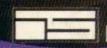
Manuel de référence

André CHENIÈRE

QWERTYUIOP { }

LE PROGRAMME BASIC EN MEMOIRE INITIATION AU LANGAGE MACHINE FONCTIONNEMENT DE L'INTERPRETEUR ATLAS DE LA ROM



Éditions

ATMOS-ORIC1 manuel de référence

par André Chénière



AVANT-PROPOS

Ce livre vous propose une exploration du logiciel de base de l'Oric. Son objectif est de vous familiariser avec le fonctionnement de l'interpréteur Basic de votre ordinateur.

C'est un projet ambitieux et forcément limité dans sa réalisation.
Nous n'entrerons pas dans le détail des routines de la ROM.
Vous trouverez cependant, au moins, l'information suffisante pour les démonter ou pour les utiliser quand c'est possible.
Ainsi, vous aurez des points de repère pour entreprendre vos propres recherches.

Le livre est articulé en quatre parties:

- Dans la première partie, après un rappel des notions de base indispensables et la présentation d'un utilitaire, en Basic évidemment, qui servira de moyen d'investigation, nous verrons comment sont stockés en mémoire le programme Basic et ses variables. Nous terminerons par l'étude du fonctionnement des principaux pointeurs en page zéro.
- La seconde partie s'intitule: initiation au langage machine.

 Nous sommes conduits à utiliser ce langage par nécessité. C'est une excellente opportunité de s'y initier. Point n'est besoin de parler couramment la langue pour se débrouiller en pays étranger.

 Ne vous attendez pas à un cours de programmation du 6502, mais plutôt à une sorte de lexique des instructions de ce microprocesseur. En cas de difficultés, consultez le dictionnaire. Vous pourrez très vite vous en passer.

Vous ne serez pas brutalement placés hors de votre cadre familier. C'est à mon avis la meilleure façon d'aborder le langage machine. Un des buts de ce livre est de vous persuader qu'il peut faire bon ménage avec le Basic. Des commandes telles que POKE, PEEK, ou CALL sont souvent sous-employées.

Plutôt que de vous laisser devant une alternative entre deux langages, la connaissance du système permet de les doser judicieusement pour résoudre la plupart des problèmes de programmation.

- La troisième partie doit vous donner un aperçu d'ensemble du fonctionnement de l'interpréteur, reconnaissance des mots-clés, points d'entrée des commandes, exécution, etc. L'accent est mis sur quelques routines importantes. Un chapitre est consacré à la représentation des nombres en virgule flottante.
- La dernière partie est un répertoire de la plupart des routines du système avec, le cas échéant, un mode d'emploi succint.

Cet ouvrage était fort avancé lorsque l'ATMOS est sorti en février dernier. Bien que la disposition générale soit restée la même, toutes les adresses en ROM ont changé

Il m'a paru judicieux de donner la priorité à cette nouvelle version qui semble définitive.

Bien entendu, le livre s'adresse indifféremment aux utilisateurs d'ATMOS ou d'ORIC-1. L'avantage accordé au nouveau venu se limite à le mentionner systématiquement en premier.

Un dernier mot.

Les fabricants de micro-ordinateurs sont toujours d'une discrétion incompréhensible sur les caractéristiques de leur logiciel système. Le manuel de l'ORIC-1 était particulièrement creux. Celui de l'ATMOS est sensiblement amélioré mais reste insuffisant.

Ce préambule pour demander votre indulgence.

Résultat de recherches personnelles de l'auteur, c'est-à-dire d'un travail solitaire, cet ouvrage en a les défauts inhérents.

Il est probablement inexact et peu clair sur certains points.

De plus, on l'aurait souhaité plus complet.

Par manque d'informations sûres, le fonctionnement complexe du boitier d'entrées-sorties 6522 et celui du 8912 sont passés sous silence. Il aurait fallu davantage de détails sur les routines graphiques et sonores.

Ces sujets mériteraient à eux-seuls d'être développés dans un second volume.

TABLE DES MATIERES

PREMIERE PARTIE

LE PROGRAMME BASIC EN MEMOIRE	
CHAPITRE 1 - NOTIONS DE BASE Binaire, bits, octets - Arithmétique binaire - Complément à deux - Nombres signés - Le code ASCII - Notation hexadécimale - Organisation de la mémoire.	3
CHAPITRE 2 - UN MOYEN D'INVESTIGATION Programme en Basic	13
CHAPITRE 3 - LE TEXTE BASIC Stockage d'une ligne - Numéros de ligne supérieurs à 63999 - Codage des mots-clés - Insertion ou suppression d'une ligne.	21
CHAPITRE 4 - LES VARIABLES Définition - Les variables simples - Variable réelle - Variable entière - Variable chaine de caractères - Fonction définie par l'utilisateur - Les tableaux de variables.	29
CHAPITRE 5 - LES POINTEURS L'espace utilisable entre #500 et HIMEM - Les principaux pointeurs Basic - Le pointeur OLDTXT - Le pointeur DATPTR - Modification du début de programme - Récupération du programme après NEW - Le programme Basic en haut de mémoire.	39
DEUXIEME PARTIE INITIATION AU LANGAGE MACHINE	
CHAPITRE 6 - LE MICROPROCESSEUR Présentation - Les registres internes - Les indicateurs d'état - Les modes d'adressage - Langage d'assemblage, le jeu d'instructions du 6502	51
CHAPITRE 7 - TRANSFERT DE DONNEES Chargement de registres - Rangement en mémoire - Transfert entre registres - Transferts avec la pile - Exercices.	59
CHAPITRE 8 - TESTS, BRANCHEMENTS, BOUCLES Incrémentation, décrémentation - Comparaisons - Branchements conditionnels - Sauts, appels et retour de sous-programme - Interruptions, BRK, NOP - Exercices.	65
CHAPITRE 9 - TRAITEMENT DE DONNEES Instructions arithmétiques - Instructions logiques - Décalages - Rotations - Exercices.	73
CHAPITRE 10 - BASIC ET CODE MACHINE Mise en place du code machine - Appels, transmission de paramètres - Conversion binaire-DATA.	83

TROISIEME	PARTIE	
FONCTIONNE	MENT D	E L'INTERPRETEUR

CHAPITRE 11 - PRINCIPES DE BASE Utilisation d'une imprimante, drapeau PRTFLG #2F1 - Terminologie - Structure de l'interpréteur - Routine d'obtention d'un caractère, CHARGET, CHARGOT - Principe de fonctionnement - Entrée, analyse, codage, mode immédiat ou différé - Exécution - Exécution d'une nouvelle instruction, NEWSTT - Evaluation d'expressions, FRMEVL	97
CHAPITRE 12 - ROUTINES FONDAMENTALES Boucle principale d'entrée des commandes, CMDLP - Entrée d'une ligne de texte, INLIN - Analyse du buffer d'entrée, codage des mots-clés, PARSE - LIST - EDIT.	107
CHAPITRE 13 - NOMBRES A VIRGULE FLOTTANTE Représentation des nombres réels - Conversion - Normalisation - Signes du nombre et de l'exposant - Registres virgule flottante - Buffer de l'accumulateur flottant - Etude des nombres à virgule flottante, programme Basic.	117
CHAPITRE 14 - ROUTINES UTILISABLES Démarrage, initialisation - Recueil de données à TXTPTR - Entrée au clavier - Affichage à l'écran, sortie imprimante Localisations - Routines virgule flottante - Vérifications diverses, syntaxe - Chaines de caractères - Conversions - Aménagements, transferts - Commandes Basic - Cassette, HIRES, son.	127
CHAPITRE 15 - APPLICATIONS Données entrées au clavier - Affichage à l'écran - Application pratique - Programme utilitaire	141
QUATRIEME PARTIE ATLAS DE LA ROM	
CHAPITRE 16 - POINTEURS, VARIABLES SYSTEME, ROM 1° PARTIE Emplacements en mémoire vive utilisés par le système - ROM Première Partie - Tables	145
CHAPITRE 17 - ROM - DEUXIEME PARTIE Entrée du texte Basic - Analyse et codage - Stockage du programme - Exécution - Boucles, etc	173
CHAPITRE 18 - ROM - TROISIEME PARTIE Evaluation d'expressions - Localisations - Manipulation de chaines de caractères - Fonctions su chaines	181
CHAPITRE 19 - ROM - QUATRIEME PARTIE Routines virgule flottante	189
CHAPITRE 20 - ROM - CINQUIEME PARTIE Divers Entréees/Sorties - Graphisme et Son.	195

ANNEXE 1 - Codes ASCII et 4502	203
ANNEXE 2 - Caractères de controle, Attributs, Codes Escape	209
ANNEXE 3 - Carte de l'écran texte	211
ANNEXE 4 - Carte de l'écran HIRES	213
ANNEXE 5 - Instructions 6502	215
ANNEXE 6 - Conversion décimal-hexadécimal	224



PREMIERE PARTIE

LE PROGRAMME BASIC EN MEMOIRE

INTRODUCTION

Un ordinateur est une machine électronique chargée du traitement automatique de l'information.

Quel que soit le langage utilisé, programme et données doivent se trouver en même temps en mémoire centrale.

Des programmes de nature différente peuvent être logés en mémoire vive.

En mémoire morte, les programmes sont figés, ils exécutent une tâche spécifique.

On peut dire ainsi que l'interpréteur, programme en langage machine en ROM, traite l'information représentée par le texte Basic.

Si vous tapez un caractère autre qu'un chiffre, une lettre par exemple, suivi de RETURN, l'interpréteur vous signale une erreur de syntaxe. Il ne comprend pas ce qui est censé être une commande directe.

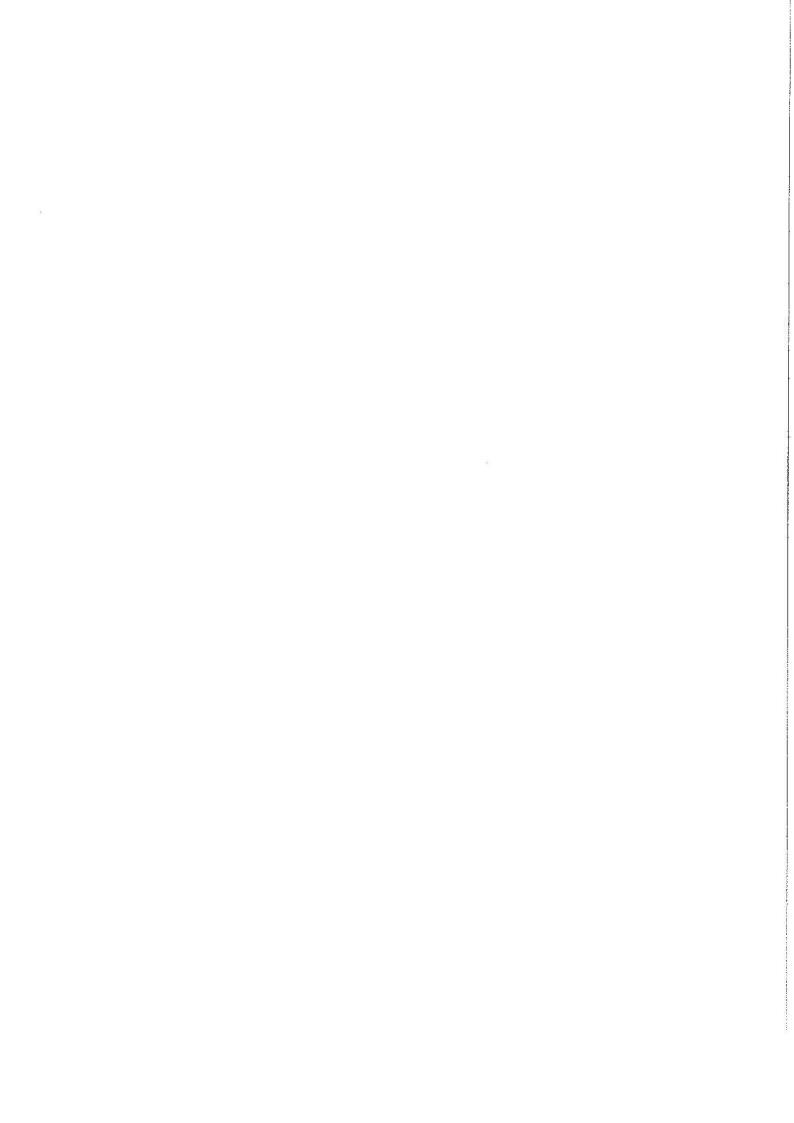
Si, par contre, ce que vous entrez commence par un chiffre, tout est accepté et stocké comme ligne de programme pour une exécution ultérieure.

La ligne n'est cependant pas rangée telle quelle. Elle est analysée, condensée, habillée. Sur certains ordinateurs, il y a même une vérification de la syntaxe à l'entrée; ce n'est pas le cas sur Oric. En même temps, l'interpréteur garde trace de cette modification du programme en mettant à jour différents "pointeurs".

Lorsque le programme est lancé, la place allouée au texte Basic ne suffit plus.

Des zones supplémentaires, délimitées par d'autres pointeurs, sont réservées à l'inscription des valeurs ou adresses des diverses variables définies par le programme ou entrées au clavier (INPUT). Avant de vérifier tout cela en le visualisant à l'écran, grâce au programme utilitaire dont nous allons nous doter, il faut rappeler quelques notions fondamentales concernant la représentation de l'information à l'intérieur même de l'ordinateur.

Binaire, bits, octets, hexadécimal seront les sujets de réflexion du premier chapitre.



CHAPITRE 1

NOTIONS DE BASE

BINAIRE, BITS, OCTETS

Aucune langue humaine n'est le langage naturel d'un ordinateur, même pas l'anglais abrégé du Basic.

Toute l'information, stockée ou manipulée à l'intérieur de la machine, est sous la forme binaire.

C'est une particularité des systèmes électroniques de travailler sur des variables binaires, susceptibles de prendre, et exclusivement, l'une de deux valeurs possibles.

Concrétement, il s'agit du potentiel électrique, du niveau de tension d'une cellule élémentaire, soit Ø volt, soit 5 volts.

Le BIT (contraction de BInary digiT), unité fondamentale d'information, traduit un de ces deux états, noté par convention Ø (ØV) ou 1 (5V).

Dans le langage courant, c'est blanc ou noir, ouvert ou fermé, oui ou non.

Sachant qu'il est doté de cette nature "tout ou rien", sans nuances, vous comprenez mieux l'intransigeance de votre ordinateur. Avec lui, pas de oui mais, peut-être? C'est la rigueur absolue, vous ne lui ferez pas admettre un point quand il attend une virgule.

En binaire, on dispose donc de deux symboles, Ø et 1, pour représenter l'information, alors qu'en décimal on en a dix (de Ø à 9). Un Bit aura la valeur Ø ou 1 suivant l'état de la cellule élémentaire concernée.

Le 6502 est un microprocesseur 8 bits comme la plupart de ceux qui équipent les systèmes individuels. Il est capable de digérer des paquets de 8 bits appelés octets.

L'espace adressable de votre Oric est de 64 K-octets. (Le K informatique vaut 1024).

Cela signifie que la mémoire centrale comporte 65636 cases d'un octet (8 bits), susceptibles de recevoir de l'information.

Si vous possédez une machine 48K, ces emplacements sont physiquement présents. 16K sont réservés au système d'exploitation en ROM, leur contenu est figé.

Les combinaisons de plusieurs bits sont évaluées suivant un système de numération par position analogue au système décimal.

Chaque chiffre d'un nombre donné possède un certain "poids" suivant la position qu'il occupe.

Ainsi, dans le système décimal (base 10), on a de la droite vers la gauche (héritage des Arabes) le chiffre des unités, celui des dizaines, celui des centaines...

Chaque chiffre d'un nombre décimal vaut entre 0 et 9 fois une certaine puissance de 10.

exemple: 2605 représente

5*10^0 + 0*10^1 + 6*10^2 + 2*10^3

De la même façon, dans le système binaire (base 2) chaque bit (chiffre binaire) vaut 0 ou 1 fois une certaine puissance de 2.

Pour faciliter leur repérage, les 8 bits d'un octet sont numérotés de 0 à 7 de la droite vers la gauche. Le bit n°0 est appelé le bit le moins significatif. Le bit n°7, le plus à gauche, est le plus significatif.

Chaque bit possède un poids (croissant de la droite vers la gauche) équivalent en décimal à une puissance de 2.

	bit n°	7	6	5	,	4	3	2	1	Ø	
		2^7	2^6	205	i 2º	4	2~3	2^2	2^1	2^0	
		128	64	32	1	6	8	4	2	1	
Exemple: le nombre	binaire		Ø	1	1	1	Ø	0	1	1	
	(bi	t n°)	7	6	5	4	3	2	1	0	
représent	e en déci	imal:	Ø +	64 +	32 +	16	+ Ø +	0 +	2 +	1 = 11	15

01110011 en binaire et 115 en décimal représentent le même nombre.

Four une base de numération B donnée, un nombre p de positions utilisées, la formule $N=B^{\rho}-1$ donne en décimal le plus grand nombre qui peut être représenté.

11111111 est le plus grand nombre binaire sur 8 bits. Son équivalent décimal est 255. Le plus petit est 00000000 = zéro.

Avec un octet, on peut donc représenter 256 valeurs différentes, ce qui correspond bien à 2^8 combinaisons.

ARITHMETIQUE BINAIRE

L'addition binaire obéit à des règles analogues à celles de l'arithmétique décimale:

0 + 0 = 0	ð Ø	+	1	=	1	1	+	Ø	=	1	1	+	1	=	(1)(2)	= Ø	(retenue	***	1)
exemple:																					
**				23	125						Ø	1	1	1	1	1	0	1			
			-3	٠	63						Ø	Ø	1	1	1	1	1	1			
											-										
				į	188						1	Ø	1	1	1	1	Ø	Ø			
Faisons n	nainte	nar	nt:								1	A	1	1	1	4	¥				
				2	255						1	1	1	1	1	1	1	1			
			1	H	1						Ø	Ø	0	0	Ø	Ø	0	1			
																	 .	-			
				2	256					(1) Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø			

Il y a une retenue dont nous ne savons que faire, provoquée par ce qu'on appelle un dépassement de capacité de l'octet. Prenons un octet supplémentaire pour représenter le nouveau nombre sur deux octets. Remarquez que le microprocesseur, lui, ne prendrait pas cette initiative. Il inscrirait la retenue dans un indicateur appelé ... indicateur de retenue.

Notre nombre s'écrit, sur 2 octets:

en décimal

Il peut être considéré comme un nombre de 16 bits

17 1 Ø Ø Ø O Ø Ø 0 Ø D 3 2 bit n° 15 14 13 12 11 10 9 8 7 5 1 0 $2^8 = 256$ en décimal

Un nombre binaire sur deux octets (16 bits) pourra représenter un nombre entier jusqu'à $2^16 - 1 = 65535$ décimal.

Les deux octets sont stockés en mémoire dans deux emplacements consécutifs mais vous noterez que, pour la plupart des microprocesseurs, dont le 6502, les octets sont inversés. L'octet faible est stocké en premier, à l'adresse la plus faible.

COMPLEMENT A DEUX

Tous les nombres binaires manipulés par l'ordinateur sont en valeur absolue.

A priori, rien ne permet de leur attribuer une valeur positive ou négative. Cela ne peut résulter que d'une convention. Il est exclus d'ajouter un nouveau symbole en plus de Ø et de 1. De plus, on est obligé de respecter un format fixé (8 bits, 16 bits...)

La méthode dite du complément à la base permet de s'affranchir du signe. Elle était déjà préconisée par Leibniz (17° siècle).

Pour une base de numération B, le complément à B d'un nombre d'un format déterminé, est le nombre qui, ajouté au premier, donnera Ø dans toutes les positions.

Supposez que vous preniez livraison d'une voiture neuve, jamais roulé, Ø au compteur.

Vous décidez de faire votre premier kilométre en marche arrière. Qu'indiquera le compteur?

Il ne peut pas afficher -1.

Ce sera 99999, complément à dix du nombre 1 pour cinq chiffres décimaux.

Si votre compteur avait été en binaire sur 8 bits, vous auriez lu 11111111, c'est le complément à deux de 00000001.

1

C'est réciproque d'ailleurs, 00000001 est le complément à deux de 11111111.

Il faut bien repartir en marche avant.

Le complément à deux d'un nombre binaire s'obtient en inversant d'abord tous les bits, puis en ajoutant 1 au nombre obtenu.

exemple le nombre binaire:	Ø	Ø	1	1	Ø	Ø	1	0
on inverse les bits:	í	1	Ø	Ø	1	1	Ø	1
on ajoute 1:	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
complément à deux =	1	1	Ø	Ø	1	1	1	0

Si on additionne le nombre et son complément à deux, on trouve bien 00000000.

La retenue doit être ignorée pour respecter le format, comme le faisait notre compteur kilométrique.

L'utilisation de cette méthode est largement répandue.

Elle permet une simplification de l'unité arithmétique du microprocesseur.

La soustraction revient à ajouter le complément à deux du nombre à soustraire.

NOMBRES SIGNES

La représentation binaire signée, ou représentation en complément à deux permet d'interpréter une valeur binaire (absolue) et de lui attribuer une valeur positive ou négative, qui, sur un octet sera comprise entre -128 et +127 décimal.

Par convention, le bit n° 7, le plus à gauche, est le bit de signe. Ø indique un nombre positif, 1 indique un nombre négatif.

En complément à deux, les valeurs binaires comprises entre 00000000 et 01111111 désigneront les nombres décimaux positifs de 0 à 127. Pas de changement par rapport au binaire classique. Remarquez que 00000000 est considéré comme positif dans cette représentation.

Les valeurs binaire comprises entre 10000000 et 11111111 désigneront les nombres décimaux négatifs de -128 à -1.

Un nombre sur deux octets peut être traité en complément à deux. Le bit de signe est alors le bit n°7 de l'octet de poids fort.

Il faut insister sur le fait que cette interprétation n'est pas systématique. C'est le programme qui décide s'il doit traiter un nombre en complément à deux ou non.

Il est évident que le nombre sur deux octets désignant une adresse ne sera pas considéré comme signé. Il pourra avoir une valeur décimale comprise entre 0 et 65535.

Par contre, une variable entière (affublée du signe %) sera traitée en complément à deux et vaudra entre -32768 et 32767 décimal.

LE CODE ASCII

Une configuration binaire de 8 bits peut servir à autre chose qu'à représenter une donnée numérique.

L'ordinateur excelle aussi dans le traitement des textes. Pour faciliter les communications, la standardisation est nécessaire.

C'est le code ASCII (American Standard Code for Information Interchange), norme internationale ISO 646, qui est universellement utilisé pour le codage des caractères. Il est reproduit en annexe 1.

Les 32 premiers codes correspondent aux caractères de contrôle. Rarement tous utilisés, ils sont destinés à une fonction particulière, plus ou moins standardisée.

Sur Oric, ils ont même une double fonction et deviennent "attributs" lorsqu'ils sont envoyés directement à l'écran par POKE, PLOT ou par l'intermédiaire d'ESCAPE. (Annexe 2)

Les codes de 32 à 127 sont ceux des caractères normaux, chiffres, lettres majuscules ou minuscules, signes de ponctuation et symboles divers.

Ces mêmes codes s'appliquent aux caractères semi-graphiques du mode LDRES 1.

L'instruction utilisée détermine le sens à donner à la valeur binaire rencontrée: donnée numérique, code d'un caractère ou tout autre chose. C'est aussi vrai en langage machine.

Entrez en commande directe: A=65

Vous avez affecté la valeur décimale 65 à la variable A. Comme il s'agit d'une variable réelle, l'ordinateur a stocké cette valeur en mémoire, en binaire, sous la forme virgule flottante (5 octets).

Entrez: ?A ou ?HEX\$(A)

L'Oric affiche la valeur de A en décimal ou en hexadécimal.

Entrez maintenant: ?CHR\$(A)

Vous ordonnez à l'ordinateur d'afficher le caractère dont le code ASCII = A.

Il n'a pas d'autre possibilité que de rechercher dans sa mémoire la valeur de A précédemment stockée.

Le code ASCII est un code sur 7 bits (0 à 127 en décimal). Le bit le plus significatif de l'octet, le bit n°7, est généralement à zéro.

Il est ignoré par la commande PRINT. En fait, il est simplement remis à zéro avant l'affichage. (voir annexe 1 pour les codes 128 à 151)

Entrez: A=65 puis ?CHR\$(A+128)

La commande PLOT affiche le caractère en vidéo inverse quand le bit 7 est à un.

PLOT 30,10,CHR\$(A)

7

PLOT 30,11,CHR\$(A+128)

Le bit 7 porte bien son nom de bit le plus significatif. Bit de signe, il est aussi abondamment utilisé à d'autres fins par la machine. Des huit bits d'un octet, c'est celui qui est le plus facile à tester.

Pour résumer.

un octet peut représenter des informations de nature très différente.

Seul, il peut désigner:

- un nombre entier en valeur absolue (Ø à 255) ou signé (-128 à +127).
- une instruction machine sur 1 octet.
- un caractère ASCII.
- un mot-clé du Basic.

Plusieurs octets pourront désigner:

- une adresse en mémoire (Ø à 65535) toujours sur 2 octets (16 bits).
- une instruction machine sur 2 ou 3 octets.
- un nombre entier sur 2 octets, sans signe ex: n° de ligne Basic, ou signé ex: valeur d'une variable entière (-32768 à 32767).
- un nombre en virgule flottante (5 octets).
- une chaîne de caractères (maxi 255 octets)

Ne vous attardez pas sur cette énumération un peu décousue. Tout deviendra plus clair au fur et à mesure que nous progresserons.

NOTATION HEXADECIMALE

Outre les précieux doubles PEEK et POKE (DEEK et DOKE) qui permettent de désigner directement les deux octets d'une adresse, l'Oric offre l'inestimable avantage d'accepter les valeurs exprimées en hexadécimal aussi bien qu'en décimal.

Si jusqu'à présent vous avez été réticent à utiliser cette facilité, vous devez vous y mettre . J'insiste, c'est absolument nécessaire.

L'hexadécimal n'a pas été inventé pour le plaisir d'ajouter un nouveau système de numération (base 16). Il reflète fidélement la structure binaire.

En raccourci, on peut dire que binaire ou hexa, c'est la même chose. Si je vous propose la valeur binaire 16 bits:

1100110000010010

C'est plutôt hermétique et difficile à retenir.

Convertie en décimal:

 $2^15 + 2^14 + 2^1 = 52242$

C'est laborieux et le résultat n'est pas très significatif.

En hexadécimal:

1100 1100 0001 0010

C C 1 2

La conversion est facile et on reconnait sans peine l'adresse de la routine de sortie d'un caractère sur ORIC-1.

Remarque:

ORIC utilise le symbole # au lieu de \$ pour désigner une valeur hexadécimale. Nous le conserverons pour éviter toute confusion.

Vous savez oue chaque chiffre hexadécimal (de 0 à F) représente 4 bits, c'est-à-dire un demi-octet ou quartet (en anglais, nibble).

Vous arriverez rapidement à vous passer du tableau ci-dessous.

décimal	bi	na	air	^E	hexadécimal
Ø	Ø	Ø	Ø	Ø	Ø
1	Ø	Ø	Ø	1	1
2	Ø	0	1	Ø	2
3	Ø	Ø	1	1	3
4	Ø	1	Ø	Ø	4
5	Ø	1	Ø	1	5
6	Ø	1	1	Ø	6
7	Ø	1	1	1	7
8	1	0	Ø	Ø	8
9	1	Ø	Ø	1	9
10	1	Ø	1	Ø	A
11	1	Ø	1	1	В
12	1	1	Ø	Ø	C
13	1	1	Ø	1	D
14	1	1	1	Ø	E
15	1	1	1	1	F

Pour respecter le format des nombre sur 1 ou 2 octets, il est préférable de reproduire les 0 non significatifs, par exemple #00 ou #00E2. Ce que ne fait pas l'Oric, qui les ignore, malheureusement.

ORBANISATION DE LA MEMOIRE

Il est classique de diviser la mémoire centrale en 256 pages de 256 octets chacune.

Ainsi, la mémoire de l'Oric contient 256*256 = 65536 emplacements désignés par les adresses de 0 à 256*256 - 1 (0 à 65535).

Exprimé en hexadécimal, le même inventaire donne #100 pages de #100 emplacements, soit au total #100000 dont les adresses, désignées sur deux octets, vont de #0000 à #FFFF.

Les deux premiers chiffres hexadécimaux indiquent la page, les deux derniers la position dans la page.

Vous voyez l'intérêt de la notation hexadécimale, notamment en ce qui concerne la désignation des adresses.

Les trois premières pages sont utilisées en grande partie par le système. On ne peut pas, sans discernement, changer les valeurs qui y sont inscrites sous peine de plantage.

La page Ø est très sollicitée. Le 6502 comporte de nombreuses instructions qui la désignent nommément. Disons, pour simplifier, que c'est la seule page dont les adresses peuvent être désignées avec un seul octet. C'est économique.

Outre de nombreux pointeurs, la page zéro contient notamment:

- le buffer d'entrée (#35-#83). C'est dans cette zone que sont inscrites les commandes directes ou les lignes Basic entrées au clavier avant analyse par l'interpréteur.

Sa taille relativement réduite explique pourquoi les lignes de

programme sont limitées à 78 caractères.

 la routine CHARGET-CHARGOT (#E2-#F2). C'est la routine d'obtention d'un caractère, le cheval de bataille de l'interpréteur qui l'appelle constamment.

Nous reparlerons de ces éléments importants dans la troisième partie de ce livre.

La page 1 est réservée à la pile. Comme ce nom l'indique, certaines informations y sont empilées. Elle est utilisée avec les commandes GOSUB-RETURN, FOR-NEXT, READ-UNTIL, POP,PULL et aussi directement par le microprocesseur qui y range l'adresse de retour des sous-routines. On peut l'utiliser en langage machine.

La page 2 est occupée en partie par des variables système. Nous aurons l'occasion d'en utiliser quelques-unes.

La page 3 contient les adresses du système d'entrées-sorties géré par le 6522 VIA (Versatile Interface Adaptor). Nous n'y toucherons pas.

La page 4 est utilisable pour de petites routines en langage machine. Elle est entièrement disponible malgré la restriction indiquée par le constructeur. Il semble que ce dernier se réserve la possibilité de l'utiliser. Peut-être au profit d'un système d'exploitation de micro-disquettes?

La page 5 marque le début de ce qu'on appelle l'espace utilisateur qui s'étend jusqu'à HIMEM. Le programme Basic est normalement stocké depuis l'adresse #500.

Le haut de la mémoire vive a une configuration différente selon le mode choisi: TEXT (ou LORES) ou HIRES. (Voir annexe A du manuel).

* * *

Nous voici arrivés à la fin de ce chapitre.

Pour vous dégourdir les doigts avant la longue page d'écriture qui vous attend et pour vous mouiller les pieds avant le grand bain, je vous propose un petit programme.

Il convertit en binaire un nombre sur 8 bits (#00-#FF, 0-255) et affiche éventuellement sa valeur décimale en complément à deux. La courte routine en langage machine qu'il contient est commentée. Si vous ne comprenez pas bien, n'en faites pas un drame, vous reviendrez plus tard.

Suivant votre système vous entrerez la ligne 10 qui convient; le reste est identique:

Pour 1'ATMOS

10 C\$="A2080680A930690020D9CCE005D00320D4CCCAD0ED60": A=#400

Pour 1'ORIC-1

10 C\$="A2080680A93069002012CCE005D003200DCCCAD0ED60":A=#400

20 FOR I=1 TO LEN(C\$) STEP 2:C=VAL("#"+MID\$(C\$,I,2)):POKE A,C: A=A+1:NEXT

30 PRINT"Entrez un nombre de #00 a #FF":A\$=" "

40 INPUT N: IF N<0 OR N>255 THEN 30 ELSE POKE#80.N

50 PRINT, N; A\$; HEX\$ (N); A\$; : CALL#400: IF N>#7F THEN PRINT A\$; N-256

60 PRINT: PRINT: GOTO 40

16

#400-	A2 Ø8	FDX >#08	;X=8 compteur 8 bits
#402-	Ø6 8Ø	ASL #80	;décale(#80) à gauche, bit 7 -> C
#404-	A9 30	LDA >#30	; A=#30 code ASC("0") dans l'acc.
#406-	69 00	ADC >#00	;A=A+0+C ajoute 00 + retenue (C)
#408-	20 D9 CC	JSR #CCD9	;PRINT CHR\$(A) Ø ou 1 ORIC-1 #CC12
#40B-	EØ Ø5	CPX >#Ø5	; IF X<>54 bits affiches?
#4ØD-	DØ Ø3	BNE +3	;non THEN #412
#4ØF-	20 D4 CC	JSR #CCD4	:PRINT " " aff.espace ORIC-1 #CCOD
#412-	CA	DEX	; X=X-1
#413-	DØ ED	BNE -19	; IF X <> Ø THEN #402
#415-	60	RTS	RETURN

Comme vous l'avez deviné, ce sous-programme en langage machine affiche en binaire la valeur entrée. Il est implanté à partir de l'adresse #400 par les lignes Basic 10 et 20. Le nombre entré est inscrit en #80 (POKE #80,N).

En #400-401 LDX >#08 signifie charger X avec la valeur #08. X est un registre du 6502 qui sert de compteur pour les 8 bits de notre octet.

En #402-403 ASL #80 est probablement l'instruction la plus difficile à comprendre. Elle signifie 'Arithmetic Shift Left', décalage arithmétique à gauche du contenu de l'emplacement mémoire #80. Tous les bits sont décalés d'une position à gauche. Le bit 7 sort de l'octet mais il n'est pas perdu car il est recueilli par l'indicateur de retenue C. C'est une particularité de l'instruction ASL. L'indicateur C vaudra donc 0 ou 1 suivant la valeur du bit 7 sorti.

En #404-405 LDA >#30, on charge l'accumulateur (c'est le registre le plus important du microprocesseur) avec la valeur #30 (48 décimal) qui est le code ASCII du chiffre 0.

En #406-407 ADC >#00 ADC signifie 'ADD with Carry', additionner avec retenue. Elle a ici pour effet d'ajouter seulement la retenue (0 ou 1) à l'accumulateur qui contient maintenant soit #30, code de "0", soit #31, code de "1".

En #408-40A JSR #CCD9 (#CC12 sur ORIC-1) On appelle la routine d'affichage du caractère dont le code est contenu dans l'accumulateur. JSR signifie Jump to Sub-Routine.

En #408-40C CPX >#05 compare X avec la valeur #05

En #40D-40E BNE +3 (Branch if Not Equal) s'il n'a pas égalité on évite l'instruction suivante. On saute en #412

En #40F-411 JSR #CCD4 (#CC0D sur ORIC-1) il y avait égalité, quatre bits étaient déjà sortis, appel de la sous-routine qui affiche un espace.

En #412 DEX 'Decrement X', le contenu de X est diminué de un.

En #413-414 BNE -19 si le contenu de X n'est pas égal à zéro on passe au bit suivant. Remarquez le déplacement qui vaut #ED, c'est-à-dire -19 en complément à deux. Toutes les instructions de branchement sont traitées en complément à deux. Une valeur négative indique un déplacement en arrière. Notez aussi que, pour trouver le point de chute, on compte Ø sur l'octet qui suit immédiatement l'instruction de branchement. Ici, on retourne en #402, nouveau décalage. Vérifiez.

11

En #415 RTS ReTurn from Sub-routine. Tous les bits ont été affichés. Retour au Basic. N'oubliez jamais cette instruction de retour.

CHAPITRE 2

UN MOYEN D'INVESTIGATION

L'Oric est dépourvu d'un moniteur intégré qui permettrait, entre autres, d'examiner ou de modifier le contenu d'une zone en mémoire. Il faut s'équiper d'un utilitaire spécial.

Des programmes de ce genre sont arrivés sur le marché.
Pour l'ORIC-1, je vous recommande particulièrement 'MONITEUR 1.0' de la Société LORICIELS. La version 1.1 pour l'ATMOS devrait être rapidement disponible. C'est un excellent programme, facile à utiliser.

Outre un désassembleur et un mini-assembleur sympathique, il possède des fonctions intéressantes.

Le programme Basic que vous allez entrer en mémoire n'est pas si sophistiqué.

Néammoins, il est mieux adapté à notre entreprise et sera suffisant pour la suite de ce livre.

Il permet d'examiner une suite d'octets en mémoire et de les modifier éventuellemnt. Il comporte aussi un désassembleur.

L'opération de désassemblage consiste à restituer la série de valeurs binaires d'un programme en langage machine sous une forme symbolique, le langage d'assemblage, bien plus facile à comprendre puisque c'est le langage de programmation.

Vous constaterez que notre utilitaire est de forme classique avec quelques particularités qui seront commentées.

Après l'avoir tapé, je vous conseille de le sauvegarder AVANT de le lancer.

Pourquoi? La première chose que fait le programme est d'inscrire à l'adresse #BFEØ une courte routine en langage machine chargée d'afficher les instructions (lignes REM Ø-3) en haut de l'écran. Si, pour une raison ou une autre, ce code machine est absent, le CALL #BFEØ aboutira au début de la ROM en #CØØØ dont la première instruction est un saut à la routine d'initialisation. Vérifiez maintenant, en entrant: CALL#BFEØ.

Un autre conseil, mais c'est facultatif.

Il est fastidieux de rechercher un programme sur une cassette. Surtout s'agissant d'un utilitaire, il est pratique de lui consacrer une face entière d'une cassette de faible capacité. Dans le cas présent, voici comment procéder:

- Lorsque vous aurez expurgé le programme des erreurs de frappe inévitables, préparez la boucle en commande directe:

FOR I=1 TO 50:CSAVE"", AUTO: WAIT 200:NEXT n'appuyez pas sur RETURN

- Mettez votre magnétophone en marche, position enregistrement.
- Appuyez sur RETURN et laissez faire la machine.

Votre magnétophone s'arrêtera bien tout seul. Alors, pour arrêter la boucle, maintenez CTRL-C appuyé ou faites RESET.

Double avantage. Cet enregistrement automatique est probablement plus fiable et vous pouvez commencer le chargement à n'importe quel endroit de la bande.

Cette procédure perd de son intérêt si votre magnétophone n'accepte que le mode lent. L'attente risque d'être prohibitive.

```
D REM 1.menu 2.recomm 3.stop -suite
 1 REM 1.menu 2.desasmb 3.modif 4.suite
2 REM ->av. <-debut @carac #hexa \fin
3 REM -Initialisation codes 6502
4 REM ne tapez pas cette ligne - voir modif ORIC-1 en 900
5 GOSUB 1000
 7 CLS:PRINT D$:PRINT E$:PRINT"Tapez 1 ou 2 (3 pour Stop)
8 CALL#E5F5: A$="": X=FRE("")
 9 GET A$: ON VAL(A$) GOTO 300,400,250:GOTO 9
 10 H$=HEX$(DEEK(A-2)); RETURN
 20 H$=HEX$(PEEK(A-1))
 30 IF MID*(H*,2)=""THEN H*="#00"
 40 IF MID$(H$,3)=""THEN H$="#"+"0"+MID$(H$,2)
 50 RETURN
 100 GOSUB 20: FRINT">": H$:: RETURN
 110 GOSUB 10:PRINT H$::RETURN
 120 GOSUB 20:PRINT H$;:RETURN
 130 D=PEEK(A-1):B=A+D+256*(D>127):IF B<0 OR B>65535 THEN B=0
 135 PRINT HEX#(B);:RETURN
 140 GOSUB 10:PRINT H$;",X";:RETURN
 150 GOSUB 10:PRINT H$; ",Y"; : RETURN
 160 GOSUB 20:PRINT H$;",X";:RETURN
 170 GOSUB 20:PRINT H$;",Y";:RETURN
 180 GOSUB 20:PRINT"(";H$;",X)";:RETURN
 190 GOSUB 20:PRINT" (": H$: "), Y": : RETURN
 200 GOSUB 10: PRINT" ("; H$; ") "; : RETURN
 210 PRINT "A";:RETURN
 220 PRINT"Entrez adresse de debut en decimal ou en hexa (pre
cedee de #)"
 230 INPUT"DEBUT: "; S$: S=VAL (S$): IF S>65535 THEN 230
 240 RETURN
 250 CALL#247
 260 CALL#BFE0, J: RETURN
 300 CLS:PRINT D$:GOSUB 220
 310 CLS: A=S
 320 CALL#E5F5: A$="":PRINT HEX$(A);" ";A,
 330 C=PEEK(A): M$=LEFT$(I$(C),3): N=VAL(MID$(I$(C),4,1)): E=VAL(MI
D$(I$(C),5))
 340 FOR A=A TO A+N-1:H$=HEX$(PEEK(A)):GDSUB 30 :PRINT MID$(H$,2
);" ";:NEXT
 350 PRINT SPC(10-N*3); M$; " ":
 360 ON E GOSUB 100,110,120,130,140,150,160,170,180,190,200,210:
PRINT
 370 IF PEEK(#268)=27 THEN J=#07:GOSUB 260:GET A$:CLS:ON VAL(A$)
GOTO 7,310,250
 380 IF A>65535 THEN POKE#268,27:GOTO370
```

390 GOTO 320

```
400 CLS:PRINT E$:GOSUB 220:CLS:A=S:Y=1
410 CALL#E5F5: A$="":PLOT 2, Y, HEX$(A):PLOT 2, Y+1, STR$(A)
420 I=0:REPEAT: X=I*3+9: C=PEEK(A+I):H$=HEX$(C):GOSUB 30
430 PLOT X,Y,MID$(H$,2):IF C<32 OR C>128 AND C<160 THEN C=63
440 PLOT X+1,Y+1,CHR$(C): I=I+1: IF A+I=65536 THEN PULL: GOTO 470
450 UNTIL I=10: A=A+I: Y=Y+2: IF Y=27 THEN 470
460 GOTO 410
470 X=8:Y=1:J=#2E:GOSUB 260:L=0:GET A$:ON VAL(A$)GOTO 7,310,600
.410:GOTO 470
500 H$="#"+H$(1)+H$(2):C=VAL(H$)
510 POKE M,C
520 PLOT X,Y+L," ":Z=0:M=M+1:X=X+3:IF X>36 THEN X=8:Y=Y+2
530 IF Y=27 THEN Y=1:M=A-130:PING:WAIT 100
540 PLOT X,Y+L,CHR$(190):PLOT 5,0,HEX$(M):RETURN
550 PLOT 5,0,"
560 PLOT X,Y+L," ": RETURN
600 M=A-130: IF M>#BFFF THEN 470
610 Z=0: X=8: Y=1: J=#55: GOSUB 260: GOSUB 540: POKE#20C, 255
620 CALL#F75A: A$=KEY$: IF A$="" THEN 620
630 C=ASC(A$): IF C=8 THEN GOSUB 560: GOTO 600
640 IF C=9 THEN GOSUB 520:GOTO 620
650 IF C=20 AND L=1 THEN POKE#268,0:PRINT CHR$(20):GOTO 620
660 IF C=35 THEN GOSUB 560:L=0:GOSUB 540:POKE#20C,255:GOTO 620
670 IF C=64 THEN GOSUB 560:L=1:GOSUB 540:GOTO 620
680 IF C=92 THEN GOSUB 550:POKE#20C,255:CALL#F75A:GOTO 470
690 IF L THEN PLOT X+2,Y+1,A$:GOSUB 510:GOTO 620
700 IF C>47 AND C<58 OR C>64 AND C<71 THEN Z=Z+1:H$(Z)=A$:PLOT
X+Z,Y,H$(Z)
710 IF Z<2 THEN 620 ELSE GOSUB 500:GOTO 620
900 REM ORIC-1: lignes 8,320,410 remplacer #E5F5 par #E563
910 REM DRIC-1: ligne 250 remplacer #247 par #F882
920 REM DRIC-1: ligne 410 remplacer STR$(A) par MID$(STR$(A),2)
 930 REM ORIC-1: lignes 620,680 remplacer #F75A par #F729
940 REM ORIC-1: ligne 1010 C$="200AD88AA49BA203202FF860" . . .
 1000 D$="1.Desassemblage":E$="2.Examen ou Modif memoire"
 1010 C$="20C5D88AA49BA2032065F860":A=#BFE0:GOSUB 1500:J=#7C:GOS
UB 260
 1020 DIM I$(255):FOR I=0 TO 255:READ I$(I):NEXT:RETURN
```

1500 FOR I=1 TO LEN(C\$)STEP 2:C=VAL("#"+MID\$(C\$,I,2))
1510 POKE A,C:A=A+1:NEXT:RETURN

15

```
2000 DATA BRK10, ORA29, ???10, ???10, ORA23, ASL23, ???10, PHP10,
ORA21, ASL112
 2010 DATA ???10,???10,ORA32,ASL32,???10,BPL24,ORA210,???10,???10
,???10,ORA27
 2020 DATA ASL27, ???10, CLC10, DRA36, ???10, ???10, ???10, ORA35, ASL35,
???10,JSR32
 2030 DATA AND29, ???10, ???10, BIT23, AND23, ROL23, ???10, PLP10, AND21,
ROL112,77710
 2040 DATA BIT32, AND32, ROL32, ???10, BMI24, AND210, ???10, ???10, ???10
,AND27,ROL27
 2050 DATA ???10,SEC10,AND36,???10,???10,???10,AND35,ROL35,???10,
RTI10,EOR29
 2060 DATA ???10,???10,???10,EOR23,LSR23,???10,PHA10,EOR21,LSR112
,???10,JMP32
 2070 DATA EOR32,LSR32,???10,BVC24,EDR210,???10,???10,???10,EDR27
,LSR27,???10
 2080 DATA CLII0, EOR36, ???10, ???10, EOR35, LSR35, ???10, RTS10,
ADC29,???10
2090 DATA ???10,???10,ADC23,ROR23,???10,PLA10,ADC21,ROR112,???10
,JMP311,ADC32
 2100 DATA ROR32, ???10, BVS24, ADC210, ???10, ???10, ???10, ADC27, ROR27
,???10,SEI10
 2110 DATA ADC36, ???10, ???10, ???10, ADC35, ROR35, ???10, ???10, STA29,
???10,???10
 2120 DATA STY23, STA23, STX23, ???10, DEY10, ???10, TXA10, ???10, STY32,
STA32,STX32
 2130 DATA ???10,BCC24,STA210,???10,???10,STY27,STA27,STX28,???10
,TYA10,STA36
 2140 DATA TXS10,???10,???10,STA35,???10,???10,LDY21,LDA29,LDX21,
???10,LDY23
 2150 DATA LDA23,LDX23,???10,TAY10,LDA21,TAX10,???10,LDY32,LDA32,
LDX32,77710
 2160 DATA BCS24,LDA210,???10,???10,LDY27,LDA27,LDX28,???10,CLV10
,LDA36,TSX10
 2170 DATA ???10,LDY35,LDA35,LDX36,???10,CPY21,CMP29,???10,???10,
CPY23,CMP23
 2180 DATA DEC23, ???10, INY10, CMP21, DEX10, ???10, CPY32, CMP32, DEC32,
???10,BNE24
 2190 DATA CMP210,???10,???10,???10,CMP27,DEC27,???10,CLD10,CMP36
,???10,???10
 2200 DATA ???10,CMP35,DEC35,???10,CPX21,SBC29,???10,???10,CPX23,
SBC23, INC23
 2210 DATA ???00, INX10, SBC21, NOP10, ???10, CPX32, SBC32, INC32, ???10,
BEQ24,SBC210
 2220 DATA ???10,???10,???10,SBC27,INC27,???10,SED10,SBC36,???10,
???10,???10
 2230 DATA SBC35, INC35, ???10
```

Important: assurez-vous que les quatre premières lignes (0-3) ont bien la longueur prévue (38 caractères). Le sous-programme d'affichage des messages doit pouvoir les trouver à une adresse précise en mémoire.

Le programme commence à la ligne 5 par un saut au sous-programme d'initialisation qui fait deux choses:

- implantation à partir de l'adresse #BFEØ du code machine contenu dans C\$ (ligne 1010).

Avant d'étudier cette routine, voyons comment sont stockés en mémoire les messages à afficher, représentés par les 4 premières lignes REM.

Vous savez que le programme Basic est rangé à partir de #500. Faites RUN, option 2, adresse #500.

Le contenu de chaque emplacement en mémoire est affiché sur deux lignes.

Sur la première on trouve la valeur hexadécimale (binaire), sur la seconde le caractère correspondant (s'il existe).

Pouvez-vous déterminer l'adresse du premier caractère de chaque message?

Cherchez les chiffres 1 précédant le mot menu, la flèche droite et le tiret. Ne vous occupez pas du reste.

Si l'hexadécimal vous gêne encore, choisissez l'option 3.modif et utilisez la flèche droite. L'adresse pointée par le curseur est affichée en haut à gauche. Vous appuierez sur \fin pour quitter le mode modif.

Notez les adresses: #507, #52E, #555 et #57C.

Revenons à la routine d'affichage.

Retournez au menu. Option 1.Désassemblage, Adresse #BFEØ

#BFEØ-	20 C5 D8	JSR #D8C5	GETBYTC, #D8ØA sur ORIC-1
#BFE3-	8A	TXA	transfère X dans l'accumulateur
#BFE4-	A4 9B	LDY #9B	;octet fort d'adresse dans Y
#BFE6-	A2 03	LDX >#03	;déplacement par rapport à #BB80
#BFE8-	20 65 F8	JSR #F865	;STOUT, #F82F sur ORIC-1
#BFEB-	60	RTS	retour Basic

C'est la routine STOUT (ATMOS #F865, ORIC-1 #F82F) qui affiche la chaine de caractères en haut de l'écran.

Avant de l'appeler, il faut charger les registres Y et A avec l'adresse de la chaîne, octet fort dans Y, octet faible dans A . En langage machine, il est courant de fournir des paramètres à une sous-routine par l'intermédiaire des registres.

STOUT a d'autres exigeances. Il faut que la chaîne ait moins de 38 caractères et qu'elle se termine par 00. C'est le cas, nos messages comportent 32 caractères et sont suivis du zéro qui marque, toujours, la fin d'une ligne Basic. Comme ils sont tous dans la même page de mémoire (normalement page 5), on peut charger le registre Y avec le contenu de #9B, octet fort du pointeur TXTTAB en #9A,9B. Ce pointeur indique le début du programme. Il contient l'adresse #501 depuis l'initialisation.

Il nous reste à charger l'accumulateur avec l'octet de poids faible

soit #07, #2E, #55 ou #7C. Cette valeur est affectée à la variable Basic J.

Avez-vous remarqué la forme inhabituelle de l'instruction CALL , CALL#BFE0.J à la ligne 260?

C'est un moyen de passer une valeur 8 bits à une routine en langage machine. L'instruction JSR #D8C5 (#D80A sur ORIC-1) permet de la recueillir.

GETBYTC est une sous-routine en ROM qui charge le registre X avec la valeur 8 bits se trouvant après la virgule.

Il nous suffit de transférer le contenu de X dans A avec TXA.

Le registre X passe à une autre tâche. STOUT l'utilise comme index pour remplir successivement les emplacements mémoire à partir de #BB80 qui est l'adresse du coin supérieur gauche de l'écran.

Charger X avec #03 a pour effet de faire commencer l'affichage à la quatrième position de la ligne supérieure.

Pour le vérifier, revenez au menu. Stop, puis entrez en commande directe:

POKE#BB83,65

L'instruction RTS assure le retour à l'instruction Basic suivante.

- construction d'un tableau de chaînes de caractères de 256 éléments à partir des données en DATA (ligne 1020).

Ce tableau sert à la fonction désassemblage.

L'indice de chaque élément correspond à un code opératoire du 6502 Le code opératoire est toujours le premier octet d'une instruction machine. Il détermine la nature de l'opération à effectuer, le nombre d'octets de l'instruction complète et le mode d'adressage.

Les modes d'adressage sont les différentes manières employées par le microprocesseur pour accéder à un emplacement en mémoire. Nous les étudierons dans la deuxième partie.

Chaque élément du tableau contient ces renseignements, nécessaires au désassemblage.

Les trois lettres représentent le mnémonique: nature de l'opération. Le chiffre qui suit 1, 2 ou 3 est le nombre d'octets.

Un nombre de 1 ou 2 chiffres (de Ø à 12) indique le mode d'adressage.

Faites RUN, 3.Stop puis entrez successivement:

?I\$(6) ?I\$(96) ?I\$(#69) ?I\$(#A5) ?I\$(#AD)

Après l'initialisation, un menu permet le choix entre les deux fonctions et branche le programme en conséquence.

La fonction desassemblage est assurée par les lignes 300-390.

La ligne 330 dissèque l'élément du tableau dont l'indice correspond à C, code opératoire de l'instruction machine. M\$ est le mnémonique, N le nombre d'octets et E le mode d'adressage.

La ligne 340 affiche les N codes binaires-hexadécimaux sans le symbole # mais pourvus des zéros non significatifs (GOSUB 30).

La ligne 350 affiche le mnémonique.

La ligne 360 appelle le sous-programme spécifique du mode d'adressage pour l'affichage de l'opérande, sauf si E=0, adressage implicite ou code opératoire inconnu.

La ligne 370 détecte le bas de l'écran (test de la variable système en #268) et appeile la sous-routine de la ligne 260 pour l'affichage du premier message dont l'octet faible d'adresse est J=#07.

L'affichage en hexadécimal du contenu d'une zone en mémoire et éventuellement des caractères correspondants est réalisé par les lignes 400-470. La commande PLOT est utilisée. Là-encore les nombres hexadécimaux sont mis au format.

Après affichage d'une page de 130 octets, le choix de l'option 3.modif envoie à la ligne 600.

Ligne 600 M = A-130, adresse du premier octet de la page affichée. Si M>#BFFF retour en 470. On ne peut pas modifier la ROM.

Ligne 610 Z servira à compter les chiffres hexadécimaux affichés. X et Y sont les coordonnées pour PLOT. Un sommaire des commandes est présenté en haut de l'écran par GOSUB 260 (avec J=#55). GOSUB 540 fait apparaître le curseur ainsi que l'adresse qu'il désigne. POKE#20C,255 met en mode majuscules.

Ligne 620 CALL#F75A (#F729 sur ORIC-1) affiche ou efface CAPS selon le mode Maj. ou Min. en vigueur. Mise en attente de la frappe d'une touche détectée par la fonction KEYS et analysée par les lignes suivantes.

Ligne 630 Si flèche gauche GOSUB 560 efface curseur actuel, retour en 600.

Ligne 640 Si flèche droite GOSUB 520 déplace le curseur d'une position, incrémente adresse et l'affiche, retour en 620.

Ligne 650 Si CTRL-T mode Min ou Maj, n'est activé qu'en mode entrée caractères (L=1), retour en 620.

Ligne 660 Si '#' entrée code hexa, GOSUB 560, L=0 et GOSUB 540 font passer curseur sur ligne supérieure, mise en mode majuscules, retour en 620.

Ligne 670 Si '®' entrée caractères, 605UB 560, L=1 et 605UB 540 font passer curseur sur ligne inférieure, retour en 620.

Ligne 680 Si '\' fin modif, GOSUB 550 efface curseur et adresse, mise en mode majuscules, retour en 470.

Ligne 690 Si L=1 entrée caractères. Les caractères sont affichés à la ligne inférieure, GOSUB 510 code inscrit en M, déplacement curseur, incrémentation M, etc..., retour en 620.

Ligne 700 Test des codes pour chiffre hexa valide (0-F), si valide affichage

Ligne 710 Si deux chiffres hexa ne sont pas affichés retour en 620, si deux chiffres affichés GOSUB 500 code évalué, puis inscrit en M, déplacement curseur, incrémentation mémoire, etc..., retour en 620.

Points particuliers:

La routine système en #E5F5 sur ATMOS est utilisée par CLOAD et CSAVE pour débarrasser la ligne du haut de l'écran. La routine équivalente en #E563 sur ORIC-1 est légérement différente, elle ne nettoie pas tout mais ce n'est pas très gênant.

Notez l'emploi de l'instruction X=FRE("") pour éviter l'encombrement du haut de la mémoire vive par les chaines de caractères inutilisées. Dans la section 600-710 la mise en mode majuscules est faite directement par POKE#20C,255, la routine en #F75A (ou #F729) se contente de constater le fait (test de #20C) et affiche CAPS.

CHAPITRE 3

LE TEXTE BASIC

Notre intention est d'étudier le rangement d'un programme Basic en mémoire.

Il est intéressant de disposer d'un utilitaire écrit lui-aussi en Basic car ainsi, il pourra s'examiner lui-même.

Déroutant à première vue, le concept est simple et indépendant du langage.

Pour fonctionner, un programme a besoin d'instructions et de données. Quand une instruction est exécutée, toute l'information qu'elle manipule est traitée en tant que données, même si cette information fait partie intégrante et constitue un élément actif du programme en cours d'exécution.

Le pointeur en page zéro TXTTAB (#9A,9B), premier de toute une série que nous découvrirons bientôt, indique le début du texte Basic. Il contient normalement l'adresse #501. C'est à partir de cette adresse que sera logée la première ligne du programme. Celle dont le numéro est le plus faible, pas forcément la première entrée en mémoire.

STOCKAGE D'UNE LIGNE

Lancez le programme et choisissez l'option 2, adresse #500.

La première ligne commence en #501 et se termine en #527 par la valeur 00.

La seconde occupe les emplacements de #528 à #54E où l'on trouve aussi 00.

Première constatation, toutes les lignes se terminent par zéro. Cette valeur est un point de repère pour l'interpréteur. A partir de là, il pourra rechercher une éventuelle fin de programme, mettre à jour ses pointeurs pour continuer à se situer dans le texte, savoir où trouver la prochaine commande à exécuter. Les deux points ':' (#3A) qui séparent deux instructions dans une même ligne sont un autre point de repère.

J'ai commencé par la fin pour vous expliquer la nécessité du 00 à l'adresse #500. Tous les programmes Basic doivent être précédés d'un octet contenant la valeur 00. Elle est automatiquement inscrite en #500 au cours de l'initialisation.

Si une autre valeur est placée en #500, on fait les constatations suivantes:

- RUN (ou RUN Ø qui signifie la même chose) n'est pas exécuté.
- Par contre, LIST l'est, ainsi que RUN 5 par exemple. Le fonctionnement du programme est cependant perturbé. Certaines commandes doivent trouver 00 en #500 pour se positionner correctement, notamment READ pour les données à lire en DATA.

Notez une fois pour toutes que lorsque zéro est cité, il s'agit de la valeur binaire #00 et non pas du chiffre '0', le caractère dont le code ASCII est #30.

Revenons au contenu de la première lione:

28 05 00 00 9D 00

C'est sensiblement différent de ce que vous avez entré au clavier. L'interpréteur l'a analysé et recodé à sa façon.

28 05 00 00 9D . . 00

adresse ligne suivante numéro de ligne REM fin de ligne

Les quatre premiers octets servent d'en-tête et permetttent à l'interpréteur de se situer. Ils ont toujours la même signification:

Les deux premiers indiquent l'adresse de la ligne suivante. On les appelle octets de chainage (en anglais: link bytes).

Vous noterez qu'ils sont inversés, octet faible d'abord.

Lorsque ces deux octets sont à zéro, ils indiquent la fin du programme qui sera ainsi marquée par une série de trois 00 en comptant le 00 de fin de dernière ligne.

Les deux octets suivants (inversés eux-aussi) représentent le numéro de ligne. C'est un nombre entier, en valeur absolue (positif), limité à 63999 (#F9FF)

La valeur #9D est le code du mot-clé REM. Tous les mots-clés du Basic, commandes, opérateurs arithmétiques ou logiques, fonctions, sont stockés sous forme codée.

Les autres octets de la ligne n'ont rien de remarquable. Vous reconnaîtrez les codes ASCII des différents caractères du texte.

Pour examiner d'autres lignes du programme nous n'allons pas fouiller la mémoire, octet par octet.

Vous pensez bien que l'interpréteur dispose d'un moyen de retrouver une ligne particulière. Nous allons lui emprunter.

Retournez au menu, Stop, puis à la suite du programme, entrez les lignes ci-dessous:

Pour ATMOS

3000 C\$="20E2CA20B3C6A5CE8580A5CF8581A9808DF202206CC74CA2C9"

Pour ORIC-1

3000 C\$="2098CA20DEC6A5CE8580A5CF8581A9808DF2022099C74C70C9"

Suite, pour l'un ou l'autre

3010 A=#400: DDKE#2F5, A: GDSUB 1500: CLS

3020 PRINT"Entrez ! suivi du numéro de ligne":STOP

3030 A=DEEK(#80):Y=11:GOTO 410

Ce programme utilise ! (point d'exclamation) qui est une commande BASIC très intéressante.

Elle provoque ce qu'on appelle un saut indirect à l'adresse contenue dans #2F5.2F6.

C'est un moyen élégant d'appeler une sous-routine en langage machine. Si cette dernière est conçue pour les recueillir et les exploiter, il permet, en même temps, de passer des paramètres. En l'occurrence, ce sera le numéro de la ligne à rechercher.

- Faites RUN 3000
- Tapez ! suivi d'un n° de ligne du programme (RETURN)

Pour recommencer, retournez au menu, puis Stop. N'utilisez pas d'autre option.

Faites plusieurs essais avec des numéros de ligne différents.

Vous remarquerez la bonne volonté de l'interpréteur. Si vous lui proposez un numéro de ligne qui n'existe pas, il vous donne l'adresse de la ligne suivante. S'il n'y en a plus, c'est celle de la fin du programme.

Essayez !6 puis !30000

Nous allons examiner la sous-routine en langage machine sans entrer dans le détail. Notez en marge de votre livre d'y revenir après avoir lu la quatrième partie.

Retournez au menu, Stop puis RUN pour reconstituer le tableau des codes 6502. (GOTO 3000 au lieu de RUN, aurait pu nous éviter cette réinitialisation.)

Option 1, adresse #400:

ATMOS ORIC-1

#400-	20	E2	CA	JSR	#CAE2	#CA98	;LINGET n°ligne dans LINNUM (#33,34)
#403-	20	B 3	CP	JSR	#C9B3	#C6DE	:FNDLIN recherche ligne
#406-	A5	CE		LDA	#CE		; adresse dans LOWTR (#CE,CF)
#408-	85	80		STA	#80		;transférée
#4ØQ-	A5	CF		LDA	#CF		; dans
#4ØC-	85	81		STA	#81		;#80,81
#40E-	A9	80		LDA	>#80		;bit 7 de #2F2 (drapeau)
#410-	80	F2	02	STA	#2F2		;mis à 1 pour retour de LIST
#413-	20	60	€7	JSR	#C76C	#C799	;LIST ligne dont n° est en #33,34.
#416-	4C	A2	C9	JMP	#C9A2	#C97Ø	:CONT reprise programme après STOP.

NUMEROS DE LIGNE SUPERIEURS A 63999

Vous savez que les numéros supérieurs à 63999 ne sont pas acceptés à l'entrée. C'est la routine LINGET (ATMOS #CAE2, ORIC-1 #CA98) qui en est cause.

Cette routine recueille les caractères décimaux dans le buffer d'entrée et les transforme en une valeur binaire sur deux octets. Avant d'inscrire ce nombre dans LINNUM (#33,34), elle teste l'octet de poids fort. S'il est égal ou supérieur à #FA, le message ?SYNTAX ERROR est affiché.

On peut court-circuiter LINGET en modifiant directement le numéro en mémoire.

Entrez la ligne:

63999 CLS:PRINT"CECI EST LA DERNIERE LIGNE"

23

RUN 3000 !63999

Choisissez l'option 3.modif

Repérez le curseur dans le fouillis et amenez le avec la flèche droite en face de l'octet fort du numéro de ligne (le troisième, qui doit être FF).

Confirmez FF puis remplacez l'octet suivant F9 par FF également.

fin modif, retour menu, Stop puis LIST 60000-

Vous ferez les constatations suivantes:

- La ligne 65535 ne peut pas être éditée et ne peut pas être désignée directement dans LIST, à cause de l'intervention de LINGET dans ces commandes.
- Par contre, RUN 65535 est exécuté.
- La ligne ne peut pas non plus être supprimée normalement.

Nous allons le faire à l'aide de notre programme en lui redonnant son numéro initial.

RUN 3000 !63999

(!65535 serait refusé car notre programme utilise LINGET)

Procédez à l'opération inverse, redonnez la valeur F9 à l'octet de poids fort.

Puis \fin, retour menu, Stop et LIST 60000-

Supprimez la ligne 63999.

On ne peut pas, sans risque, modifier de cette façon un numéro de ligne à l'intérieur du programme, sauf si la modification n'implique pas de déplacement de la ligne.

L'intervention de LINGET (et des autres routines d'analyse) est nécessaire pour l'insertion, le remplacement ou la suppression d'une lione

CODAGE DES MOTS-CLES

Au cours de vos manipulations, vous aurez découvert le code de plusieurs mots-clés.

Pour coder les mots-clés en vue du stockage de la ligne, ou pour les décoder en vue du listage, l'interpréteur se sert d'une table en #CØEA.

Dans les deux sens, il utilise la correspondance:

Code du mot-clé - #80 = position dans la table.

Vous mesurez ici l'activité intense que doit déployer l'interpréteur, surtout au moment du codage.

Chaque mot doit être analysé, caractère par caractère, pour savoir s'il s'agit d'un mot-clé.

Cela explique pourquoi certains noms de variable sont à éviter. L'ordinateur n'aime pas le pouLET, il déteste le ciTRON et la famille en général, FREre, TANte, couSIN. Pour afficher la table des mots-clès. à partir du menu, option 2, adresse: #CØEA.

Les mots sont de longueur inégale. Afin de les délimiter, le code ASCII de la dernière lettre a le bit 7 à un (code ASCII+128). On parle quelquefois d'ASCII négatif. Cette dernière lettre est affichée en inverse par la commande PLOT.

Rappelez-vous: code du mot-clé = position dans la table + #80. Ce petit programme Basic vous évitera de compter:

4000 PRINT:PRINT:K=#80:T=#C0EA 4010 PRINT,HEX\$(K);"-"; 4020 C=PEEK(T):PRINT CHR\$(C);:T=T+1 4030 IF C<128 THEN 4020 4040 IF KEY\$<>""THEN GET A\$ 4050 K=K+1:IF K=#F7 THEN 4000 ELSE PRINT:GOTO 4010

RUN 4000

Le programme boucle sur lui-même.

Pour obtenir une pause ou pour continuer, appuyez sur une touche quelconque.

Pour l'arrêter, CTRL-C.

Sur ORIC-1, les mots-clés INVERSE et NORMAL, qui figurent dans la table, ne sont pas actifs. Les auteurs ont sans doute eu, un moment, l'idée de les mettre en oeuvre. Ils ont été remplacés par STORE et RECALL sur ATMOS.

Après le dernier mot MID\$, toujours sur ORIC-1, on trouve GO (même remarque que précédemment), supprimé sur ATMOS, puis les messages d'erreur. Si cela vous amuse de les afficher, remplacez #F7 par #10A à la ligne 4050.

L'interpréteur retrouve les messages d'erreur par leur adresse. D'autres messages, ailleurs en mémoire, au lieu d'avoir le dernier caractère en ASCII négatif, sont suivis d'un 00. En ce qui concerne ceux-ci, les codes n'ont plus de signification. Remarquez néammoins que la commande LIST, jusqu'à #FF, continue de les interpréter comme des mots-clés, comme le montre l'expérience suivante.

Entrez la ligne:

6 PRINT

Faites RUN 3000 puis !6 pour retrouver son adresse.

Option 3. modif. Essayez de retrouver le curseur et amenez-le en face de BA, code de PRINT. Ce devrait être en #5AC. Inscrivez FA à la place.

\fin modif. Option 1.menu puis 3.Stop et maintenant LIST.

Le message d'erreur est affiché à la ligne 6. N'oubliez-pas de la supprimer.

INSERTION OU SUPPRESSION D'UNE LIGNE

L'exercice suivant a deux buts: vous entrainer à utiliser le programme et vous faire progresser dans la connaissance de l'interpréteur.

Faites RUN, Option 2, adresse #400

130 octets sont à votre disposition. Peu importe ce que vous y voyez. Considérez l'écran comme une page blanche.

Vous allez entrer le programme en langage machine suivant:

adresse	hexa	mnémonique	
#400-	A2 00	LDX >#ØØ	; X=Ø
#402-	BD 14 Ø4	LDA #414,X	charge A avec contenu de #414+X
#405-	95 35	STA #35,X	range A dans #35+X, buffer d'entrés
#407-	FØ Ø4	BEQ #40D	;si A=0 fin de ligne,sortie boucle
#409-	E8	INX	; X=X+1, caractère suivant
#40A-	4C Ø2 Ø4	JMP #402	:GOTO #402
#4ØD-	A2 34	LDX >#34	;adresse BUF-1, #0034
#4ØF-	AØ ØØ	LDY >#00	dans Y.X pour entrée dans
#411-	4C BD C4	JMP #C4BD *	;CMDLP (ATMOS) analyse,traitement

Pour ORIC-1: remplacer le code ci-dessus par le suivant: #411- 4C CD C4 JMP #C4CD ; CMDLP (ORIC-1) analyse, traitement

mode entrée caractères pour la ligne qui suit, terminée par 00 hexa. #414- 6 PRINT"Bonjour les amis"00

Choisissez l'option 3.modif et entrez le code hexadécimal sans vous préoccuper du reste pour le moment.

Arrivé à l'adresse #414, passez en mode @ entrée caractères et tapez le texte, exactement comme vous le feriez pour composer une ligne de programme.

Attention cependant, à la fin n'appuyez pas sur RETURN mais plutôt repassez en # hexadécimal et entrez 00 comme indiqué ci-dessus.

Lorsque c'est fait, appuyez sur \fin, puis option 2.désassemblage pour contrôler.

Vous reconnaîtrez le mnémonique mais pas la ligne Basic. Elle n'a aucun sens pour le microprocesseur. Vérifiez quand même la présence du 00.

Maintenant: 3.Stop, CALL#400 puis LIST.

La ligne n°6 a été insérée. Laissez-la.

RUN, option 2, adresse #400 à nouveau. 3.modif. Amenez le curseur en #415.

Restez en mode hexadécimal et tapez 00 à la place de 20 (code du caractère espace) qui devrait se trouver là, ou, si vous n'avez pas mis d'espace, à la place de 50 (P de PRINT).

\fin, menu, Stop, CALL#400, LIST.

La ligne 6 a été supprimée.

Maintenant un peu de technique.

Nous avons reproduit à peu près ce qui se passe lorsque vous entrez une ligne de programme au clavier.

Cette ligne est copiée, caractère après caractère, dans le buffer d'entrée en #35. Petite différence: c'est le fait d'appuyer sur RETURN qui inscrit automatiquement 00 à la fin de la ligne. Ici, nous avons inclus cette valeur dans le texte à recopier par le programme.

Comme dans la réalité, Y et X sont chargés avec l'adresse #34 et la rentrée dans la boucle principale d'entrée des commandes (CMDLP Command loop) se fait en #C48D (ORIC-1 C4CD) comme au retour d'une entrée au clavier.

Beaucoup de choses se passent ensuite.

Le pointeur de texte TXTPTR (#E9,EA) reçoit le contenu de X et Y et bondit dans le buffer d'entrée sur l'impulsion de CHARGET pour l'analyse et le codage de la ligne.

Si la ligne existe déjà, l'ancienne ligne est écrasée par un déplacement de tout le programme vers le bas et la nouvelle est insérée grâce à un nouveau déplacement du programme, vers le haut cette fois.

Si elle n'existait pas, il y a seulement insertion.

Evidemment, tout cela s'accompagne d'une mise à jour des octets de chainage et des pointeurs.

Faites d'autres essais en modifiant la ligne à l'adresse #414. N'oubliez pas la valeur 00 à la fin et faites attention aux lignes existantes.

Vous pouvez inscrire une commande directe, par exemple: LIST 100-200 (00).

Avant de clore ce chapitre, je voudrais vous poser une petite énigme. Plus tard, je risque d'oublier. Vous n'y répondrez sans doute pas aujourd'hui, peu importe. Vous n'avez qu'à ajouter une croix en marge.

Stoppez le programme et entrez en commande directe:

POKE#500,58 DOKE#501,#A5

maintenant, faites LIST, puis RUN.

Un tuyau: 58 décimal est le code ASCII des deux points ':' et il faut étudier le phénomène en relation avec la routine importante NEWSTT en #C8C1 pour ATMOS, #C8AD pour ORIC-1.

Pour remettre le programme en état:

POKE#500,0 DOKE#501,#528

27

Si vous décidez de vous arrêter pour aujourd'hui, sauvegardez le programme avec ses appendices. Nous en aurons encore besoin.



CHAPITRE 4

LES VARIABLES

DEFINITION

Une variable est une expression alphanumérique servant à désigner une donnée susceptible de prendre différentes valeurs. En langage informatique, à ce terme est associée la définition d'une zone en mémoire d'un format (nombre d'octets) déterminé. L'ordinateur y trouvera les renseignements actualisés qui lui permettront d'évaluer la donnée à traiter.

Si vous entrez en commande directe la déclaration: A=10, vous procédez à l'affectation de la valeur décimale 10 à la variable dénommée A. L'ordinateur réserve aussitôt le nombre d'octets nécessaire et y range le nom et la valeur de la variable ainsi définie. Si cette déclaration est rencontrée au cours de l'exécution d'un programme, la procédure est la même.

Si vous déclarez ensuite: A=A+10, cette affirmation incongrue est parfaitement comprise comme étant l'ordre d'ajouter 10 à la valeur précédemment stockée.

Vous noterez que le signe = , qui indique normalement l'égalité, revêt ici une signification différente.

La déclaration: A=10 sous-entend: LET A=10.

Le mot LET qui veut dire SOIT est facultatif en Basic Microsoft mais les puristes ne manqueront pas de l'utiliser.

Un nom de variable peut comporter un ou plusieurs caractères, lettres ou chiffres exclusivement, dont le premier (ou unique) est obligatoirement une lettre. Seuls les deux premiers caractères sont significatifs.

Le Basic de l'Oric distingue 7 types de variables répartis en deux groupes.

Les quatre premiers types forment le groupe des variables simples: variables numériques réelles ou entières, variables alphanumériques (chaînes de caractères), fonctions définies par l'utilisateur.

Le deuxième groupe est celui des variables indicées ou tableaux: tableaux de variables numériques réelles ou entières, tableaux de variables alphanumériques.

Les variables qui composent ces tableaux ont les mêmes caractéristiques que les variables simples correspondantes.

Lors de l'affectation d'une valeur, le nom des variables entières est suivi du symbole %, celui des chaînes de caractères par \$. Les variables réelles n'ont pas de suffixe.

L'interpréteur se sert de ce symbole pour identifier les variables dans le programme mais ce n'est pas ainsi qu'il les stocke et les retrouve dans la table.

Lors de l'analyse de la ligne, le nom est rangé sur deux octets selon une combinaison ASCII positif ou négatif appropriée. Si le nom ne comporte qu'une seule lettre, le deuxième caractère est remplacé par un caractère nul, #00 ou #80 (128) suivant le cas.

Le pointeur VARTAB (#9C,9D), deuxième pointeur fondamental, marque la fin du texte Basic.

L'adresse qu'il contient correspond au début de la zone des variables dont la première partie est occupée par les variables simples.

?HEX\$(DEEK(#9C)) notez l'adresse

RUN 3000 puis ! suivi du numéro de la dernière ligne du programme.

Vous devriez repérer sans difficultés les trois zéros de la fin du programme. Après cette série de zéros, (la première de toutes) on trouve, à l'adresse que vous avez notée, la valeur #43. Il s'agit du code ASCII de "C", première lettre de la variable alphanumérique C\$ qui contient le code machine à la ligne 3000.

Si vous aviez lancé le programme principal puis entré GOTO 3000 au lieu de RUN, il y aurait eu là une autre variable. Vous savez que RUN est équivalent à CLEAR suivi de GOTO.

LES VARIABLES SIMPLES

Elles sont stockées sur 7 octets dont les deux premiers représentent le nom.

Pour cette étude, nous définirons nos propres variables pourvues de noms et de valeurs caractérisques.

Entrez les lignes suivantes:

5000 DEF FN F(R)=R^2*PI

5010 AB=123.456:CD%=-17:EF\$="Salut":GH\$="les copains":IJ\$=EF\$+" "+GH\$

5020 CLS:PDKE#2F2,128:LIST 5000-5015:PRINT:PRINT

5025 GOTO 5030

5030 A=DEEK(#9C):PRINT"VARTAB=";HEX\$(A):Y=17:GOTO 410

L'instruction POKE#2F2,128 met à un le bit 7 de la variable système #2F2 et permet l'utilisation de LIST sans interruption du programme Basic.

Nous parlerons plus longuement de ce drapeau dans la troisième partie.

RUN 5000

La commande RUN ayant eu pour effet d'annuler les valeurs précédemment déclarées, toutes nos variables sont rangées à partir de VARTAB, dans l'ordre suivant lequel elles ont été définies (lignes 5000-5010)

Sachant qu'elles occupent 7 octets, vous devriez les retrouver à l'écran.

F(R) d'abord et sa variable R dont nous nous occuperons plus tard, puis AB et la suite que nous étudierons maintenant.

Avec l'aide de l'interpréteur nous allons les sélectionner une par une.

Retournez au menu, Stop, modifiez la ligne 5025 comme suit:

5025 GOTO 5040

Puis ajoutez les lignes suivantes:

Pour ATMOS

5040 DOKE#400,#8820:DOKE#402,#4CD1:DOKE#404,#C9A2:DOKE#2F5,#400

Pour ORIC-1

5040 DOKE#400, #FC20: DOKE#402, #4CD0: DOKE#404, #C970: DOKE#2F5, #400

Suite commune

5050 PRINT"Entrez ! suivi nom variable complet": END

5060 PLOT 10,13,CHR*(PEEK(#B4)):PLOT 14,13,CHR*(PEEK(#B5))

5070 DOKE#80, DEEK (#86) -2: A=DEEK (#80): PRINT: PRINT HEX\$ (A),: FOR I=0 TO 6

5080 C(I)=PEEK(A+I): H\$=HEX\$(C(I)): GOSUB 30: PRINT MID\$(H\$,2); " (4ω/μαω) "::NEXT:PRINT

5090 IF C(0)<128 AND C(1)>127 THEN A=DEEK(A+3):Y=21:GOTO 410 ELSE END

La ligne 5040 inscrit en #400 une routine en langage machine de deux instructions, qui sera appelée par la commande !

ATMOS ORIC-1

#400-20 88 D1 JSR #D188 #DØFC ;appel de PTRGET #402- 4C A2 C9 JMP #C9A2 #C970 ; CONT programme Basic

La sous-routine PTRGET est utilisée par l'interpréteur pour retrouver dans la table une variable désignée par son nom assorti éventuellement du suffixe.

Si la variable n'existe pas, 7 octets sont réservés dans la table, deux octets pour le nom cité suivis de cinq zéros.

Lorsque l'interpréteur est branché sur CONT, il se repositionne dans le programme Basic pour en reprendre l'exécution interrompue par END ou STOP.

(Remplacez éventuellement END par STOP à la ligne 5050 et constatez la différence)

La ligne 5060 affiche les contenus de #B4 et #B5. Ce "pointeur" en page zéro s'appelle VARNAM. Il contient le nom de la dernière variable traitée par l'interpréteur.

Le pointeur VARPNT en #86,87 (ligne 5070) contient l'adresse de la valeur de cette variable. DEEK(#B6)-2 est donc l'adresse du début des 7 octets qui nous

intéressent. Remarquez la nécessité de sauvegarder cette adresse (en #80.81) car le

fait de déclarer ensuite les variables A et I, modifie évidemment VARNAM et VARPNT.

Les 7 octets sont affichés suivant un format hexadécimal convenable (lignes 5070-5080) puis le nom est testé à la ligne 5090. S'il s'agit d'une chaîne de caractères, sa position en mémoire est localisée et elle est affichée grâce à un saut au programme utilitaire, sinon le programme s'arrête.

Passons à la pratique, vous comprendrez mieux.

VARIABLE REELLE

RUN 5000 puis !AB, sélection de la variable réelle AB.

Précédés de leur adresse, les 7 octets qui décrivent la variable sont affichés.

Ils correspondent à la contexture ci-dessous:

AB = 123.456

octet n°	signification	hexadécimal
Ø	ler caractère du nom - ASCII positif	41
1	2ème caractère - ASCII positif	42
2	valeur en virgule flottante - exposant	87
3	" - mantisse	76
4	" בער	E9
5	11 4	78
6	" octets	D5

Nous étudierons la représentation des nombres en virgule flottante dans la troisième partie de ce livre.

Sachez qu'elle permet, sur 5 octets, de désigner des nombres réels, entiers ou fractionnaires, compris entre 10^-38 et 10^38, avec une précision de 9 chiffres.

VARIABLE ENTIERE

Voyons comment est stockée la variable entière CD%

RUN 5000 puis !CD%

CD% = -17

octet	n°	signification				hexadécimal
	Ø	ler caractère du nom				C3
	1	2ème caractère	-	ASCII	négatif	C4
	2	Valeur entière signée	_	octet	FORT	FF
	3				FAIBLE	EF
	4					00
	5	inutilisés				00
	6					00

Les variables entières ont le nom inscrit en ASCII négatif.

Leur emploi est rarement préconisé, du moins en ce qui concerne les variables simples.

L'interpréteur doit les transformer en virgule flottante avant utilisation, ce qui prend du temps.

En outre, elles occupent un octet de plus (pour le signe %) dans le texte Basic.

Elles prennent la même place en mémoire que les variables simples réelles (sept octets dont trois inutilisés et remplis avec des zéros). Nous verrons que ce n'est pas le cas pour les variables indicées.

REMARQUE IMPORTANTE: contrairement à l'habitude, les deux octets ne sont pas inversés. #FFEF vaut -17 décimal en complément à deux.

:

VARIABLE CHAINE DE CARACTERES

Passons à la chaîne de caractères EF\$

RUN 5000 !EF\$

EF\$="Salut"

octet	n °	signification	hexadécimal
	Ø	1er caractère du nom - ASCII positif	45
	1	2ème caractère - ASCII négatif	C6
	2	longueur de la chaîne	Ø5
	3	adresse de la chaîne octet faible	B7 ?
	4	" octet fort	16 ?
	5	inutilisés	00
	6	11	00

La variable chaîne de caractères elle-même n'est pas rangée dans la table.

Les trois octets qui suivent le nom (ASCII positif/négatif) jouent le rôle de descripteur, longueur et adresse (sur deux octets). Les deux derniers octets sont inutilisés et contiennent 00.

Dans le cas présent, la chaîne est rangée dans le texte Basic. Son adresse est interne. Vous ferez la même constatation pour GH\$="les copains"

Par contre lorsque l'affectation d'une valeur à une variable chaîne implique une concaténation, la nouvelle chaîne est stockée en haut de mémoire, en descendant à partir de HIMEM.

Retournez au menu, Stop, puis:

RUN 5000 !IJ\$

IJ\$ = EF\$ + " " + GH\$ concaténation

octet	n°	signification	hexadécimal
	Ø	1er caractère du nom - ASCII positif	49
	1	2ème caractère - ASCII négatif	CA
	2	longueur de la chaîne	11
	3	adresse de la chaîne octet faible	E9 ?
	4	" octet fort	9E ?
	5	inutilisés	02
	6	ıı	00

En regardant de plus près, on s'aperçoit que l'opération s'est effectuée en deux temps. EF\$+" " a été inscrit en haut de mémoire, puis à la suite, en descendant, la nouvelle chaîne complète.

Une chaîne de caractères entrée au cours de l'exécution du programme sur un INPUT ou même le caractère unique obtenu par GET sont aussi rangés en haut de mémoire.

Faites l'expérience en insérant successivement une ligne 5015

5015 INPUT"Votre prenom"; N\$

Par la même occasion, remarquez que le deuxième caractère du nom, manquant, est remplacé par un caractère nul, ici négatif soit #80.

Le stockage des chaînes en haut de mémoire présente une particularité qui peut se révéler génante.

Les chaînes périmées restent en haut de mémoire.

La nouvelle "valeur" qui leur est attribuée ne vient pas les écraser mais vient se placer à la suite aux adresses plus faibles, réduisant l'espace libre.

Il n'y guère moyen de faire autrement car les chaines de caractères sont de longueur inégale.

Si la mémoire disponible est mesurée, il y a risque d'encombrement et l'ordinateur doit s'arrêter pour procéder à un 'nettoyage' de la mémoire.

Les chaînes en vigueur sont déplacées vers le haut.

La routine utilisée porte un nom évocateur, GARBAGE en #D650 (ORIC-1 #D595).

Un nettoyage préventif plus rapide et qui passe pratiquement inaperçu peut être fait grâce à une instruction Basic de la forme X=FRE(""). Elle figure dans le programme à la ligne 8.

Vous pouvez la supprimer et faire l'expérience suivante:

Conservez la ligne 5015 INPUT"Votre prenom"; N\$

Lancez le programme plusieurs fois de suite, après retour menu, Stop, en utilisant GOTO 5000 au lieu de RUN.

Inscrivez votre prénom puis entrez !N\$ à chaque fois.

Votre prénom s'inscrit à des adresses de plus en plus basses en mémoire. Replacez l'instruction X=FRE("") à la fin de la ligne 8, puis recommencez la même opération.

Le pointeur FRETOP en #A2,A3 indique l'adresse la plus basse occupée par les chaînes en haut de mémoire, il marque la fin de l'espace libre.

FONCTION DEFINIE PAR L'UTILISATEUR

Supprimez la ligne 5010, remplacez la ligne 5015 par la suivante:

5015 KL=FN F(10)

RUN 5000 puis !KL

La variable KL a les caractéristiques d'une variable réelle. Le nombre en virgule flottante affiché représente la valeur de la fonction définie à la ligne 5000 pour R=10. Vérifiez avec ?KL Examinons les éléments qui permettent à l'ordinateur d'effectuer ce calcul.

Supprimez la ligne 5025, RUN 5000

A partir de VARTAB est rangé le pointeur de la fonction:

octet r	n°	signification	hexadécimal
(Ø	1er caractère du nom - ASCII négatif	C6
	1	2ème caractère - ASCII positif	ØØ
2	2	adresse de la définition octet faible	F1 ?
;	3	" octet fort	15 ?
	4	adresse de la variable R octet faible	50 ?
	5	" octet fort	19 ?
	6	caractère suivant le signe =. (ASCII pos	52

La définition est l'expression R^2*PI de la ligne 5000. Vous pouvez vérifier en retournant au menu; option 2 et en entrant son adresse (probablement différente de celle indiquée ci-dessus)

La présence dans l'octet n°6 du caractère suivant le signe = de la définition (ici R) n'a pas d'utilité connue.

Il y a des remarques à faire au sujet de la variable R.

- c'est obligatoirement une variable simple réelle.
- dans le cas présent, elle n'avait pas été définie et pourtant les octets n° 4 et 5 indiquent son adresse.

C'est que l'interpréteur a réservé 7 octets pour cette variable à la suite du pointeur de fonction.

Sa valeur en virgule flottante est pour la moment zéro.

Si une valeur lui avait été affectée avant la définition de la fonction, elle aurait été rangée avant cette dernière dans la table et l'adresse aurait été différente.

Retournez au menu, Stop, insérez la ligne: 5010 R=20 et modifiez la ligne 5015 5015 KL=FN F(R)

RUN 5000

La valeur de R a été inscrite à l'emplacement prévu. Elle a été utilisée pour le calcul de KL=FN F(R). Vérifiez en retournant au menu, Stop, puis ?KL

LES TABLEAUX DE VARIABLES

Un tableau est un ensemble de variables portant le même nom et affectées d'un indice désignant leur position relative dans le tableau.

Il comporte un en-tête, servant à l'identifier, suivi des valeurs de ses éléments, rangées par indice croissant.

Suivant la nature des variables indicées, ces valeurs occupent 5 octets (valeur en virgule flottante des variables réelles), ou 2 octets (valeur signée des variables entières) ou 3 octets (descripteur de chaîne de caractères).

Le format n'est donc pas unique, seuls les octets "utiles" que nous avons appris à reconnaître avec les variables simples, sont rangés en mémoire.

Les tableaux de variables sont stockés après les variables simples dans une zone qui leur est réservée. L'adresse du début, détenue par le pointeur ARYTAB en #9E,9F, est essentiellement "mouvante" en fonction du volume des variables simples définies, contrairement à celle pointée par VARTAB qui reste stable tant que le texte Basic lui-même n'est pas modifié. La fin de la zone des tableaux, début de l'espace libre, est indiquée

par le pointeur STREND (Storage end) en #AØ,A1.

La contexture générale de l'en-tête d'un tableau est la suivante: - les deux premiers octets contiennent les caractères du nom en ASCII positif ou négatif.

- les deux octets suivants représentent un déplacement (octet faible d'abord) à ajouter à l'adresse de début du tableau pour obtenir l'adresse du tableau suivant.
- le cinquième octet spécifie le nombre de dimensions.
- si 'n' est le nombre de dimensions, on trouve ensuite 'n' paires d'octets indiquant le nombre d'éléments dans chaque dimension. Ces paires d'octets commencent à la dernière dimension pour finir à la première, l'octet fort est stocké avant l'octet faible.

Après l'en-tête, les valeurs sur 5, 2, ou 3 octets sont rangées dans l'ordre des indices.

Le programme ci-dessous vous montre quelques exemples. Comme il utilise par ailleurs des variables simples, vous remarquerez l'acrobatie à laquelle il a fallu recourir pour retrouver ARYTAB, début d'un en-tête (ligne 6040).

6000 DIM AB%(5,20,75):GOSUB 6060:GOTO 6040

6010 A(0)=123.45:GOSUB 6060:GOTO 6040

6020 GOSUB 1020: GOSUB 6060: Z=7: GOTO 6040

6030 INPUT"valeur pour B%(0)"; B%(0):GOSUB 6060

6040 A=DEEK(#9E)+14-Z:PRINT HEX\$(A):FOR I=0 TO 11

6050 H\$=HEX\$(PEEK(A+I)):GOSUB 30:PRINT MID\$(H\$,2);" ";:NEXT:PRINT:END 6060 CLS:PLOT 2,2,CHR\$(PEEK(#B4)):PLOT 5,2,CHR\$(PEEK(#B5)):RETURN

Vous ferez successivement RUN 6000, RUN 6010, RUN 6020, RUN 6030.

RUN 6020 affiche l'en-tête du tableau alphanumérique I\$ des codes

On aperçoit le début des pointeurs de chaînes qui indiquent l'adresse des éléments en DATA (lignes 2000-2230).

L'en-tête du tableau à 3 dimensions AB%, obtenu par RUN 6000, est commenté ci-dessous:

DIM AB%(5,20,75)

octet n	•	signification		hexadécimal
Ø	ı	1er caractère du nom	ASCII négatif	Ci
1		2ème caractère	ASCII négatif	C2
2	!	déplacement pour	octet faible	DB
3		tableau suivant	octet fort	4A
		nombre de dimensions		03
5	i	N. d'éléments dernière dim	octet fort	00
6		н и	octet faible	4C
7		N.d'éléments deuxième dim	octet fort	00
8	1	н н	octet faible	15
9		N.d'éléments première dim	octet fort	00
10	ı	0 0	octet faible	06

Vous aurez note le retard à l'affichage de l'en-tête dû à l'importance du tableau. L'ordinateur doit inscrire la valeur 00 dans plus de 1900 emplacements en mémoire, exactement 6*21*76*2 = 19152 Le déplacement #4ADB=19163 correspond à ce nombre d'octets augmenté des 11 octets de l'en-tête.

Remarque:

Lorsque vous dimensionnez un tableau avec une instruction de la forme DIM A(n) cela signifie que l'indice du dernier élément sera n. Votre tableau aura ainsi 'n+1' éléments en comptant le premier d'indice Ø.

RUN 6010 ou 6030 vous montrent qu'à défaut d'une déclaration de dimension, le tableau est construit à une dimension, de 11 éléments.

A la ligne 6030, vous pouvez remplacer B%(0) par A\$(0), par exemple, et entrer une chaîne de caractères. Le pointeur de chaîne vous indiquera que ce premier élément est stocké en haut de mémoire.

N'oubliez pas de sauvegarder le programme et ses annexes avant de débrancher votre ordinateur.



CHAPITRE 5

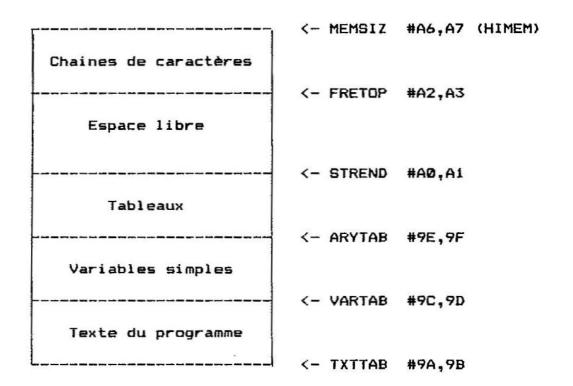
LES POINTEURS

Nous avons évoqué différents pointeurs çà et là au cours des précédents chapitres.

Il est temps de prendre un peu de hauteur et de faire le point, si j'ose dire.

Une carte de l'espace mémoire compris entre #500 et HIMEM avec les pointeurs indiquant les limites entre les différentes zones est plus explicite qu'un long discours.

L'ESPACE UTILISABLE ENTRE #500 ET HIMEM



Il convient de dire quelques mots sur l'interdépendance des principaux pointeurs. Ce n'est pas toujours évident. Voyons ce qui se passe dans deux cas concrets.

- Insertion ou suppression d'une ligne de programme.
 VARTAB est modifié du nombre de caractères de la ligne.
 ARYTAB et STREND sont faits égaux à VARTAB.
 De meme, FRETOP = MEMSIZ.
 Toutes les variables sont effacées.
 L'espace libre s'étend de VARTAB à HIMEM.
- Inscription dans la table d'une nouvelle variable simple. Prenons un cas assez complexe. Entrée d'une chaîne de caractères au clavier sur INPUT. Elle doit être placée en haut de mémoire. FRETOP est diminué d'un nombre d'octets égal à la longueur de la chaîne qui est recopiée à partir de cette adresse ARYTAB est testé.

C'est à l'adresse désignée par ce pointeur que la nouvelle variable sera installée.

Tous les tableaux sont déplacés de 7 octets vers le haut.
ARYTAB et STREND sont augmentés de 7.
La place est faite pour l'inscription, nom, longueur, pointeur d'adresse (recopie de FRETOP) et les deux octets nuls.
Avant tous ces mouvements, il y a vérification de l'espace disponible.

Un nettoyage du haut de la mémoire peut être entrepris. Un message OUT OF MEMORY sera affiché si la place reste insuffisante.

LES PRINCIPAUX POINTEURS BASIC

Les principaux pointeurs forment une suite continue en page zéro, de #9A à #B1.

L'octet faible est rangé en premier suivant la règle habituelle. Ils sont présentés dans le tableau ci-dessous.

TXTPTR en #E9,EA (pointeur de texte) qui a été ajouté à la liste est très spécial.

C'est une espèce d'index qui sert à déchiffrer n'importe quoi, caractère par caractère, aussi bien dans le texte Basic que dans le buffer d'entrée.

Il est déplacé par la routine d'obtention d'un caractère CHARGET (#E2) dont il fait partie.

NOM	position	fonction
TXTTAB VARTAB	#9A,9B #9C,9D	 Début du programme., normalement #501. Début de la zone des variables simples, des pointeurs de chaînes de caractères et des pointeurs de fonctions.
ARYTAB	#9E,9F	- Début de la zone des tableaux numériques et alphanumériques.
STREND	#AØ,A1	- Fin de l'espace utilisé par le programme, début de l'espace libre.
FRETOP	#A2,A3	- Bas de la zone des chaînes de caractères stockées en-dessous de HIMEM, fin de l'espace libre.
FRESPC	#A4,A5	- Pointeur temporaire utilisé par certaines routines de déplacement de chaînes de caractères.
MEMSIZ	#A6,A7	- Plus haute adresse en mémoire utilisable par le programme Basic + 1. Le stockage des chaînes commence à MEMSIZ - 1 en descendant. Peut être modifié par la commande HIMEM.
CURLIN	#A8,A9	Numéro de la ligne en cours d'exécution. En mode immédiat, #A9 contient #FF.
OLDLIN	#AA,AB	 Numéro de la ligne où se trouve l'instruction interrompue par CTRL-C, STOP ou END.
OLDTXT	#AC,AD	- Adresse du dernier octet de l'instruction qui vient d'être exécutée. Cet octet contient '3A' fin d'instruction ou '00' fin de ligne.
DATLIN	#AE, AF	- Numéro de ligne de DATA en cours de lecture.
DATPTR	#BØ,B1	- Pointeur d'adresse utilisé par READ pour lire la prochaine donnée en DATA.
TXTPTR	#E9,EA	- Adresse du dernier octet 'lu' par CHARGET.

Pour vous familiariser avec ces pointeurs, essayez le programme suivant:

7000 CLS: PRINT

7010 DOKE#B0, DEEK (DEEK (#AC)+1)-1

7020 DATA TXTTAB, VARTAB, ARYTAB, STREND, FRETOP, FRESPC

7030 DATA MEMSIZ, CURLIN, OLDLIN, OLDTXT, DATLIN, DATPTR

7040 A=#9A:DIM PTR\$(22):FOR I=0 TO 22 STEP 2:P=A+I:READ PTR\$(I)

7050 PRINT PTR\$(I), HEX\$(P); ", "; HEX\$(P+1), HEX\$(DEEK(P)), DEEK(P):

NEXT: END

RUN 7000

Ce programme présente deux particularités.

- Il se localise lui-meme grâce au pointeur OLDTXT (#AC,AD).
- Cela lui permet de renseigner directement DATPTR (#80,81) utilisé par READ pour lire la prochaine donnée en DATA.

Nous allons étudier de plus près ces deux pointeurs intéressants.

LE POINTEUR OLDTXT (#AC, AD)

Ce pointeur contient l'adresse de l'octet qui marque la fin de l'instruction qui vient d'être exécutée. Le contenu de cet octet est, soit #3A (:) fin d'instruction, soit #00 fin de ligne.

OLDIXI est renseigné en permanence par la routine NEWSTT, exécution d'une nouvelle instruction, que nous verrons en détail dans la troisième partie.

Il sert essentiellement à la commande CONT qui permet la pousuite de l'exécution d'un programme interrompu par CTRL-C, END ou STOP. Le contenu de OLDTXT est alors transféré dans TXTPTR et le programme repart.

Il faut noter aussi que CURLIN reçoit OLDLIN, mais seul le transfert de OLDTXT est décisif, comme le montre l'expérience suivante.

Recherchons la ligne 7000 en mémoire.

RUN 3000 puis !7000

Repérez avec précision l'adresse des deux-points ': après CLS. Au besoin, 3.modif et aidez-vous du curseur.

Retournez au menu, Stop et entrez successivement:

DOKE #AC , adresse des deux-points

CONT

L'exécution du programme a repris à partir de l'adresse des deux-points.

Vous remarquerez qu'une nouvelle adresse est affichée pour OLDTXT (fin de l'avant-dernière ligne). C'était son contenu au moment de l'affichage.

Il pointe maintenant sur la fin de programme. Vérifiez:

?HEX\$(DEEK(#AC)) et

CONT (inopérant)

CLEARC ou STKINI (voir 4°partie) ont pour effet de mettre l'octet fort de OLDTXT à zéro empéchant l'action de CONT. Vérifiez:

CLEAR puis

?PEEK(#AD) puis

CONT

LE POINTEUR DATPTR (#80,81)

READ utilise ce pointeur pour lire les données en DATA. Il est automatiquement fixé à (TXTTAB)-1 (normalement #500) par les commandes NEW, RUN ou RESTORE.

La commande READ parcourt le programme de ligne en ligne jusqu'à ce qu'elle rencontre le mot-clé DATA. Elle assure alors l'inscription des données en zone des variables.

Notre programme inscrit directement dans DATPTR l'adresse du début de la recherche (adresse de la ligne DATA concernée - 1) qui est l'adresse du 00 de fin de ligne précédente. C'est un RESTORE particulier.

Regardez attentivement les trois premières lignes.

A la ligne 7010, si #AC,AD contient l'adresse du dernier octet de l'instruction qui vient d'être exécutée, il s'agit du 00 de fin de ligne 7000.

La ligne 7010 commence donc à l'adresse suivante, c'est-à-dire DEEK(#AC)+1.

Nous avons vu qu'une ligne Basic commence toujours par deux octets contenant l'adresse de la ligne suivante.

DEEK(DEEK(#AC)+1) est donc l'adresse de la ligne 7020 DATA.

DEEK(DEEK(#AC)+1)-1 est l'adresse du zéro précédent à placer en #BØ,B1.

En réalité, le fonctionnement de READ est plus complexe. Il faudrait décortiquer cette routine, en #CD89 (ORIC-1 CCFD) pour expliquer ce qui se passe. Peut-être en aurez-vous envie après les essais suivants?

Remplacez le mot DATA par REM à la ligne 7020 et faites RUN 7000.

Maintenant modifiez la ligne 7010 comme suit

7010 DOKE#B0, DEEK (DEEK (#AC)+1)+12

RUN 7000

Replacez le mot DATA à la ligne 7020

RUN 7000

Remarque: On peut l'avouer maintenant, une ligne 7010 plus simple aurait suffi pour le bon fonctionnement de notre programme:

7010 DOKE#BO, DEEK (#AC)

Tout ceci a été compliqué à dessein pour vous entraîner à manipuler ces deux pointeurs.

La possibilité de se situer en mémoire au cours de l'exécution d'un programme est particulièrement intéressante.

Je vous laisse imaginer toutes les applications que l'on peut lui trouver.

MODIFICATION DU DEBUT DE PROGRAMME

Ayant reconnu une commande d'exécution, telle que RUN ou GOTO, l'interpréteur met l'adresse du zéro précédant la ligne à exécuter dans le pointeur de texte TXTPTR (#E9,EA).

La routine NEWSTT (nouvelle instruction) en #C8C1 (ORIC-1 C8AD) intervient alors pour identifier le mot-clé et exécuter la première instruction.

S'agissant d'une commande RUN seule, l'interpréteur se contente de faire TXTPTR = TXTTAB-1.

On peut le tromper en modifiant l'un ou l'autre de ces pointeurs par rapport à l'adresse d'une ligne quelconque.

Soit par exemple la ligne 7000 que nous venons de taper. Cherchons son adresse:

RUN 3000 puis !7000 notez l'adresse

Retournez au menu, Stop

- Première expérience: TXTPTR = adresse de la ligne - 1

DOKE#E9,adresse ligne 7000 - 1

Nous venons de reproduire un 60TO 7000

- Deuxième expérience: TXTTAB = adresse de la ligne

DOKE#9A,adresse ligne 7000

Faites RUN, puis LIST.

Dorénavant, l'ordinateur ignore tout ce qui se trouve avant la ligne 7000

Redonnez à TXTTAB sa valeur initiale: DOKE#9A,#501

Peut-être trouvez-vous inesthétiques les quatre premières lignes de votre programme?

Elles seront définitivement cachées pour l'utilisateur si vous ajoutez une ligne 4 comme ci-dessous et si vous enregistrez le programme en AUTO.

4 DOKE#9A, DEEK (DEEK (#AC)+1)

RECUPERATION DU PROGRAMME APRES NEW

Nous allons passer à un exercice qui devrait éveiller l'attention des plus blasés.

Bien que son aspect pratique soit discutable, il colle parfaitement à notre sujet car il concerne les deux pointeurs fondamentaux TXTTAB et VARTAB.

VARTAB, qui indique le début de la zone des variables, sert aussi à situer la fin du texte Basic, c'est-à-dire du programme proprement dit, égale en effet à VARTAB-1.

Supposez que vous avez fait un NEW malencontreux alors qu'un programme important se trouve en mémoire.

A moins d'avoir, par dépit, arraché la prise de courant, votre programme n'est pas perdu.

Il est utile de rappeler un principe essentiel.

Sauf nécessité, l'ordinateur ne prend jamais la peine d'effacer une information périmée. Il se contente de modifier ses pointeurs. L'information reste en mémoire jusqu'à ce qu'une autre prenne sa place (vienne l'écraser).

Quel sont les effets de la commande NEW?

- Elle met à zéro les deux premiers octets du programme (#501 et #502). Ce sont les octets de chainage de la première ligne. Ils contenaient l'adresse de la ligne suivante.
- Elle met en VARTAB (#9C,9D) l'adresse #503.

Ensuite, elle exécute l'équivalent d'un RUN, TXTPTR à #500, CLEAR qui réajuste les autres pointeurs en fonction de VARTAB, RESTORE puis réinitialisation de la pile.

Pour les commandes RUN et LIST, il n'y a plus de programme en mémoire, car partant de #500 elles trouvent trois octets nuls et ne vont pas plus loin.

Le problème est simple.

Il faut retrouver les valeurs initiales. Comment?

Fouiller la mémoire avec des PEEK et localiser les 00 de fin de ligne et de programme?

Il faut procéder octet par octet en commande directe. On ne peut pas utiliser une boucle car la variable FOR écraserait le début du programme à partir de #503.

C'est un travail de besogneux qu'un programme devrait pouvoir faire plus rapidement.

Comment implanter et faire tourner un tel programme sans 'écraser' ce que nous voulons récupérer?

On pourrait, par exemple, par une série de POKE, installer en page 4, une routine en langage machine.

Il existe une solution Basic plus simple et, je dirais , plus parlante puisque c'est votre langue maternelle.

Vous devriez pouvoir suivre le processus sans trop de peine.

Avant de commencer, ouvrons une parenthèse.

L'adresse de fin de programme est lue sur la bande par CLOAD et inscrite en #2AB,2AC (#5F,60 pour DRIC-1).

C'est de peu d'intérêt car, ou bien le programme n'a pas été modifié

après chargement et il suffit de le recharger, ou bien il a été modifié et cette adresse n'est plus valable.

Néammoins, cela peut être une indication. Fermons la parenthèse.

Votre utilitaire étant en mémoire, supprimez-le (virtuellement) avec

Nous allons entrer en haut de mémoire, un autre programme Basic. A partir de l'adresse #8501, par exemple, ce qui laisse une bonne marge de sécurité (32K).

POKE #8500,0

(indispensable)

DOKE #9A, #8501

(début de programme)

(réajuste les autres pointeurs)

Entrez le programme:

10 A=#505

20 IF PEEK(A)=0 THEN DOKE#501,A+1:PRINT HEX\$(A+1):GOTO 40

3Ø A=A+1:GOTO 2Ø

40 REPEAT: A=A+1:UNTIL PEEK(A)=0 AND PEEK(A+1)=0 AND PEEK(A+2)=0

50 PRINT HEX\$(A+3):DOKE#9A,#501:DOKE#9C,A+3:RUN

La recherche commence en #505 après les octets d'identification. La ligne 20 recherche le zéro de fin de première ligne et inscrit son adresse + 1 en #501.

La ligne 40 recherche les trois zéros de fin de programme.

La ligne 50 réajuste les pointeurs TXTTAB et VARTAB, les autres seront rétablis automatiquement par RUN.

Faites RUN et patientez deux ou trois minutes.

Par curiosité, regardez en #8500 notre programme resté en place et devenu inutile.

Note: (confidentielle)

Il existe une méthode plus radicale pour récupérer un programme après NEW.

L'adresse à inscrire en #501 est relativement facile à trouver en tàtonnant. Nous savons depuis longtemps que la nôtre est #528.

En ce qui concerne VARTAB, l'ORIC n'est pas si intransigeant. Il suffit de lui proposer une adresse supérieure à celle de la fin du programme et il l'acceptera docilement, sans jamais la remettre en cause.

Si la place en mémoire n'est pas critique, on prendra une bonne marge de sécurité.

Recommençons l'opération:

Faites NEW

DOKE#501,#528

DOKE#9C,#5000

et maintenant seulement RUN.

Entre le programme et ses variables, que de place perdue, direz-vous! Pas forcément. On peut y loger du code machine.

C'est ce que nous verrons au chapitre 10.

Pour l'exercice suivant, redonnez à VARTAB une valeur plus convenable ou rechargez votre utilitaire.

LE PROGRAMME BASIC EN HAUT DE MEMOIRE

Il peut être intéressant de faire tourner un programme Basic en haut de mémoire.

Ne serait-ce qu'un utilitaire comme celui-ci, pour examiner un programme en langage machine placé à partir de l'adresse #500.

Le programme étant sur cassette, la première solution qui vient à l'esprit est de donner une nouvelle valeur à TXTTAB et d'essayer CLOAD.

Malheureusement ça ne marche pas car les routines CSAVE et CLOAD sont ainsi conçues qu'elles prennent en compte la fin du programme et non pas sa longueur.

Elles se servent de VARTAB.

On pourrait probablement remanier la routine CLOAD et employer le langage machine mais c'est, à première vue, assez compliqué.

Nous utiliserons une méthode moins élégante mais éprouvée. Elle consiste à transférer le programme en haut de mémoire et à modifier les pointeurs.

Votre utilitaire étant en place, lancez-le et choisissez l'option 2, adresse #400, puis 3.modif.

Entrez le programme en langage machine ci-dessous: Comme d'habitude, vous n'avez qu'à recopier le code hexadécimal, à gauche.

ATMOS		ORIC-1	
	.DA >#ØØ	A9 00 LDA >#00	;les valeurs
85 ØC S	ITA #ØC	8D 00 02 STA #200	;'fixes'
85 ØE S	ITA #ØE	8D 02 02 STA #202	;sont rangées
A9 Ø5 L	.DA >#Ø5	A9 Ø5 LDA >#Ø5	; aux endroits
95 ØD S	TA #ØD	8D Ø1 Ø2 STA #201	;ad-hoc.
A5 9C LI	.DA #9C	A5 9C LDA #9C	;Nb d'octets =
85 10 S	TA #10	8D 04 02 STA #204	;long.progr.+1
38 S	EC	38 SEC	; (00 en #500)
A5 90 L	.DA #9D	A5 9D LDA #9D	;rangé en
E9 Ø5 S	BC >#05	E9 Ø5 SBC >#Ø5	;#10,11 ou
85 11 S	STA #11	8D 05 02 STA #205	; #204,205
20 C5 D8 J	SR #D8C5	20 0A DB JSR #D80A	;GETBYTC,n°page
BA T	XA	BA TXA	;dans A puis
85 9B 5	TA #9B	85 9B STA #9B	;dans TXTTAB+1
85 ØF S	TA #ØF	8D 03 02 STA #203	;en #QF (#203)
18 C	CLC	18 CLC	;calcul
65 11 A	DC #11	6D Ø5 Ø2 ADC #2Ø5	;act fort de
85 9D S	TA #9D	85 9D STA #9D	; VARTAB
20 C4 ED J	SR #EDC4	20 0C EC JSR #ECOC	; MOVE, transfert
20 0F C7 J	SR #C70F	20 3A C7 JSR #C73A	; CLEARC
20 5F C5 J	SR #C55F	20 AF C5 JSR #C5AF	;LINKSET
4C A8 C4 JI	MP #C4AB	4C B5 C4 JMP #C4B5	Basic, Ready

Lorsque c'est fait, \fin puis 2.desassemblage pour vérifier.

C'est MOVE qui va transférer le programme en haut de mémoire. Cette routine est utilisée par le système pour déplacer le jeu de caractères, de ROM en RAM pendant l'initialisation, puis entre les adresses #8400 et #9800 en fonction du mode d'affichage. Les emplacements #0C-#ii (ORIC-1 #200-#205) doivent contenir les valeurs suivantes:

	adresse	départ	destin	ation	nombre d	'octets
	faible	fort	faible	fort	faible	fort
	octet	octet	octet	octet	octet	octet
ORIC-1	#200	#201	#202	#203	#204	#205
ATMOS	#ØC	#ØD	#ØE	#ØF	#10	#11

Note:

La routine MOVE est utilisable à partir du Basic, avec restrictions. (voir plus loin).

Vous appellerez la routine de transfert avec une instruction de la forme:

CALL#400,P où P est le numéro (décimal pour ATMOS) de la nouvelle page à partir de laquelle vous voulez placer le programme.

Attention, ce nombre doit être choisi en fonction de la longueur du programme et de l'espace de travail qui lui est nécessaire. La routine en langage machine est rudimentaire et n'a pas de sécurités.

Exemples:

Votre programme commence en #501, vous voulez le transférer en:

- #6001 P=#60 =96 décimal il faudra faire CALL#400,96 - #8501 P=#85 =133 décimal CALL#400,133

Transférons le programme en 7501.

option 3.Stop. Entrez en commande directe:

CALL#400,117 (pour ORIC-1, CALL#400,#75 est valable)

Votre programme est maintenant en haut de mémoire et opérationnel.

Pour vérifier, faites RUN, option 2, adresse #7500.

Et pour être absolument certain, retournez au menu, option 2, adresse #500, 3.modif et inscrivez n'importe quoi.

Il est possible de placer dans la zone rendue libre un programme en langage machine ou en Basic qui pourra être examiné et éventuellement modifié.

Attention, un programme sur cassette ne doit pas être chargé directement avec l'utilitaire en mémoire. Cela perturberait à nouveau les pointeurs.

Il faut d'abord sauvegarder le programme modifié. Vous pouvez le faire de la manière habituelle, sans précaution particulière.

Mettez cependant une étiquette car l'opération de chargement sera différente.

Faites CALL#C000 afin de réinitialiser le système, puis chargez le programme à examiner.

Pour recharger votre utilitaire:

POKE#7500,0 DOKE#9A,#7501 puis CLOAD""

TRANSFERT EN BASIC

Il est intéressant de disposer d'une routine de transfert de blocs.

Malheureusement, le système se sert par ailleurs des emplacements où sont rangés les paramètres utilisés par MOVE. Il y a donc certaines restrictions à l'emploi de cette routine à partir du Basic.

Sur ATMOS, le pointeur d'adresse #0C,0D est utilisé dans l'évaluation d'un nombre hexadécimal (conversion hexa-binaire en #E94C). La suite d'instructions Basic devra avoir cette forme: DOKE#0E, destination: DOKE#10, nombre d'octets: DOKE#0C, départ en décimal: CALL 60868

Sur ORIC-1, les emplacements #204 et #205 reçoivent le nombre de caractères de la mémoire écran à déplacer par un éventuel scrolling (26×40).

Les instructions nécessaires doivent être obligatoirement dans un programme.

Voici une procédure de transfert d'un programme Basic qui tient compte de ces limitations. Attention de toujours choisir avec soin la nouvelle page.

60000 INPUT"nouvelle page";N 60010 D=N*256:L=DEEK(#9C)-#500

Pour ATMOS 60020 DOKE#0E,D:DOKE#10,L:DOKE#0C,1280:CALL 60868

Pour ORIC-1 60020 DOKE#200,#500:DOKE#202,D:DOKE#204,L:CALL#EC0C

Pour l'un et l'autre 60030 PRINT"entrez en commande directe" 60040 PRINT"DOKE#9A,D+1" 60050 PRINT"DOKE#9C,D+L"

Pour ATMOS 60060 PRINT"CALL#C70F" 60070 PRINT"CALL#C55F"

Pour ORIC-1 60060 PRINT"CALL#C73A" 60070 PRINT"CALL#C56F"

RUN 40000

DEUXIEME PARTIE

INITIATION AU LANGAGE MACHINE

INTRODUCTION

Cette deuxième partie s'adresse aux débutants et plus généralement à tous ceux qui ne connaissent pas le 6502.

Les pages qui suivent ne vous apprendront pas à vous exprimer couramment dans le langage de la machine. Elles présentent le vocabulaire de base qui vous permettra de lire dans le texte, à tête reposée.

Le but recherché est de vous donner l'information suffisante pour comprendre le mécanisme des routines de la ROM afin de les utiliser directement ou simplement d'améliorer les performances de votre Basic.

Contrairement aux langages évolués qui sont plus ou moins transportables d'un système à l'autre, le langage machine est celui du microprocesseur utilisé.

L'Oric est bâti autour du 6502A, version récente, plus rapide, du 6502 qui, concurremment avec le 780, équipe la quasi-totalité des ordinateurs individuels.

C'est probablement le microprocesseur pour lequel le plus de logiciels ont été écrits.

Sa conception est différente de celle d'autres microprocesseurs, dont le Z80 que vous connaissez peut-être.

La principale particularité concerne la gestion des entrées-sorties. La communication avec les périphériques ne nécessite pas d'instructions spéciales.

Elle se fait par l'intermédiaire d'adresses en mémoire vive. En contre-partie, la complexité des opérations est reportée sur le boitier d'interface spécialisé.

Nous allons parcourir le jeu complet des instructions du 6502, regroupées par grandes fonctions. Quelques exercices sont proposés.

N'hésitez pas à écrire vos propres compositions. Laissez libre cours à votre imagination.

Dans ce domaine, la pratique est irremplaçable.

Auparavant, il est nécessaire de situer le microprocesseur dans son environnement, de se familiariser avec sa structure interne et avec certaines caractéristiques de fonctionnement.

!



CHAPITRE 6

LE MICROPROCESSEUR

PRESENTATION

L'Unité Centrale de l'Oric (CPU) est le 6502A, dernière version du 6502 de MOS Technology

La base de temps est fournie par une horloge de fréquence 2 MHz. C'est un critère qui entre en jeu pour apprécier la vitesse de traitement mais il n'est pas le seul.

Le concept de fonctionnement est au moins aussi déterminant.

Par rapport à un standard des microprocesseurs, s'il existait, les caractérisques du 6502 pourraient être résumées ainsi:

- jeu d'instructions relativement limité.
- richesse des modes d'adressage.
- banalisation des entrées-sorties.

Le microprocesseur ne travaille pas en circuit fermé. Cette puce est reliée par ses quarante pattes au monde extérieur, ROM, RAM, ULA, 6522.

Les liaisons sont de trois sortes:

- les lignes qui assurent l'alimentation et celles qui acheminent les différents signaux de synchronisation (lignes de commande).
- le bus de données, bi-directionnel, 8 lignes groupées, transporte 8 bits en parallèle (0 ou 1 suivant le niveau de tension de la ligne concernée).
- le bus d'adresses, unidirectionnel, 16 lignes groupées (16 bits, 2 octets). Il permet de désigner les 65536 emplacements de la mémoire centrale.

Sous un boitier unique, le 6502 cache une structure complexe formée de divers éléments: unité de commande, registre instruction, unité arithmétique et logique et surtout 6 registres internes importants qu'il est nécessaire de connaître.

LES REGISTRES INTERNES

- Le compteur ordinal PC (Program Counter), seul registre 16 bits, sert à rechercher les instructions en mémoire. Il contient l'adresse de la prochaine instruction à exécuter. Aussitôt exploité, son contenu est automatiquement incrémenté pour chaque octet de l'instruction, de sorte qu'avant même que débute l'exécution, il pointe déjà l'adresse de l'instruction suivante. Il n'est pas directement accessible au programmeur mais les instructions de saut ou de branchement le modifient, rompant ainsi le cours séquentiel du programme.
- Le registre A ou Accumulateur est le registre 8 bits le plus important à la disposition du programmeur. Son contenu représente toujours l'un des termes des opérations arithmétiques ou logiques.

Il est souvent employé pour le transfert de données d'un emplacement à un autre.

Certaines instructions de rotation ou de décalage peuvent affecter directement son contenu.

- Les registres index X et Y ont trois fonctions importantes.

Comme leur nom l'indique, ils peuvent servir d'index pour rechercher, stocker, manipuler ou récupérer des données en mémoire.

Le contenu du registre index est ajouté à une adresse de base.

Il est éventuellemnt changé pour accéder à des emplacements différents (256 maximum).

Ces registres sont souvent employés comme compteurs de boucles car des instructions d'incrémentation et de décrémentation leur sont directement applicables.

Enfin, ils ont le même usage général que l'accumulateur pour le tranfert de données.

- Le registre pointeur de pile S (Stack pointer), 8 bits, est une sorte d'index qui peut se déplacer le long de la page 1 réservée à la pile.

Il indique le "sommet" de la pile - 1

La **pile** occupe 256 octets en mémoire vive, des adresses #1FF à #100.

Utilisée principalement par le système pour le stockage des adresses de retour de sous-programme (Basic ou langage machine), elle peut servir au programmeur pour le rangement temporaire de données. L'information est empilée, un octet à la fois, à partir de l'adresse #1FF dans le sens des adresses décroissantes.

Elle est récupérée dans l'ordre inverse.

Dernière entrée, première sortie (LIFO - Last In, First Out). A chaque fois qu'une valeur est empilée ou retirée de la pile, le pointeur de pile est décrémenté ou incrémenté respectivement. Il indique toujours le prochain emplacement disponible.

- Le registre d'état P (Processor status register), plutôt qu'un véritable registre, est une collection de bits ayant chacun une fonction particulière, les indicateurs d'état. Sept des huit bits du registre P sont significatifs. Le bit n' 5 n'est pas utilisé.

LES INDICATEURS D'ETAT

registre d'état

bit n° 7 6 5 3 4 2 Ø V Z N В D I C signe débordement break décimal interrupt zéro retenue

Le 6502 ne dispose pas d'instructions du genre IF..THEN... Ses seules instructions conditionnelles sont des branchements basés sur le test des quatre principaux indicateurs C, Z, V et N.

- L'indicateur de retenue C (Carry) est affecté par les opérations arithmétiques.

Après une addition. il est mis à un (retenue) si le résultat dépasse la capacité de l'octet, à zéro dans le cas contraire.

Après une soustraction, sa mise à zéro indiquera un "emprunt", sa mise à un l'absence d'emprunt. Les comparaisons, qui sont des soustractions virtuelles, ont le même effet.

L'indicateur de retenue prend la valeur du bit sortant lors des décalages et des rotations.

Il peut être forcé à zéro par l'instruction CLC (CLear Carry) ou à un par SEC (SEt Carry).

- L'indicateur de zèro Z est affecté par les opérations arithmétiques ou logiques, l'incrémentation ou la décrémentation, les rotations ou les décalages.
- Il est mis à un si la valeur zéro est produite à la suite d'une de ces opérations, sinon il est mis à zéro.
- L'indicateur de débordement V (oVerflow) indique un dépassement de capacité en arithmétique binaire signée. Il est mis à un lorsque le résultat d'une opération arithmétique sur des nombres signés est erroné par suite d'une retenue du bit 6 vers le bit 7, bit de signe en complément à deux. Il est forcé à zéro par CLV (CLear oVerflow)
- L'indicateur de signe N (Negative) est mis à un lorsque le résultat d'une opération est supérieur à #7F, c'est-à-dire représente un nombre négatif en complément à deux, il est mis à zéro dans le cas contraire.

Cet indicateur étant le bit n° 7 du registre d'état, il peut être considéré comme la recopie du bit de signe du résultat.

- L'indicateur de break B est mis à un lorsque le microprocesseur rencontre l'instruction BRK (#00). Il indique alors une interruption logicielle qui doit être traitée en conséquence par la routine d'interruption.

Les deux derniers sont des indicateurs d'un mode de fonctionnement choisi par le programmeur.

- L'indicateur de mode décimal D est positionné à un par l'instruction SED (SEt Decimal mode) qui met le 6502 en mode décimal codé binaire (BCD) pour les opérations arithmétiques. Il est mis à zéro par l'instruction CLD (CLear Decimal mode) qui rétablit le mode arithmétique binaire.
- L'indicateur de masque d'interruption I (Interrupt flag) est mis à un par l'instruction SEI (SEt Interrupt disable) qui interdit les interruptions. Il est mis à zéro par CLI (CLear Interrupt disable) qui les autorise à nouveau.

LES MODES D'ADRESSAGE

Il vous tarde certainement de retrouver votre clavier. , Lancez le programme utilitaire. Option 2, adresse #400 puis 3.modif Entrez le code machine:

A9 35 AD 20 04 A5 35 BD 30 04 B5 80 B6 80 6C F5 02 A1 10 B1 E9 D0 E6 AA 0A

Appuyez sur \fin puis 2.désassemblage.

Vous avez à l'écran un exemple de chacun des treize modes d'assemblage du 6502. Les voici, cités dans l'ordre:

immédiat - absolu - page zéro - absolu,X - absolu,Y - page zéro,X
page zéro,Y - indirect - indirect,X - indirect,Y - relatif
implicite - accumulateur.

Vous remarquerez que huit des instructions ont pour but de charger l'accumulateur (LDA LoaD Accumulator).

Si les opérations sont de même nature, précisée par le mnémonique LDA, elles différent cependant par la façon dont est spécifiée l'adresse de la donnée à charger, c'est-à-dire par leur mode d'adressage. Vous noterez aussi que les premiers octets de ces instructions (colonne de gauche) qui représentent leur code opératoire ont tous une valeur différente.

C'est ce code opératoire qui a servi à notre désassembleur à déterminer le nombre d'octets et le mode d'adressage à afficher. C'est ce qui se passe lorsque le microprocesseur décode une instruction. Reconnaissant le code opératoire, il sait comment trouver la donnée à traiter, ici à charger dans l'accumulateur.

- adressage immediat

Ce n'est pas à proprement parler un mode d'adressage puisque la donnée elle-même suit le code opératoire.

Dans notre exemple, l'accumulateur serait chargé avec la valeur #35. Le symbole > indique qu'il s'agit d'un nombre et non d'une adresse. Avec un désassembleur classique on aurait LDA #\$35.

- adressage absolu

C'est le mode le plus naturel.

L'adresse sur deux octets suit le code opératoire.

Le microprocesseur y trouvera la donnée à traiter ou y branchera le programme s'il s'agit d'un saut (JMP).

Notez l'inversion des octets dans le code machine.

- adressage page zéro

C'est un cas particulier du précédent.

La page zéro est la seule zone en mémoire dont les adresses peuvent être désignées sur un octet, en ignorant l'octet fort 00 n° de page. Il faut, bien sûr, disposer d'instructions connues du microprocesseur. C'est le cas, le code opératoire est différent du précédent. L'instruction occupe deux octets au lieu de trois et elle est exécutée plus rapidement (trois cycles d'horloge au lieu de quatre)

- adressage absolu, X

Adressage indexé classique.

Le contenu du registre index est ajouté à une adresse de base dont les deux octets suivent le code opératoire.

Il permet d'accéder aux divers éléments d'une table ou d'une suite de codes.

Le registre index joue le même role que l'indice dans les tableaux de variables.

Sa capacité limite l'indexation possible à 256 octets.

- adressage absolu, Y

Fonctionnement strictement identique au précédent.

- adressace page zéro, X

Même principe de fonctionnement que les modes précédents.

Le contenu du registre index est ajouté à l'adresse de base limitée à la page zéro.

Si la somme obtenue est supérieure à #FF, le dépassement est ignoré et il y retour au début de la page zéro.

- adressage page zero, Y

Ce mode d'adressage n'est pas utilisable pour le chargement de l'accumulateur.

Il n'est disponible que pour les seules opérations, LDX et STX, chargement et rangement de X.

Fonctionnement identique au mode page zéro,X

- adressage indirect

Dans ce mode d'adressage, c'est l'adresse de l'adresse qui est spécifiée.

Il n'est utilisé qu'avec JMP pour le saut indirect.

L'adresse d'un emplacement en mémoire, toujours sur deux octets, suit le code opératoire.

A cet emplacement et à l'emplacement suivant se trouvent respectivement l'octet faible et l'octet fort de l'adresse cherchée, destination du saut.

L'exemple présenté reproduit le saut indirect obtenu par la commande Basic! (entrée en #CD13 ATMOS, en #CC89 ORIC-1).

Pour les deux modes suivants, il s'agit aussi d'adressage indirect mais il est combiné avec une indexation.

Remarque: Le pointeur d'adresse est toujours en page zéro, les instructions occupent deux octets et chacun des registres index est spécialisé.

Ils ne sont plus interchangeables.

- adressage indirect, X

Ce mode porte également l'appellation indexé indirect (ou pré-indexé) qui en traduit mieux le mécanisme.

Le contenu de X est ajouté à une adresse en page zéro ce qui détermine une nouvelle adresse, toujours en page zéro (le dépassement éventuel est ignoré comme dans le mode direct).

A l'emplacement ainsi défini et à l'emplacement suivant se trouvent respectivement l'octet faible et l'octet fort de l'adresse cherchée. Ce mode d'adressage n'est pratiquement jamais utilisé.

Je pense que l'on n'en trouverait pas un seul exemple dans toute la ROM de l'Oric.

- adressage indirect.Y

Par contre celui-ci, appelé aussi indirect indexé ou post-indexé, est abondamment employé.

L'emplacement désigné par le pointeur en page zèro et l'emplacement suivant contiennent l'adresse de base à laquelle est ajouté le contenu du registre Y, formant ainsi l'adresse cherchée.

Dans notre exemple, l'accumulateur est chargé avec le contenu de l'emplacement situé à une certaine 'distance' Y de l'adresse pointée par TXTPTR (pointeur de texte en #E9,EA).

- adressage relatif

Ce mode d'adressage est exclusivement réservé aux instructions de branchement conditionnel basé sur l'état d'un des quatre indicateurs C. Z. V ou N.

Un déplacement signé, positif ou négatif (-128 + 127), est ajouté au contenu du compteur ordinal provoquant le branchement en avant ou en arrière désiré.

Les instructions de branchement occupent deux octets.

Le deuxième octet représente le déplacement traité en complément à deux.

Le compteur ordinal contenant l'adresse de l'instruction suivante, c'est à partir de cette adresse qu'est calculé le déplacement (adresse de l'octet suivant l'instruction de branchement).

Dans notre exemple, #E6 valant -26 en complément à deux (voir table en annexe), vérifiez que le branchement aboutirait bien en #400.

- adressage implicite

Les instructions qui utilisent ce mode sont toutes sur un octet. La désignation de l'adresse est inutile.

Elle est 'implicitement' contenue dans l'instruction elle-même. Exemple: TAX transférer le contenu de A dans X.

- adressage accumulateur

Ce mode est employé dans les instructions de décalage ou de rotation portant sur le contenu de l'accumulateur (quatre instructions d'un octet).

L'indication de ce registre "A" est souvent omise par les assembleurs.

LANGAGE D'ASSEMBLAGE LE JEU D'INSTRUCTIONS DU 6502

Il est difficile de programmer directement en binaire, ou même en hexadécimal.

Le langage de programmation employé est le langage d'assemblage. Tout en respectant le découpage en actions élémentaires du langage machine, il traduit sous une forme symbolique, plus compréhensible, ce qui ne serait qu'une suite de codes numériques, difficiles à déchiffrer.

Nous n'allons pas nous attarder sur sa présentation.

Vous avez déjà fait connaissance avec bon nombre de mnémoniques du langage.

Précisons qu'il est souvent improprement appelé assembleur. En réalité, l'assembleur est un programme chargé de traduire directement en binaire, les instructions entrées en langage d'assemblage.

A défaut d'assembleur, tout à fait superflu pour ce qui nous occupe, vous ferez l'assemblage à la main, c'est-à-dire que vous traduirez en hexadécimal les instructions en langage d'assemblage.

C'est sous cette forme hexadécimale que vous entrerez le code machine en mémoire.

Le 6502 est capable d'effectuer 56 opérations de nature différente rappelée par le mnémonique: LDA, JMP, ASL etc

En combinaison avec les modes d'adressage disponibles, cela donne 151 instructions différentes.

50

Nous allons les passer successivement en revue.

Elles sont, plus ou moins arbitrairement, regroupées par fonctions dans les prochains chapitres.

Une instruction occupe 1. 2 ou 3 octets.

Le temps d'exécution, exprimé en cycles d'horloge, varie suivant les instructions. Il est compris entre 2 et 7 cycles.

Un temps moyen, tout à fait approximatif, pourrait être 1/2 (MHz)*4 = 2 microsecondes.

Le 6502A exécuterait ainsi environ 500000 instructions par seconde. Ce n'est qu'un ordre de grandeur, le microprocesseur ne tourne pas à pleine vitesse sur Oric.



CHAPITRE 7

TRANSFERT DE DONNEES

Le microprocesseur passe la moitié de son temps à déplacer des informations d'un endroit à un autre.

Les instructions de transfert sont fondamentales en langage machine.

Il est bon de rappeler qu'elles opérent sur des données de 8 bits (1 octet) et que l'information est simplement copiée à l'emplacement spécifié.

Le contenu de l'emplacement source reste intact jusqu'à ce qu'une autre information prenne sa place.

Les opérations étudiées dans ce chapitre se répartissent en quatre groupes.

Les premières lettres du mnémonique en rappellent la nature.

- chargement de registres (LoaD)
- rangement de leur contenu en mémoire (STore)
- transferts entre registres (T..)
- transferts avec la pile (PusH, PulL)

CHARGEMENT DE REGISTRES

- Chargement de l'accumulateur

mnémoni que	code	mode d'adressage
LDA >donnée	A9 FF	immédiat
LDA ADR	AD ff FF	absolu
LDA adr	A5 FF	page zéro
LDA ADR,X	BD ff FF	absolu,X
LDA ADR,Y	B9 ff FF	absolu,Y
LDA adr, X	B5 FF	page zéro,X
LDA (adr, X)	A1 FF	indexé indirect
LDA (adr),Y	B1 FF	indirect indexé

Indicateurs affectés: Z, N

LDA est certainement la plus employée de toutes les instructions. L'accumulateur étant impliqué dans beaucoup d'opérations, elle permet de recueillir l'information dans ce registre important.

Notez l'inversion des octets désignant l'adresse absolue dans le code machine.

- chargement des registres index

mnémonique	code	mode d'adressage
LDX >donnée	A2 FF	immédiat
LDX ADR	AE ff FF	absolu
LDX adr	A6 FF	page zéro
LDX ADR,Y	BE ff FF	absolu,Y
LDX adr,Y	B6 FF	page zéro,Y
LDY >donnée	AØ FF	immédiat
LDY ADR	AC ff FF	absolu
LDY adr	A4 FF	page zéro
LDY ADR,X	BC ff FF	absolu,X
LDY adr,X	B4 FF	page zéro,X

Indicateurs affectés: Z, N

Chargement des registres index en vue de leur emploi, indexation, compteur de boucle, transfert d'information.
Remarquez la symétrie de X et Y. Ils peuvent s'indexer mutuellement.

RANGEMENT EN MEMOIRE

- rangement du contenu de l'accumulateur

mnémonique	code	mode d'adressage
STA ADR	8D ff FF	absolu
STA adr	85 FF	page zéro
STA ADR,X	9D ff FF	absolu,X
STA ADR,Y	99 ff FF	absolu,Y
STA adr, X	95 FF	page zéro,X
STA (adr, X)	81 FF	indexé indirect
STA (adr), Y	91 FF	indirect indexé

Indicateurs affectés: aucun

STA est une instruction très souvent employée.

Elle sert, à la fin de nombreuses opérations, à ranger le résultat en mémoire.

En général, le couple LDA/STA est utilisé pour transférer des données d'un emplacement à un autre.

Comme pour LDA, notez l'absence d'adressage page zéro, Y.

- rangement du contenu des registres index

mnémonique	code	mode d'adressage
STX ADR STX adr STX adr,Y STY ADR STY adr STY adr	8E ff FF 86 FF 96 FF 8C ff FF 84 FF 94 FF	absolu page zéro page zéro,Y absolu page zéro
orr aur, A	74 FF	page zéro,X

Indicateurs effectés: aucun.

Utilisation pour le transfert de données de LDX/STX ou LDY/STY.

TRANSFERT ENTRE REGISTRES

Mode d'adressage implicite

code	signification
AA	transfert de A dans X
AB	transfert de A dans Y
8A	transfert de X dans A
98	transfert de Y dans A
BA	transfert de 5 dans X
	AA AB 8A 98

Indicateurs affectés: Z, N

TXS 9A transfert de X dans S

Indicateurs affectés: aucun

Les instructions de transfert entre registres ont beaucoup d'applications.

Notamment, TXA et TYA sont un moyen de charger l'accumulateur avec une donnée à traiter. Ce dernier étant seul utilisable pour de nombreuses opérations.

Le transfert entre X et Y peut être réalisé en faisant transiter la donnée par l'accumulateur. Exemple: pour $X \rightarrow Y$ TXA puis TAY TSX et TXS sont les seules instructions qui permettent de connaître ou de modifier la valeur du pointeur de pile S.

En outre, TSX permet de recueillir une donnée en pile.

Exemple: les instructions TSX, LDA #101,X sont équivalentes à PLA. (En faisant varier X, ou l'adresse de base #100, on peut récupérer de cette façon une donnée "enfouie" sans déranger la pile).

TRANSFERTS AVEC LA PILE

Mode d'adressage implicite.

mnémonique	code	signification					
PHA PHP	48 Ø8	empile l'accumulateur empile le registre d'état P					
Indicateurs affectés: aucun							
PLA	68	"désempile" vers l'accumulateur					
Indicateurs affectés: Z, N							
PLP ,	28	"désempile" vers registre d'état					

Indicateurs affectés: tous (-> contenu de l'octet récupéré)

L'instruction PHA est couramment employée avec PLA pour stocker temporairement des données et les recouvrer ensuite.

D'une manière générale, à chaque PHA doit correspondre un PLA plus tard dans la routine car le microprocesseur range en pile ses adresses de retour et doit pouvoir les retrouver.

Les commandes POP ou PULL du Basic peuvent être simulées avec deux PLA successifs.

De même, deux PHA peuvent provoquer le retour à une adresse calculée. Le contenu du registre d'état peut être sauvegardé par PHP en vue d'un test ultérieur des indicateurs. Il est recouvré par PLP.

Rappelez-vous que toute opération d'empilage ou de désempilage est suivie automatiquement de la mise à jour du pointeur de pile S. Ce dernier indiquant le prochain emplacement disponible.

Les instructions où la pile est impliquée doivent être maniées avec précaution et méthode.

Il faut toujours savoir avec précision dans quel ordre sont empilées les informations.

EXERCICES

Les exercices proposés consistent à bien saisir le mécanisme de courts programmes tout faits dont vous entrerez le code hexadécimal de la manière habituelle, à partir de l'adresse #400. Il vous faudra les documenter en quelque sorte.

Appelez le désassembleur pour vérifier.

Pour l'exécution, il suffit de stopper le programme utilitaire puis de faire CALL#400 en commande directe.

Vous rencontrerez des instructions qui ne seront introduites que dans les prochains chapitres ou des appels de routines du système en ROM. Contentez vous de les entrer en mémoire sans chercher à comprendre. Vous y reviendrez.

Les utilisateurs d'ORIC-1 modifieront le code conformément aux adresses indiquées entre parenthèses.

Dans ce livre, les branchements en arrière sont, hélas, inévitables. Heureusement, l'imprimeur a choisi un papier résistant.

Petite fantaisie hors du sujet pour vous mettre en train. Lettres seulement

				ATMOS		ORIC-1	
#400- #403-	20 E8			#C5E8 #D216			caractère en A est-ce une lettre?
#406-	90 0			#4ØE	, 1000	(#0100)	non
#408-	20 D			#CCD9	OUTDO	(#CC12)	oui, l'afficher
#4ØB-	4C Ø			#400	1	(110012)	on recommence
#4ØE-	4C CI	B FA		#FACB		(#FAB1)	boum, fini
ORIC							
#400-	A9 4F	=	LDA	>#4F			
#402-	8D 70	BD BD	STA	#BD7Ø			
#405-	A2 52	2	LDX	>#52			
#407-	8E 71	BD	STX	#BD71			
#4ØA-	AØ 49	7	LDY	>#49			
#4ØC-	8C 72	2 BD	STY	#BD72			
#40F-	AØ 43	3	LDY	>#43			
#411-	98		TYA				
#412-	8D 73	S BD	STA	#BD73			
#415-	60		RTS				

La mémoire d'écran en mode texte occupe les adresses #BB80-#BFDF (voir annexe)

On peut y inscrire des caractères directement.

Attention aux adresses des deux premières colonnes, réservées aux couleurs du fond et de l'encre.

Jeu de caractères Oric

Remarque: Inutile d'effacer le code excédentaire, n'oubliez pas RTS

#400-	A2 20	LDX >#20	
#402-	20 7C F7	JSR #F77C	;VDU (ORIC-1 #F73F) affiche car. en X
#405-	E8	INX	;X=X+1, caractère suivant
#406-	10 FA	BFL #402	;branche -6 si plus (positif, < #80)
#408-	40	RTS	

Alphabet

#400-	A9 20		LDX	>#20			
#402-	8A		TXA				
#403-	20 16	D2	JSR	#D216	; ISLETC	(ORIC-1	#D186)
#406-	90 03		BCC	#4ØB			
#408-	20 D9	CC	JSR	#CCD9	; OUTDO	(ORIC-1	#CC12)
#4ØB-	E8		INX				
#4ØC-	10 F4		BPL	#402			
#40E-	60		RTS				

Mots-clés

#400-	AØ ØØ	LDY >#00
#402-	BE EA CØ	LDX #CØEA,Y
#405-	20 7C F7	JSR #F77C ;VDU (ORIC-1 #F73F)
#408-	C8	INY
#409-	DØ F7	BNE #402
#4ØB-	60	'RTS

Les mêmes, un peu plus lisibles

```
#400-
        A2 00
                     LDX >#ØØ
                     LDA #C@EA,X
#402-
        BD EA CØ
        20 D9 CC
                     JSR #CCD9 ; OUTDO (ORIC-1 #CC12) affiche car en A
#405-
#408-
        10 03
                     BPL #40D
#40A-
        20 D4 CC
                      JSR #CCD4 ; OUTSPC (ORIC-1 #CCOD) affiche espace
#4ØD-
        E8
                      INX
#4ØE-
                     BNE #402
        DØ FØ
#410-
        60
                     RTS
```

En #40A, si le bit n°7 du caractère est à un (dernier caractère du mot-clé), la routine OUTSPC affiche un espace.

Un peu plus de mots-clés grâce à l'adressage indirect indexé

```
#400-
       A9 EA
                   LDA >#EA
#402- 85 80
                   STA #80
#4Ø4- A9 CØ
                   LDA >#CØ
#406- 85 81
                   STA #81
                   LDX >#02
#408- A2 02
      B1 80
                   LDA (#80),Y
#4ØA-
                              ;OUTDO (ORIC-1 #CC12)
#40C-
       20 D9 CC
                   JSR #CCD9
                   BPL #414
#41F-
      10 03
      20 D4 CC
#411-
                   JSR #CCD4 ;OUTSPC (ORIC-1 #CCØD)
                   INY
#414-
       CB
       DØ F1
                   BNE #40A
#415-
                   INC #81
#417-
       E6 81
#419-
                   DEX
       CA
                   BNE #40A
#41A-
       DØ EC
#41C-
       60
                   RTS
```

Deux pages sont affichées soit 512 caractères. Le registre X est ici compteur de boucle. Il compte les pages. N° de page en #81.

CHAPITRE 8

TESTS, BRANCHEMENTS, BOUCLES

Dans ce chapitre, nous allons passer en revue les instructions que l'on pourrait qualifier de dynamiques, c'est-à-dire celles qui permettent d'infléchir le cours d'un programme, de prendre une décision à un moment donné.

En simplifiant à l'extrême, on peut oser trouver une lointaine analogie avec le Basic.

Les instructions précédentes permettaient l'initialisation des variables.

Le prochain chapitre sera consacré aux instructions de traitement de l'information proprement dit.

Celles que nous verrons maintenant donnent l'équivalent des IF..THEN, FOR-NEXT, GOTO, GOSUB, etc..

INCREMENTATION - DECREMENTATION

Par leurs effets, les instructions qui suivent auraient dû figurer dans le prochain chapitre.

Il convient de les citer ici, compte-tenu de leur importance dans la dynamique d'un programme.

L'incrémentation consiste à augmenter de un le contenu d'un emplacement en mémoire ou d'un registre index.

La décrémentation le diminue de un.

L'accumulateur n'est pas concerné. On ne peut pas incrémenter ou décrémenter directement son contenu.

Etant donné le format 8 bits, il se produit le phénomène cyclique que nous avions imagé avec le compteur kilométrique.

L'incrémentation de la valeur #FF donne #00.

La décrémentation de #00 donne #FF.

La retenue ou l'emprunt sont ignorés.

L'indicateur 'C' n'est pas affecté.

- incrémentation ou décrémentation mémoire

mnėmonique	code	mode d'adressage
INC ADR	EE ff FF	absolu
INC adr	E6 FF	page zéro
INC ADR, X	FE ff FF	absolu,X
INC adr,X	F6 FF	page zéro,X
DEC ADR	CE ff FF	absolu
DEC adr	C6 FF	page zéro
DEC ADR, X	DE ff FF	absolu,X
DEC adr,X	D6 FF	page zéro,X

Indicateurs affectés: Z, N (seulement)

Parmi ces instructions, la plus employée est l'incrémentation du contenu d'un emplacement en mémoire (adressage absolu ou page zéro), dans le cadre de l'incrémentation d'un pointeur d'adresse sur deux octets.

Vous constatez qu'il n'y a pas de règle permettant de déterminer les modes d'adressage disponibles pour une instruction donnée. Ici, l'adressage indexé n'est possible qu'avec X.

- incrémentation ou décrémentation des registres index

mnémonique	code	mode d'adressage
INX	E8	implicite
INY	C8	н
DEX	CA	implicite
DEY	88	u

Indicateurs affectés: Z, N (seulement)

Ces instructions permettent aux registres index de remplir leur fonctions principales, indexation ou compteurs de boucles.

COMPARAISONS

Les instructions de comparaison permettent de détecter une valeur précise et sont souvent employées.

Elles consistent à comparer le contenu de l'accumulateur ou d'un registre index avec une constante ou avec le contenu d'un emplacement en mémoire.

L'opération réalisée est une soustraction (contenu du registre - valeur à comparer) mais elle est virtuelle.

Le résultat n'est pas conservé, les données ne sont pas modifiées.

Une comparaison a pour seul effet de positionner les indicateurs de retenue C, de zéro Z et de signe N. Elle est généralement suivie d'un branchement conditionnel.

Si contenu registre < valeur comparée: C=0 Z=0 N=1 ou Ø Si signe du Si contenu registre = valeur comparée: C=1 Z=1 N=0 Si contenu registre > valeur comparée: C=1 Z=0 N=0 ou 1

- comparaisons avec l'accumulateur

mnémonique	code	mode d'adressage
CMP >donnée	C9 FF	immédiat
CMP ADR	CD ff FF	absolu
CMP adr	C5 FF	page zéro
CMP ADR,X	DD ff FF	absolu,X
CMP ADR,Y	D9 ff FF	absolu,Y
CMP adr,X	D5 FF	page zéro,X
CMP (adr,X)	C1 FF	indexé indirect
CMP (adr),Y	D1 FF	indirect indexé

Indicateurs affectés: C, Z, N

- comparaisons avec les registres index

mnémonique	code	mode d'adressage
CPX >donnée	EØ FF	immédiat
CPX ADR	EC ff FF	absolu
CPX adr	E4 FF	page zéro
CPY >donnée	CØ FF	immédiat
CPY ADR	CC ff FF	absolu
CPY adr	C4 FF	page zéro

Indicateurs affectés: C, Z, N

CPX et CPY sont souvent utilisées en mode immédiat pour détecter une fin de boucle de longueur définie.

Cela peut être, en particulier, la limite d'une indexation assurée par le registre concerné.

BRANCHEMENTS CONDITIONNELS

Les branchements sont basés sur le test d'un des quatre principaux indicateurs, C, Z, V ou N.

Si l'indicateur est dans la position requise le branchement est effectué, sinon le programme continue en séquence.

Adressage relatif

mnémonique	code	signification	indicateur
BCC	90 FF	Branch Carry Clear	C=Ø
BCS	BØ FF	Branch Carry Set	C=1
BNE	DØ FF	Branch Not Equal	Z=Ø
BEQ	FØ FF	Branch EQual	Z=1
BVC	50 FF	Branch oVerflow Clear	V=Ø
BVS	70 FF	Branch oVerflow Set	V=1
BPL	10 FF	Branch on PLus	N=Ø
BMI	30 FF	Branch on MInus	N=I

Indicateurs affectés: aucun (l'indicateur concerné est testé)

Les instructions de branchement sont fondamentales et abondamment utilisées, surtout les quatre premières.

Rappelez vous que le deuxième octet de l'instruction représente le déplacement signé.

Nous avonc déjà parlé des instructions qui forcent à zéro ou à un certains indicateurs. Il s'agit de:

CLC	(Clear Carry)	code #18	
SEC	(SEt Carry)	code #38	
CLV	(CLear oVerflow)	code #B8	(SEV n'existe pas).

Les couples CLC/BCC, SEC/BCS ou CLV/BVC réalisent des branchements forcés, sortes de JMP limités (-128 +127).

Cela peut être intéressant pour assurer la relogeabilité d'un programme en évitant l'adressage absolu interne.

SAUTS, APPEL ET RETOUR DE BOUS-PROGRAMME

Sauf pour le saut indirect, ce sont strictement les équivalents des GOTO, GOSUB et RETURN du Basic, avec la différence que la destination est représentée par une adresse absolue en mémoire au lieu d'un numéro de ligne.

- sauts

mnémonique	code	mode d'adressage
JMP ADR	4C ff FF	absolu
JMP (ADR)	6C ff FF	indirect

Indicateurs affectés: aucun

Le 6502 comporte une "bogue" qu'il faut connaître pour l'utilisation du saut indirect.

Si les pointeurs utilisés sont à cheval sur une frontière de page, exemple: JMP (#6FFF), 6C FF 6F, le saut n'aboutira pas et provoquera probablement un plantage.

L'octet de poids fort n'est pas correctement incrémenté par le 6502. Dans l'exemple ci-dessus, l'octet faible serait lu en #6FFF et l'octet fort en #6F00.

- appel et retour

mnémonique	code	mode d'adressage
JSR ADR	20 ff FF	absolu
RTS	40	implicite

Indicateurs affectés: aucun

Lorsque l'instruction JSR est rencontrée, l'adresse de destination est déposée dans le compteur ordinal PC et l'adresse du JSR + 2 est mise en pile.

L'instruction RTS désempile dans PC et augmente ce dernier de un. L'exécution est pousuivie à partir de l'instruction suivant le JSR d'appel.

INTERRUPTIONS, BRK, NOP

Pour beaucoup d'entre nous, une interruption revêt un sens bien prosaïque. Alors que nous sommes profondément absorbés, aux prises avec notre ordinateur préféré, on nous annonce soudain que la soupe est servie.

Pour préserver la paix familiale il faut bien obtempérer, après avoir pris la précaution de sauvegarder le travail en cours.

interruption matérielle

Le microprocesseur aussi doit obeir à un signal d'interruption, notamment à intervalles réguliers pour lire le clavier.

Dès qu'il a reconnu le signal, acheminé par la ligne IRQ (Interrupt

ReQuest), dite des interruptions masquables, le 6502 termine l'exécution de l'instruction en cours.

Il sauvegarde en pile les contenus de PC et du registre d'état et effectue un saut indirect à l'adresse détenue par le pointeur #FFFE,FFFF, tout en haut de la ROM.

Cette adresse est #244, pour l'ATMOS, où l'on trouve normalement un JMP #EE22 (ORIC-1 #228 -> #ECØ3 -> #EDØ9), saut vers la routine d'interruption qui comprend la scrutation du clavier.

La routine d'interruption se termine par RTI, via un JMP #24A pour l'ATMOS, JUMP #230 pour l'ORIC-1.

Tout cela passe complétement inaperçu pour l'utilisateur.

L'instruction RTI (ReTurn from Interrupt), code opératoire #40, est utilisée pour le retour d'une routine d'interruption de la même façon que RTS pour une routine normale.

Elle remet la machine dans son état initial en restaurant PC et P. L'exécution du programme reprend à l'endroit où il avait été interrompu.

Il faut signaler que l'indicateur I (ne pas déranger) est automatiquement mis à un dès l'acceptation de l'interruption. Il est remis à zéro par RTI.

Rappelons le role des instructions: SEI code #78 inhibition des interruptions (n'empêche pas un RESET) CLI code #58 autorisation des interruptions

Les deux autres vecteurs d'interruption sont:

- #FFFA,FFFB (NMI, Non Maskable Interrupt) qui conduit à RESET en #F8B2, via #247 (ORIC-1 #F430 -> #F8B2, via #22B)
- #FFFC,FFFD Le saut indirect JMP (#FFFC) correspond à JMP #F88F, réinitialisation du système, après une coupure de courant par exemple (ORIC-1 #F42D -> #F84A).
- interruption logicielle, l'instruction BRK

Nous venons de voir que le microprocesseur pouvait être interrompu pour les besoins du système. Il peut l'être aussi par l'instruction BRK, code opératoire #00.

Le signal d'interruption par Break est acheminé par la même ligne IRQ que le signal émis par la machine.

La procédure suivie par le microprocesseur est strictement la même dans les deux cas.

Vous vous souvenez que lorsque l'instruction BRK est rencontrée, l'indicateur B, bit n°4 du registre d'état, est mis à un. Il est nécessaire que la routine d'interruption teste cet indicateur pour déterminer la nature (matérielle ou logicielle) de l'interruption.

Ce n'est pas le cas sur Oric.

L'instruction BRK n'est pas directement utilisable en langage machine. Comme STOP en Basic, cette instruction pourrait être utile pour la mise au point des programmes. En guise d'exercice, vous essaierez un court programme de dérivation de la routine d'interruption qui teste l'indicateur B.

- l'instruction NOP

Cette instruction, code opératoire #EA, signifie No Opération. Elle ne fait donc rien. Le 6502 passe à l'instruction suivante. Elle est cependant très utile pour remplacer du code machine indésirable. Ainsi, le code valide n'a pas à être déplacé.

On peut l'utiliser pour introduire de courtes temporisations car le microprocesseur prend quand même un certain temps pour l'"exécuter" (2 cycles).

EXERCICES

Interruption par BRK

Entrez le code machine à partir de l'adresse #450.

#450-	85 80	STA	#80		
#452-	68	PLA			
#453-	48	PHA			
#454-	29 10	AND	>#10		
#456-	FØ Ø8	BEQ	#460		
#458-	20 26 C	7 JSR	#C726	; (ORIC-1	#C751)
#45B-	38	SEC			
#45C-	58	CLI			
#45D-	4C 8A C	7 JMP	#C98A	; (ORIC-1	#C958}
#460-	A5 80	LDA	#8Ø		
#462-	4C 22 E	JMP	#EE22	; (ORIC-1	#ECØ3)

Ne lancez pas ce programme.

Entrez en commande directe:

Pour ATMOS DOKE#245,#450

Pour ORIC-1 DOKE#229,#450

Pour les deux POKE#400,0 puis

CALL#400

Vous devriez obtenir l'affichage du mot BREAK

Ce programme détecte l'interruption logicielle par BRK.

La première instruction sauvegarde le contenu de l'accumulateur dans un emplacement temporaire.

PLA retire le premier octet de la pile qui est le contenu du registre d'état empilé au déclenchement de l'interruption.

FHA le remet en place. il faut que la pile soit intacte pour le RTI de l'interruption matérielle.

AND >#10 teste le bit n°4, indicateur de break B. S'il est à zèro on obtient zèro dans l'accumulateur. Nous étudierons l'instruction AND au prochain chapitre, encore un retour en arrière à prévoir!

Si le bit n°4 est à zéro, il s'agissait d'une interruption ordonnée par le système.

Branchement en #460 où le contenu initial de l'accumulateur est récupéré puis saut à la routine normale d'interruption.

Si le bit n°4, indicateur B, est à un. Nous avons affaire à une interruption logicielle. Il faut arrêter le programme et revenir au Basic.

JSR #C726 (ORIC-1 #C751), STKINI, réinitialise la pile.

SEC pour l'affichage de BREAK, simule le STOP du Basic.

CLI est nécessaire pour autoriser à nouveau les interruptions, l'indicateur I ayant été mis automatiquement à un. Le microprocesseur sera prêt à accepter la prochaine interruption (matérielle) qui remettra à zéro l'indicateur B.

JMP #C98A (ORIC-1 #C958),PREND , fin de programme, rentrée Basic, via affichage BREAK.

Le message proposé ici est rudimentaire. Généralement, on fait apparaître le contenu des registres, renseignement utile pour la mise au point d'un programme.

Recherche d'un mot-clé.

Ce programme reproduit, à peu près, une phase de la routine LIST, lorsqu'ayant identifié le code d'un mot-clé, >#80, elle le recherche dans la table pour afficher le mot en clair à l'écran dans la ligne de programme.

Il s'agit du décodage, opération que nous avons déjà évoquée. Pour retrouver ce passage, désassemblez la ROM, à partir de #C7DØ pour l'ATMOS, de #C7F7 pour l'ORIC-i.

Ne vous préoccupez pas des premières lignes pour l'instant. Essayez d'analyser ce qui se passe dans la phase de recherche et d'affichage, à partir du moment où le code du mot-clé se trouve dans l'accumulateur (#40D).

Appelez le programme par CALL#400, ou si vous préférez, faites DOKE#2F5,#400 et appelez-le avec la commande !

A la demande du point d'interrogation (INPUT) entrez le code d'un mot-clé, compris entre #80 et #FF ou, en décimal, entre 128 et 255.

La sortie du programme se fait par l'entrée d'une valeur inférieure à #80 (128 décimal) ou sur message d'erreur.

```
#400- 20 F0 CB
                   JSR #CBFØ
                                ;CR+LF,(ORIC-1 #CB9F)ret.à la lione
#403-
      20 80 CD
                   JSR #CD80
                                ;OUTQST,OUTSPC->INLIN,(ORIC-1 #CCF4)
#406-
      86 E9
                   STX #E9
                                ;(TXTPTR) = ...
       84 EA
#408-
                   STY #EA
                                              BUFF - 1 (#0034)
#4ØA-
       20 C5 D8
                   JSR #D8C5
                                ;GETBYTC, (ORIC-1 #D80A), valeur en X
#40D-
       8A
                   TXA
                                transférée dans accumulateur
#4ØE-
                   BPL #43B
       10 2B
                                ;si valeur < #80 retour Basic
#410-
       38
                   SEC
                                ;code mot-clé - #7F = position
                                ;dans la table, ex EDIT 2° position.
#411-
      E9 7F
                   SBC >#7F
#413-
       AA
                   TAX
                                ;X compteur
#414-
       AØ ØØ
                  LDY >#00
                                :post-indexation par Y
                  LDA >#E9
#416-
       A9 E9
                                ;adresse table - i
#418-
       85 18
                   STA #18
                                      #CØE9
                                ;
#41A-
                  LDA >#CØ
       A9 CØ
                                      dans
#41C-
       85 19
                  STA #19
                                      #18,19
                                :
#41E-
       CA
                  DEX
                                ;si (X) = Ø ....
#41F-
       FØ ØC
                  BEQ #42D
                                ; .. mot-clé trouvé, affichage
#421-
      E6 18
                   INC #18
                                ;boucle ...
                  BNE #427
#423-
       DØ Ø2
                                      de ..
#425-
       E6 19
                   INC #19
                                      recherche ..
#427-
      B1 18
                  LDA (#18),Y
                                      dernier....
                                ;
#429-
       10 F6
                  BPL #421
                                ;caractère, ASCII négatif
#42B-
       30 F1
                  BMI #41E
                                ;trouvé, mot suivant.
#42D-
       C8
                  INY
                                ;boucle ...
#42E-
                  LDA (#18),Y
       B1 18
                                      d'affichage
                                ;
#430-
       08
                  PHP
                                      du
                                ş
       29 7F
                  AND #7F
#431-
                                      mot-clé
       20 D9 CC
                  JSR #CCD9
#433-
                                :OUTDO, (ORIC-1 #CC12), affichage
#436-
       28
                  PLP
                                ;dernier caractère?
#437-
       10 F4
                  BPL #42D
                                inon, caractère suivant
#439-
       30 C5
                  BMI #400
                                ;oui, affichage terminé, on recommence
#43B-
       60
                  RTS
                                retour Basic
```

Remarque: Il semble que l'instruction AND >#7F en #431 (et donc PHP et PLP), qui remet à zéro le bit 7 du dernier caractère du mot-clé, soit superflue, compte-tenu des caractèristiques de la routine d'affichage à l'écran.
Elle figure effectivement dans la routine LIST (ATMOS #C7A6, ORIC-1 C7CD)

CHAPITRE 9

TRAITEMENT DE DONNÉES

Les instructions spécialisées dans le traitement de l'information sont les instructions arithmétiques, les instructions logiques, les rotations et les décalages.

Les opérations arithmétiques se réduisent à la simple addition ou soustraction de nombres entiers sur 8 bits.

Multiplications, divisions, opérations sur des nombres fractionnaires doivent être programmées.

Nous verrons que la plupart des routines arithmétiques en ROM sont utilisables.

Les opérations logiques, AND, ORA ou EOR, sont effectuées bit à bit (excusez-moi!) entre le contenu de l'accumulateur et une valeur spécifiée.

Notez que le premier terme de toutes ces opérations est toujours l'accumulateur. Il est implicitement désigné.

Nous avons déjà rencontré cette caractèristique avec CMP. Les modes d'adressage applicables sont les mêmes que pour cette instruction.

Par exemple:

ADC >donnée signifie accumulateur + donnée (+ C)

AND >donnée signifie accumulateur AND donnée

Les rotations et décalages agissent directement sur le contenu d'un octet en déplaçant les 8 bits d'un rang dans un sens ou dans l'autre. Ces instructions se différentient par la façon dont sont pris en compte le bit entrant, le bit sortant et l'indicateur de retenue.

INSTRUCTIONS ARITHMETIQUES

- addition

mnémonique	code	mode d'adressage
ADC >donnée	69 FF	immédiat
ADC ADR	6D ff FF	absolu
ADC adr	65 FF	page zéro
ADC ADR,X	7D ff FF	absolu,X
ADC ADR,Y	79 ff FF	absolu,Y
ADC adr, X	75 FF	page zéro,X
ADC (adr, X)	61 FF	indexé indirect
ADC (adr),Y	71 FF	indirect indexé

Indicateurs affectés: C, Z, V, N

ADC signifie ADd with Carry (additionner avec retenue).

La valeur spécifiée est ajoutée au contenu de l'accumulateur ainsi que la valeur actuelle de l'indicateur de retenue C.

Le résultat est conservé dans l'accumulateur.

- soustraction

Le 6502 emploie la méthode du complément à deux. Il ajoute au premier nombre le complément à deux du nombre à soustraire. A-B -> A+(-B). L'emprunt éventuel est signalé par l'indicateur C dont l'interprétation est inversée par rapport à celle de la retenue: Si C=1, A > B, pas d'emprunt. Si C=0, A < B, emprunt.

Exemples:

Soit l'opération #5F-#28, le complément à deux de #28 est #D8.

	#5F		01011111
+	#D8	+	11011000
	#37	(1)	00110111

Soit l'opération #2E-#AA, le complément à deux de #AA est #56

	#2E	00101110
+	#56	+ 01010110
	#84	(0) 10000100

Les instructions de soustraction sont les suivantes:

mnémonique	code	mode d'adresssage
SBC >donnée SBC ADR SBC adr SBC ADR,X SBC ADR,Y SBC adr,X SBC (adr,X) SBC (adr),Y	E9 FF ED ff FF E5 FF FD ff FF F9 ff FF F5 FF E1 FF	immédiat absolu page zéro absolu,X absolu,Y page zéro,X indexé indirect indirect indexé

Indicateurs affectés: C, Z, V, N.

L'opération SBC (SuBtract with Carry) soustrait de l'accumulateur la valeur spécifiée ainsi que le complément de l'indicateur C (1-C), c'est-à-dire 1 si C=0 ou 0 si C=1. Le résultat est conservé dans l'accumulateur.

ADC et SBC sont les seules opérations arithmétiques du 6502. La prise en compte de l'indicateur de retenue simplifie les opérations sur les nombres multi-octets.

La valeur de C intervient automatiquement dans l'addition ou la soustraction des octets de poids successif.

Cette particularité impose deux règles importantes:

Avant une simple addition sur 8 bits ou, s'agissant de nombres sur plusieurs octets, avant l'addition des octets de poids faible, l'indicateur de retenue doit être mis à zéro par l'instruction CLC. Avant une soustraction sur 8 bits ou avant la soustraction des octets de poids faible dans le cas de nombres sur plusieurs octets, l'indicateur C doit être mis à un par l'instruction SEC.

- mode arithmétique décimal, BCD, Binary Coded Decimal.

Les mêmes opérateurs sont employés en mode décimal codé binaire (BCD), utile pour certaines applications.

Dans ce mode, les demi-octets (quartets ou nibbles) de 0000 à 1001 inclus (les autres ne sont pas utilisés) représentent les chiffres décimaux de 0 à 9.

Sur 8 bits, en BCD, les valeurs binaires sont comprises entre 00000000 et 10011001 et représentent les nombres décimaux de 0 à 99.

Le programmeur doit indiquer au microprocesseur dans quel mode doivent opérer ADC ou SBC, à l'aide des instructions: SED (SEt Decimal mode), mise à un de l'indicateur D, mode décimal. CLD (CLear Decimal mode) mise à zéro de l'indicateur D, mode binaire normal.

- débordement.

Pour le microprocesseur tous les nombres sont en valeur absolue. Lorsque, par convention, ils sont considérés comme signés (représentation en complément à deux) la capacité 'utile' de l'octet n'est plus que de 7 bits, le bit n°7 étant le bit de signe.

L'indicateur de débordement V, mis à un après une opération, signale le changement accidentel du signe du résultat en arithmétique binaire signée

Exemples:

$$60 + 112 = -84$$

#9D - #D0 = #CD C=0 V=0 pas de débordement -99 -(-48) = -51 correct.

L'indicateur de débordement est forcé à zéro par l'instruction CLV.

INSTRUCTIONS LOGIQUES

les opérateurs logiques ou booléens agissent sur des variables susceptibles de prendre exclusivement une de deux valeurs possibles notées Ø et 1. AND, ET logique, forme le produit logique de deux variables booléennes.

Le résultat n'est égal à un que si les termes sont tous deux égaux à un, conformément à la table de vérité:

Ø AND Ø = Ø Ø AND 1 = Ø 1 AND Ø = Ø 1 AND 1 = 1

OR, OU logique, forme la somme logique, égale à un si l'un des termes est égal à un ou si les deux sont égaux à un:

0 OR 0 = 0 0 OR 1 = 1 1 OR 0 = 1 1 OR 1 = 1

XOR, OU exclusif, différe du précédent sur un point. Le résultat n' égal à un que si l'un des termes seulement est égal à un:

> Ø XOR Ø = Ø Ø XOR 1 = 1 1 XOR Ø = 1 1 XOR 1 = Ø

Le 6502 met en oeuvre ces opérateurs, respectivement, avec les instructions logiques AND, ORA, et EOR.

Les opérations sont effectuées entre les 8 bits de l'accumulateur et les bits correspondant d'un octet spécifié.

Le résultat est conservé dans l'accumulateur.

- l'instruction AND

code	mode d'adressage
29 FF	immédiat
2D ff FF	absolu
25 FF	page zéro
3D ff FF	absolu,X
39 ff FF	absolu, Y
35 FF	page zéro,X
21 FF	indexé indirect
31 FF	indirect indexé
	29 FF 2D ff FF 25 FF 3D ff FF 39 ff FF 35 FF 21 FF

Indicateurs affectés: Z, N.

L'instruction AND est principalement utilisée pour le masquage. Cette opération consiste à annuler sélectivement certains bits d'un octet.

Le masque est créé en mettant à un chaque bit dont on veut conserver la valeur initiale et à zéro le ou les autres.

Par exemple, le code du caractère déposé dans la mémoire clavier (#2DF) a le bit 7 à un.

Le code est chargé dans l'accumulateur (LDA #2DF) puis le bit 7 est mis à zéro par AND #7F.

- l'instruction ORA

mnémonique	code	mode d'adressage
ORA >donnée	09 FF	immédiat
ORA ADR	ØD ff FF	absolu
ORA adr	05 FF	page zéro
ORA ADR.X	1D ff FF	absolu,X
ORA ADR,Y	19 ff FF	absolu.Y
ORA adr.X	15 FF	page zéro,X
ORA (adr.X)	Ø1 FF	indexé indirect
ORA (adr),Y	11 FF	indirect indexé

Indicateurs affectés: Z. N.

Le procédé du masquage peut être appliqué avec ORA. A l'inverse du masquage par AND, il permet de forcer certains bits à un.

- l'instruction EOR

mnémonique	code	mode d'adressage
EOR >donnée	49 FF	immédiat
EOR ADR	4D ff FF	absolu
EOR adr	45 FF	page zéro
EOR ADR.X	5D ff FF	absolu,X
EOR ADR,Y	59 ff FF	absolu,Y
EOR adr,X	55 FF	page zéro,X
EOR (adr, X)	41 FF	indexé indirect
EDR (adr),Y	51 FF	indirect indexé.

Indicateurs affectés: Z, N.

L'instruction EOR est surtout utilisée pour inverser certains bits d'un octet. Elle peut produire un phénomène de bascule.
Ainsi l'instruction EOR #FF inverse tous les bits de l'accumulateur.
Appliquée une nouvelle fois, elle lui redonne sa valeur initiale.
Un exemple concret d'utilisation de EOR:
L'emplacement #20C contient #FF en mode majuscules, #7F en mode minuscules. L'action sur CTRL-T provoque essentiellement la suite d'instructions suivante: LDA #20C, EOR #80, STA #20C.
Si le contenu de #20C était #FF, il passera à #7F et inversement.

- l'instruction BIT

Cette instruction est apparentée à l'instruction AND. Elle effectue un ET logique entre le contenu de l'accumulateur et une valeur en mémoire.

C'est pourquoi elle figure dans ce chapitre bien qu'elle ne modifie pas le contenu de l'accumulateur. L'opération est virtuelle. Comme les comparaisons, il s'agit d'une instruction de test dont le seul effet est de positionner certains indicateurs d'état.

Les bits 6 et 7 de l'emplacement en mémoire sont transférés directement dans les bits correspondants du registre d'état, c'est-à-dire les indicateurs V et N.
L'indicateur Z est mis à un si le résultat du AND est zéro, il est mis à zéro dans le cas contraire.

mnémonique	code	mode d'adressage
BIT ADR	2C ff FF	absolu
BIT adr	24 FF	page zéro

Indicateurs affectés: Z, V (m6),N (m7)

L'instruction BIT offre un moyen commode de tester les bits 6 ou 7 d'un emplacement en mémoire.

Utilisée dans ce but, elle sera suivie de l'instruction de branchement appropriée (BPL, BMI, BVC ou BVS)

Pour le test des bits 0 à 7 du contenu d'un emplacement en mémoire ou de l'accumulateur, on se servira d'un masque dont tous les bits seront à zéro sauf celui à tester.

Le masque sera déposé dans l'accumulateur si l'on teste un bit d'un emplacement mémoire et vice-versa.

L'indicateur I sera mis à un si le résultat est zéro, c'est-à-dire si le bit testé vaut zéro. Il sera mis à zéro si le bit testé est à un.

DECALAGES

Ces opérations décalent d'un rang à gauche ou à droite les huit bits de l'accumulateur ou d'un emplacement en mémoire spécifié. Le bit sortant va dans l'indicateur de retenue. Un zéro prend la place laissée vacante.

- décalage arithmétique à gauche ASL (Arithmetic Shift Left)

Tous les bits de l'accumulateur ou d'un emplacement en mémoire sont décalés à gauche par cette instruction. Le bit n°0 est mis à zéro et le bit n°7 passe dans l'indicateur de retenue.

mnémonique	code	mode d'adressage
ASL A	ØA	accumulateur
ASL ADR	ØE ff FF	absolu
ASL adr	06 FF	page zéro
ASL ADR,X	1E ff FF	absolu,X
ASL adr, X	16 FF	page zéro,X

Indicateurs affectés: C, Z, N

L'opération ASL multiplie par deux la valeur initiale de l'octet. Cette instruction peut aussi être employée pour déterminer la valeur d'un bit donné.

Par un nombre de décalages approprié, ce bit est transféré dans l'indicateur de retenue qui est testé.

Dans le plupart des cas, les instructions AND ou BIT conviendront mieux pour ce test.

- décalage logique à droite LSR (Logical Shift Right)

Cette instruction décale d'une position vers la droite chaque bit de l'accumulateur ou d'un emplacement en mémoire spécifié. Le bit n°7 est forcé à zéro et le bit n°0 passe dans l'indicateur de retenue.

mnémonique	code	mode d'adressage
LSR A	4A	accumulateur
LSR ADR	4E ff FF	absolu
LSR adr	46 FF	page zéro
LSR ADR,X	SE ff FF	absolu,X
LSR adr, X	56 FF	page zéro,X

Indicateurs affectés: C, Z, N (mis à zéro)

Cette instruction permet de diviser par deux la valeur concernée (valeur absolue).

Elle est souvent utilisée pour mettre à zéro le bit n°7 d'un emplacement en mémoire utilisé comme drapeau.

ROTATIONS

Ces instructions décalent aussi d'une position chaque bit de l'accumulateur ou d'un emplacement en mémoire mais les opérations se font en circuit fermé sur 9 bits incluant l'indicateur de retenue.

- rotation à gauche ROL

Tous les bits de l'octet sont déplacés d'une position vers la gauche. L'indicateur de retenue C remplace le bit n°0. Le bit n°7 passe dans l'indicateur de retenue.

mnémonique	code	mode d'adressage
ROL A ROL ADR	2A 2E ff FF	accumulateur absolu
ROL adr	26 FF	page zéro
ROL ADR,X ROL adr,X	3E ff FF 36 FF	absolu,X page zéro,X

Indicateurs affectés: C, Z, N.

Les bits sont décalés à droite. L'indicateur de retenue passe dans le bit n°7. Il prend la valeur du bit n°0 sortant.

mnémonique	code	mode d'adressage
ROR A	6A	accumulateur
ROR ADR	6E ff FF	absolu
ROR adr	66 FF	page zéro
ROR ADR,X	7E ff FF	absolu,X
ROR adr, X	76 FF	page zéro,X

Indicateurs affectés: C, Z, N.

les instructions ROL et ROR sont le plus souvent utilisées dans les routines de multiplication ou de division.

Ainsi, en combinaison avec ASL, ROL permet le décalage à gauche d'un nombre de 16 bits. ASL octet faible, ROL octet fort (multiplication par deux).

⁻ rotation à droite ROR

EXERCICES

Les exercices proposés jusqu'ici n'étaient pas très pédagogiques. Ceux-ci ne le seront pas davantage, je le crains. Essayez-les quand même. Ils vous feront progresser dans la connaissance de votre ordinateur. Surtout, ne vous découragez pas devant la complexité apparente. Encore une fois, ce qui parait compliqué aujourd'hui deviendra plus clair demain.

Conversion hexadécimal -> décimal et décimal -> hexadécimal.

Après avoir entré ce programme, faites DOKE#2F5,#400 Appelez le avec la commande ! suivie du nombre à convertir (0 à 65535 ou #0 à #FFFF).

Exemple: !#7530 ou !64218

```
#400- 20 E8 00
                 JSR #E8
                               ; CHARGOT, recueille caractère à TXTPTR
#403- C9 23
                 CMP >#23
                               ;caractère '#' ?
#405- DØ 07
                 BNE #40E
                               ;non, conversion déc. -> hexa
#407- 20 E7 DF
                 JSR #DFE7
                               ;FIN, (ORIC-1 #DFCF) Nb & TXTPTR -> FAC
#40A- 20 CF E0
                 JSR #EØCF
                               ;PRNTFAC, (ORIC-1 #EØCB) affiche FAC
#40D- 60
                 RTS
                               :retour Basic
#40E- 20 53 E8
                 JSR #E853
                               :FRMNUM,GETADR, (ORIC-1 #E7AD)
#411- A2 ØØ
                 LDX >#00
                                   nombre en LINNUM(#33,34)
#413- 86 2F
                 STX #2F
                               ;compteur
#415- A9 23
                 LDA >#23
                               ;caractère '#'
#417- 85 FF
                 STA #FF
                               ;stocké en #FF
#419- A5 34
                 LDA #34
                               :octet fort
#41B- 20 93 D9
                 JSR #D993
                               ;conversion hexa (ORIC-1 #D8F5)
#41E- A5 33
                 LDA #33
                               :octet faible
#420- 20 93 D9
                 JSR #D993
                               ;conversion hexa
#423- A9 ØØ
                 LDA >#00
                               ; valeur 00
#425- 9D 00 01
                 STA #100, X
                               ;en fin de chaîne nombre hexa
#428- A8
                 TAY
                               ;Y=0 octet fort adresse chaîne
#429- A9 FF
                 LDA >#FF
                               ; A=#FF octet faible "
#42B- 20 B0 CC
                 JSR #CCBØ
                               ;STROUT, (ORIC-1 #CBED) affichage nombre
#42E- 60
                 RTS
                               :retour Basic
```

Remarque: On pourrait remplacer JSR par JMP en #40A et #42B et supprimer les RTS qui suivent. Le RTS des sous-routines appelées assurerait le retour au Basic.

Laissez ce programme en mémoire, il servira de sous programme au suivant.

Codage des mots-clés

Ce programme est inspiré de PARSE, routine importante en #C5FA (ORIC-1 #C6ØA)

qui sera commentée dans la troisième partie.

Il reconnait un mot-clé dans le buffer d'entrée en le comparant à ceux de la table en #CØEA et affiche le code correspondant en hexadécimal. Il vérifie même la syntaxe, ce que ne fait pas l'ORIC.

L'opération étudiée au chapitre précédent (décodage avec LIST) est l'inverse de celle-ci.

C'est un bon exemple de recherche dans une table à l'aide de l'adressage indirect indexé.

Vous remarquerez que le contenu du registre index Y reste à zéro. Seul le pointeur (#18,19) est incrémenté. A ce propos, notez l'emploi de BNE qui détecte le franchissement d'une frontière de page pour incrémentation éventuelle de l'octet fort.

Vous appellerez le programme par CALL#430, ou avec la commande ! après un DOKE#2F5,#430. A sa demande, vous entrerez l'un quelconque des mots-clès du Basic, y compris les opérateurs arihmétiques.

Pour en sortir, appuyez simplement sur RETURN.

```
#430- 20 FØ CB
                 JSR #CBF0 ;CR+LF (ORIC-1 #CB9F)ret à la ligne
#433- 20 80 CD
                 JSR #CD80 ;OUTQST,OUTSPC,->INLIN,(ORIC-1 #CCF4)
#436- A2 35
                 LDX >#35 ;X=#35, indexera buffer d'entrée
                 LDY >#00
                            :post indexation par Y
#438- AØ ØØ
#43A- 84 26
                 STY #26
                             ;compteur pour déterminer code mot-clé
#43C- A9 E9
                 LDA >#E9
                             ;adresse ..
#43E- 85 18
                 STA #18
                                   table - 1...
                             3
#440- A9 C0
                 LDA >#CØ
                                   dans pointeur ..
#442- 85 19
                 STA #19
                             ;#18,19
#444- CA
                DEX
                             ;recalage index
                             ;caractère dans
#445- E8
                INX
                INC #18
#446- E6 18
                                 buffer
                             •
                 BNE #44C
#448- DØ Ø2
                             ;
                                   et
#44A- E6 19
                 INC #19
                                   dans
                            *
#44C- B5 00
                 LDA #00,X ;
                                   la table
#44E- 38
                 SEC
                                   sont
                 SBC (#18),Y ;comparés
#44F- F1 18
                 BEQ #445 ;c'était carac mot-clé, carac.suivant
#451- FØ F2
                            ;mot-clé complet?
#453- C9 8Ø
                CMP >#80
                 BNE #465
                            ;non, mot suivant dans la table
#455- DØ ØE
#457- Ø5 26
                            ;oui, #80 + contenu #26 = mot-clé
                 ORA #26
#459- 85 33
                 STA #33
                            ;rangé ...
#45B- A9 00
                LDA >#00
                                   dans
                            ,
#45D- 85 34
                 STA #34
                                   #33,34 (LINNUM) pour
                             •
#45F- 20 11 04 JSR #411
                             ;conversion en héxadécimal et affichage
                 JMP #430
#462- 40 30 04
                             ;nouvelle entrée.
#465- E6 26
                 INC #26
                             repositionne
#467- Bi 18
                 LDA (#18),Y; pointeur..
#469- Ø8
                 PHP
                                  sur..
#46A- E6 18
                 INC #18
                                  premier..
                            ;
#46C- DØ Ø2
                 BNE #47Ø
                                   caractère..
                            ;
#46E- E6 19
                 INC #19
                                   du..
                             ;
#470- 28
                 PLP
                                   mot suivant..
                 BPL #467 ;dans la table
LDA (#18),Y ;fin de table?
#471- 10 F4
#473- B1 18
               BEQ #47B ; oui, mot proposé n'existe pas
LDX >#35 ; non, on recommence la comparaison
BNE #44C ; avec mot suivant (branch.forcé)
#475- FØ Ø4
#477- A2 35
#479- DØ D1
#47B- 4C 70 D0 JMP #D070 ;?SYNTAX ERROR (ORIC-1 #CFE4)
```

.

CHAPITRE 10

BASIC ET CODE MACHINE

Ce livre peut vous aider à aborder en douceur le langage machine. Il n'a pas la prétention de faire de vous un expert dans cette discipline.

Pour cela, une longue pratique est nécessaire.

Son but essentiel est de favoriser une utilisation optimale du langage Basic de votre ordinateur.

C'est dans un environnement Basic que vous serez amenés, le plus souvent, à vous servir des quelques octets de code machine nécessaires.

Nous allons examiner dans ce chapitre les problèmes de communication entre les deux langages et les solutions possibles.

MISE EN PLACE DU CODE MACHINE

- Implantation du code par POKE ou DOKE

C'est la méthode la plus simple, à utiliser pour les sous-routines courtes, 10 à 20 octets maximum.

Le programme inscrit le code dans des zones préférentielles où il n'y a pas de risque d'interférence. Page quatre en général, ou même page zéro (buffer d'entrée, par exemple).

L'ORIC permet de charger deux emplacements à la fois avec DOKE qui, autre avantage, accepte aussi l'hexadécimal.

Il faut se rappeler que les octets seront inversés en mémoire. Ainsi DOKE#400,#FF00 inscrira la valeur #00 en #400 et #FF en #401.

Nous allons faire tourner une courte routine témoin, empruntée au système.

Il l'utilise pour afficher le titre à l'initialisation. Elle sera implantée en page zéro, dans le buffer d'entrée.

ATMOS		ORIC-1	
A9 96	LDA >#96	A9 51 LDA >#51	;adresse message
AØ ED	LDY >#ED	AØ EB LDA >#EB	;en A(-) et Y(+)
20 B0 CC	JSR #CCBØ	20 ED CB JSR #CBED	;pour STROUT
60	RTS	60 RTS	;retour à l'appelant.

NEW

Pour ATMOS

10 DOKE#60, #96A9: DOKE#62, #EDA0: DOKE#64, #B020: DOKE#66, #60CC

Pour ORIC-1

10 DDKE#60,#51A9:DDKE#62,#EBA0:DDKE#64,#ED20:DDKE#66,#60CB

Pour l'un ou l'autre 20 CALL#60 30 END

RUN

- Code machine en DATA

Le code rangé en DATA est lu par READ et inscrit aux emplacements spécifiés dans une boucle FOR-NEXT. Cette méthode est pratique. Elle est valable pour un nombre d'octets allant jusqu'à une ou deux pages.

NEW

Pour ATMOS 10 DATA #A9,#96,#A0,#ED,#20,#B0,#CC,#60

Pour ORIC-1 10 DATA #A9,#51,#A0,#EB,#20,#ED,#CB,#60

Pour l'un ou l'autre 20 FOR = #60 TO #67:READ C:POKE A,C:NEXT 30 CALL#60 40 END

RUN

Comme la précédente, cette méthode présente l'inconvénient de placer le code machine à l'extérieur du programme Basic. Les emplacements convenables sont relativement peu nombreux. De plus, les lignes DATA et la boucle ayant servi à l'implantation deviennent inutiles après avoir joué leur rôle.

Une solution intéressante consiste à inscrire le code aux emplacements occupés par les lignes DATA elles-mêmes. En tous cas, c'est un excellent exercice de style que nous allons étudier.

Cette technique est possible car le pointeur DATPTR est automatiquement incrémenté lorsqu'une donnée a été lue par READ. Tout ce qui précéde peut être écrasé sans inconvénient. Le code, étant plus compact que la ligne DATA, ne risque pas de 'rattraper' la boucle FOR-NEXT et d'empêcher son fonctionnement.

Conservez la ligne 10 et ajoutez les lignes suivantes:

20 FOR A≃#500 TO #507:READ C:POKE A,C:NEXT 30 DOKE#9A,DEEK(DEEK(#AC)+1) 40 CALL#500 50 END

RUN, puis LIST

Vous aurez reconnu, à la ligne 30, TXTTAB et surtout OLDTXT dont je vous disais le plus grand bien au chapitre 5. Retournez-y si vous avez besoin de vous rafraîchir la mémoire.

Avantage de cette technique: si les lignes DATA sont placées, comme nous l'avons fait, en début de programme le code sera à son emplacement définitif. Il peut être non relogeable, l'adresse d'appel est fixe.

Attention: TXTTAB est modifié. NEW ne suffit pas pour lui redonner sa valeur initiale. Vous trouverez, à la fin de ce chapitre, un programme qui convertit du code machine en mémoire sous forme de lignes de DATA directement utilisables en Basic.

Une alternative à l'emploi des DATA consiste à mettre le code formatté (pourvu de tous ses zéros) dans une chaîne de caractères. La chaîne est découpée pour l'implantation en mémoire. La ligne est plus compacte mais moins lisible. Vous en avez un exemple dans le programme utilitaire, ligne 1010.

- La méthode transparente

Elle est ainsi appelée parce que le code machine, pourtant solidaire du programme Basic, n'apparaît pas au listage. Il est rangé entre le texte Basic et la zone des variables, généralement accolé après les trois zéros qui marquent la fin du programme proprement dit, donc à partir de VARTAB. Ce pointeur doit être modifié de façon à ce que les variables s'inscrivent après le code machine.

L'exemple utilisé sera simple mais aussi démonstratif que possible.

Débranchez puis rebranchez votre ordinateur ou, CALL#CØØØ.

Entrez le programme Basic:

100 CALL DEEK(#9C)-8

Cherchons VARTAB: ?HEX\$(DEEK(#9C))

Si la ligne 100 a été tapée comme ci-dessus, ce devrait être #512.

Nous allons installer la routine témoin, 8 octets. On aurait pu commencer par là et l'inscrire à partir de #503, mais il vaut mieux suivre la procédure normale

Faisons-lui de la place, VARTAB = VARTAB + 8

DOKE#9C, DEEK (#9C) +8

Cela devrait donner #51A, vérifions: ?HEX\$(DEEK(#9C))

Entrons le code machine

Pour ATMOS

DOKE#512,#96A9 DOKE#514,#EDAØ DOKE#516,#BØ2Ø DOKE#518,#6ØCC

Pour ORIC-1

DOKE#512,#51A9 DOKE#514,#EBAØ DOKE#516,#ED2Ø DOKE#518,#6ØCB

RUN

Vous pouvez modifier ce programme à votre guise.

Ajoutez des lignes, puis supprimez-les. Conservez la ligne 100, pour continuer à tester le fonctionnement du sous-programme qui se déplace selon les modifications apportées au texte Basic.

Par exemple, ajoutez les lignes:

10 FOR I=1 TO 5:CLS:WAIT 50:GOSUB 100:ZAP:WAIT 50:NEXT:END

110 RETURN

Avantages de la méthode:

Le code machine est invisible.

Il occupe une place minimum en mémoire.

Il fait partie intégrante du programme Basic et sera, avec lui, sauvegardé sur cassette de façon normale, comme dans les méthodes précédentes

Après chargement en mémoire du programme Basic, le code machine est immédiatement opérationnel. Pas de longue boucle FOR-NEXT pour le mettre en place.

Inconvénients:

Le défaut majeur de cette méthode, à moins d'être absolument certain que le programme Basic ne sera pas modifié, est qu'il impose du code relogeable, sans adressage absolu interne.

Le code machine doit être implanté par un moyen extérieur au programme Basic, soit par POKE en commande directe, soit à l'aide d'un moniteur-assembleur, par exemple.

Notre utilitaire en haut de mémoire pourrait convenir moyennant une manipulation des pointeurs TXTTAB et VARTAB.

L'application la plus courante est la suivante:
Tout le code machine est accolé au programme Basic.
Le code 'utile' est précédé d'une routine de transfert qui, dès le chargement terminé, le recopie à son emplacement définitif, généralement en haut de mémoire où il sera en sécurité grâce à une modification de HIMEM.

APPELS Transmission de parametres

Il existe différents moyens de communication entre un programme Basic et un sous-programme en langage machine.

Pour certaines routines un simple appel suffit.

D'autres ont besoin de paramètres qui peuvent varier et qu'on leur passera au moment de l'appel.

Notez la similitude avec le Basic pur où l'on affecte une valeur à une variable avant un GOSUB.

L'adaptation est nécessaire entre le programme appelant et le sous-programme. Le principe est valable quel que soit le langage. En particulier, il est important de connaître le fonctionnement des routines en ROM pour les utiliser correctement.

- L'instruction CALL

Le simple CALL permet d'appeler un sous-programme en langage machine.

Des paramètres seront éventuellement passés par l'intermédiaire d'emplacements en mémoire.

86

Vous en avez un exemple à la fin de ce chapitre. Les numéros de ligne à supprimer sont inscrits en page zéro où le sous-programme les recueille.

Un CALL amélioré permet de passer une donnée 8 bits récupérée par GETBYTC. (page 18)

Une extension de cette technique donne la possibilité de charger les trois principaux registres du 6502, A, X et Y à partir du Basic et d'actionner une routine quelconque (en ROM ou RAM).

Chargez votre utilitaire et entrez le programme suivant:

	ATI	405				OR	IC-	i				
#400- #403- #404-	88	C5	DB	JSR TXA PHA	#D8C5	2Ø 8A 48	ØA	D8	JSR TXA PHA	#D8ØA	;GETE ;1er ;empi	paramètre
#405- #408- #409-	88	C5	DB	JSR TXA PHA	#D8C5		ØA	D8		#D8ØA	GETE	BYTC parametre
#4ØA- #4ØD- #4ØE-	8A	C5	D8	JSR TXA PHA	#D8C5	2Ø 8A 48	ØA	DB	JSR TXA PHA	#D8ØA	GETE	BYTC paramètre
#40F- #412- #415-	20	65 53			#DØ65 #E853	20 20 68		CF E7	JSR	#CFD9 #E79D	; CHKC	COM, virgule? NUM,GETADR paramètre
#416- #417- #418-	A8 68			TAY PLA TAX		A8 68 AA			TAY PLA TAX		dans	Y paramètre
#419- #41A-	68	33	ØØ	PLA	(#0033)	68	33	00	PLA	(#ØØ33)	;1er	param dans A (LINNUM)

#420- CHARGEMENT DES REGISTRES

Pour l'application qui suit, à partir de l'adresse #420, entrez la chaine de caractères: CHARGEMENT DES REGISTRES (zéro à la fin, inutile)

Vous devez suivre sans peine le processus du recueil des paramètres par GETBYTC dans X, empilages et désempilages dans les registres concernés.

En #40F, CHKCOM vérifie la présence de la virgule. GETBYTC la sautait, ce que ne fait pas FRMNUM.

En #412, le JSR envoie successivement à FRMNUM, où le nombre est évalué, mis dans l'accumulateur flottant (FAC), puis GETADR qui le prend en FAC et l'inscrit dans LINNUM (#33,34).

La routine se termine par un saut indirect à l'adresse contenue dans LINNUM.

Entrez les lignes Basic:

8000 CLS:DOKE#A4, #BEF8: CALL#400, #18, #20, #04, #D7B2 '(ORIC-1 #D6F7)
8010 PRINT"FAIT"

RUN 8000

La première instruction charge le pointeur FRESPC (#A4,A5) avec #BEF8, une adresse du bas de l'écran.

Les trois valeurs après CALL#400 sont respectivement, la longueur de

la chaîne entrée en #420 (24 caractères) l'octet faible puis l'octet fort de l'adresse.

Elles sont transférées, dans l'ordre, dans les registres A, X et Y. L'adresse qui suit est celle de la routine à actionner. Il s'agit de MOVSTR en #D7B2 (ORIC-1 #D6F7) qui déplace une chaine de longueur (A), pointée par X(-) et Y(+) à l'adresse contenue dans FRESPC.

- La commande !

Vous avez employé cette commande à diverses occasions en mode direct. Pour la transmission de paramètres, il faut savoir que TXTPTR est en #36 après! et que l'accumulateur contient le premier caractère suivant la commande.

Cela convient à la plupart des routines du système comme LINGET ou PTRGET qui actionnent CHARGET pour les caractères suivants. Si votre propre routine doit analyser le contenu du buffer d'entrée au delà du caractère après!, il faudra qu'elle utilise CHARGET.

La commande ! est utilisable dans un programme. N'oubliez pas de placer en #2F5,2F6 l'adresse de la routine concernée.

Modifiez la ligne 8000 précédente:

8000 CLS: DOKE#A4, #BFE8: DOKE#2F5, #400: !, #18, #20, #04, #D7B2 'ORIC-1 #D6F7

- La fonction USR

C'est le moyen privilégié de transmission d'une valeur numérique à une routine en langage machine.

Encore une fois, il faudra prévoir un retour en arrière. Les explications qui suivent se réfèrent à des notions qui seront étudiées dans la troisième partie (traitement des fonctions en général et nombres en virgule flottante).

Dans le cas de USR, sachez simplement qu'après évaluation et mise en FAC de l'argument (l'expression numérique entre parenthèses), l'interpréteur ordonne un JSR #21.

L'utilisateur est censé avoir placé en #22 et #23 l'adresse d'une routine en langage machine chargée du traitement approprié de l'expression numérique.

Il le fait au moyen de l'instruction DEF USR = ADR.

Notez que cette instruction a pour seul effet de placer l'adresse apécifiée en #22,23. Elle n'est pas indispensable. Un DOKE#22,ADR suffit.

Le résultat du traitement de l'expression N est conservé dans l'accumulateur flottant et peut être affiché par PRINT USR(N) ou sous forme entière hexadécimale par HEX\$(USR(N)), s'il est compris entre Ø et 65535.

Vous pourrez recommencer à loisir l'expérience suivante, en modifiant l'expression numérique.

Entrez successvement en commande directe:

DEF USR=#F2

A=144

N=SQR(A)

?USR(N)*3

88

#F2 est l'adresse du RTS de CHARGET. Il y a donc un simple aller et retour sans traitement. (contenu de FAC inchangé)

Voici, parmi beaucoup d'autres, une application simple de l'emploi de USR.

La recherche d'une adresse dans une table.

Il s'agit ici de retrouver, dans la table en #C006, l'adresse d'entrée d'une commande (de END à NEW) définie par le code correspondant (de #80 à #C1).

Entrez la routine USR:

	TA	105				OR	IC-	i			
#400-	20	СВ	DB	JSR	#D8CB	20	10	DB	JSR	#D81Ø	CONINT
#403-	88			TXA		88			TXA		;
#404-	10	10		BPL	#416	10	10		BPL	#416	;si C<#80
#406-	C9	C2		CMP	>#C2	C9	C2		CMP	>#C2	; ou C>=#C2
#408-	BØ	ØC		BCS	#416	BØ	ØC		BCS	#416	; ?ILL QUANT
#4ØA-	ØA			ASL	A	ØA			ASL	A	; C=(C-#8Ø)*2
#4ØB-	AA			TAX		AA			TAX		;index table
#4ØC-	BD	06	CØ	LDA	#CØØ6,X	BD	06	CØ	LDA	#C006,X	;octet faible
#40F-	A8			TAY		8A			TAY		jen Y
#410-	BD	07	CØ	LDA	#C007,X	BD	Ø7	CØ	LDA	#C007,X	;fort en A
#413-	4C	40	DF	JMP	#DF4Ø	4C	D3	ED	JMP	#D8D5	;GIVAYF2
#416-	4C	36	D3	JMP	#D336	4C	AØ	D2	JMP	#D2AØ	; ILL.QUANT.ERR

Puis le programme Basic:

8000 DEF USR=#400:PRINT:INPUT"code";C 8010 PRINT"entree en: ";HEX\$(USR(C)+1):GOTO 8000

RUN 8000

CONINT convertit le nombre en FAC (0 à 255) en un entier signé sur un octet dans X.

Après transfert dans A, les valeurs inférieures à #80 (END) ou supérieures à #C1 (NEW) sont rejetées, avec message d'erreur.

ASL produit le résultat indiqué. En effet, prenons la valeur #82, par exemple: en binaire 10000010, après décalage à gauche, elle devient: 00000100 = #04. Essayez d'autres valeurs.

Le nombre obtenu est retransféré dans X pour l'indexation dans la table.

L'octet faible d'adresse est placé dans Y, l'octet fort dans A pour mise en FAC par GIVAYF2, routine qui rend flottant le nombre entier non signé contenu dans Y(-),A(+). Le RTS de cette routine est utilisé pour le retour au Basic.

L'adresse en FAC augmentée de un (adresse d'entrée effective) sera affichée par le PRINT USR du Basic.

Remarque:

Cet exemple est volontairement très simple.

Il faut rappeler que l'expression numérique manipulée par la fonction USR peut représenter tout nombre réel compris entre 2.93874E-39 et 1.70141E+38.

- la fonction &

Elle fait apparemment double emploi avec la précedente. L'adresse de la routine doit être placée en #2FC,2FD. Si l'argument est un nombre entier hexadécimal, il doit évidemment être compris entre #00 et #FFFF, mais tous les nombres réels sont acceptés comme pour USR.

De même, le résultat est conservé dans l'accumulateur flottant et peut être affiché par PRINT &(N), ou par PRINT HEX\$(&(N)) dans les mêmes conditions.

Entrez le programme Basic:

9000 POKE#420,76:DOKE#421,#DDA7:DOKE#2FC,#420 9010 INPUT N:PRINT &(N):GOTO 9010

RUN 9000

Les deux premières instructions inscrivent en #420-422 un JMP #DDA7 (à remplacer par #DDA3 sur ORIC-1). Cette routine, MUL10, multiplie le contenu de FAC par 10

Entrez un nombre quelconque, entier ou fractionnaire, et voyez le résultat.

CONVERSION BINAIRE -> DATA

Ce programme est un produit ancien, probablement plein de défauts. Je vous le livre tel quel. Espérons au moins qu'il ne se plantera pas.

Il comporte une partie Basic qui garnit le buffer d'entrée exactement comme si la ligne de DATA était entrée au clavier.

La ligne 17 entre le numéro de ligne.

La ligne 18 entre le mot DATA et insère un espace.

La boucle des lignes 19 à 23 traite 17 octets.

Elle accomplit successivement:

- Lecture en hexadécimal pour obtenir DATA en décimal, remplacer A\$=HEX\$(PEEK(A)) par A\$=MID\$(STR\$(PEEK(A)),2) (MID\$ supprime l'espace qui remplace le signe d'un nombre positif).
- mise en buffer,

90

- virgule après chaque valeur, sauf après la 17°.

Après cela, la ligne étant complète, un zéro est placé à la fin. Le CALL#7006 appelle la sous-routine en langage machine qui analyse la ligne dans le buffer d'entrée et l'insère.

Auparavant, l'adresse du prochain octet à lire, A, et le numéro de ligne, L, sont sauvegardés (comme l'avait été F, fin du code, à la ligne 15) car la sous-routine efface les variables (CLEAR).

C'est d'ailleurs ce qui se passe lorsque vous entrez une ligne de programme au clavier.

A la ligne 24, les variables sont récupérées. Le numéro de ligne est

96

augmenté de 10.

Vous remarquerez l'appel à un autre sous-programme en #705D. Celui-là est indépendant de l'autre. Il supprime les lignes 0 à 25 à la fin de l'opération.

Les numéros lui sont passés par l'intermédiaire des emplacements #80,81 et #82,83. C'est à la fin du buffer d'entrée, mais on ne s'en sert plus à ce moment-là.

Le programme utilitaire qui permet de supprimer des lignes en bloc est pratique. Vous le retrouverez, à peine modifié, au chapitre 15.

Essayez le programme sur le code qu'il utilise: Adresse de début #7000, de fin #70BC. Vous devriez obtenir les mêmes lignes DATA avec des numéros différents.

Le listing du code machine est sommairement commenté. Une bonne partie est empruntée à la routine principale d'entrée des commandes que nous étudierons au chapitre 12.

Le code machine doit être présent en mémoire avant chargement de ce programme utilitaire.

Pour ATMOS

- Ø DATA #FF, #FF, #FF, #FF, #FF, #A5, #E9, #A4, #EA, #B5, #, #B4, #1, #A2, #34, #AØ
- 1 DATA #,#86,#E9,#84,#EA,#20,#E2,#,#20,#E2,#CA,#20,#FA,#C5,#84,# 26,#20
- 2 DATA #B3, #C6, #18, #A5, #9C, #85, #C9, #65, #26, #85, #C7, #A4, #9D, #84, #CA, #90, #1
- 3 DATA #C8,#84,#C8,#20,#F4,#C3,#A5,#A0,#A4,#A1,#85,#9C,#84,#9D,# A4,#26,#88
- 4 DATA #89,#31,#,#91,#CE,#88,#10,#F8,#20,#08,#C7,#20,#5F,#C5,#A5,#,#A4
- 5 DATA #1,#85,#E9,#84,#EA,#4C,#C1,#C8,#A5,#80,#85,#33,#A5,#81,#8 5,#34,#20
- 6 DATA #B3, #C6, #A5, #CE, #85, #91, #A5, #CF, #85, #92, #A5, #82, #85, #33, #A5, #83, #85
- 7 DATA #34, #E6, #33, #D0, #2, #E6, #34, #20, #B3, #C6, #A5, #CE, #C5, #91, #A 5, #CF, #E5
- 8 DATA #92, #80, #1, #60, #A0, #, #B1, #CE, #91, #91, #E6, #CE, #D0, #2, #E6, #CF, #E6
- 9 DATA #91, #DØ, #2, #E6, #92, #A5, #9C, #C5, #CE, #A5, #9D, #E5, #CF, #BØ, #E 6, #A6, #92
- 10 DATA #A4, #91, #D0, #1, #CA, #88, #86, #9D, #84, #9C, #20, #08, #C7, #20, #5F, #C5, #4C
- 11 DATA #A8, #C4

ORIC-1

- Ø DATA #FF,#FF,#FF,#FF,#FF,#FF,#A5,#E9,#A4,#EA,#85,#,#84,#1,#A2, #34.#AØ
- 1 DATA #,#86,#E9,#84,#EA,#20,#E2,#,#20,#98,#CA,#20,#A,#C6,#84,#26,#20
- 2 DATA #DE, #C6, #18, #A5, #9C, #85, #C7, #65, #26, #85, #C7, #A4, #9D, #84, #CA, #90, #1
- 3 DATA #C8, #84, #C8, #20, #F8, #C3, #A5, #A0, #A4, #A1, #85, #9C, #84, #9D, #A4, #26, #88
- 4 DATA #89,#31,#,#91,#CE,#88,#10,#F8,#20,#33,#C7,#20,#6F,#C5,#A5,#,#A4
- 5 DATA #1,#85,#E9,#84,#EA,#4C,#AD,#C8,#A5,#80,#85,#33,#A5,#81,#85,#34,#20
- 6 DATA #DE, #C6, #A5, #CE, #85, #91, #A5, #CF, #85, #92, #A5, #82, #85, #33, #A5, #83, #85
- 7 DATA #34, #E6, #33, #D0, #2, #E6, #34, #20, #DE, #C6, #A5, #CE, #C5, #91, #A 5, #CF, #E5
- 8 DATA #92,#B0,#1,#60,#A0,#,#B1,#CE,#91,#91,#E6,#CE,#D0,#2,#E6,#CF,#E6
- 9 DATA #91, #DØ, #2, #E6, #92, #A5, #9C, #C5, #CE, #A5, #9D, #E5, #CF, #BØ, #E 6, #A6, #92
- 10 DATA #A4,#91,#DØ,#1,#CA,#88,#86,#9D,#84,#9C,#20,#33,#C7,#20,#6F,#C5,#4C
 - 11 DATA #85,#C4

ATMOS et ORIC-1

- 12 FOR I=0 TO 188: READ C: POKE#7000+I, C: NEXT
- 13 TEXT:CLS:INPUT"No 1ere ligne DATA (>25)";L:IF L<26 THEN 13
- 14 INPUT"Adresse debut code machine";A
- 15 INPUT"Adresse fin code machine": F: DOKE#7004, F: IF F <= A THEN 15
- 16 IF A>=F THEN DOKE#80,0:DOKE#82,25:CALL#705D
- 17 A\$=MID\$(STR\$(L),2):BUF=#35:GOSUB 25
- 18 A\$="DATA":GOSUB 25:POKE BUF,32:BUF=BUF+1
- 19 J=0:REPEAT: A\$=HEX\$ (PEEK(A)):GOSUB 25
- 20 IF A=F THEN PULL:POKE BUF,0:CALL#7006:DOKE#80,0:DOKE#82,25:CALL#705D
 - 21 A=A+1:J=J+1:IF J=17 THEN 23
 - 22 POKE BUF, 44: BUF=BUF+1
 - 23 UNTIL J=17:POKE BUF, 0: DOKE#7000, A: DOKE#7002, L: CALL#7006
 - 24 A=DEEK(#7000):L=DEEK(#7002)+10:F=DEEK(#7004):GOTO 16
- 25 FOR I=1 TO LEN(A\$):POKE BUF,ASC(MID\$(A\$,I,1)):BUF=BUF+1:NEXT: RETURN

Stockage des lignes DATA

```
Les variables Basic
#7000:
                 sont sauvegardées
#7001:
                 dans
#7002:
#7003:
                 ces
#7004:
                 emplacements
#7005:
                                  ;la position actuelle
                 LDA TXTPTR
#7006: début
                                  ;du pointeur de texte
                 LDY TXTPTR+1
#7008:
                 STA #00
                                  est sauvegardée
#700A:
                                  ;pour reprise exécution ultérieure
#700C:
                 STY #01
                 LDX >#34
                                  ; TXTPTR
#700E:
                 LDY >#00
                                  ;placé à
#7010:
                 STX TXTPTR
                                  ; BUF-1
#7012:
                 STY TXTPTR+1
#7014:
                                  pour
                                  recueil ler caractère du buffer
#7016:
                 JSR CHARGET
                 JSR LINGET
                                  ¡No de ligne dans LINNUM
#7019:
                 JSR PARSE
                                  ;analyse du contenu de la ligne
#7Ø1C:
                 STY TEMP
                                  ;nombre de caractères
#701F:
                                  ;adresse ad-hoc en LOWTR (#CE,CF)
                 JSR FNDLIN
#7021:
#7024:
                 CLC
                                  :cette
#7025;
                 LDA VARTAB
                                  ;partie
                 STA HIGHTR
                                  ; est
#7027:
#7029:
                 ADC TEMP
                                  ; empruntée
                 STA HIGHDS
#702B:
                                  ; à
                 LDY VARTAB+1
#702D:
                                  ; CMDLP
                 STY HIGHTR+1
                                  ;insertion
#702F:
                 BCC +1
                                  ; d'une
#7031:
                 INY
                                  ;ligne
#7033:
                 STY HIGHDS+1
#7034:
                                  ; Basic
                                  ;déplacement queue du programme
#7036:
                 JSR BLTU
                 LDA STREND
#7039:
                                  :mise
                 LDY STREND+1
                                  ; à jour
#7Ø3B:
#703D:
                 STA VARTAB
                                  ; de
#703F:
                 STY VARTAB+1
                                  ; VARTAB
                 DEY
                                  ; puis
#7043:
                 LDA BUF-4,Y
#7044:
                                  ; mise
                 STA (LOWTR),Y
#7047:
                                  ; en
#7049:
                 DEY
                                  ;place
#704A:
                 BPL -8
                                  ;de la ligne
                 JSR SETPTRS
#7Ø4C:
                                  ; positionnement
                 JSR LINKSET
                                  ; des octets de chainage
#704F:
                 LDA #00.
#7052:
                                  ;TXTPTR reprend
#7054:
                 LDY #01
                                  ;sa valeur initiale
#7056:
                 STA TXTPTR
                                  dans le texte Basic
                 STY TXTPTR+1
#7058:
                                  ; pour
                 JMP NEWSTT
                                  ;une nouvelle instruction.
#705A:
```

Suppression de lignes

```
LDA #80
                                 ; limite
#7Ø5D:
                 STA LINNUM
#705F:
                                 ;basse
#7061:
                 LDA #81
                                 ; placée
#7063:
                 STA LINNUM+1
                                 en LINNUM
                                 ;adresse ligne en LOWTR
#7065:
                 JSR FNDLIN
                 LDA LOWTR
#7068:
                                 ;transférée
#706A:
                 STA INDEX
                                 ; dans
#706C:
                 LDA LOWTR+1
                                 ;pointeur
                 STA INDEX+1
#706E:
                                 :temporaire
#7070:
                 LDA #82
                                 ; limite
#7072:
                 STA LINNUM
                                 ; haute
#7074:
                 LDA #83
                                 dans
                 STA LINNUM+1
#7076:
                                 ; LINNUM
                  INC LINNUM
#7078:
                                 ; puis
#707A:
                 BNE +2
                                 :incrémentée
#707C:
                  INC LINNUM+1
                                 ;pour recherche
#707E:
                 JSR FNDLIN
                                 ;adresse ligne suivante
#7081:
                 LDA LOWTR
                                 :cette adresse
                 CMP INDEX
#7083:
                                 ;est comparée à la
                 LDA LOWTR+1
#7085:
                                 ; limite basse
                 SBC INDEX+1
                                 ¡Remarque: c'est superflu ici
#7087:
#7089:
                 BCS +1
                                 ;ce branchement est toujours pris
#708B:
                 RTS
#708C: suppr
                 LDY >#00
                                 ; index
#708E: écrase
                 LDA (#CE),Y
                                 ; la première ligne valide
                                 ; vient écraser la
#7090:
                 STA (#91),Y
#7092:
                 INC LOWTR
                                 ;première ligne
#7094:
                 BNE +2
                                 ; à supprimer
#7096:
                  INC LOWTR+1
                                 ;et
#7098:
                  INC INDEX
                                 ; ainsi
                 BNE +2
#709A:
                                 ; de suite
#7Ø9C:
                 INC INDEX+1
                                 ; jusqu'à
#709E:
                 LDA VARTAB
                                 ;la fin
#7ØAØ:
                 CMP LOWTR
                                 ; du
                 LDA VARTAB+1
#70A2:
                                 ; programme
                 SBC LOWTR+1
#70A4:
#70A6:
                 BCS écrase
                                 ; boucle
#70A8:
                 LDX INDEX
                                 ; terminé
#7ØAA:
                 LDY INDEX+1
                                 ; mise
#70AC:
                 BNE +1
                                 ; à
#7ØAE:
                 DEX
                                 ; jour
#70AF:
                 DEY
                                 ; de
                                 ; VARTAB
#70B0:
                 STX VARTAB
#70B2:
                 STY VARTAB+1
#70B4:
                                 ; positionnement
                 JSR SETPTRS
#70B7:
                 JSR LINKSET
                                 ;octets de chainage
#70BA:
                 JSR READY2
                                 ;ready
```

TROISIEME PARTIE

FONCTIONNEMENT DE L'INTERPRETEUR

INTRODUCTION

1

Au premier contact avec le code machine en ROM, on a l'impression de se trouver devant un énorme puzzle découpé dans le plus hermétique des tableaux abstraits.

Cette troisième partie doit vous donner une idée de ce qu'il représente.

Un éclairage particulier donné à certaines zones vous aidera à commencer à mettre en place les éléments du puzzle. Le reste est affaire de patience et de méthode. Les mordus de micro-informatique tels que vous n'en manquent pas.

Le chapitre 11 schématise le fonctionnement global de l'interpréteur. Il rappelle les concepts de base et énumére les divers moyens utilisés.

Les routines essentielles CHARGET et NEWSTT sont étudiées.

Quelques autres routines fondamentales sont commentées dans le chapitre suivant.

Vous remarquerez qu'elles sont toutes situées au début de la ROM. Ce n'est pas un hasard. Il y a un ordre logique dans le mécanisme. Les passages choisis sont aussi des routines-clés pour l'interpréteur.

L'étude vous paraîtra peut-être ardue. Surtout, n'insistez pas, vous y reviendrez plus tard.

Certains détails ne sont pas indispensables à connaître pour bien utiliser la machine.

Néammoins, ce n'est pas de l'information pour homme du monde. Si vous avez l'intention de vous perfectionner en langage machine, vous tirerez le plus grand profit à découvrir les techniques employées. Il s'agit, en général, de programmation de haut niveau.

Le chapitre 13 est consacré à la représentation des nombres en virgule flottante.

Le programme de démonstration, en fin de chapitre, n'est pas très long à taper. Je vous recommande de l'essayer intensivement.

Les principales routines utilisables sont regroupées par grandes fonctions dans le chapitre 14.

C'est plus commode quand il s'agit de résoudre un problème particulier.

Enfin, quelques exemples d'applications sont proposés. L'aspect pratique a été recherché, plutôt que le côté purement démonstratif.



CHAPITRE 11

PRINCIPES DE BASE

UTILISATION D'UNE IMPRIMANTE DRAPEAU PRTFLG, #2F1

Dans ce chapitre et dans les suivants, de nombreuses routines seront citées.

Il est souhaitable que vous utilisiez votre désassembleur pour les examiner. Encore mieux si vous disposez d'une imprimante. Dans ce cas, il n'est pas nécessaire de modifier en LPRINT toutes les commandes PRINT de votre utilitaire. Vous ajouterez simplement les lignes suivantes:

315 POKE#2F1,128

375 IF A> adresse fin désassemblage THEN END.

L'imprimante étant branchée, il suffit de faire RUN, option 1 désassemblage et d'entrer l'adresse de début. La ligne 375 est indispensable pour arrêter l'impression.

* * *

Avant d'examiner la structure de l'interpréteur et les moyens dont il dispose, il convient de bien préciser le sens des termes courants. Ils sont parfois indifféremment employés l'un pour l'autre. Pour éviter toute confusion, nous respecterons strictement les définitions ci-après.

TERMINOLOGIE

- Mot-clé.

, 1

On désigne sous ce nom l'un des 119 mots réservés du Basic, de END à MID\$, qui figurent dans la table en #CØEA.

Reconnus au cours de l'opération d'analyse, ils sont remplacés par un code numérique compris entre #80 et #F6 qui correspond à leur position relative dans la table (codage).

Ils sont de quatre sortes de signification très différente: commande, fonction, opérateur, modificateur.

- Commande.

Une commande est l'un des 66 premiers mots-clés, de END à NEW (code compris entre #80 et #C1).

Obligatoirement présente au début de toute instruction Basic, avec une exception, dans le cas de l'affectation d'une valeur à une variable (la commande LET est sous-entendue), elle joue un rôle comparable à celui du code opératoire du langage machine.

Lors de la phase d'exécution, le code de la commande ayant été reconnu valide par la routine NEWSTT, un branchement est effectué sur la routine appropriée dont l'adresse (-1) se trouve dans la table de #C006 à #C089.

La syntaxe est vérifiée par cette routine d'exécution. Le processus est identique quelle que soit l'instruction.

- Fonction.

Une fonction est l'un des 43 mots-clés de SGN à MID\$ (codes compris entre #D6 et #F6).

Numérique ou alphanumérique, elle est suivie d'un argument entre parenthèses.

Si présent, le code d'une fonction est reconnu lors de l'opération d'évaluation de l'expression (FRMEVL) qui suit la commande, ordonnée par la routine d'exécution elle-même.

Un appel est alors fait à la routine appropriée, calcul ou découpage de chaîne de caractères, dont l'adresse est trouvée dans la table en #CØ8A-CØCB.

- Opérateur.

Le code d'un des 10 opérateurs arithmétiques ou logiques (#CC à #D5) est aussi détecté par l'évaluateur d'expressions (FRMEVL) appelé par la routine de traitement de la commande.

Les adresses des routines correspondantes (-1) figurent dans la table en #COCC-COE9 (2°et 3°octet des dix groupes de 3 octets).

- Modificateur

Les modificateurs, de TAB(à STEP (codes compris entre #C2 et #CC) sont associés à une commande pour la modifier éventuellement. Par exemple, TAB(avec PRINT.

Ils n'ont pas de point d'entrée répertorié, car ils sont traités en meme temps que la commande.

Le terme est assez mal choisi car certains sont obligatoires, ainsi THEN après IF, TO après FOR.

Notez que dans le cas particulier de la commande PRINT, les symboles , ou ; ont aussi une action modificatrice mais ne sont pas codés.

- Instruction.

Une instruction est une phrase Basic complète, un ordre exécutable par l'ordinateur.

Elle comprend obligatoirement une commande qui peut être seule (ex: CLS, NEW..) ou suivie d'une expression numérique ou alphanumérique incluant éventuellement fonction ou opérateur.

Plusieurs instructions peuvent être groupées dans une même ligne entrée au clavier. Elles sont séparées par deux-points 'i'

- Mode immédiat ou différé

Une instruction, ou un groupe d'instructions, entrés au clavier sans numéro de ligne sont exécutés immédiatement. On parle de mode direct ou immédiat, ou de commande directe.

Lorsque ces mêmes instructions sont précédées d'un numéro de ligne, l'interpréteur les range en mémoire pour une exécution ultérieure. C'est le mode différé ou mode programme.

La fin d'une ligne de programme est repérée par la valeur 00, la fin d'instruction par le symbole ':' (ou par 00 si c'est le dernière ou la seule instruction de la ligne) et la fin de programme par trois 00 consécutifs.

STRUCTURE DE L'INTERPRETEUR

L'interpréteur est un programme en langage machine qui traduit les instructions Basic et les rend exécutables par l'ordinateur.

Il réside en ROM à partir de l'adresse #C3C6 (#C3CA pour ORIC-1).

Le point d'entrée est en #C4B7 sur ATMOS, en #C4C7 sur ORIC-1. Il correspond au début de la boucle principale d'entrée des commandes, CMDLP.

L'interpréteur trouve les données à traiter et l'information actualisée dont il a besoin dans différentes zones en mémoire.

- pages 0 et 2, divers drapeaux, pointeurs et variables système.
- buffer d'entrée (#35-83), l'information entrée au clavier.
- pile (#100-1FF), adresses de retour, informations temporaires.
- normalement à partir de #501, le programme Basic et ses variables.
- cinq tables au début de la ROM.
- disséminées en ROM, des informations occasionnellement utilisées, messages divers, constantes virgule flottante, tables d'adresses des routines HGR et son, etc..

ROUTINE D'OBTENTION D'UN CARACTERE - CHARGET - CHARGOT

ATMOS et ORIC-1 #E2-#F2

C'est le coeur de l'interpréteur, la routine la plus importante et la plus utilisée.

A chaque fois qu'il a besoin de connaître le contenu de l'octet suivant, dans un programme ou dans le buffer d'entrée, l'interpréteur appelle CHARGET.

Il le fait souvent car sa fonction essentielle est de déchiffrer, d'interpréter, les phrases du Basic.

La routine CHARGET elle même, longue de 17 octets, réside en mémoire morte, en #EC9C (ORIC-1 #EA24).

Elle est recopiée en page zéro (#E2-F2) pendant l'initialisation (ATMOS #ECCC, ORIC-1 #EA59) en même temps que le nombre en virgule flottante RNDSEED (#FA-FF) qui sert à la fonction RND. Incidemment, vous remarquerez que ce nombre n'est pas correctement

Incidemment, vous remarquerez que ce nombre n'est pas correctement recopié, il manque le 5° octet (bug?).

Entre CHARGET et RNDSEED, les octets sont sans signification.

Revenons à CHARGET, en page zéro.

#E2- CHARGET	E6 E9 DØ Ø2	INC TXTPTR BNE +2	;fait pointer TXTPTR ;sur octet
#E6-	E6 EA	INC TXTPTR+1	;suivant
#E8- CHARGOT	AD ff FF	LDA (TXTPTR)	;valeur à TXTPTR dans A
#EB-	C9 20	CMP " "	;est-ce un espace?
#ED-	FØ F3	BEQ CHARGET	;oui, l'ignorer
#EF-	20 B9 EC	JSR SETINDIC	;non, qu'est-ce?
#F2-	60	RTS	;à disposition interpréteur ;caractère en A, ;indicateurs renseignés.
			, indicaced a rensergies,

Cette routine est conçue pour se modifier elle-même, c'est pourquoi elle est installée en mémoire vive.

A chaque fois qu'elle est appelée, TXTPTR en #E8,E9, qui est inclus dans la routine, pointe l'octet suivant dont le contenu est chargé

dans l'accumulateur.

Notez qu'un JSR CHARGOT (#E8) a pour effet de recueillir la valeur pointée par TXTTPTR, sans incrémentation préalable de ce pointeur de texte.

Si l'accumulateur a reçu le code d'un caractère espace, ce dernier est ignoré, on passe à l'octet suivant.

Sinon, il est fait appel à la sous-routine SETINDIC restée en ROM. Le rôle important de cette sous-routine est de positionner les indicateurs Z et C, selon la nature de la valeur recueillie dans l'accumulateur.

A la sortie, (A) = le caractère.

```
- Z = 1
                      (A) = \#CB
               si
                                            ELSE
                                     (')
                      (A) = #27
               QU
                                            REM
                                            fin d'instruction.
                      (A) = #3A
                                     (:)
               OH
               OIL
                      (A) = \#\emptyset\emptyset
                                            fin de ligne.
- C = \emptyset
                  (A) est un chiffre, de Ø à 9.
```

La sous-routine est en #ECB9 pour ATMOS, en #EA41 pour ORIC-1. Elle n'est pas identique sur les deux systèmes car leur façon de traiter ELSE après IF.. THEN, ou le symbole '(REM) en fin d'instruction, est différente.

Nous n'entrerons pas dans le détail pour le moment (voir NEWSTT, plus loin). Le résultat est le même.

```
#ECB9- C9 C8
                  CMP "else"
                              ;ELSE? devenu inutile
                   BEQ RTS
       FØ ØE
#ECBB-
                               ;oui, Z=0, fin de l'instruction
                  CMP " ' "
#ECBD- C9 27
                               ; REM?
#ECBF- FØ ØA
                  BEQ RTS
                               ;oui, Z=0, fin de l'instruction
#ECC1- C9 3A
                  CMP " : "
                               ;deux-points? si =, Z=1 fin ins...
#ECC3-
       BØ Ø6
                   BCS RTS
                                  ou si >, pas un chiffre, C=1
#ECC5-
       38
                  SEC
                              ;est-ce un
       E9 30
                   SBC >#30
#ECC6-
                              ;chiffre? (de Ø à 9)
#ECC8-
       38
                  SEC
                              ; si oui faire C=0..
#ECC9- E9 DØ
                   SBC >#DØ
                              ; sinon faire C=1
#ECCB- 60
                  RTS
                               ;retour CHARGET
```

Si l'une ou l'autre des deux premières comparaisons est concluante, l'indicateur Z est mis à un pour indiquer que la valeur concernée marque la fin de l'instruction. A charge pour l'interpréteur de se positionner pour l'instruction suivante.

En ce qui concerne la première, il s'agit d'un ELSE inutile, la condition vraie entre IF et THEN ayant été traitée par ailleurs.

La troisième comparaison fournit deux renseignements. Si C = 1, il ne s'agit pas d'un chiffre; code du caractère égal ou supérieur à #3A.

De plus, si ce code est égal à #3A (deux-points), fin de l'instruction, Z est mis à un par la comparaison.

Les dernières instructions sont un peu plus difficiles à comprendre.

Après la première soustraction, l'accumulateur contiendra une valeur comprise entre #00 et #09 si le caractère est un chiffre. Sinon, cette valeur sera comprise entre #00-#30 = #00 et #2F=#30 =

#FF.

1 !

1 |

! }

La deuxième soustraction remet l'accumulateur dans son état initial, car, en ignorant la retenue, #30+#D0 = #00.

De plus, si le caractère est un chiffre, il y aura un emprunt et l'indicateur C sera mis à zéro.

Enfin, si le contenu de l'accumulateur était zéro, il retrouve cette valeur zéro et l'indicateur Z est mis à un, signalant cette fois la fin de ligne.

PRINCIPE DE FONCTIONNEMENT

On peut distinguer deux phases dans l'intervention de l'interpréteur:

- Une phase qui comporte le recueil du texte composé au clavier dans le buffer d'entrée, l'analyse et le codage, suivie d'une exécution immédiate dans le cas d'une commande directe ou du rangement en mémoire s'il s'agit d'une ligne de programme.
- Une phase d'exécution où les instructions codées sont traitées tour à tour.

Pour chaque instruction, le traitement commence par un branchement à la routine d'exécution de la commande concernée.

ENTREE, ANALYSE, CODAGE MODE IMMEDIAT OU DIFFERE

Cette première phase peut être schématisée ainsi:

- 1. Initialisation.
- A la mise sous tension, l'ordinateur est mis en état. Pointeurs et variables système en pages Ø et 2 sont réglés.
- 2. Affichage Ready.
- 3. Début de la boucle principale d'entrée des commandes (CMDLP). L'interpréteur attend l'information entrée au clavier.
- 3. Recueil de l'information dans le buffer d'entrée (#35-83) par INLIN.

La valeur 00 est placée à la fin indiquée par RETURN.

- 4. Au retour de INLIN, le premier caractère dans le buffer est recueilli par CHARGET.
- Si ce n'est pas un chiffre, il s'agit d'une commande directe, analyse et codage (PARSE), exécution (NEWSTT) puis GOTO 2.
- Si c'est un chiffre, le numéro de ligne entier est lu par LINGET et inscrit dans LINNUM. Au retour de LINGET le pointeur de texte est laissé sur le premier caractère, autre qu'un espace, suivant le numéro de ligne.

PARSE effectue le codage du texte, puis FNDLIN détermine si la ligne existait déjà. Si oui, l'ancienne ligne est supprimée par écrasement (déplacement du reste du programme vers le bas).

La nouvelle ligne est insérée à l'emplacement approprié.

Ajustement des octets de chainage (LINKSET), GOTO 3 pour entrée ligne suivante éventuelle.

Toutes ces opérations sont controlées par CMDLP, dont vous trouverez le commentaire au chapitre 12.

Remarque:

au cours de l'analyse et du codage, la seule vérification de la syntaxe est effectuée par LINGET (mode différé) et porte sur le numéro de ligne entré au clavier qui doit être inférieur à 64000.

EXECUTION

L'instruction Basic, après codage par PARSE, commence par une valeur binaire comprise entre #80 (code de END) et #C1 (code de NEW). C'est cette valeur (code opératoire) qui permet le branchement à la routine de traitement spécifique dont l'adresse moins un est trouvée dans la table en #C006.

Nous verrons le pourquoi du 'moins un' tout à l'heure. Auparavant, essayez ce programme.

```
8000 FOR I=0 TO 23:READ C:POKE #400+I,C:NEXT:CLS
8010 PRINT:PRINT"mot-cle";:CALL#400:K=PEEK(#35):IF K<#80 THEN END
8020 IF K<#C2 THEN PRINT"commande, ";:GOTO 8040
8030 IF K>#D5 THEN PRINT"fonction, ";:K=K-20 ELSE END
8040 PRINT"entree en ";HEX*(DEEK(#C006+(K-#80)*2)-1*(K<#C2)):GOTO 8010
```

ATMOS

```
8050 DATA #A5,#E9,#A4,#EA 'LDA #E9, LDY #EA ;TXTPTR transféré
8060 DATA #85,#AC,#84,#AD 'STA #AC, STY #AD ;dans OLDTXT
8070 DATA #20,#80,#CD 'JSR #CD80';'INPUT'
8080 DATA #86,#E9,#84,#EA 'STX #E9, STY #EA ;(TXTPTR)=BUF-1
8090 DATA #20,#E2,#00' 'JSR #E2 ;pour CHARGET
8100 DATA #20,#FA,#C5 'JSR #C5FA ;PARSE, analyse, codage
8110 DATA #4C,#A2,#C9 'JMP #C9A2 ;CONT basic
```

```
ORIC-1, remplacer les lignes 8070, 8100 et 8110 par les suivantes: 8070 DATA #20,#F4,#CC 'JSR #CCF4 ;'INPUT' 8100 DATA #20;#0A,#C6 'JSR #C60A ;PARSE, analyse, codage 8110 DATA #4C,#70,#C9 'JMP #C970 ;CONT basic
```

RUN 8000

Entrez le nom d'une commande, mot-clé en toutes lettres. Vous obtiendrez l'adresse de la routine correspondante. Ce programme donne aussi l'adresse d'entrée des routines de traitement des fonctions mais oubliez-le pour le moment.

L'interpréteur se sert de NEWSTT pour identifier la commande et obtenir le branchement approprié, aussi bien en mode direct qu'en mode programme.

Cette routine est d'une extrême importance. TOUTES les instructions Basic sont traitées par elle. C'est le centre névralgique, le poste d'aiguillage de l'interpréteur.

EXECUTION D'UNE NOUVELLE INSTRUCTION - NEWSTT

ATMOS #C8C1-#C951 ORIC-1 #C8AD-#C91E

Alors que PARSE et LIST utilisent la table des mots-clés en #CØEA pour le codage et le décodage, NEWSTT se référe exclusivement à la table d'entrée des commandes (de END à NEW) en #CØØ6-CØ89.

Elle détecte la fin d'instruction, de ligne ou de programme et établit plusieurs règles syntaxiques de base.

Les possesseurs d'ATMOS constateront qu'elle traite aussi le mot-clé ELSE devenu sans objet et l'apostrophe (REM) pour passer à l'instruction suivante.

A l'entrée, le pointeur de texte, TXTPTR, pointe sur les deux-points précédant l'instruction ou sur le 00 de fin de ligne précédente.

Sous-routines utilisées par NEWSTT:

	ATMOS	ORIC-1		ATMOS	ORIC-1
ISCNTC	#C962	#0930	LINPRT	#EØC5	#EØC1
PREND	#C98A	#C958	LET	#CB1C	#CAD2
VIDOUT	#CCFB	#CC4D	ADDON2	#CAB1	
SYNTERR	#D07Ø	#CFE4	REM	#CA99	

Table d'entrée des commandes, CMDTBL en #C006

```
1. C8C1 C8AD NEWSTT JSR ISCNTC
                                   test clavier, BREAK si CTRL-C
 2. CBC4 CBBØ
                    LDA TXTPTR
                                   TXTPTR sera inscrit
 3. CBC6 CBB2
                     LDY TXTPTR+1
                                   ...dans OLDTXT
 4. C8C8 C8B4
                     BEQ +6, N1
                                   Cde directe, TXTPTR pointe BUF
 5. CBCA C8B6
                     STA OLDTXT
                                   TXTPTR
                                          dans OLDTXT
 6. CBCC CBB8
                     STY OLDTXT+1
                                   ...
 7. CBCE CBBA
                     LDY >#00
                                   test octet à..
 8. C8DØ C8BA N1
                     LDA (TXTPTR),Y .... TXTPTR
9. C8D2 C8BC
                     BNE COLON
                                   si non 0, doit etre fin instruction
10. C8D4
                     LSR IFFLAG
                                   ATMOS, drapeau IF.. THEN baissé
11. C8D7 C8CØ
                     LDY >#02
                                   test fin de programme
12. C8D9 C8C2
                     LDA (TXTPTR),Y ... valeur 0 à fin de ligne + 2
13. CBDB CBC4
                     CLC
                                   C=0 pour sortie normale, sans BREAK
14. CBDC CBC5
                     BNE +3
                                   ce n'est pas fini
15.- CBDE CBC7
                     JMP PREND
                                   sortie, fin de programme
16. CBE1 CBCA
                     INY
                                   Y indexe maintenant
17. C8E2 C8CB
                     LDA (TXTPTR),Y
                                       ···· le numéro
18. CBE4 C8CD
                     STA CURLIN
                                       ....de ligne
19. C8E6 C8CF
                     INY
                                       ....mise à jour
20. C8E7 C8D0
                     LDA (TXTPTR),Y
                                       ....de
21. C8E9 C8D2
                     STA CURLIN+1
                                       CURLIN (#AB, A9)
22. CSEB CSD4
                     TYA
                                   TXTPTR est ajusté
23. C8EC C8D5
                     ADC TXTPTR
                                       .... caractère suivant
                                       .... devrait être 1er
24. CBEE CBD7
                     STA TXTPTR
25. C8FØ C8D9
                     BCC +2
                                       ....caractère d'une
26. CBF2 CBDB
                     INC TXTPTR+1
                                       ....commande à exécuter
27. C8F4 C8DD EXEC
                     BIT TRACE
                                   TRACE? (#2F4)
28. C8F7 C8EØ
                     BPL L1
                                   non
29. CBF9 CBE2
                     PHA
                                   ? sauvegarde TXTPTR, pourquoi?
30. C8FA C8E3
                     LDA >#5B
                                   affiche '['
31. C8FC C8E5
                     JSR VIDOUT
```

33. 34. 35. 36. 37. 38. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51.	C90F C912 C915 C917 C919 C918 C91D C91F C920 C921 C924 C925 C928 C929	C8EA C8EC C8EF C8F1 C8F4 C8F5 C8F8 C8FB C8FE GOCMD C900 C902 C904 C904 C906 C908 C909 C909 C900 C901	LDX JSR LDA JSR JSR JSR JSR JSR JSR JSR JSR JSR JSR	CHARGET GDCMD NEWSTT RTS >#80 NOCMD >#42 ERR A	affiche numéro de ligne en cours , J, affiche 'J' ? code commande présumée dans A on va voir! mission accomplie, suivante fin de ligne, ligne suivante Si positif, n'est pas une commande doit être variable en affectation code est-il >= #C2? (#80+#42) oui, ce n'est pas une commande index = (code-#80)*2 dans Y Y octet fort adresse trouvée dans la table en #C006 est empilé octet faible idem. Cette adresse désempilée + 1 sera adresse retour de CHARGET, entrée commande. nom de variable en affectation.
	C931	C918 COLON		EXEC	deux-points? oui, c'est bon
ORIC	C-1 li	gne 67			Control (Control (Con
ΔТМГ	18 eeu	11ement			
	C933	ELSE	CMP	'else'	ELSE?
	C935			rem	non, voir si ' (rem)
58.	C937			IFFLAG	drapeau IFTHEN est-il levé?
	C939			ERR	non, ELSE intempestif, erreur
	C93C			ADDON2	TXTPTR -> fin instruction
	C93F			IFFLAG	drapeau IF. THEN baissé
	C942			NEWSTT	nouvelle instruction
	C945	'rem		>#27	' ?
	C947			ERR	non, plus d'espoir
	C949 C94C			REM	TXTPTR -> fin de ligne
00.	U74U		שמוני	NEWSTT	nouvelle ligne
				for a	

ATMOS ORIC-1 TO THE TOTAL BR REM CHAS

67. C94F C91C ERR JMP SYNTERR erreur de syntaxe.

Note:

L'entrée d'une commande directe se fait à EXEC (voir CMDLP). Après exécution de la première instruction, retour à NEWSTT, même processus qu'en mode programme.

Le mode direct est détecté par le fait que TXTPTR pointe dans le buffer d'entrée, octet fort (#EA) à zéro. La mise à jour de OLDTXT est évitée (ligne 4).

Voyons ce qui se passe dans un cas simple, le lancement d'un programme.

Après codage de la commande RUN, mode immédiat, il y a branchement sur EXEC (ligne 27).

Si le mode TRACE n'est pas actif, branchement en L1 (ligne 38) où CHARGET recueille dans l'accumulateur ce qui devrait être le code d'une commande (#80 à #C1).

Le contenu de l'accumulateur est testé (lignes 41-45). Dans notre exemple, le code de RUN étant identifié, l'adresse - 1 est trouvée dans la table et mise en pile.

Le JMP CHARGET (ligne 52) recueille le caractère suivant, dans ce cas 00 puisque la commande RUN est seule.

Le RTS de CHARGET désempile l'adresse dans PC qui est incrémenté (voir page 68) ce qui donne le branchement correct.

Vous savez que RUN a pour effet de mettre l'adresse #500 dans TXTPTR. Au retour (ligne 40) le JMP NEWSTT recommence le cycle depuis le début de programme.

Toutes les instructions sont traitées à leur tour de la même façon jusqu'à ce que la fin de programme soit détectée (lignes 11-15). Remarquez la constante tenue à jour de OLDTXT et CURLIN en mode programme.

EVALUATION D'EXPRESSIONS - FRMEVL

1 ;

Certaines commandes sont suivies d'une expression numérique ou alphanumérique.

L'évaluateur d'expressions FRMEVL ou/et les routines associées (FRMNUM, GETADR etc) sont appelés, le cas échéant, par la routine de traitement de la commande elle-même.

Vous le constaterez pour les commandes: DOKE, UNTIL, FOR, LPRINT, LET, GOTO, IF..THEN, GOSUB, HIMEM, commandes HGR et SON (pour recueil des paramètres), WAIT, CLOAD, CSAVE, DEF, POKE, PRINT, CALL.

L'opération d'évaluation menée par FRMEVL est extrêmement difficile à suivre.

La complexité vient de ce qu'elle concerne aussi bien les chaînes que les nombres et aussi de l'usage intensif de la pile qui est fait. J'ai essayé de donner quelques points de repère au chapitre 18.

Le résultat de l'évaluation est laissé en FAC. Pour une chaîne, ce sera le pointeur d'adresse du descripteur. Voir chapitre 14, manipulation de chaines de caractères.

Si le code d'une fonction ou d'un opérateur est rencontré au cours de l'opération, l'adresse correspondante est cherchée dans la table en #C08A ou en #C08C.

Pour un opérateur, l'adresse est mise en pile et recueillie ensuite par un RTS.

Pour une fonction, l'adresse est inscrite en #C4,C5 et l'entrée dans la routine de traitement se fait par JSR #C3.

FRMEVL peut être invoqué à nouveau par ces diverses routines. Le processus devient souvent très compliqué par ses imbrications.

Rappelez-vous qu'après exécution de l'instruction complète, le programme revient à NEWSTT pour traiter une nouvelle instruction.

Vous pouvez examiner les sections de code suivantes, chargées de la détermination des points d'entrée et du branchement approprié: ATMOS #CF99-CFD2 ORIC-1 #CF0D-CF46 (opérateurs) ATMOS #D0A0-D0E2 ORIC-1 #D014-D056 (fonctions)



CHAPITRE 12

ROUTINES FONDAMENTALES

BOUCLE PRINCIPALE D'ENTREE DES COMMANDES - CMDLP

Entrée de l'interpréteur.

ATMOS #C4B7-#C55E

ORIC-1 #C4C7-#C56E

Cette routine n'est pas utilisable en langagé machine car elle boucle sur elle-même, pas de retour.

Elle mérite cependant d'être étudiée en raison de son importance. En attente après l'affichage du message READY, elle détecte toute entrée au clavier et la traite en conséquence.

L'information entrée est analysée et codée.

S'il s'agit d'une commande directe, il y a branchement dans NEWSTT. Sinon, les lignes de programme sont rangées en mémoire.

Sous-routines utilisées par CMDLP:

	ATMOS	ORIC-1		ATMOS	ORIC-1
INLIN	#C592	#C5A2	SETPTRS	#C708	#0733
PARSE	#C5FA	#C6ØA	LINKSET	#C55F	#C56F
LINGET	#CAE2	#CA98	BLTU	#C3F4	#C3F8
FNDLIN	#C6B3	#C6DE			

ATMOS ORIC-1

1.	C4B7	C4C7	CMDLP	JSR	VHTAB	parm sort.écran, ORIC-1 LSR PRTFLG
2.	C4BA	C4CA		JSR	INLIN	recueil ligne entrée au clavier
3.	C4BD	C4CD		STX	TXTPTR	TXTPTR=BUF-1
4.	C4BF	C4CF		STY	TXTPTR+1	(X=#34, Y=#00 au retour de INLIN)
5.	C4C1	C4D1		JSR	CHARGET	inc.TXTPTR, recueille ier carac.
6.	C4C4	C4D4		TAX		transféré dans X (voir note)
7.	C4C5	C4D5		BEQ	CMDLP	entrée RETURN seul, on recommence.
8.	C4C7	C4D7		LDX	>#FF	pour mode immédiat éventuel, #FF
9.	C4C9	C4D9		STX	CURLIN+1	dans octet fort de CURLIN.
10.	C4CB	C4DB		BCC	NXLIN	C=0, carac = 1er chiffre n°ligne
11.	C4CD	C4DD		JSR	PARSE	mode immédiat, analyse, codage et
12.	C4DØ	C4EØ		JMP	EXEC	exécution (voir note)
13.	C4D3	C4E3	NXLIN	JSR	LINGET	lit n° ligne (LINNUM), inc.TXTPTR.
14.	C4D6	C4E6		JSR	PARSE	analyse et codage
15.	C4D9	C4E9		STY	TEMP	index BUFF (nb.carac.+5)
16.	C4DB	C4EB		JSR	FNDLIN	la ligne existe-elle déja?
17.	C4DE	C4EE		BCC	NEWLN	non (C=0), nouvelle ligne.
18.	C4EØ	C4FØ	DEL	LDY	>#Ø1	oui,
19.	C4E2	C4F2		LDA	(LOWTR),Y	supression
20.	C4E4	C4F4		STA	INDEX+1	ancienne
21.	C4E6	C4F6		LDA	VARTAB	ligne
22.	C4E8	C4F8		STA	INDEX	par
23.	C4EA	C4FA		LDA	LOWTR+1	déplacement
24.	C4EC	C4FC		STA	DEST+1	du
25.	C4EE	C4FE		LDA	LOWTR	reste
26.	C4FØ	C5ØØ		DEY		du

```
27. C4F1 C501
                    SBC (LOWTR), Y ....programme
28. C4F3 C503
                    CLC
                                  ....vers
29. C4F4 C5Ø4
                    ADC VARTAB
                                  ....le
30. C4F6 C506
                   STA VARTAB
                                  ....bas.
31. C4F8 C508
                   STA DEST
                                  .... Ecrasement
32. C4FA C5ØA
                   LDA VARTAB+1
                                  ....de
33. C4FC C5ØC
                   ADC >#FF
                                  ....la
34. C4FE C5ØE
                   STA VARTAB+1
                                  ....ligne
                   SBC LOWTR+1
TAX
35. C500 C510
                                  ....à
36. C502 C512
                                  ....supprimer
37. C503 C513
                    SEC
                                  .....
38. C504 C514
                    LDA LOWTR
                                  ....pointeurs utilisés
                   SBC VARTAB
39. C506 C516
                                  ....INDEX #91,92
40. C508 C518
                    TAY
                                  ....DEST
                                              #93.94
41. C509 C519
                   BCS NL1
                                  ....VARTAB #9C,9D
42. C50B C51B
                                  ....LOWTR
                   INX
                                              #CE,CF ·
43. C50C C51C
                   DEC DEST+1
44. C50E C51E NL1
                   CLC
45. C50F C51F
                    ADC INDEX
46. C511 C521
                   BCC MVDWN
47. C513 C523
                   DEC INDEX+1
                                  ....
48. C515 C525
                    CLC
49. C516 C526 MVDWN LDA (INDEX), Y déplacement
50. C518 C528 STA (DEST),Y du
51. C51A C52A
                    INY
                                  reste
52. C51B C52B
                    BNE MVDWN
                                  du
53. C51D C52D
                    INC INDEX+1
                                  programme
54. C51F C52F
                    INC DEST+1
                                 vers
55. C521 C531
                   DEX
                                 le
56. C522 C532
                   BNE MVDWN
                                 bas
57. C524 C534 NEWLN JSR SETPTRS met TXTPTR début de programme pour
58. C527 C537
                    JSR LINKSET positionnement octets de chaînage
59. C52A C53A
                               si pas de caractère après ...
                    LDA BUFF
60. C52C C53C
                   BEQ CMDLP
                                  n°ligne, pas d'insertion.
61. C52E C53E
                   CLC
62. C52F C53F
                   LDA VARTAB
                                  positionne paramètres pour BLTU
63, C531 C541
                                 ....pointeurs utilisés
                   STA HIGHTR
64. C533 C543
                   ADC TEMP
                                  .....VARTAB #9C,9D
                                 ....HIGHTR #C9,CA
....HIGHDS #C7,C8
....STREND #A0,A1 (sortie BLTU)
65. C535 C545
                   STA HIGHDS
66. C537 C547
                   LDY VARTAB+1
67. C539 C549
                   STY HIGHTR+1
68. C53B C54B
                   BCC +1
                                  ....TEMP (#26) contient..
69. C53D C54D
                   INY
                                  ....longueur de la ligne..
70. C53E C54E
                   STY HIGHDS+1
                                 ....à insérer.
71. C540 C550 MOVUP JSR BLTU
                                  déplacement queue du programme
72. C543 C553
                   LDA STREND
                                  ajuste..
73. C545 C555
                   LDY STREND+1
                                  . .
74. C547 C557
                   STA VARTAB
                                  VARTAB
75. C549 C559
                   STY VARTAB+1
                  LDY TEMP
76. C54B C55B
                                  longueur de la ligne...
77. C54D C55D
                   DEY
                                  -1 sert d'index pour insertion..
78. C54E C55E INSER LDA (#31),Y
                                  de (BUFF-4),Y à adresse ...
79. C551 C561 STA (LOWTR), Y pointée par LOWTR.
8Ø. C553 C563
                   DEY
                                 ... Note: BUFF-4 pour loger en-tête
81. C554 C564
                   BPL inser
                                  ...n°ligne déja en place (#33,34)
82. C556 C566
                  JSR SETPTRS
                                 TXTPTR=TXTTAB-1
                 JSR LINKSET mise en place octets de chaînage.
JMP CMDLP retour CMDLP pour nouvelle entrée.
83. C559 C569
84. C55C C56C
```

Note:

ligne 6: à première vue cette instruction est superflue. Elle permet cependant l'exécution d'une commande directe précédée de deux-points, marqueur de fin d'instruction.

En effet, au retour de CHARGET, l'indicateur Z est mis à un si l'accumulateur contient le code #3A ':', fin d'instruction ou #00, fin de ligne.

TAX reconditionne l'indicateur Z, et le branchement qui suit, retour à CMDLP, ne sera pris que si la valeur 00 a été recueillie.

ligne 12: une instruction (ou un groupe d'instructions) entrée en commande directe est exécutée immédiatement, entrée dans NEWSTT à EXEC (voir chapitre 11).

ENTREE D'UNE LIGNE DE TEXTE - INLIN

Cette routine est appelée à chaque fois qu'une entrée est demandée par l'interpréteur.

Elle peut être utilisée au cours de l'exécution d'un programme (INPUT).

En temps normal (Ready), elle recueille dans le buffer d'entrée une instruction en commande directe ou une ligne de programme à stocker.

L'information est recueillie caractère par caractère par appel de INCHR et rangée dans le buffer d'entrée avec écho à l'écran.

Vous remarquerez les précautions prises pour ne pas dépasser la capacité relativement limitée du buffer d'entrée (78 caractères).

ATMOS ORIC-1

			DEL			retour en arrière
2.	C58D	C59D		BPL	INL+2	si non début de ligne
3.	C58F	C59F	INL-3	JSR	CRDO	ret. à la ligne, nouvelle entrée
4.	C592	C5A2	INLIN	LDX	>#ØØ	X = index BUF
						caractère en A
6.	C597	C5A7		CMP	'ctrl-A'	CTRL-A?
7.	C599	C5A9		BNE	affich	non
8.	C59B	C5AB		LDY	CURCOL	oui, lit caractère
					(CURBAS),Y	
	C5AØ					ASCII positif
11.	C5A2	C5B2		CMP	ti n	caractère de contrôle?
12.	C5A4	C5B4		BCS	affich	non
13.	C5A6	C5B6		LDA	ctrl-I	oui, affiche ->
14.	C5A8	C588	affich	PHA		sauv carac
15.	C5A9	C5B9		JSR	OUTDO	affichage
16.	C5AC	C5BC		PLA		recup carac
17.	C5AD	C5BD		CMP	'del'	effacement?
18.	C5AF	C5BF		BEQ	del	oui
19.	C5B1	C5C1		CMP	'return'	RETURN?
20.	C5B3	C5C3		BEQ	FIN	oui, fin d'entrée
21.	C5B5	C5C5		CMP	'ctrl-C'	CTRL-C?
22.	C587	C5C7		BEQ	ctl-C	oui, annule l'entrée
23.	C5B9	C5C9		CMP	'ctrl-X'	CTRL-X?
24.	C5BB	C5CB		BEQ	annul	oui

- NAME OF THE OWNER	The second second					
	C5BD			CMP		autre carac de contrôle?
	C5BF				INL+2	oui, ignorer
	C5C1				BUF,X	caractère rangé dans buffer
	C5C3			INX		incrémente index
	C5C4				>#4F	>78 caractères?
	C5C6				>75?	non, voir si >75 carac
			annul			oui
	C5CA				OUTDO	affiche , annule l'entrée.
	C5CD				INL-3	nouv. entrée (branch forcé)
			>75?		>#4C	>75 caractères?
	C5D1			BCC	INL+2	non, caractère suivant
	C5D3			TXA		oui
37.	C5D4	C5E4		PHA		sauvegarde
38.	C5D5	C5E5		TYA		registres
39.	C5D6	C5E6		PHA		X et Y
40.	C5D7	C5E7		JSR	PING	cloche
41.	C5DA	C5EA		PLA		restaure
42.	C5DB	C5EB		TAY		registres
43.	C5DC	C5EC		PLA		Yet
44.	C5DD	C5ED		TAX		X
45.	C5DE	C5EE		JMP	INL+2	caractère suivant
						The second secon
46.	C5E1	C5F1	Ct1-C	INC	#17	drapeau, sert pour INPUT
47.	C5E3	C5F3		LDX	>#@@	annule l'entrée, 00 en #35
48.	C5E5	C5F5	FIN	JMP	EOL	fin de ligne.
						•
49.	C5E8	C5F8	INCHR	JSR	RDKEY	lecture mémoire clavier
50.	C5EB	C5FB		BPL	INCHR	si pas de touche enfoncée
51.	C5ED	C5FD		CMP	'ctrl-0'	CTRL-0?
52.	C5EF	C5FF		BNE	RTS	non, retour carac en A
53.	C5F1	C6Ø1		PHA		oui,
54.	C5F2	C602			#2E	bascule
	C5F4	100 CO.			>#FF	inhibition
	C5F6				#2E	écran
	C5F8			PLA		property of the second of the
	C5F9			RTS		retour
von dien dien	1000 March 1987					Part of the Control o

ANALYSE DU BUFFER D'ENTREE CODAGE DES MOTS-CLES - PARSE

Cette routine analyse *le texte* Basic entré au clavier, rangé dans le buffer d'entrée, qu'il s'agisse d'une commande directe ou d'une ligne de programme.

Dans le premier cas, à l'entrée dans la routine, le pointeur de texte est en #35, au début du buffer.

Dans le second, TXTPTR est à l'emplacement où l'a laissé LINGET après la lecture du numéro de ligne.

Cela ne dépend pas seulement du nombre de chiffres.

LINGET utilise CHARGET et sort par cette routine qui recueille le caractère autre qu'un espace (elle les ignore) suivant le numéro de ligne.

Si ce dernier est suivi d'espaces, le pointeur de texte sera déplacé d'autant dans le buffer d'entrée.

Profitant de cette parenthèse, vous pouvez faire l'expérience suivante.

Entrez une ligne de programme dont vous placerez les chiffres du

numéro aux quatre coins de l'écran, séparés par des blancs. Ils seront recollés par LINGET.

Une autre remarque. L'espace qui apparaît à l'écran après un numéro de ligne est affiché par LIST. Il n'existe pas en mémoire.

La routine PARSE intervient sur le texte lui-même dans le buffer d'entrée.

Elle ignore, en fait, s'il s'agit de mode direct ou de mode différé. Elle n'utilise pas CHARGET, tous les espaces inclus dans la ligne sont conservés.

Le nouveau texte, codé, est replacé au fur etimesure dans le même buffer d'entrée à partir de #35 (NEWBUF).

C'est possible car sa longueur est toujours inférieure (ou égale) à celle du texte original (OLDBUF).

ATMOS ORIC-1

1 -	C5FA	C60A	PARSE	LDX	TXTPTR	X = index OLDBUF
	C5FC				>#Ø4	Y = index NEWBUF
3.	C5FE	C40E		STY	DATAFLG	drapeau baissé (bit 6 à zéro)
			GETINP			recueille caractère dans OLDBUF
	C602			CMP		espace?
6.	C6Ø4	C614		BEQ	PUTBUF	oui, stocker
6.	C606	C616			ENDCHR	drapeau littéral si (A)='"'
7.	C408	C618		CMP	* n *	guillemet?
8.	C60A	C61A		BEQ	PUTLIT	littéral, entrer tel quel
9.	C40C	C61C		BIT	DATAFLG	bit 6 DATAFLG à un?
10.	C40E	C61E		BVS	PUTBUF	oui, DATA, entrer tel quel
11.	C610	C620		CMP	. 3 .	symbole de PRINT?
12.	C612	C622		BNE	+4	nan
13.	C614	C624		LDA	'print'	oui, substituer code PRINT (#BA)
14.	C616	C626		BNE	PUTBUF	et stocker (branchement forcé)
15.	C618	C628		CMP	, Q ,	chiffres de Ø à 9,
15.	C61A	C62A		BCC		ni deux-points, ni point-virgule
16.	C61C	C62C		CMP	' < '	ne peuvent etre ler caractère
17.	C61E	C62E		BCC	PUTBUF	d'un mot-clé. On stocke
18.	C620	C630		STY	STRNG2	sauvegarde index NEWBUF
19.	C622	C632		LDY	>#00	Y = index table des mots-clés
20.	C624	C634		STY	TEMP	TEMP + #80 = code mot-clé
21.	C626	C636		LDA	>#E9	adresse table
22.	C628	C938		STA	#18	des mots-clés - 1
23.	C62A	C63A		LDA	>#CØ	dans pointeur
24.	C62C	C63C		STA	#19	#18,19 (post-indexation par Y)
25.	C62E	C43E		STX	TXTPTR	sauvegarde index OLDBUF
26.	C630	C64Ø		DEX		ajuste index pour boucle recherche
27.	C631	C641	KEYCHR	INX		boucle recherche concordance
28.	C632	C642		INC	#18	(#18,19 pointe début table
29.	C634	C644		BNE	+2	en #CØEA
30.	C636	C646		INC	#19	pour la 1ere passe)
31.	C938	C648	SETTKN	LDA	#ØØ,X	recueille caractère OLDBUF
32.	C63A	C64A		SEC		et le compare
33.	C93B	C64B		SBC	(#18),Y	au caractère de la table
34.	C63D	C64D		BEQ	KEYCHR	concordance, caractères suivants
35.	C63F	C64F		CMP	>#80	pas bon, sans doute dernier carac?
36.	C641	C651		BNE	NXKEY	non, essayons mot-clé suivant.
	C643				TEMP	tout bon, code=TEMP + #80 (A)
			PUTTKN		STRNG2	restaure index NEWBUF
			PUTBUF	INX		incrémente index OLDBUF
40.	C648	C658		INY		incrémente index NEWBUF

41. C649 C659		BUF-5,Y	stockage carac en A dans NEWBUF
42. C64C C65C		BUF-5,Y	positionnne indicateur Z
43. C64F C65F		FIN	si zéro, fin de la ligne entrée
44. C651 C661	SEC		tests
45. C652 C662		' : '	fin d'instruction?
46. C654 C664	BEQ	DTFLG	oui, baisse drapeau DATAFLG
47. C656 C666	CMP	data-':'	DATA? (#91-#3A=#57)
48. C658 C668	BNE	CHKREM	non, voir si REM
49. C65A C66A D	TFLG STA	DATAFLG	=00 si fin instr. =#57 si DATA
50. C65C C66C C	HKREM SEC		test
51. C65D C66D	SBC	rem-':'	REM? (#9D-#3A=#63)
52. C65F C66F	BNE	GETINP	non, on continue
53. C661 C671	STA	ENDCHR	=00, drapeau littéral baissé
54. C663 C673 GE	ETLIT LDA	#00,X	recueille caractère OLDBUF
55. C665 C675	BEQ	PUTBUF	fin de ligne
56. C667 C677	CMP	ENDCHR	guillemet?
57. C669 C679	BEQ	PUTBUF	oui, fin de littéral
58. C66B C67B PL	UTLIT INY		incrémente index NEWBUF
59. C66C C67C	STA	BUF-5,Y	stocke dans NEWBUF
60. C66F C67F	INX	•	incrémente index OLDBUF
61. C670 C680		GETLIT	branchement forcé
62. C672 C682 N		TXTPTR	restaure index OLDBUF
63. C674 C684		TEMP	incrémente indicateur code mot-clé
64. C676 C686 NR		(#18),Y	recueille caractère dans la table
65. C678 C688	PHP	•	sauvegarde P pour le signe
66. C679 C689		#18	pointe
67. C67B C68B	BNE		caractère
68. C67D C68D		#19	suivant
69. C67F C68F	PLP		recupère P
70. C680 C690		NKY1	positif, autre caractère
71. C682 C692		(#18),Y	ier caractère du mot-clé suivant
Tet dode days			THE CAN MELLI & MA MOL LIE SHITAITE
ATMOS			
72. C684	BNE	SETTKN	n'est pas la fin de table (00)
73			
ORIC-1			
72. C694	BEQ	+3	fin de table
73. C696	JMP	SETTKN	saut inconditionnel
ATMOS ORIC-1			
74. C686 C699		#ØØ,X	fin de table, le carac n'était pas
75. C688 C69B		PUTTKN	celui d'un mot-clé, branch. forcé.
76. C68A C69D FI		BUF-3,Y	place 00, 2 oct au-delà fin ligne
77. C68D C6AØ	LDA	BUF-1	retour avec TXTPTR
78. C68F C6A2	STA	TXTPTR	pointant sur BUF-1 (#34)
79. C691 C6A4	RTS		retour (CMDLP)

Note:

ligne 76: la valeur 00 est inscrite deux octets au delà de la fin de ligne dans le buffer d'entrée.

Cette valeur n'est pas recopiée dans la zone programme en mode différé.

Elle sert seulement en mode immédiat.

Une ou plusieurs instructions en commande directe, forment une espèce de mini-programme dans le buffer d'entrée, traité directement par NEWSTT qui en détecte ainsi la fin, de la même façon qu'en zone programme.

- LIST

ATMOS #C748-#C7CF

ORIC-1 #C773-#C7F6

La routine LIST effectue l'opération de décodage, inverse de celle réalisée par PARSE. Elle utilise aussi la table des mots-clés en #CØE9 Elle prend les lignes codées en zone programme, les décode et les affiche en clair à l'écran, numéro de ligne en décimal et mots-clés en toutes lettres.

Cette routine sort normalement par Ready2, sauf si elle est appelée par EDIT qui positionne à un le bit 7 du drapeau en #2F2. On peut utiliser cette particularité dans un programme Basic. La commande LIST ne provoquera pas un arrêt du programme si elle est précédée de l'instruction POKE#2F2,128 (exemple, page 30).

ATMOS ORIC-1

1.	C748	C773	LIST	PHP		sauvegarde registre d'état
2.	C749	C774		JSR	LINGET	n° 1ere ligne dans LINNUM
3.	C74C	C777		JSR	FNDLIN	adresse dans LOWTR
4.	C74F	C77A		PLP		restaure P
5.	C750	C77B		BEQ	ENDMAX	LIST seul, pas de limites
6.	C752	C77D		JSR	CHARGOT	quoi après LINGET?
7.	C755	C78Ø		BEQ	NXLN	rien, 1 seule ligne (LINNUM)
8.	C757	C782		CMP	'-'	trait d'union?
9.	C759	C784		BNE	RTS	non, SYNTERR déclarée par NEWSTT
10.	C75B	C786		JSR	CHARGET	oui, caractère suivant
11.	C75E	C789		BEO	ENDMAX	rien après -
12.	C76Ø	C78B		JSR	LINGET	n° limite max?
13.	C763	C78E		BEQ	NXLN	-> LINNUM, forcé si rien après n°
14.	C765	C790		RTS		retour NEWSTT, erreur syntaxe
15.	C766	C791	ENDMAX	LDA	>#FF	inscrit
16.	C768	C793		STA	LINNUM	valeur max (65535)
	C76A			STA	LINNUM+1	dans LINNUM
18.	C74C	C799	NXLN	LDY	>#Ø1	Y = index dans programme
19.	C76E	C79B		LDA	(LOWTR),Y	test octet de chaînage fort
20.	C77Ø	C79D		BEQ	FIN	si 00, fin de programme
21.	C772	C79F			ISCNTC	CTRL-C?, si oui arrêt listing
22.	C775	C7A2		CMP	B O	espace?
23.	C777	C7A4		BNE	L1	non, continuer
	C779				ICHAR	strobe mémoire clavier
25.	C77C	C7A9	PAUSE			touche enfoncée?
26.	C77F	C7AC		BPL	PAUSE	non, attendre (diff sur ORIC-1)
	C781			JSR	ISCNTC	oui, est-ce CTRL-C?
28.	C784			LSR	ICHAR	non, continue listage
29.	C787	C7AE	Li	INY		recueil
	C788			LDA	(LOWTR),Y	
31.	C78A	C7B1		TAX		octet faible dans X
32.	C78B	C7B2		INY		actet fort
33.	C78C	C783		LDA	(LOWTR),Y	dans A
	C78E			CMP	LINNUM+1	comparé
35.	C79Ø	C7B7		BNE	LIM	à
	C792			CPX	LINNUM	n° ligne
37.	C794	C7BB		BEQ	PRTNR	limite

```
38. C796 C7BD LIM BCS FIN
                                  limite atteinte
39. C798 C7BF PRTNR STY FORPNT
                                  sauvegarde index ligne
40. C79A C7C1
                    PHA
                                  retour
41. C79B C7C2
                    JSR CRDO
                                  àla
42. C79E C7C5
                    PLA
                                  ligne
43. C79F C7C6
                    JSR LINPRT
                                  affiche n° de ligne décimal
44. C7A2 C7C9
                    LDA " "
                                  espace après n° de ligne
45. C7A4 C7CB LSTLN LDY FORPNT
                                  restaure index ligne
46. C7A6 C7CD
                    AND >#7F
47. C7AB C7CF PRTCHR JSR OUTDO
                                  affichage caractère
48. C7AB C7D2
                    INY
                                  caractère
49. C7AC C7D3
                    BEQ FIN -
50. CTAE C7D5
                    LDA (LOWTR),Y suivant
51. C780 C7D7
                    BNE TSTCHR
                                  pas fin de ligne, à tester
52. C7B2 C7D9
                    TAY
                                  fin de ligne, index = 0
53. C7B3 C7DA
                    LDA (LOWTR), Y octet chainage faible
54. C7B5 C7DC
                                  dans X
                    TAX
55. C7B6 C7DD
                    INY
                                  octet chainage fort
56. C7B7 C7DE
                    LDA (LOWTR),Y dans A
57. C7B9 C7E0
                   STX LOWTR
                                  transférés dans
58. C7BB C7E2
                   STR LOWTR+1
                                  LOWTR
59. C7BD C7E4
                   BNE NXLN
                                  pour ligne suivante, branch forcé
                  BIT #2F2
60. C7BF C7E6 FIN
                                  drapeau levé par EDIT?
61. C7C2 C7E9
                    BPL OUT
                                  sortie Ready2
62. C7C4 C7EB
                    RTS
                                  retour progr. appelant (EDIT)
63. C7C5 C7EC OUT
                    JSR CRDO
                                  à la ligne
64. C7C8 C7EF
                    JSR VHTAB
                                  parm. sort.écran, ORIC-1 LSR PRTFLE
65. C7CB C7F2
                    PLA
                                  POP
66. C7CC C7F3
                    PLA
                                  adresse de retour
                    JMP Ready2
67. C7CD C7F4
                                  sortie
68. C7DØ C7F7 TSTCHR BPL PRTCHR
                                  n'est pas un mot-clé, afficher
69. C7D2 C7F9
                                  code mot-clé - #7F
                    SEC
70. C7D3 C7FA
                    SBC >#7F
                                  ...= index
71. C7D5 C7FC
                    TAX
                                   ...dans table des mots-clés
72. C7D6 C7FD
                    STY FORPNT
                                  sauvegarde index ligne
73, C7D8 C7FF
                    LDY >#ØØ
                                  pour adr.indirect,Y
                    LDA >#E9
74. C7DA C801
                                  adresse table
75. C7DC C803
                    STA #18
                                  des
76. C7DE C805
                    LDA >#CØ
                                  mots-clés (#CØE9)
77. C7EØ C8Ø7
                    STA #19
                                  dans pointeur #18,19
78. C7E2 C8Ø9 NXKEY DEX
                                  décr.index, quand (X)=0
79. C7E3 C8ØA
                   BEQ PRTKY
                                    ..mot-clé localisé
80. C7ES C80C N1
                    INC #18
                                  boucle
81. C7E7 C8ØE
                   BNE +2
                                  positionnement
82. C7E9 C810
                    INC #19
                                  sur
83. C7EB C812
                    LDA (#18),Y
                                  mot-clé suivant
                    BPL NI
84. C7ED C814
                                  pas trouvé
85. C7EF C816
                    JMP NXKEY
                                  mot-clé suivant
86. C7F2 C819 PRTKY INY
                                  caractère mot-clé
87. C7F3 C81A
                   LDA (#18),Y chargé dans A pour affichage
                   BMI LSTLN
                                 si nég.(dernier carac)
88. C7F5 C81C
89. C7F7 C81E
                    JSR OUTDO
                                  affichage
90. C7FA C821
                   JMP PRTKY
                                  carac suivant, non dernier
```

- EDIT

ATMOS #C692-C6B2

ORIC-1 #C6A5-C6DD

Nous n'entrerons pas dans le détail de cette routine, légérement différente selon le système. Vous devriez pouvoir la démonter sans difficultés.

Elle utilise LIST en partie, entrée à NXLN (ligne 18 ci-dessus) Vous remarquerez la mise à un du bit 7 du drapeau #2F2 évoquée plus haut pour empêcher la sortie après listage de la ligne.

Il faut reconnaitre que cette fonction d'édition d'une ligne Basic est assez rudimentaire sur ORIC.

Elle n'est pas facile à réaliser à cause de la protection des deux premières colonnes de l'écran.

Une solution, pas très satisfaisante non plus, est présentée au chapitre 15.



CHAPITRE 13

NOMBRES A VIRGULE FLOTTANTE

Nous avons vu que le 6502, comme les autres microprocesseurs en général, était singulièrement limité en ce qui concerne le traitement des nombres.

Il ne reconnait que les entiers sur 8 bits, éventuellement signés, c'est-à-dire compris entre Ø et 255 ou -128 et +127 décimal. Les opérations autres que la simple addition ou soustraction binaire sur un octet doivent être programmées.

Des routines arithmétiques sont généralement incluses dans le logiciel de base de la machine qui donne le langage Basic.

Certains Basic simples permettent de traiter des nombres entiers, signés ou non, compris entre Ø et 65535 ou entre -32768 et +32767.

Les Basics étendus, tel le Microsoft de l'ORIC, en sont capables aussi, bien sûr, mais ils comportent, en plus, des programmes qui permettent d'étendre considérablement les possibilités de la machine dans le domaine du calcul.

C'est ainsi que votre ordinateur reconnait et manipule des nombres dits réels, entiers ou fractionnaires, positifs ou négatifs, compris entre -1.70141183E+38 et +1.70141183E+38.

Pour arriver à ce résultat, un autre système de représentation a dû être élaboré: le système de représentation en virgule flottante.

Comme tout système, il est bâti sur des conventions que nous allons examiner.

Dans tout ce qui va suivre nous parlerons de la virgule mais vous savez qu'elle est représentée par le point décimal anglo-saxon.

REPRESENTATION DES NOMBRES REELS

Les nombres réels sont représentés sur cinq octets par le Basic de l'ORIC.

Si vous vous reportez à la page 32, vous constatez, par exemple, que la valeur 123.456 est rangée en mémoire sous la forme:

87 76 E9 78 D5

en binaire: 10000111 01110110 11101001 01111000 11010101

exposant >---- mantisse ----

Notre propos est de voir comment, partant du nombre décimal 123.456, on arrive à cette représentation à l'intérieur de l'ordinateur.

CONVERSION

La conversion manuelle en binaire est fastidieuse mais simple. Elle consiste en une série de comparaisons avec les puissances de deux successives.

			chiffre binaire	reste
123.456	<	2^7 = 128		123.456
123,456	>	$2^6 = 64$	1	59.456
59.456	>	$2^5 = 32$	1	27.456
27.456	>	$2^4 = 16$	1	11.456
11.456	>	$2^3 = 8$	1	3.456
3.456	<	$2^2 = 4$	Ø	3,456
3.456	>	$2^1 = 2$	1	1.456
1.456	>	$2^0 = 1$	1	0.456
virgule			•	
Ø. 456	<	$2^{-1} = 0.5$	Ø	0.456
0.456	>	$2^-2 = 0.25$	1	0.206
0.206	>	$2^{-3} = 0.125$	1	0.081
0.081	>	$2^{-4} = 0.0625$	1	0.0185
0.0185	<	$2^{-5} = 0.0312$	5 0	0.0185
0.0185	>	$2^{-6} = 0.0156$	25 1	0.002875

et ainsi de suite pour obtenir la précision souhaitée.

Prenez le temps de taper le petit programme de conversion qui suit, identique pour ATMOS ou ORIC-1, et de noter ces remarques.

- Le nombre binaire affiché est de 32 chiffres (32 bits). Cela correspond à la précision offerte par le Basic de l'ORIC.

- Le programme n'est pas véloce.

Il est handicapé par la bogue regrettable qui affecte l'opération d'élévation à une puissance sur ORIC (corrigée, lignes 20 et 30). Des décimales indésirables altérent le résultat pour certains exposants. Ce défaut n'a malheureusement pas été supprimé sur ATMOS. Vous pouvez le constater avec la simple boucle:

FOR I=0 TO 23:PRINT 2^I:NEXT

ou cette autre

FOR I=0 TO 23:PRINT EXP(LN(2)*I):NEXT

(le défaut n'est que partiellement éliminé)

```
10 CLS: GOTO 100
20 Q=INT(2^ABS(P)): IF P<0 THEN Q=1/Q
30 RETURN
40 IF D<Q THEN B$=B$+"0":GOTO 60
50 D=D-Q:B$=B$+"1": IF LEN(B$)>33 THEN 80
60 P=P-1: IF P=-1 THEN B$=B$+"."
70 GOSUB 20: IF LEN(B$) <34 THEN 40
80 PRINT B$:PRINT: X=FRE("")
100 INPUT"decimal"; D$: D=VAL(D$): IF D=0 THEN B$="0": GOTO 80
110 IF D>0 THEN B$="+":GOTO 130
120 B$="-":D=ABS(D)
130 P=-1:GOSUB 20:IF D<1 THEN B$=B$+".":GOTO 40
140 IF D>Q THEN P=P+1:GOSUB 20:GOTO 140
150 IF DKQ THEN P=P-1:GOSUB 20:GOTO 150
160 GOTO 40
RUN
```

Entrez un nombre décimal, positif ou négatif, entier ou non. Pour sortir du programme, CTRL-C.

NORMALISATION

Revenons à notre valeur décimale témoin, 123.456. Convertie en binaire:

+1111011.01110100101111100011010101

Deux problèmes se posent pour le stockage en mémoire.

Comment conserver trace de la virgule et du signe, symboles inconnus en binaire?

Occupons-nous d'abord de la virgule.

La solution est inspirée de la notation scientifique et utilise mantisse et exposant.

En décimal, les notations suivantes

0.123456 * 10^3 123456 * 10^-3 ou 1.23456E+2 (Basic)

désignent le même nombre 123.456

On peut mettre un nombre binaire sous une forme analogue, à condition de le normaliser.

La virgule ne pouvant pas être représentée occupera toujours la même place, fictive, dans la mantisse.

Sa position réelle sera déterminée par la valeur de l'exposant ajusté en conséquence, d'où le nom de virgule flottante.

Par convention, elle est située à gauche du bit le plus significatif à un de la mantisse, les décalages nécessaires étant effectués. Ainsi, la valeur de la mantisse est comprise entre .1 et 1.

.111101101110100101111100011010101 * 10^111

L'exposant 10^111 est en binaire.

· ... i

Rappelez-vous, 10 (base 2) = 2 décimal et 111 (base 2) = 7 décimal.

L'exposant étant positif, il détermine un décalage de la virgule, de 7 positions vers la droite.

Un exposant négatif la décalerait à gauche.

Dans notre exemple tous les signes sont positifs mais faisons-les figurer.

+.111101101110100101111100011010101 * +10^(+111)

Plaçons l'exposant en tête, à sa place normale

+10^(+111) * +.111101101110100101111100011010101

Maintenant, à partir de la droite, séparons le nombre proprement dit (la mantisse) et sa caractéristique (l'exposant) en cinq groupes distincts.

+10^(+111) +.11110110 11101001 01111000 11010101 ?? ?? E9 78 D5

On reconnait, dans les trois derniers groupes, les trois derniers octets du nombre réel 123.456, tels qu'ils sont stockés par l'ORIC.

119

SIGNES DU NOMBRE ET DE L'EXPOSANT

Les deux premiers groupes sont à mettre au format et il reste le problème des signes qui doivent être inclus quelque part.

Dans le premier groupe, +10 est superflu.

Cette valeur est constante quel que soit le nombre considéré. Supprimons-la et complétons l'octet avec des Ø. Le potentiel de l'exposant en sera augmenté.

+00000111 +.11110110 11101001 01111000 11010101

Dans le deuxième groupe, la virgule est inutile. Elle est toujours à cet endroit pour un nombre normalisé.

De même, après normalisation, le premier bit, à gauche, est toujours à un.

Il n'est pas nécessaire d'en garder trace.

Cet emplacement servira au bit de signe.

O pour un nombre positif (c'est ici le cas) ou 1 pour un nombre négatif.

+00000111 01110110 11101001 01111000 11010101
?? 76 E9 78 D5

Revenons au premier groupe, octet de l'exposant. Ici encore le bit le plus significatif est utilisé pour indiquer le signe. Dans le cas présent, l'exposant est positif, ce sera 0.

200000111 01110110 11101001 01111000 11010101 ?? 76 E9 78 D5

Nous n'avons pas encore donné sa valeur définitive à l'octet n°1 car il reste une opération à effectuer.

Cette dernière modification n'est pas si évidente.

Elle est liée aux particularités de fonctionnement des routines en ROM.

La valeur de l'exposant ayant été déterminée, il faut lui ajouter #80 (en ignorant la retenue éventuelle).

Cela revient à inverser le bit n°7 de l'octet.

La caractéristique du nombre aura pour valeur #80 + EXP.

On dit quelquefois que l'exposant est sous la forme 'en excès de #80'. (in excess #80 form).

10000111 01110110 11101001 01111000 11010101 87 76 E9 78 D5

Un nombre réel est rangé en mémoire sous cette forme que l'on qualifie de condensée.

Nous verrons plus loin que lorsque le nombre est transféré dans un registre flottant (FAC ou ARG), en vue d'un calcul, le bit n°7 du premier octet de la mantisse est remis à un et le signe est conservé dans un sixième octet.

Voici, résumées en forme de check-list, les opérations à effectuer

pour convertir un nombre décimal en nombre binaire à virgule flottante sur cinq octets.

- 1. convertir le nombre en binaire en veillant à bien tenir compte du signe et de la position de la virgule.
- normaliser le nombre sur 32 bits, le bit le plus à gauche doit être à un, et passer à la notation scientifique.
- 3. placer l'exposant devant la mantisse et séparer en cinq groupes. Conserver les signes.
- 4. supprimer la valeur binaire 10 dans l'exposant, remplir avec des 0.
- 5. supprimer les symboles mathématiques et ajuster les signes de l'exposant et du nombre (bits n°7 des octets 1 et 2).
- 6. inverser le bit n°7 de l'exposant (octet n°1)

La conversion de binaire virgule flottante en décimal se fera dans l'ordre inverse.

Remarques:

Le nombre réel 0 est un cas particulier. Il est défini comme un nombre dont l'exposant est nul.

Ainsi tout nombre dont l'exposant est zéro (#00) est interprété comme ayant la valeur zéro par les routines du Basic, quels que soient les contenus des octets de la mantisse.

En ce qui concerne la précision, neuf chiffres significatifs sont affichés à l'écran. La précision interne est un peu plus élevée. En effet $2^-32 = 10^-9.633$.

Le tableau ci-dessous donne les valeurs limites.

	décimal	vir	flottante			
plus grand positif	1.70141183E+38	FF	7F	FF	FF	FF
plus petit positif	2.93873588E-39	Øi	00	00	00	00
zéro	Ø	00	00	00	00	00
plus grand négatif	-2.93873588E-39	01	80	00	00	00
plus petit négatif	-1.70141183E+38	FF	FF	FF	FF	FF

Afin de vous familiariser avec les nombres à virgule flottante, je vous propose, à la fin de ce chapitre, un programme Basic assez efficace.

Il permet de visualiser la variation d'un nombre en fonction de la modification, soit de l'exposant, soit d'un des quatre octets de la mantisse.

Rappelez-vous que la modification de l'exposant a pour effet de déplacer la virgule flottante binaire. L'équivalent décimal est multiplié ou divisé par deux pour chaque décalage Vous remarquerez la variation plus ou moins rapide du nombre suivant que l'octet de la mantisse affecté est plus ou moins significatif. Notez aussi la valeur Ø pour un exposant nul et les changements de signe, entre #FF et #01 pour l'exposant et entre #7F et #80 pour le nombre (octet n°2).

Le fonctionnement du programme est relativement simple. La version ATMOS est plus performante, grâce à l'emploi de ARINT AT. Les utilisateurs d'ORIC-1 pourront sans doute améliorer la leur en utilisant la commande PLOT au lieu de PRINT. La variable de travail VAR est définie au tout début.

Elle est donc inscrite à VARTAB où le programme peut la retrouver.

Sa valeur est en Ri (VARTAB + 2).

De nombreuses adresses du système sont utilisées. Vous devriez les identifier facilement (au besoin, voir chapitre 16).

La sous-routine en langage machine (pour la rapidité) sert uniquement à afficher les 40 bits après chaque modification.

Je vous laisse le soin de la désassembler.

REGISTRES VIRGULE FLOTTANTE

Les adresses des principales routines sont regroupées dans le prochain chapitre.

Il convient de parler des emplacements en page zéro dont se sert l'interpréteur pour manipuler les nombres à virgule flottante. Ils sont appelés 'registres virgule flottante' ou quelquefois en abrégé 'registres flottants'.

Les deux principaux sont: Le registre Accumulateur Flottant (FAC), 6 octets, #DØ-D5. Le registre Argument (ARG), 6 octets, #D8-DD.

Lorsqu'un nombre est transféré dans FAC ou ARG pour un calcul, le bit de poids le plus fort de la mantisse est remis à un et le signe du nombre est détenu par le sixième octet (SGN) dont seul le bit n°7 est significatif.

Prenons par exemple les nombres -20 et +20 décimal:

85	20	00	00 00	00 00	
85	AØ	00	00	00	FF ØØ
	85 FAC ou	85 20 FAC ou ARG 85 A0	85 20 00 FAC ou ARG, six oct 85 A0 00	85 20 00 00 FAC ou ARG, six octets. 85 A0 00 00	FAC ou ARG, six octets. 85 A0 00 00 00

D'autres registres flottants, temporaires, sur cinq octets, sont utilisés.

Le tableau récapitulatif figure ci-dessous. Pour la commodité, il est reproduit au chapitre 19.

Voici la signification des noms barbares qui désignent les différents octets de la mantisse:

HD, High Order byte - MOH, Middle Order High byte - MO, Middle Order byte - LO, Low Order byte.

	FAC	ARG	TEMP1	TEMP2	TEMP3	RND
EXP	DØ	80	C6	CB	BD	FA
HO	D1	D9	C7	CC	BE	FB
MOH	D2	DA	CB	CD	BF	FC
MD	D3	DB	C9	CE	CØ	FD
LO	D4	DC	CA	CF	C1	FE
SGN	D5	DD	_	format	condensé	-

BUFFER DE L'ACCUMULATEUR FLOTTANT

Lorsqu'un nombre en virgule flottante est converti en décimal ou en hexadécimal, habituellement en vue de l'affichage à l'écran, la chaîne de caractères ASCII est rangée temporairement entre les adresses #FF et #110.

Cette zone est appelée Buffer FAC, en abrégé FBUFF.

C'est la routine FOUT en #EØD5 sur ATMOS, #EØD1 sur ORIC-1 qui convertit le nombre en FAC et crée la chaine de caractères dans FBUFF, terminée par ØØ.

Le contenu de FAC est détruit.

Cette routine est utilisée notamment par LINPRT pour l'affichage d'un numéro de ligne, et plus généralement à chaque fois qu'une expression numérique doit etre affichée, c'est-à-dire souvent.

ETUDE DES NOMBRES A VIRQULE FLOTTANTE

--- programme pages suivantes ---

10 GOSUB 1000

20 VT=15:FOR I=0 TO 4:BV=PEEK(R1+I):GOSUB 200:PRINT@ 3+I*8,VT;H\$: NEXT

30 DOKE#12, B: CALL#400: PRINT@ 12,20; VAR; " 13 ω μων ": X=FRE("")

100 BV=PEEK(R2):GET C\$:C=VAL(C\$)

110 IF C>0 AND C<6 THEN GOSUB 300:R2=R1+C-1:GOTO 100

120 IF C\$="V"THEN POKE#BCC9,12:GOTO 400

130 IF C\$="S"THEN CALL#F8B2

140 C=ASC(C\$): IF C<>10 AND C<>11 THEN 100

150 IF C=10 THEN BV=BV-1: IF BV<0 THEN BV=255

160 IF C=11 THEN BV=BV+1: IF BV>255 THEN BV=0

170 POKE R2, BV: GOSUB 200: PRINT@ 3+HT, 15; H\$: GOTO 30

200 H\$=HEX\$(BV): IF MID\$(H\$,2)=""THEN H\$="#00"

210 IF MID\$(H\$,3)=""THEN H\$="#"+"0"+MID\$(H\$,2)

220 H\$=MID\$(H\$,2):RETURN

300 HT=(C-1) *8: VT=14: PRINT@ 0, VT; E\$: PRINT@ 3+HT, VT; F\$

310 VT=16:PRINT@ 0,VT; E\$:PRINT@ 0+HT,VT; G\$:RETURN

400 POKE#26A,35:PRINT@ 0,26;"";:INPUT V\$:VAR=VAL(V\$)

410 POKE#BCC9,8:POKE#26A,34:GOTO 20

1000 VAR=123.456:R1=DEEK(#9C)+2:B=#BE50:DOKE#71,R1

1010 POKE#26A, 34: PAPERO: INK7: DOKE#27C, 0: CLS

1020 E\$=" ← 40 ispaces → ":F\$="/\":G\$="/-

1030 PRINT CHR\$(4):PRINT@ 6,1;CHR\$(27);"JNOMBRES A VIRGULE FLOTTA

1040 FOR I=0 TO 30: READ C: POKE#400+I.C: NEXT

1050 PRINT CHR\$(4):PRINT@ 1,5;"- touches 1,2,3,4,5 -> selection o

1060 PRINT@ 1,6;"- fleches haut, bas -> modification"

1070 PRINT@ 1,7;"- 'V' -> entree nouvelle valeur"

1080 PRINT@ 1,8;"- 'S' -> sortie du programme"

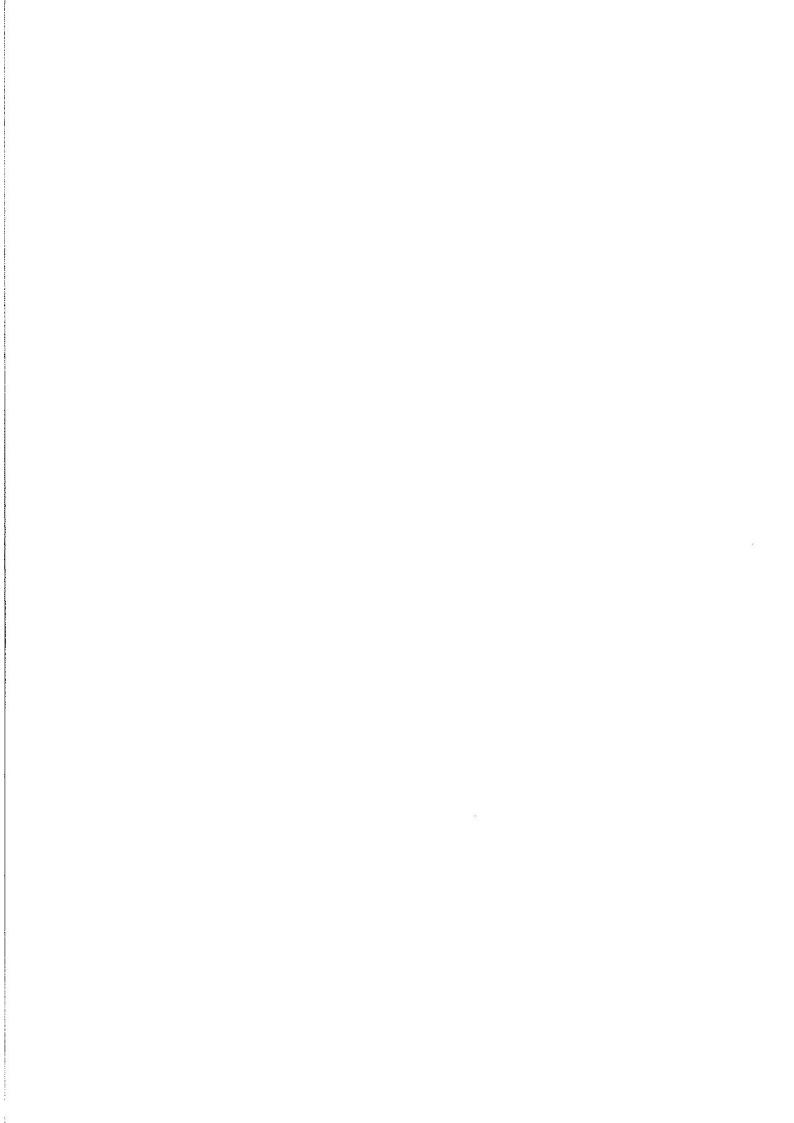
1090 PRINT@ 2,13; "No1 No2 No3 No4 No5"

1100 R2=R1:C=1:GDSUB 300:RETURN

2000 DATA #A0, #00, #B1, #71, #A2, #08, #0A, #48, #B0, #04, #A9, #30, #D0, #02

2010 DATA #20, #D9, #CC, #68, #CA, #D0, #EF, #C0, #C0, #05, #D0, #E6, #4C, #F0, #CB

```
10 GOSUB 1000
20 DOKE#12, #BE00: FOR I=0 TO 4: BV=PEEK(R1+I)
30 GOSUB 200:PRINT" ";H$;"
                               ";:NEXT
40 DOKE#12, #BE50: CALL#400: DOKE#12, #BEAC: PRINT VAR: " - 43 6/000 - "
: X=FRE("")
100 BV=PEEK (R2):GET C$:C=VAL (C$)
110 IF C>0 AND C<6 THEN GOSUB 300:R2=R1+C-1:GOTO 100
120 IF C$="V"THEN POKE#BCC9,12:DOKE#12, #BFBA: PRINT" + 14 0/200 >
":GOTO 400
130 IF C$="S"THEN CALL#F882
140 C=ASC(C$): IF C<>10 AND C<>11 THEN 100
150 IF C=10 THEN BV=BV-1: IF BV<0 THEN BV=255
160 IF C=11 THEN BV=BV+1: IF BV>255 THEN BV=0
170 POKE R2, BV: GOSUB 200: DOKE#12, #BE03+HT: PRINT H$: GOTO 40
200 H$=HEX$(BV): IF MID$(H$,2)=""THEN H$="#00"
210 IF MIDs(Hs,3)=""THEN Hs="#"+"0"+MIDs(Hs,2)
 220 H$=MID$(H$,2):RETURN
300 DOKE#12, #BDDB+HT: PRINT E$: DOKE#12, #BE28+HT: PRINT E$
310 HT=(C-1)*8:DOKE#12, #BDDB+HT:PRINT F*:DOKE#12, #BE28+HT:PRINT G
$: RETURN
400 POKE#26A, 35: DOKE#12, #BFB8: INPUT V$: VAR=VAL (V$)
410 POKE#BCC9,8:POKE#26A,34:GOTO 20
 1000 VAR=123.456:R1=DEEK(#9C)+2:DOKE#71,R1:POKE#26A,34:PAPER0:INK
 1010 Es=" - 8 cohaces - ": Fs="/\": Gs="/----\"
 1020 PRINT CHR$(4):PRINT SPC(5);CHR$(27);"JNOMBRES A VIRGULE FLOT
TANTE"
 1030 FOR I=0 TO 28:READ C:POKE#400+I,C:NEXT:PRINT
 1040 PRINT CHR$(4):PRINT:PRINT" - touches 1,2,3,4,5 -> selection
octet"
 1050 PRINT" - fleches haut, bas -> modification"
 1060 PRINT" - 'V' -> entree nouvelle valeur"
 1070 PRINT" - 'S' -> sortie du programme":PRINT:PRINT:PRINT:PRINT
 1080 PRINT" No1
                                                No5"
                               No3
                                        No4
                       No2
 1090 R2=R1:C=1:GOSUB 300:DOKE#26D,#BFB8:RETURN
 2000 DATA #A0, #00, #B1, #71, #A2, #08, #0A, #48, #80, #04, #A9, #30, #D0, #02
,#A9,#31
 2010 DATA #20,#12,#CC,#68,#CA,#D0,#EF,#C8,#C0,#05,#D0,#E6,#60
```



CHAPITRE 14

ROUTINES UTILISABLES

Les routines avec leur commentaire présentées dans ce chapitre figurent aussi dans la quatrième partie.

D'un point de vue pratique, il est hautement préférable qu'elles soient regroupées par grandes fonctions.

C'est donc ici que vous chercherez plus volontiers lorsque vous aurez un problème particulier à résoudre.

Vous retournerez à la dernière partie pour compléter l'information ou pour mieux situer une routine dans son environnement.

DEMARRAGE, INITIALISATION

TO DO	ATHOS	ORIC-1	description
COLDSTART	F88F	F84A	initialisation complète du système.
WARMSTART	F8B2	F882	RESET, programme et variables conservés
INIT	ECCC	EA59	initialisation pointeurs et variables système en page zéro et deux.
STDCHR	F800	F89B	rétablit les caractères normaux.
READY1	C471	C475	confirme mode texte puis READY2
READY2	C4A8	C4B5	baisse différents drapeaux, affiche 'Ready' puis entre dans CMDLP, boucle d'entrée de l'interpréteur.
SCRTCH	C6FØ	C61B	commande 'NEW', efface programme variables et pile.
SETPTRS	C7Ø8	C733	positionne TXTPTR au début du programme (appelle STXTPT) puis CLEARC.
CLEARC	C70F	C73A	commande 'CLEAR', efface variables et pile
STKINI	C726	C751	initialise la pile
STXTPT	C73A	C765	positionne TXTPTR au début du programme (normalement #500)
RESTORE	C952	C91F	met le pointeur de DATA (#BO,B1) en début de programme (normalement #500)

RECUEIL DE DONNEES A TXTPTR

nom	ATHOS	ORIC-1	description
CHARGET	E2	E2	incrémente TXTPTR (#E9,EA) et recueille dans l'accumulateur le caractère à TXTPTR. C=0 si (A) est un chiffre (de 0 à 9). Z=1 si (A)=#3A, fin d'instruction ou 00, fin de ligne.
CHARGOT	E8	E8	recueille dans l'accumulateur le caractère à TXTPTR sans incrémentation préalable de ce dernier. Même effet sur les indicateurs.
LINGET	CAE2	CA98	met dans LINNUM le numéro de ligne (décimal) (0-63999) pointé par TXTPTR. Suppose que les registres aient été renseignés par l'appel de CHARGET qui a trouvé le premier chiffre. Sort normalement par CHARGET qui cherche le caractère autre qu'un espace après le numéro de ligne. LINNUM contient 00 si pas de nombre à TXTPTR
FRMNUM	CFØ3	CE77	évalue l'expression à TXTPTR, place le résultat en FAC et vérifie qu'il s'agissait bien d'un nombre. TYPE MISMATCH ERROR si c'était une chaine.
FRMEVL	CF17	CE8B	évalue l'expression numérique ou alphanumérique à TXTPTR et place le résultat en FAC. A l'entrée, TXTPTR pointe sur le premier caractère de l'expression.
MAKINT	D29F	D2ØD	évalue l'expression numérique à TXTPTR et convertit le résultat, qui doit être positif et inférieur à 32768 en un entier sur deux octets dans FACMO(+),FACLO(-) (#D3,D4).
MAKINT2	E853	E79D	évalue l'expression numérique à TXTPTR et convertit le résultat, qui doit être positif (0-65535) en un entier sur deux octets dans LINNUM (#33,34). Utilise FRMNUM et GETADR.
GTBYTC	D8C5	D8ØA	appelle CHARGET pour escamoter un caractère puis exécute GETBYT
GETBYT	DSC8	DBØD	évalue l'expression numérique à TXTPTR. Le résultat en FAC est converti par CONINT en un entier sur un octet dans X et FACLO (#D4)
GETNUM	D916	D85B	lit le nombre entier sur deux octets (0-65535, #00-#FFFF) à TXTFTR, le place dans LINNUM (#33,34), vérifie la présence d'une virgule puis recueille un nombre sur un octet (0-255, #00-#FF) dans X. Ce dernier doit être décimal sur ORIC-1. (voir chapitre 18, en #D894, ORIC-1, POKE).

RECUEIL DE DONNEES A TXTPTR (suite)

nom	ATHOS	ORIC-1	description
COMBYTE	D91C	D861	vérifie la présence d'une virgule et recueille un nombre de Ø à 255 dans X. A l'entrée, TXTPTR pointe sur la virgule.
FIN	DFE7	DFCF	recueille le nombre décimal ou hexadécimal à TXTPTR à l'aide de CHARGET et le met en FAC. Suppose que A et les indicateurs aient été renseignés par l'appel de CHARGET ayant trouvé le ler caractère.
HEXGET	E94C	E813	lit un nombre hexadécimal à l'aide de CHARGET et le convertit en un entier non signé sur deux octets dans Y(-),A(+). A l'entrée, TXTPTR pointe sur le caractère précédent le nombre (symbole #).

ENTREE AU CLAVIER

ROM	ATHOS	ORIC-1	description
INCHAR	C5E8	C5F8	recueille dans l'accumulateur le caractère tapé au clavier. Reste en attente de la frappe d'une touche. Appelle RDKEY, lecture mémoire clavier. Sur ATMOS cet appel est vectorisé en #23B.
INLIN	C592	C5A2	rangement d'une ligne de texte entrée au clavier dans le buffer d'entrée (#35,83) Utilise INCHR.
INPUTRQ	CD80	CCF4	affiche un point d'interrogation, un espace puis entre dans INLIN. Utilisée par INPUT.

AFFICHAGE A L'ECRAN SORTIE IMPRIMANTE

Note: la sortie normale des caractères se fait vers l'écran. La plupart des routines utilisent OUTDO qui teste le bit n°7 de PRTFLG (#2F1).

S'il est à un, les caractères sont sortis vers l'imprimante.

nom	ATHOS ORIC-1	description
CRDO	CBFØ CB9F	affiche un retour chariot.
LNFEED	CCØ2 CBA6	affiche un saut de ligne.
STROUT	CCBØ CBED	affiche la chaîne de caractères pointée par A(-),Y(+). La chaîne doit se terminer par un 00 ou un guillemet (#22).

SORTIE DE CARACTERES (suite)

non	ATHOS	ORIC-1	description
STRPRT	CCB3	CBFØ	affiche la chaîne de caractères dont le descripteur est pointé par FACMO,LO (#D3,D4)
CURTGL.	CCD1		bascule curseur (CTRL-Q) ATMOS seulement.
OUTSPC	CCD4	CCØD	affiche un espace.
OUTQST	CCD7	CC10	affiche un point d'interrogation.
סמדטס	CCD9	CC12	affiche le caractère dans l'accumulateur.
TUDGIV	CCFB	CC4D	sort sur l'écran le caractère en A.
INPRT	EØBA	EØB6	affiche 'IN' et le numéro de ligne en cours dans CURLIN. Utilise LINPRT.
LINPRT	EØC5	EØC1	affiche en décimal (0-65535) le nombre non signé sur deux octets dans X(-),A(+).
PRNTFAC	EØCF	EØCB	affiche le contenu de FAC en décimal. FAC est détruit. Utilise FOUT et STROUT.
PRTCHR	F5C1	F578	sort sur imprimante le caractère en A.
טמע	F77C	F73F	sort sur l'écran le caractère en X.
STOUT	F865	F82F	affiche en haut de l'écran, à partir de la colonne indiquée par X, la chaîne pointée par A(-),Y(+), et se terminant par 00.

LOCALISATIONS

ROM	ATHOS	ORIC-1	description
FNDLIN	C4B3	C6DE	recherche la ligne de programme dont le numéro est en LINNUM (#33,34). A la sortie, Si C=1, LOWTR (#CE,CF) pointe sur les octets de chainage de la ligne cherchée. Si C=0, ligne non trouvée, LOWTR pointe sur les octets de chainage de la ligne suivante.
DATA	CA3C	CAØA	positionne TXTPTR à la fin de l'instruction. utilise DATAN.
ADDON	CA3F	CAØD	déplace TXTPTR à la fin de l'instruction ou de la ligne. Ajoute le contenu de Y à TXTPTR.
DATAN	CA4E	CA1C	calcule le déplacement dans Y depuis TXTPTR jusqu'au prochain : ou 00, fin d'instruction.
REMN	CA51	CA1F	calcule le déplacement dans Y depuis TXTPTR jusqu'au prochain 00, fin de ligne.

LOCALISATIONS (suite)

TI O M	ATHOS	ORIC-1	description
ADDON2	CAB1		positionne TXTPTR à la fin de l'instruction. peut être appelée par NEWSTT (ATMOS).
PTRGET	D188	DØFC	lit le nom d'une variable à l'aide de CHARGET et la trouve en mémoire. A l'entrée, TXTPTR pointe sur le premier caractère du nom. A la sortie, l'adresse de la variable se trouve en VARPNT (#B6,B7) et en A(-),Y(+). Si PTRGET ne peut pas trouver une variable simple, elle en crée une. Si elle ne peut pas trouver un tableau, elle en crée un, dimensionné 0-10, et dont elle fixe tous les éléments à zéro.

ROUTINES VIRGULE FLOTTANTE

Voir fin du chapitre 13 et début du chapitre 19, définition des registres virgule flottante, emplacements en page zéro concernés et remarques concernant l'utilisation de ces routines.

- opérations arithmétiques

nom	ATHOS	ORIC-1	description
FSUB	DBØB	DA8Ø	inscrit dans ARG (#D8-DD) le nombre virgule flottante en mémoire pointé par A(-),Y(+) puis exécute FSUBT
FSUBT	DBØE	DA83	soustrait FAC de ARG, résultat dans FAC.
FADD	DB22	DA97	inscrit dans ARG (#D8-DD) le nombre virgule flottante en mémoire pointé par A(-),Y(+) puis exécute FADDT.
FADDT	DB25	DA9A	additionne FAC et ARG, résultat dans FAC.
FMULT	DCED	DCB7	inscrit dans ARG (#D8-DD) le nombre virgule flottante en mémoire pointé par A(-),Y(+) puis exécute FMULTT.
FMULTT	DCFØ	DCBA	multiplie FAC par ARG, résultat dans FAC.
FDIV2	DDDE	DDDA	inscrit dans FAC (#DØ-D5) le nombre virgule flottante en mémoire pointé par A(-),Y(+) puis exécute FDIVT.
FDIV	DDE4	DDEØ	inscrit dans ARG (#D8-DD) le nombre virgule flottante en mémoire pointé par A(-),Y(+) puis exécute FDIVT.
FDIVT	DDE7	DDE3	divise ARG par FAC, résultat dans FAC.
FPWRT	E238	E234	élève ARG à la puissance FAC, résultat dans FAC.

- fonctions

ROB	ATHOS	ORIC-1	description
LN	DCAF	DC79	log base e de FAC, résultat dans FAC.
L06	DDD4	DDDØ	log base 10 de FAC, résultat dans FAC.
PI	DE77	DBEE	FAC = PI
FALSE	DFØB	DEFC	FAC = 00.
TRUE	DFØF	DFØØ	FAC = FF (-1).
SGN	DF21	DF12	détermine le signe de FAC, résultat dans FAC FAC = 1 si FAC était positif. FAC = 0 si FAC était égal à zéro. FAC =-1 si FAC était négatif.
ABS	DF49	DF31	valeur absolue de FAC (bit 7 FACSGN mis à 0)
INT	DFBD	DFA5	plus grande valeur entière de FAC, résultat dans FAC.
SQR	E22E	E22A	racine carrée de FAC, résultat dans FAC.
EXP	E2AA	E2A6	e puissance FAC, résultat dans FAC.
RND	E34F	E34B	forme un nombre aléatoire dans FAC.
COS	E38B	E387	cos(FAC), résultat dans FAC.
SIN .	E392	E38E	sin(FAC), résultat dans FAC.
TAN	E3DB	E3D7	tg(FAC), résultat dans FAC.
ATN	E43F	E43B	arctg(FAC), résultat dans FAC.

- déplacements

Note: lorsqu'une adresse est pointée par des registres, celui contenant l'octet faible est mentionné en premier.

TION	ATHOS ORIC-1	description
CONUPK	DD51 DD4D	(A,Y) -> ARG. inscrit dans ARG le nombre virgule flottante en mémoire pointé par A(-),Y(+). A la sortie A est chargé avec FACEXP (#D0).
MOVFM	DE7B DE73	(A,Y) -> FAC. transfère dans FAC le nombre virgule flottante en mémoire pointé par A(-),Y(+). A la sortie A et Z reflètent la valeur de FACEXP.

déplacements (suite)

行口册	ATHOS ORIC-1	description
MOV2F	DEAØ DE98	FAC -> TEMP2. met FAC sous forme condensée et le transfère dans TEMP2 (#CB-CF). A la sortie, A et Z reflètent la valeur de FACEXP.
MOV1F	DEA3 DE9B	FAC -> TEMP1. met FAC sous forme condensée et le transfère dans TEMP1 (#C6-CA). A la sortie, A et Z reflètent la valeur de FACEXP.
MOVML	DEAS DE9D	FAC -> (X,Ø). met FAC sous forme condensée et le transfère en page zéro, à l'adresse pointée par X. A la sortie, A et Z reflètent la valeur de FACEXP.
MOVMF	DEAD DEAS	FAC -> (X,Y). met FAC sous forme condensée et le transfère en mémoire à l'adresse pointée par X(-),Y(+). A la sortie A et Z reflètent la valeur de FACEXP.
MOVFA	DED5 DECD	ARG -> FAC. transfère ARG dans FAC. A la sortie, (A) = FACEXP et Z = 1.
MOVAF	DEES DEDD	FAC -> ARG. transfère FAC dans ARG. A la sortie, (A) = FACEXP et Z = 1.

- utilitaires

Ti O TR	ATHOS ORIC-1	description
FADDH	DBØ4 DA79	ajoute 1/2 à FAC
MUL10	DDA7 DDA3	multiplie FAC par 10, résultat dans FAC. Fonctionne pour nombres positifs ou négatifs
DIVID	DDC3 DDBF	divise FAC par 10, résultat dans FAC, positif seulement.
SIGN	DF13 DF04	modifie le contenu de l'accumulateur suivant la valeur de FAC. A = 1 si FAC > 0 A = 0 si FAC = 0 A = FF si FAC < 0

n o m	ATHOS	ORIC-1	description
FCOMP	DF4C	DF34	compare FAC et un nombre virgule flottante en mémoire pointé par A(-),Y(+). Le contenu de A est modifié selon le résultat: A = 1 si (A,Y) < FAC A = 0 si (A,Y) = FAC A = FF si (A,Y) > FAC
ADDACC	EØ76	EØ72	ajoute à FAC le contenu de l'accumulateur.
FOUT	EØD5	EØD1	crée une chaines de caractères dans FBUFF (#FF-110) correspondant à la valeur de FAC et terminée par un 00. A la sortie, A(-),Y(+) pointe la chaine, le contenu de FAC est perdu.
NEGOP	E271	E26D	FAC = -FAC.
- constant	25		
décimal	ATMOS	ORIC-1	virgule flottante
-32768	D297	D205	90 80 00 00 00
65535		D8E4	91 00 00 00 00
LN(10)	DC77	DC46	82 13 5D 8D DE
PI	DC7C	D8E9	82 49 0F DA 9E
1	DC81	DC4B	81 00 00 00 00
SQR(.5)	DC9B	DC65	80 35 04 F3 34
SQR (2)	DCAØ	DC4A	81 35 Ø4 F3 34
-1/2	DCA5	DC6F	80 80 00 00 00
LN(2)	DCAA	DC74	8Ø 31 72 17 F8
10	DDBE	DDBA	84 20 00 00 00 .
99999999.9	EØAB	EØA7	9B 3E BC 1F FD
999999999	EØBØ	EØAC	9E 6E 6B 27 FD
1E+Ø9	EØB5	EØB1	9E 6E 6B 28 00
1/2	E2 0 5	E201	80 00 00 00
1/LN(2)	E27C	E278	81 38 AA 3B 29
PI/2	E4Ø7	E403	81 49 ØF DA A2
2PI	E40C	E4Ø8	83 49 ØF DA A2
1/4	E411	E4ØD	7F 00 00 00 00

VERIFICATIONS DIVERSES - SYNTAXE

71 の飛	ATHOS (ORIC-1	description
ISCNTC	C962	C93Ø	teste le clavier pour CTRL-C. Sortie du programme par BREAK si CTRL-C.
CHKNUM	CFØ6	CE7A	vérifie que le contenu de FAC correspond à une valeur numérique.
CHKSTR	CFØ8	CE7C	vérifie que le contenu de FAC concerne une chaîne de caractères.
CHKVAL	CFØ9	CE7D	teste le résultat de la dernière opération sur le FAC pour savoir si elle concernait une variable numérique ou alphanumérique. A l'entrée, C=1, test pour chaine C=0, test pour nombre TYPE MISMATCH si FAC et C ne concordent pas.
ISLETC	D216	D186	teste l'accumulateur pour une lettre ASCII, (A à Z). A la sortie, C=1 si A contient une lettre.
PARCHK	DØ59	CFCD	vérifie la présence de parenthèses et évalue l'expression incluse. Utilise CHKOPN, FRMEVL puis CHKCLS.
CHKCLS	DØ5F	CFD3	vérifie la présence d'une parenthèse fermée ')' à TXTPTR. Utilise SYNCHR.
CHKOPN	DØ62	CFD6	vérifie la présence d'une parenthèse ouverte '(' à TXTPTR. Utilise SYNCHR.
CHKCOM	DØ65	CFD9	vérifie la présence d'une virgule à TXTPTR. Utilise SYNCHR.
SYNCHR	DØ67	CFDB	compare le contenu de l'accumulateur au caractère pointé par TXTPTR. Sortie par CHARGET si concordance, sinon SYNYAX ERROR.
SYNTERR	DØ7Ø	CFE4	sortie, affichage ?SYNTAX ERROR.

CHAINES DE CARACTERES

La manipulation des chaînes de caractères fait intervenir trois éléments:

- la chaîne de caractères elle-même qui peut se trouver n'importe où en mémoire (dans le texte Basic ou sous HIMEM).
- Un descripteur qui comprend la longueur de la chaîne et l'adresse du premier caractère (voir page 33).
- Un pointeur d'adresse du descripteur.

135

Dans la plupart des routines, le descripteur est laissé en mémoire et son adresse est conservée dans le pointeur détenu en FAC.

135

CHAINES DE CARACTERES (suite)

ROM	ATHOS ORIG	description	
COPY	CBBD CB4	libère un descripteur temporaire de chaîne pointé par A(-),Y(+) et le déplace à l'adresse pointée par FORPNT (#B8,B9).	e
STRTXT	DØ25 CF9	'évalue' une chaîne de caractères. A(-) et Y(+) sont chargés avec l'adresse de l'er caractère après ", à (TXTPTR)+C, puis un déscripteur de la chaine est construit par STRLIT, pointeur dans FACMO,LO (#D3,D4 la sortie, TXTPTR pointe sur le caractère suivant la chaîne.	4)
STRINI	D5A3 D48	fait de la place pour une chaîne et crée descripteur pour elle dans DSCTMP(#DØ,D1,D4,D4) A l'entrée, (A) = longueur de la chaîne.	
STRSPA	DSAB D4	appelle GETSPA et range le descripteur en DSCTMP (#DØ,D1,D2).	
STRLIT	D585 D48	range un guillemet dans CHARAC (#24) et d ENDCHR (#25) de façon à ce que STRLT2 s'arrête sur lui ->	ans
STRLT2	D588 D50	prend une chaîne dont le premier caractère est pointé par A(-),Y(+) et construit un descripteur pour elle dans DSCTMP (#D0,D1 Poursuit avec PUTNEW.	
PUTNEW	D5F4 D53	range DSCTMP dans un descripteur temporaim pointé par FACMO,LO (#D3,D4) et fait VALT égal à #FF (chaîne).	
GETSPA	D61E D56	fait de la place pour une chaîne de caractères. A l'entrée, (A) = longueur de la chaine. A la sortie, (A) est inchangé. Le début de l'espace créé est pointé par X(-),Y(+), FRETOP et FRESPC.	e
CAT	D767 D66	concaténation de deux chaînes de caractère A l'entrée, FACMO,LO (#D3,D4) pointe sur descripteur de la première chaîne et TXTP sur le signe +.	le
MOVINS	D7A4 D6E	déplace une chaîne dont le descripteur es pointé par STRNG1 (#DE,DF) vers l'emplace en mémoire pointé par FRESPC (#A4,A5).	
MOVSTR	D782 D6F	déplace une chaîne pointée par X(-),Y(+) de longueur (A) vers l'emplacement en mémoire pointé par FRESPC (#A4,A5).	
FRESTR	D7CD D71	s'assure que FAC adresse une chaîne puis exécute FREFAC	

CHAPITRE 14 136

CHAINES DE CARACTERES (suite)

ti O R	ATHOS ORIC-1	description
FREFAC	D7DØ D715	charge le pointeur du descripteur contenu dans FACMO,LO (#D3,D4) dans A(-),Y(+) puis exécute FRETMP.
FRETMP	D7D4 D719	libère l'espace occupé par une chaîne temporaire. A l'entrée, le pointeur du descripteur est en A(-),Y(+). A la sortie, l'adresse de la chaîne est en INDEX (#91,92) et en X(-),Y(+) et sa longueur en A.
FRETMS	D805 D74A	libère un descripteur temporaire sans libérer la chaîne. A l'entrée, A(-),Y(+) pointe le descripteur à libérer. A la sortie, Z=1 indique qu'un descripteur a été libéré.

CONVERSIONS

TOB	ATHOS	ORIC-1	description
AYINT	D2A9	D217	si FAC > -32768 et < 32767 alors QINT rend entier le FAC. Sinon, ILL.QUANT. ERR.
GIVAYF	D499	D3ED	rend flottant l'entier signé sur deux octets dans Y(octet faible) et A(octet fort).
GIVAYF2	DF4Ø	D8D5	rend flottant l'entier non signé sur deux octets dans Y(octet faible) et A(octet fort)
SNGFLT	D4B6	D3FD	rend flottant l'entier non signé dans Y.
FLOAT	DF24	DF15	rend flottant l'entier signé dans A.
CONINT	DSCB	D810	convertit le nombre en FAC en un entier sur un octet dans X et FACLO (#D4). Sort normalement par CHARGET. ?ILLEGAL QUANTITY ERROR si FAC < 0 ou > 255.
GETADR	D922	D867	convertit le nombre en FAC (0-65535) en un entier sur deux octets dans LINNUM (#33,34) et dans FACMO,#D3 (+) et FACLO,#D4 (-).
QINT	DF8C	DF74	laisse INT(FAC) dans mantisse FAC (#D1-D4) signé. Suppose FAC < 2^23.

AMENAGEMENTS, TRANSFERTS

nom	ATHOS ORIC-1	description				
BLTU	C3F4 C3F8	BLock Transfer Utility, fait de la place mémoire par transfert de bloc vers le hau Utilisée par CMDLP pour insertion de lign de programme. A l'entrée: A(-),Y(+) et HIGHDS (#C7,C8) contiennent Adr.destination haute + 1. LOWTR (#CE,CF) = + basse Adr.à déplacer. HIGHTR (#C9,CA)= + haute Adr.à déplacer +				
REASON	C444 C448	vérifie espace disponible en mémoire. Compare l'adresse en A(-),Y(+) à FRETOP. appelle éventuellement GARBAGE. message ?OUT OF MEMORY si pas de place.				
LINKSET	C55F C56F	positionne les octets de chainage du programme Basic à partir de TXTTAB (#9A,9B). Utilise pointeur INDEX (#91,92).				
GARBAGE	D650 D595	déplace les chaînes en vigueur le plus haut possible en mémoire afin d'augmenter l'espace libre disponible.				
MOVE	EDC4 ECØC	déplace les jeux de caractères en fonction du mode en vigueur.				
		ATMOS ORIC-1 Adr.départ en #0C,0D #200,201 Adr.destination #0E,0F #202,203 nombre d'octets #10,11 #204,205				

COMMANDES BASIC

Note:

La plupart de ces routines sont sans retour.

nom	ATHOS	ORIC-1	description
LIST	C803	C82C	appelle CHARGOT puis -> LIST.
NEWSTT	C8C1	C8AD	exécute une nouvelle instruction. A l'entrée, CHARGET pointe sur les deux-points ou le 00, marquant la fin de l'instruction précédente.
CONT	C9A2	C97Ø	place OLDTXT (pointant un ':' ou 00, fin d'instruction) et OLDLIN dans TXTPTR et CURLIN.
GOTO	C9E5	C9B3	utilise FRMNUM et GETADR pour modifier TXTPTR. Suppose que les registres aient été renseignés par l'appel de CHARGET ayant trouvé le premier chiffre du n° de ligne.

COMMANDES BASIC (suite)

non	ATMOS ORIC-1	description
G0T02	C9E8 C9B6	branchement à n° de ligne contenu dans LINNUM (#33,34)
LET	CB1C CAD2	utilise PTRGET pour obtenir l'adresse de la variable, vérifie la présence du signe '=' évalue l'expression et range le résultat en zone des variables. A l'entrée, TXTPTR pointe sur le premier caractère du nom de la variable.
CLS	CCCE CCØA	vide l'écran texte.
LORES1	D9EA D943	mode LORES1
LORESØ	D9ED D946	mode LORESØ
GRAB	EBE7 E974	récupère mémoire écran HIRES pour texte.
RELEASE	ECØC E994	rétablit mémoire écran HIRES.
TEXT	EC21 E9A9	met en mode texte.
HIRES	EC33 E9BB	met en mode HIRES.

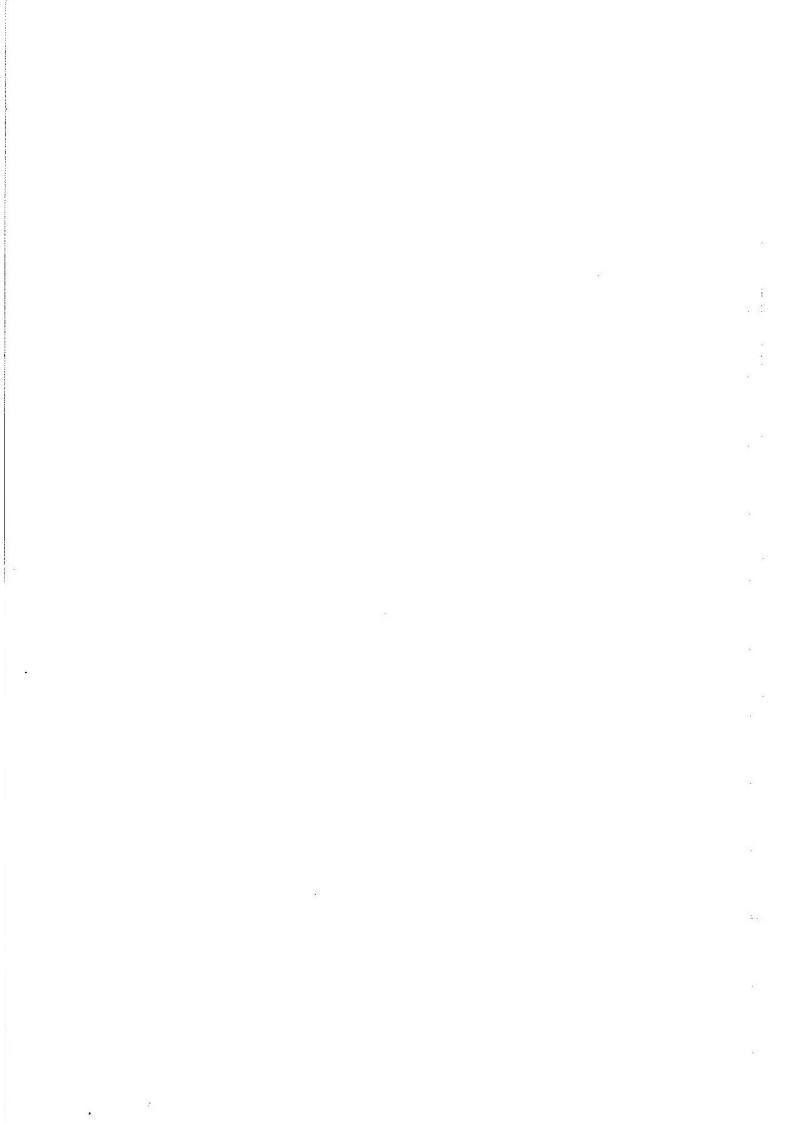
CASSETTE, HIRES, SON

Ces routines sont groupées dans la dernière partie de la ROM. Voir chapitre 20.

CASSETTE: ATMOS #E4AC-EACO, DRIC-1 #E4A8-E80C

HIRES: ATMOS #EED8-F494, ORIC-1 #EDBC-F3FF

SON: ATMOS #FA86-FC77, ORIC-1 #FA6C-FC5D



CHAPITRE 15

APPLICATIONS

Il est impossible de passer en revue toutes les routines utilisables. Quelques exemples significatifs sont donnés dans ce chapitre.

Nous terminerons par une application pratique que vous devriez trouver intéressante. En tous cas, elle vous donnera des idées pour des expériences futures.

Chargez votre utilitaire. Vous entrerez le code machine de la manière habituelle.

Avant de lancer un programme, utilisez la fonction désassemblage pour vérifier.

DONNEES ENTREES AU CLAVIER

Nous avons vu, pages 87 et 88, un moyen de passer des informations avec la commande CALL.

Dans cette section, nous étudierons particulièrement les routines d'entrée spécialisées INCHR et INLIN.

- routine INCHR

C'est le moyen le plus simple de recueillir une information à partir du clavier. Il peut être comparé à la commande Basic GET. Quand la routine INCHR est appelée, elle se met en attente de la frappe d'une touche.

Au retour, le code du caractère, correspondant à la touche enfoncée, est placé dans l'accumulateur.

Si l'information doit être exploitée, elle sera analysée au moyen de l'instruction de comparaison CMP.

Entrez en commande directe:

ATMOS: CALL#C5E8, ORIC-1: CALL#C5F8

L'action sur une touche quelconque provoque le retour au Basic.

La routine ISLETC permet de déterminer s'il s'agit d'une lettre.

	AIMUS	URIC-1	
#400:	20 E8 C5	20 F8 C5 entrée	JSR INCHR
#403:	20 16 D2	20 86 D1	JSR ICLETC ;est-ce une lettre?
#406:	9Ø F8	90 F8	BCC entrée ;non
#408:	60	60	RTS ; oui retour Basic

Le programme suivant forme une boucle de laquelle on ne sortira qu'en tapant dans l'ordre les lettres M, O, T.

MOT de passe

#400:	20	E8	C5	20	F8	C5	entrée	JSR	INCHR
#403:	C9	4D		C9	4D			CMP	'M'
#405;	DØ	F9		DØ	F9			BNE	entrée
#407:	20	E8	C5	20	F8	C5		JSR	INCHR
#4ØA:	C9	4F		C9	4F			CMP	,0,
#4ØC:	DØ	F2		DØ	F2			BNE	entrée
#40E:	20	E8	C5	20	F8	C5		JSR	INCHR
#411:	C9	54		C9	54			CMP	'T'
#413:	DØ	EB		DØ	EB			BNE	entrée
#415:	60			60				RTS	

Voici une technique souvent employée pour aiguiller le programme en fonction d'un choix dans un menu.

Options

#400:	20	E8	C5	20	F8	C5	entrée	JSR	INCHR
#403:	C9	43		C9	43			CMP	' C'
#405:	DO	Ø 3		DØ	03			BNE	suivant
#407:	4C	CE	CC	4C	ØA	CC		JMP	CLS
#40A:	C 9	5A		C9	5A			CMP	'Z'
#40C:	DØ	F2		DØ	F2			BNE	entrée
#40E:	4C	EI	FA	4C	C7	FA		JMP	ZAP

- routine INLIN

Cette routine, ou sa variante INPUTRQ, permet de recueillir une suite de caractères dans le buffer d'entrée.

Cette information peut simplement être rangée ailleurs en mémoire pour traitement ultérieur, affichage par exemple.

#400:	20	80	CD	20	F4	CC	entrée	JSR	INPUTRO
#403:	A2	FF		A2	FF			LDX	>#FF
#405:	E8			E8			nxtchr	INX	
#406:	B 5	35		B5	35			LDA	BUF, X
#408:	9D	20	Ø4	9D	20	04		STA	adr,X
#4ØB:	DØ	F8		DØ	F8			BNE	nxtchr
#40D:	4C	8A	C4	4C	B5	C4		JMP	Ready2
#410:	A9	20		A9	20			LDA	adrL
#412:	AØ	Ø4		AØ	04			LDY	adrH
#414:	4C	BØ	CC	4C	ED	CB		JMP	STROUT

Faites CALL#400, entrez un message quelconque puis CALL#410.

L'information peut être traitée par une des routines de recueil de données à TXTPTR.

Il faut positionner TXTPTR en fonction des caractéristiques de la routine.

Sachant qu'au retour de INLIN, X(-),Y(+) contiennent l'adresse BUF-1 (#34).

Certaines routines, dont LINGET doivent être précédées d'un appel de CHARGET.

Remarque: cet appel est automatique après la lecture d'une commande. (voir utilisation de la commande! pages 23 et 88).

#400:	20	80	CD	20	F4	CC	entrée	JSR	INPUTRQ
#403:	86	E9		86	E9			STX	TXTPTR
#405:	84	EA		84	EA			STY	TXTPTR+1
#407:	20	E2	00	20	E2	00		JSR	CHARGET
#40A:	20	E2	CA	20	98	CA		JSR	LINGET
#40D:	20	B 3	C6	20	DE	C6		JSR	FNDLIN
#410:	A6	CE		A6	CE		4	LDX	LOWTR
#412:	A5	CF		A5	CF			LDA	LOWTR+1
#414:	20	C5	EØ	20	C1	EØ		JSR	LINPRT
#417:	40	AB	€4	4C	B 5	C4		JMP	Ready2

CALL#400. Entrez le numéro d'une ligne du programme Basic en mémoire. Vous obtiendrez l'adresse de la ligne en décimal.

Pour FRMNUM et les autres routines d'évaluation, il suffit de positionner TXTPTR au début du buffer d'entrée. Voici le même programme de recherche. Le numéro de ligne est recueilli par MAKINT2 (FRMNUM, GETADR). Seule la portion de code entre #403 et #400 est à modifier.

#400:		80	CD		F4	CC	entrée		INPUTRO	V V-4-4
#403:	FR			E8				INX		; X=#34+1
#404:	86	E9		86	E9			STX	TXTPTR	; TXTPTR=#35
#406:	84	EA		84	EA			STY	TXTPTR+1	; TXTPTR+1=0
#408:	EA			EA				NOP		
#409:	EA			EA				NOP		
#4ØA:	20	53	E8	20	9D	E7		JSR	MAKINT2	
#40D:	20	B 3	CQ.	20	DE	C6		JSR	FNDLIN	
#410:	A6	CE		A6	CE			LDX	LOWTR	
#412:	A5	CF		A5	CF			LDA	LOWTR+1	
#414:	20	C5	EØ	20	C1	EØ		JSR	LINPRT	
#417:	4C	A8	C4	4C	B 5	C4		JMP	Ready2	

Remarque: à défaut d'entrée, l'adresse de la première ligne est affichée par la première version, alors que FRMNUM produit une erreur de syntaxe en l'absence d'expression à évaluer.

Si l'expression numérique comporte des mots-clés, ils doivent être codés par PARSE avant évaluation.

#400:	20	80	CD	20	F4	CC	entrée	JSR	INPUTRQ
#403:	86	E9		86	E9			STX	TXTPTR
#405:	84	EA		84	EA			STY	TXTPTR+1
#407:	20	E2	00	20	E2	00		JSR	CHARGET
#40A:	FØ	F4		FØ	F4			BEQ	entrée
#40C:	20	FA	C5	20	ØA	C6		JSR	PARSE
#4ØF:	E6	E9		E6	E9			INC	TXTPTR
#411:	20	03	CF	20	77	CE		JSR	FRMNUM
#414:	4C	CF	EØ	4C	CB	EØ		JMP	PRTFAC

CALL#400.

Entrez une expression numérique valide telle que, par exemple:

2^11-SQR(64)*INT(4/COS(PI/3)+.5)

Le résultat en FAC est affiché par PRTFAC.

Voici un exemple d'utilisation des routines arithmétiques:

#400:	20	1A	Ø4	20	1A	04		JSR	entrée	;multiplicande
#403:	A2	27		A2	27			LDX	mem	jen FAC transféré
#405:	AØ	Ø4		AØ	04			LDY	mem+1	;en #427
#407:	20	AD	DE	20	A5	DE		JSR	MOVMF	;par MOVMF
#40A:	20	1A	Ø4	20	1A	Ø4		JSR	entrée	;multiplicateur
#40D:	A9	27		A9	27			LDA	mem	;multiplicande
#40F:	AØ	Ø4		AØ	04			LDY	mem+1	;placé en ARG
#411:	20	ED	DC	20	B7	DC		JSR	FMULT	; opération
#414:	20	CF	EØ	20	CB	EØ		JSR	PRTFAC	;résultat en FAC
#417:	4C	A8	C4	4C	B 5	C4		JMP	Ready2	;retour Basic
#41A:	20	80	CD	20	F4	CC	entrée	JSR	INPUTRO	;entrée clavier
#41D:	86	E9		86	E9			STX	TXTPTR	;positionne TXTPTR
#41F:	84	EA		84	EA			STY	TXTPTR+1	;nombre à BUF
#421:	20	E2	00	20	E2	00		JSR	CHARGET	;mis en FAC
#424:	4C	E7	DF	4C	CF	DF		JMP	FIN	;par FIN

CALL#400, entrez le multiplicande puis le multiplicateur. Tous les nombres sont acceptés, positifs, négatifs ou hexadécimaux. Pour changer d'opération, remplacez la routine arithmétique en #411 par FSUB, FADD ou FDIV.

AFFICHAGE A L'ECRAN

Différents messages à afficher sont proposés pour chacun des exemples qui suivent. Essayez-les successivement. Constatez l'effet des caractères spéciaux qui sont inclus. Consultez les annexes 1 et 2 pour les identifier.

Les deux principales routines utilisables sont OUTDO et STROUT.

Voici une technique d'utilisation de OUTDO

#400: #402: #405: #407: #40A: #40B:	BD FØ 2Ø E8 DØ	ØE Ø6 D9	Ø4		BD FØ 20 E8	00 0E 06 12 F5		boucle fin	LDA BEQ JSR INX	fin OUTDO boucle	;X=index ;caractère en A ;00 terminé ;affichage ;carac. suivant ;jusqu'à 255 car ;terminé
#40E: #40E: #40E:	ØC	ØB	84	4F	52				ØC Ø	(C' 00 08 84 '(18 45 '(DRIC' 00 DRIC' 00

CALL#400

Autre méthode avec OUTDO.

Une sous-routine d'affichage séparée utilise la pile pour se positionner au début des données à afficher puis calcule et réempile sa propre adresse de retour

#400:	20	14	04		20	14	04	msg 1	JSR print	
#403:	48	45	4C	4C	4F	00			HELLO' 00	
#409:	60				60			fin	RTS	

#4ØA:	20					14	Ø4	msg2	JSR print	r ma
#40D:		8C	48	65		00			ØC 8C 'Hep	00
#413:	60				60			fin	RTS	
#414:	68				68			print	PLA	;adr.désempilée
#415:	85	80			85	80			STA ptr	;= adr.retour
#417:	68				68				PLA	;de JSR print - 1
#418:	85	81			85	81			STA ptr+1	;soit adr.début
#41A:	AØ	Ø1			AØ	Ø1			LDY >#Ø1	;des données - 1
#41C:	B1	80			B1	80		boucle	LDA (ptr),Y	;caractère en A
#41E:	FØ	06			FØ	26			BEQ terminé	;00 terminé
#420:	20	D9	CC		20	12	CC		JSR OUTDO	; affichage
#423:	CB				C8				INY	;carac.suivant
#424:	DØ	F6			DØ	F6			BNE boucle	; jusqu'à 255 carac
#426:	18				18			terminé	CLC	;nombre de
#427:	98				98				TYA	;caractères
#428:	65	80			65	80			ADC ptr	;est ajouté
#42A:	85	80			85	80			STA ptr	;à adresse
#42C:	A5	81			A5	81			LDA ptr+1	;retour - 1
#42E:	69	00			69	00			ADC >#ØØ	;détenue par ptr
#430:	48				48				PHA	;cette nouvelle
#431:	A5	80			A5	80			LDA ptr	;adr. est remise
#433:	48				48				PHA	;en pile pour
#434:	60				60			sortie	RTS	;retour & 'fin'

CALL#400, puis CALL#40A.

Routine STROUT:

#400:	A9	07			A9	07							LDA dataL	
#402:	AØ	04			AØ	04							LDY dataH	
#404:	4C	BØ	CC		4C	ED	CB						JMP STROUT	
#407:	ØC.	4F	52	49	43	00							ØC 'ORIC' ØØ	
#407:	ØC	ØA.	04	88	4F	52	49	43	04	ØA	ØØ		ØC ØA Ø4 8A 'ORIC' Ø4 ØA ØØ	
#407:	ØC	ØA	04	18	44	4F	52	49	43	04	ØA	00	id.avec ESC-J	

L'affichage double hauteur doit commencer sur une ligne paire. Pour l'assurer, quelle que soit la ligne actuelle:

#400:	AD	68	02	AD	68	02		LDA	CURROW	
#403:	4A			4A				LSR	A	
#404:	90	03		90	03			BCC	affichage	
#406:	20	02	CC	20	A6	CB		JSR	LNFEED	
#409:	A9	10		A9	10		affichage	LDA	dataL	
#40B:	AØ	04		AØ	04			LDY	dataH	
#40D:	4C	80	CC	4C	ED	CB		JMP	STROUT	

#410: 04 8A 4F 52 49 43 04 0A 00 04 8A 'ORIC' 04 0A 00 #410: 04 8E 4F 52 49 43 04 0A 00 04 8E idem, clignotant

routine STOUT:

#400:	A9	07	A9	29	LDA	dataL
#402:	AØ	04	AØ	Ø4	LDY	dataH
#404:	A2	10	A2	10	LDX	colonne
#406:	4C	65 F8	4C	2F F8	JMP	STOUT

#409: 06 0C 53 54 4F 55 54 00 06 0C 'STOUT' 00

Pour changer la position d'écriture, la solution la plus simple est d'écrire directement dans la mémoire d'écran.

#400:	20	CE	CC		20	ØA	CC			JSR	CLS	
#403:	A2	00			A2	00				LDX	>#00	
#405:	BD	14	04		BD	14	04		boucle	LDA	data,X	
#408:	FØ	09			FØ	09				BEQ	fin	
#40A:	9B	00	BF		9D	00	BF			STA	mem,X	
#40D:	EA				EA					NOP		
#40E:	EA				EA					NOP		
#40F:	EA				EA					NOP		
#410:	E8				E8					INX		
#411:	DØ	F2			DØ	F2				BNE	boucle	
#413:	60				60				fin	RTS		
#414:	02	48	45	4C	4C	4F	ØØ			02	HELLO'	00
#414:	04	ØC	C8	C5	CC	CC	CF	00		04 6	MC id.nég	00

Les caractères de contrôle deviennent attributs. Les caractères en ASCII négatif sont affichés en vidéo inverse.

Pour la double hauteur, il est nécessaire d'écrire deux lignes identiques superposées dont la première doit être paire.

#400:	20 (CE C	C	20	ØA	CC			JSR CLS
#403:	A2 (00		A2	00				LDX >#ØØ
#405:	BD :	14 🛭	14	BD	14	Ø4		boucle	LDA data,X
#408:	FØ (09		FØ	09				BEQ fin
#4ØA:	9D (00 E	F	9D	00	BF			STA mem, X) double
#40D:	9D :	28 B	F	9D	28	BF			STA mem+40,X)ligne
#410:	E8			E8					INX
#411:	DØ f	F2		DØ	F2				BNE boucle
#413:	60			60				fin	RTS
#414:	Ø1 (ØA 4	8 45	4C	4C	4F	00	data	01 0A 'HELLO' 00
#414:	Ø5 (ØE C	8 C5	CC	CC	CF	00		05 OE inverse clignotant

Autre solution pour imposer la position d'écriture.

```
#400: 20 CE CC
                   20 0A CC
                                           JSR CLS
#403: A9 10
                   A9 10
                                           LDA >#10
                                                          ;ligne
#405: 20 0C DA
                   20 65 D9
                                           JSR BASCALC
                                                          ;calc.adr.base
#408: A2 00
                   A2 00
                                           LDX >#00
                                                          ;index data
                                           LDY >#10
                                                          ;colonne
#40A: A0 10
                   AØ 10
                   BD 18 Ø4
                                           LDA data, X
#40C: BD 18 04
                                 boucle
                                                          ;carac.en A
#40F: F0 06
                   FØ Ø6
                                                          ;00 terminé
                                           BEQ fin
                                           STA (#1F),Y
#411: 91 1F
                   91 1F
                                                          ; poke écran
#413: C8
                   C8
                                           INY
                                                          ; col. suiv.
#414: E8
                   E8
                                           INX
                                                          ; carac.suiv.
#415: DØ F5
                   DØ F5
                                           BNE boucle
                                                          ; jusqu'à 255
#417: 60
                                  fin
                                           RTS
                   60
#418: 04 0C 4F 52 49 43 00
                                           Ø4 ØC 'O R I C' ØØ
                                  data
#418: 06 C1 D4 CD CF D3 00
                                           05 'A T M O S' 00
```

Le message ne doit pas déborder de la ligne.

APPLICATION PRATIQUE

Nous allons construire un ensemble utilitaire destiné à la mise au point des programmes Basic.

Il comprendra les fonctions classiques de numérotation automatique, d'édition d'une ligne et de suppression de lignes.

Ces modules, utilisables séparément, sont susceptibles d'être améliorés. Leurs principaux défauts sont soulignés.

Le but recherché était qu'ils soient courts, moins d'une page, et relogeables mais ce n'est pas un impératif.

Les trois programmes sont inclus en DATA dans un programme de chargement en Basic mais vous trouverez aussi un listing de désassemblage sommairement documenté.

Vous avez le choix pour l'entrée du code en mémoire

Si vous en avez la patience, vous pourrez compléter l'ensemble par un programme de renumérotation, assez volumineux, dont le code est reproduit en fin de chapitre.

Pour faciliter l'opération, nous placerons chacun des programmes en début de page. Ils sont relogeables (sauf Renum).

Suppression de lignes (D)el en #8F00. Edition d'une ligne (E)dit en #9000. Numérotation automatique (N)um en #9100. Renumérotation (R)enum en #9200 (éventuellement)

Ils seront enfin reliés par une espèce de menu, appelé par la commande !, qui permettra de les sélectionner facilement.

Vous pouvez dés maintenant fixer HIMEM à #8E00.

- suppression de lignes Basic.

Les paramètres doivent être transmis sous la forme:

A-Z où A est le numéro de la première ligne à supprimer et Z le numéro de la dernière.

A défaut d'un des numéros ou des deux, la limite est la première ou la dernière ligne du programme.

Ainsi, le trait d'union seul produit le même effet que NEW.

Vous remarquerez que ce programme, comme le suivant, n'assure pas la demande d'information.

Les paramètres peuvent être passés avec la commande ! (moyennant un DOKE#2F5 préalable avec son adresse) ou à l'aide du menu que nous établirons.

Adresses en ROM:

	ATMOS	ORIC-1	utilisation
LINGET	#CAE2	#CA98	recueil n°lign.limites -> LINNUM (#33,34)
FNDLIN	#C6B3	#C6DE	rech.adr.lignes -> LOWTR (#CE,CF)
SYNCHR	#DØ67	#CFDB	vérifie présence '-'
SETPTRS	#C7Ø8	#C733	TXTPTR début programme pour LINKSET
LINKSET	#C55F	#C56F	positionne octets de chainage
Ready2	#C4A8	#C4B5	retour entrée interpréteur.

- 1

1 !

Remarque importante:

Lorsque le programme est utilisé seul, l'ensemble constitué par la commande ! suivie des paramètres est codé par PARSE avant exécution. Ainsi, le trait d'union est interprété comme le symbole de la soustraction, signe -, et codé #CD.

C'est cette valeur qui est proposée à SYNCHR pour la vérification de la syntaxe dans le programme tel qu'il figure ci-dessous.

Dans notre ensemble utilitaire, les paramètres recueillis dans le buffer d'entrée ne seront pas codés, le trait d'union doit conserver son code ASCII normal soit #2D.

Il convient de changer #CD en #2D dans les données en DATA.

10 REM suppression de lignes 20 A=#8F00:FOR I=0 TO 100:READ C:POKE A+I,C:NEXT

ATMOS

à changer en #2D

100 DATA/#20,#E2,#CA,#20,#B3,#C6,#A5,#CE,#85,#93,#A5,#CF,#85,#94,#A9,#CD,#20

110 DATA #67, #DØ, #8, #A2, #FF, #86, #33, #86, #34, #28, #FØ, #9, #20, #E2, #CA, #E6, #33

120 DATA #D0, #2, #E6, #34, #20, #B3, #C6, #A5, #CE, #C5, #93, #A5, #CF, #E5, #94, #B0, #1

130 DATA #60, #A0, #0, #B1, #CE, #91, #93, #E6, #CE, #D0, #2, #E6, #CF, #E6, #93, #D0, #2

140 DATA #E6,#94,#A5,#9C,#C5,#CE,#A5,#9D,#E5,#CF,#B0,#E6,#A6,#94,#A4,#93,#D0

150 DATA #1, #CA, #88, #86, #9D, #84, #9C, #20, #8, #C7, #20, #5F, #C5, #4C, #A8, #C4

ORIC-1

à changer en #2D

100 DATA/#20,#98,#CA,#20,#DE,#C6,#A5,#CE,#85,#93,#A5,#CF,#85,#94,#A9,#CD,#20

110 DATA #DB, #CF, #8, #A2, #FF, #86, #33, #86, #34, #28, #F0, #9, #20, #98, #CA, #E6, #33

120 DATA #D0, #2, #E6, #34, #20, #DE, #C6, #A5, #CE, #C5, #93, #A5, #CF, #E5, #94, #B0, #1

130 DATA #60, #A0, #0, #B1, #CE, #91, #93, #E6, #CE, #D0, #2, #E6, #CF, #E6, #93, #D0, #2

140 DATA #E6,#94,#A5,#9C,#C5,#CE,#A5,#9D,#E5,#CF,#B0,#E6,#A6,#94,#A4,#93,#D0

150 DATA #1, #CA, #88, #86, #9D, #84, #9C, #20, #33, #C7, #20, #6F, #C5, #4C, #85, #C4

Mis à part le recueil des numéros de ligne limites et la vérification de la syntaxe (trait d'union), le programme est identique à celui de la page 94. Le pointeur temporaire #93,94 remplace INDEX (#91,92) car ce dernier est utilisé par LINGET.

		ATMOS		ORIC-1	
#8FØØ	36608	20 E2 CA	JSR #CAE2	20 98 CA	JSR #CA98
#8FØ3	36611	20 B3 C6	JSR #C6B3	20 DE C6	JSR #C6DE
#8FØ6	36614	A5 CE	LDA #CE	A5 CE	LDA #CE
#8FØ8	36616	85 93	STA #93	85 93	STA #93
#8FØA	36618	A5 CF	LDA #CF	A5 CF	LDA #CF
#8FØC	36620	85 94	STA #94	85 94	STA #94
#8FØE	36622	A9 CD	LDA >#CD	A9 CD	LDA >#CD
#8F1Ø	36624	20 67 DØ	JSR #DØ67	20 DB CF	JSR #CFDB
#8F13	36627	08	PHP	Ø8	PHP
#8F14	36628	A2 FF	LDX >#FF	A2 FF	LDX >#FF
#8F16	36630	86 33	STX #33	86 33	STX #33
#8F18	36632	86 34	STX #34	86 34	STX #34
#8F1A	36634	28	PLP	28	PLP
#8F1B	36635	FØ Ø9	BEQ #8F26	FØ Ø9	BEQ #8F26
#8F1D	36637	20 E2 CA	JSR #CAE2	20 98 CA	JSR #CA98
#8F2Ø	36640	E6 33	INC #33	E6 33	INC #33
#8F22	36642	DØ Ø2	BNE #8F26	DØ Ø2	BNE #8F26
#8F24	36644	E6 34	INC #34	E6 34	INC #34
#8F26	36646	20 B3 C6	JSR #C6B3	20 DE C6	JSR #C6DE
#8F29	36649	A5 CE	LDA #CE	A5 CE	LDA #CE
#8F2B	36651	C5 93	CMP #93	C5 93	CMP #93
#8F2D	36653	A5 CF	LDA #CF	A5 CF	LDA #CF
#8F2F	36655	E5 94	SBC #94	E5 94	SBC #94
#8F31	36657	BØ Ø1	BCS #8F34	BØ Ø1	BCS #8F34
#8F33	34659	60	RTS	60	RTS
#8F34	36660	AØ ØØ	LDY >#00	AØ ØØ	LDY >#00
#8F36	36662	B1 CE	LDA (#CE),Y	B1 CE	LDA (#CE),Y
#8F38	36664	91 93	STA (#93),Y	91 93	STA (#93),Y
#8F3A	36666	E6 CE	INC #CE	E6 CE	INC #CE
#8F3C	36668	DØ Ø2	BNE #8F40	DØ Ø2 E6 CF	BNE #8F4Ø INC #CF
#8F3E	36670	E6 CF E6 93	INC #CF INC #93	E6 93	INC #93
#8F4Ø #8F42	36672 36674	DØ Ø2	BNE #8F46	DØ Ø2	BNE #8F46 .
#8F44	36676	E6 94	INC #94	E6 94	INC #94
#8F46	36678	A5 9C	LDA #9C	A5 9C	LDA #9C
#8F48	36680	C5 CE	CMP #CE	C5 CE	CMP #CE
#8F4A	36682	A5 9D	LDA #9D	A5 9D	LDA #9D
#8F4C	36684	ES CF	SBC #CF	E5 CF	SBC #CF
#8F4E	36686	BØ E6	BCS #8F36	BØ E6	BCS #8F36
#8F5Ø	36688	A6 94	LDX #94	A6 94	LDX #94
#8F52	36690	A4 93	LDY #93	A4 93	LDY #93
#8F54	36692	DØ Ø1	BNE #8F57	DØ Ø1	BNE #8F57
#8F56	36694	CA	DEX	CA	DEX
#8F57	36695	88	DEY	88	DEY
#8F58	36696	86 9D	STX #9D	86 9D	STX #9D
#8F5A	36698	84 9C	STY #9C	84 9C	STY #9C
#8F5C	36700	20 08 07	JSR #C7Ø8	2Ø 33 C7	JSR #C733
#8F5F	36703	20 5F C5	JSR #C55F	20 6F C5	JSR #C56F
#8F62	36706	4C A8 C4	JMP #C4A8	4C B5 C4	JMP #C4B5
	10.100 VG (100) (TeSt 10)				

- Edition d'une ligne Basic.

Cette fonction, évoquée brièvement page 115, est déficiente sur ORIC. En particulier, l'insertion ou la suppression de caractères à l'intérieur d'une ligne sont compliquées.

Il est malaisé d'implanter ces facilités en raison du mode de gestion de l'écran texte.

Dans le but de faire simple et court, j'ai adopté une solution radicale, quoique peu élégante, pour le programme qui suit.

Elle consiste à afficher la ligne à modifier sur 40 colonnes à partir d'une adresse fixe de l'écran.

Ainsi, les adresses des caractères forment une suite continue, sans interruption par les colonnes protégées du mode normal.

L'insertion et la suppression sont simplifiées, de même que la saisie de la ligne modifiée.

Le programme, conçu à l'origine pour ORIC-1, a été adapté à l'ATMOS avec quelques difficultés. Il devrait fonctionner correctement, sauf vice caché hélas toujours possible.

Il peut être employé seul, la ligne à modifier étant appelée au moyen de la commande ! suivie du numéro.

Les routines classiques, LINGET, FNDLIN, CLS, OUTDO, INCHR etc, sont utilisées.

Vous noterez l'emploi des variables système #209, 268, 2F2 et surtout 26A pour le mode d'affichage (curseur, 38 ou 40 col).

Après la lecture du n° de ligne (saut à UNDF'D STAT ERR si absent), le drapeau #2F2 est levé pour empêcher la sortie de LIST. Remarquez la technique employée. La ligne ayant été trouvée par FNDLIN, l'indicateur de retenue C est à un. Il devient le bit 7 de #2F2 grâce à l'instruction ROR.

Après mise en 40 colonnes, LIST est appelée. Entrée dans la routine à l'adresse ad-hoc pour l'affichage d'une seule ligne.

Le nombre de caractères de la ligne est calculé et rangéen #82, puis le curseur, dont la position est indexée par X, est placé sur le premier chiffre du numéro de ligne.

La routine INCHR recueille le caractère entré au clavier qui est analysé.

Certains caractères de controle sont actifs.

Il s'agit de:

- CTRL-H, flèche gauche, code #08, recul du curseur, associé à SHIFT il ramène le curseur en début de ligne.
- CTRL-I, flèche droite, code #09, avance le curseur, jusqu'en fin de ligne avec SHIFT.
- CTRL-T, maj./min, code #14, fonction normale.
- CTRL-X, code #18, suppression, efface le caractère sous le curseur.
- CTRL-Z, code #1A, bascule insertion; un témoin est affiché en haut de l'écran quand ce mode est activé.
- ESC, code #1B, saisie de la ligne après modification. Il m'a semblé préférable à RETURN pour éviter une frappe intempestive.

La nouvelle ligne est rangée dans le buffer d'entrée puis le programme rentre dans CMDLP qui assure la mise en place.

Vous devriez suivre sans trop de difficultés la manipulation des caractères en mode insertion ou suppression.

La validation de la ligne modifiée est assurée par ESC quelle que soit la position du curseur.

10 REM edition d'une ligne Basic 20 A=#9000:FOR I=0 TO 254:READ C:POKE A+I,C:NEXT

ATMOS

100 DATA #20, #E2, #CA, #20, #B3, #C6, #B0, #3, #4C, #23, #CA, #6E, #F2, #2, #A9, #22, #BD

110 DATA #6A,#2,#20,#CE,#CC,#20,#6C,#C7,#A0,#51,#88,#B9,#D0,#BB,#C9,#21,#90

120 DATA #F8,#84,#82,#A9,#D0,#85,#12,#A9,#D,#20,#D9,#CC,#20,#D1,#CC,#A9,#9

130 DATA #20, #D9, #CC, #A2, #0, #A9, #20, #85, #80, #8D, #82, #88, #20, #E8, #C5, #C9, #1A

140 DATA #D0,#6,#A5,#80,#49,#80,#D0,#EE,#C9,#8,#D0,#F,#E0,#0,#F

150 DATA #20, #D9, #CC, #2C, #9, #2, #10, #E2, #30, #F1, #C9, #9, #D0, #F, #E4, #82, #F0

160 DATA #D8,#20,#D9,#CC,#E8,#2C,#9,#2,#10,#CF,#30,#F1,#C9,#14, #F0,#3E,#C9

170 DATA #18, #D0, #1F, #E0, #0, #F0, #C1, #E4, #82, #F0, #BD, #86, #83, #A4, #82, #BD, #D2

180 DATA #BB, #9D, #D1, #BB, #E8, #88, #C4, #83, #D0, #F4, #A6, #83, #C6, #82, #B8, #50, #A6

190 DATA #C9,#18,#F0,#3E,#C9,#20,#90,#9E,#C9,#7E,#B0,#9A,#E0,#4 E,#F0,#96,#24

200 DATA #80,#30,#C,#E4,#82,#D0,#2,#E6,#82,#E8,#20,#D9,#CC,#D0,#DD,#E4,#82

210 DATA #F0, #D9, #A4, #82, #C0, #4E, #F0, #D3, #86, #83, #48, #98, #38, #E 5, #83, #AA, #B9

220 DATA #D1, #BB, #99, #D2, #BB, #88, #CA, #10, #F6, #68, #A6, #83, #BB, #50, #D6, #20, #D1

230 DATA #CC, #A4, #82, #A2, #0, #BD, #D1, #BB, #9D, #35, #0, #E8, #88, #D0, #F6, #A9, #20

240 DATA #8D,#82,#BB,#A9,#3,#8D,#6A,#2,#20,#EC,#CB,#20,#2,#CC,# 4C,#BD,#C4 Remarque concernant la version ATMOS:

Si, au listage, la ligne comporte 79 caractères le programme ne fonctionnera pas correctement (l'impasse est faite sur l'octet fort de l'adresse de base, #13).

Cela peut arriver si la ligne entrée est longue de 78 caractères sans espace après le numéro. La commande LIST en place un automatiquement.

Dans ce cas, la seule solution est d'appuyer sur ESC, de faire CALL#F8B2 (RESET) et de retaper la ligne. Vous auriez aussi des problèmes avec la commande EDIT.

Conclusion, laissez un espace après le numéro de ligne et n'entrez pas plus de 78 caractères (voir programme suivant, Autonum).

ORIC-1

100 DATA #20,#98,#CA,#20,#DE,#C6,#B0,#3,#4C,#F1,#C9,#6E,#F2,#2, #A9,#22,#8D

110 DATA #6A, #2, #20, #A, #CC, #20, #99, #C7, #A0, #51, #88, #B9, #CF, #BB, #C9, #21, #90

120 DATA #F8, #84, #82, #A9, #23, #8D, #6A, #2, #A2, #0, #A9, #1, #8D, #68, #2, #20, #9F

130 DATA #CB,#A9,#20,#85,#80,#8D,#82,#BB,#20,#FB,#C5,#C9,#1A,#D 0,#6,#A5,#80

140 DATA #49,#80,#D0,#EE,#C9,#8,#D0,#F,#E0,#0,#F0,#EB,#CA,#20,#
12,#CC,#2C

150 DATA #9,#2,#10,#E2,#30,#F1,#C9,#9,#D0,#F,#E4,#82,#F0,#D8,#20,#12,#CC

160 DATA #E8, #2C, #9, #2, #10, #CF, #30, #F1, #C9, #14, #F0, #3E, #C9, #18, #D0, #1F, #E0

170 DATA #0, #F0, #C1, #E4, #82, #F0, #BD, #86, #83, #A4, #82, #BD, #D1, #BB, #9D, #D0, #BB

180 DATA #E8, #88, #C4, #83, #D0, #F4, #A6, #83, #C6, #82, #B8, #50, #A6, #C 9, #1B, #F0, #3E

190 DATA #C9,#20,#90,#9E,#C9,#7E,#B0,#9A,#E0,#4E,#F0,#96,#24,#8 0,#30,#C,#E4

200 DATA #82, #D0, #2, #E6, #82, #E8, #20, #12, #CC, #D0, #DD, #E4, #82, #F0, #D7, #A4, #82

210 DATA #C0,#4E,#F0,#D3,#86,#83,#48,#98,#38,#E5,#83,#AA,#B9,#D 0,#BB,#99,#D1

220 DATA #BB, #88, #CA, #10, #F6, #68, #A6, #83, #88, #50, #D6, #A9, #11, #2 0, #12, #CC, #A4

230 DATA #82, #A2, #0, #BD, #D0, #BB, #9D, #35, #0, #E8, #88, #D0, #F6, #A9, #20, #8D, #82

240 DATA #BB, #A9, #3, #8D, #6A, #2, #8D, #68, #2, #20, #9B, #CB; #4C, #CD, #C4, #0, #0

			4000000					960-3000				
#9000	20 E2			#CAE2			#9089		D1	BB	STA	#BBD1,X
#9003	2Ø B3			#C6B3			#9Ø8C	E8			INX	
#9006	BØ Ø3	5	BCS	#900B			#9Ø8D	88			DEY	
#9008	4C 23	CA	JMP	#CA23			#9Ø8E	C4	83		CPY	#83
#900B	6E F2	02	ROR	#2F2			#9090	DØ	F4		BNE	#9086
#900E	A9 22	2	LDA	>#22			#9092	A6				#83
#9010		02		#26A	26	820	#9094	63				#82
#9013	20 CE			#CCCE			#9096	B8			CLV	1102
#9016		C7		#C76C			#9097	50	Λ4			#9Ø3F
#9019	AØ 51			>#51			#9099		1B			>#1B
#901B	88	S.	DEY	71101								
		n n n		ADDDG V			#9Ø9B		3E			#90DB
#901C	89 DØ			#BBDØ,Y				C9	20			>#20
#901F	C9 21			>#21			#9Ø9F	90				#903F
#9021	90 FE			#9Ø1B			#90A1	69				>#7E
#9023	84 82		STY				#9ØA3		9A		BCS	#9Ø3F
#9025	A9 D0			>#DØ			#90A5	EØ			CPX	>#4E
#9027	85 12	2	STA	#12			#9ØA7	FØ	96		BEQ	#903F
#9029	A9 ØD)	LDA	>#ØD			#90A9	24	80		BIT	#80
#902B	20 D9	CC	JSR	#CCD9			#9ØAB	30	ØC		BMI	#9089
#902E	20 D1	CC	JSR	#CCD1			#9ØAD	E4				#82
#9031	A9 Ø5			>#09			#90AF	DØ				#9083
#9033		CC		#CCD9			#90B1	E6				#82
#9036	A2 Ø0			>#ØØ			#9ØB3	E8	UZ		INX	WOZ.
	A9 20			>#20					no.	00		#CCDC
#9038							#9ØB4		D9	LL		#CCD9
#903A	85 80			#80			#9ØB7	DØ	DD			#9096
#903C		2 BB		#BB82			#9ØB9	E4				#82
#903F	20 E8			#C5E8		31	#90BB	FØ				#9096
#9042	C9 16			>#1A			#90BD	A4				#82
#9044	DØ Ø8			#904C			#9ØBF	CØ			CPY	>#4E
#9046	A5 80	3	LDA	#80			#9ØC1	FØ	D3		BEQ	#9096
#9048	49 80	3	EOR	>#80			#9ØC3	86	83		STX	#83
#9Ø4A	DØ EE		BNE	#903A			#9ØC5	48			PHA	
#9Ø4C	C9 Ø8	3	CMP	>#Ø8	-		#9ØC6	98			TYA	
#9Ø4E	DØ ØF	=	BNE	#905F			#90C7	38			SEC	
#9050	EØ 00			>#ØØ			#9ØC8	E5	83			#83
#9052	FØ E			#903F			#90CA	AA			TAX	
#9054	CA		DEX				#9ØCB		Di	BB		#BBD1,Y
#9055	20 D9	ם כר		#CCD9			#90CE			BB		
	20 09			#209					UZ	DB		#BBD2,Y
#9058							#9ØD1	88			DEY	
#9Ø5B	10 E2			#903F			#9ØD2	CA	-,		DEX	
#9Ø5D	30 F1			#9050			#90D3		F6			#9ØCB
#9Ø5F	C9 Ø9			>#09			#9ØD5	88			PLA	
#9061	DØ ØF			#9072			#90D6		83			#83
#9063	E4 82			#82			#9ØD8	88			CLV	
#9065	FØ D8	3	BEQ	#903F			#9ØD9	50	D6		BVC	#90B1
#9067	20 D9	7 CC	JSR	#CCD9			#90DB	20	Di	CC	JSR	#CCD1
#906A	E8		INX				#90DE	A4	82		LDY	#82
#9Ø6B	20 09	02	BIT	#209			#90E0	A2	00		LDX	>#00
#906E	10 CF	=		#903F		*	#90E2		D1	BB		#BBD1,X
#9070	30 F			#9063			#90E5		35			#35,X
#9072	C9 14			>#14			#90E8	E8			INX	,
#9074	FØ 38			#9ØB4			#90E9	88			DEY	
#9076	C9 18			>#18			#90EA		F6			#9ØE2
#9078	DØ 15			#9099			#90EC		20	DP.		>#20
#907A	EØ Ø			>#ØØ			#9ØEE		82	RR		#BB82
#907C	FØ C			#903F			#9ØF1		03			>#03
#907E	E4 82			#82			#90F3		6A			#26A
#9080	FØ BI			#9Ø3F			#90F6		EC		JSR	#CBEC
#9082	84 83			#83			#90F9-	20	02	CC	JSR	#CCØ2
#9084	A4 82	2	LDY	#82			#90FC	4C	BD	C4	JMP	#C4BD
#9086	BD D	2 BB	LDA	#BBD2,X								

#9000	20	98	CA	JSR	#CA98	#9088	E8			INX	
#9003	20	DE	C9	JSR	#C6DE	#9089	88			DEY	
#9006	BØ	03		BCS	#900B	#9Ø8A	C4	83		CPY	#83
#7008	4C	F1	C9	JMP	#C9F1	#9Ø8C	DØ	F4		BNE	#9082
#900B	6E	F2	02	ROR	#2F2	#9Ø8E	A6	83		LDX	#83
#900E	A9	22		LDA	>#22	#9090		82		DEC	#82
#9010	80		02		#26A	#9092	88			CLV	
#9013	20		CC		#CCØA	#9093		A6			#9Ø3B
#9016	20		C7		#C799	#9095		18		CMP	>#1B
#9019		51			>#51	#9097		3E			#9ØD7
#901B	88			DEY		#9099		20		CMP	
#901C		CE	BB		#BBCF,Y	#909B	90			BCC	#9Ø3B
#901F		21		CMP		#909D	C9	7E		CMP	>#7E
#9021		FB			#9Ø1B	#909F	BØ			BCS	
#9023		82			#82	#9ØA1	EØ	4E		CPX	>#4E
#9025		23			>#23	#90A3		96		BEQ	#9Ø3B
#9027	8D		02		#26A	#90A5	24				#80
#7027 #702A	A2	00	UZ	LDX	>#00	#70A7	30	ØC		BMI	#9ØB5
#702H #902C	A9				>#01	#9ØA9		82			
#702C	8D		02		#268	#90AB		02		BNE	#90AF
	20		CB		#CB9F	#70AD		82		INC	#82
#9031		20	LD		>#20	#70AD	EB	02		INX	#02
#9034		80			#80	#70B0	20	12	CC		#CC12
#9036	85		DD.		#BB82		DØ	DD	LL	BNE	#9092
#9038	8D		BB			#9ØB3					
#903B	20		C5		#C5F8	#9ØB5	E4			CPX	
#9Ø3E	C9	1A			>#1A	#9ØB7	FØ	D9		BEQ	#9092
#9040	DØ	06			#9048	#9089	A4	82		LDY	#82
#9042	A5	80			#80	#7ØBB		4E		CPY	>#4E
#9044	49	80		EOR		#90BD		D3		BEQ	#9092
#9046	DØ	EE			#9036	#90BF	86	83		STX	#83
#9048	C9	Ø8		CMP	>#Ø8	#9ØC1	48			PHA	
#7Ø4A	DØ	ØF		BNE	#905B	#90C2	98			TYA	
#9Ø4C	EØ	00		CPX	>#00	#9ØC3	38			SEC	
#9Ø4E	FØ	EB		BEQ	#9 0 3B	#9ØC4	E5	83		SBC	#83
#9050	CA			DEX		#9ØC6	AA			TAX	
#9051	20		CC		#CC12	#9ØC7	B9		BB	LDA	
#9054	2C		02		#209	#90CA		D1	BB		#BBD1,Y
#9057		E2			#903B	#90CD	88			DEY	
#9059		F1			#9Ø4C	#90CE	CA	10.00000.11		DEX	
#905B		09			>#09	#90CF		F6			#9ØC7
#9Ø5D		ØF			#906E	#9ØD1	88			PLA	
#905F		82			#82	#9ØD2	A6	83		LDX	#83
#9061		D8			#9Ø3B	#9ØD4	B8			CLV	
#9063		12	CC		#CC12	#90D5	50				#9ØAD
#9066	E8			INX		#9ØD7	A9				>#11
#9067		09	02		#209	#9ØD9			CC		#CC12
#9Ø6A		CF			#9Ø3B	#90DC	A4			LDY	
#906C	30	Fi			#905F	#9ØDE	A2				>#00
#9Ø6E	C9	14			>#14	#90E0			BB		#BBDØ,X
#9070	FØ	3E		BEQ	#90B0	#9ØE3		35	00		#35,X
#9072	C9	18		CMP	>#18	#90E6	E8			INX	
#9074	DØ	1F		BNE	#9095	#9ØE7	88			DEY	
#9076		00			>#ØØ	#9ØE8	DØ				#90E0
#9078	FØ	C1			#903B	#90EA	A9				>#20
#9Ø7A	E4	82		CPX	#82	#9ØEC	8D		BB	STA	#BB82
#9Ø7C	FØ	BD		BEQ	#903B	#90EF	A9				>#Ø3
#907E	86	83			#83	#90F1	8D				#26A
#9080	A4	82				#90F4	8D				#268
#9082			BB		#BBD1,X		20				#CB9B
#9085	9D	DØ	BB	STA	#BBDØ,X	#90FA	4C	CD	C4	JMP	#C4CD

- Numérotation automatique.

Ce programme est instructif car il fait appel à de nombreuses routines du système, notamment à des routines virgule flottante.

Il commence par un test de l'octet pointé par TXTTAB.

Si c'est zéro, il n'y a pas de programme en mémoire. Le numéro initial sera 10, (00+0A, #9149).

Si un programme est présent, une boucle de recherche est entamée (#9106-91ED) pour déterminer le numéro de ligne le plus élevé. Le programme est parcouru par transferts successifs du contenu des octets de chainage dans les pointeurs #91,92 et #93,94.

Au passage, à chaque fois, le pointeur #10,11 recueille le numéro de ligne.

Lorsque la fin de programme est rencontrée, octet fort de chainage à zéro (#9117), ce pointeur contient le numéro de la dernière ligne.

Le numéro est mis en FAC, puis arrondi à la dizaine inférieure par DIVIØ, INT et MULIØ. Après GETADR, qui le convertit en un entier sur deux octets dans LINNUM, le rôle de cette deuxieme partie est terminé.

Seule la dernière partie (#9142-919D) devient active. Le numéro de ligne dans LINNUM est mis ou remis en FAC. ADDACC ajoute 10 (#0A) au contenu de FAC qui est converti par GETADR dans LINNUM.

Le numéro de ligne occupe alors son emplacement correct pour la future insertion de la ligne. Il est affiché par LINPRT.

OUTSPC le fait suivre d'un espace.

La routine INLIN est appelée pour l'entrée du texte de la ligne. A défaut d'entrée (RETURN), il y a sortie du programme par Ready2 (#9165).

Seul le texte est placé dans le buffer d'entrée pour analyse et codage par PARSE.

Après mise en place de la ligne à l'adresse inscrite dans LOWTR (#CE,CF) par FNDLIN et le réglage des pointeurs et des octets de chainage, le programme retourne en #9142 pour la ligne suivante.

Principal défaut de ce programme, il ne fait pas entrer le numéro de ligne dans le buffer d'entrée.

Une ligne peut ainsi être longue de 78 caractères en plus du numéro.

D'autre part, et c'est lié au mode de fonctionnement de INLIN, on ne sort pas de cette routine en cas de dépassement du nombre de caractères.

Il convient d'être vigilant.

Si cela arrivait, indiqué par le 'slash' (\), ne retapez pas le numéro de ligne. Il est déjà enregistré en mémoire.

Simplement, entrez le texte à nouveau.

Autre caractèristique, puisque LINGET n'est pas utilisé, les numéros de ligne peuvent dépasser 63999 avec les particularités que cela comporte (page 23).

10 REM autonum

1

.----

20 A=#9100:FOR I=0 TO 161:READ C:POKE A+I,C:NEXT

- 100 DATA #A0, #0, #B1, #9A, #F0, #40, #A5, #9A, #85, #91, #A5, #9B, #85, #92, #A0, #0, #B1
- 110 DATA #91,#85,#93,#C8,#B1,#91,#F0,#16,#85,#94,#C8,#B1,#91,#85,#10,#C8,#B1
- 120 DATA #91,#85,#11,#A5,#93,#85,#91,#A5,#94,#85,#92,#D0,#DF,#A4,#10,#A5,#11
- 130 DATA #20,#40,#DF,#20,#C3,#DD,#20,#BD,#DF,#20,#A7,#DD,#20,#22,#D9,#A4,#33
- 140 DATA #A5,#34,#20,#40,#DF,#A9,#A,#20,#76,#E0,#20,#22,#D9,#A6, #33,#A5,#34
- 150 DATA #20, #C5, #E0, #20, #D4, #CC, #20, #92, #C5, #86, #E9, #84, #EA, #20, #E2, #0, #F0
- 160 DATA #38,#20,#FA,#C5,#84,#26,#20,#B3,#C6,#18,#A5,#9C,#85,#C9,#65,#26,#85
- 170 DATA #C7, #A4, #9D, #84, #CA, #90, #1, #C8, #84, #C8, #20, #F4, #C3, #A5, #A0, #A4, #A1
- 180 DATA #85, #9C, #84, #9D, #A4, #26, #88, #89, #31, #0, #91, #CE, #88, #10, #F8, #20, #8
 - 190 DATA #C7, #20, #5F, #C5, #F0, #A3, #4C, #A8, #C4

ORIC-1

- 100 DATA #A0, #0, #B1, #9A, #F0, #40, #A5, #9A, #85, #91, #A5, #9B, #85, #92, #A0, #0, #B1
- 110 DATA #91-,#85,#93,#C8,#B1,#91,#F0,#16,#85,#94,#C8,#B1,#91,#85,#10,#C8,#B1
- 120 DATA #91,#85,#11,#A5,#93,#85,#91,#A5,#94,#85,#92,#D0,#DF,#A4,#10,#A5,#11
- 130 DATA #20, #D5, #D8, #20, #BF, #DD, #20, #A5, #DF, #20, #A3, #DD, #20, #A7, #D8, #A4, #33
- 140 DATA #A5, #34, #20, #D5, #D8, #A9, #A, #20, #72, #E0, #20, #67, #D8, #A6, #33, #A5, #34
- 150 DATA #20, #C1, #E0, #20, #D, #CC, #20, #A2, #C5, #86, #E9, #84, #EA, #20, #E2, #0, #F0
- 160 DATA #38,#20,#A,#C6,#84,#26,#20,#DE,#C6,#18,#A5,#9C,#85,#C9, #65,#26,#85
- 170 DATA #C7, #A4, #9D, #84, #CA, #90, #1, #C8, #84, #C8, #20, #F8, #C3, #A5, #A0, #A4, #A1
- 180 DATA #85,#9C,#84,#9D,#A4,#26,#88,#B9,#31,#0,#91,#CE,#88,#10, #F8,#20,#33
 - 190 DATA #C7, #20, #6F, #C5, #F0, #A3, #4C, #B5, #C4

ATMOS ORIC-1

```
GIVAYF2
          #DF4Ø
                 #D8D5
                        convertit en virg. flot. nb. en Y(-),A(+).
DIVIO
          #DDC3
                 #DD8F
                        divise le contenu de FAC par 10.
                        plus grande valeur entière de FAC.
INT
          #DFBD
                 #DFA5
                 #DDA3
                        multiplie le contenu de FAC par 10.
MUL10
          #DDA7
          #D922
                 #D867
                        FAC -> entier dans LINNUM (#33,34).
GETADR
                 #EØ72
                        ajoute à FAC le contenu de l'accumulateur.
ADDACC
          #EØ76
                        affiche en décimal le nombre en X(-),A(+).
LINPRT
          #EØC5
                 #EØC1
OUTSPC
          #CCD4
                 #CCØD
                        affiche un espace.
INLIN
          #0592
                #C5A2 recueille texte dans le buffer d'entrée.
                       analyse et codage de la ligne entrée.
PARSE
          #C5FA
                #C60A
                #C6DE recherche ligne dont n° est en LINNUM.
          #C6B3
FNDLIN
BLTU
          #C3F4
                 #C3F8
                       transfert de bloc.
SETPTRS
          #C7Ø8
                 #C733
                       place TXTPTR début programme puis CLEARC.
          #C55F
                 #C56F
                        positionne octets de chainage du programme
LINKSET
READY2
          #C4A8
                 #C4B5
                        attente d'une commande, CMDLP.
```

```
AØ ØØ
                  LDY >#00
#9100
        B1 9A
#9102
                  LDA (#9A),Y
#9104
        FØ 40
                  BEQ #9146
        A5 9A
#9106
                  LDA #9A
#9108
        85 91
                  STA #91
        A5 98
#910A
                  LDA #9B
        85 92
                  STA #92
#910C
        AØ ØØ
#910E
                  LDY >#00
        B1 91
                  LDA (#91),Y
#9110
        85 93
                  STA #93
#9112
        CB
                   INY
#9114
#9115
        B1 91
                   LDA (#91),Y
#9117
        FØ 16
                   BEQ #912F
#9119
        85 94
                   STA #94
#911B
        C8
                   INY
        B1 91
                   LDA (#91),Y
#911C
        85 10
                   STA #10
#911E
        C8
                   INY
#9120
#9121
        B1 91
                   LDA (#91),Y
#9123
        85 11
                   STA #11
        A5 93
                  LDA #93
#9125
#9127
        85 91
                   STA #91
                  LDA #94
#9129
        A5 94
                   STA #92
#912B
        85 92
                   BNE #910E
        DØ DF
#912D
```

idem ATMOS, ORIC-1

ATMOS			ORIC-1	
#912F	A4 10 LDY	′ #10		OY #10
#9131		#11		OA #11
#9133		#DF4Ø		SR #D8D5
#9136		#DDC3		SR #DDBF
#9139		t #DFBD		SR #DFA5
#913C		#DDA7		SR #DDA3
#913F		#D922		SR #D867
#9142		' #33		OY #33
#7142 #9144		#34		DA #34
#9146		#DF40		SR #D8D5
#9149		>#ØA		OA >#ØA
#914B		#E076		SR #EØ72
#914E		#D922		SR #D867
#714E #9151		#33		OX #33
#9153		#34		DA #34
#7133 #9155		#EØC5		SR #EØC1
		#CCD4		SR #CCØD
#9158 #915B	20 92 C5 JSF			SR #C5A2
#915E		(#C592 (#E9		TX #E9
#9160		/ #EA		TY #EA
#7160 #7162		#E2		SR #E2
#9165		1 #919F		EQ #919F
#9167		#C5FA		SR #C60A
#916A	84 26 STY	426		TY #26
#716H		#C6B3		SR #C6DE
	18 CL0			_C
#916F		, 1 #9C		DA #9C
#9170		#C9		TA #C9
#9172 #9174		; #26		DC #26
		. #20 1 #C7		TA #C7
#9176 #9178		' #9D		DY #9D
#7176 #917A		′ #CA		TY #CA
#717H #917C		#917F		CC #917F
	CB IN			4Y 4Y
#917E #917F		′ #C8		TY #C8
	20 F4 C3 JSF		20 FB C3 J9	
#9181 #9184		4AØ		DA #AØ
		' #A1		DY #A1
#9186		49C		TA #9C
#9188		/ #9D		TY #9D
#918A		′ #26		DY #26
#918C	88 DE\			EY #20
#918E		#31,Y		DA #31,Y
#918F	91 CE STA			
#9192	88 DE	the commence of the commence o		TA (#CE),Y EY
#9194		. #918F		-: PL #918F
#9195		. #C708		SR #C733
#9197		* #C55F		SR #C56F
#919A				EQ #9142
#919D		#9142 # #6400		MP #C4B5
#91 <i>9</i> F	4C A8 C4 JMF	#C4AB	40 00 04 dl	.ı∟ #८4₽9

- menu

Pour batir un ensemble cohérent il nous reste à installer un 'menu'. Le code machine est à la page suivante. Vous vous servirez de votre utilitaire pour l'entrer en mémoire. Auparavant, par sécurité, entrez à nouveau l'instruction: HIMEM#8E00.

Le module en #8E00 est en deux parties:

- La première en #8E00 donne la valeur #8E11 au vecteur en #2F5 ce qui permettra d'appeler le menu avec la commande ! . Elle protége l'ensemble en faisant HIMEM=#8E00, puis revient au Basic.
- La deuxième partie en #8E11 est le menu lui-même, appelé par ! Il fait deux choses:
 - il permet le choix d'une option à l'aide de sa première lettre.
 - il recueille dans le buffer d'entrée les paramètres éventuels.

Si pour une raison ou une autre, le vecteur ! était débranché, il suffira d'un CALL#8E00 pour le reconnecter.

Procédure: commande ! <RETURN> puis D, E, N ou R.

Vous trouverez dans les deux dernières pages le code machine d'un programme de renumérotation que vous inscrirez à partir de l'adresse #9200.

C'est facultatif. Si vous le faites, n'oubliez pas de modifier le saut de l'adresse #8E48 en JMP #9200, 4C 00 92.

Pour l'utiliser sur ORIC-1:

DOKE#9669, #CBED : DOKE#9681, #FA85 : DOKE#9684, #C73A

Ce programme est une adaptation, assez sommaire, du RENUMBER de l'Apple //.

Le code est probablement redondant mais le programme semble fonctionner correctement.

Son gros défaut est qu'il supprime les espaces après tous les mots-clés qui pourraient être concernés par la renumérotation, c'est-à-dire GOTO, RUN, GOSUB, LIST, ELSE, THEN.

Il faudrait remanier le programme pour trouver une autre solution, à vous de jouer!

J'ai inclus LIST bien que cette commande ne soit pas normalement utilisable en mode programme (voir page 113). Ainsi, par exemple, LIST 10-100 sera traité.

Le mode d'emploi est le suivant: option R, RENUM suivi ou non des paramètres:

- A -> numéro souhaité de la première ligne renumérotée.
- P -> pas
- D -> début renumérotation, n° lere ligne à renuméroter.
- F -> fin, n° dernière ligne à renumérater.

Exemple: RENUM A2000, P10, D500, F750

A défaut A = 10 et P = 10

Avant de procéder aux essais, sauvegardez le tout -

CSAVE"", A#8EØØ, E#97FF, AUTO

	ATMOS	ORIC-1	
#8E00: #8E02: #8E05: #8E07: #8E09: #8E0B: #8E0E: #8E10:	A9 11 8D F5 02 A9 00 85 A6 A9 8E 8D F6 02 85 A7	A9 11 8D F5 02 A9 00 85 A6 A9 8E 8D F6 02 85 A7 60	LDA >#11 ;fixe STA #2F5 ;vecteur ! LDA >#00 ; = #8E11 STA MEMSIZ ;et LDA >#8E ;HIMEM STA #2F6 ; = #8E00 STA MEMSIZ+1 RTS
#8E11:	A9 3E	A9 3E menu	LDA '>' ;entrée menu ! JSR OUTDO ;affiche symbole > JSR INCHR ;option dans A CMP 'D' ;est-ce 'D'
#8E13:	20 D9 CC	20 12 CC	
#8E16:	20 E8 C5	20 F8 C5	
#8E19:	C9 44	C9 44 del?	
#8E1B:	DØ ØA	DØ ØA	BNE edit? ;non, voir 'E' LDA M1L ;pour aff.'DEL' JSR param ;param dans BUF
#8E1D:	A9 5D	A9 5D	
#8E1F:	20 4E 8E	20 4E 8E	
#8E22:	FØ 27	FØ 27	BEQ sortie ;pas d'entrée JMP DEL ;suppression CMP 'E' ;est-ce 'D' BNE num? ;non. voir 'N'
#8E24:	4C ØØ 8F	4C 00 8F	
#8E27:	C9 45	C9 45 edit?	
#8E29:	DØ ØA	DØ 0A	
#8E2B:	A9 62	A9 62	BNE num? ;non, voir 'N' LDA M2L ;pour aff.'EDIT' JSR param ;param dans BUF BCS sortie ;pas un chiffre
#8E2D:	20 4E 8E	20 4E 8E	
#8E3Ø:	80 19	B0 19	
#8E32:	4C 00 70	4C 00 90	JMP EDIT ; édition CMP 'N' ; est-ce 'N' BNE renum? ; non, voir 'R' JSR CRDO ; à la ligne
#8E35:	C9 4E	C9 4E num?	
#8E37:	D0 06	D0 06	
#8E39:	20 F0 CB	20 9F CB	
#8E3C:	4C 00 91	4C 00 91	JMP NUM ; numérotation
#8E3F:	C9 52	C9 52 renum?	CMP 'R' ; est-ce 'R'
#8E41:	D0 08	D0 08	BNE sortie ; non, on sort
#8E43: #8E45: #8E48: #8E4B:	A9 68 00 00 00 00 00 00 00 00 00 00 00 00 00	A9 68 00 00 20 4E 8E 4C 85 C4 4C 85 C4 sortie	LDA M3L ;pour aff.'RENUM' JSR param ;param dans BUF JMP RENUM ;renumérotation JMP Ready2 ;retour Basic
#8E4E:	AØ 8E	AØ 8E param	LDY MH ;oct.fort adr.msg JSR STROUT ;affiche mnémo
#8E5Ø:	20 BØ CC	20 ED CB	
#8E53:	20 92 C5	20 A2 C5	JSR INLIN ; recueil param. STX TXTPTR ; TXTPTR=BUF-1 STY TXTPTR+1 ; pour CHARGET JMP CHARGET ; 1er car dans A
#8E56:	86 E9	86 E9	
#8E58:	84 EA	84 EA	
#8E5A:	4C E2 00	4C E2 00	
#8E5D; #8E62; #8E68;	44 45 4C 2Ø 45 44 49 54 52 45 4E 55	00 M1 20 00 M2	'DEL '00 'EDIT '00 'RENUM '00

#9200: A4 E9 8C 91 96 A4 EA 8C 92 96 A0 00 B1 9A D0 03 #9210: 4C 64 96 A9 FF A2 0A A0 00 85 D0 85 D1 84 D2 84 #9220: D3 86 D4 84 D5 86 D6 84 D7 20 90 96 90 1D F0 4B #9230: DØ ØA 20 90 96 C9 2C DØ 12 20 88 96 A2 Ø3 DD F4 #9240: 97 F0 0D CA 10 F8 E8 C9 80 F0 05 A9 84 4C 66 96 #9250: 8A 0A A8 20 88 96 F0 06 90 0A C9 2C D0 F5 A2 00 #9260: 86 DB FØ Ø5 20 9F 96 A6 DA 96 DØ A6 DB 96 D1 EØ #9270: FA 90 05 A9 8C 4C 66 96 AA D0 B7 A5 D4 05 D5 D0 #9280: 05 A9 93 4C 66 96 A2 03 B5 9A 95 D8 CA 10 F9 86 #9290: DE AØ Ø2 B1 D8 C8 C5 D2 B1 D8 C8 E5 D3 BØ 47 20 #92A0: E5 96 D0 ED A9 A6 4C 66 96 A2 00 86 DE A0 02 B1 #92B0: D8 81 DA C8 20 F5 96 C0 03 F0 F4 A5 D6 81 DA 20 #92C0: F5 96 A5 D7 81 DA 20 F5 96 20 E5 96 F0 2E 18 A5 #92D0: D6 65 D4 85 D6 A5 D7 65 D5 85 D7 B0 04 C9 FA 90 #92E0: 05 A9 99 4C 66 96 A5 D0 A0 02 D1 D8 C8 A5 D1 F1 #92F0: D8 B0 B6 A5 DE 10 05 A9 A6 4C 66 96 A2 00 A9 FF #9300: 81 DA 20 F5 96 A9 FF 81 DA 20 F5 96 A5 9A 85 D8 #9310: A5 9B 85 D9 A0 02 84 DE B1 D8 85 D0 C8 B1 D8 85 #9320: D1 20 4A 97 B0 0E A0 00 84 DE 20 4A 97 90 05 A9 #9330: BA 4C 66 96 20 E5 96 D0 DB A5 DA 85 D8 A5 DB 85 #9340: D9 A5 9A 8D 91 96 A5 9B 8D 92 96 A5 9C 85 D6 A5 #9350: 9D 85 D7 D0 0C C0 4F 90 05 A9 C6 4C 66 96 20 88 #9360: 96 20 88 96 A8 D0 03 4C 09 94 20 88 96 20 88 96 #9370: A0 04 20 88 96 C8 F0 E1 AA F0 DA 10 F5 A2 06 DD #9380: F8 97 F0 05 CA 10 F8 30 E9 20 88 96 C9 20 F0 F9 #9390: 20 90 96 B0 E0 20 9F 96 A5 DA 85 D0 A5 DB 85 D1 #93A0: 20 0E 97 A2 00 86 DC 86 DD 86 DE A2 0F 06 D0 26 #9380: D1 F8 A5 DC 65 DC 85 DC A5 DD 65 DD 85 DD A5 DE #93C0: 65 DE 85 DE D8 CA 10 E5 E8 86 D0 A2 02 A9 01 85 #93D0: D1 B5 DC 4A 4A 4A 4A 29 ØF C5 DØ FØ ØB C6 DØ C8 #93E0: DØ Ø3 4C 59 93 B5 DC C6 D1 FØ EC CA 10 DF A5 DØ #93F0: D0 03 C8 F0 ED 20 90 96 C9 CD F0 07 C9 2C F0 03 #9400: 4C 75 93 C8 F0 E0 4C 89 93 A2 FF 18 B5 A7 95 D1 #9410: F5 D9 95 D5 E8 FØ F5 A5 D8 85 D2 A5 D9 85 D3 AØ #9420: 00 C6 D1 C6 D3 88 B1 D2 91 D0 98 18 65 D2 45 9A #9430: DØ 08 A5 D3 69 00 45 98 FØ 05 98 DØ E8 FØ E2 38 #9440: A5 9C 65 D4 85 D6 A5 9D 65 D5 85 D7 A5 9A 85 D8 #7450: 65 D4 8D 91 96 A5 9B 85 D9 65 D5 8D 92 96 90 14 #9460: 18 98 A0 00 65 D8 91 D8 C8 AA A5 D9 69 00 91 D8 #9470: 86 D8 85 D9 20 88 96 20 88 96 A8 D0 03 4C 31 95 #9480: 20 88 96 85 DØ 20 88 96 85 D1 20 ØE 97 AØ Ø2 A5 #9490: DØ 91 D8 C8 A5 D1 91 D8 C8 20 88 96 91 D8 C8 AA #94A0: F0 BE 10 F5 A2 06 DD F8 97 F0 05 CA 10 F8 30 E9 #94B0: 20 88 96 C9 20 F0 F9 20 90 96 B0 E0 20 9F 96 A5 #94C0: DA 85 DØ A5 DB 85 D1 20 0E 97 A2 00 86 DC 86 DD #94DØ: 86 DE A2 ØF Ø6 DØ 26 D1 F8 A5 DC 65 DC 85 DC A5 #94E0: DD 65 DD 85 DD A5 DE 65 DE 85 DE D8 CA 10 E5 E8 #94F0: 86 DØ A2 Ø2 A9 Ø1 85 D1 B5 DC 4A 4A 4A 4A 29 ØF

#9500: C5 D0 F0 07 C6 D0 09 30 91 D8 C8 B5 DC C6 D1 F0 #9510: ED CA 10 E0 A5 D0 D0 05 A9 30 91 D8 C8 20 90 96 #9520: C9 CD F0 07 C9 2C F0 03 4C 9C 94 91 D8 C8 4C B0 #9530: 94 18 A5 D8 69 02 85 9C A5 D9 69 00 85 9D A9 00 #9540: A0 07 91 D8 88 10 FB 38 A5 A6 E9 06 85 D0 A8 A5 #9550: A7 E9 00 85 D1 C4 9C E5 9D E9 01 B0 05 A9 D8 4C #9560: 66 96 A0 05 A9 00 91 D0 88 10 FB A5 9A 85 D2 85 #9570: D8 A5 98 85 D3 85 D9 A0 00 84 D4 84 D5 C8 20 F1 #9580: 96 DØ Ø3 4C 5F 96 AØ Ø2 A5 D4 D1 D8 C8 A5 D5 F1 #7570: D8 BØ 11 B1 D8 85 D5 88 B1 D8 85 D4 A5 D8 85 D2 #95A0: A5 D9 85 D3 20 E5 96 D0 DD A0 00 38 B1 D2 85 D8 #95BØ: E5 D2 85 D6 C8 B1 D2 85 D9 E5 D3 85 D7 A5 DØ 85 #95C0: DA E5 D6 85 D0 A5 D1 85 DB E5 D7 85 D1 A4 D6 88 #95DØ: B1 D2 91 DØ 88 DØ F9 B1 D2 91 DØ A5 D8 85 DA A5 #95E0: D9 85 DB 20 E5 96 A0 00 B1 DA 91 D2 C8 AA D0 F8 #95F0: C0 05 90 F4 A5 D8 05 D9 F0 16 18 98 A0 00 65 D2 #9600: 91 D2 AA C8 A9 00 65 D3 91 D2 86 D2 85 D3 D0 CB #9610: A0 01 B1 9A F0 03 4C 6B 95 A5 9A 85 D8 A5 9B 85 #9620: D9 38 A5 D0 E9 01 8D 91 96 A5 D1 E9 00 8D 92 96 #9630: AØ ØØ 2Ø 88 96 91 DB C8 AA DØ F7 CØ Ø5 9Ø F3 84 #9640: DE AØ Ø1 20 F1 96 FØ 17 18 AØ ØØ A5 DE 65 DB 91 #9650: D8 AA C8 A9 00 65 D9 91 D8 86 D8 85 D9 D0 D1 A9 #9660: ED DØ Ø3 EA A9 77 AØ 97 20 BØ CC A9 F3 85 EE A2 #9670: ØE A9 ØØ 95 DØ 95 35 CA 1Ø F9 85 EA A9 35 85 E9 #9680: 20 9F FA 4C 0F C7 60 00 EE 91 96 D0 03 EE 92 96 #9690: AD FE 90 C9 3A B0 06 38 E9 30 38 E9 D0 60 00 A2 #76AØ: ØØ 86 DA 86 DB AA Ø6 DA 26 DB BØ 31 A5 DB 85 DE #96B0: A5 DA ØA 26 DE BØ 26 ØA 26 DE BØ 21 65 DA 85 DA #96C0: A5 DE 65 DB 85 DB BØ 15 8A 29 ØF 65 DA 85 DA A5 #96D0: DB 69 00 85 DB 80 06 20 88 96 90 C9 60 68 68 A9 #96E0: 8C 4C 66 96 00 A0 00 B1 D8 C8 AA B1 D8 86 D8 85 #96F0: D9 B1 D8 60 00 E6 DA D0 02 E6 DB A5 DA C5 A6 A5 #9700: DB E5 A7 90 07 68 68 A9 D8 4C 66 96 60 00 84 DE #9710: A5 D6 85 DA A5 D7 85 DB A0 00 B1 DA 45 D0 D0 07 #9720: C8 B1 DA 45 D1 FØ 15 18 A5 DA 69 Ø4 85 DA 90 Ø2 #9730: E6 DB A0 01 B1 DA C9 FF D0 DE F0 0A C8 B1 DA 85 #9740: DØ C8 B1 DA 85 D1 A4 DE 60 90 A5 9C 85 D6 A5 9D #9750: 85 D7 A4 DE 18 B1 D6 45 DØ DØ Ø7 C8 B1 D6 45 D1 #9760: F0 12 A5 D6 69 04 85 D6 90 02 E6 D7 A0 01 B1 D6 #9770: C9 FF DØ DE 60 00 00 4D 65 6D 6F 69 72 65 20 76 #9780: 69 64 65 00 53 79 6E 74 61 78 65 00 3E 36 33 39 #9790: 39 39 00 50 61 73 30 30 00 50 61 73 20 65 78 63 #97A0: 65 73 73 69 66 00 4C 69 67 6E 65 73 20 69 6E 65 #9780: 78 69 73 74 61 6E 74 65 73 00 4D 65 6D 65 20 6E #97CØ: 75 6D 65 72 6F ØØ 4C 69 67 6E 65 2Ø 74 72 6F 7Ø #97DØ: 20 6C 6F 6E 67 75 65 00 4D 65 6D 6F 69 72 65 20 #97E0: 69 6E 73 75 66 66 69 73 61 6E 74 65 00 46 61 69 #97F0: 74 21 00 00 46 44 50 41 97 98 9B BC C8 C9 00 00

QUATRIEME PARTIE ATLAS DE LA ROM

Cette dernière partie est un répertoire, aussi complet que possible, des adresses significatives du système ORIC.

Je crois utile de faire quelques remarques.

Toutes les adresses sont en hexadécimal.
 Vous devez être convaincus maintenant de la nécessité d'adopter cette notation.

Puisqu'elle est acceptée par l'ORIC, il était inopportun d'alourdir le texte avec l'équivalent décimal.

- Parmi les adresses en mémoire morte, sont cités tous les points d'entrée des routines Basic, commandes, fonctions, opérateurs. Ils sont repérés par une flèche à gauche <- , placée après l'adresse, devant le mot-clé.

Exemple: C971 C93F <- STOP

Les routines utilisables sont indiquées par une flèche à droite ->, placée devant l'adresse.

Exemple: -> C962 C930 ISCNTC

- Il était indispensable de donner un nom abrégé à toutes ces routines.

Notre langue se prête mal à ces d'abréviations.

De plus, qu'on le veuille ou non, l'anglais est la langue de
l'informatique, il faut en prendre son parti si on veut s'y retrouver.
L'Apple 2 étant un standard de fait pour les micro-systèmes à base de
6502, j'ai emprunté sans vergogne la plupart des appellations à
l'Applesoft.

J'ai perdu tout scrupule en m'apercevant que les revues britanniques consacrées à l'ORIC les employaient. La décision était bonne.

- Si vous possédez une imprimante, il sera utile de tirer des listings de désassemblage que vous documenterez. Pour cela, à défaut d'un moniteur spécialisé, vous pouvez vous servir du programme utilitaire. (voir page 97).
- En approfondissant, vous constaterez que cet atlas de la ROM comporte encore de nombreuses lacunes et sans doute des inexactitudes. Ce sera un motif de satisfaction que d'y porter remède. Il faut des années pour connaître à fond un système. Si ce livre vous met sur la bonne voie, son objectif sera atteint.
- Les erreurs de frappe sont inévitables dans ce genre d'ouvrage. Le mode de reproduction a été choisi pour les limiter au maximum. Vous en trouverz probablement encore. Vous nous rendrez service en les signalant à l'éditeur.

CHAPITRE 16

POINTEURS, VARIABLES SYSTEME ROM - 1° PARTIE

EMPLACEMENTS EN MEMOIRE VIVE UTILISES PAR LE SYSTEME

	adresse	ATHOS	ORIC-1	utilisation
_	page zéro			
	bage rele			
	ØC,ØD	Α	0	HGR, son, transfert carac, divers
	ØE,ØF	A	ō	и и п и и
	10,11	A	ō	a u u u
	12,13	Ä	Ö	CURBAS, adresse de base de la ligne.
	14,15	A	0	son, bruits clavier ou autres sons.
	17	A	0	drapeau (INPUT, INLIN) détecte CTRL-C
	18,17	Ä	0	
	7	A	0	pointeur table mots-clés, PARSE, LIST
	1A,1B,1C			JMP STROUT
	1D,1E	A	0	pointeur pour DOKE
	1F,20	A	0	temp. pour calcul adresse ligne.
	21,22,23	A	0	JMP USR.
	24	A	0	CHARAC, (STRLT2, DATAN, OR, AND etc)
	25	Α	0	ENDCHR, " " " PARSE)
	26	Α	0	temp. CMDLP, PARSE etc.
	27	A	0	DIMFLG, drapeau tableaux, PTRGET etc
	28	A	0	VALTYP, ØØ=nombre, FF=chaine (FRMEVL)
	29	Α	0	INTFLG, 00=var.réelle, 80=var.entière
	2A	A	O	DATAFLG, utilisé par PARSE.
	2B	A	Ω	SUBFLG, 00 indice permis, 80 non.
	2C	Α	0	INPUTFLG, 40 pour GET, 98 pour READ.
	2D	Α	O	CPRMASK, utilisé en FRMEVL.
	2E	Α	O	drapeau inhibition écran (CTRL-0)
	30,31,32	Α	0	utilisés pour calcul tab.Horiz. (PRINT)
	31		a	largeur ligne imprimante (norm.80 col)
	33,34	Α	O	LINNUM, pointeur usage général.
	35-83	Α	O	BUF, buffer d'entrée
	5F, 60 -> 2A9.		0	adr. début enreg. CLOAD, CSAVE (ORIC-1)
	61,62 - 288.		ō	" fin " "
	43 - 2 gg .		ō	0 = non AUTO, 1 = AUTO (ou <>0)
	64 - 28E-		ō	0 = Basic, 1 = code machine
	67 - 245		Ō	0 = fast, 1 = slow
	85	Α	ā	TEMPTT, manip. chaînes, STKINI, GARBAGE.
	86	Ä	Ö	LASTPT, " "
	91,92	Ä	O	INDEX, pointeur usage général.
	93,94	A	0	pointeur temporaire, CONINT, VAL.
	95-99	A	0	RESULT, résultat dernière mult. ou div.
	9A,9B	A	0	TXTTAB, début programme Basic.
	9C,9D	A	0	VARTAB, début zone des variables.
	9E,9F	A	۵	ARYTAB, début zone des tableaux.
	AØ,A1	Α	0	STREND, fin zone de stockage variables.
	A2,A3	Α	0	FRETOP, fin de l'espace libre.
	A4,A5	A	0	FRESPC, temp pour chaînes.
	A6,A7	Α	0	MEMSIZ, haut mémoire vive utilisable.
	A8,A9	A	0	CURLIN, n°ligne en cours.

	adresse	ATHOS	ORIC-1	utilisation
	AA,AB	A	0	OLDLIN, n°ligne, après interrupt.
	AC, AD	A	Ö	OLDTXT, adr.dern.oct.instr.précéd.
	AE, AF	A	Ö	DATLIN, n°ligne DATA en cours lecture.
	BØ, B1	A	Ö	DATPTR, adresse DATA à lire par READ.
			0	
	B2,B3	A		INPTR, adr. source INPUT.
	B4,B5	Α	0	VARNAM, nom dern.variable utilisée.
	B6,B7	A	0	VARPNT, adr.dern.variable, PTRGET.
	BB, B9	A	0	FORPNT, usage général, utilisé par COPY.
	BA,BB,BC	A	0	utilisés par FRMEVL.
	BD-C1	A	0	TEMP3, reg.flot.5 oct, voir ch.13 et 19.
	C2	Α	0	DSCLEN, utilisé dans GARBAGE.
	C3,C4,C5	A	0	JMPADRS, saut ADR modifiable, fonctions.
	C6-CA	Α	Ω	TEMP1, reg.flot.5 oct, voir ch.13 et 19.
	C7,C8	A	0	HIGHDS, pointeur utilisé par BLTU.
	C9,CA	A	0	HIGHTR, " "
	CB-CF	Α	O	TEMP2, reg.flot.5 oct, voir ch.13 et 19.
	CE,CF	Α	O	LOWTR, pointeur nombreux usages.
	DØ-D5	A	0	FAC, Acc.flot.6 oct, voir ch.13 et 19.
	DØ, D1, D2	Α	a	DSCTMP, descripteur temporaire de chaines
	D3,D4	A	O	VPNT, temp.adr.variable, VARL (FRMEVL).
	D6,D7	Α	O	routines FP, voir chap 19.
	D8-DD	A	0	ARG, Arg.flot.6 oct, voir ch.13 et 19.
	DE, DF	A	0	STRNG1, ptr (MOVINS) voir aussi chap 19.
	EØ,E1	Α-	0	STRNG2, ptr (STRLT2) " " "
	E2-F2	A	a	CHARGET, CHARGOT, voir 3° partie, ch.11.
	E9,EA	Α	0	TXTPTR, pointeur de texte.
	FA-FE	A	O	RNDSEED, utilisés par RND.
	FF	A	0	FBUFF, symbole # hexa ou signe Nb. (STR\$)
_	page un			
		100		*
	100-1FF	A	0	pile du microprocesseur.
	FF-110	Α	a	FBUFF, buffer FAC, utilisé par FOUT.
_	page deux			
	200-205	A	O	affichage, HGR, transfert carac.(ORIC-1)
	208,209	Α	G	codage des touches, scrutation clavier.
	20A	A	0	SHIFTG = A4, SHIFTD = A7, CTRL = A2.
	20C	Α	Ð	CAPLCK, FF = majuscules, 7F = minuscules.
	2ØD	A	0	scrutation clavier.
	20E	A	O	répétition automatique.
	210,211	A	0	scrutation clavier.
	212-218	A	0	routines graphiques.
	213	A	0	PAT, pattern, (CIRCLE, DRAW)
	219	Α	0	CURX, coordonnée X en HGR
	21A	Α	0	CURY, coordonnée Y en HGR.
	21B-21E	Α	a	routines graphiques, CIRCLE
	21F	Α	O	GRA, drapeau, 1=HIRES, Ø=TEXT/LORES.
	228,229,22A	The same of the sa	Ö	JMP #ECØ3 (IRQ)
	22B,22C,22D		0	JMP #F430 (RESET)
	230		0	RTI (#40)
	238,239,23A	Α		JMP #F77C (VDU)
	23B,23C,23D	A		JMP #EB78 (RDKEY)
	23E,23F,240	A		JMP #F5C1 (PRTCHR)
	5. The state of th			

adresse	ATHOS	ORIC-1	utilisation
241,242,243	A		JMP #F865 (STOUT)
244,245,246			JMP #EE22 (IRQ)
247,248,249			JMP #F8B2 (NMI) RESET
24A	A		RTI (#4Ø)
24D	Α		TSPEED, CLOAD, CSAVE (vitesse, Ø=Fast)
24E	A		tempo entre 2 rep.auto, norm.#20(*30ms)
24F	A		" sort.carac.en rep.auto (norm.#04)
252	Α		IFFLG, drapeau IF, pour ELSE éventuel.
256	A		PWIDTH, largeur ligne Impr (norm 80 col)
25A	A		CLOAD, CSAVE, STORE, RECALL (Join).
25B	A		" " " (Verify)
25C,25D	Α		H II
268	A	O	CURROW, ligne écran en cours.
269	A	0	CURCOL, colonne écran en cours.
26A	A	0	MODEØ, si bit concerné à un:
			n°0, curseur visible
			n°1, vidéo active
			n°2, non utilisé
			n°3, bruit clavier OFF
			n°4, fonction ESC
			nº5, 40 col _ nº6 double hauteur
			n°6 et 7, inutilisés.
26B	A	O	BGND, papier, couleur+16
26C	A	0	FGND, encre, couleur
26D,26E		0	adresse haut de l'écran (norm.#BB80)
272,273	Α	O	TIMER1, clavier
274,275	Α	O	TIMER2, curseur
276,277	Α	O	TIMER3, utilisé par WAIT.
278,279	Α		VDUL2, adresse 2me ligne écran.
27A,27B	A		VDUL1, adresse 1re ligne écran.
27C,27D	A		capacité écran, norm. 1040 carac.
27E	Α		nombre de lignes écran, norm.27
2A9,2AA	A		routines cassette, début enregistrement
2AB,2AC	A		" , fin "
2AD-2B1	Α		" ,différents drapeaux.
200	Α	0	#02 = mode TEXT, #03 = HIRES.
201,202	Α	0	HIMEM, GRAB, RELEASE (recueille MEMSIZ)
2DF	Α	0	ICHAR, mém.clavier, carac.ASCII négatif
2EØ-2E7	Α	0	params. routines graphiques et sonores.
2F1	Α	0	PRTFLG, sortie, <#80=vidéo, >#7F=imprim.
2F2	Α	0	utilisé par EDIT, empeche sortie LIST.
2F4	Α	0	TRACE, <#80 = TROFF, >#7F = TRON
2F5,2F6	A	0	adresse routine commande !
2FB,2FC,2FD	Α	O	saut adresse routine fonction &

- page trois

adresses des registres du 6522. Entrées/Sorties.

- #9800-BFE0

Ecran HIRES, écran texte, jeux de caractères. Répartition différente suivant le mode utilisé. En mode texte, la commande GRAB permet de récupérer la zone comprise entre #9800 et #B3FF.

ROM - PREMIERE PARTIE - TABLES

ATMOS #C000-C3C5 ORIC-1 #C000-C3C9

Après un JMP INIT et un JMP READY2, on trouve dans cette première partie de la ROM, cinq tables différentes.

1. table d'entrée des commandes #C006-C089

Cette table est utilisée par NEWSTT pour le branchement à la routine d'exécution appropriée.

Elle contient 66 pointeurs d'adresse sur deux octets, (octet faible en premier), rangés dans l'ordre croissant des codes, de END (#80) à NEW (#C1).

Chaque adresse correspond au point d'entrée (moins un) dans la routine de la commande concernée.

Les adresses reproduites ci-dessous sont les adresses d'entrée effectives.

Notamment, celles des commandes graphiques et sonores qui ont un point d'entrée commun dans la table.

adresse en hexadécimal

code hexa	commande	ATMOS	ORIC-1
80	END	C973	C941
81	EDIT	C692	C6A5
82	STORE	E987	1
83	RECALL	E9D1	1
84	TRON	CD16	CCBC
85	TROFF	CD19	CC8F
86	POP	CA12	C9EØ
87	PLOT	DA51	D9C6
88	PULL	DAA1	DA16
87	LORES	D9DE	D937
BA	DOKE '	D967	DBAC
88	REPEAT	DA85	D9FA
8C	UNTIL	DAAI	DA16
8D	FOR	C855	C841
8E	LLIST	C7FD	C824
8F	LPRINT	C807	C832
90	NEXT	CE98	CEØC
91	DATA	CA3C	CAØA
92	INPUT	CD55	CCC9
93	DIM	D17E	DØF2
94	CLS	CCCE	CCØA
95	READ	CD89	CCFD
96	LET	CB1C	CAD2
97	GOTO	C9E5	C9B3
98	RUN	C9BD	C98B
99	IF	CA70	CA3E
9A	RESTORE	C952	C91F
9B	GOSUB	C9C8	C996
9C	RETURN	CA12	C9EØ
9D	REM	CA99	CA61
9E	HIMEM	EBCE	E958

code hexa	commande	ATMOS	ORIC-1
9F	GRAB	EBE7	E974
AØ	RELEASE	ECØC	E994
A1	TEXT	EC21	E9A9
A2	HIRES	EC33	E9BB
A3	SHOOT	FAB5	FA9B
A4	EXPLODE	FACB	FAB1
A5	ZAP	FAE1	FAC7
A6	PING	FA9F	FA85
A7	SOUND	FB4Ø	FB26
A8	MUSIC	FC1B	FBFE
A9	PLAY	FBDØ	FBB6
AA	CURSET	FØC8	FØ2D
AB	CURMOV	FØFD	FØ64
AC	DRAW	F110	FØ79
AD	CIRCLE	F37F	F2E5
AE	PATTERN	F11D	FØ93
AF	FILL	F268	F1E5
BØ	CHAR	F12D	FØA5
B1	PAPER	F204	F17F
B2	INK	F210	F188
B3	STOP	C971	C93F
B4	ON	CAC2	CA78
B5	WAIT	D958	D89D
B6	CLDAD	E858	E7AA
B7	CSAVE	E909	E7DB
88	DEF	D4BA	D4Ø1
B9	POKE	D94F	D894
BA	PRINT	CBAB	CB61
BB	CONT	C9AØ	C96E
BC	LIST	C748	C773
BD	CLEAR	C7ØD	C738
BE	GET	CD46	CCBA
BF	CALL	E946	EBØD
CØ	į.	CD13	CC89
C1	NEW	CAEE	C719

2. table d'entrée des fonctions #C08A-C0CB

Cette table contient 33 pointeurs d'adresse sur deux octets, (octet faible en premier) correspondant aux points d'entrée des fonctions, de SGN (#D6) à MID\$ (#F6).

Elle est utilisée par la routine d'évaluation d'expressions (FRMEVL) pour le branchement approprié.

code hexa	fonction	ATMOS	ORIC-1
D6	SGN	DF21	DF12
D7	INT	DFBD	DFA5
DB	ABS	DF49	DF31
D9	USR	0021	0021
DA	FRE	D47E	D3D6
DB	POS	D4A6	D3FA
DC	HEX \$	D9B5	D917
DD	&	02FB	02FB

code hexa	fonction	ATMOS	ORIC-1
DE	SQR	E22E	E22A
DF	RND	E34F	E34B
EØ	LN	DCAF	DC79
E1	EXP	E2AA	E2A6
E2	COS	E38B	E387
E3	SIN	E392	E38E
E4	TAN	E3DB	E3D7
E5	ATN	E43F	E43B
E6	PEEK	D938	D87D
E7	DEEK	D98 3	DSC8
E8	LOG	DDD4	DDDØ
E9	LEN	D8A6	D7EB
EA	STR\$	D593	D4D8
EB	VAL	D8D7	D81C
EC	ASC	D8B5	D7FA
ED	CHR\$	D816	D75B
EE	PI	DE77	DBEE
EF .	TRUE	DFØF	DFØØ
FØ	FALSE	DFØB	DEFC
F1	KEY\$	DADA	DA4F
F2	SCRN	DA3F	D9B4
F3	POINT	EC45	E9CD
F4	LEFT\$	D82A	D76F
F5	RIGHT\$	D856	D79B
F6	MID\$	D861	D7A6

3. table d'entrée des opérateurs #COCC-COE9

Cette table se compose de 10 groupes de 3 octets, deux octets pour l'adresse d'entrée (moins un) de l'opérateur arithmétique ou logique précédés d'un octet formant préfixe.

Elle est utilisée au cours de l'évaluation des expressions, le préfixe servant à déterminer la priorité attribuée à l'opérateur concerné.

Comme pour les commandes, les adresses indiquées sont les adresses d'entrée effectives (adresse dans la table + 1).

adresse en hexadécimal

code hexa	opérateur	préfixe	ATMOS	ORIC-1
CC	+	79	DB25	DA9A
CD	-	79	DBØE	DAB3
CE	*	7B	DCFØ	DCBA
CF	/	7B	DDE7	DDE3
DØ	^	7F	E238	E234
D1	AND	50	DØE6	DØ5A
D2	QR	46	DØE3	DØ57
D3	>	7D	E271	E26D
D4	=	5A	DØ3C	CFBØ
D5	<	64	D113	DØ87

4. table des mots-clés #C@EA-C2A7 (ORIC-1 #C@EA-C2AB)

Cette table contient tous les mots-clés du Basic de END à MID\$, y compris les modificateurs et les mots-clés associés aux commandes tels que TAB(, THEN, STEP etc., qui n'ont pas leur propre adresse d'entrée. Celle de l'ORIC-1 comporte un mot supplémentaire, GO, inopérant. La dernière lettre du mot-clé est inscrite en ASCII négatif (code ASCII+#80).

La table se termine par la valeur 00. Elle est utilisée pour le codage (PARSE) ou le décodage (LIST) des mots-clés.

5. messages d'erreur et divers #C2A8-C3C6 (ORIC-1 #C2AC-C3C9)

Cette dernière table contient la plupart des messages d'erreur et d'autres comme Ready et BREAK. Ils sont délimités, soit par leur dernière lettre en ASCII négatif, soit par la valeur 00 qui les suit.



ROM - 2° PARTIE

ATMOS #C3C6-CFØ2 ORIC-1 #C3CA-CE76

ENTREE DU TEXTE BABIC, ANALYSE ET CODAGE STOCKAGE DU PROGRAMME, EXECUTION, BRANCHEMENTS, BOUCLES, ETC

ATHOS	ORIC-1	description	
C3C6	C3CA	GTFRPNT, recherche dans la pile si variable FOR existe.	
-> C3F4	C3F8	BLTU, fait de la place en mémoire par transfert de bloc vers le haut. Utilisée dans CMDLP pour insertion ligne de programme. A l'entrée: LOWTR (#CE,CF) = plus basse adresse à déplacer HIGHTR (#C8,C9) = plus haute adr. à déplacer + 1 HIGHDS (#C7,C8) et A(-),Y(+) = destination + 1 de la plus haute adresse à déplacer.	
C437	C43B	CHKMEM, utilisée par FOR, GOSUB, FRMEVL vérifie qu'il y a de la place en pile	
-> C444	C448	REASON, vérifie espace disponible, compare adresse en A(-),Y(+) à FRETOP, peut provoquer GARBAGE collection, message OUT OF MEMORY si pas de place.	
-> C471	C475	READY1, confirme mode texte (bit 0 de #200=0) puis READY2.	
C47C	C483	OUT OF MEMORY ERROR	
C47E	C485	ERRMSG, affiche message d'erreur, indexé par X dans table en #C2A8 ATMOS, #C2AC ORIC-1, init pile affiche IN suivi n° ligne si mode programme.	
-> C4A8	C485	READY2, TROFF (#2F4=0), mise à zéro différents drapeaux, affichage READY puis entrée dans CMDLP.	
C4B7	C4C7	CMDLP, boucle principale d'entrée des commandes, analyse et codage de la ligne entrée, exécution si mode direct ou stockage si mode différé. Utilise INLIN, PARSE, LINGET, FINDLIN, STXTPT, LINKSET. N'est pas utilisable, pas de retour. (voir 3° partie, chapitre 11)	

ATHOS	ORIC-1	description	
-> C55F	C56F	LINKSET, positionne octets de chainage à partir du début de programme en TXTTAB (#9A,9B). Utilise pointeur INDEX (#91,92).	
-> C592	C5A2	INLIN, inscrit une ligne de texte entrée au clavier dans le buffer d'entrée (#35), écho à l'écran, utilise INCHR. (voir 3° partie, chapitre 12)	
-> C5E8	C5F8	INCHR, recueille caractère entré au clavier dans l'accumulateur, appelle RDKEY (voir 3° partie, chapitre 12)	
-> C5FA	C6ØA	PARSE, analyse du buffer d'entrée, codage des mots-clés. (voir 3° partie, chapitre 12)	
C692	C6A5 <-	EDIT, (voir 3° partie, chapitre 12)	
-> C4B3	C6DE	FNDLIN, parcourt le programme pour trouver la ligne dont le numéro est en LINNUM (#33,34). Utilise le pointeur LOWTR (#CE,CF). A la sortie: Si C = 1, LOWTR pointe l'adresse de la ligne cherchée (octets de chainage) Si C = 0, ligne non trouvée, LOWTR pointe l'adresse de la ligne suivante.	
CAEE	C719 <-	NEW	
	0,1,		
-> C6FØ	C71B	SCRATCH, 'NEW', efface programme, variables et pile (modification des pointeurs).	
		SCRATCH, 'NEW', efface programme, variables	
-> C6FØ	C71B	SCRATCH, 'NEW', efface programme, variables et pile (modification des pointeurs). SETPTRS, appelle STXTPT puis -> CLEAR. En Basic, un CALL de cette adresse est équivalent à 'RUN'	
-> C6FØ -> C7Ø8	C71B C733	SCRATCH, 'NEW', efface programme, variables et pile (modification des pointeurs). SETPTRS, appelle STXTPT puis -> CLEAR. En Basic, un CALL de cette adresse est équivalent à 'RUN'	
-> C6FØ -> C7Ø8	C71B C733 C738 <-	SCRATCH, 'NEW', efface programme, variables et pile (modification des pointeurs). SETPTRS, appelle STXTPT puis -> CLEAR. En Basic, un CALL de cette adresse est équivalent à 'RUN' CLEAR CLEARC, 'CLEAR', efface variables et pile	
-> C6FØ -> C7Ø8 -> C7ØD -> C7ØF	C71B C733 C738 <- C73A	SCRATCH, 'NEW', efface programme, variables et pile (modification des pointeurs). SETPTRS, appelle STXTPT puis -> CLEAR. En Basic, un CALL de cette adresse est équivalent à 'RUN' CLEAR CLEARC, 'CLEAR', efface variables et pile appelle RESTORE.	
-> C6FØ -> C7Ø8 -> C7ØD -> C7ØF -> C726	C71B C733 C738 <- C73A C751 C765	SCRATCH, 'NEW', efface programme, variables et pile (modification des pointeurs). SETPTRS, appelle STXTPT puis -> CLEAR. En Basic, un CALL de cette adresse est équivalent à 'RUN' CLEAR CLEARC, 'CLEAR', efface variables et pile appelle RESTORE. STKINI, réinitialise la pile. STXTPT place TXTPTR (#E9,EA) au début du programme, (TXTTAB)-1, normalement #500. En Basic, un CALL de cette adresse est équivalent	
-> C6FØ -> C7Ø8 -> C7ØD -> C7ØF -> C726 -> C73A	C71B C733 C738 <- C73A C751 C765	SCRATCH, 'NEW', efface programme, variables et pile (modification des pointeurs). SETPTRS, appelle STXTPT puis -> CLEAR. En Basic, un CALL de cette adresse est équivalent à 'RUN' CLEAR CLEARC, 'CLEAR', efface variables et pile appelle RESTORE. STKINI, réinitialise la pile. STXTPT place TXTPTR (#E9,EA) au début du programme, (TXTTAB)-1, normalement #500. En Basic, un CALL de cette adresse est équivalent à 'RUN' sans 'CLEAR.	

- ATMOS ORIC-1 description
 - C809 C832 <- LPRINT, bit 7 #2F1 mis à un, appelle PRINT Remarque: en Basic, POKE#2F1,128 a pour effet de provoquer la sortie vers l'imprimante par tous les PRINT subséquents.
- C816 et ATMOS seulement, sous-routines positionnement C82F des drapeaux pour LLIST et LPRINT.
- C855 C841 <- FOR, installation boucle, empilage éléments appel nombreuses routines. La valeur STEP = 1 par défaut (virgule flottante) est recueillie en #DC81 ATMOS, #DC48 ORIC-1.
- -> CBC1 CBAD NEWSTT, exécute une nouvelle instruction, pas de retour. (voir 3° partie, chapitre 11).
- -> C952 C91F <- RESTORE, met le pointeur de DATA, DATPTR (#80,81) en début de programme.
- -> C962 C930 ISCNTC, teste le clavier pour CTRL-C. Si oui, C=1, arrêt programme après affichage BREAK IN (n° ligne).
 - C971 C93F <- STOP, arrêt programme avec affichage BREAK. (C=1) (TXTPTR, #E9, EA) et (CURLIN, #A8, A9) sont inscrits dans OLDTXT, #AC, AD et OLDLIN, #AA, AB -> PREND.
 - C973 C941 (- END, arrêt normal. (C=0)
 (TXTPTR) et (CURLIN) sont inscrits dans
 OLDTXT et OLDLIN -> PREND.
 - C98A C958 PREND, fin de programme, affichage BREAK et/ou sortie par READY2.
 - C9AØ C96E <- CONT
- -> C9A2 C970 CONT, place (OLDTXT) et (OLDLIN) dans TXTPTR
 et CURLIN pour reprise exécution du programme après
 arrêt par END ou STOP.
 ?CAN'T CONTINUE ERROR si OLDTXT non utilisable,
 (#AD=00).
 Peut être utilisée en langage machine pour reprise
 exécution Basic (ex: pages 23,102)
 - C9BD C9BB <- RUN, si numéro de ligne spécifié CLEARC, STKINI puis GOTO. Sinon JMP SETPTRS (ci-dessous)
 - C9BF C9BD en Basic, un CALL de cette adresse est équivalent à RUN.
 - C9C8 C996 <- GOSUB, empilage TXTPTR, CURLIN et code GOSUB (#98) puis GOTO et NEWSTT.

William alize I describer.	ATMOS	ORIC-1	description
----------------------------	-------	--------	-------------

- -> C9E5 C9B3 <- GOTO, évalue l'expression numérique à TXTPTR et modifie ce dernier.

 Fermet un branchement à un n°de ligne variable (GOTO calculé). Utilise FRMNUM, GETADR, FNDLIN.

 ?UNDF'D STATEMENT ERROR si le résultat de l'évaluation ne correspond pas à un numéro existant Remarque: seuls les octets forts de CURLIN et LINNUM sont comparés.

 Si déplacement > 256 octets en avant, la recherche commence à partir de la ligne suivante.

 Sinon, recherche depuis le début du programme.
- -> C9E8 C9B6 'GOTO', le numéro de ligne étant dans LINNUM, un JMP à cette adresse provoque le branchement.
 - CA12 C9EØ <- POP RETURN, recherche en pile, désempilage en conséquence.
 - CA20 C9EE ?RETURN WITHOUT GOSUB ERROR
 - CA23 C9F1 ?UNDEF'D STATEMENT ERROR
- -> CA3C CAØA <- DATA, déplace TXTPTR à la fin de l'instruction. Recherche ':' (#3A) ou fin de ligne (00).
- -> CA3F CA0D ADDON, ajoute contenu de Y à TXTPTR.
- -> CA4E CA1C DATAN, calcule le déplacement dans Y, depuis TXTPTR jusqu'au prochain : ou 00.
- -> CA51 CA1F REMN, calcule le déplacement dans Y depuis TXTPTR jusqu'au prochain 00 de fin de ligne
 - CA70 CA3E <- IF, si assertion est vraie GOTO ou GOCMD (NEWSTT) si elle est fausse, recherche ELSE (effectuée par NEWSTT sur ATMOS, drapeau IFFLAG #252 levé)
 - CA99 CA61 <- REM, appelle REMN puis -> ADDON, ignore le reste de la ligne.
 - CA9E CA66 ELSE?, si code du mot-clé (#C8) est trouvé, il est sauté (JMP REMN), retour à IF pour exécution.
 Sinon, RTS.
 - CAB1 ADDON2, recherche la fin d'instruction après ELSE puis positionne TXTPTR (-> ADDON). utilisée par NEWSTT (ATMOS).
 - CABF CA75 JMP ?SYNTAX ERROR
 - CAC2 CA78 <- ON (goto-gosub), recueil variable par GETBYT dans FACLO (#D4). Test pour code GOTO ou GOSUB puis recherche d'un n° de ligne correspondant à rang spécifié par variable.
 Si trouvé, JMP GOCMD (NEWSTT). Sinon, RTS.

- -> CAE2 CA98 LINGET, met dans LINNUM (#33,34) le n° de ligne pointé par TXTPTR.

 Suppose que A et les indicateurs aient été renseignés par l'appel de CHARGET qui a trouvé le premier chiffre.

 Sort normalement par CHARGET qui recueille le caractère autre qu'un espace suivant les chiffres du n° de ligne.

 Si le nombre est > 63999, sortie par SYNTAX ERR Si pas de nombre à TXTPTR, (LINNUM) = 0.
- -> CB1C CAD2 <- LET, utilise PTRGET pour obtenir l'adresse de la variable dans FORPNT (#B8,B9), vérifie la présence du signe '=', évalue l'expression et en range la valeur à l'emplacement ad-hoc en zone des variables.

 A l'entrée, TXTPTR pointe sur la première lettre du nom de la variable.
- -> CB3B CAF1 FAC->INT, convertit FAC (inférieur à 2^15) en un entier sur deux octets dans FORPNT (#B8,B9).
- -> CB8D CB43 COPY, libére un descripteur temporaire de chaîne pointé par A(-),Y(+) et le déplace à l'adresse pointée par FORPNT (#B8,B9).
 - CBAB CB61 <- PRINT, la commande PRINT provoque l'exécution d'une boucle dont quelques jalons sont indiqués ci-dessous (#CCOC, test virgule). Après CHARGOT, le caractère en A est comparé aux codes de TAB(, SPC(, ', ' et '; ' pour espacement éventuel. L'ATMOS effectue un test supplémentaire portant sur le code de « et le cas échéant un saut en #CC59 (PRINT AT) est exécuté. L'expression à afficher est évaluée. Suivant sa nature, numérique ou alphanumérique, la procédure d'affichage appropriée est appliquée. Le processus est recommencé tant qu'un code compatible avec PRINT est détecté. Sortie par CRDO (retour à la ligne et saut de ligne).
- -> CBEA CB99 EOL, fin de ligne, sortie normale de INLIN après détection RETURN. La valeur 00 est placée à la fin de la ligne inscrite dans le buffer d'entrée. Si CTRL-C est détecté, au lieu de RETURN, la valeur 00 est placée en #35 pour annuler l'entrée. (également valable pour une entrée sur INPUT). Adresse #0034 (buffer-1) dans X(-),Y(+) puis ->

ATMOS	ORIC-1	description
-> CBFØ	CB9F	CRDO, Carriage Return + Line Feed, retour à la ligne + saut de ligne. Remarque: cette portion est responsable du fonctionnement incorrect de la fonction TAB(sur ORIC-1. Les emplacements #30 et #31 servent au calcul de la tabulation horizontale associée à PRINT. A la sortie, sur ORIC-1, #30 contient la valeur #0D (13 décimal, code de RETURN) d'où l'erreur avec TAB(. Le défaut a été corrigé sur ATMOS. Le contenu de #30 est remis à zéro avant le retour.
-> CCØ2	CBA6	LNFEED, saut de ligne.
CCØC	CRAC	virgule, espacement vers champ suivant ATMOS 8 caractères, ORIC-1 5 caractères
CC2E	CBCA	TAB ou SPC, GETBYTC dépose valeur en X.
CC39	CBD5	TAB(
CC47	CBDB	SPC(
CC4B	CBDF	NXTCHR, caractère suivant.
CC59		(PRINT) AT, ATMOS seulement.
-> CCBØ	CBED	STROUT, affiche la chaine de caractères dont l'adresse est pointée par A(-) et Y(+). La chaine doit se terminer par " ou 00.
-> CCB3	CBFØ	STRPRT, affiche la chaîne de caractères dont le descripteur est pointé par FACMO,FACLO (#D3,D4).
-> CCCE	CCØA <-	CLS, vide l'écran.
-> CCD1		CURTGL, bascule curseur, ATMOS seulement.
-> CCD4	CCØD	OUTSPC, sort un espace.
-> CCD7	CC10	OUTQST, sort un point d'interrogation.
-> ccp9	CC12	OUTDO, sort le caractère en A.
	CC1B	PRTOUT, sortie imprimante du caractère en A.
	CC3A	?PRINTER ERROR.
-> CCFB	CC4D	VIDOUT, sortie écran du caractère en A.
	CC9D	message PRINTER ERROR.
CD13	CC89 <-	commande !, saut indirect, JMP (#2F5)

-> CD16 CC8C <- TRON, drapeau #2F4 = #80.

1	ATHOS	ORIC-1	description
	CD1F	CC95	INPUTERR, branchement éventuel de MAININP.
	CD31	CCA7	erreur, ATMOS Type mismatch, ORIC-1 syntax.
	CD36	CCAA	?REDO FROM START
	CD46	CCBA <-	GET, test mode, ILLEGAL DIRECT si immédiat. Simule INPUT, caractère unique (place un 00 en #36). INPUTFLG (#2C) = #40, -> MAININP
	CD55	CCC9 <-	INPUT, test pour " (chaine explicative) si oui STRTXT puis test ';'. Test mode, ILL.DIRECT si mode immédiat. Test BUF et CTL-C. On recommence si pas d'entrée. Sortie normale si CTL-C, sinon -> MAININP
>	CD80	CCF4	INPUTRQ, appelée par INPUT, affiche ? et espace puis -> INLIN.
	CD89	CCFD <-	READ, charge DATPTR dans X et Y puis MAININP. (INPUTFLG=#98)
	CD8F	CDØ3	MAININP, routine INPUT, utilisée par GET, INPUT et READ.
	CE3C	CDBØ	charge X pour message éventuel ?OUT OF DATA.
	CE6C	CDEØ	?EXTRA IGNORED.
	CE74	CDE8	messages ?EXTRA IGNORED et ?REDO FROM START.
	CE98	CEØC <-	NEXT, maintient la boucle jusqu'à valeur limite, utilise la pile, sortie vers NEWSTT.

CEAA CE1E ?NEXT WITHOUT FOR ERROR



ROM - 3° PARTIE

ATMOS #CF03-DB03 ORIC-1 #CE77-DA78

EVALUATION D'EXPRESSIONS, LOCALISATIONS, MANIPULATION DE CHAINES DE CARACTERES, FONCTIONS SUR CHAINES...

ATMOS	ORIC-1	description	
-> CFØ3	CE77	FRMNUM, évalue l'expression pointée par TXTPTR, place le résultat en FAC et s'assure qu'il s'agit bien d'un nombre. A l'entrée TXTPTR pointe sur le premier caractère de la formule.	
-> CFØ6	CE7A	CHKNUM, vérifie que le contenu de FAC correspond à une valeur numérique.	
-> CFØ8	CE7C	CHKSTR, vérifie que le contenu de FAC correspond à une chaîne de caractères.	
-> CFØ9	CE7D	CHKVAL, vérifie le résultat de la dernière opération sur le FAC suivant VALTYP (#28). A l'entrée: C=1 test pour chaîne. C=0 test pour nombre. TYPE MISMATCH ERROR si C et FAC ne concordent pas.	
CF12	CE86	?TYPE MISMATCH ERROR	
-> CF17	CE8B	FRMEVL, évalue l'expression pointée par TXTPTR et laisse le résultat en FAC. A l'entrée, TXTPTR pointe sur le premier caractère de la formule. Utilisée pour les nombres et pour les chaînes. Si la formule contient une chaîne entre guillemets, FRMEVL escamote le guillemet de début (C=1 pour STRTXT) et exécute STRLIT et STRLT2.	
Remarque:	qui man: complexe Notez le	est la sous-routine principale adaptée aux commandes ipulent des expressions. Elle est extrêmement e. Quelques points de repère sont cités ci-après. e rôle de GETVAL qui identifie le premier caractère ormule et détermine le traitement approprié.	
CF22	CE96	FEVLOOP, entrée boucle évaluation, appel de GETVAL.	
CF34	CEA8	test pour '>','=' ou '<'.	
CF5Ø	CEC4	CHKTYP, type d'expression?, numérique ou chaîne branchement approprié selon opérateur et type NOTMATH ou DOMATH.	
CF61	CED5	ARITH, opérateurs arithm,	

	ATMOS	ORIC-1	description	
	CF69	CEDD	PREFTEST, test priorité opérateur.	
	CF73	CEE7	SAVOP, sauvegarde opération en pile (JSR PSHMAD)	
	CF8Ø	CEF4	COMP, comparaison chaînes de caractères.	
	CF99	CFØD	PSHMAD, adresse opérateur récupérée dans la table et mise en pile. JSR PSHF, retour par JMP (#91) puis JMP FEVLOOP.	
	CFAC	CF20	PSHF, empilage FAC, RTS par JMP (INDEX)	
	CFD5	CF49	NOTMATH, comparaison chaînes de caractères éventuelle ou sortie.	
	CFE3	CF57	DOMATH, exécution routine arithmétique, désempilage dans ARG, puis entrée routine via RTS (adresse précédemment empilée).	
	DØØØ	CF74	GETVAL, teste premier caractère de l'expression. si nombre -> FIN, mise en FAC si lettre -> VARL, recherche variable. si symbole '.' ou '#', nombre -> FIN si '-' opérateur moins, si '+' ignoré. si '"', chaine -> STRTXT	
->	DØ25	CF99	STRTXT, fait A(-),Y(+) = (TXTPTR) + C puis exécute STRLIT.	
	DØ34	CFA8	tests GETVAL étant négatifs, test mots-clés NOT, FN et SGN. Si infructueux, il s'agit d'une fonction -> FUNCT	
	DØ3C	CFBØ <-	opérateur '=' ,EQUOP	
->	DØ59	CFCD	PARCHK, vérifie présence de parenthèses et évalue l'expression incluse, CHKOPN, FRMEVL, CHKCLS.	
->	DØ5F	CFD3	CHKCLS, vérifie ') ' à TXTPTR, utilise SYNCHR.	
->	DØ62	CFD6	CHKOPN, vérifie '(' à TXTPTR, utilise SYNCHR.	
->	DØ65	CFD9	CHKCOM, vérifie ',' à TXTPTR, utilise SYNCHR.	
->	DØ67	CFDB	SYNCHR, compare contenu accumulateur avec caractère pointé par TXTPTR. Si idem -> CHARGET, sinon SYNTAX ERR.	
	DØ7Ø	CFE4	?SYNTAX ERROR.	
	DØ7C	CFFØ	VARL, recherche variable par PTRGET. adresse rangée dans ptr. temporaire VPNT (#D3,D4). Si chaine, retour pour évaluation, si nombre, mis en FAC.	
	DØ9Ø	DØØ4	INT->FP, convertit variable entière (%) pointée par FACMD,FACLO (#D3,D4) en virgule flottante, résultat en FAC. VALTYP (#28) mis à zéro.	

A	ATMOS	ORIC-1	description
	DØAØ	DØ14	FUNCT, traitement des fonctions. Code mot-clé multiplié par 2, servira d'index en Y pour recherche adresse d'entrée routine. Si fonction chaîne, évaluation opération, empilage éléments (VPNT, ler parametre) Adresse d'entrée en Adr.Base,Y et Adr.Base+1,Y (Adr.Base = début table fonctions - (code SGN)*2) PTR routine placé en #C4,C5.(JMPADRS+1,JMPADRS+2) Exécution par JSR JMPADRS (#C3) (sans retour pour LEFT\$, RIGHT\$ ou MID\$) puis -> CHKNUM.
	DØE3	DØ57 <-	ØR, opérateur, FAC = FAC 'OR' ARG.
	DØE6	DØ5A <-	AND, opérateur, FAC = FAC 'AND' ARG.
	D113	DØ87 <-	opérateur '<', POSOP
	D17E	DØF2 <-	DIM, JSR PTRGET+5, DIMFLAG <> 0, utilise ARRAY, reboucle sur DIM-3 pour dimension suivante éventuelle.
->	D188	DØFC	PTRGET, lit un nom de variable à l'aide de CHARGET et le trouve en mémoire. A l'entrée, TXTPTR pointe sur le premier caractère du nom. A la sortie, l'adresse de la valeur de la variable se trouve en VARPNT (#B6,B7) et en A(-),Y(+). Si PTRGET ne peut pas trouver une variable simple elle en crée une. Si elle ne peut pas trouver un tableau elle en crée un, dimensionné 0-10 et dont elle fixe tous les éléments à zéro. (FAC est détruit dans ce cas).
->	D216	D186	ISLETC, teste si le contenu de l'accumulateur est une lettre ASCII (A à Z). A la sortie C=1 si (A) est une lettre, sinon C=0. Le contenu de l'accumulateur reste intact.
	D235	D1A3	NEWVAR, déplace les tableaux pour faire de la place à une nouvelle variable simple.
	D297	D2Ø5	constante -32768 90 80 00 00 00
->	D29F	D2ØD	MAKINT, évalue l'expression numérique à TXTPTR et convertit le résultat, qui doit être positif et inférieur à 3276B, en un entier sur deux octets dans FACMO,LO (#D3,D4).
->	D2A9	D217	AYINT, si FAC > -32768 et < 32767 alors QINT rend entier le FAC. Sinon ILLEGAL QUANTITY.
	D2BB	D229	ARRAY, appelée par PTRGET, localise l'élément d'un tableau ou en crée un.

D333

D336

D29D

D2AØ

?BAD SUBSCRIPT ERROR

?ILLEGAL QUANTITY ERROR

- ATMOS ORIC-1 description
 - D47E D3D6 <- FRE, fonction, appelle GARBAGE, calcule FRETOP STREND.
- -> D499 D3ED GIVAYF, rend flottant l'entier signé dans Y(-),A(+) Attention, octet faible Y.
 - D4A6 D3FA <- POS, fonction, charge Y avec CURCOL puis -> SNGFLT
- -> D486 D3FD SNGFLT, rend flottant l'entier non signé dans Y.
 - D4BA D401 <- DEF, si USR, met adresse dans #22,23. Si FN, inscrit variable dans la table.
 - D4D7 D41E ?ILLEGAL DIRECT ERROR
 - D4DA D421 ?UNDEF'D FUNCTION ERROR
 - D593 D4D8 <- STR\$, fonction, convertit un nombre en chaîne dans buffer FAC, #FF-110 (signe en #FF) puis -> STRLIT.
- -> D5A3 D4E8 STRINI, fait de la place pour une chaîne et crée un descripteur pour elle dans DSCTMP (#DØ,D1,D2). A l'entrée (A) = longueur de la chaîne.
- -> D5AB D4F0 STRSPA, appelle GETSPA et range le descripteur en DSCTMP (#D0,D1,D2).
- -> D5B5 D4FA STRLIT, range un guillemet dans CHARAC (#24) et dans ENDCHR (#25) de façon à ce que STRLT2 s'arrête sur lui.
- -> D58B D500 STRLT2, prend une chaîne dont le premier caractère est pointé par A(-),Y(+) et construit un descripteur pour elle dans DSCTMP (#D0,D1,D2). PUTNEW transfère ce dernier dans un registre temporaire et laisse un pointeur dans FACMO,LO. Les caractères autres que zéro qui terminent la chaine doivent être inscrits dans CHARAC et ENDCHR Le guillemet de début doit être supprimé avant l'entrée dans STRLT2. Poursuit avec PUTNEW.
- -> D5F4 D539 PUTNEW, range DSCTMP dans un descripteur temporaire pointé par FACMO,LO et fait VALTYP (#28) = #FF (chaîne).
 - D5FA D53F ?FORMULA TOO COMPLEX ERROR.
- -> D61E D563 GETSPA, fait de la place pour une chaîne de caractères. Peut appeler GARBAGE. Déplace FRESPC et FRETOP vers le bas de façon à pouvoir ranger la chaîne. A l'entrée (A) = nombre de caractères. A la sortie, (A) est inchangé; pointeur vers l'espace créé en X(-),Y(+), FRETOP et FRESPC. ?OUT OF MEMORY ERROR si pas de place.

ATHOS	ORIC-1	description	
-> D650	D595	GARBAGE, déplace toutes les chaînes en vigueur le plus haut possible en mémoire pour augmenter l'espace libre disponible.	
-> D767	DAAC	CAT, concaténation de deux chaînes. FACMO(+) FACLO(-) pointe sur le descripteur de la première chaîne et TXTPTR sur le signe +	
D782	D6A7	?STRING TOO LONG ERROR	
-> D7A4	D6E9	MOVINS, déplace une chaîne dont le descripteur est pointé par STRNG1 (#DE,DF) vers l'emplacement en mémoire pointé par FRESPC (#A4,A5).	
-> D7B2	D6F7	MOVSTR, déplace une chaîne pointée par X(-),Y(+) de longueur (A) vers l'emplacement en mémoire pointé par FRESPC (#A4,A5).	
-> D7CD	D712	FRESTR, s'assure que FAC adresse une chaîne puis exécute FREFAC.	
-> p7pø	D715	FREFAC, charge le pointeur du descripteur contenu dans FACMO,LO (#D3,D4) dans A(-),Y(+) puis exécute FRETMP.	
-> D7D4	D719	FRETMP, libère l'espace occupé par une chaîne temporaire. A l'entrée, le pointeur du descripteur est en A(-),Y(+). Un test est fait pour déterminer s'il s'agit d'un descripteur temporaire alloué par PUTNEW. Si oui, ce dernier est libéré par mise à jour de TEMPPT (#85) puis un autre test détermine si la chaîne occupe la plus basse position en mémoire. Si oui, cette zone est libérée par mise à jour de FRETOP (#A2,A3). A la sortie, l'adresse de la chaîne est en INDEX (#91,92) et en X(-),Y(+) et sa longueur en A.	
-> D8Ø5	D74A	FRETMS, libère le descripteur temporaire sans libérer la chaîne. A l'entrée A(-) et Y(+) pointent le descripteur à libérer. A la sortie, Z = 1 si le descripteur est libéré.	
D816	D75B <-	CHR\$, fonction, appelle CONINT, STRSPA poursuit avec PUTNEW.	
D82A	D76F <-	LEFT\$, fonction, cette routine et les deux suivantes, isolent un élément d'une chaîne (découpage) et le rangent en haut de mémoire.	
D856	D79B <-	RIGHT\$, fonction	
1880	D7A6 <-	MID\$, fonction	
D888	D7DØ	INSTRNG, sous-routine commune à LEFT\$, RIGHT\$ et MID\$. Vérifie ')', POP l'adresse de retour et recueille le pointeur du descripteur et le premier paramètre.	

D8A6 D7EB <- LEN, fonction

DBAC D7F1 GETSTR, sous-routine appelée par LEN, ASC, VAL utilise FRESTR. Adresse chaîne en INDEX (#91,92) et longueur en A.

D8B5 D7FA <- ASC, fonction

- -> D8C5 D8ØA GETBYTC, appel de CHARGET pour escamoter un caractère puis -> GETBYT.
- -> D8C8 D8ØD GETBYT, évalue la formule à TXTPTR, laisse le résultat en FAC puis exécute CONINT.
- -> DBCB DB10 CONINT, convertit FAC en un nombre entier sur un octet dans X et FACLO (#D4).

 Sort normalement par CHARGET.

 Si FAC > 255 ou < 0 sortie ILLEGAL QUANTITY ERROR.

DBD7 DB1C <- VAL, fonction

- -> D916 D85B GETNUM, lit un nombre sur deux octets pointé par TXTPTR et l'inscrit dans LINNUM (#33,34), vérifie la présence d'une virgule puis lit une valeur sur un octet et la place dans X.

 Cette dernière valeur doit être en décimal sur ORIC-1 (voir POKE en #D894).

 A l'entrée, TXTPTR pointe sur le premier caractère de la formule. Utilise FRMNUM, GETADR, CHKCOM, GETBYT.
- -> D91C D861 COMBYTE, vérifie la présence d'une virgule et inscrit une valeur sur un octet dans X.
 A l'entrée, TXTPTR pointe sur la virgule.

D894 <- POKE, utilise GETNUM.

- -> D922 D867 GETADR, convertit le nombre en FAC (0-65535) en un entier sur deux octets dans LINNUM (#33,34).
 - D938 D87D <- PEEK, fonction, utilise GETADR. Contenu adresse pointée par LINNUM transféré dans Y et mis en FAC.
 - Note pour ORIC-1: La valeur à inscrire en mémoire doit être obligatoirement en décimal.

 Si le POKE d'une valeur hexa était totalement sans effet, ce ne serait que génant. En réalité, il y a un risque de plantage irrécupérable avec certaines valeurs.

 POKE appelle GETNUM qui inscrit l'adresse dans LINNUM (#33,34). Mais sur ORIC-1, la routine de conversion hexa-binaire en #E813 utilise aussi ce pointeur. Résultat, la valeur hexa est inscrite en page zéro (zone sensible) à l'adresse qui lui correspond. Exemple, POKE#400,#1A aboutit à un POKE#1A,#1A qui est catastrophique.

 Le défaut a été corrigé sur ATMOS.

Le pointeur #0C,0D remplace LINNUM dans la routine de conversion hexa-binaire en #E94C.

D94F

- ATMOS ORIC-1 description
 - D958 D89D <- WAIT, utilise FRMNUM et GETADR. Temporisation en Y(-),X(+), A=#02, -> routine WAIT (#EEC9,#EDAD)
 - D967 D8AC <- DOKE, utilise FRMNUM, GETADR, CHKCOM.
 - D983 D8C8 <- DEEK, fonction, utilise GETADR, nombre sur deux octets, contenu dans adresse pointée par LINNUM et dans adresse suivante est mis en FAC.
- -> D8D5 GIVAYF2, rend flottant l'entier non signé dans Y(-),A(+). Attention octet faible Y. ORIC-1. (pour ATMOS, #DF40, chapitre 19).
 - D8E4 constante 65535 91 00 00 00 00 (ORIC-1)
 - DBE9 constante PI 82 49 0F DA 9E (ORIC-1)
 - D8EE <- PI, fonction, FAC = PI (QRIC-1)
 - D993 D8F5 routine conversion hexa pour HEX\$.
 - D985 D917 <- HEX\$, fonction, utilise GETADR.

 Le nombre est converti, affublé du symbole # puis rangé sous forme de chaîne terminée par Ø dans le buffer FAC, FBUFF en #FF-11Ø (symbole # en #FF).
 - D9DE D937 <- LORES, met en mode texte puis GETBYT recueille le format en X.
- -> D9EA D943 LORES 1
- -> D9ED D946 LORES Ø
- -> DAØC D965 BASCALC, calcule l'adresse de base d'une ligne à l'écran. A l'entrée, (A)=ligne+1 (coord.vert.+1). A la sortie, adresse en #1F,20.
- -> DA22 D996 GTVALS, recueille coordonnées x et y pour SCRN et PLOT. x+1 -> #2F8 (GCOL), y+1 détermine adresse de base de la ligne (#1F,20).
 - DA3F D9B4 <- SCRN, fonction
 - DA51 D9C6 <- PLOT
 - DA85 D9FA <- REPEAT
 - DAA1 DA16 <- PULL, UNTIL
 - DAAD DA22 ?BAD UNTIL ERROR
 - DADA DA4F <- KEY\$, fonction, enregistre une chaine de longueur 1, le caractère en A, si une touche est enfoncée. Sinon, de longueur Ø (chaine nulle).
 - DAF6 DA6B vérification mode texte.
 - DAFD DA72 ?DISPLAY TYPE MISMATCH ERROR.



ROM - 4° PARTIE

ROUTINES VIRGULE FLOTTANTE

ATMOS #DB04-E4AB ORIC-1 #DA79-E4A7

Principaux emplacements en page zéro concernés:

Remarque: le système se sert de la plupart de ces emplacements à d'autres usages en dehors des routines virgule flottante.

Registres:

-	FAC	ARG	TEMP1	TEMP2	TEMP3	RND
EXP	DØ	DB	C6	CB	BD	FA
HO	D1	D9	C7	CC	BE	FB
MOH	D2	DA	CB	CD	BF	FC
MO	D3	DB	C9	CE	CØ	FD
LO	D4	DC	CA	CF	Ci	FE
SGN	D5	DD	- f	orme co	ndensée	-

Autres:

2D 91,92	SIGNFLG INDEX	indicateur de signe fonction TAN. pointeur temporaire
95-98	RESULT	résultat dernière multiplication ou division.
C3-C5	JMPADRS	saut à adresse modifiable.
C5	EXTRASV	extra précision
D6	SERLEN	utilisé en calcul de séries.
D7	FPGEN	usage général.
DE	SGNCPR	reçoit résultat EOR entre signes FAC et ARG.
DF	EXTRAFAC	octet d'extension pour précision FAC.
EØ,E1	STRNG2	aussi SERPNT pour calcul série.

ATMOS ORIC-1 description

- routines arithmétiques
- -> DB04 DA79 FADDH, ajoute 1/2 à FAC.
- -> DBØB DABØ FSUB, inscrit dans ARG (#D8-DD) le nombre en mémoire pointé par A(-),Y(+) puis exécute FSUBT.
- -> DBØE DA83 <- FSUBT, opérateur '-', soustrait FAC de ARG. Résultat dans FAC. A l'entrée A et I doivent refléter la valeur de FACEXP.

Remarques: La plupart des routines de déplacement donnent la valeur correcte à A et Z mais il est prudent de placer un LDA #DØ avant l'entrée dans les routines arithmétiques.

Avant utilisation de FSUBT, FADDT, FMULTT et FDIVT le résultat de l'opération OU exclusif (EOR) entre les octets de signe de FAC et ARG (#D5 et #DD) doit être placé dans SGNCPR (#DE).

Les routines citées ne font pas appel à CONUPK qui en est normalement responsable.

ATHOS	ORIC-1	description

- -> DB22 DA9A FADD, inscrit dans ARG (#D8-DD) le nombre en mémoire pointé par A(-),Y(+) puis exécute FADDT.
- -> DB25 DA9A <- FADDT, opérateur '+', additionne FAC et ARG, résultat dans FAC. A l'entrée A et I doivent refléter la valeur de FACEXP.
 - DC39 DBEØ ?OVERFLOW ERROR
 - DC77 DC46 constante LN(10) 82 13 5D 8D DE
 - DC7C constante PI 82 49 ØF DA 9E (ATMOS)
 - DC81 DC4B constante 1 81 00 00 00 00 utilisée par FOR pour STEP par défaut.
 - DC86 DC50 index (#03) et coefficients calcul série LN.
 - DC9B DC65 constante SQR(.5) 80 35 04 F3 34
 - DCAØ DCAA SQR(2) 81 35 Ø4 F3 34
 - DCA5 DCAF 1/2 80 80 00 00 00
 - DCAA DC74 LN(2) 80 31 72 17 F8
- -> DCAF DC79 <- LN, fonction, log base e de FAC, résultat en FAC.
- -> DCED DCB7 FMULT, inscrit dans ARG (#D8-DD) le nombre en mémoire pointé par A(-),Y(+) puis exécute FMULTT.
- -> DCFØ DCBA <- FMULTT, opérateur '*', multiplie FAC par ARG, résultat dans FAC. A l'entrée A et Z doivent refléter la valeur de FACEXP.
- -> DD51 DD4D CONUPK, inscrit dans ARG (#D8-DD) le nombre virgule flottante en mémoire pointé par A(-),Y(+). A la sortie A et Z reflètent la valeur de FACEXP.
- -> DDA7 DDA3 MUL10, multiplie FAC par 10, résultat dans FAC. Fonctionne pour nombres positifs ou négatifs.
 - DDBE DDBA constante 10 84 20 00 00 00
- -> DDC3 DDBF DIV10, divise FAC par 10, résultat dans FAC, positif seulement.
- -> DDD4 DDD0 <- LOG, fonction, log base 10 de FAC, résultat en FAC.
- -> DDDE DDDA FDIV2, inscrit dans FAC (#DØ-D5) le nombre en mémoire pointé par A(-),Y(+) puis exécute FDIVT.
- -> DDE4 DDEØ FDIV, inscrit dans ARG (#D8-DD) le nombre en mémoire pointé par A(-),Y(+) puis exécute FDIVT.
- -> DDE7 DDE3 <- FDIVT, opérateur '/', divise ARG par FAC, résultat dans FAC. A l'entrée A et Z doivent refléter la valeur de FACEXP.

DE5F DE5B ?DIVISION BY ZERO ERROR.

-> DE77 <- PI, fonction, FAC = PI (ATMOS)

- déplacements
- -> DE7B DE73 MOVFM, transfère dans FAC le nombre en mémoire pointé par A(-),Y(+). A la sortie A et Z reflètent la valeur de FACEXP.
- -> DEAØ **DE98** MOV2F, met FAC sous forme condensée et le transfère dans le registre TEMP2 (#CB-CF). A la sortie A et Z reflètent la valeur de FACEXP.
- -> DEA3 DE9B MOV1F, met FAC sous forme condensée et le transfère dans le registre TEMP1 (#C6-CA). A la sortie A et Z reflètent la valeur de FACEXP.
- -> DEA5 DE9D MOVML, met FAC sous forme condensée et le transfère en page zéro à l'adresse pointée par X. Utilise MOVMF. A la sortie A et Z reflètent la valeur de FACEXP.
- -> DEAD DEA5 MOVMF, met FAC sous forme condensée et le transfère en mémoire à l'adresse pointée par X(-),Y(+). A la sortie A et Z reflètent la valeur de FACEXP.
- -> DED5 DECD MOVFA, transfère ARG (#D8-DD) dans FAC (#D0-D5). A la sortie (A) = FACEXP et Z = 1.
- -> DEE5 DEDD MOVAF, transfère FAC (#DØ-D5) dans ARG (#D8-DD). A la sortie (A) = FACEXP et Z = 1.
- fonctions, affichage ...

DEF4 DEEC RNDB, arrondit FAC, utilise bit 7 de EXTRAFAC (#DF)

DFØB DEFC <- FALSE, fonction, A = 00 -> FLOAT.

DFØF DF00 <- TRUE, fonction, A = FF -> FLOAT.

-> DF13 DFØ4 SIGN, modifie le contenu de A selon la valeur de FAC.

Si FAC est positif, A = 1

Si FAC = Ø A = Ø

Si FAC est négatif, A = FF.

-> DF21 DF12 <- SGN, fonction, appelle SIGN et place le résultat dans FAC. A la sortie,

FAC = 1 si FAC était positif. FAC = 0 si FAC était égal à zéro.

FAC = -1 si FAC était négatif.

FLOAT, rend flottant l'entier signé dans -> DF24 DF15 l'accumulateur A.

,	ATHOS	ORIC-1	description
->	DF40		GIVAYF2, rend flottant l'entier non signé dans Y(-),A(+) (0-65535). Attention, octet faible Y. (ORIC-1, voir en D8D5, chapitre 18)
->	DF49	DF31 <-	ABS, fonction, valeur absolue de FAC, met à zéro le bit 7 de FACSGN (#D5).
->	DF4C	DF34	FCOMP, compare FAC et un nombre virgule flottante en mémoire pointé par A(-),Y(+). A la sortie, A = 1 si (A,Y) < FAC A = 0 si (A,Y) = FAC A = FF si (A,Y) > FAC
V. = V. = V	DF8C	DF74	QINT, Quick greatest integer function, laisse INT(FAC) dans FACHO,MO,LO signé. Suppose FAC < 2^23 (8388608 décimal).
->	DFBD	DFA5 <-	INT, fonction, plus grande valeur entière de FAC. Utilise QINT et rend le résultat flottant.
->	DFE7	DFCF	FIN, conversion en virgule flottante et mise en FAC d'un nombre décimal ou hexadécimal à TXTPTR à l'aide de CHARGET. Suppose que A et les indicateurs ont été renseignés par l'appel de CHARGET ayant trouvé le premier caractère. Si nombre hexa, branchement à FIN-3 pour saut à routine de traitement particulière.
->	EØ76	EØ72	ADDACC, ajoute le contenu de A à FAC.
	EØAB	EØA7	Cte 9999999999 (100 millions-1/10°) 9B 3E BC 1F FD
	EØBØ	EØAC	999999999 (milliard-1) 9E 6E 6B 27 FD
	EØB5	EØB1	1000000000 1E+09 (milliard) 9E 6E 6B 28 00
->	EØBA	EØ86	INPRT, affiche IN et le numéro de ligne en cours dans CURLIN. Utilise LINPRT.
->	EØC5	EØC1	LINPRT, affiche en décimal le nombre entier non signé sur deux octets dans X(-) et A(+).
->	EØCF	EØCB	PRNTFAC, affiche en décimal le contenu de FAC. Utilise FOUT et STROUT. FAC est détruit.
->	EØD5	EØD1	FOUT, crée une chaine dans FBUFF (#FF-110) équivalente à la valeur de FAC. A la sortie, A(-),Y(+) pointe la chaine. FAC est détruit. Remarque: bug sur ORIC-1 en #E0D3; LDA >#02 au lieu de LDA >#20, provoque affichage attribut encre verte par PRINT STR\$() au lieu du caractère espace destiné à remplacer le signe +.
	E205	E2Ø1	constante 1/2 80 00 00 00 00
	E20A	E2Ø6	table de puissances de 10 sur 4 octets.

- ATMOS ORIC-1 description
- -> E22E E22A <- SQR, fonction, extrait la racine carrée de FAC, résultat en FAC.
- -> E238 E234 <- FPWRT, opérateur '^', élève ARG à la puissance FAC, résultat dans FAC. A l'entrée A et Z doivent refléter la valeur de FACEXP.
- -> E271 E26D <- opérateur '>', NEGOP, FAC = -FAC.
 - E27C E278 constante 1/LN(2) 81 38 AA 38 29
 - E281 E27D index (#07) et coefficients calcul série EXP.
- -> E2AA E2A6 <- EXP, fonction, calcule e puissance FAC, résultat en FAC.
 - E347 E343 valeur utilisée par RND 98 35 44 7A
 - E34B E347 " " 68 28 B1 46
- -> E34F E34B <- RND, fonction, forme un nombre aléatoire dans FAC. Utilise RNDSEED (#FA-FE).
- fonctions trigonomètriques
- -> E38B E387 <- COS, fonction, calcule COS(FAC), résultat en FAC.
- -> E392 E38E <- SIN, fonction, calcule SIN(FAC), résultat en FAC.
- -> E3DB E3D7 <- TAN, fonction, calcule TAN(FAC), résultat en FAC.
 - E407 E403 constante PI/2 81 49 0F DA A2
 - E40C E408 constante 2 PI 83 49 0F DA A2
 - E411 E40D constante 1/4 7F 00 00 00 00
 - E416 E412 index (#05) et coefficients calcul série SIN.
- -> E43F E43B <- ATN, fonction, calcule ARCTAN(FAC), résultat en FAC
 - E46F E46B index (#ØB) et coefficients calcul série ATN.



ROM - 5° PARTIE

ATMOS #E4AC-FFFF ORIC-1 #E4A8-FFFF

DIVERS ENTREES/SORTIES, GRAPHISME ET SON

- lecture et enregistrement sur cassette

Ces routines ont été remaniées pour l'ATMOS, afin d'inclure les nouvelles facilités, Join et Verify, et d'intégrer les nouvelles commandes STORE et RECALL. Les pointeurs et drapeaux en page zéro ont été reportés en page 2.

ATHOS.	ORIC-1	description
E4AC +/E4EØ	E4A8	appelée par CLOAD, lecture en-tête
TEACH	E4AB	affiche Searching
E4EØ		vérifie ou charge
	E4DB	affiche Loading
E50B	E 50 3	messages chargement.
E56C	E554	teste fin de programme.
E570		affiche Searching
E585	E57B	appelée par CSAVE, affiche Saving
E594		affiche Found, Loading ou Verifying
ESES	<u></u>	<pre>asic, <c>ode machine, <s>trng, <i>nt, <r>eel.</r></i></s></c></pre>
E607	E57B	enregistrement en-tête, OUTLED, OUTBYT
E62E	E5A7	enregistrement programme ou données.
E645	E5BC	message Saving
E651		test erreur
E656		errors found
-> E65E	E5C6	OUTBYT, sort une valeur en A vers le magnétophone à la vitesse prévue.
-> E6C9	E63Ø	RDBYTE, recueille en A à la vitesse prévue une valeur provenant du magnétophone.
-> E735	E696	GETSYN, lit les valeurs en provenance du magnétophone jusqu'à synchronisation.

-> E76A E6CA VIA ON, appelée par CLOAD ou CSAVE, charge registres 6522 pour E/S cassette.

E782 E6E2 parametres pour 6522

E7B2 E725 recueil des paramètres CLOAD ou CSAVE.

-> E853 E79D MAKINT2, évalue l'expression numérique à TXTPTR et convertit le résultat, qui doit être positif (0-65535) en un entier sur deux octets dans LINNUM. utilise FRMNUM et GETADR.

E85B E7AA <- CLOAD

E909 E7DB <- CSAVE

-> E93D E804 VIA OFF, reconnection clavier.

fin des routines d'enregistrement et de chargement classiques.

E946 E80D <- CALL, appelle FRMNUM, GETADR, saut indirect à l'adresse contenue dans LINNUM.

-> E94C E813 HEXGET, lit un nombre hexadécimal à l'aide de CHARGET et le convertit en un entier non signé sur deux octets dans Y(-),A(+). A l'entrée TXTPTR pointe sur le caractère précédant le nombre (symbole #).

E981 E848 appelée par FIN, met en FAC le nombre hexadécimal (précédé de #) à TXTPTR.

sauvegarde ou chargement de tableaux sur cassette, ATMOS seulement.
 Les tableaux doivent être préalablement dimensionnés.

E987 <- STORE, sauvegarde sur cassette d'un tableau de variables numériques (réelles ou entières) ou alphanumériques.

E9D1 <- RECALL, chargement d'un tableau de variables à partir d'une cassette.

- tables pour HIRES et son

EAC1 E84E table d'adresses des routines graphiques et sonores. Les adresses sont rangées dans l'ordre des codes, de SOUND (A7) à INK (B2).

Une adresse dans la table correspond à l'adresse d'entrée - 1 (recueil en pile comme pour les autres commandes).

EAD9 E866 nombre de paramètres pour chaque commande.

EAES E872 valeur Ø ou 1 (CURMOV et DRAW traitement spécial), sera rangée en #2C3.

- entrées communes routines graphiques et sonores.
 - EAFO = E87D <- entrée de toutes instructions graphiques HR. test mode HIRES? si oui -> HIRES-SON, si non erreur.
 - EAF7 E884 ?DISPLAY TYPE MISMATCH ERROR.
 - EAFC E889 <- HIRES-SON, détermination adresse d'entrée commande graphique ou sonore, recueil des paramètres puis exécution.

 Le registre Y, (code du mot-clé #80)*2, sert d'index dans les tables précédentes.

 Retour en #EB6E ATMOS, #E8FB. Si la commande n'a pas été exécutée (#2E0=1, paramètre invalide) message ?ILLEGAL QUANTITY ERROR.
- lecture mémoire clavier
 - EB78 E905 RDKEY, lit le caractère en #2DF, mémoire clavier, (ASCII négatif).

 A la sortie, remet la mémoire clavier
 à zéro et le caractère en A en ASCII positif.

 N = 1, positionné éventuellement par PLP,
 indique qu'un caractère a été lu.
- mode d'affichage
 - EBCE E95B <- HIMEM, mise à jour MEMSIZ, utilise FRMNUM et GETADR, sortie par CLEARC ou OUT OF MEMORY si HIMEM < VARTAB.
- -> EBE7 E974 <- GRAB, teste mode non HIRES puis modifie MEMSIZ, ajoute #1C à l'octet de poids fort, sortie par CLEARC.
- -> ECOC E994 <- RELEASE, rétablit MEMSIZ (si possible), soustrait #1C à l'octet de poids fort, sortie par CLEARC.
- -> EC21 E9A9 <- TEXT, si mode texte déja en vigueur, RTS. Sinon, bit Ø de #2CO mis à zéro puis transfert jeu de caractères de #9800 en #8400.
- -> EC33 E9BB <- HIRES, TYPE MISMATCH ERROR si #2C0=01 (GRAB), prépare mémoire d'écran (#A000-BF3F) puis transfère jeu de caractères de #B400 à #9800.
 - EC45 E9CD <- POINT, fonction, recueil des paramètres par FRMEVL et GETADR, rangés en #2E1,2E2 et #2E3,2E4. résultat mis en FAC.
- charget
 - EC9C EA24, code de CHARGET, routine d'obtention d'un caractère, recopié en #E2 à l'initialisation.

 (voir 3° partie, chapitre 11)

ECB9 EA41 SETINDIC, sous-routine appelée par CHARGET, positionne les indicateurs selon le caractère en A. Z = 1 si (A) = #3A (fin d'instruction) ou (A) = 00 (fin de ligne).

C = 0 si (A) est un chiffre de 0 à 9.

(voir 3° partie, chapitre 11)

- initialisation

-> ECCC EA59 INIT, routine d'initialisation du système, exécutée à la mise sous tension.

ED88 EB43 BYTES FREE et message de présentation.

EBDØ série de JMPs (ORIC-1)

- transfert jeu de caractères

-> EDC4 ECOC MOVE, routine de transfert du jeu de caractères.

Peut servir à transférer le contenu d'une zone
en mémoire. Avant l'appel, les paramètres doivent
être inscrits dans les emplacements suivants:

ATMOS	ORIC-1				
ØC	200	adresse	e départ,	octet	faible
ØD	201	11		21	fort
ØE	202	adr.des	stination,	octet	faible
ØF	203	11	11	u	fort
10	204	nombre	d'octets,	octet	faible
11	205	11	11	11	fort

Utilisation en Basic, voir page 48.

- interruptions

->	EE22	EDØ9	IRQ, routine des interruptions masquables, utilisée par le système pour la surveillance du clavier. Elle peut être déroutée. Un JMP IRQ se trouve en #244 ATMOS, en #228 ORIC-1. Le RTI est en #24A ATMOS, #230 ORIC-1.
	EE34	ED1B	décrémentation TIMERS, saut à routine de scrutation du clavier.
	EE65	ED4C	retour de scrutation clavier, caractère rangé en #2DF (ASCII négatif).
->	EEC9	EDAD	WAIT, octet faible temporisation en Y, octet fort en X, A=#02.

- routines graphiques: ATMOS #EED8-F494, ORIC-1 #EDBC-F3FF

Reportez-vous au manuel Basic pour la définition de x, y, fb, etc.. Les paramètres sont des nombres 16 bits signés. L'octet faible est rangé en premier. Au retour #2E0=1 si param. hors limite, affich. message d'erreur. Les valeurs limites et les adresses indiquées sont en hexadécimal.

ATHOS ORIC-1 description

- -> F0C8 F02D <- CURSET, positionnement du curseur paramètres: x (00-EF) y (00-C7) fb (00-03) à ranger en: 2E1,2E2 2E3,2E4 2E5,2E6 Au retour, les nouvelles coordonnées du curseur sont inscrites en CURX (#219) et CURY (#21A). Ces emplacements peuvent être renseignés directement avant l'appel d'une routine de tracé.
- -> FØFD FØ64 <- CURMOV, déplacement du curseur paramètres: x (00-EF) y (00-C7) fb (00-03) à ranger en: 2E1,2E2 2E3,2E4 2E5,2E6 x et y représentent un déplacement par rapport à la position actuelle du curseur. La nouvelle position doit rester dans les limites autorisées.
- -> F110 F079 <- DRAW, tracé d'un trait
 paramètres: idem CURMOV.
 Les registres A et X doivent etre sauvegardés
 avant l'appel.
- -> F11D F093 <- PATTERN, configuration des pixels affichés.
 paramètre: masque (00-FF) à ranger en #2E1,2E2.
 Le contenu du registre X est détruit.
- -> F12D F0A5 <- CHAR, affiche un caractère ASCII sur l'écran HR. params: ASCII (20-7F), Std=00 Alt=01, fb (00-03) en: 2E1,2E2 2E3,2E4 2E5,2E6
- -> F204 F17F <- PAPER, couleur du fond, en HIRES ou LORES couleur (00-07) en 2E1,2E2.
- -> F210 F18B <- INK, couleur de l'encre idem PAPER.
- -> F268 F1E5 <- FILL, colore une zone déterminée.

 params: nb lign.(Ø1-C8) nb cell.(Ø1-28) n (Ø0-7F)

 en: 2E1,2E2 2E3,2E4 2E5,2E6

 Le fonctionnement de FILL est complexe et encore

 mal connu, notamment en ce qui concerne

 l'incidence du 3° paramètre.
- -> F37F F2E5 <- CIRCLE, trace un cercle.

 paramètres: rayon (01-63) fb (00-03)

 en: 2E1,2E2 2E3,2E4

 Le centre du cercle est à la position actuelle du curseur.

_	entrées	at	cortine	diverses.
	entrees		SULLIES	III VELSES.

- entrees	ec sor	cies diverses.
	F400	série de JMPs. (ORIC-1).
F495	F43C	routine de scrutation du clavier. Variables système concernées #208, 209, 20A. Code caractère en X, ASCII négatif.
-> F590	F535	W8912, sortie son, la valeur en X est inscrite dans le registre du boitier 8912 spécifié par A.
-> F5C1	F57B	PRTCHR, envoie le caractère en A vers l'imprimante.
F5E2	F583	valeurs permettant de déterminer adresse sous-routine 'affichage' caractère de controle.
-> F6Ø2	F5D3	appelée par VDU, 'affiche' le caractère de controle contenu dans l'accumulateur.
	F663	messages PRINT, CAPS et espaces (ORIC-1)
-> F75A	F729	affiche ou efface CAPS suivant valeur de #200. Utilise STOUT.
-> F75F	F72E	affiche CAPS.
-> F766	F735	efface CAPS.
F770		message CAPS et espaces (ATMOS)
-> F77C	F73F	VDU, affiche à l'écran le caractère en X et déplace le curseur à la position suivante.
-> F865	F82F	STOUT, affiche un message en haut de l'écran, adresse en A(-),Y(+), colonne de début en X. Le message doit se terminer par 00.
- mise en	route,	initialisation.
F87C	F841	sauts à différentes adresses, inscrits en page 2 au démarrage du système.

F87C	F841	sauts à différentes adresses, inscrits en page 2 au démarrage du système.
-> F88F	F84A	COLDSTART, démarrage à froid, un appel de cette routine produit le même effet que de débrancher puis de rebrancher la prise de courant. Le système est réinitialisé.
	F874	message MEMORY ERROR (ORIC-1)
-> F8B2	F882	WARMSTART, RESET, démarrage à chaud, programme et variables sont conservés.
-> F8DØ	F898	STDCHR, établit les caractères normaux. Transfert du jeu de caractères en mémoire vive.

ATMOS	ORIC-1	description
FBDC	F8A9	réduit l'écran texte aux deux lignes du bas pour HIRES.
F90E	F8D1	vidéo + curseur.
-> F92Ø	F8E3	HIRES, nettoie la mémoire écran HR et met en mode HIRES. Pour sortir, CALL #TEXT (ci-après) ou RESET.
-> F967	F923	TEXT
F982	F93E	renseigne les pointeurs pour transfert jeu de caractères, ATMOS #0C,11, ORIC-1 #200,205.
F9C9	F97F	même effet que CLS. Etablit les variables concernées par la gestion de l'écran texte.
FA14	F9C0	aménagement mémoire vive par COLDSTART.
- routine	s sonore	5
FA86	FA6C	exécution d'un son, appelle W8912.
-> FA9F	FA85 <-	PING (CTRL-G)
-> FAB5	FA98 <-	SHOOT
-> FACB	FAB1 <-	EXPLODE
-> FAE1	FAC7 <-	ZAP
-> FB14	FAFA	KBBEEP, bruit clavier, caractères normaux.
-> FB2A	FB10	CONTBP, bruit clavier, caractères de controle.
-> FB40	FB26 <-	SOUND params: canal (00-06) ton.(00-FFFF) vol.(00-0F) en: 2E1,2E2 2E3,2E4 2E5,2E6
-> FBD0	FBB6 <-	PLAY params: canal son (00-07) canal bruit (00-07) en: 2E1,2E2 2E3,2E4 params: enveloppe (00-07) durée (00-7FFF) en: 2E5,2E6 2E7,2E8
FC10	FBF4	valeurs utilisées par PLAY
-> FC18	FBFE <-	MUSIC params: canal son (00-03) octave (00-07) en: 2E1,2E2 2E3,2E4 params: note (00-0C) volume (00-0F) en: 2E5,2E6 2E7,2E8
FC5E	FC44	valeurs utilisées par MUSIC.

- matrices de caractères.

FC78 FC70 jeu de caractères, matrices sur 8 octets.

FF70 valeurs utilisées par la routine de scrutation du clavier.

- vecteurs d'interruption.

FFFA, FFFB NMI, contient adresse saut vers RESET,

#247 sur ATMOS, #228 sur ORIC-1.

FFFC,FFFD réinitialisation, #F88F ATMOS, #F42D ORIC-1.

FFFE, FFFF IRQ, contient adresse saut vers routine

d'interruption IRQ, #244 ATMOS, #228 ORIC-1.

hexa	déc	binaire	ASCII	code op 6502
00	2	00000000	NUL	BRK
01	1	00000001	SOH	ORA (adr, X)
02	2	00000010	STX	
03	3	00000011	ETX	
Ø4	4	00000100	EOT	
05	5	00000101	ENQ	ORA adr
06	6	00000110	ACK	ASL adr
07	7	00000111	BEL	
08	8	00001000	BS	PHP
09	9	00001001	HT	ORA >donnée
ØA	10	00001010	LF	ASL A
ØB	- 11	00001011	ντ	
9C	12	00001011	FF	
ØD	13	00001101	CR	ORA ADR
ØE	14	00001101		ASL ADR
			S0	HOL HUN
ØF	15	00001111	SI	
10	16	00010000	DLE	BPL
11	17	00010001	DC1	ORA (adr),Y
12	18	00010010	DC2	
13	19	00010011	DC3	
14	20	00010100	DC4	
15	21	00010101	NAK	ORA adr, X
16	22	00010110	SYN	ASL adr, X
17	23	00010111	ETB	
18	24	00011000	CAN	CLC
19	25	00011001	EM	ORA ADR,Y
1A	26	00011010	SUB	
1B	27	00011011	ESC	
10	28	00011100	FS	
1 D	29	00011101	GS	ORA ADR,X
1E	30	00011110	RS	ASL ADR, X
1F	31	00011111	us	
20	32	00100000		JSR ADR
21	33	00100001	į	AND (adr,X)
22	34	00100010	U	
23	35	00100011	#	
24	36	00100100	\$	BIT adr
25	37	00100101	%	AND adr
26	38	00100110	&	ROL adr
27	39	00100111	,	
28	40	00101000	(PLP
29	41	00101001)	AND >donnée
2A	42	00101010	*	ROL A
2B	43	00101011	+	
2C	44	00101100	,	BIT ADR
2D	45	00101101	_	AND ADR
2E	46	00101110	•	ROL ADR
2F	47	00101111	,	
_,	17	~~~~	6 ₽ 30	

Les 32 premiers codes (00 à 1F) sont interprétés comme caractères de contrôle ou comme attributs suivant la façon dont ils sont envoyés à l'écran. Voir Annexe 2.

hexa	déc	binaire	ASCII	code op 6502
30	48	00110000	Ø	BMI
31	49	00110001	1	AND (adr),Y
32	50	00110010	2	 _
33	51	00110011	3	
34	52	00110100	4	
35	53	00110101	5	AND adr, X
36	54	00110110	6	ROL adr, X
37	55	00110111	7	
38	56	00111000	8	SEC
39	57	00111001	9	AND ADR,Y
3A	58	00111010	:	
3B	59	00111011	;	
3C	60	00111100	, <	
3D	61	00111101	=	AND ADR, X
3E	62	00111110	>	ROL ADR.X
3F	63	00111111	?	
40	64	01000000	ė	RTI
41	65	01000001	Ā	EOR (adr,X)
42	66	01000010	В	
43	67	01000011	č	
44	68	01000100	D	
45	69	01000101	Ē	EOR adr
46	70	01000110	F	LSR adr
47	71	01000111	G	
48	72	01001000	H	PHA
49	73	01001001	ï	EOR >donnée
4A	74	01001010	J	LSR A
4B	75	01001011	ĸ	
4C	76	01001100	Ĺ	JMP ADR
4D	77	01001101	M	EOR ADR
4E	78	01001110	N	LSR ADR
4F	79	01001111	Ö	
50	80	01010000	P	BVC
51	81	01010001	Q.	EOR (adr),Y
52	82	01010010	Ŕ	
53	83	01010011	S	
54	84	01010100	Ť	
55	85	01010101	ប់	EOR adr, X
56	86	01010110	v	LSR adr, X
57	87	01010111	W	
58	88	01011000	x	CLI
59	89	01011001	Ŷ	EOR ADR,Y
5A	90	01011010	Ž	
5B	91	01011011	Ē	
5C	92	01011100	Š	
5D	93	01011101	ì	EOR ADR, X
5E	94	01011110		LSR ADR,X
5F	95	01011111	*	
	• -			

hexa	dėc	binaire	ASCII	code op 6502	not-clés
60	96	01100000	(6)	RTS	
61	97	01100001	a	ADC (adr, X)	
62	98	01100010	b		
63	99	01100011	C		
64	100	01100100	d		
65	101	01100101	е	ADC adr	
66	102	01100110	f	ROR adr	
67	103	01100111	g		
68	104	01101000	ĥ	PLA	
69	105	01101001	i	ADC >donnée	
6A	106	01101010	j	ROR A	
6B	107	01101011	k		
6C	108	01101100	1	JMP (ADR)	
6D	107	01101101	m	ADC ADR	
6E	110	01101110	n	ROR ADR	
6F	111	01101111	0		
70	112	01110000	P	BVS	
71	113	01110001	9	ADC (adr),Y	
72	114	01110010	ŕ		
73	115	01110011	s		
74	116	01110100	t		
75	117	01110101	u	ADC adr, X	
76	118	01110110	V	ROR adr, X	
77	119	01110111	W		
78	120	01111000	×	SEI	
79	121	01111001	У	ADC ADR,Y	
7A	122	01111010	z		
7B	123	01111011	€		
7C	124	01111100	1		
7D	125	01111101	}	ADC ADR,X	
7E	126	01111110		ROR ADR, X	
7F	127	01111111	DEL		
80	128	10000000			END
81	129	10000001		STA (adr, X)	EDIT
82	130	10000010			STORE
83	131	10000011			RECALL
84	132	10000100		STY adr	TRON
85	133	10000101		STA adr	TROFF
86	134	10000110		STX adr	POP
87	135	10000111			PLOT
88	136	10001000		DEY	PULL
89	137	10001001			LORES
8A	138	10001010		TXA	DOKE
88	139	10001011			REPEAT
80	140	10001100		STY ADR	UNTIL
ap	141	10001101		STA ADR	FOR
8E	142	10001110		STX ADR	LLIST
8F	143	10001111			LPRINT

Les codes 80-97 hexa sont affichés comme attributs par PRINT. Voir Annexe 2

90 144 10010000 BCC NEXT DATA 91 145 10010010 STA (adr),Y DATA 92 146 10010010 DIM 93 147 10010011 DIM 94 148 10010100 STY adr,X READ 95 149 1001010 STX adr,Y LET 96 150 1001010 STX adr,Y LET 97 151 10010011 STA ADR,Y LET 98 152 10011000 TYA RUN 99 153 1001101 STA ADR,Y IF 9A 154 10011010 STA ADR,Y IF 9A 154 10011010 TXS RESTORE 9B 155 10011011 GOSUB 9P 155 10011011 GOSUB 9P 157 10011101 STA ADR,X REM 9D 157 10011101 STA ADR,X REM 9D 157 10011101 STA ADR,X REM 9F 159 10011011 STA ADR,X REM 9F 159 10010111 GRAB A1 160 10100000 LDY donnée RELEASE A1 161 10100001 LDY donnée RELEASE A1 164 10100000 LDY donnée HIRES A3 163 10100011 LDA adr ZAP A6 166 10100010 LDX adr PING A7 167 10100111 SOUND A8 168 10101000 TAY A9 169 10101000 TAY A9 173 1010101 TAY A9 174 1010011 TAY A9 175 1010011 TAY A9 177 1010001 TAY A9 178 10100101 TAY A9 179 179 170 170 170 TAY A9 179 TAY A9 TA	L	 _	h /			
91 145 10010001 STA (adr),Y DATA 92 146 10010010 DIM 93 147 10010011 DIM 94 148 10010100 STY adr,X CLS 95 149 10010110 STA adr,X READ 96 150 10010111 GOTO 97 151 10010111 GOTO 98 152 10011001 STA ADR,Y LET 98 153 10011001 STA ADR,Y IF 98 154 1001101 STA ADR,Y IF 98 155 10011011 GOSUB 99 153 10011011 GOSUB 99 153 10011011 GOSUB 90 157 10011110 STA ADR,X REMINEN 90 157 10011110 STA ADR,X REMINEN 90 157 10011111 GOSUB 91 159 10011111 GRAB 92 159 10011111 GRAB 94 150 1000000 LDY >donnée RELEASE 95 150 1000000 LDY >donnée RELEASE 96 150 1000001 LDA (adr,X) TEXT 97 150 1000011 LDA (adr,X) TEXT 98 150 1000011 LDA adr EXPLODE 97 150 10001110 LDX adr EXPLODE 98 150 10100101 LDA adr SOUND 99 157 10001110 LDA ADR CIRCLE 96 156 10100101 LDA ADR CIRCLE 97 159 10011110 LDA ADR DRAW 98 150 1010101 LDA ADR DRAW 99 150 1010101 LDA ADR DRAW 99 150 1010101 LDA ADR DRAW 90 157 10101101 LDA ADR DRAW 90 157 10101101 LDA ADR CIRCLE 96 150 1010000 BCS CHAR 97 169 1010001 LDA ADR DRAW 97 169 1010001 LDA ADR DRAW 97 169 1010001 LDA ADR CIRCLE 97 150 1010111 GRAB 98 151 10110001 LDA ADR, Y PAPER 98 150 1011101 LDA ADR, Y PAPER 98 150 1011101 LDA ADR, Y CLOAD 99 150 1011101 LDA ADR, Y CLOAD 99 150 1011101 LDA ADR, Y CLOAD 99 150 10111101 LDA ADR, Y CLOAD 90 150 169 10111101 LDA ADR, Y CLOAD 90 169 169 10111101 LDA ADR, Y CLOAD	hexa	dec	binaire	ASCII	code op 6502	mot-cles
91 145 10010001 STA (adr),Y DATA 92 146 10010010 DIM 93 147 10010011 DIM 94 148 10010100 STY adr,X CLS 95 149 10010110 STA adr,X READ 96 150 10010111 GOTO 97 151 10010111 GOTO 98 152 10011001 STA ADR,Y LET 98 153 10011001 STA ADR,Y IF 98 154 1001101 STA ADR,Y IF 98 155 10011011 GOSUB 99 153 10011011 GOSUB 99 153 10011011 GOSUB 90 157 10011110 STA ADR,X REMINEN 90 157 10011110 STA ADR,X REMINEN 90 157 10011111 GOSUB 91 159 10011111 GRAB 92 159 10011111 GRAB 94 150 1000000 LDY >donnée RELEASE 95 150 1000000 LDY >donnée RELEASE 96 150 1000001 LDA (adr,X) TEXT 97 150 1000011 LDA (adr,X) TEXT 98 150 1000011 LDA adr EXPLODE 97 150 10001110 LDX adr EXPLODE 98 150 10100101 LDA adr SOUND 99 157 10001110 LDA ADR CIRCLE 96 156 10100101 LDA ADR CIRCLE 97 159 10011110 LDA ADR DRAW 98 150 1010101 LDA ADR DRAW 99 150 1010101 LDA ADR DRAW 99 150 1010101 LDA ADR DRAW 90 157 10101101 LDA ADR DRAW 90 157 10101101 LDA ADR CIRCLE 96 150 1010000 BCS CHAR 97 169 1010001 LDA ADR DRAW 97 169 1010001 LDA ADR DRAW 97 169 1010001 LDA ADR CIRCLE 97 150 1010111 GRAB 98 151 10110001 LDA ADR, Y PAPER 98 150 1011101 LDA ADR, Y PAPER 98 150 1011101 LDA ADR, Y CLOAD 99 150 1011101 LDA ADR, Y CLOAD 99 150 1011101 LDA ADR, Y CLOAD 99 150 10111101 LDA ADR, Y CLOAD 90 150 169 10111101 LDA ADR, Y CLOAD 90 169 169 10111101 LDA ADR, Y CLOAD	90	144	10010000		BCC	NEVT
92 146 10010010 INPUT 93 147 10010011 DIM 94 148 10010100 STY adr,X CLS 95 149 10010101 STA adr,X READ 96 150 1001010 STX adr,Y LET 97 151 10010111 GOTO 98 152 10011000 TYA RUN 99 153 10011001 STA ADR,Y IF 9A 154 10011010 TXS RESTORE 9B 155 10011011 GOSUB 9P 155 10011011 GOSUB 9P 157 10011011 STA ADR,X REM 9D 157 10011101 STA ADR,X REM 9F 158 10011101 STA ADR,X REM 9F 159 10011111 GRAB A0 160 10100000 LDY >donnée RELEASE A1 161 10100001 LDA (adr,X) TEXT A2 162 10100010 LDX >donnée HIRES A3 163 10100011 SHOOT A4 164 10100100 LDY adr EXPLODE A5 165 10100101 LDA adr ZAP A6 166 10100101 LDA adr ZAP A6 166 10100100 TAY A7 167 1010101 STA ADR,X REM A8 168 10101000 TAY A9 169 10101000 TAY A9 169 10101001 LDA ADR PARW A9 169 10101001 LDA ADR CIRCLE A7 167 1010101 STA ADR PATTERN A8 171 1010101 STA ADR ADR CIRCLE AB 171 1010111 SOUND AD 173 10101101 LDA ADR CIRCLE AF 174 10101111 STAP AF 175 10110000 BCS CHAR B1 177 10110001 LDA ADR PATTERN AF 175 1011011 LDA ADR PATTERN B1 177 10110001 LDA ADR PATTERN B1 177 10110001 LDA ADR, Y PAPER B1 180 10110100 LDY ADR, X CLCA B1 179 1011001 LDA ADR, Y PAPER B1 180 10110101 LDA ADR, Y CLOAD B9 185 1011101 STOP B8 184 1011100 LDA ADR, X CLEAR BB 196 1011110 LDA ADR, X CLEAR BB 197 1011111 CONT						
93 147 10010011						
94 148 10010100 STY adr,X CLS 95 149 10010101 STA adr,X READ 96 150 10010110 STA adr,Y LET 97 151 10010111 GOTO 98 152 10011000 TYA 99 153 10011001 STA ADR,Y IF 98 155 10011010 TXS RESTORE 98 155 10011011 RETURN 98 155 10011011 RETURN 99 157 10011101 STA ADR,X REM 90 157 10011101 STA ADR,X REM 90 158 10011111 GRAB 90 160 10100000 LDY >donnée RELEASE 150 10100010 LDA (adr,X) TEXT 161 10100010 LDA (adr,X) TEXT 162 10100011 SHOOT 164 10101000 LDY adr EXPLODE 165 10100101 LDA adr ZAP 166 10100101 LDA adr ZAP 167 1010111 SOUND 168 168 10101000 TAY 169 1010100 TAY 169 10101001 LDA >donnée PLAY 161 1010100 TAX 170 1010101 TAX 171 1010101 LDA ADR 171 1010101 LDA ADR 171 1010101 LDA ADR 172 10101101 LDA ADR 173 10101101 LDA ADR 174 10101101 LDA ADR 175 10110010 LDY ADR 176 10110010 BCS 177 10110001 LDA ADR 178 10101101 LDA ADR 179 10110010 BCS 179 10110010 LDA ADR 170 10110010 BCS 170 10110010 BCS 171 101010010 BCS 172 10110010 BCS 173 10101101 LDA ADR 174 10101001 LDA ADR 175 10110010 BCS 176 10110010 BCS 177 10110001 LDA ADR 178 10110010 BCS 179 10110010 LDA ADR,Y 180 181 1011010 LDA ADR,Y 180 181 1011010 LDA ADR,Y 181 180 1011010 LDA ADR,Y 181 181 1011010 TX 182 183 10110011 LDA ADR,Y 183 184 1011100 LDA ADR,X 184 186 1011100 LDA ADR,X 185 186 10111101						
95 149 10010101 STA adr,X READ 96 150 10010100 STX adr,Y LET 97 151 10010101 GOTO 98 152 10011000 TYA RUN 99 153 10011001 STA ADR,Y IF 9A 154 10011010 TXS RESTORE 9B 155 10011010 RETURN 9B 155 10011101 GOSUB 9C 156 100111101 GOSUB 9C 157 100111101 STA ADR,X REM 9E 158 10011110 GRAB 9F 159 10011111 GRAB 160 10100000 LDY >donnée RELEASE A1 161 10100001 LDA (adr,X) TEXT A2 162 10100010 LDX >donnée HIRES A3 163 10100011 LDA adr EXPLODE A4 164 10100100 LDY adr EXPLODE A5 165 10100110 LDA adr PING A7 167 10100111 SOUND A6 168 10101000 TAY MUSIC A9 169 10101001 TAY MUSIC A9 169 10101001 TAY MUSIC A9 169 10101001 TAX CURSET AB 171 10101011 GURMOV AD 173 10101101 LDA ADR CIRCLE AB 174 1010111 GURMOV AD 175 10101101 LDA ADR PATTERN AF 175 10101101 LDA ADR PATTERN AF 175 10101101 LDA ADR PATTERN B1 177 10110001 LDA Adr,X DN B1 177 10110010 BCS CHAR B1 177 10110010 BCS CHAR B1 177 10110010 LDY adr,X DN B5 181 1011010 LDX adr,Y PAPER B1 184 10111000 LDY adr,Y CLOAD B7 185 10111010 LDX ADR,Y POKE B8 184 10111000 LDY ADR,Y CLOAD B0 189 10111101 LDA ADR,Y CLEAR BE 190 10111101 LDA ADR,X CLEAR BE 190 10111101 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,Y GET					STV adr Y	
96 150 10010110 STX adr,Y LET 97 151 10010111 GOTO 98 152 10011000 TYA RUN 99 153 10011001 STA ADR,Y IF 9A 154 10011010 TXS RESTORE 9B 155 10011010 TXS RESTORE 9C 156 10011101 STA ADR,X REM 9C 156 10011101 STA ADR,X REM 9C 157 10011101 STA ADR,X REM 9D 157 10011111 GRAB AD 160 10100000 LDY donnée RELEASE AD 160 10100000 LDY donnée RELEASE AD 160 10100000 LDY donnée RELEASE AD 160 10100001 LDA (adr,X) TEXT AD 162 10100010 LDX donnée HIRES AD 163 10100010 LDX donnée HIRES AD 164 10100010 LDX donnée HIRES AD 165 10100101 LDA adr ZAP AD 166 10100101 LDA adr ZAP AD 166 10100101 LDA adr SOUND AD 168 10101000 TAY MUSIC AP 169 10101010 TAY CURSET AD 169 1010101 LDA ADR DRAW AD 173 10101011 CDA ADR CIRCLE AD 174 1010111 CDA ADR CIRCLE AD 175 1010111 CDA ADR CIRCLE AD 176 1011001 LDA ADR CIRCLE AD 177 1011001 LDA ADR CIRCLE AD 178 1011001 LDA ADR CIRCLE AD 179 1011001 LDA ADR CIRCLE BI 177 10110001 LDA ADR CIRCLE BI 177 10110001 LDA ADR CIRCLE BI 177 10110001 LDA ADR CIRCLE BI 178 10110010 CDA ADR CIRCLE BI 179 10110011 LDA ADR, WAIT BE 180 1011010 LDA ADR, Y CLOAD BT 183 1011011 LDA ADR, Y CLOAD BT 183 10111011 LDA ADR, Y POKE BP 185 10111001 LDA ADR, Y CLOAD BD 189 10111101 LDA ADR, Y CLOAD BD 189 10111101 LDA ADR, Y CLEAR BE 190 10111101 LDA ADR, Y CLEAR						
97 151 10010111						
98 152 10011000 TYA RUN 97 153 10011001 STA ADR,Y IF 9A 154 10011010 TXS RESTORE 9B 155 10011011 RETURN 9C 154 10011101 STA ADR,X RETURN 9C 156 10011101 STA ADR,X RETURN 9E 158 10011110 GRAB 160 10100000 LDY >donnée RELEASE 161 10100001 LDA (adr,X) TEXT AZ 162 10100010 LDY >donnée HIRES A3 163 10100011 SHOOT A4 164 10100100 LDY adr EXPLODE A5 165 10100101 LDA adr ZAP A6 166 10100101 LDA adr ZAP A7 167 10100111 SOUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 1010101 LDA ADR AA 171 1010101 TAX CURSET AB 171 1010101 LDA ADR CIRCLE AB 171 1010110 LDA ADR CIRCLE AB 174 1010110 LDA ADR CIRCLE AF 175 10101111 FILL B0 176 1011001 LDA ADR PATTERN AF 175 10101111 FILL B0 176 1011001 LDA ADR CIRCLE B1 177 10110001 LDA ADR PATTERN AF 175 10101111 FILL B0 176 1011001 LDA ADR CIRCLE B1 177 10110001 LDA ADR CIRCLE B2 178 10110010 LDA ADR, Y PAPER B3 179 10110011 LDA ADR, Y PAPER B4 184 10111010 LDA ADR, Y POKE B5 181 1011010 LDA ADR, Y POKE B6 184 10111010 LDA ADR, X CLEAR B8 184 10111010 LDA ADR, X CLEAR B8 187 10111101 LDA ADR, X CLEAR B8 190 10111110 LDA ADR, Y CLEAR B9 187 10111101 LDA ADR, X CLEAR B0 189 10111110 LDA ADR, Y CLEAR						
97 153 10011001					TYA	
9A 154 10011010 TXS RESTORE 9B 155 10011011 GOSUB 9C 156 10011100 RETURN 9D 157 10011101 STA ADR, X REM 9F 158 10011110 GRAB A0 160 10100000 LDY >donnée RELEASE A1 161 10100001 LDA (adr, X) TEXT A2 162 10100010 LDY >donnée HIRES A3 163 10100011 SHOOT A4 164 10100100 LDY adr EXPLODE A5 165 10100101 LDA adr ZAP A6 166 10100100 LDX adr PING A7 167 10100111 SGUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 1010101 LDA >donnée PLAY AA 170 1010101 LDA >donnée PLAY AB 171 1010101 LDA ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 1010110 LDX ADR PATTERN AF 175 10101101 LDA ADR CIRCLE AE 174 1010110 BCS B1 177 10110000 BCS B1 177 10110001 LDA Adr, X DN B5 181 10110100 LDY ADR DRAW B1 177 10110001 LDA ADR CIRCLE B3 179 10110101 LDA ADR, Y PAPER B4 180 10110100 LDY ADR, X WAIT B6 182 10110101 LDA ADR, Y POKE B9 185 10111001 LDA ADR, Y POKE B9 185 10111001 LDA ADR, Y POKE B9 187 1011101 LDA ADR, X CLEAR BB 184 10111000 LDY ADR, X LIST BB 187 10111101 LDA ADR, X CLEAR BD 189 10111101 LDA ADR, X CLEAR BD 189 10111110 LDA ADR, X CLEAR BD 189 10111110 LDA ADR, X CLEAR BD 189 10111110 LDA ADR, Y CLOAD BC 188 10111110 LDA ADR, X CLEAR BD 189 10111110 LDA ADR, Y CLOAD BC 188 10111110 LDA ADR, Y CLOAD BC 188 10111110 LDA ADR, X CLEAR						
9B 155 10011011						
9C 156 10011100 RETURN 9D 157 10011101 STA ADR, X REM 9E 158 10011110 HIMEM 9F 159 10011111 GRAB A0 160 10100000 LDY >donnée RELEASE A1 161 10100001 LDA (adr, X) TEXT A2 162 10100010 LDX >donnée HIRES A3 163 10100011 SHOOT A4 164 10100100 LDY adr EXPLODE A5 165 10100101 LDA adr ZAP A6 166 10100110 LDX adr PING A7 167 10100111 SOUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 1010101 LDA >donnée PLAY AB 171 10101010 TAX CURSET AB 171 10101100 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDA ADR CIRCLE AE 175 10101110 LDA ADR PATTERN AF 175 10101111 FILL B0 176 10110000 BCS CHAR B1 177 10110010 LDA (adr),Y PAPER B2 178 10110010 LDA adr,X WAIT B3 179 10110110 LDA ADR,X WAIT B6 182 1011010 LDA ADR,Y CLOAD B7 183 1011011 LDA ADR,Y POKE BA 186 10111010 CLY ADR,Y GET						
9D 157 10011101 STA ADR, X REM 9E 158 10011110 HIMEM 9F 159 10011111 GRAB A0 160 10100000 LDY >donnée RELEASE A1 161 10100001 LDA (adr, X) TEXT A2 162 10100010 LDX >donnée HIRES A3 163 10100010 LDX >donnée HIRES A5 165 10100101 SHOOT A4 164 10100100 LDY adr EXPLODE A5 165 10100101 LDA adr ZAP A6 166 10100110 LDX adr PING A7 167 10100111 SOUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 10101101 TAX CURSET AB 171 10101011 TAX CURSET AB 171 10101101 LDA ADR DRAW AD 173 10101101 LDA ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101111 FILL B0 176 10110001 LDA ADR PATTERN AF 175 10101111 FILL B0 176 10110001 LDA (adr),Y PAPER B1 177 10110001 LDA (adr),Y PAPER B2 178 101101101 LDA ADR,X UNIT B3 179 10110110 LDA ADR,X UNIT B4 180 10110100 LDY ADR,Y POKE B9 185 10111010 LDA ADR,Y POKE BA 186 10111010 LDA ADR,Y POKE BA 186 10111010 LDA ADR,X CLEAR BE 190 1011110 LDA ADR,X CLEAR BD 187 10111110 LDA ADR,X CLEAR BD 189 10111110 LDA ADR,X CLEAR BD 189 10111110 LDA ADR,X CLEAR BD 189 10111110 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,Y GET						
9E 158 10011110 HIMEM 9F 159 10011111 GRAB A0 160 10100000 LDY >donnée RELEASE A1 161 10100010 LDX >donnée HIRES A2 162 10100011 SHOOT A3 163 1010001 LDX adr HIRES A3 163 1010010 LDX adr PING A5 165 1010010 LDX adr PING A5 165 1010010 LDX adr PING A6 166 1010010 LDX adr PING A7 167 1010001 LDX adr PING A7 167 1010000 TAY MUSIC MUSIC A7 167 1010000 TAY MUSIC CURST A8 171 10101001 TAX CURSET CURSET					STA ADR.X	
9F 159 10011111						
A0 160 10100000 LDY >donnée RELEASE A1 161 10100001 LDA (adr,X) TEXT A2 162 10100010 LDX >donnée HIRES A3 163 10100011 SHOOT A4 164 10100100 LDY adr EXPLODE A5 165 10100101 LDA adr ZAP A6 166 10100110 LDX adr PING A7 167 10100111 SOUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 10101010 TAX CURSET AB 171 10101010 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL B0 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110101 LDA (adr),Y PAPER B2 178 10110101 LDA adr,X WAIT B6 182 10110101 LDA adr,X WAIT B6 182 10110110 LDA ADR,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111010 LDA ADR,Y PRINT BB 187 10111011 CONT BC 188 10111101 LDA ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BB 190 10111110 LDA ADR,X CLEAR						
A1 161 10100001					LDY >donnée	
A2 162 10100010 LDX >donnée HIRES A3 163 10100101 SHOOT A4 164 10100100 LDY adr EXPLODE A5 165 10100101 LDA adr ZAP A6 166 10100110 LDX adr PING A7 167 10100111 SOUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 10101010 TAX CURSET AB 171 10101011 CURMOV AC 172 10101100 LDY ADR DRAW ADD 173 10101101 LDA ADR CIRCLE AE 174 10101111 FILL B0 176 10101010 LDX ADR PATTERN AF 175 10101111 FILL B0 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B3 179 10110010 INK B3 179 10110010 LDY ADR DX CIRCLE B4 180 10110100 LDY ADR,X WAIT B6 182 10110110 LDA ADR,X CLOAD B7 183 10110111 STOP B8 184 10110101 LDA ADR,Y POKE BA 186 10111010 LDA ADR,Y POKE BA 186 10111010 LDA ADR,Y POKE BA 186 10111010 LDA ADR,Y POKE BA 186 10111100 LDA ADR,X LIST BD 187 10111011 LDA ADR,X CLEAR BE 190 10111101 LDA ADR,X CLEAR BE 190 101111101 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,Y GET						
A3 163 10100011			10100010			
A4 164 10100100 LDY adr EXPLODE A5 165 10100101 LDA adr ZAP A6 166 10100110 LDX adr PING A7 167 10100101 LDX adr PING A7 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 10101010 TAX CURSET AB 171 10101011 CURMOV AC 172 10101100 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL B0 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110010 INK B3 179 10110011 LDA adr,X WAIT B4 180 10110100 LDY adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111101 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,Y GET	A3					
A5 165 10100101 LDA adr ZAP A6 166 10100110 LDX adr PING A7 167 10100111 SOUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 10101010 TAX CURSET AB 171 10101011 CURMOV AC 172 10101100 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL B0 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110101 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 LDA adr,X WAIT B4 180 10110100 LDX adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111011 LDA ADR,Y POKE BA 186 10111010 LDA ADR,Y POKE BA 186 10111010 LDA ADR,Y PRINT BB 187 1011101 LDA ADR,X LIST BD 189 1011110 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,Y GET					LDY adr	
A6 166 10100110 LDX adr PING A7 167 10100111 SOUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY CURSET AB 171 10101010 TAX CURSET AB 171 10101010 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL BØ 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110101 LDA (adr),Y PAPER B3 179 10110101 LDA adr,X WAIT B6 182 10110101 LDA adr,X WAIT B6 182 10110101 LDA adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111101 LDA ADR,Y GET BD 189 10111101 LDA ADR,Y GET	A5	165	10100101			
A7 167 10100111 SOUND A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 10101010 TAX CURSET AB 171 10101011 CURMOV AC 172 10101100 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL BØ 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110101 LDA adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX BB 187 1011101 CONT BC 188 10111101 LDA ADR,Y GET	A6	166	10100110			
A8 168 10101000 TAY MUSIC A9 169 10101001 LDA >donnée PLAY AA 170 10101010 TAX CURSET AB 171 10101011 CURMOV AC 172 10101100 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL BØ 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 1011101 LDA ADR,X CLEAR BE 190 1011110 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,Y GET	A7	167	10100111			
A9 169 10101001 LDA >donnée PLAY AA 170 10101010 TAX CURSET AB 171 10101011 CURMOV AC 172 10101100 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL BØ 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDA adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 1011101 LDA ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,X CLEAR	AB	168	10101000		TAY	
AA 170 10101010 TAX CURSET AB 171 10101011 CURMOV AC 172 10101100 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL BØ 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110010 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110101 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y PRINT BB 187 10111011 CONT BC 188 10111101 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111101 LDX ADR,Y GET	A9	169	10101001		LDA >donnée	
AC 172 10101100 LDY ADR DRAW AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL BØ 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111101 LDA ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET	AA	170	10101010		TAX	
AD 173 10101101 LDA ADR CIRCLE AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL B0 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111101 LDA ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDA ADR,Y GET	AB	171	10101011			CURMOV
AE 174 10101110 LDX ADR PATTERN AF 175 10101111 FILL B0 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 1011100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111101 LDA ADR,Y GET	AC	172	10101100		LDY ADR	DRAW
AF 175 10101111 FILL B0 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,Y GET	AD	173	10101101		LDA ADR	CIRCLE
BØ 176 10110000 BCS CHAR B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 — INK B3 179 10110011 — STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 — CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111011 — CONT BB 187 10111011 — CONT BC 188 10111101 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET		174	10101110		LDX ADR	PATTERN
B1 177 10110001 LDA (adr),Y PAPER B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET			10101111			FILL
B2 178 10110010 INK B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT DB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET						CHAR
B3 179 10110011 STOP B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET					LDA (adr),Y	PAPER
B4 180 10110100 LDY adr,X ON B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET						INK
B5 181 10110101 LDA adr,X WAIT B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET						STOP
B6 182 10110110 LDX adr,Y CLOAD B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT DB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET			10110100			ON
B7 183 10110111 CSAVE B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET			10110101			WAIT
B8 184 10111000 CLV DEF B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET			10110110		LDX adr,Y	CLOAD
B9 185 10111001 LDA ADR,Y POKE BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET			10110111			CSAVE
BA 186 10111010 TSX PRINT BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET					CLV	DEF
BB 187 10111011 CONT BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET					the state of the s	POKE
BC 188 10111100 LDY ADR,X LIST BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET						
BD 189 10111101 LDA ADR,X CLEAR BE 190 10111110 LDX ADR,Y GET						
BE 190 10111110 LDX ADR,Y GET						
					[1] [1] (1) (1) (1) (1) (1) (1) (1) (1) (1) (1)	
BF 191 101111111 CALL					•	
	BF	191	10111111			CALL

Les codes 80-97 hexa sont affichés comme attributs par PRINT. Les caractères correspondant aux codes #A0-FE (ASCII+#80) sont affichés en inverse par PLOT. Voir Annexe 2.

hexa	déc	binaire	ASC11	code op 6502	mot-clés
CØ	192	11000000		CPY >donnée	!
Ci	193	11000001		CMP (adr,X)	NEW
C2	194	11000010			TAB (
C3	195	11000011			TO
C4	196	11000100		CPY adr	FN
C5	197	11000101		CMP adr	SPC (
C6	198	11000110		DEC adr	@
C7	199	11000111			AUTO
CB	200	11001000		INY	ELSE
C9	201	11001001		CMP >donnée	THEN
CA	202	11001010		DEX	NOT
CB	203	11001011			STEP
CC	204	11001100		CPY ADR	+
CD	205	11001101		CMP ADR	_
CE	206	11001110		DEC ADR	*
CF	207	11001111			1
DØ	208	11010000		BNE	^
D1	209	11010001		CMP (adr),Y	AND
D2	210	11010010			OR
D3	211	11010011			>
D4	212	11010100			=
D5	213	11010101		CMP adr, X	<
Ď6	214	11010110		DEC adr, X	SGN
D7	215	11010111			INT
D8	216	11011000		CLD	ABS
D9	217	11011001		CMP ADR,Y	USR
DA	218	11011010			FRE
DB	219	11011011			POS
DC	220	11011100			HEX\$
DD	221	11011101		CMP ADR,X	&
DE	222	11011110		DEC ADR,X	SQR
DF	223	11011111			RND
EØ	224	11100000		CPX >donnée	LN
E1	225	11100001		SBC (adr, X)	EXP
E2	226	11100010			COS
E3	227	11100011			SIN
E4	228	11100100		CPX adr	TAN
E5	229	11100101		SBC adr	ATN
E6	230	11100110		INC adr	PEEK
E7	231	11100111			DEEK
E8	232	11101000		INX	LOG
E9	233	11101001		SBC >donnée	LEN
EA	234	11101010		NOP	STR\$
EB	235	11101011			VAL
EC	236	11101100		CPX ADR	ASC
ED	237	11101101		SBC ADR	CHR\$
EE	238	11101110		INC ADR	PI
EF	239	11101111			TRUE

Les caractères correspondant aux codes #AØ-FE (ASCII+#8Ø) sont affiches en inverse par PLOT. Voir Annexe 2.

hexa	dec	binaire	ASCII	code op 6502	mot-cles
FØ	240	11110000		BEQ	FALSE
F1	241	11110001		SBC (adr),Y	KEY\$
F2	242	11110010			SCRN
F3	243	11110011			POINT
F4	244	11110100			LEFT\$
F5	245	11110101		SBC adr,X	RIGHT\$
F6	246	11110110		INC adr, X	MID\$
F7	247	11110111			
F8	248	11111000		SED	
F9	249	11111001		SBC ADR,Y	
FA	250	11111010			
FB	251	11111011			
FC	252	11111100			
FD	253	11111101		SBC ADR,X	
FE	254	11111110		INC ADR,X	
FF	255	11111111		-	

Les caractères correspondant aux codes #A0-FE (ASCII+#80) sont affichés en inverse par PLOT. Voir Annexe 2.

ANNEXE 2 - Caractères de controle Attributs, codes ESCAPE

Les 32 premiers codes ASCII sont utilisés de deux façons différentes par l'ORIC.

Ils peuvent représenter des caractères de contrôle classiques, activés par une instruction de la forme PRINT CHR\$(), ou devenir attributs lorsqu'ils sont inscrits directement dans la mémoire d'écran par POKE, PLOT ou par l'intermédiaire de la fonction Escape.

Un attribut modifie les conditions d'affichage des caractères qui le suivent sur la ligne.

hexa	déc	contrôle	attribut	escape
ØØ	Ø		encre noire	ESC 🙊
Ø1	1	CTRL-A copie	encre rouge	ESC A
02	2		encre verte	ESC B
Ø 3	3	CTRL-C break	encre jaune	ESC C
Ø4	4	CTRL-D double ligne	encre bleue	ESC D
Ø 5	5		encre mauve	ESC E
ወሪ	6	CTRL-F bruit clavier	encre ciel	ESC F
07	7	CTRL-6 ping	encre blanche	ESC G
08	8.	CTRL-H curseur <-	carac normaux	ESC H
09	9	CTRL-I curseur ->	carac graphiques	ESC I
ØA	10	CTRL-J curseur 🍑	double hauteur norm	
ØB	11	CTRL-K curseur T	double hauteur graph	
ØC	12	CTRL-L efface écran	norm clignotant	ESC L
ØD	13	CTRL-M return	graph clignotant	ESC M
ØE	14	CTRL-N efface ligne	norm DH clign	ESC N
ØF	15	CTRL-O inhibe écran	graph DH clign	ESC O
10	16		papier noir	ESC P
11	17	CTRL-Q curseur	papier rouge	ESC Q
12	18		papier vert	ESC R
13	19	CTRL-S écran	papier jaune	ESC S
14	20	CTRL-T maj/min	papier bleu	ESC T
15	21		papier mauve	ESC U
16	22		papier ciel	ESC V
17	23		papier blanc	ESC W
18	24	CTRL-X annule ligne		
19	25			
1A	26			
1B	27	ESCAPE		
10	28			
1 D	29	CTRL-3 40 colonnes		

La configuration binaire des codes de contrôle diffère de 64 décimal de celle des lettres 0, A, B et la suite (voir pages 203, 204). Le bit 6 est à zéro pour les uns, à un pour les autres.

En mode direct, l'association avec CTRL a pour effet de mettre à zéro les bits 5 et 6 du code ASCII de la lettre concernée et d'en faire un caractère de contrôle.

Tapé au clavier ou inclus dans une instruction PRINT, ESC agit de façon analogue mais au niveau suivant.

Disons, pour simplifier, que la fonction du caractère de controle ESC est de fabriquer un attribut avec le code de la lettre qui le suit.

Vous pouvez suivre le processus de traitement des caractères de contrôle, et notamment de ESC, en examinant la routine VDU en #F77C pour l'ATMOS, #F73F pour l'ORIC-1 qui assure l'affichage effectif au profit de OUTDO.

A l'entrée le caractère à afficher est en X. Après la sauvegarde des registres il se retrouve en A. Si un caractère de contrôle est reconnu, #F799(A), #F748(O), il n'est pas envoyé à l'écran mais à une routine spéciale en #F602(A), #F5D3(O).

Le saut indirect JMP(#261) renvoie à une sous-routine spécifique dont l'adresse aura été trouvée dans la table en #F5E2(A), #F5B3(O). L'action de contrôle appropriée est exécutée, déplacement curseur, effacement, positionnement d'un drapeau, bascule, etc. Quand ESC est traité par cette routine, le bit n°4 de la variable système #26A est mis à un.

Si le code du caractère normal suivant est supérieur ou égal à #40, test par SBC #40 en #F7A4 (A) #F75A(O), l'instruction AND #1F en fait un attribut, sinon il est remolacé par un espace.

un attribut, sinon il est remplacé par un espace. Au retour de l'affichage, le code d'ESC (#1B) est rechargé dans A et la sortie s'effectue par la routine de traitement des caractères de contrôle qui remet #26A dans son état initial (bascule).

Remarque: Les bits 5 et 6 (et 7) étant mis à zéro par AND #1F, ESC a, ESC b, etc conviennent aussi bien que ESC A, ESC B pour former des attributs.

La signification des différents bits du code d'un caractère est rappelée ci-dessous:

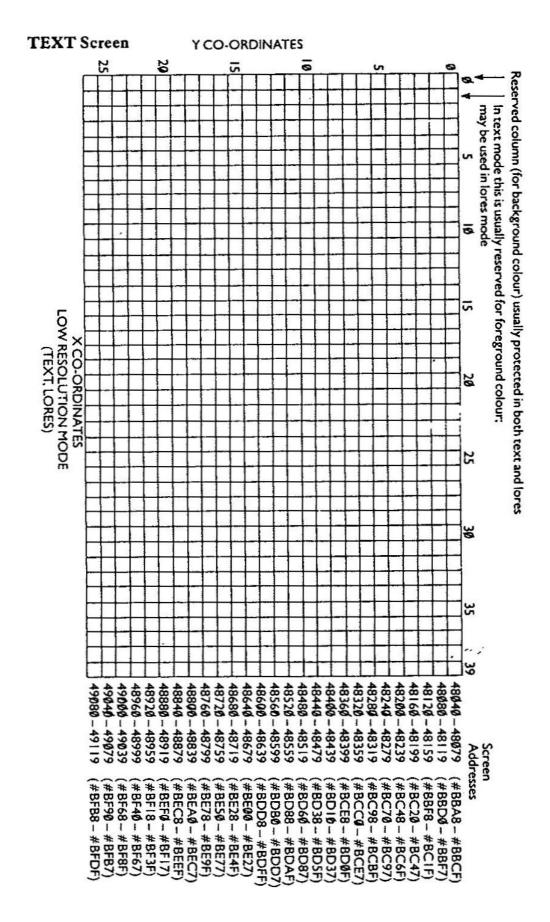
Un caractère en ASCII négatif (bit 7 à un) est affiché en vidéo inverse s'il est envoyé directement dans la mémoire d'écran par POKE ou PLOT.

La routine d'affichage, en #F7E4(A) #F7AC(O), ignore le bit 7, et affiche le caractère en normal.

En fait, il semble que l'opération se fasse en deux temps, le caractère étant d'abord inscrit tel quel dans la mémoire d'écran. Il est modifié ensuite par la sous-routine de déplacement du curseur conformément aux attributs des deux premières colonnes. Le caractère est recueilli dans A, le bit 7 est mis à 0 par AND #7F, puis il est remis en place.

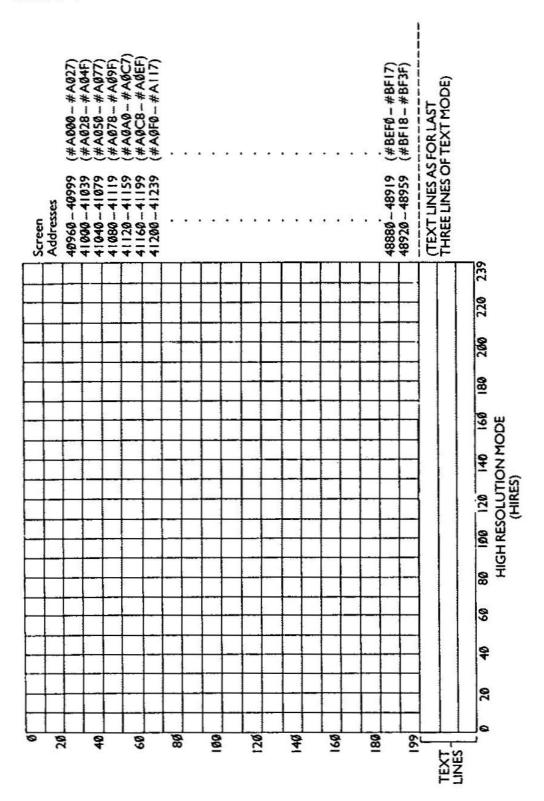
Les codes compris entre #80 et #97 (128 et 151 décimal) étant supérieurs à #1F ne sont pas reconnus comme caractères de contrôle (comparaison en #F797 ATMOS; #F746 ORIC-1). Ils deviennent des attributs authentiques en passant par la routine d'affichage.

Avec POKE et PLOT, le bit 7 conserve évidemment sa valeur et joue son rôle pour l'affichage éventuel en vidéo inversée.





HIRES Screen



ANNEXE 5 - Instructions 6502

ADC	Add Memory to Accumulator with	LDA	Load Accumulator with Memory
	Carry	LDX	Load Index X with Memory
AND	"AND" Memory with Accumulator	LDY	Load Index Y with Memory
ASL	Shift Left One Bit (Memory or	LSR.	Shift Right one Bit (Memory or
	Accumulator)		Accumulatori
BCC	Branch on Carry Clear	NOP	No Operation
BCS	Branch on Carry Set	ORA	"OR" Memory with Accumulator
BEQ	Branch on Result Zero	PHA	Push Accumulator on Stack
BIT	Test Bits in Memory with	PHP	Push Processor Status on Stack
	Accumulator	PLA	Pull Accumulator from Stack
BMI	Branch on Result Minus	PLP	Pull Processor Status from Stack
BNE	Branch on Result not Zero	5 378	
BPL	Branch on Result Plus	ROL	Rotate One Bit Left (Memory or
BRK	Force Break		Accumulator)
BVC	Branch on Overflow Clear	ROR	Rotate One Bit Right (Memory or
BVS	Branch on Overflow Set		Accumulator)
CLC	Clear Carry Flag	RTI	Return from Interrupt
CLD	Clear Decimal Mode	RTS	Return from Subroutine
CLI	Clear Interrupt Disable Bit	SBC	Subtract Memory from Accumulator
CLV	Clear Overflow Flag		with Borrow
CMP	Compare Memory and Accumulator	SEC	Set Carry Flag
CPX	Compare Memory and Index X	SED	Set Decimal Mode
CPY	Compare Memory and Index Y	SEI	Set Interrupt Disable Status
DEC	Decrement Memory by One	STA	Store Accumulator in Memory
DEX	Decrement Index X by One	STX	Store Index X in Memory
DEY	Decrement Index Y by One	STY	Store Index Y in Memory
EOR	"Exclusive-Or" Memory with	TAX	Transfer Accumulator to Index X
	Accumulator	TAY	Transfer Accumulator to Index Y
INC	Ingrament Mamon; by One	TSX	Transfer Stack Pointer to Index X
	Increment Memory by One	TXA	Transfer Index X to Accumulator
INX	Increment Index X by One	TXS	Transfer Index X to Stack Pointer
INY	increment Index Y by One	TYA	Transfer Index Y to Accumulator
JMP	Jump to New Location		
JSR	Jump to New Location Saving		

Return Address

LES NOTATIONS SUIVANTES SERONT UTILISÉES DANS CE RÉCAPITULATIF

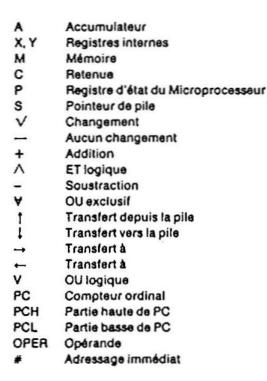


Figure 1 : ASL (Décalage à gauche)

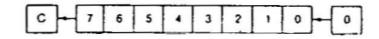


Figure 2 : ROL (Rotation à gauche)

peut porter sur l'accumulateur ou une mémoire

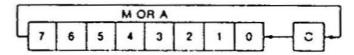
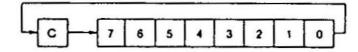


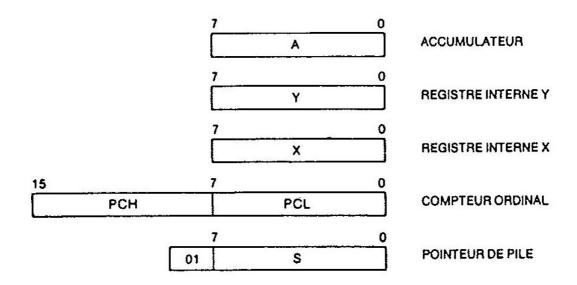
Figure 3 ROR (Rotation a droite)

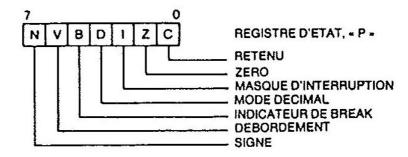
peut porter sur l'accumulateur ou une mémoire



Note: BiT (Test de bits en Mémoire): Les bits 6 et 7 sont transférés dans le registre d'état. Si le résultat de A/M est 0 alors Z=1, sinon Z=0.

MODÈLE DE PROGRAMMATION





CODES DES INSTRUCTIONS

Nom Description	Opération	Mode d'adressage	Ecriture Assembleur	Code	Nbre d octets	Registre P N Z C I D V
ADC Ajoute la mémoire à l'accumulateur avec retenue	A+M+C→A,C	Immédiat Page Zéro Page Zéro, X Absolu Absolu, X (Indirect, X) (Indirect), Y Absolu, Y	ADC #Oper ADC Oper, X ADC Oper ADC Oper, X ADC Oper, X ADC (Oper, X) ADC (Oper), Y ADC Oper, Y	69 65 75 6D 7D 61 71 79	2 2 2 3 3 2 2 3	VVVV
AND ET logique entre accumulalteur et donnée	A'M→A	Immédiat Page Zézo Page Zéro, X Absolu Absolu, X Absolu, Y (Indirect, X) (Indirect), Y	ADC #Oper ADC Oper, X ADC Oper, X ADC Oper, X ADC Oper, X ADC Oper, Y ADC (Oper, X) ADC (Oper), Y	29 25 35 2D 3D 39 21 31	2 2 2 3 3 3 2 2	· VV
ASL Décalage à gauche (mémoire ou accumulateur)	Voir figure 1	Accumulateur Page Zéro Page Zéro, X Absolu Absolu, X	ADC Oper ADC Oper ADC Oper, X ADC Oper ADC Oper, X	0A 06 16 0E 1E	1 2 2 3 3	VVV
BCC Branche si pas de retenue	Branche si C=0	Relatif	BCC Oper	90	2	
BCS Branche si retenue	Branche si C=1	Relatif	BCS Oper	В0	2	
BEQ Branche si résultat nul	Branche și Z=1	Relatif	BEQ Oper	FO	2	
BIT Compare l'accumulateur et une mémoire*	A \M, M7 → N M6 → V	Page Zéro Absolu	BIT Oper BIT Oper	24 2C	2 3	M7√M6
BMI Branche si résultat négatif	Branche si N=1	Relaitf	BMI Oper	30	2	
BNE Branche si résultat non nul	Branche si Z=0	Relatif	BNE Oper	D0	2	
BPL Branche si résultat positif	Branche si N≠0	Refatif	BPL Oper	10	2	
BRK** Interruption programmée	PC+2 P	Implicite	BRK	00	1	1
BVC Branche si pas de débordement	Branche si V=0	Relatif	BVC Oper	50	2	

^{&#}x27;Si A et M sont égaux Z=1 sinon Z=0.

[&]quot;Un BRK ne peut être masqué en positionnant l'à 1.

Nom Description	Opération	Mode d'adressage	Ecriture Assembleur	Code	Nbre d'octets	Registre P N Z C I D V
BVS Branche si débordement	8ranche si V = 1	Relatif	BVS Oper	70	2	* 5 *
CLC Mise à zéro de la retenue	0 C	Implicite	cic	18	1	. 0 -
CLD Mise à zéro du mode décimal	0 - - 0	Implicite	CLD	08	1	0
CLI Mise à zéro du masque d'in terruption	0-1	Implicite	CLI	58	1	- 0
CLV Mise à zéro de l'indicateur de débordement	0 - - V	Implicite	CLV	88	1	0
CMP Compare accumulateur et mémoire	A — M	Immédiat Page Zéro Page Zéro, X Absolu Absolu, X Absolu, Y (Indirect, X) (Indirect), Y	CMP #Oper CMP Oper,X CMP Oper,X CMP Oper,X CMP Oper,Y CMP (Oper,X) CMP (Oper),Y		2 2 2 3 3 3 2 2 2	V VV · ·
CPX Compare la mémoire et X	X M	Immédiat Page Zéro Absolu	CPX #Oper CPX Oper CPX Oper	E0 E4 EC	2 2 3	VV
CPY Compare la mémoire et Y	Y M	immédiat Page Zéro Absolu	CPY #Oper CPY Oper CPY Oper	C0 C4 CC	2 2 3	VVV
DEC Décrémente la mémoire de 1	M — 1 → M	Page Zéro Page Zéro, X Absolu Absolu, X	DEC Oper DEC Oper,X DEC Oper DEC Oper,X	C6 D6 CE DE	2 2 3 3 3	√√···
DEX Décrémente X de 1	X — 1 — X	Implicite	DEX	CA	1	/ /
DEY Décrémente Y de 1	Y − 1 → Y	Implicite	DEY	88	1	/ /

Nom Description	Opération	Mode d'adressage	Ecriture Assembleur	Code	Nbre d'octets	Registre P NZCIDV
EOR	-,				3 301613	1120104
OU exclusif entre mémoire et accumulateur	A V M A	Immédiat Page Zéro Page Zéro, X Absolu Absolu, X Absolu, Y (Indirect, X) (Indirect), Y	EOR *Oper EOR Oper EOR Oper,X EOR Oper,X EOR Oper,Y EOR (Oper,X) EOR (Oper,X)	49 45 55 40 50 59 41 51	2 2 2 3 3 3 2 2	✓✓
INC Incrémente de 1 la mémoire	M • 1 -+ M	Page Zéro Page Zéro, X Absolu Absolu, X	INC Oper INC Oper.X INC Oper INC Oper.X	E6 F6 EE FE	2 2 3 3	√√ -
INX Incrémente X de 1	X • 1 → X	Implicite	INX	E8	1	/ /
INY Incrémente Y de 1	Y • 1 Y	Implicite	INY	C8	1	/ /
JMP						
Saute à une adresse	(PC+1) - PCL (PC+2) - PCH	Absolu Indirect	JMP Oper JMP (Oper)	4C 6C	3	
JSR Saute à un sous-programme	PC+2 † (PC+1) PCL (PC+2) PCH	Absolu	JSR Oper	20	3	
LDA Charge l'accumulateur	M -+ A	Immédiat Page Zéro Page Zéro, X Absolu Absolu, X Absolu, Y (Indirect, X) (Indirect), Y	LDA #Oper LDA Oper LDA Oper,X LDA Oper LDA Oper,X LDA Oper,Y LDA (Oper,X) LDA (Oper),Y	A9 A5 B5 AD BD B9 A1 B1	2 2 2 3 3 3 2 2 2	V
LDX Charge le registre X	M X	Immédiat Page Zéro Page Zéro, Y Absolu Absolu, Y	LOX #Oper LOX Oper LOX Oper.Y LOX Oper LOX Oper,Y	A2 A6 B6 AE BE	2 2 2 2 3 3	√√
LDY Charge le registre Y	M -+Y	Immédiat Page Zéro Page Zéro, X Absolu Absolu, X	LDY #Oper LDY Oper LDY Oper,X LDY Oper LDY Oper,X	A0 A4 B4 AC BC	2 2 2 3 3	√√ -

Nom Description	Opération	Mode d'adressage	Ecriture Assembleur	Code	Nbre d'octets	Registre P NZCIDV
LSR Décalage à droite	(voir figure 1)	Accumulateur Page Zéro Page Zéro, X Absolu Absolu, X	LSR A LSR Oper LSR Oper,X LSR Oper LSR Oper,X	4A 46 56 4E 5E	1 2 2 3 3	0√√
NOP Aucune action		Implicite	NOP	EA	146	
ORA OU logique entre la donnée et l'accumulateur	AVM A	immédiat Page Zéro Page Zéro, X Absolu Absolu, X Absolu, Y (Indirect, X) (Indirect), Y	ORA #Oper ORA Oper ORA Oper ORA Oper ORA Oper.X ORA Oper.Y ORA (Oper.X) ORA (Oper).Y	09 05 15 00 10 19 01	2 2 2 3 3 2 2 2	√√
PHA Empile l'accumulateur	A	Implicite	РНА	48	1	
PHP Empile le registre d'état	P	Implicite	РНР	08	1	
PLA Dépile l'accumulateur	A f	Implicite	PLA	68	1	vv
PLP Dépile le registre d'état	P	Implicite	PLP	28	1	(vient de la pile)
Rotation à gauche	(See Figure 2)	Accumulateur Page Zéro Page Zéro, X Absolu Absolu, X	ROL A ROL Oper ROL Oper,X ROL Oper ROL Oper.X	2A 26 36 2E 3E	1 2 2 3 3	√√√ ·
ROR Rotation à droite	(See Figure 3)	Accumulateur Page Zéro Page Zéro, X Absolu Absolu, X	ROR A ROR Oper ROR Oper,X ROR Oper ROR Oper,X	6A 66 76 6E 7E	1 2 2 3 3	VVV

Nom Description	Opération	Mode d'adressage	Ecriture Assembleur	Code	Nbre d'octets	Registre P NZCIDV
RTI Retour d'interruption	PIPCI	Implicite	RTI	40	1	(vient de la pile)
RTS Relour de sous-programme	PCI. PC-1 PC	Implicite	-RTS	60	1	
SBC Soustraction avec emprunt	A · M · Ĉ → A	Immédiat Page Zéro Page Zéro, X Absolu Absolu, X Absolu, Y (Indirect, X) (Indirect), Y	SBC *Oper SBC Oper SBC Oper.X SBC Oper.X SBC Oper.X SBC (Oper.X) SBC (Oper.X)	E9 E5 F5 ED F0 F9 E1	2 2 2 3 3 3 2 2 2	V VVV
SEC Met la retenue à 1	1 +C	Implicite	SEC	38	1	1
SED Mise en mode décimal	1 -+ D	Implicite	SED	F8	1	1-
SEI Mise à 1 du masque d'interruption	1-+1	Implicite	SEI	78	1	1
STA Stocke l'accumulateur en memoire	A -+ M	Page Zéro Page Zéro, X Absolu Absolu, X Absolu, Y (Indirect, X) (Indirect), Y	STA Oper STA Oper.X STA Oper STA Oper.X STA Oper.Y STA (Oper.X) STA (Oper).Y	85 95 80 90 99 81 91	2 2 3 3 3 2 2 2	
STX Stocke X en mémoire	X M	Page Zéro Page Zéro, Y Absolu	STX Oper STX Oper,Y STX Oper	86 96 8E	2 2 3	
STY Stocke Y en mémoire	Y M	Page Zéro Page Zéro, X Absolu	STY Oper STY Oper.X STY Oper	84 94 8C	2 2 3	
TAX Transfert de A dans X	A X	Implicite	TAX	AA	1	VV
TAY Transfert de A dans Y	A Y	Implicite	TAY	A8	1	V
TSX Transfert de S dans X	5 X	Implicite	TSX	BA	1	VV

Nom Description	Opération	Mode d'adressage	Ecriture Assembleur	Code	Nbre d'octets	Registre P N Z C I D V
TXA Transfert de X dans A	X + A	Implicite	TXA	8A	1	/ / · · -
TXS Transfert de X dans S	x s	Implicite	TXS	9A	1	
TYA Transfert de Y dans A	Y A	Implicite	TYA	98	1	V V

	DECIMA	L 0-65280		не	XA 00-F	F OCTET HA	uT
DECIMAL	HEXA	DECIMAL	HEXA	DECIMAL	HEXA	DECIMAL	HEX
0	00	16384	40	32768	80	49152	co
256	01	16640	41	33024	81	49408	CI
512	02	16896	42	33280	82	49664	CZ
768	03	17152	43	33536	83	49920	C3
1024	04	17408	44	33792	84	50176	C 4
1280	05	17664	45	34048	85	50432	C 5
1536	06	17920	46	34304	86	50688	C6
1792	07	18176	47	34560	87	50944	C7
2048	08	18432	48	34816	88	51200	C 8
2304	09	18688	49	35072	89	51456	C9
2560	OA	18944	44	35328	· 8Å		
						51712	CA
2816	08	19200	48	35584	86	51968	CB
3072	OC	19456	4C	35840	8C	52224	CC
3328	00	19712	40	36096	8D	52480	CD
3584	OE	19968	4E	36352	8 <i>E</i>	52736	CE
3840	0 F	20224	4F	36608	8 F	52992	CF
4096	10	20480	50	36864	90	53248	00
4352	11	20736	51	37120	91	53504	DI
4608	12	20992	52	37376	92	53760	D2
4864	13	21248	53	37632	93	54016	03
5120	14	21504	54	37888	94	54272	D4
5376	15	21760	55	38144	95	54528	05
5632	16	22016	56	38400	96	54784	06
5888	17	22272	57	38656	97	55040	07
6144	18	22528	58		98	55296	08
			59	38912			09
6400	19	22784		39168	99	55552	
6656	14	23040	5A	39424	9 A	55808	DA
6912	18	23296	58	39680	98	56064	DB
7168	10	23552	5C	39936	9 C	56320	DC
7424	10	23808	50	40192	90	56576	00
7680	1 E	24064	5E	40448	9 E	56832	DE
7936	1F	24320	5F	40704	9 F	57088	DF
8192	20	24576	60	40960	AO	57344	EO
8448	21	24832	61	41216	1A	57600	El
8704	22	25088	62	41472	A2	57856	E2
8960	23	25344	63	41728	A3	58112	E3
9216	24	25600	64	41984	A4	58368	E4
9472	25	25856	65	42240	A5	58624	E5
9728	26	26112	66	42496	A6	58880	E6
9984	27	26368	67	42752	AF	59136	E 7
	28		68	43008	Ã8	59392	E8
10240		26624	69		Ã9	59648	E9
10496	29	26880	27	43264			EA
10752	2A	27136	6A	43520	AA	59904 60160	
11008	28	27392	68	43776	Yg		E8 EC
11264	2C	27648	6C	44032	AC	60416	
11520	20	27904	60	44288	AD	60672	ED
11776	2 E	28160	6E	44544	A.E	60928	EE
12032	2F	28416	6F	44800	AF	61184	EF
12238	30	28672	70	45056	80	61440	FO
12544	31	28928	71	45312	81	61696	FI
12800	32	29184	72	45568	B 2	61952	F 2
13056	33	29440	73	45824	83	62208	F 3
13312	34	29696	74	46080	84	62464	F 4
13568	35	29952	75	46336	35	62720	F 5
13824	36	30208	76	46592	86	62976	F6
14080	37	30464	77	46848	87	63232	F 7
14336	38	30720	78	47104	38	63488	F 8
14592	39	30976	79	47360	89	63744	F9
14848	3 Å	31232	74	47616	BA	64000	FÁ
		31488	18	47872	88	64256	FB
15104	38				BC	64512	FC
15360	3 C	31744	7C	48128			FD
15616	30	32000	70	48384	80	64768	
15872	3E	32256 32512	1E 1F	48640	BE	65024	FE
16128	3F	20110			BF		

**			
			**
	2		i
			3 1
			.
		•	
			ř
			100 000
•			
	* #		
			4
1			:
			1
•			•
	a a		
			N
			<i>4</i> *
	·		
•			
	2		
	-34		8.

Les utilisateurs d'Oric ont pu le constater, il manquait à cette remarquable machine un ouvrage de référence traitant de ses capacités d'une façon complète.

Ce livre répond à leur attente.

Sa conception est à la fois originale et ambitieuse.

Il tente de présenter le système ORIC dans son ensemble et de favoriser la communication directe avec la machine.

Son contenu est dense mais que les débutants se rassurent; il leur est parfaitement accessible.

Ils pourront aussi échapper à la banalité des exercices basic et découvrir des horizons insoupçonnés.