

Deuxième partie

L'INTERPRÉTEUR BASIC MICROSOFT

Notre but n'est pas de donner ici une simple liste d'adresses de routines, ou de renseignements, sur l'interpréteur BASIC des micro-ordinateurs THOMSON T07 et T07-70.

Nous décrivons tout au contraire une *méthode* permettant de décoder n'importe quel interpréteur Microsoft, ceux-ci étant conçus toujours de la même (excellente !) manière.

Une fois compris et décrit le fonctionnement de l'interpréteur (chapitre I et II), nous étudions (chapitre III) le traitement de quelques instructions fondamentales du BASIC.

Nous signalons enfin que certaines routines détaillées ci-après demandent un effort de compréhension ; le lecteur se verra récompensé par toutes les applications qui découleront de cette étude.

Comprendre l'exécution d'un programme BASIC

Les instructions d'un programme sont décodées et exécutées par une routine qui se présente sous la forme d'une boucle, décrite à chaque nouvelle instruction.

Grâce à une table des adresses, l'interpréteur détermine, à partir du code de l'instruction BASIC à exécuter, l'adresse du sous programme correspondant, qui réalise le traitement; l'instruction suivante est alors prise en compte pour poursuivre l'exécution du programme.

Une fois décrit le codage d'un programme, puis trouvée la table des adresses et la boucle d'exécution, nous aurons tous les éléments nécessaires à la compréhension du fonctionnement de l'interpréteur.

I. Implantation et codage d'un programme

1 — Pour trouver l'adresse à partir de laquelle est implanté un programme, on peut par exemple utiliser le programme suivant :

```
10 REM Programme 1
20 AB=0 'AB Code &H41,42
30 FOR I=&H6000 TO &H7FFF 'RAM
40 IF PEEK(I)=&H41 THEN IF PEEK(I+1)=&H42 THEN PRINT HE
X$(I)
50 NEXT I:END
```

Pour les TO7, on obtient les adresses \$660C et \$6619 (et aussi \$6680, situé dans la zone des variables: voir chapitre suivant); le programme commence donc une vingtaine d'octets auparavant, soit exactement en \$65F5.

2 — Le programme suivant permettra alors de visualiser en hexadécimal le codage d'un programme :

```
100 REM Programme 2
110 ADR0=&H65F5
120 CLS:FOR I=ADR0 TO ADR0+229 STEP 10
130 PRINT HEX$(I)+";";FOR J=I TO I+9
140 PRINT USING"% 2";HEX$(PEEK(J));:NEXT
150 PRINT:NEXT I:END
```

On obtient :

65F5	66	8	0	64	8C	20	20	50	72	6F	67	72
6601	61	6D	6D	65	20	32	0	66	18	0	6E	41
660D	44	52	30	D4	26	48	36	35	46	35	0	66
6619	37	0	78	9D	3A	81	20	49	D4	41	44	52
6625	30	20	BB	20	41	44	52	30	C7	20	35	39

3 — Connaissant le code ASCII (Annexes), on peut en déduire les éléments suivants :

— Chaque ligne possède un en-tête de 4 octets :

- Les deux premiers contiennent l'adresse de l'en-tête de la ligne suivante ;
- Les deux suivants contiennent le numéro de la ligne.

- Chaque mot BASIC (REM, =, CLS, FOR, etc...) est codé sur un ou deux octets (le premier étant alors égal à &HFF), de valeur supérieure ou égale à &H80.
- Chaque ligne se termine par un octet contenant 0, qui précède l'en-tête de la ligne suivante.
- La fin du programme est marquée par trois zéros consécutifs (ici en \$6677); ils sont suivis de la zone où l'on trouve les variables.

II. La table et les codes des mots réservés du BASIC

1. Recherche de la table

Lors de la saisie des instructions d'un programme, l'interpréteur reconnaît et code les différents mots du BASIC, rangés dans une table par ordre de codes croissants.

Celle-ci contient les codes ASCII des caractères composant les mots; la dernière lettre est toutefois codée différemment pour marquer la fin de chaque mot.

La table sera donc trouvée à l'aide du programme 1 précédent, en recherchant dans la ROM BASIC (adresses de \$0 à \$3FFF) les caractères EN (codés &H45,4E) du mot END situé au début de la table (code &H80).

On obtient \$92,\$162 et \$1758; mais un examen de ces adresses (programme 2 précédent, en modifiant ADR0) montre que le END (suivi de FOR, de code &H81) est bien en \$92.

Le début de la table est donc en \$92, et la fin en \$269; \$26A contient un 0 marquant la fin de la table.

On constate de plus que le bit de plus fort poids du dernier caractère de chaque mot est mis à 1 dans la table.

Le programme suivant permet de lister les 125 mots du BASIC, avec leurs codes (le programme 2, où l'on écrit différentes instructions et fonctions à la ligne 20, permet de constater que le codage change après le 87^e mot, codé &HD5).

```

200 REM  Programme 3
210 A$="" : CODE=&H80
220 FOR I=&H92 TO &H269
230 X=PEEK(I) : IF X<&H80 THEN A$=A$+CHR$(X) : GOTO260
240 A$=A$+CHR$(X-&H80) : IF CODE<=&HD5 THEN PRINT HEX$(CODE); ELSE PRINT"FF"+HEX$(CODE-86)
250 PRINT": "+A$, : CODE=CODE+1 : A$=""
260 NEXT : END

```

Le résultat est donné en annexe; on constatera que les fonctions sont codées sur 2 octets, le premier étant égal à &HFF.

La table des mots est donc en fait composée de 2 tables:

- celle des instructions commence en \$92 et comporte 86 mots (&H56),
- celle des fonctions commence en \$1CF (trouvé toujours grâce au programme 1, en recherchant les caractères SG de SGN) et comporte 39 mots (&H27).

2. Utilisation par l'interpréteur

L'interpréteur utilise la table des mots en initialisant un des registres avec l'adresse du début de la table; ce registre r est ensuite incrémenté pour balayer la table, jusqu'à ce que soit trouvée la suite des caractères composant le mot à coder.

L'initialisation du registre peut être à priori faite soit par l'instruction *LDr # \$0092* ou *\$01CF* pour les fonctions), soit par une instruction *LDr adresse*, "adresse" étant l'adresse d'une mémoire contenant \$0092 (ou \$01CF).

Le programme suivant recherche la valeur &H0092 dans la ROM et dans la RAM (de \$6000 à \$65F4):

```

300 REM  Programme 4
310 A=0 : B=&H92
320 I0=0 : I1=&H3FFF : GOSUB350
330 I0=&H6000 : I1=&H65F4 : GOSUB350
340 BEEP : END
345 '
350 FOR I=I0 TO I1
360 IF PEEK(I)=A THEN IF PEEK(I+1)=B THEN PRINT HEX$(I)
, HEX$(PEEK(I-1)), HEX$(A)+HEX$(B)
370 NEXT : RETURN

```

On trouve \$38AA et \$6202, précédés tous deux de &H56 (code de RORB, qui ne convient pas); pour la valeur &H1CF (obtenue avec 20 A=1 : B=&HCF), on trouve \$38AF et \$6207, précédés tous deux de &H27 (code de BEQ, ce qui ne convient pas davantage).

Pour l'initialisation du début des tables, on doit donc chercher une instruction ayant \$38AA ou \$6202 comme partie adresse, puis \$38AF ou \$6207 (on verra que la page 0 du BASIC n'est pas située en \$62, et on n'a donc pas à chercher une adresse \$02 ou \$07).

Le programme 4 permet (en modifiant la ligne 310) de constater que l'on a nulle part une telle instruction.

On peut pourtant constater que la zone de \$6201 à \$620A contient les mêmes informations que la zone de \$38A9 à \$38B2; cette dernière zone est donc recopiée dans la RAM à l'initialisation du système (par une routine située en \$37E5), dans le but évident d'être utilisée plus tard !...

On doit donc chercher dans l'interpréteur les 2 instructions :

```
LD r # valeur
LEAr d,r (ou ADD # d si r=D)
```

avec d=\$6202 — valeur (ou \$6207 — valeur)

Le programme 4 légèrement modifié permet de lister toutes les instructions ayant une adresse de \$61F0 à \$620F par exemple; on en trouve 9, dont seulement 4 sont précédées d'un code d'instruction à adressage immédiat (LDU dans les quatre cas); il s'agit de \$2969, 2A1F, 2A60 et 37E9.

Un examen attentif, par le désassembleur de la première partie, des instructions situées autour de ces adresses montre que les instructions :

```
2A1E LDU # $61F7,
2A5F LDU # $61FC,
```

correspondent respectivement à l'initialisation des registres Y et B par les valeurs situées en \$6202 (adresse du début de la table) et \$6201 (nombre de mots), et par celles situées en \$6207 et \$6206.

Dans les deux cas, il y a en effet utilisation de la routine suivante, située en \$2A21 :

2A21	CLR	\$45	Numéro du mot dans la table
	LEAU	10,U	\$6201 ou 6206
	LDB	,U	Nombre de mots dans la table

	BEQ	\$2A5F	
	LDY	1,U	Contenu de \$6202 ou 6207 → Y
	LDX	,S	Adresse début du mot du buffer
2A2E	LDA	,X+	Une lettre du mot à coder
	BSR	\$2A88	Minuscule → majuscule
	SUBA	,Y+	Une lettre de la table
	BEQ	\$2A2E	Lettre suivante
	CMPA	# \$80	Si égal, on a trouvé le mot
	BNE	\$2A76	Mot suivant
	etc...		

A titre indicatif, signalons que le codage d'une instruction est réalisé par une routine commençant en \$29A3, appelée elle-même en \$44C (ou en \$424 si on est en mode commande) ; cette routine code les mots du BASIC directement dans le buffer clavier contenant les caractères de l'instruction, situés à partir de \$6445 ; le codage est effectué lors de la frappe de la touche "ENTRÉE" terminant l'instruction.

Signalons aussi que la routine contenant l'instruction \$2968 est utilisée par LIST ; elle opère en sens inverse de \$29A3.

3. Application immédiate

Il suffit de modifier (POKE ou LOADM) le contenu des octets \$6201 à 6203, et \$6206 à 6208, pour pouvoir créer son propre vocabulaire BASIC, par exemple avec des mots français : voir troisième partie.

III. La table des adresses d'instructions

La table des adresses de traitement des instructions et fonctions contient forcément une suite d'adresses de routines situées dans le BASIC ; le 1^{er} octet doit donc être toujours inférieur ou égal à &H3F (ROM) ou, éventuellement, compris entre &H60 et &H65 (RAM avant le programme).

Le programme 5 suivant cherche donc dans la ROM une suite d'au moins 20 couples d'octets consécutifs dont le 1^{er} est inférieur ou égal à &H3F.

```

400 REM  Programme 5
410 FOR I=0 TO &H3FE0
420 IF PEEK(I)>&H3F GOTO460
430 FOR J=I+2 TO I+248 STEP 2      '250 adresses maxi

```

```

440 IF PEEK(J)<&H40 THEN NEXT
450 IF J>I+40 THEN PRINT HEX$(I), HEX$(J-1): I=J+1    '>20
460 NEXT I

```

On obtient \$1C à \$6D et \$26B à \$2C8 ; or, si l'on observe (programme 2) la fin de cette dernière zone, on s'aperçoit en fait qu'elle continue jusqu'en \$2DE (soit 58 adresses), puisque jusque-là on ne trouve qu'en \$2C9 et \$2CB les adresses \$6233 et \$6236, adresses où l'on a JMP \$7F3 renvoyant à une adresse de la ROM.

Or, on pourra constater que seuls les 58 premiers mots du BASIC (de END à PLAY) sont directement exécutables.

Une ultime vérification consiste à relever par exemple les 1^{ère}, 9^e, 30^e et 36^e adresses de la table, soient \$53B, \$5F2, \$35EB et \$35E6, correspondant en principe respectivement à END, RUN, CLS et BEEP.

- EXEC &H53B inséré dans un programme provoque l'arrêt de celui-ci ;
- EXEC &H5F2 provoque le redémarrage du programme à partir de la première ligne ;
- EXEC &H35EB efface l'écran ;
- EXEC &H35E6 génère un "bip" sonore.

La table des adresses de traitement des instructions se situe donc bien en \$26B ; on trouvera ces adresses en annexe.

Nous verrons au chapitre suivant que la zone de \$20 à \$6D représente la table des adresses des fonctions BASIC.

Premières applications :

Dans un programme en langage machine, un JSR \$35EB effacera l'écran ; un JSR \$35E6 génèrera un "bip".

Un JSR \$5F2 permettra d'appeler un sous-programme écrit en BASIC à partir d'un programme en langage machine.

On pourra enfin étudier le traitement des diverses instructions du BASIC en listant les routines correspondantes.

IV. Le traitement des instructions

1. Utilisation de la table des adresses

Il nous faut trouver l'endroit où s'effectue le branchement aux différentes adresses de traitement des instructions.

On vient de voir que le BASIC doit pour celà, à partir du code C d'une instruction, calculer l'adresse :

$$A = \&H26B + (C - \&H80) * 2,$$

où se trouve l'adresse du sous-programme de traitement de l'instruction.

On va donc chercher s'il existe dans la ROM ou la RAM une valeur $\&H26B$ (ou à défaut une valeur de $\&H267$ à $26F$ par exemple, puisque la formule exacte du calcul de A est encore inconnue); cette valeur pourra être précédée d'un code d'instruction à adressage immédiat (ce ne pourra être que LDr ou ADDD) ou indexé, le pré-octet indiquant alors un déplacement sur 16 bits.

On reprend donc le programme 4 ; on obtient les deux adresses $\$38AC$ et $\$6204$, les deux octets $\$38AB$ et $\$6203$ contenant la valeur $\&H92$ (code de SBCA direct); ceci ne correspond pas à la condition ci-dessus.

Il faut donc chercher une instruction ayant $\$38AC$ ou $\$6204$ comme partie adresse ; la page 0 du BASIC n'étant en effet située ni en $\$38$, ni en $\$62$ (voir paragraphe suivant), l'adresse ne peut être $\$AC$ ou $\$04$.

On reprend donc le programme 4 pour chercher $\$38AC$, puis $\$6204$; on ne trouve pas $\$38AC$, par contre on trouve $\$6204$ en $\$2B36$; l'octet $\$2B35$ contient $\&HBE$, c'est-à-dire le code de LDX en adressage étendu, ce qui correspond bien à ce que l'on cherche.

Remarquons que si l'on n'avait pas trouvé, on ferait comme au paragraphe précédent, c'est-à-dire que l'on chercherait une instruction LDr # valeur, avec "valeur" située par exemple entre $\$61F0$ et $620F$.

On peut donc maintenant à l'aide du désassembleur examiner les instructions autour de $\$2B35$; on y trouve effectivement le calcul de A ci-dessus.

Le listing complet est le suivant :

2B25	JSR	\$6270	Contient RTS
	BNE	\$2B2B	
	RTS		Si 3A en tête (')
2B2B	CMPA	# \$80	Mot BASIC?
	LBLO	\$722	Caractère ASCII (affectation)
	CMPA	# \$B9	Code de PLAY
	BHI	\$2B42	Code > &HB9
2B35	LDX	\$6204	Début table (&H26B)
	LSLA		(Code-&H80) * 2 → A
	TFR	A,B	
	ABX		(Code-80) * 2 + \$26B → X
	LDX	,X	Adresse traitement
	JSR	\$B2	Voir ci-après
	JMP	,X	Traitement de l'instruction
2B42	CMPA	# \$FF	
	BEQ	\$2B4E	
	CMPA	# \$D5	Code de <
	BLS	\$2AFF	Contient JMP \$7F3 (SN Error)
	JMP	[\$620E]	Contient \$7F3 (SN Error)
2B4E	JSR	\$B2	Voir ci-après
	CMPA	# \$9C	Code de MID\$
	LBEQ	\$1100	
	CMPA	# \$A1	Code de INPUT
	LBEQ	\$27A5	
	CMPA	# \$A4	Codé de SCREEN
	LBEQ	\$33CC	
	JMP	\$6273	Contient RTS

Application immédiate :

En modifiant le contenu des octets \$6204 et 6205, on pourra faire utiliser par le BASIC sa propre table d'adresses.

On pourra alors créer son propre langage (par exemple un BASIC "entier" très rapide) en écrivant les sous-programmes correspondants.

L'avantage est que l'on profitera ainsi de toute la partie de l'interpréteur réalisant le codage et l'édition.

2. Routine \$B2

Cette routine fondamentale permet le "balayage" des caractères d'un programme.

Le programme suivant, appelé par USR, nous donnera tout d'abord l'emplacement de la page 0 du BASIC :

TFR	DP,B	Registre de page 0 → B
STB	3,X	Poids faible accu. entier
RTS		

Ceci correspond à la suite des codes hexadécimaux 1F(31), B9 (code de PLAY), E7, 3, 39 (code de 9), qui seront donc implantés en mémoire (en \$65F8) le plus simplement possible par l'instruction 31 du programme suivant :

```
31 PLAYXX9$:POKE &H65FA,&HE7:POKE &H65FB,3
40 DEFUSR=&H65F8:PRINT HEX$(USR(0))
```

On obtient \$61, qui constitue l'emplacement de la page 0 du BASIC.

D'où le listing de la routine \$B2 :

61B2	INC	\$BA	
	BNE	\$61B8	
	INC	\$B9	
61B8	LDA	\$xxxx	Adresse caractère courant
	CMPA	# \$3A	Caractère ":" ?
	BHS	\$61C9	
	CMPA	# \$20	Espace ?
	BNE	\$61C5	
	JMP	\$B2	Élimine les espaces
61C5	SUBA	# \$30	Chiffre de 0 à 9 ?
	SUBA	# \$D0	
61C9	RTS		

Cette routine incrémente donc l'adresse du caractère courant, contenue dans \$61B9 et 61BA, puis retourne dans le registre A le code du caractère.

Le registre CC des codes conditions est positionné de la manière suivante :

- Z est mis à 1 si l'on a un 0 ou le caractère ":", c'est-à-dire une fin d'instruction ;
- C est mis à 1 si l'on a un chiffre, et à 0 dans le cas contraire.

REMARQUE: Cette routine existe sur tous les interpréteurs ; elle peut être trouvée directement par le programme 1 (légèrement modifié) en cherchant l'endroit de la RAM où se trouve l'adresse du caractère courant du programme.

3. Contrôles de validité

La syntaxe des instructions, ou les valeurs de certains paramètres, sont systématiquement contrôlés par l'interpréteur au fur et à mesure de l'exécution d'un programme (par exemple ici, \$2B42 contrôle la validité d'un code).

Si l'un de ces contrôles est positif, il y a branchement en \$353, le registre B contenant le code de l'erreur : voir \$7F3 par exemple.

Les variables systèmes ERR (code de l'erreur) et ERL (numéro de la ligne où s'est produite l'erreur), situées respectivement en \$6189 et \$618A (voir chapitre II la routine \$770, en \$7C2) sont alors positionnées.

L'exécution est ensuite soit interrompue et le message d'erreur (contenu dans une table située en \$1722) affiché, soit poursuivie en α (jusqu'à l'instruction RESUMÉ) s'il existe dans le programme une instruction ON ERROR GO TO α .

V. Boucle d'exécution d'un programme

Pour la trouver, il nous faut chercher dans la ROM un JSR, BSR ou LBSR \$2B25 (ou \$2B2B).

Or, en remontant avant le sous-programme de traitement des instructions, on trouve immédiatement en \$2B21 les instructions :

2B21	BSR	\$2B25	Traitement
	BRA	\$2AED	Début de la boucle

D'où le listing de la boucle :

2AED	STS	\$8C	
	JSR	\$32BB	Surveillance du clavier
	LDX	\$B9	Adresse caractère courant
	STX	\$34	
	LDA	,X+	Ø ou " : " ?
	BEQ	\$2B02	Nouvelle ligne
	CMPA	# \$3A	
	BEQ	\$2B1F	
	JMP	\$7F3	SN Error

2B02	LDD	,X++	A-t-on trois 0?
	BEQ	\$2B65	Fin du programme
	LDD	,X+	Numéro de la ligne
	STD	\$2C	
	STX	\$B9	Sur dernier octet de l'en-tête
	LDA	\$86	≠ 0 si TRON (voir \$139E)
	BEQ	\$2B1F	Pas de trace demandée
	LDA	# \$5B	Code de "["
	JSR	\$1055	Écrit sur l'écran
	LDA	\$2C	Numéro de la ligne
	JSR	\$1ED1	Écrit le numéro
	LDA	# \$5D	Code de "]"
	JSR	\$1055	Écriture
2B11	JSR	\$B2	1 ^{er} octet de l'instruction
	BSR	\$2B25	Traitement
	BRA	\$2AED	Instruction suivante

Routine de surveillance du clavier:

32BB	JSR	\$6294	Contient RTS
	JSR	\$E809	KTST\$ du moniteur
	BCC	\$32BA	RTS (pas de touche)
	PSHS	B	Sauvegarde
	JSR	\$E806	GETC\$ du moniteur
32C8 à ...			Arrêt si CNT/C; boucle si STOP;
...32E1			Code touche → \$65B1 sinon

Application immédiate:

En intervenant en \$6294, on pourra supprimer ou modifier l'action du clavier: voir 4^e partie.

2

Le traitement des variables

Le traitement des variables sera décodé en étudiant l'instruction d'affectation ; on trouvera ainsi du même coup le traitement des opérateurs et des fonctions BASIC.

Avant d'étudier la routine correspondante, il est nécessaire de connaître la représentation mémoire des variables et des tableaux, et aussi la manière dont est gérée la mémoire.

1. Représentation des variables

Les variables rencontrées lors de l'exécution d'un programme sont placées au fur et à mesure dans une zone de mémoire située après le programme lui-même ; cette zone sera donc examinée à l'aide du programme 2.

Pour les TO7, on trouve que chaque nom de variable est précédé d'un octet contenant la valeur du type de la variable (dans les 4 bits de plus fort poids), puis le nombre de caractères du nom diminué de un (les %, \$, ! ou # éventuels ne comptent pas) ; on trouve ensuite le nom codé en ASCII, et enfin la valeur contenue dans un nombre d'octets égal à la valeur du type.

Exemple : on écrit dans le programme 2 :

```
110 AB=1.5:CDE%=-3:AB$="chaîne":CBA=-2.25:ADR0=&H669F
```

On obtient :

669F	0	0	0	<u>41</u>	41	42	81	40	0	0	<u>22</u>	43
66AB	44	45	FF	FD	<u>31</u>	41	42	6	66	D	<u>42</u>	43
66B7	42	41	82	90	<u>0</u>	0	<u>43</u>	41	44	52	<u>30</u>	8F

- Le bit de plus fort poids d'une variable entière (type égal à 2) représente le signe ; les 15 autres bits contiennent la valeur en complément à deux ;
- Une variable réelle simple précision (type égal à 4) est codée en mode virgule flottante : le premier octet est égal à l'exposant de 2 augmenté de &H80 ; les trois autres octets contiennent la mantisse normalisée (inférieure à 1 et supérieure ou égale à 0,5), le bit de plus fort poids étant remplacé par le signe de la valeur.

Exemple :

$$-2,25 = -4 \times 0,5625 = -2^2 (2^{-1} + 2^{-4})$$

d'où le codage :

82 90 00 00

- Une variable double précision (type égal à 8) est codée de la même manière, la mantisse occupant 7 octets.
- Le premier octet d'une variable chaîne (type égal à 3) contient le nombre de caractères de la chaîne ; les deux autres octets représentent l'adresse de la chaîne, située soit dans le programme lui-même en cas d'affectation simple (variable = "chaîne"), soit en fin de mémoire en cas de concaténation.

REMARQUE : Le nombre d'octets d'une variable réelle peut changer selon les ordinateurs ; de même, la représentation des noms de variable peut être différente. Par exemple, lorsque le nom est limité à deux caractères, le codage se fait toujours sur 2 octets, dont le bit de plus fort poids est positionné pour indiquer le type.

II. Représentation des tableaux

Les tableaux sont placés dans une zone située après celle des variables ; un tableau est créé dans cette zone par l'instruction DIM, ou à défaut par la première utilisation du tableau (la taille étant alors égale à 11).

Le nom d'un tableau est codé comme celui d'une variable ; il est suivi de l'en-tête suivant, qui précède les valeurs :

- deux octets contiennent le nombre total d'octets occupé par le tableau (y compris l'en-tête) ;
- un octet contient le nombre d'indices ;
- pour chaque indice, on a ensuite deux octets contenant la valeur maximale augmentée de un, en commençant par le dernier indice.

Pour les valeurs, c'est le premier indice qui varie d'abord, puis le second est incrémenté, etc...

Exemple : On écrit dans le programme 2 :

```
110 DIM AB%(3,2):AB%(1,0)=1:AB%(2,0)=2:AB%(3,0)=3:AB%(0,1)=4:AB%(1,1)=5:ADR0=&H66D3
```

On obtient :

66D3	21	41	42	0	1F	2	0	3	0	4	0	0
66DF	0	1	0	2	0	3	0	4	0	5	0	0

III. Gestion de la mémoire

L'interpréteur gère la mémoire à partir des adresses de chaque zone.

Les mémoires contenant ces adresses peuvent être trouvées soit par le programme 1 (on recherchera par exemple la mémoire contenant l'adresse du début de la zone des variables, trouvée par comptage à partir de \$65F5), soit par observation (le programme 2 permettra de trouver l'emplacement des chaînes, le contenu de \$618C, 618D donne la valeur du

pointeur S, etc...) soit par l'étude de l'instruction d'affectation (voir par exemple la routine \$A48, en \$A8A).

On trouve que ces mémoires sont situées en page 0 du BASIC ; contenant des adresses, elles occupent deux octets chacune :

- \$611C pointe sur la première instruction du programme, c'est-à-dire sur \$65F5 ;
- \$611E pointe sur le premier octet de la zone des variables ;
- \$6120 pointe sur le premier octet de la zone des tableaux ;
- \$6122 pointe sur le premier octet libre situé après les tableaux, c'est-à-dire sur le dernier octet utilisable par la pile S ;
- \$6124 pointe sur le "fond" de la pile S ; la zone des chaînes commence juste après ;
- \$612A pointe sur le dernier octet de la zone des chaînes ; les caractères utilisateurs éventuels sont situés juste après, en commençant par la ligne du bas du dernier ; la ligne du haut de GR\$(0) est située en FIN-1, FIN étant la plus haute adresse du BASIC (deuxième paramètre d'un CLEAR, ou plus haute adresse de la RAM par défaut).

IV. Recherche d'une variable ou d'un tableau

L'adresse de cette routine fondamentale de l'interpréteur sera trouvée en listant les premières instructions du traitement de l'affectation ; celle-ci doit en effet d'abord déterminer l'adresse où devra être rangé le résultat, qui sera ensuite calculé.

La routine recherche une variable ou un élément de tableau dans la zone de mémoire correspondante ; elle crée cette variable (sauf si elle est située dans un calcul d'expression, ceci pour le cas où la valeur devrait être affectée à un élément de tableau, non déplaçable pendant l'affectation elle-même) ou le tableau lui-même, en les initialisant à 0 s'ils n'existent pas encore ; elle retourne enfin l'adresse de la valeur correspondante.

Dans le cas des TO7, la routine est située en \$A48 (on trouve en effet en \$722 l'instruction JSR \$A48) ; elle retourne l'adresse de la valeur dans le registre X et dans la mémoire \$613D ; le type de la valeur est placé dans l'octet d'adresse \$6105 ; enfin, \$61B9 (contenant l'adresse du caractère courant du programme) est positionné sur le premier caractère qui suit la variable ou l'élément de tableau.

Le listing est en effet le suivant :

A48	CLRB		
	JSR	\$B8	1 ^{er} caractère du nom à chercher
A4B	STB	\$04	Entrée pour DIM, avec B≠0
	JSR	\$6297	Contient RTS
	BSR	\$A0A	Nom rangé à partir de \$657A ; nombre de caractères en \$613C
A52 à ...A75			Valeur du type → \$6105
A76	JSR	\$B2	1 ^{er} caractère après le nom
	LDB	\$07	Contient normalement 0
	DECB		
	LBEQ	\$B64	Recherche début d'un tableau
	INCB		
	BNE	\$A88	Pas de tableau (cas de FOR)
	SUBA	#\$28	Code de "("
	LBEQ	\$B12	Elément de tableau
A88	CLR	\$07	Variable
	LDX	\$1E	Début de zone des variables
A8C	CMPX	\$20	Fin zone des variables?
	BEQ	\$AA2	Variable n'existe pas encore
	LDB	,X	1 ^{er} octet d'un variable
	LSRB		
	LSRB		
	LSRB		
	LSRB		Garde les 4 bits de fort poids
	PSHS	B	Valeur du type
	JSR	\$ADB	Comparaison des noms
	PULS	B	
	BEQ	\$AD8	Variable trouvée
	ABX		Pas la bonne variable
	BRA	\$A8C	Variable suivante
AA2 à ...			Variable ajoutée en fin de zone (sauf si \$A48 a été appelée en \$800), après déplacement de la zone des tableaux
...AD7			
AD8	STX	\$3D	Adresse dans X et \$3D
	RTS		

Puis on a, pour les tableaux :

B12 à ...			
...B62			Calcule et empile les indices
B63	LDB	#\$5F	Ou CLRB en \$B64
B65	PSHS	B	5F ou 0 dans la pile
B67 à ...			Cherche nom du tableau (et RTS si pile contient 0) ; tabl. créé par \$B97 si n'existe pas encore
...BE8			
BE9 à ...			
...C12			Calcule l'adresse de l'élément
C13	STX	\$3D	Adresse dans X et \$3D
	RTS		

Le lecteur est bien entendu vivement invité à étudier les instructions ou sous-programme non listés ici pour des raisons de place.

Amélioration possible : Lorsqu'on étudie le traitement des instructions du BASIC, on s'aperçoit que l'interpréteur appelle la routine \$A48 chaque fois qu'il rencontre une variable ou un élément de tableau.

Par exemple dans le cas d'une boucle, il y a donc à *chaque passage* exploration de la RAM jusqu'à trouver la valeur correspondante, située pourtant toujours au même endroit dans le cas d'une variable.

On verra dans la 4^e partie que ceci peut être évité, en intervenant en \$6297 ; la vitesse d'exécution des programmes sera alors très nettement augmentée.

V. Traitement d'une expression

1. Instruction d'affectation

L'instruction se présente sous la forme :

variable = expression

("variable" pouvant être un élément de tableau).

La routine de traitement est la suivante :

722	JSR	\$A48	Recherche de la variable
	STX	\$3F	Adresse de la valeur
	LDB	#\$D4	Code de "="
	JSR	\$D0	Contient JMP \$7EB
	LDA	\$05	Type de la variable
	PSHS	A	
	JSR	\$81A	Calcul de l'expression
	PULS	A	
734	JSR	\$2510	Conversion éventuelle de l'exp. dans le type de la variable
	JSR	\$CD	Contient JMP \$2502
	LBNE	\$1C36	Si type 2,4 ou 8 ; range valeur à l'adresse contenue dans \$3F
73D à ...			Si type 3 ; affecte la chaîne résultat à la variable
...76A			

Les routines situées en \$7EB et \$2502 sont appelées de nombreuses fois par l'interpréteur.

- \$7EB est utilisée pour tous les contrôles de syntaxe; elle teste si le caractère courant est bien égal à celui contenu dans l'accumulateur B (SN Error sinon), puis elle retourne le caractère suivant; on a en effet :

7EB	CMPB	[\$61B9]	Code du caractère courant
	BNE	\$7F3	
	JMP	\$B2	Caractère suivant
7F3	LDB	#\$02	Code erreur SN: syntax error
	JMP	\$353	Affichage erreur; arrêt

- \$2502 positionne le registre CC du 6809 selon la valeur x d'un type, contenue dans l'octet \$6105 (ou dans le registre A si le point d'entrée est en \$2504):

Si x = 2 (entier), N et C sont mis à 1, Z et V à 0
 Si x = 3 (chaîne), Z et C sont mis à 1, N et V à 0
 Si x = 4 (réel), V et C sont mis à 1, N et Z à 0
 Si x = 8 (double précision), N,Z,V et C sont mis à 0.

On a en effet :

2502	LDA	\$05	
2504	CMPA	#\$08	0 → C si 8
	DECA		
	DECA		
	DECA		1 → Z si 3; 1 → N si 2
	BLE	\$250F	Si 2 ou 3
	BCC	\$250F	Si 8
	ORCC	#\$02	Si 4; 1 → V
250F	RTS		

2. Calcul d'une expression

Il est effectué par la routine \$81A; celle-ci range le résultat x dans "l'accumulateur flottant" (utilisé par l'instruction USR du BASIC) situé à partir de \$6155.

- Si x est entier, il est placé en \$6157 et 6158;
- Si x est une chaîne, l'adresse du descripteur de la chaîne (c'est-à-dire toujours \$658A, adresse où l'on trouve la longueur et l'adresse de la chaîne) est placée encore en \$6157 et 6158;

- Si x est un réel, la valeur absolue est rangée de \$6155 à 6158, le signe étant contenu dans \$615D (dans le bit de plus fort poids);
- Si x est un réel double précision, l'accumulateur va de \$6155 à 615C; le signe est toujours en \$615D.

Enfin, le type du résultat est placé dans \$6105, et il y a positionnement de \$61B9 sur le premier caractère qui suit l'expression.

Le listing est le suivant:

81A	BSR	\$815	Revient sur caractère précédent
	CLRA		
81D	CMPX	# \$9641	Ou LDA \$41 en \$81E
	PSHS	A	
	LDB	# \$01	
	JSR	\$336	Test de débordement mémoire
	JSR	\$770	Traite un opérande; valeur dans l'accumulateur flottant
82A à	...875		Décodage d'un opérateur ou RTS
876	BSR	\$883	Calcule une sous-expression
878	BRA	\$82A	Suite de l'expression

3. Traitement d'un opérande

On verra que la routine \$883 contient un JSR \$81E pour chercher la valeur de l'opération situé après l'opérateur T; tous les opérandes sont donc traités par la routine suivante, située en \$770:

770	JSR	\$627C	Contient RTS
	LDX	\$B9	
	JSR	\$B2	1 ^{er} caractère de l'opérande
	BEQ	\$76B	MO Error (Missing Opérand)
	BCC	\$780	On n'a pas un chiffre
77B	STX	\$B9	On a un chiffre
	JMP	\$1E03	Traitement des constantes
780	JSR	\$A36	Retourne C=0 si on a une lettre
	BCC	\$800	Variable
	CMPA	# \$2E	Code de "."
	BEQ	\$77B	Constante réelle
789 à	...		Décode et traite ± unaires, " (chaîne), NOT, &, ERR et ERL, USR
...7D4			
7D5	CMPA	# \$BD	Code FN
	LBEQ	\$623C	Contient JMP \$7F3 (SN Error)

	INCA		A-t-on FF (fonction)?
	BEQ	\$80D	Contient JMP \$2A93
7DE à ...7F7	...		Teste si on a "(" et SN Error sinon; calcule l'exp. entre ()
800	JSR	\$A48	Recherche de la variable
	STX	\$57	Adresse
	JSR	\$CD	Contient JMP \$2502
	BNE	\$80A	Si variable numérique
	RTS		Si variable chaîne
80A	JMP	\$1C02	Valeur dans accu. flottant

4. Applications

En intervenant en \$623C, on pourra utiliser des fonctions utilisateurs (FN) : voir troisième partie.

En intervenant en \$627C, on pourra modifier la routine \$770 du traitement d'un opérande, par exemple au niveau du calcul des constantes, ou de la recherche d'une variable : voir quatrième partie.

VI. Traitement des fonctions BASIC

On vient de voir qu'il est effectué en \$2A93; on a :

2A93	JSR	\$6288	Contient RTS
	JSR	\$B2	Deuxième octet du code
	TFR	A,B	Code C → B
	LSLB		$x = (C - \$80) * 2 \rightarrow B$
	JSR	\$B2	Caractère suivant
	CMPB	# \$4C	PTRIG
	BLS	\$2AA5	Si inférieur ou égal
	JMP	[\$6213]	Contient \$7F3 (SN Error)
2AA5	PSHS	B	x → pile
2AA7 à ...2AE1	...		Traitement des paramètres; conversion en réel pour les fonctions de code \$FF84 à 89 (SQR à TAN)
2AE2	PULS	B	Dépile x
	LDX	\$6209	Contient \$0020
	ABX		X pointe sur x ^e adresse
	JSR	[0,X]	Traitement de la fonction
	JMP	\$24FD	Contrôle du type

On constate immédiatement que la table des fonctions commence en \$0020, ce qui correspond à ce que l'on avait trouvé au chapitre I; elle se termine en \$006D.

On trouvera en annexe les adresses de traitement de toutes les fonctions BASIC (lues dans la table, ou décodées différemment, par exemple pour TAB ou SPC, etc...).

Application immédiate : Un programme écrit en langage machine peut utiliser directement des fonctions BASIC, par exemple SQR, CSNG, etc... : voir 3^e partie.

VII. Traitement des opérateurs BASIC

L'enchaînement des routines correspondantes est relativement complexe.

1. Table des opérateurs

On a vu que les opérateurs sont décodés en \$82A; on a :

82A à 833			Initialisations
834 à ...			Codage dans \$6143 des opérat. de relation :
...84A			1 si > , 2 si =, 3 si ≥ , 4 si < , 5 si ≠ , 6 si ≤
84B	LDB	\$43	
	LBNE	\$97A	Traite les opérat. de relation
851 à ...			x = (code opérat. - \$C7) → B, et test de
...85E			concaténation de chaîne
85F	LBEQ	\$DBB	Traitement de la concaténation
863	PSHS	B	x → pile
	LSLB		2 * x → B
	ADDB	,S1	3 * x → B et dépilement de x
	LDX	#\$006E	Début de la table
	ABX		\$6E + 3x → X
86C à ...875			Voir ci-après

La table des opérateurs commence donc en \$006E; elle comprend trois octets pour chaque opérateur :

- le premier représente la priorité de l'opérateur (&H7F pour la plus grande, celle de la puissance);

— les deux autres l'adresse de traitement.

Les informations concernant tous les opérateurs sont listées en annexe.

2. La routine \$883

Elle calcule une sous-expression aTb; le listing est le suivant :

883	STB	\$41	Priorité de l'opérat. courant T
	CMPB	# \$7F	
	BEQ	\$8F3	Opérateur puissance
	LDU	1,X	
	PSHS	U	Adresse traitement de T → pile
	CMPB	# \$51	
	BLO	\$905	Opérateur logique (AND à IMP)
	ANDB	# \$FE	
	CMPB	# \$7A	
	BEQ	\$90E	Opérateur MOD et @
	JSR	\$939	Opérande a → pile
	JSR	\$81E	Calcule b; bit le T' suivant
89D	JSR	\$24FD	b numérique (SN Error sinon)?
	JSR	\$95A	Dépilement dans \$6163 à 616B de l'opérande en haut de la pile (on y trouve donc maintenant l'adresse du traitement de T)
8A3 à ...			Conversion si a et b de types ≠; permutation s'ils sont entiers
... 8B1			
8B2	RTS		Branche au traitement de T

On observera qu'en \$8F3 (opérateur puissance) et \$905 (opérateurs logiques), on a exactement la même succession d'appels des routines \$939, \$81E et \$95A, suivie du branchement à la routine de traitement de l'opérateur (\$2391 pour ↑ ; l'adresse située en \$7B sert en effet pour les opérateurs de relation : voir \$97A à 99B, puis \$99C).

Le lecteur observera aussi le dépilement dans Y des adresses de retour des routines \$939 et \$95A, le retour se faisant alors par une instruction JMP ,Y; ceci permet de n'empiler que les opérandes et opérateurs successifs.

3. La gestion des priorités

On constate que \$883 appelle lui-même la routine \$81E; celle-ci range donc la priorité de T (opérateur courant) dans la pile, puis lit l'opérande b et l'opérateur T' suivants; on arrive alors en \$86C, où l'on a :

86C	LDA	,S	Priorité de T, rangée en \$81E
	CMPA	,X	Priorité de T', dans la table
	BHS	\$853	Si priorité de T \geq celle de T'
	BSR	\$812	b numérique (SN Error sinon) ?
	BSR	\$883	

\$853 contient l'instruction PULS A,PC.

Donc si la priorité de T est supérieure ou égale à celle de T', on dépile la priorité de T et on retourne en \$89D (car le JSR \$81E en \$89A place \$89D dans la pile); d'où le calcul de l'opération correspondant à T (cas de $a * b + \dots$).

Si la priorité de T est inférieure à celle de T', il y a un nouvel appel de \$883, c'est-à-dire que la priorité et l'adresse du traitement de T sont empilés, ainsi que la valeur de l'opérande b; l'opération correspondante ne sera donc exécutée que plus tard (cas de $a + b * \dots$).

4. Applications

Un programme écrit en langage machine peut utiliser les routines de traitement des opérateurs BASIC, en particulier *, @ et MOD : voir la troisième partie.

On pourra aussi écrire une routine simplifiée de traitement des opérateurs (pour + par exemple) utilisable en BASIC et conduisant à des calculs plus rapides de 50 % : voir la troisième partie (INC).

3

Etude de quelques instructions BASIC

Nous détaillons ici le fonctionnement de certaines instructions fondamentales du BASIC TO7.

Les instructions sont bien sûr traitées sous des formes très voisines sur tous les micro-ordinateurs, d'où l'intérêt général de cette étude.

Celle-ci nous permettra d'envisager diverses interventions sur l'interpréteur, que nous verrons dans la suite de notre ouvrage.

1. Instruction de branchement

Le mot GO est traité en \$606 ; selon qu'il est suivi de TO ou de SUB, on va respectivement en \$624 ou en \$612, avec \$61B9 positionné sur le premier caractère suivant.

1. GO TO

L'instruction GO TO α est donc traitée de la manière suivante :

624	JSR	\$B8	Premier chiffre de α
	JSR	\$6FD	Calcul de α , rangé en \$6130
	BSR	\$66E	Cherche le 0 de fin de ligne
	LEAX	1,X	X pointe sur la ligne suivante
	LDD	\$30	$\alpha \rightarrow D$
	CMPD	\$2C	n° β de la ligne du GO TO
636	BHI	\$636	$\alpha > \beta \Rightarrow \alpha$ après la ligne actuelle ; sinon, chercher avant
	LDX	\$1C	Adresse du début du programme
	JSR	\$4A4	Retourne dans X l'adresse de la ligne de n° α (ou C=1 si n'existe pas)
	BCS	\$655	UL Error
	LEAX	-1,X	Adresse du 0 précédant α
	STX	\$B9	Positionne le caractère courant
	RTS		Provoque l'exécution de α

Routine \$66E (ou \$66B) : Elle commence par :

66B	LDB	# \$3A	Code de " : "
66D	LDA	# \$5F	Ou CLRB en \$66E

On a ensuite de \$66F à 696 la recherche, à partir du caractère courant du programme, à la fois d'un 0 et du caractère dont le code a été mis dans B (en \$66B ou \$66E) ; l'adresse correspondante est placée dans X.

\$66E recherche donc la fin de la ligne courante (0) ; \$66B recherche la fin de l'instruction en cours (0 ou \$3A).

Cette routine fondamentale est utilisée chaque fois que l'interpréteur doit sauter quelque chose ; elle est donc en particulier utilisée aussi par RETURN, REM, DATA, ON, IF et FOR.

Dans le cas des TO7, la routine détecte les caractères de code \$22 (guillemets ; les caractères de code \$3A sont alors ignorés jusqu'aux deuxièmes guillemets), FF (fonction ; un octet est alors sauté) et 89 (ceci pour les cas où l'on a des IF imbriqués ; la mémoire \$6171 est alors incrémentée).

Conséquence immédiate :

Si l'on intervient dans le programme lui-même, par exemple, pour remplacer des constantes par leur valeur binaire, ou des variables par leur adresse (voir quatrième partie), il ne faudra jamais placer les valeurs &H0, 22, 3A, 89 ou FF.

2. GOSUB et RETURN

Le traitement de GOSUB α est le suivant :

612	LDB	#\$03	On va empiler trois registres
	JSR	\$336	Test de débordement mémoire
	LDU	\$B9	Caractère courant
	LDX	\$2C	Numéro β de la ligne courante
	LDA	#\$BC	Code de SUB
	PSHS	U,X,A	
	BSR	\$624	GO TO : positionne sur ligne α
	JMP	\$2AED	Boucle d'exécution du programme

Lorsque l'interpréteur rencontre RETURN (traité en \$640), il positionne S sur la valeur en haut de la pile (RG Error si ce n'est pas &HBC), puis arrive en \$65D où l'on trouve :

65D	PULS	A,X,U	
	STX	\$2C	Restaure β (ligne du GOSUB)
	STU	\$B9	Caractère courant sur α
663	BSR	\$66B	Cherche fin de GOSUB α
	CMPX	#\$8D06	Ignoré ici (sert pour REM)
	STX	\$B9	Caractère courant
66A	RTS		On est revenu après GOSUB

REMARQUE: Les boucles non terminées sont dépilées en \$646 (routine \$2F3 appelée avec &HFF dans \$613F : voir paragraphe III), avant le retour au programme principal).

3. ON

L'instruction ON expression GO... est traitée en \$6D7 (voir \$36B5), où la valeur x de l'expression est rangée dans \$6158 ; l'interpréteur se positionne alors sur le x^e numéro de la liste (lu et calculé par la routine \$6FD) et se branche en \$608, c'est-à-dire au traitement de GOTO et GOSUB.

4. Amélioration possible

A chaque instruction de branchement rencontrée, il y a calcul de l'étiquette et exploration du programme ; ceci est refait chaque fois si l'on repasse sur le même branchement.

On verra dans la quatrième partie que ceci peut être évité, ce qui améliore nettement la vitesse d'exécution.

II. Instruction de test

L'instruction IF expression... est traitée en \$697, où l'on trouve :

697	JSR	\$81A	Calcul de l'expression
69A	à 6AF		SN Error si pas THEN ou GOTO
6B0	JSR	\$1CC8	1 → Z si exp. fausse (0), 0 si vraie
	BNE	\$6C8	Exécute le GOTO ou la suite du THEN ; sinon, doit être sauté
	CLR	\$71	Va compter nbre de IF imbriqués
6B7	BSR	\$663	Cherche 0 ou \$3A (voir RETURN)
	TSTA		0 ou 3A ?
	BEQ	\$66A	Pas ELSE ⇒ ligne suivante (RTS)
	JSR	\$B2	\$3A ; ELSE ou suite du THEN ?
	CMPA	≠\$8F	Code de ELSE
	BNE	\$6B7	Suite du THEN
	DEC	\$71	A-t-on des IF imbriqués ?
	BPL	\$6B7	Pas encore le bon ELSE
	JSR	\$B2	Trouvé
6C8	JSR	\$B8	Caractère courant
	LBCS	\$624	Chiffre ⇒ branchement (GOTO)
	JMP	\$2B25	Traitement des instructions

REMARQUE: Lors de la saisie d'un programme, ELSE est codé par &H8F précédé de &H3A, d'où l'utilisation de la routine \$66B (par BSR \$663).

III. Instruction FOR... NEXT

1. Traitement de FOR

Il est effectué en \$1578 ; après avoir initialisé la variable de contrôle, la routine empile successivement :

- l'adresse du 0 ou &H3A terminant l'instruction FOR (2 octets),
- le numéro (2 octets) de la ligne courante (grâce à ces deux informations, on pourra se repositionner après le NEXT sur la première instruction de la boucle),
- la valeur finale de la variable de contrôle (4 octets),
- le signe du pas d'incrémentation (1 octet),

- la valeur du pas (4 octets)
- le type du pas (1 octet, égal à 2 ou 4),
- l'adresse du NEXT (2 octets) correspondant au FOR (trouvé par la routine \$16AD décrite ci-après),
- l'adresse de la variable de contrôle (2 octets),
- la valeur &H81, c'est-à-dire le code de FOR (1 octet).

La routine positionne ensuite \$612C avec le numéro de la ligne du NEXT (\$61B9 est déjà positionné sur le NEXT, par la routine \$16AD) et appelle en \$1602 la routine \$1604, correspondant au traitement de NEXT sans incrémentation de la variable de contrôle (grâce à la mémoire \$6185 initialisée ici à &H4F et non à 0).

L'exécution se poursuit donc soit après le NEXT, soit à partir de la première instruction de la boucle.

REMARQUE: Cet appel de la routine du traitement de NEXT (sans incrémentation) n'est pas effectué par tous les BASIC; la boucle est alors toujours exécutée au moins une fois.

2. Traitement de NEXT

NEXT est traité en \$1605, où la mémoire \$6185 est initialisée à 0.

Puis en \$161A est appelée la routine \$2F3, qui balaye la pile S à partir du sommet pour trouver le FOR correspondant au NEXT; le BASIC Microsoft autorisant en effet les sorties anormales de boucle par GOTO, le bon FOR peut ne pas être situé au sommet de la pile.

Le registre S est alors modifié pour pointer sur la valeur &H81 correspondant à ce FOR, ce qui dépile les boucles dont on est sorti anormalement; puis les valeurs empilées sont récupérées.

La variable de contrôle est alors incrémentée (sauf si \$6181 est différent de 0) et le test de fin de boucle exécuté.

S'il est vrai, le numéro de la ligne du FOR est récupéré (en \$1622, par LDX 15,S et STX \$2C), puis l'adresse de la première instruction de la boucle (par LDX 17,S et STX \$B9); il y a enfin branchement en \$2AED, c'est-à-dire à la boucle d'exécution du programme; l'instruction exécutée est donc la première après le FOR.

Si le test est faux, les 19 octets empilés par le FOR sont dépilés (en \$166E) et il y a encore branchement en \$2AED; l'exécution se poursuit donc après le NEXT.

3. La routine \$16AD

Il s'agit d'une deuxième routine d'exploration du programme (la première étant \$66B ou \$66E) ; son point d'entrée est en fait soit en \$16AC (\$6170 est alors initialisé par &H4F), soit en \$16AD qui initialise \$6170 à 0.

Le programme est ici exploré à partir du caractère courant jusqu'à trouver un octet contenant 0, &H3A (" : "), &H8F (ELSE) ou &HC4 (THEN).

Selon la valeur de \$6170, la routine teste si le caractère suivant est FOR (&H81) ou WHILE (&HAF) ; si c'est le cas, on a deux boucles imbriquées ; le registre B est alors incrémenté et il y a retour au début.

Si l'on a un NEXT (&H82) ou un WEND (&HB0), toujours selon la valeur de \$6170, la routine examine si c'est le bon ; pour cela, B est décrémenté et il y a retour au début si l'on n'a pas 0.

Si c'est le cas, la routine retourne le numéro de la ligne dans \$6178 ; le caractère courant est positionné sur le NEXT ou le WEND.

Conséquence immédiate :

Si l'on intervient dans le programme lui-même, on ne devra jamais introduire des valeurs &H3A81, ou &H8F81, ou &HC4AF, etc...

4. Applications

La routine \$16AC trouve le WEND correspondant à un WHILE.

Il sera donc très facile de créer les routines complètes de traitement de WHILE et WEND : voir 3^e partie.

IV. Les instructions graphiques

1. Le graphisme des TO7

L'écran est divisé en 200 lignes de 40 segments de huit points ; chaque segment est décrit par deux octets de la mémoire écran.

Celle-ci est en effet composée de deux blocs de 8K-octets situés à la même adresse \$4000 ; ils sont sélectionnés par la valeur du bit 0 d'un port d'un PIA (port C du circuit 6846), lui même situé à l'adresse \$E7C3.

- la valeur 1 sélectionne la mémoire de "forme"; chaque bit d'un octet représente alors un point de l'écran, qui appartient soit à la forme (bit égal à 1), soit au fond (bit égal à 0).

En général, la forme est constituée par un message ou un graphisme.

Exemple :

```
100 POKE &HE7C3,PEEK(&HE7C3) OR 1:ADR=&H4000+C*8+L*40
110 POKE ADR,2^(7-C MOD 8)
```

affiche le point de coordonnées (C,L), C étant la colonne (0 à 319) et L la ligne (0 à 199).

REMARQUE: La routine \$F161 du moniteur du TO7 met à 1 le bit 0 de \$E7C3 (\$F328 pour le TO7-70).

- la valeur 0 sélectionne la mémoire de "couleur"; chaque octet représente alors la couleur de la forme (dans les bits 3,4,5) et du fond (bits 0,1,2) d'un segment de 8 points.

Exemple :

```
200 POKE &HE7C3,PEEK(&HE7C3) AND &HFE:POKE ADR,&H0F
```

affiche le point précédent en rouge (code 1=&B001), le fond du segment étant blanc (code 7=&B111).

REMARQUE 1: Sur le TO7-70, les bits 6 et 7 de la mémoire couleur sélectionnent respectivement la couleur pastel pour la forme et pour le fond s'ils sont mis à 0; ils sont toujours à 1 sur le TO7.

REMARQUE 2: \$E7C3 contrôle aussi entre autres l'affichage en majuscules ou minuscules (bit 3), et la couleur du cadre de l'écran (bits 4,5 et 6); voir annexes.

2. Line (x_1, y_1) - (x_2, y_2)

L'instruction trace une droite entre les points $P_1 (x_1, y_1)$ (qui peut être omis) et $P_2 (x_2, y_2)$; le traitement est le suivant:

34FB	BSR	\$354B	Affiche P_1 ; $P_2 \rightarrow X$ et Y
	LDX	\$6576	Abscisse de P_2 (colonne)
3500	LDY	\$6578	Ordonnée de P_2 (ligne)
3504	JMP	\$E80C	Tracé (DRAW\$ du moniteur)

Routine \$354B : elle est utilisée par les 4 instructions LINE, BOX, BOXF et PSET, quel que soit le mode (graphique ou caractère).

La routine place d'abord dans les registres X et Y les coordonnées du dernier point affiché ; elles sont remplacées par les coordonnées de P_1 si elles figurent dans l'instruction (routine \$34CB) ; X et Y sont alors placés respectivement en \$6572 et \$6574, puis rangés dans la pile.

Les coordonnées de P_2 sont ensuite lues (toujours par \$34CB), puis stockées en \$6576 et \$6578.

Si l'instruction opère en mode caractère, les attributs sont traités en \$3574, qui se termine par un JMP \$E833 affichant un caractère (routine CHPL\$ du moniteur).

Si l'on est en mode graphique, l'attribut éventuel couleur est traité en \$35C1 ; s'il ne figure pas dans l'instruction, la routine prend (en \$35B7) la couleur forme courante contenue dans le registre moniteur \$603B (COLOUR) ; dans les deux cas, la couleur est rangée dans le registre moniteur \$6038 (FORME).

X et Y sont enfin dépilés, ce qui restaure les coordonnées de P_1 , le registre \$6041 (CHDRAW) est mis à 0 et il y a branchement en \$E80F qui affiche le point graphique (routine PLOT\$ du moniteur).

3. BOX et BOXF $(x_1, y_1) - (x_2, y_2)$

L'instruction BOX est traitée en \$3507, où la mémoire \$6571 est initialisée à 0 ou à &H46 si l'instruction est suivie de la lettre F.

Il y a ensuite appel de la routine \$354B, qui range donc entre autres x_1 et y_1 dans X et Y ; selon la valeur de \$6571, il y a branchement en \$351A pour BOX et en \$352C pour BOXF ; on a alors :

351A	LDX	\$6576	$x_2 \rightarrow X$; on a $Y=y_1$
	BSR	\$3504	Trace drte. horiz. $P_1 - (x_2, y_1)$
	BSR	\$3500	Trace drte. vert. $(x_2, y_1) - P_2$
	LDX	\$6572	$x_1 \rightarrow X$; on a $Y=y_2$
	BSR	\$3504	Trace drt. horiz. $P_2 - (x_1, y_2)$
	LDY	\$6574	$y_1 \rightarrow Y$; on a $X=x_1$
352A	BRA	\$3504	Trace drt. $(x_1, y_2) - P_1$ et RTS

La routine \$352C (BOXF) opère en traçant la droite horizontale d'ordonnée y_1 , puis celle d'ordonnée $y_1 \pm 1$ (selon que y_2 est supérieur ou inférieur à y_1), etc..., jusqu'à arriver à y_2 .

Applications :

En s'inspirant des routines ci-dessus, on pourra facilement écrire des routines dessinant par exemple des losanges, ou même des cercles "pleins" : voir troisième partie.

4. PSET (x, y)

C'est l'instruction la plus simple ; elle est traitée en \$34EC, où l'on trouve :

34EC	BSR	\$34CB	$x \rightarrow X, y \rightarrow Y$
	BSR	\$34F1	
	RTS		
34F1	PSHS	X,Y	
	BRA	\$3565	Traite couleur ; affiche P

\$3565 appartient en effet à la routine \$354B.

Les autres instructions

L'étude accomplie jusqu'ici débouche comme nous allons le voir sur un grand nombre d'applications.

Il reste cependant bien d'autres instructions à étudier, que le lecteur choisira en fonction de son intérêt personnel ; il s'aidera pour cela des listes des principales routines et adresses du BASIC, données en annexe (listes non exhaustives !)

Par exemple, si l'on désire lever la protection d'un programme, on devra étudier le traitement de LIST (ou de PEEK ou POKE par exemple) ; on constatera alors que la protection est assurée par la routine \$2D29, qui teste la mémoire \$61A2.

La protection sera donc levée en chargeant (par LOADM, puisque POKE est interdit) la mémoire \$61A2 ; on devra pour cela enregistrer préalablement l'octet \$61A2 sur cassette (par SAVEM "nom", &H61A2, &H61A2, Ø).

4

Méthode pratique de décodage d'un interpréteur

Nous avons déjà largement expliqué, en particulier au chapitre I, la démarche à suivre pour décoder un interpréteur BASIC quelconque, à priori totalement hermétique !...

Nous résumons ici cette méthode, opérant avec 5 programmes BASIC de cinq ou six lignes chacun et un peu de réflexion...

1. Début d'un programme

Le programme 1 permet de trouver l'adresse B du début d'un programme, la recherche devant commencer à l'adresse A du début de la RAM (ligne 30 à modifier).

2. Codage d'un programme

Les octets situés à partir de B seront examinés grâce au programme 2 ; on déterminera ainsi le codage des instructions et de quelques mots-clés du BASIC.

3. Tables des mots-clés du BASIC

Le programme 1 permet de trouver dans le BASIC (ligne 30 à modifier) l'adresse C du début de la table des noms d'instructions, et l'adresse D du début de la table des noms de fonctions; on cherchera pour cela (ligne 20 à modifier) les (n-1) premières lettres d'un mot-clé de n caractères situé vers le début de la table à localiser.

On en déduira les codes de tous les mots clés du BASIC, grâce au programme 3 (valeurs &H80, &HD5 et 86 de la ligne 240 à modifier, ainsi que la ligne 220).

4. Utilisation des tables de noms

Le programme 4 permet de trouver dans le BASIC et dans la RAM, en principe entre A et B (lignes 320 et 330 à modifier) les valeurs C et D.

S'il existe une adresse E de la RAM contenant C ou D, il suffira en principe de modifier les octets E et E+1 pour que le BASIC puisse utiliser un autre vocabulaire (à créer).

5. Tables des adresses

Le programme 5 permet de trouver dans le BASIC (ligne 410 à modifier) l'adresse F du début de la table des adresses des instructions.

On disposera alors déjà de toutes les adresses de traitement des instructions du BASIC, que l'on pourra donc étudier.

6. Recherche des octets contenant l'adresse de la table

Le programme 4 permet de trouver dans le BASIC et dans la RAM la valeur F (si on ne la trouve pas, on cherchera une valeur comprise entre F-d et F+d, avec d=10 par exemple); on obtient une ou plusieurs adresses G (\$38AC et \$6204 pour le T07).

7. Routine de traitement des instructions

On examinera les octets situés "autour" de chaque adresse G trouvée à l'étape précédente.

S'ils contiennent pour une de ces adresses le calcul de : $x = (F \pm d) + (C - C0) * 2$ (d pouvant être éventuellement différent de 0, et C0 étant le plus petit code d'instruction BASIC, c'est-à-dire en principe toujours &H80), on a trouvé la routine de traitement des instructions ; on va donc directement à l'étape suivante.

Si l'on ne trouve pas le calcul précédent, le programme 4 permet de trouver la (ou les) valeur G précédentes dans la mémoire (ligne 310 à modifier) ; on obtient donc une ou plusieurs adresses H (si l'on ne trouve aucune adresse contenant G, on cherchera en modifiant la ligne 360 une valeur comprise entre G-I et G+I).

Pour les TO7 par exemple, on obtient la seule adresse \$2B36.

On doit forcément trouver alors "autour" d'une de ces adresses H le calcul de :

$$y = ([G] \pm d) + (C - C0) * 2$$

ce qui correspond à la routine de traitement des instructions ([G] désigne le contenu de la mémoire d'adresse G, c'est-à-dire F).

8. Boucle d'exécution des programmes

Elle est en principe toujours située juste avant la routine de traitement des instructions (si ce n'est pas le cas, on cherchera dans le BASIC l'appel de cette routine, grâce au programme 1).

9. Les routines fondamentales

L'étude de l'instruction d'affectation donnera enfin l'adresse des routines de recherche d'une variable, de traitement d'un opérande (qui traite elle-même les fonctions) et du calcul d'une expression (où l'on trouvera le traitement des opérateurs).

Ceci terminera le décodage de l'interpréteur et ouvrira la porte aux applications...