS
159
160 ;AFFA AFFICHAGE D'UN OCTET :
161 AFFA2 INX
162 AFFA3 LDAB  $0,X
163         BSR   CONVO
164         BSR   AFFA
165         TBA
166         DFO   $BC           ;CPX #
167 AFA60 LDAA  #$20
168 AFFA  JMP   OUTCA
169
170 ; AFFR     AFFICHAGE REGISTRES
171 AFFR
172         LDAA  #$D
173         BSR   AFFA      ;SAUT DE LIGNE
174 TL005 LDX   #ATXDB-$1
175         JSR   OUTTX
176 TL010 LDX   #SAUVP
177         LDAA  #$6
178         LDAB  $0,X
179         ROLB
180         RDLB
181 AFFR3 PSHA
182         CLRA
183         ROLB
```

```
184         ADCA  #$30
185         BSR   AFFA
186         PULA
187         DECA
188         BNE   AFFR3
189         LDAB  #$4
190 AFFR2   PSHB
191         BGR   AFA60
192         BSR   AFFA2
193         BGR   AFFA2
194         PULB
195         DECB
196         BNE   AFFR2
197         LDAA  #$D
198         BRA   AFFA
199
200 ;6.     DEBUG : Deviation du SWI
201 ;--
202 DEBUG
203         TSX
204         LDD   $5,X
205         SUBD  #$1
206         GTD   $5,X
207         TSX
208         DEX
209 TL015   STX   TRANS
210 TL020   LDX   #SAUVC
211 TL025   STX   TRAND
212         LDAB  #$8
213         BSR   TRASF
214         LDAB  #$6
215         TSX
216         ABX
217 TL030   STX   SAUVS
218 TL035   LDX   SAUPC
219 TL040   LDAA  SAUVC
220         STAA  $0,X
221 TL042   LDX   #SAUVA
222         LDD   $0,X
223         STAA  $1,X
224         STAB  $0,X
225         LDAA  (DEVNU
226         PSHA
227         LDAA  #$0
228         STAA  (DEVNU
229         BSR   ATBRK       ;ATTENTE
230 TL043   LDX   #SAUVA
231         LDD   $0,X
232         GTAA  $1,X
233         STAB  $0,X
234 TL045   LDX   SAUVS
235         LDAB  #$07
236 DEB05   DEX
237         DECB
238         BNE   DEB05
239 TL050   STX   TRAND
240 TL055   LDX   #SAUVC
241 TL060   STX   TRANS
242         LDAB  #$7
243         BSR   TRASF
244         PULA
245         STAA  (DEVNU
246 TL065   LDS   SAUVS
```

```
247         LDAB  #$7
248 DEB10   DEB
249         DECB
250         BNE   DEB10
251         RTI
252 ;
253 ;TRASF
254 ;
255 TRASF
256         LDX   TRANS
257         ABX
258         LDAA  $0,X
259 TL070   LDX   TRAND
260         ABX
261         BTAA  $0,X
262         DECB
263         BNE   TRASF
264         RTS
265
266 ;7.     BEGIN : Lancement du programme .
267 ;--      INSTALLE UN POINT D'ARRET
268 ;       AU DEBUT DU PROGRAMME UTILISATEUR
269 BEGIN
270         LDX   #TXDEB-$1
271         JSR   OUTTX
272 TL072   LDX   #SAUPC
273         CLR   $0,X
274         CLR   $1,X
275         LDX   SWINT
276         LDAA  #$7E      ;JMP DEBUG
277         BTAA  $0,X
278 TL075   LDD   #DEBUG
279         STD   $1,X
280         BSR   ATBRK
281 TL085   LDX   AD
282         JMP   $0,X      ;EXEC USER
283
284 ;8.     ATBRK : Traitement d'une commande :
285 ;--
286  ;8.1 Attente d'une commande :
287 ATBRK
288         JSR   AFFR
289 DBOUC
290         JSR   KDINP
291         JSR   OUTCA
292         CLRB
293 DB010   LDX   #TABCD  ;TABLEAU COMMANDE
294         ABX
295         CMPA  $0,X
296         BEQ   DB020
297         INCB
298         CMPB  #TABTR-TABCD+$1
299         BNE   DB010
300         BRA   DBOUC
301
302 DB020   LDX   #TABTR ;TABLEAU TRAITEMENT
303         ABX
304         ABX
305         LDD   $0,X
306 TL090   ADDD  OFFST
307         PSHB
308         PSHA
309         PULX
```

```
310         JMP   $0,X
311
312   ;8.2 Ajout d'un point d'arret :
313 COMMH JSR   AC4DG
314         BHS   ATBRK
315 TL095 STD   AD
316 TL100 LDX   AD
317         LDAA  $0,X
318 TL105 STAA  BAUVC
319         LDAA  #$3F        ;'RTI'
320         STAA  $0,X
321 EXIT  RTS
322
323   ;8.3 Registres A,B,F :
324 COMMA LDAB  #$1
325         DFD   $BC
326 COMMB LDAB  #$2
327         DFD   $21
328 COMMF CLRB
329 REG10 LDX   #SAUVP
330         ABX
331         PSHX
332         BSR   AC2DG
333         BHS   REG30
334         PULX
335         STAB  $0,X
336         BRA   ATBRK
337    ;ERREUR
338 REG30 PULX
339 REB40 BRA   ATBRK
340
341
342   ;8.4 Registres S,X,PC :
343 COMMS LDAB  #$04
344         DFD   $21
345 COMMX CLRB
346         DFD   $BC
347 COMMP LDAB  #$02
348 RGD10 LDX   #SAUVX
349         ABX
350         PSHX
351         BSR   AC4DG
352         BHS   REG30
353         PULX
354         STD   $0,X
355 RGD20 BRA   REG40
356
357   ;8.5 Modification en memoire :
358 COMMM BSR   AC4DG
359         BHS   RGD20
360 TL110 STD   ADCOU
361 COMC1 BSR   AFA6B
362         BSR   SPAD
363 TL120 JSR   AFFA3
364         BSR   AFA6B
365         BSR   AC2DG
366         BHB   COM20
367         BSR   SPAD
368         STAB  $0,X
369 COM10
370         JSR   KDINP
371         BSR   AFFAB
372         BRA   COM30
```

```
373 COM20 TBA
374 COM30 BSR   SPAD
375       CMPA  #$2B
376       BEQ   COMP
377       CMPA  #$2D
378       BEQ   COMM
379       BSR   AFFCR
380 TL130 JMP   DBOUC
381 COMP  INX
382       BRA   COMCO
383 COMM  DEX
384 COMCO STX   ADCOU
385       BSR   AFFCR
386 TL135 LDX   #ADCOU
387 TL140 JSR   AFFA3
388 TL145 JSR   AFFA2
389       BRA   COMC1
390
391  ;8.6 AFA6B : affichage d'un espace :
392 AFA6B LDAA  #$20
393       DFO   $8C
394 AFFCR LDAA  #$D
395 AFFAB JMP   OUTCA
396
397  ;8.7 SPAD : Chargement ADCOU
398 SPAD  LDX   ADCOU
399       RTS
400
401 ;9.   Acquisition d'une valeur HEXA :
402 ;--
403 AC4DG
404       BSR   AC2DG
405       BHS   AC4RT
406       TBA
407 AC2DG
408       PSHA
409       BSR   AC1DG
410       BHS   AC4ER
411       ASLA
412       ASLA
413       ASLA
414       ASLA
415       TAB
416       BSR   AC1DB
417       BHS   AC4ER
418       ABA
419       SEC
420 AC4ER
421       TAB
422       PULA
423 AC4RT
424       RTS
425
426 AC1DG
427       JSR   KDINP
428       BSR   AFFAB
429       PSHB
430       TAB
431       SUBA  #$30
432       BLO   AC1ER
433       CMPA  #$A
434       BLO   AC1OK
435       SUBA  #$7
```

```
436         CMPA  #$A
437         BLO   AC1ER
438         CMPA  #$F
439         BHI   AC1ER
440 AC1OK
441         PULB
442         SEC
443         RTS
444 AC1ER
445         PSHB
446         LDAA  #$2A
447         BSR   AFFAB
448         PULA
449         PULB
450         CLC
451         RTS
452
453 ;10.  Donnees du moniteur :
454
455  ;10.1 Sauvegarde des registres :
456 TABRG
457 AD      DFD   $0
458 SAUVC DFD   $0
459 SAUVP DFD   $0
460 SAUVA DFD   $0
461 SAUVB DFD   $0
462 SAUVX DFD   $0
463 SAUPC DFD   $0
464 SAUVS DFD   $0
465
466  ;10.2 Mots pour transfert :
467 TRANS DFD   $0
468 TRAND DFD   $0
469
470  ;10.3 Adresse courante :
471 ADCOU DFD   $0
472
473  ;10.4 Texte :
474 TXDEB 'MONITEUR ALICE . REV 1.00*
475         DFD   $0D00
476
477 ATXDB 'HINZVC A B  X    PC   SP
478         DFD   $0D00
479
480  ;10.5 Tableau des commandes :
481 TABCD DFD   $0D
482         'A
483         'B
484         'S
485         'X
486         'P
487         'F
488         'M
489         'H
490         'C
491
492 TABTR DFD   ATBRK
493         DFD   COMMA ;MODIF REGISTRE A
494         DFD   COMMB ;               B
495         DFD   COMMB ;               S
496         DFD   COMMX ;               X
497         DFD   COMMP ;      PC
498         DFD   COMMF ;      DRAPEAU
```

```
499         DFD   COMMM ;       MEMOIRE
500         DFD   COMMH ;POINT D'ARRET
501         DFD   EXIT  ;CONTINUE
502
503 ;11.    Fin
```

Annexe 4

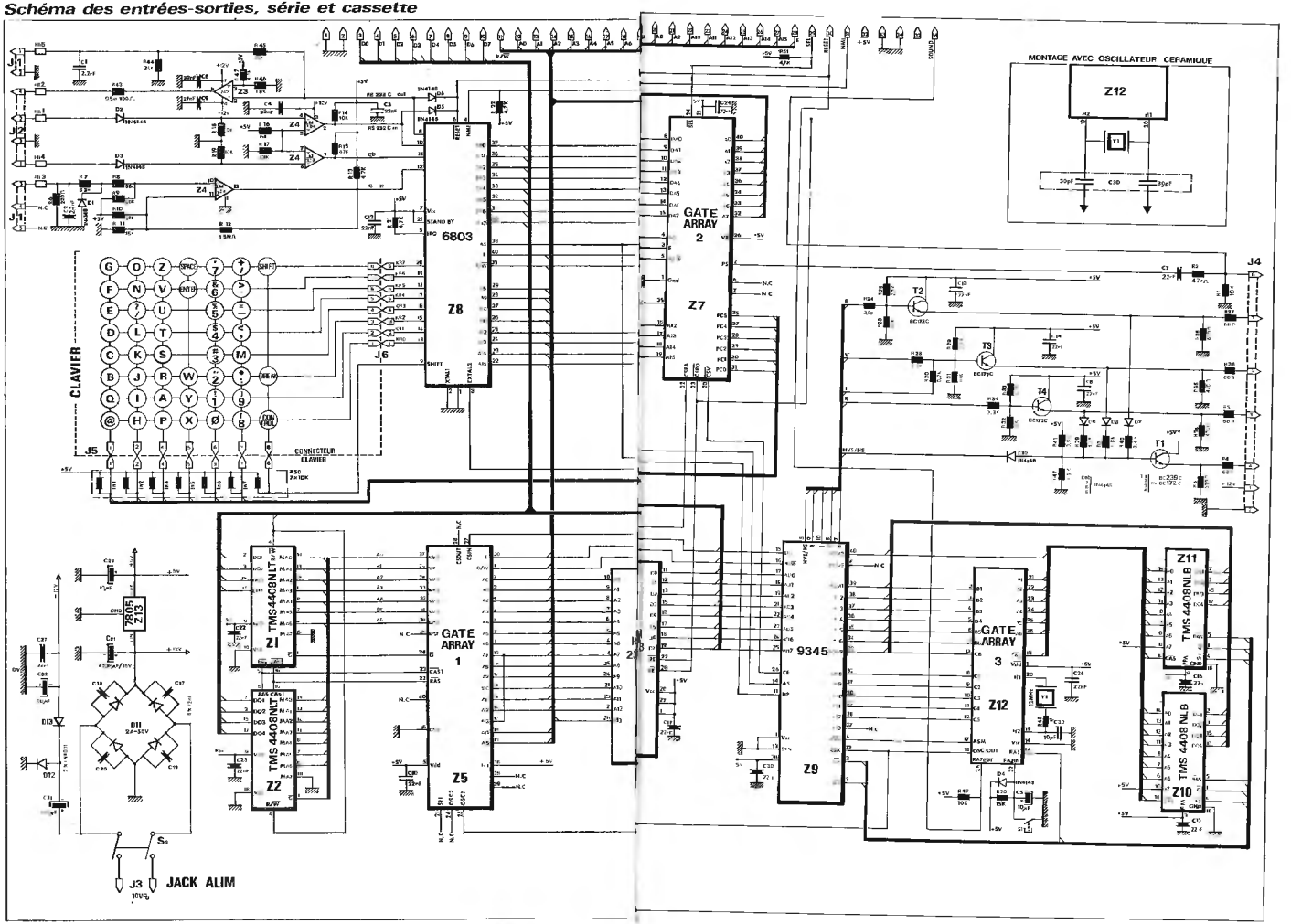
Schéma du bus (connecteurs d'entrées-sorties)

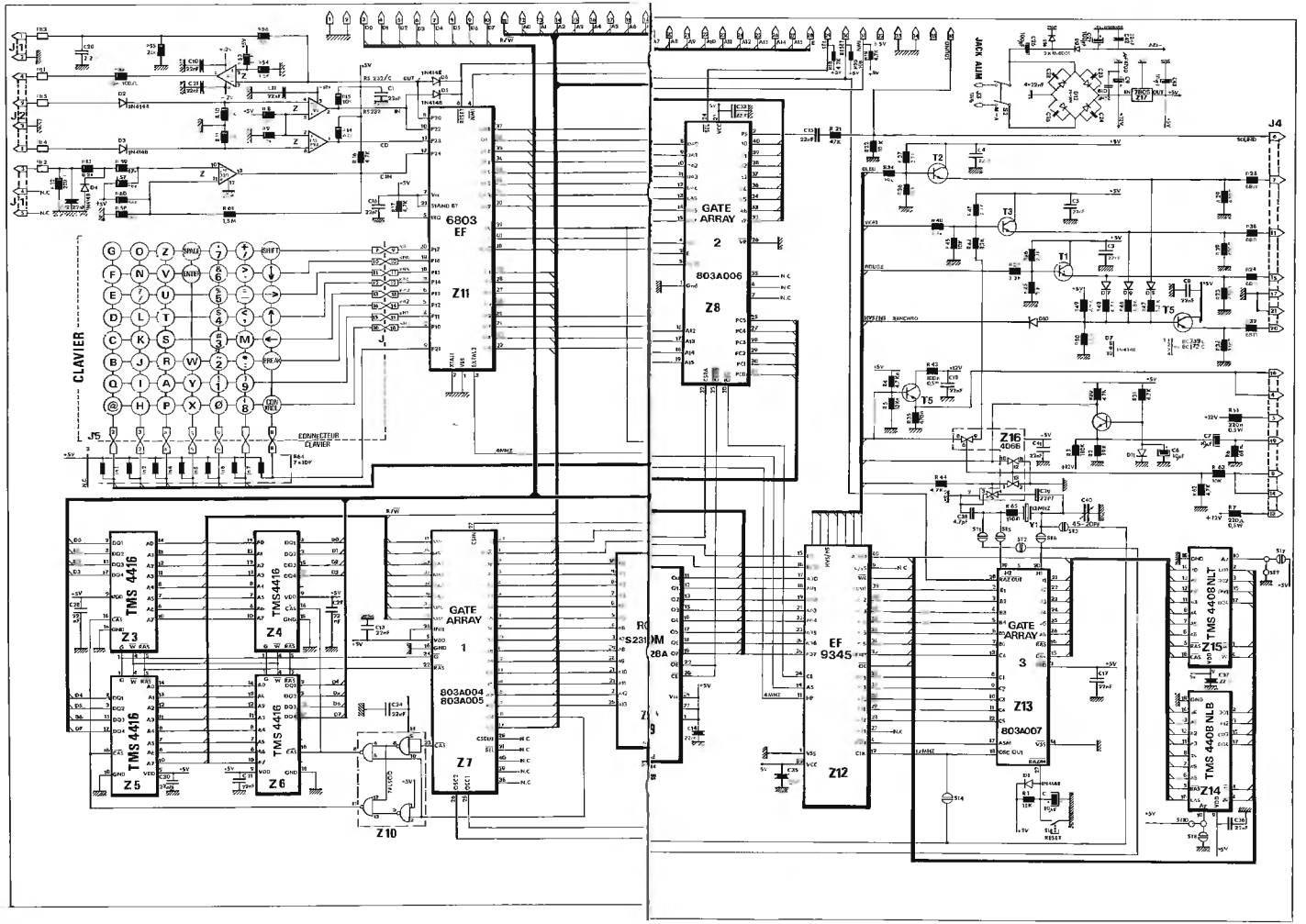
En plus des interfaces séries, ALICE possède une prise bus accessible par un connecteur de 34 points. Ce connecteur permet les extensions d'ALICE. Le tableau ci-dessous donne une brève description de la prise bus.

Broche	Signal	Description
1	GND	masse
2	GND	masse
3	DØ	bit Ø de donnée
4	D1	bit 1 de donnée
5	D2	bit 2 de donnée
6	D3	bit 3 de donnée
7	D4	bit 4 de donnée
8	D5	bit 5 de donnée
9	D6	bit 6 de donnée
1Ø	D7	bit 7 de donnée
11	R/W	signal de lecture écriture
12	AØ	bit d'adresse Ø
13	A1	bit d'adresse 1
14	A2	bit d'adresse 2
15	A3	bit d'adresse 3
16	A4	bit d'adresse 4
17	A5	bit d'adresse 5
18	A6	bit d'adresse 6
19	A7	bit d'adresse 7
2Ø	A8	bit d'adresse 8
21	A9	bit d'adresse 9
22	A1Ø	bit d'adresse 1Ø
23	A11	bit d'adresse 11
24	A12	bit d'adresse 12
25	A13	bit d'adresse 13
26	A14	bit d'adresse 14
27	A15	bit d'adresse 15
28	E	horloge du CPU (1 Mhz)
29	SEL	entrée d'invalidation des périphériques.
3Ø	Reset	reset principal et RAZ mise sous tension.
31	NMI	interruption non masquable du CPU.
32	+5v	alimentation 5 volts 3ØØ mA
33	GND	masse
34	GND	masse

Annexe 5

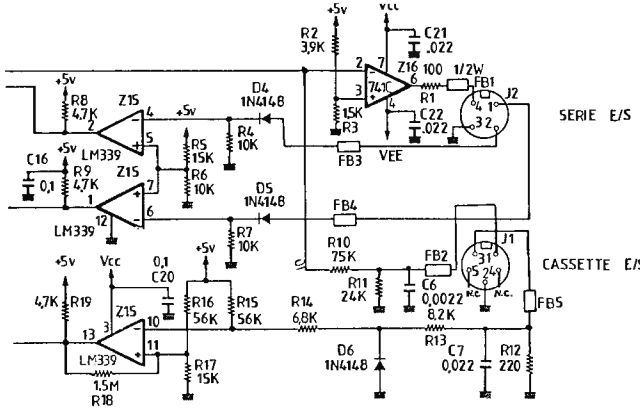
Schéma des entrées-sorties, série et cassette




CLAVIER
J5
CONNECTEUR CLAVIER
6803 EF
Z11
GATE ARRAY 2
803A006
Z8
GATE ARRAY 1
803A004
803A005
Z7
TMS 4416
Z3
TMS4416
Z4
TMS 4416
Z5
TMS 4416
Z6
Z10
EF 9345
Z12
GATE ARRAY 3
803A007
Z13
Z16 4066
TMS 4408 NLT
Z15
TMS 4408 NLB
Z14
JACK AUDIO
J4
T1
T2
T3
T5
RESET

Annexe 5

Schéma des entrées-sorties série et cassette



Annexe 6

Récapitulatif des adresses de la mémoire d'ALICE

Attention : les étiquettes données sont parfois sur six caractères, alors que l'assembleur intégré à ALICE n'accepte que cinq caractères.

Nom	Valeur	Taille	Fonction
	Ø2	1	port 1 utilisé par le clavier
	Ø3	1	port 2 utilisé par le clavier
	Ø8	1	registre de contrôle et d'état de l'horloge
	Ø9	2	compteur de l'horloge
	ØB	2	registre de comparaison en sortie (horloge)
	ØD	2	registre de capture en entrée (horloge)
	1Ø	2	registre de contrôle (interface série)
	12	1	registre de réception (interface série)
	13	1	registre d'émission (interface série)
TXTAB	93	2	pointeur sur le début du programme BASIC
VARTA	95	2	pointeur sur les variables BASIC
DEVNU	E8	1	support d'impression (Ø=écran, 1=imprimante)
	EA	1	indicateur de mise sous tension/RESET (RESET)
	3ØØ9	8	adresse de la table des tables de description
TECRAN	3Ø1A	1	type de l'écran (4Ø/8Ø colonnes)
NBNAM	3ØFØ	1	taille du nom du fichier à enregistrer (⩽8)
VARFI	3ØF1	8	nom du fichier à enregistrer
FILTY	3ØF9	1	type du fichier à enregistrer (BASIC, assembleur)
ASCFL	3ØFA	1	type du fichier (ASCII)
GAPFL	3ØFB	1	type du fichier (contigu/par blocs)
ADSTR	3ØFC	2	adresse d'exécution
ADLOA	3ØFE	2	adresse de chargement
ADDEB	31ØØ	2	adresse du premier octet à enregistrer
ADFIN	31Ø2	2	adresse du dernier octet à enregistrer
	3223	2	vitesse de l'imprimante
ROLTB	3231	8	mémorisation de la dernière touche entrée
ENDMM	325Ø	2	adresse de la fin de la mémoire vive accessible
FICNM	3257	8	nom du fichier
FICTY	3267	1	type du fichier
REALL	3272	2	indicateur de lecture effective
BLKTY	3275	1	type du bloc
K7ALC	3278	2	adresse des données
CURAD	328Ø	2	adresse du curseur (n° rangée, n° colonne)
	3746		première instruction BASIC
RØ à R7	BF2Ø	8	8 registres du processeur de visualisation EF 9345
	BFFF	1	bit B7 utilisé pour le son
TABAD	D421	8	tables de description du clavier
INASS	D42C		programme d'initialisation de l'écran
TABSHF	D8ØØ	56	tableau des touches + SHIFT
TABCON	D838	56	tableau des touches + CONTROL
K7WRT	DFAØ		programme d'enregistrement sur cassette
K7REA	DFBC		programme de lecture sur cassette
OUTTX	E7A8		programme d'impression d'une chaîne de caractères
LIPRT	F419		programme d'impression d'un nombre
	F7AD		programme de transfert de tableau
KSINP	F868		programme d'acquisition d'une touche au clavier (avec curseur)

POLCA	F883		programme de scrutation du clavier
TABNOR	F986	56	tableau des touches 'ordinaires'
OUTCA	F9C6		programme d'impression d'un caractère
GRNCH	FBD4		programme d'effacement de l'écran
CMPNA	FE37		programme de lecture de l'en-tête
K7LEC	FEB9		programme de lecture d'un bloc
	FFAB		émission d'un son
SCI	FFFØ	2	vecteur d'interruption SCI (32ØØ)
TOF	FFF2	2	vecteur d'interruption TOF (32Ø3)
OCF	FFF4	2	vecteur d'interruption OCF (32Ø6)
ICF	FFF6	2	vecteur d'interruption ICF (32Ø9)
IRQ1	FFF8	2	vecteur d'interruption IRQ (32ØC)
SWI	FFFA	2	vecteur d'interruption SWI (32ØF)
NMI	FFFC	2	vecteur d'interruption NMI (3212)
RESET	FFFE	2	vecteur d'interruption RESET (F72E)

Annexe 7

```
 1 ;-----------------------------------------+
 2 ;     Programme pour ALICE 90 :           !
 3 ;     Ce programme affiche successive-    !
 4 ;     ment une ligne normale et une ligne !
 5 ;     avec incrustation . N'oubliez pas!  !
 6 ;     de retourner le cable de liaison !  !
 7 ;     avec votre televiseur .             !
 8 ;-----------------------------------------;
 9
10       ORG   $9000
11       EXC   INCRU
12
13 ;1.   Definition des registres du 9345 :
14 ;---
15 R0    =     $BF20
16 R1    =     $BF21
17 R2    =     $BF22
18 R3    =     $BF23
19 R4    =     $BF24
20 R5    =     $BF25
21 R6    =     $BF26
22 R7    =     $BF27
23 EXEC  =     $8
24
25 ;2.   Modification des registres indirects :
26 ;---
27 INCRU
28       LDD   ##1881      ;REGISTRE TGS
29       BGR   INDIR
30       LDD   ##6882      ;REGISTRE MAT
31       BSR   INDIR
32       LDD   ##6783      ;REGISTRE PAT
33       BGR   INDIR
34       LDD   ##B584      ;REGISTRE DOR
35       BSR   INDIR
36       LDD   ##0887      ;REGISTRE ROR
37       BSR   INDIR
38
39 ;3.   Affichage des lignes sur l'ecran :
```

```
 40 ;--
 41        LDAA   #$0
 42        BSR    EFFL1
 43        LDAA   #$8
 44 BOUCL
 45        BSR    EFFLO
 46        INCA
 47        CMPA   #$20
 48        BEQ    FIN
 49        BSR    EFFL1
 50        INCA
 51        CMPA   #$20
 52        BNE    BOUCL
 53 FIN
 54        BRA    FIN
 55
 56
 57
 58 ;4.    Affichage d'une ligne sur l'ecran :
 59 ;--
 60   ;4.1 Affichage des rangees sans incrustation :
 61 EFFL1
 62        LDAB   #$1
 63        BRA    EFFLG
 64   ;4.2 Affichage des rangees avec incrustation :
 65 EFFLO
 66        CLRB
 67 EFFLG
 68        STAB   R2
 69        LDAB   #$41
 70        STAB   R1
 71        LDAB   #$20
 72        STAB   R3
 73        STAA   R6
 74        CLR    R7
 75        LDAB   #$28
 76 EFF1O
 77        PSHB
 78        LDAB   #$01
 79        STAB   RO+EXEC
 80        BSR    BUSY
 81        PULB
 82        DECB
 83        BNE    EFF1o
 84        RTS
 85
 86 ;5.    Programme de test du bit BUSY :
 87 ;--
 88 BUSY
 89        TST    RO
 90        BMI    BUSY
 91        RTS
 92
 93 ;6.    Ecriture dans les registres indirects :
 94 ;--
 95 INDIR
 96        STAA   R1
 97        STAB   RO+EXEC
 98        BGR    BUSY
 99        RTS
100
101 ;7.    Fin
```

Quelques rappels sur l'assembleur

Vous pouvez être tenté d'utiliser des programmes écrits en code machine pour plusieurs raisons. D'une part, l'exécution d'un programme en langage machine est beaucoup plus rapide que celle d'un programme écrit en BASIC. Cette rapidité est un atout important pour certains jeux, par exemple, où l'on souhaite une animation rapide et soignée, ainsi qu'une vive réaction aux commandes provenant du clavier.
Vous pouvez d'autre part utiliser certaines possibilités de la machine qui restent inexploitées par le BASIC, si vous avez recours au langage machine. Ce livre explore tout ce que peut faire ALICE, sans se limiter aux possibilités reconnues par le BASIC.
Enfin, vous pouvez souhaiter comprendre plus profondément comment fonctionne votre micro-ordinateur, et enrichir votre culture informatique.

Mais l'usage du code machine présente un certain nombre d'inconvénients et de difficultés qu'il va falloir surmonter.

Les difficultés du code machine

En code machine, vous perdez toute une série de facilités du BASIC. En BASIC, par exemple, vous aviez l'habitude de traiter des données éventuellement complexes (par exemple une chaîne de caractères) sans vous préoccuper de leur longueur (combien d'octets occupent-elles en mémoire ?) ni de leur emplacement en mémoire. Ces données étaient en effet désignées commodément grâce à des identificateurs de variables, et le BASIC se chargeait du reste.
En code machine les données sont traitées octet par octet (ou éventuellement par groupe de deux octets), et il n'y a pas d'identificateurs de variables ; le microprocesseur ne connaît que des adresses en mémoire ! A vous de savoir ce qu'elles recèlent...
De même, pour l'organisation du programme, plus de GOTO ou de GOSUB vers des numéros de lignes, le code machine demande là encore des adresses précises dans la mémoire.
Si vous deviez calculer précisément l'emplacement de toutes vos données, de tous vos branchements de programmes à la main, puis entrer votre programme sous forme de codes numériques, octet par octet, par exemple à l'aide du programme MONITEUR fourni dans ce livre, la tâche serait tout à fait déraisonnable. Heureusement, diverses aides sont à votre disposition : plus personne n'écrit directement en code machine, mais en vue d'obtenir un programme en code machine.
C'est ici qu'intervient la notion d'ASSEMBLEUR.

Qu'est-ce que l'assembleur ?

On confond souvent plusieurs choses sous le nom d'assembleur :
— le programme directement exécutable par la machine, qui est en fait écrit en code machine. C'est ce qu'on appelle le programme-objet.
Comme nous venons de le voir, ce programme serait très fastidieux à écrire, et à corriger : les instructions sont codées, il n'y a pas de variables, les paramètres sont désignés par leur adresse, de même que les branchements du programme...
— un texte, écrit sous EDITEUR, selon des normes de présentation définies, prêt à être traduit en code machine par un programme spécialisé ; c'est le programme source. C'est ce programme que vous allez écrire, en respectant quelques règles. Le programme source représente les instructions et les données sous forme conventionnelle, et non sous la forme des codes numériques compris par la machine. En particulier, les instructions sont représentées par des mots de trois ou quatre lettres, les mnémoniques, qui représentent clairement — pour qui sait l'anglais — ce que fera l'instruction. Le programme source est donc (relativement) facile à comprendre à la lecture.

— un programme spécialisé, l'Assembleur proprement dit, qui prend le texte du programme source, l'analyse et le vérifie, puis le traduit en programme objet exécutable. C'est l'opération d'assemblage, qui traduit les mnémoniques en code machine, calcule les adresses des données ou des branchements à partir des indications du programme source, et range le code machine résultant dans un fichier et/ou dans la mémoire de l'ordinateur.

Le rôle du programme assembleur

Le rôle du programme assembleur est donc de traduire un texte prêt à l'assemblage en code machine, c'est-à-dire en une suite de nombres, que le microprocesseur peut traiter.

Tous les éléments du programme source doivent être ainsi traduits, que ce soient des instructions, des données numériques ou des caractères. Chacun de ces éléments doit être codé de manière spécifique. Il existe donc plusieurs règles de codification que l'assembleur sélectionne à bon escient. Le point commun est que le résultat est toujours codé sous la forme de nombres binaires, seule forme que le microprocesseur reconnaît.

Avant de détailler les différentes formes de codification, il est bon de rappeler comment lire un nombre binaire.

Les notations

La notation binaire

La notation binaire n'utilise pour noter les nombres que deux symboles, 1 et Ø. On peut obtenir l'équivalent décimal d'un nombre binaire en commençant sa lecture par la droite et en additionnant les nombres que chaque 1 représente (chaque position marque une puissance de 2, suivant son rang : ainsi la quatrième position à partir de la droite est celle de 2^3=8).

Nombre binaire	1	Ø	1	Ø	Ø	Ø	Ø	1
Valeur de chaque position	128	64	32	16	8	4	2	1
Puissance de 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

1Ø1ØØØØ1 vaut donc 1 × 128 + 1 × 32 + 1 × 1 = 161.

Les règles de l'addition et de la soustraction en binaire sont très simples :
1 + 1 donne Ø et une retenue à reporter sur le chiffre binaire de gauche.
1 + Ø donne 1
Ø + 1 donne 1
Ø + Ø donne Ø
1 − 1 donne Ø
1 − Ø donne 1
Ø − 1 donne 1 et une retenue à soustraire du chiffre binaire de gauche
Ø − Ø donne Ø

Suivant le type de données que l'on veut manipuler, on considérera chaque nombre binaire stocké par la machine différemment : on ne fait pas les mêmes opérations sur un caractère, un nombre entier strictement positif, un nombre entier signé (c'est-à-dire positif ou négatif), ou un nombre décimal.

La notation hexadécimale

Tant que nous en sommes aux notations des nombres, examinons la notation en base 16, ou notation hexadécimale. Cette notation est particulièrement utile en informatique, car elle coïncide facilement avec les unités manipulées par les microprocesseurs actuels : la plupart des processeurs travaillent avec des données sur 8 bits, qui sont représentées en hexadécimal par deux symboles hexadécimaux. Toutes les adresses accessibles dans la mémoire d'un micro-ordinateur de 64 Ko sont représentables par quatre symboles hexadécimaux. Il est donc indispensable de connaître les règles de cette notation.

L'assembleur d'ALICE ne sait reconnaître que les nombres donnés en notation hexadécimale, c'est-à-dire en base 16 (les unités vont de Ø à 9, puis A à F pour les nombres de 1Ø à 16 en décimal) ; de plus, il faut toujours faire précéder les nombres fournis à l'assembleur du symbole $.

Il est facile de traduire un nombre binaire en hexadécimal ou vice-versa si l'on se souvient que chaque symbole du nombre en notation hexadécimale correspond à un ensemble de quatre bits (un quartet) du nombre noté en binaire :
$13BF se décompose ainsi en $1 = ØØØ1
$3 = ØØ11
$B = 1Ø11
$F = 1111
ce qui donne $13BF = ØØØ1ØØ11 1Ø111111, sur deux octets.
Pour traduire un nombre hexadécimal en décimal, dans un programme BASIC, procédez ainsi ;

```
$13BF → $1 =  1*16↑3
          =  3*16↑2
          =  11*16↑1
      $F = 15
$13BF = 1*16↑3+3*16↑2+11*16+15
```

Poids fort, poids faible

On utilise très souvent les termes de poids fort et poids faible. La partie 'gauche' d'un nombre est appelée sa partie de poids fort (ou partie la plus significative), celle de droite (vers les unités) partie de poids faible (partie la moins significative).
Ainsi, le bit de poids fort d'un octet sera le huitième à partir de la droite. L'octet de poids fort d'une adresse notée sur deux octets est l'octet de gauche : dans $13BF, $13 est la partie de poids fort, $BF celle de poids faible.

Dans un octet, on numérote les bits du poids faible vers le poids fort. Voici la numérotation ordinaire des bits d'un octet :

B7 B6 B5 B4 B3 B3 B2 B1 BØ

La représentation des données

Les données d'un octet peuvent être considérées de trois manières différentes :

1. La représentation d'nn nombre en notation non signée.

Cette notation permet de coder sur un octet un nombre sans considération de signe, entre Ø et 255.
Dans cette notation, 1Ø1ØØØØ1 représente le nombre 161. Cette représentation sert pour les entiers positifs, et pour le codage des instructions ou des caractères (code ASCII).

2. La représentation d'nn nombre en complément à deux.

Cette notation permet de représenter un nombre compris entre −128 et +127, inclus. Le signe est indiqué par le bit le plus "à gauche", qu'on appelle le bit de poids fort (1 pour le signe négatif, Ø pour le signe positif). Pour écrire un nombre négatif, en complément à deux, on peut partir du nombre positif correspondant. La première opération à effectuer est la complémentation, qui consiste à inverser tous les bits du nombre positif : les bits à 1 sont mis à Ø et inversement. La deuxième opération consiste à ajouter 1 au résultat. On obtient ainsi un nombre négatif.

Si l'on part de +5 pour trouver la représentation en complément à 2 de −5, voici le détail des opérations :

+5 noté sur un octet	: ØØØØØ1Ø1
complémentation	: 11111Ø1Ø
ajout de 1	: 11111Ø11 = −5 en notation décimale.

A l'inverse, si vous voulez trouver la valeur d'un nombre binaire en complément à 2, vous devez lui soustraire 1, puis le complémenter pour trouver sa valeur absolue.
Prenons pour exemple 1Ø1ØØØØ1, comme plus haut. Si nous considérons que cet octet note un nombre en complément à deux, quelle est sa valeur en notation décimale ?

Soustrayons 1	: 1Ø1ØØØØØ
Prenons le complément	: ØlØ11111 = 95

Comme le bit le plus "à gauche", le bit de poids lourd était à 1, nous pouvons dire que 1Ø1ØØØØ1 représente −95. Cette représentation sert pour les nombres entiers positifs et négatifs. C'est aussi la base des systèmes des nombres décimaux.

3. La représentation de Décimal Codé Binaire (DCB).

Dans cette notation, on choisit de représenter sur un octet un nombre compris entre ØØ et 99, en notant chaque chiffre décimal sur un groupe de quatre bits, un quartet. C'est une procédure utile pour des calculs que l'on veut faire très précisément. Voici un tableau de positions significatives pour un quartet noté en DCB :

ØØØØ = Ø	Ø1Ø1 = 5
ØØØ1 = 1	Ø11Ø = 6
ØØ1Ø = 2	Ø111 = 7
ØØ11 = 3	1ØØØ = 8
Ø1ØØ = 4	1ØØ1 = 9

Toutes les autres positions sont sans signification en notation DCB. Si l'on veut faire une addition 9+1 (décimal) en DCB, il faut passer
de ØØØ1ØØ1 (9)
à ØØ01ØØØØ (10).

Les règles de l'addition sont donc différentes sur un octet en DCB de celles sur un octet en complément à 2, ou en notation non signée. En particulier, on utilise une retenue entre les deux quartets qui constituent l'octet. Cette retenue entre les deux quartets est marquée par un bit du registre d'état du processeur, le bit H. pour half-carry, c'est-à-dire demi-retenue, retenue entre deux quartets.
On se sert de la notation DCB surtout dans des calculs comptables où l'on veut éviter des erreurs d'arrondi.

Comment travaille le microprocesseur d'ALICE ?

Le microprocesseur d'ALICE est un MC68Ø3, qui comporte les registres suivants (ce sont des mémoires propres aux microprocesseurs, sur lesquelles il peut effectuer les opérations qu'on lui demande à l'aide des mnémoniques) :

Les registres

Deux registres ou accumulateurs de travail à 8 bits
A
B
(Ces deux registres peuvent être considérés comme un unique registre double, pour faire des opérations sur 16 bits d'un coup ; dans ce cas, on parle du registre D.)
Un registre d'index à 16 bits
IX
Un registre pointeur de pile à 16 bits
SP
Un registre d'état à 8 bits
P
Un compteur programme à 16 bits.

A quoi servent ces registres ?

Le **compteur programme,** sur lequel on ne peut intervenir, garde constamment en mémoire l'adresse de la prochaine instruction à exécuter. C'est lui qui permet au programme de se dérouler.

Le **registre SP,** pointeur de pile, garde en mémoire l'adresse du sommet de la pile, où l'on enregistre des données pendant le traitement. C'est sur la pile qu'on enregistre l'adresse à laquelle il faudra revenir après l'exécution d'un sous-programme, ou les données qui précédaient une interruption (cf. le chapitre sur les interruptions).

Le **registre d'index** sert principalement pour calculer l'adresse de données dans le cas de l'adressage indexé (voir plus loin les modes d'adressage).

Les **registres A et B** servent au transfert des données, et à la plupart des opérations qu'on doit leur appliquer. Une opération consiste donc en général à aller chercher une donnée en mémoire, en prendre une copie dans un des registres, à lui appliquer le traitement voulu, et à remettre le résultat dans un emplacement approprié de la mémoire.

Le registre d'état, P, contient un certain nombre d'indicateurs qui résultent des opérations qui viennent d'être effectuées sur les registres, et permettent de décider de la suite du programme.
Le registre d'état du 68Ø3 comporte les indicateurs suivants :

ØØHINZVC

Les deux bits de poids le plus fort sont sans signification.

L'indicateur C marque qu'une opération a donné lieu à un report, une retenue (*en anglais :* carry). L'indicateur est mis à 1 s'il y a eu retenue (reportez-vous au tableau du manuel éditeur-assembleur pour le détail des opérations qui peuvent affecter C).

L'indicateur V marque qu'une opération a occasionné un dépassement (*en anglais :* overflow). L'indicateur est mis à 1 si une opération exige une taille de mot supérieure à celle considérée par l'opération ; par exemple V sera mis à 1 si une addition affecte le bit de poids le plus lourd du mot, par le jeu des retenues.

L'indicateur Z est mis à 1 chaque fois qu'une opération a pour résultat de mettre l'accumulateur concerné à Ø.

L'indicateur N est mis à 1 lorsque le résultat d'une opération fait passer le bit le plus lourd d'un mot à 1, dans un contexte où l'on considère les valeurs comme signées.

L'indicateur I est mis à 1 lorsqu'on veut interdire les interruptions masquables (voir le chapitre sur les interruptions).

L'indicateur H est mis à 1 dans les opérations d'addition quand il y a une retenue entre les deux quartets d'un octet. Cet indicateur est utile pour les calculs en DCB.

Comme le code machine ne connaît l'emplacement des données que sous forme d'adresses, il est important de disposer de procédés souples pour indiquer ces adresses dans le programme source, en laissant à l'assembleur et au microprocesseur la charge de traduire et calculer les adresses efficacement.

Les types d'adressage

Quand une instruction du microprocesseur doit trouver dans la mémoire la ou les données sur lesquelles elle va travailler (certaines instructions n'ont pas à chercher dans la mémoire, on parle dans ce cas d'adressage implicite ou inhérent : par exemple, INX qui augmente de 1 le contenu du registre IX n'a pas besoin de paramètre extérieur) plusieurs cas se présentent.

1) La donnée à traiter se trouve immédiatement à côté de l'instruction. C'est l'adressage immédiat.
On le note dans le programme source en faisant précéder la donnée par le symbole #. Ainsi : LDAA #$Ø8 donne l'instruction au processeur de charger dans l'accumulateur A (LDAA load accumulator A) la donnée qui se trouve dans l'octet adjacent. L'assembleur placera la valeur $Ø8 à côté du code de l'opération LDAA. Si l'opération joue sur deux octets (par exemple LDX, charger le registre double IX), deux octets seront réservés à la donnée. C'est utile pour des constantes dans un programme.

2) La donnée est désignée par l'adresse où elle se trouve.
C'est l'adressage direct.
Le microprocesseur MC68Ø3 permet deux variantes d'adressage direct :
— l'adressage direct dans les 256 premiers octets de la mémoire, qu'on appelle la page Ø parce que les adresses, écrites en hexadécimal vont de $ØØØØ

à $ØØFF. C'est l'adressage direct en page Ø. On l'indique dans le programme source en faisant précéder l'adresse définie en deux chiffres hexadécimaux par le symbole <. Ainsi : LDAA <$1F donne l'instruction de charger l'accumulateur A avec la donnée qui se trouve à l'adresse $ØØ1F. L'assembleur traduira cette instruction par un code spécial pour LDAA, et le fera suivre d'un seul octet, qui indique l'adresse en page Ø;

— l'adressage direct dans le reste de la mémoire, pour lequel il est nécessaire de définir l'adresse sur deux octets. C'est l'adressage direct étendu. Il permet d'accéder à des données rangées n'importe où dans la mémoire. Il est indispensable pour les adresses qui vont de $Ø1ØØ à $FFFF.
Ce mode d'adressage n'est pas marqué par un symbole spécial dans le programme source. Ainsi : LDAA $Ø1ØØ donne l'instruction de charger l'accumulateur A avec la donnée qui se trouve à l'adresse $Ø1ØØ. L'assembleur traduira cette instruction avec le code convenable et stockera l'adresse de la donnée sur deux octets après le code de LDAA.
L'adressage direct est l'analogue le plus proche de l'usage des variables en BASIC : le processeur travaillera sur une adresse précise, où l'on stockera une valeur éventuellement modifiable.

3) On peut vouloir que le processeur calcule l'adresse d'une donnée au moment de l'exécution à partir de plusieurs paramètres. On peut dans ce cas avoir recours à l'adressage indexé. Le processeur additionnera le registre double IX qui contient à ce moment une adresse sur 16 bits et la donnée qui se trouve sur un octet, juste après le code de l'instruction. L'opération portera sur la donnée dont l'adresse est le résultat de l'addition.
Ainsi si IX contient $Ø1ØØ, LDAA $ØB,X additionnera $Ø1ØØ+$Ø8, soit $Ø1Ø8 et chargera dans A la donnée d'adresse $Ø1Ø8.
On a recours à l'adressage indexé pour traiter des séries de données, par exemple dans des tableaux.

4) Pour les branchements de programme, on peut définir le point de branchement par rapport à la position du compteur programme (et non absolument par rapport à la mémoire du microprocesseur). C'est l'adressage relatif. On peut l'utiliser quand le point de branchement est situé à proximité de l'instruction de branchement.
Qu'entend-on par « à proximité » ?
L'aboutissement du branchement est assez près pour qu'on puisse utiliser l'adressage relatif quand le point de branchement se trouve 128 octets de code avant, ou 127 octets après la position du compteur programme. En effet, le branchement relatif se fait en ajoutant au compteur programme la valeur d'un octet de déplacement, considéré en complément à deux. Rassurez-vous, c'est l'assembleur qui se charge de calculer pour vous le déplacement, grâce aux étiquettes que vous lui fournissez. Il vous signalera une erreur si vous tentez d'utiliser un branchement relatif hors des limites; à vous, dans ce cas, de remplacer l'adressage relatif par un adressage direct, avec une adresse absolue.

```
Ainsi :          BSR SSPRG
        SSPRG    LDAA ...
```

sera traduit par l'assembleur avec un déplacement de $Ø7 si SSPRG est implanté neuf octets après l'instruction BSR (branchement à la sous-routine). En effet, le compteur programme pointera après le dernier octet de l'instruction en cours d'exécution; comme BSR et son paramètre de déplacement occupent deux octets, le compteur programme marque l'adresse de BSR+2. 2+7 font 9, adresse où sera transférée l'exécution. Si la routine avait été placée avant l'instruction BSR, l'adresse de départ du calcul aurait été équivalente. Mieux vaut laisser l'assembleur se charger des calculs!

Les opérations effectuées par le 68Ø3

Les opérations que peut effectuer le microprocesseur 68Ø3 sont énumérées dans le manuel de l'éditeur-assembleur livré avec ALICE.

Les opérations de transfert

Le premier type d'opération revient à des opérations de transfert : on recopie une donnée d'un endroit à un autre. Il y a des opérations de chargement d'un ou deux octets de la mémoire dans l'un des registres du 68Ø3. Les mnémoniques commencent dans ce cas par LD (de l'*anglais* : LOAD, charger). Ces instructions servent à transporter des données d'un endroit à un autre. La donnée d'origine n'est pas modifiée. On peut aussi faire passer une donnée d'un registre à un autre au sein même du 68Ø3. Les mnémoniques commencent dans ce cas par T pour transfert. A nouveau le registre d'origine n'est pas modifié par l'opération. Le dernier type d'opération de transfert est le passage du 68Ø3 à un emplacement de la mémoire. Les mnémoniques commencent dans ce cas par ST (pour STORE, ranger).

Les opérations arithmétiques

Le 68Ø3 peut opérer des additions et des soustractions entre certains registres, et entre un registre et un emplacement de la mémoire. Les opérations d'addition sont marquées par AD dans les mnémoniques (sauf l'addition de registres AB). Si le bit Carry du registre d'état entre dans le total (pour tenir compte d'une retenue entre octets, en cas de dépassement de la valeur notée par un octet), le mnémonique a pour troisième caractère C. La soustraction est notée par SUB (sans prise en compte de Carry) ou SBC (avec prise en compte de Carry). Il est aussi possible de multiplier le registre A par le registre B.
Le guide d'instructions de l'éditeur-assembleur regroupe avec les opérations arithmétiques l'instruction DAA, qui transforme la valeur du registre A en valeur DCB.
Se rattachent aux opérations arithmétiques les inversions arithmétiques, qui reviennent à changer le signe d'un registre, considéré en mode complément à 2. (Voir le détail des opérations plus haut.)

Décalages et rotations

Les opérations de décalage et rotation consistent à décaler tous les bits d'un octet d'un cran vers la gauche ou la droite. Si le bit "sortant" est récupéré du côté du bit "entrant", on parle de rotation, et non de décalage.
Il y a deux types de décalages : l'un préserve le bit de poids fort de l'octet, il est dit *arithmétique,* d'où son mnémonique qui commence par AS (pour Arithmetic Shift). Si le bit de signe n'est pas préservé, le décalage est dit *logique :* LS (pour Logical Shift). Le sens du décalage est indiqué par la troisième partie du mnémonique : R pour Right (à droite) et L pour Left (à gauche).
Les rotations sont respectivement indiquées par ROR et ROL suivant qu'elles se font vers la droite ou vers la gauche. Notez qu'un décalage revient à diviser le nombre par deux s'il se fait vers la droite, à le multiplier par deux s'il se fait à gauche.
Le bit sortant dans un décalage à droite permet de tester facilement si le nombre d'origine était pair (bit à Ø) ou impair (bit à 1), puisque l'indicateur C prend la valeur du bit sortant.

Opérations logiques

Le 68Ø3 permet de faire les opérations logiques classiques (ET logique, OU inclusif et exclusif) ainsi que l'inversion logique. Les mnémoniques correspondants commencent respectivement par AND, OR et EOR, et COM.
Attention : ne pas confondre l'inversion arithmétique NEG et la commutation logique COM.
Voici l'effet de chacune de ces fonctions logiques sur les nombres binaires :

Ø ET Ø → Ø	Ø OU (inclusif) Ø → Ø	Ø OU (exclusif) Ø → Ø
Ø ET 1 → Ø	Ø OU (inclusif) 1 → 1	Ø OU (exclusif) 1 → 1
1 ET Ø → Ø	1 OU (inclusif) Ø → 1	1 OU (exclusif) Ø → 1
1 ET 1 → 1	1 OU (inclusif) 1 → 1	1 OU (exclusif) 1 → Ø

Dans l'inversion logique : Ø donne 1
1 donne Ø

N'oubliez pas que les opérations logiques ne connaissent pas la retenue, mais travaillent bit à bit.

Diverses opérations sur les bits ou les octets entiers

Le 68Ø3 permet de positionner les indicateurs C, I et V du registre d'état : cela permet de transmettre des signaux à des sous-programmes, par exemple, ou de mettre le registre d'état en conformité avec les exigences de la suite du programme. Voir le rôle du bit I dans le chapitre sur les interruptions. CL dans les mnémoniques signifie CLEAR, mettre à zéro, et SE signifie SET, mettre à 1.
Les fonctions logiques permettent l'usage des masques : si vous souhaitez mettre à Ø tous les bits d'un octet en ménageant l'état précédent de l'un d'entre eux, dont vous ne connaissez pas l'état a priori, il suffit d'opérer un ET entre l'octet à modifier et un octet composé de bits à Ø aux emplacements que vous voulez mettre à Ø, et d'un (ou plusieurs) 1 aux emplacements dont vous voulez conserver l'état.
L'octet de référence, qui permet de sélectionner des bits sur un octet est appelé masque. A l'inverse, on peut mettre à 1 tous les bits d'un octet, en préservant l'état d'un ou plusieurs des bits à tester, grâce à un OU exclusif.
On veut vérifier l'état des bits B4, B5 et B6 d'un octet, sans tenir compte des autres bits. On va les sélectionner en créant le masque %Ø111ØØØØ; un ET entre le masque et la donnée à tester donnera dans l'accumulateur choisi %Ø1Ø1ØØØØ si la donnée d'origine avait 1Ø1 dans les bits B4, B5 et B6. Vous trouverez des usages aux masques, par exemple dans la création des registres de commandes du 9345.

Incrémentation, décrémentation et remise à Ø

Le 68Ø3 permet d'augmenter ou de diminuer de 1 la valeur de registres ou d'adresses de la mémoire. On appelle ces fonctions incrémentation (addition de 1) et décrémentation (soustraction de 1). Ces fonctions sont particulièrement utilisées pour gérer des compteurs.
Les mnémoniques de remise à zéro commencent tous par CLR (pour CLEAR en anglais).

Gestion de la pile

Lorsque vous avez besoin de sauvegarder momentanément la valeur d'un registre, pendant des opérations qui vont affecter le contenu de ce registre, vous disposez d'un rangement provisoire sous la forme d'une pile. On peut y sauvegarder directement les registres A, B et IX (mnémonique PSH pour PUSH, pousser sur la pile).

L'opération inverse a pour mnémonique PUL (PULL : tirer de la pile). Grâce aux possibilités de transfert entre registres, on peut ainsi sauvegarder n'importe quel registre.

Les comparaisons et tests

Les instructions de comparaison déposent sur le registre de référence le résultat de la comparaison (soustraction pour les instructions de comparaison, mnémonique en C, et ET logique pour les tests en BIT). La valeur à laquelle la donnée est comparée, celle du registre est perdue dans l'opération. L'instruction TST permet de conserver l'état des registres, sauf celui d'état. En revanche elle ne positionne que deux indicateurs.

Les branchements

• Les instructions JMP (JUMP : sauter en) et JSR (JUMP TO SUBROUTINE) sont les équivalents des GOTO et GOSUB du BASIC. Les instructions de branchement, en dehors de leur mode particulier d'adressage (adressage relatif) sont l'outil des branchements conditionnels en assembleur.
• BRA (BRANCH : se brancher en) et BSR (BRANCH TO SUBROUTINE) sont les équivalents, en adressage relatif, de JMP et JSR.
Les autres instructions de branchement sont des instructions conditionnelles : le branchement n'est effectué que si le ou les indicateurs spécifiés sont correctement positionnés.
• BEQ et BNE (BRANCH IF EQUAL : se brancher si égal à zéro, et BRANCH IF NOT EQUAL : se brancher si différent de zéro) dépendent de l'indicateur Z. Ils permettent de contrôler le passage à zéro d'un registre ou un octet.
BNE et BEQ sont particulièrement commode pour contrôler le passage à zéro d'un compteur, utilisé en décrémentation.
• BCS et BCC (BRANCH IF CARRY SET / IF CARRY CLEAR) contrôlent la position de la retenue.
BMI et BPL (BRANCH IF MINUS, se brancher si négatif/IF PLUS si non négatif) contrôlent l'indicateur N.
BVS et BVC (BRANCH IF OVERFLOW SET/IF OVERFLOW CLEAR) contrôlent la position de l'indicateur de dépassement.

Le programme source

Contrairement au BASIC où l'on peut tester ces programmes ligne à ligne si l'on veut, l'assembleur exige que l'on ait écrit un programme source complet pour fonctionner. On ne peut fabriquer un morceau de programme, le faire assembler et exécuter pour le vérifier. Il est nécessaire de programmer au moins des modules cohérents qui forment des programmes exécutables indépendamment pour tester sa programmation.

En effet, l'assembleur s'attend à trouver dans le programme source des indications générales qui lui permettent de savoir où installer dans la mémoire le programme objet, et à quel endroit faire commencer l'exécution de ce programme objet. Ces deux renseignements sont indispensables à l'assemblage.

Voici le plan à adopter pour votre programme source.

• **En tête :**

Identification du programme, de la version
Date de réalisation, de modification
ORG directive définissant l'adresse où installer le programme
EXC adresse où doit commencer l'exécution

- Étiquettes avec leurs équivalents (valeurs en hexadécimal)
  Adresses des programmes extérieurs auxquels on fait appel
  Paramètres constants du programme
- Blocs de programmes
- Données, messages, définition des zones de travail

La composition d'une ligne d'assembleur

Une ligne d'assembleur est régie strictement.
Une ligne peut se diviser en un maximum de quatre zones, ou champs, et un minimum de deux, sauf pour les lignes de commentaires. Le premier champ doit être présent, même s'il est vide : il est réservé à la définition d'étiquettes. Il est conseillé de le fixer à 6 colonnes : la longueur maximum des étiquettes reconnues par ALICE est de 5 caractères. Si vous ajoutez une colonne vide avant le deuxième champ, vous aurez d'emblée une mise en page claire.

Attention : les étiquettes doivent impérativement commencer dans la première colonne ; si vous mettez un blanc avant l'étiquette, vous aurez un message d'erreur (l'assembleur aura essayé de comprendre votre étiquette comme un mnémonique ou une directive).

Le deuxième champ doit être par conséquent mis à la colonne 7. Il est conseillé de lui consacrer 5 colonnes : les mnémoniques les plus longs font 4 caractères. Il contient les mnémoniques ou les directives d'assemblage.

Le troisième champ doit recevoir les paramètres traités par l'instruction. Pour aligner correctement vos remarques, qui seront rejetées en quatrième champ si les trois premiers sont remplis, il est conseillé de réserver au champ des paramètres au moins 8 colonnes : une adresse ou une étiquette peut occuper 5 colonnes (le signe dollar + 4 chiffres en hexadécimal) et si vous utilisez l'adressage immédiat, vous devez ajouter le signe dièse #. Songez aussi qu'il peut être commode de demander à l'assembleur d'effectuer quelques opérations sur les paramètres (par exemple l'ajout d'une constante). Le champ dépassera alors les 7 colonnes.

La gestion des étiquettes

Il est utile d'organiser systématiquement les étiquettes de votre code source. Utilisez pleinement la possiblité de mettre cinq caractères pour identifier les sous-programmes. Les étiquettes seront ainsi plus claires.
A l'intérieur d'un sous-programme, pour les branchements internes (boucles, branchements conditionnels), fabriquez des étiquettes "décimales" à partir de l'étiquette globale, pour faire ressortir la hiérarchie des blocs de programmation.

Si vous avez appelé un sous-programme SPROG, le premier niveau des étiquettes de branchement pourra être constitué ainsi :

```
SPR1Ø
SPR2Ø
...
```

Si vous devez diviser encore l'un des ces sous-blocs, donnez pour étiquettes, par exemple dans le bloc 2 :

```
SPR21
SPR22
...
```

Notez aussi que vous pouvez mettre en évidence la définition d'une étiquette en l'isolant sur une ligne : l'étiquette doit commencer à la première colonne de la ligne, mais il n'est pas obligatoire que le mnémonique ou la directive qu'elle va désigner soit sur la même ligne.

Séparez, dans la définition des constantes en début de programme, les étiquettes des entrées de routine externes des étiquettes de constantes numériques ou de registres de travail, la lecture du listing en sera facilitée.

Table des matières

L'impression de ce livre
a été réalisée sur les presses
des Imprimeries Aubin
à Poitiers/Ligugé

Achevé d'imprimer en février 1985
N° d'édition, 0023-2-1985 — N° d'impression, L 19647
Collection 10 — Édition 01
01/0479/4

ISBN 2-905-552-00-X

Imprimé en France