1

Notions préliminaires

Avant d'étudier la programmation en assembleur, c'est-à-dire de pouvoir travailler en langage-machine, il est important de connaître un minimum de notions utiles.

Ces notions sont les suivantes :

A. Généralités sur le fonctionnement de votre ordinateur.
B. Bases de numération, adresses, arithmétique binaire.
C. Code-ASCII.
D. Langage-assembleur.
E. Cartouches-assembleur, en bref.

A. L'ORDINATEUR

L'unité centrale (U.C.) d'un ordinateur se compose d'un processeur, de mémoires pour le stockage des données, de fils (appelés BUS) reliant les boîtiers les uns aux autres (tableau n° 1).

Cette U.C. communique avec l'extérieur : clavier, écran, L.E.P., etc. (périphériques*).

Les mémoires peuvent être mortes ou permanentes (R.O.M.*), car ne s'effacent pas lors de l'extinction de l'appareil, ou vives ou volatiles

* périphérique : ce qui gravite autour de l'U.C.
* R.O.M. : read only memory

(R.A.M.*) car se vident lorsqu'il n'y a plus d'alimentation électrique. La R.O.M. appartient au constructeur ; elle contient des indications qu'on ne peut changer, et qui guident l'appareil.

Toute cartouche, basic (TO7(70)), assembleur, ou autre, est une R.O.M.. La R.A.M. vous est réservée. Elle contient cependant, à l'initialisation à froid (mise en route de l'appareil) ou à chaud (RESET), quelques informations que le système utilise pour son propre fonctionnement. Ces informations, en R.A.M., peuvent être modifiées par le programmeur.

Exemple : les pages zéro du moniteur ou de la cartouche memo-7 (TO7(70)) ou de l'interpréteur basic incorporé (MO5, TO9), sont des informations du système, mises en R.A.M..

La mémoire réservée à l'utilisateur se situe entre les adresses &H6000 et &HDFFF pour le TO7-70 et le TO9 (avec des banques d'extension, s'implantant entre &HA000 et &HDFFF). De &H6000 à &H60FF se trouve la page 0 du moniteur, et de &H6100 è &H61FF la page 0 du basic. Pour les correspondances avec les TO7 et MO5, voyez le tableau numéro 2, à la fin de l'ouvrage.

Une page fait 256 octets, et 4 pages font 1K-octet soit 1 024 octets.

La R.O.M., incluse dans l'appareil par le constructeur, place à la mise en route du système des variables dans l'espace & H6000 à &H60FF (voir tableau n° 6, et pour le MO5 : voir tableaux n°s 2 et 6). De même, le BASIC 1.0 place-t-il ses variables entre & H6100 et & H61FF (pour le MO5, voir tableau n° 2).

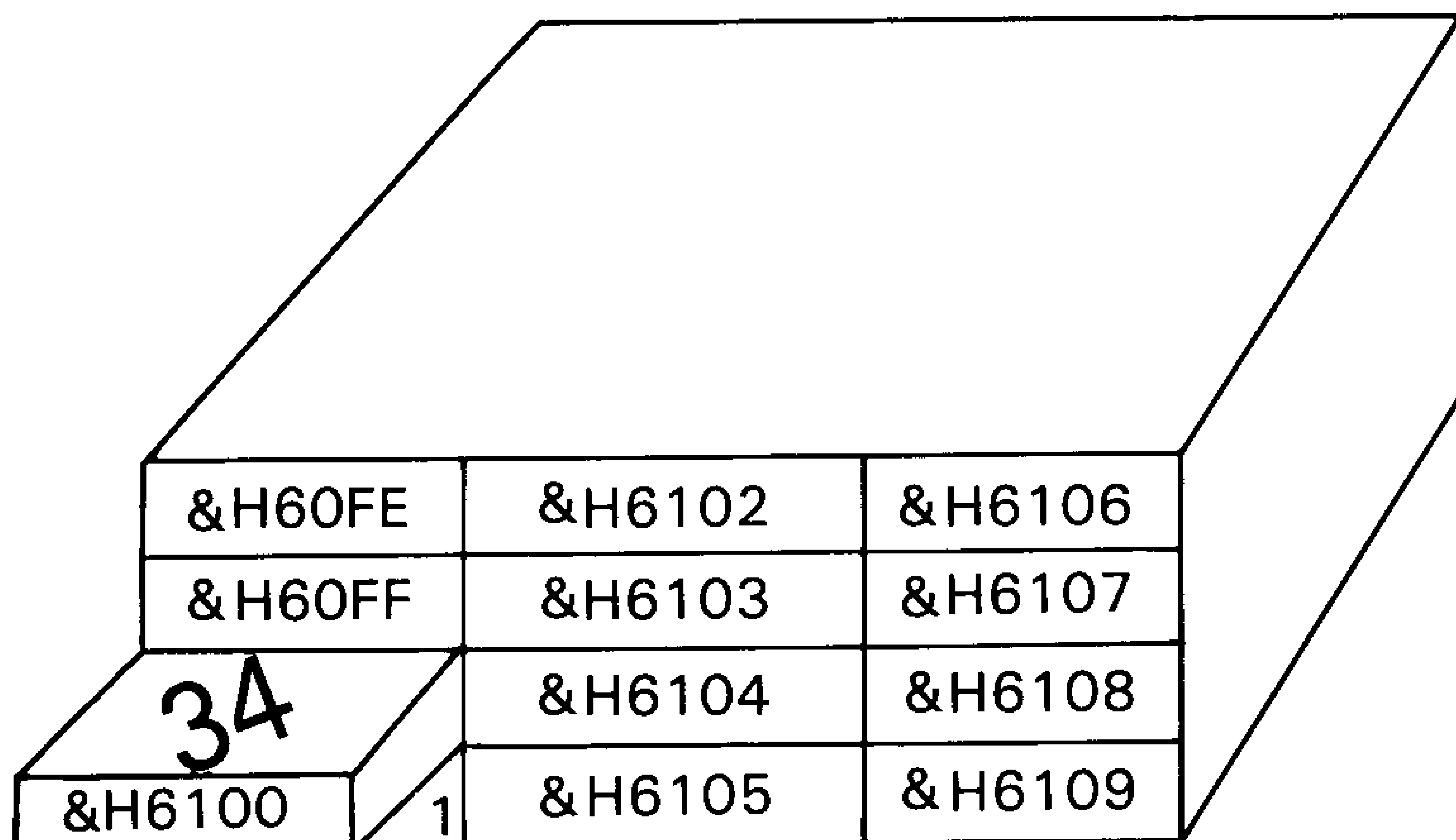
Toute case-mémoire possède un numéro (par exemple : &H6100*) et contient un octet ayant une certaine valeur décimale égale à 8 chiffres binaires.

* R.A.M. : random access memory

*&H6100 en hexadécimal = 24 832 en décimal (voir bases de numération).

Exemple :

La case &H6100 contenant un octet, dont la valeur décimale est : 34



Un octet est donc un ensemble de 8 bits ou chiffres binaires numérotés de 0 à 7, car le zéro compte en informatique. Chaque bit de cet octet peut être « allumé » ou « éteint », c'est-à-dire prendre une valeur égale à 1 (bit « allumé ») ou 0 (bit « éteint »).

Ce sont des petits transistors, contenant ou non un peu de "courant", qui, sur le plan matériel, représentent les 0 ou les 1. L'ordinateur travaille donc en binaire (bi = deux), avec deux chiffres (0 et 1), et par octets (huit chiffres pouvant être égaux chacun à 0 ou à 1).

Le bit est la plus petite unité de travail logique, en informatique. Son nom résulte de la contraction des mots anglais « Binary Digit », ce qui signifie chiffre binaire.

On distingue le plan logiciel ou de la logique informatique (0 et 1) appelé SOFTWARE, du plan matériel ou HARDWARE (petit « courant » ou ensemble d'électrons dans les transistors). On oppose, de même, le digital ou numérique (discontinu dans le temps : 0 et 1) à l'analogique (phénomène physique continu, par exemple : le courant électrique).

Numéros des bits	7	6	5	4	3	2	1	0
État de ces bits	0	1	1	0	1	0	1	0

un octet (= 8 bits numérotés de 0 à 7)

Dans cet exemple :

Le bit numéro zéro est « éteint » (= 0).
Le bit numéro un est « allumé » (= 1).
Le bit numéro deux est « éteint » (= 0), etc..

Par convention, le bit numéro zéro est appelé de poids le plus faible, ou le moins significatif. Le bit numéro sept est celui de poids le plus fort, ou le plus significatif. Nous reviendrons sur ce sujet dans le paragraphe suivant, après avoir défini les « bases de numération ».

Lors de l'allumage de l'ordinateur, le système est prêt à recevoir vos instructions, très rapidement. En tapant sur le clavier, vous écrivez en R.A.M.-utilisateur (nous verrons plus loin où commence l'implantation d'un programme en basic, et comment déterminer le début de celle-ci). La cartouche-basic (TO7(70)) ou le basic résident (MO5, TO9), qui sont des R.O.M., décodent ensuite vos informations, font appel au moniteur-système, avant de vous « rendre la main ». L'écran n'est utile que pour votre travail, l'ordinateur fonctionnant très bien sans lui.

Une note : Le moniteur du système est la R.O.M. qui fait « tourner la machine », cela n'a pas de rapport avec le « téléviseur », appelé parfois lui aussi moniteur. En fait nous devons distinguer :

1. Moniteur-système = R.O.M.
2. Moniteur-vidéo = écran vidéo
3. Moniteur de programmation binaire = zone de travail en binaire pour la cartouche-assembleur (nous l'étudierons par la suite).

Un Conseil enfin : N'allumez jamais un périphérique après l'unité centrale, cela pouvant fausser quelques octets de la R.A.M..

B. BASES DE NUMÉRATION

On peut concevoir une infinité de bases de numération, suivant le nombre de chiffres que l'on utilise.

1. La base 10 (décimale) :

Les dix doigts de l'homme étant la machine à calculer la plus primitive, nous avons l'habitude de compter en base 10, dans laquelle il existe 10 chiffres, de 0 à 9.

Dans ce système, un nombre de 1 chiffre donnera 10 possibilités (10^1), un nombre de 2 chiffres donnera 100 possibilités (10^2), un nombre de 3 chiffres donnera 1 000 possibilités (10^3), etc..

2. La base 2 (binaire) :

Seuls les deux chiffres 0 et 1 sont utilisés.

Dans ce système, un nombre de 1 chiffre donnera 2 possibilités (2^1), un nombre de 2 chiffres donnera 4 possibilités (2^2), un nombre de 3 chiffres donnera 8 possibilités (2^3), etc..

Comme nous l'avons dit précédemment, une case-mémoire contient un octet, soit 8 bits numérotés de 0 à 7. En valeur décimale, le premier bit (n° 0) vaudra 0 ou 1, selon son état (« allumé » ou « éteint »). Le deuxième bit, lui, vaudra 0 ou 2, selon son état (0 ou 1). Le troisième bit vaudra 0 ou 4, le quatrième 0 ou 8, et ainsi de suite jusqu'au bit numéro 7 (= huitième bit) qui vaudra 0 ou 128 (128 lorsqu'il est « allumé » ou mis à 1).

Un octet permet donc l'écriture d'un nombre de 8 chiffres binaires, soit donc de 256 possibilités (2^8), c'est-à-dire qu'on pourra mettre dans un octet une valeur décimale comprise entre 0 et 255.

Exemple : 136 (décimal) à mettre en binaire dans 1 octet.

Valeur décimale	128 +	64 +	32 +	16 +	8 +	4 +	2 +	1
nombre binaire	1	0	0	0	1	0	0	0

Seuls les bits n°s 7 et 3 sont « allumés » : 128 + 8 = 136.

Pour écrire en binaire un nombre décimal supérieur à 255, on devra utiliser un deuxième octet dit de poids fort, car chaque bit de ce deuxième octet aura une valeur 256 fois plus grande que le bit correspondant de l'octet de poids faible.

Exemple :

Valeur décimale
[128+ 64 + 32 + 16 + 8 + 4 + 2 + 1] *256+[128+ 64 + 32 + 16 + 8 + 4 + 2 + 1]
nombre binaire

0	0	0	1	0	0	0	1

0	0	1	0	1	0	0	1

Nous avons : (16*256) + (1*256) + 32 + 8 + 1 = 4 393 en décimal.

En résumé, deux octets permettent d'utiliser seize chiffres binaires. La manipulation de nombre de seize chiffres n'étant guère aisée, on a cherché une base de numération dont tous les chiffres pouvaient être parfaitement codés sur un demi-octet (4 bits). Quatre bits permettant 16 combinaisons (2^4), on a naturellement pensé à la base 16 (ou hexadécimale).

3. La base 16 (hexadécimale) :

La base 16 fait appel à 6 nouveaux chiffres sous forme de lettres : A, B, C, D, E, F.

Héxadécimal	*Décimal*	*Binaire*
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

La conversion des nombres décimaux en nombres hexadécimaux (et réciproquement) n'étant pas évidente, on utilisera avec profit des tables (tableau numéro 3), des méthodes de calcul par le binaire, etc.. Le tableau numéro 3 peut être photocopié pour l'avoir rapidement sous la main en cas de nécessité.

Par contre, la conversion binaire-hexadécimal ou hexadécimal-binaire est d'une remarquable simplicité. Reprenons le codage binaire du nombre décimal 4393 : 00010001 00101001.

Par demi-octet, nous aurons :

code binaire

0	0	0	1		0	0	0	1		0	0	1	0		1	0	0	1

Valeur décimale
[128+64+32+16+ 8 + 4 + 2 + 1]*256+[128+64+32+16+ 8 + 4 + 2 + 1]

code hexadécimal

8 +4 +2 +1	8 +4 +2 +1	8 +4 +2 +1	8 +4 +2 +1
1	1	2	8 + 1 = 9

Le nombre décimal 4 393 équivaut au nombre hexadécimal 1 129.

Un octet code donc 2 chiffres hexadécimaux. Il faut 2 octets pour coder 4 chiffres hexadécimaux.

A l'inverse, convertissons le nombre hexadécimal E7C3.

code hexadécimal

8 + 4 + 2 =E	4 + 2 + 1 = 7	8 + 4 = C	2 + 1 = 3
8 + 4 + 2 + 1	8 + 4 + 2 + 1	8 + 4 + 2 + 1	8 + 4 + 2 + 1

valeur décimale

[128+64+32+16 + 8 +4 +2 +1]*256+[128+64+32+16 + 8 +4 +2 +1]

code binaire

1	1	1	0		0	1	1	1		1	1	0	0		0	0	1	1

Le nombre hexadécimal E7C3 équivaut au nombre décimal 59 331.

A noter que la conversion en hexadécimal peut se faire en interrogeant votre ordinateur (en basic) grâce à la fonction HEX$ (pour MO5 nécessité du D.O.S.).

Exemple :

PRINT HEX$ (&B0001000100101001) et ENTREE (ou PRINT HEX$(4 393) et ENTREE sur MO5 avec D.O.S.).

Réponse : 1 129.

La numération implicite de votre appareil est la numération décimale. Pour utiliser une autre base, en basic, il faudra utiliser le préfixe &

(ET COMMERCIAL) suivi de B pour binaire ou H pour hexadécimal (&B n'existe pas sur MO5).

Exemple :

PRINT &H1129 + ENTREE

Réponse : 4 393

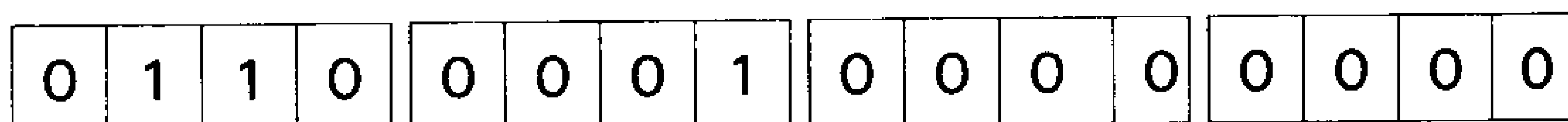
REMARQUE : Ne pas confondre la valeur décimale d'un nombre binaire, avec le décimal codé binaire (BCD), que nous n'utiliserons qu'en fin d'ouvrage. Quand il sera dit décimal, par la suite, comprenez : valeur décimale.

4. **Adresses :** numéro du contenant et valeur du contenu.

Revenons à notre adresse : &H6100. Cette adresse est, rappelons-le, le numéro d'une case-mémoire contenant un octet. Ce numéro de case est un nombre hexadécimal de 4 chiffres, donc il sera codé sur 2 octets.

numéro en hexadécimal 6 1 0 0

numéro en binaire



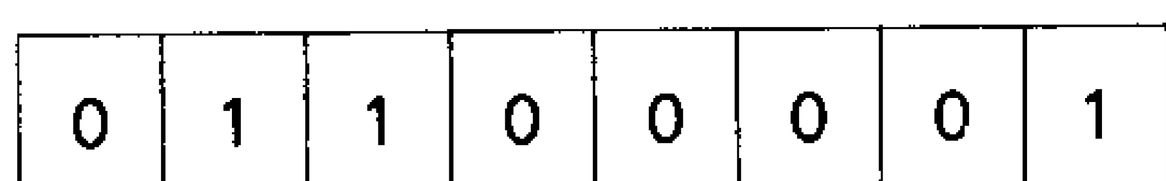
octet de poids fort octet de poids faible

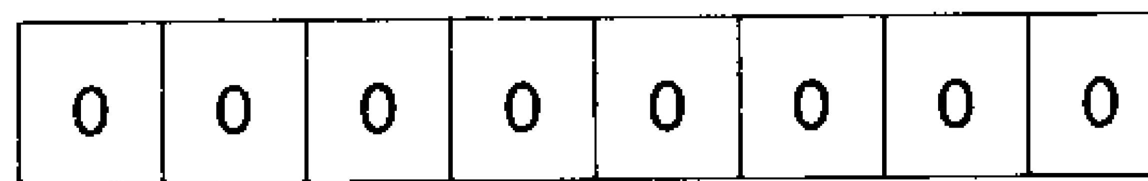
Adresse &H6100 codée sur 2 octets

Cette adresse peut contenir, par exemple, l'information binaire égale à 22 en hexadécimal ou 34 en décimal.

n° d'adresse du contenant :

6 1 0 0



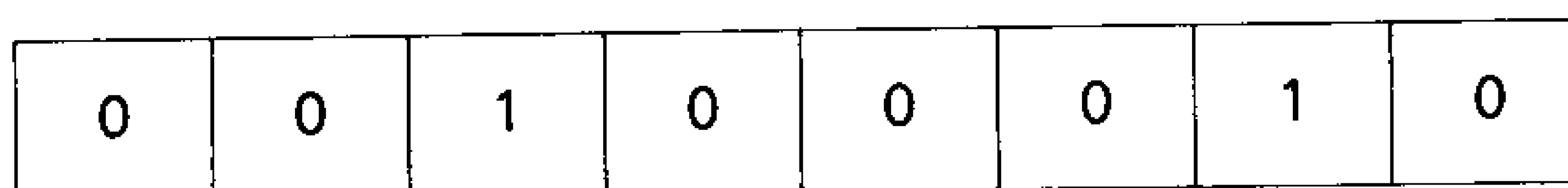


BINAIRE

[128 + 64 + 32 + 16 + 8 + 4 + 2 + 1] × 256 + [128 + 64 + 32 + 16 + 8 + 4 + 2 + 1] DECIMAL

valeur du contenu :

8 +4 +2 +1 8 +4 +2 +1 HEXADECIMAL



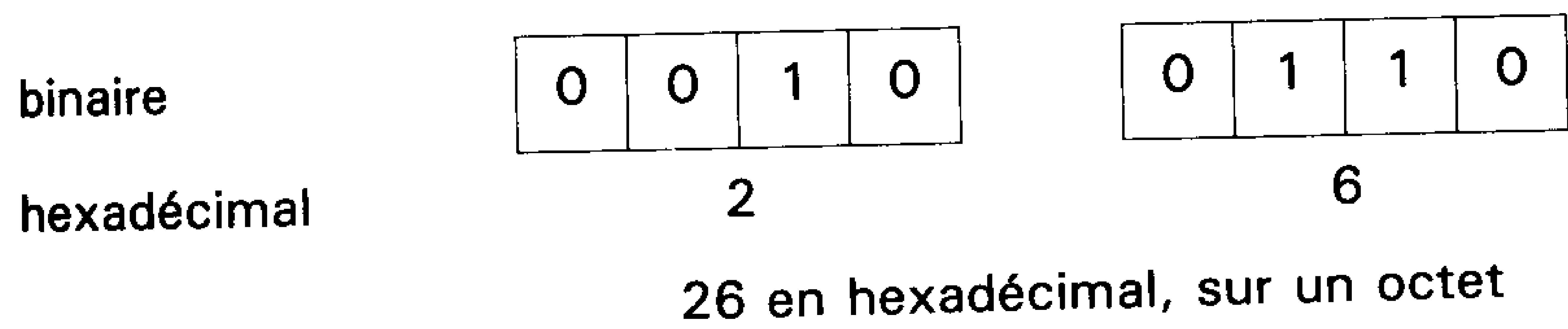
BINAIRE

128 +64 +32 +16 +8 +4 +2 +1 DECIMAL

Adresse &H6100, contenant l'information "22" en hexadécimal ou "34" en décimal.

Comme on peut le constater, l'hexadécimal est plus facilement utilisable que le décimal, pour travailler avec du binaire. En effet, dans l'octet précédent, on voit moins la valeur 34 apparaître que le nombre 22 en hexadécimal : les quatre bits de poids fort représentent les "seizaines" (2) et les quatre bits de poids faible, les unités (2).

Dernier exemple :



26 en hexadécimal, sur un octet

5. Implantation du programme-basic :

A la lumière de ce que nous savons maintenant, essayons de déterminer le début d'un programme basic, dans la mémoire-utilisateur. Ce qui suit n'est pas valable en BASIC 128 (voir fin de paragraphe).

L'adresse du début de ce programme est contenue dans la page 0 du basic.

Pour les TO7, TO7-70 et TO9, il s'agit des registres : &H611C-&H611D.

Pour le MO5 : &H2113-&H2114.

En multipliant la valeur décimale contenue dans l'octet de poids fort par 256, et en y ajoutant la valeur de l'octet de poids faible, nous obtenons :

Pour le TO7(70) : ? PEEK (&H611C)*256 + PEEK (&H611D) → 26101 lorsqu'il n'y a pas de système D.O.S..

Pour le TO9* : ? PEEK(&H611C)*256+ PEEK(&H611D) → 26145 lorsqu'il n'y a pas de système D.O.S..

Pour le MO5 : ? PEEK(&H2113)*256 + PEEK(&H2114) → 9636 lorsqu'il n'y a pas de système D.O.S..

Vérifions-le (sans D.O.S., ou de manière déconnectée sur nano-réseau) :

* Sur TO9, ajouter 44 à chaque adresse des TO7(70). 26101 devient donc 26145.

Tapez NEW, puis :

10 RUN (laissez deux espaces entre 10 et RUN).

Faites :

TO7(70) :*	*MO5 :*	*Signification*
? PEEK (26101)	? PEEK (9636)	adresse de la ligne
? PEEK (26102)	? PEEK (9637)	suivante sur 16 bits
? PEEK (26103)	? PEEK (9638)	0 N° de la 1re ligne
? PEEK (26104)	? PEEK (9639)	10 sur 16 bits
? PEEK (26105)	? PEEK (9640)	32 = code ASCII de : espace
? PEEK (26106)	? PEEK (9641)	136 = code-basic de : RUN
? PEEK (26107)	? PEEK (9642)	0 = séparateur de ligne

En 26108-26109 (TO7(70))* ou 9643-9644 (MO5), nous aurions l'adresse du début de la ligne suivante, codée sur 2 octets, et ainsi de suite...

RUN = 136 en code-basic. Chaque instruction connue de la cartouche-basic (TO7(70)) ou du basic intégré (MO5, TO9) a ainsi un numéro de code. Celui-ci permet au basic de savoir quelle orientation il doit prendre (après avoir détecté une éventuelle erreur).

A noter qu'un espace, entre le numéro de ligne et la première instruction de cette ligne, n'est pas comptabilisé.

En BASIC 128, sur TO7-70 et TO9, le programme est implanté en BANK1, à partir de l'adresse 40964. Cette adresse est immuable, et n'est pas contenue en page 0. Le BASIC 128 dispose d'ailleurs de 2 fois 16 K. R.O.M. commutables (voir dernier chapitre, partie n° V, sur l'extramon.) et de 2 pages de base (&H6100-&H62FF). Le repérage des lignes est différent : après la ligne d'instructions codées vous avez, entre deux délimiteurs (0), un nombre permettant de calculer la longueur de la ligne suivante.

6. Arithmétique binaire :

L'arithmétique binaire peut paraître au début un peu rébarbative. Nous nous contenterons donc d'étudier deux opérations de base : l'addition et la soustraction. Pour ceux qui souhaiteraient davantage d'informations, on ne peut que leur recommander l'ouvrage de M. Bui Minh Duc, cité en bibliographie.

Jusqu'ici, nous avons toujours travaillé en arithmétique dite non signée

* Sur TO9, ajouter 44 à chaque adresse des TO7(70). 26101 devient donc 26145.

(c'est-à-dire dans l'absolu), où &B11111111 = 255 en décimal, ou &HFF. En arithmétique signée, on considère le bit numéro 7 comme bit de signe : positif si le bit numéro 7 est à 0, et négatif si le bit numéro 7 est égal à 1. Dans ces conditions, il ne reste que 7 bits utilisables qui permettront 128 possibilités d'écriture (2^7). On pourra donc coder sur un octet tous les nombres compris entre – 128 (10000000) et +127 (01111111). A ce sujet, voyez le tableau numéro 4, qu'il peut être utile de photocopier pour l'avoir rapidement sous la main en cas de nécessité. Les nombres négatifs sont dits complémentés à deux, nous en reparlerons.

Pour un nombre positif supérieur à &H7F ou + 127 en décimal il faudra, en arithmétique signée, deux octets pour le coder.

Pour un nombre signé, codé sur 16 bits, le bit de signe est le n° 7 de son octet de poids fort.

a. Addition de 2 nombres binaires, *non signés* :

Mais revenons tout d'abord à l'arithmétique non signée, et étudions une addition :

L'addition des nombres décimaux se réalise au moyen de tables d'addition. Il en va de même pour les nombres binaires et celles-ci sont très simples :

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

Exemple :

		7	6	5	4	3	2	1	0
nos des bits :									
11 hexadécimal ou 17 décimal		0	0	0	1	0	0	0	1
23 hexadécimal ou 35 décimal	+	0	0	1	0	0	0	1	1
34 hexadécimal ou 52 décimal	=	0	0	1	1	0	1	0	0

Il suffit d'additionner bit à bit en se référant à la table :

bit n° 0 : 1 + 1 = 10. On pose 0 et on retient 1.
bit n° 1 : 1 (retenue) + 0 + 1 = 10. On pose 0 et on retient 1.
bit n° 2 : 1 (retenue) + 0 + 0 = 1.
bits nos 3, 6 et 7 : 0 + 0 = 0.
bits nos 4 et 5 : 1 + 0 = 1.

La retenue d'octet :

Si les deux bits n° 7 à additionner étaient à 1, ou si un de ces deux bits était à 1 et qu'il y avait retenue sur ce même bit, il y aurait alors une retenue d'octet que le processeur mettrait dans un indicateur spécial, appelé CARRY (C). Cet indicateur ou FLAG serait donc positionné à 1. La retenue, mise dans "C", pourrait ensuite être reportée sur l'octet de poids fort.

A propos des FLAG du processeur, sachez qu'il existe aussi un indicateur de résultat nul : ZERO FLAG (Z). Ce « drapeau » se positionne à 1 lorsqu'un résultat égale zéro.

De même, il existe un indicateur de « demi-retenue » ou HALF-CARRY (H). Celui-ci se positionne à 1 lorsqu'il y a retenue d'un bit n° 3 vers un bit n° 4 (au milieu de l'octet). Ce dernier FLAG n'est utile que pour les nombres décimaux codés en binaire. Ne confondons toujours pas la valeur décimale d'un nombre binaire et le BCD (BINARY CODED DECIMAL). Le décimal codé binaire ne nous sera utile qu'en fin d'ouvrage ; sachez seulement qu'il s'agit du même code que celui de l'hexadécimal, mais sans les chiffres A, B, C, D, E, F. Le binaire comporte donc des combinaisons, inutilisables dans ce code. Ce dernier n'a pas de correspondance, en valeur, avec les nombres qu'il représente. Partant, il serait plus juste de parler du codage des chiffres d'un nombre décimal, que du codage du nombre lui-même.

b. Addition de 2 nombres binaires *signés* :

Si les bits numéro 7 sont à zéro, et s'il n'y a pas de retenue d'un bit numéro 6 vers un bit numéro 7, cela revient à faire une addition en arithmétique non signée.

Par contre, en mode signé, toute erreur de signe sur le bit numéro 7 du résultat provoque la mise à 1 d'un indicateur appelé OVERFLOW (V), pour indiquer le débordement du calcul sur 8 bits et donc, un résultat faussé.

Lorsque le bit numéro 7 du résultat est égal à 1, indiquant la négativité de ce dernier, un indicateur, NEGATIVE FLAG (N), se positionne également à 1.

Il faut remarquer qu'en arithmétique non signée l'opération est la même, mais qu'on ne s'occupe pas des indicateurs "V" et "N". C'est donc au programmeur de savoir dans quel mode il travaille :

Exemple :

Binaire :	*Décimal :*
0111 1111	127
+ 0111 1111	+ 127
= 1111 1110	= 254

Ici : 1 (retenue) + 1 + 1 = 10 + 1 = 11. On pose 1 et on retient 1.

Dans le cas présent :

"N" se positionne à 1
"V" se positionne à 1.

Il faudra donc travailler sur deux octets : 00000000 11111110.

A retenir que l'indicateur à surveiller en arithmétique non signée est "C" (retenue externe), et qu'en arithmétique signée c'est "V" (retenue interne). Il faut travailler sur 16 bits dans le cas d'une retenue d'octet, ou d'un débordement en mode signé.

c. Les compléments :

Toute soustraction de type A moins B peut être considérée comme étant l'addition de A et de l'opposé de B. Pour ce faire, en binaire, on complémente à deux le nombre à soustraire. Le complément à 2 d'un nombre est en effet l'opposé de ce nombre.

Complément à un :
C'est le « symétrique » d'un nombre binaire.

Exemple : 10 hexa = 00010000 et son complément à un est : 11101111

Complément à deux :

C'est le complément à un, auquel on ajoute 1.

Exemple :
10 hexa = 00010000 et son complément à deux est : 11101111 + 1, soit : 11110000.

Donc, 10 hexa = 16 en décimal et F0 hexa = – 16 en décimal (revoir le tableau numéro 4) sont des nombres opposés. Leur somme est égale à zéro, la retenue d'octet étant à inverser lors d'une addition avec un nombre complémenté à deux, c'est-à-dire négatif.

d. Soustraction de 2 nombres binaires :

Considérons les 2 façons de poser une soustraction :

1er exemple : le mode-programmeur

hexadécimal :	*décimal :*
&H4B	75
– &H22	– 34
= &H29	= 41

C'est de cette manière que vous travaillerez.

Les indicateurs "N", "V" et "C" sont affectés par une soustraction ; ici, ils seront à zéro après l'opération.

2e exemple : le mode-processeur (réservé à ce dernier).

En complémentant &H22 à deux, on obtient : &HDE. Additionnons &H4B et &HDE.

binaire :

```
       01001011
     + 11011110
   ------------
 =1)   00101001
```

Le résultat est le même, mais il existe une retenue externe.

C'est de cette manière que travaille le processeur ; il inverse toutefois systématiquement la retenue "C", pour vous donner le résultat de l'exemple n° 1 (avec "C" = 0).

Vous pouvez opter pour ce 2e mode de travail, mais vous retomberez alors dans le cas d'une addition, et "C" sera égal à 1 pour vous.

e. Conversion hexadécimal-décimal :

Outre la méthode qui consisterait à passer par le binaire, voici ce que l'on peut faire (à l'aide du tableau numéro 3) en décomposant le nombre hexadécimal à convertir :

Exemple : convertir &H16A2 en décimal

hexadécimal :	conversion :	décimal :
1000	16^3*1 (4096*1)	4096
+ 600	16^2*6 (256 *6)	+ 1536
+ A0	16^1*A (16 *10)	+ 160
+ 2	16^0*2 (1 *2)	+ 2
= 16A2		= 5794

f. Conversion décimal-hexadécimal :

On divise par 16 le nombre à convertir autant de fois qu'il est possible, et le nombre converti est formé par le dernier quotient suivi de tous les restes des autres divisions, dans le sens inverse.

```
5794  | 16
  99  |----
  34  | 362 | 16
   2  |  42 |----
      |  10 | 22 | 16
      |     |  6 |----
      |     |    |  1
< ---------------------
   2     A    6    1
```

A noter que ces 2 méthodes de conversion sont valables pour toutes les bases.

REMARQUES :

Les nombres signés nous seront utiles lors des branchements (GOTO et GOSUB du basic), ainsi que lors des déplacements dits signés : un nombre positif branche vers l'avant d'un programme et un nombre négatif branche vers l'arrière d'un programme.

Il existe des instructions fonctionnant uniquement en mode signé, et d'autres en mode non signé. Certaines instructions fonctionnent dans les deux modes.

C. LE CODE ASCII

ASCII signifie : AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE.

Le code ASCII (tableau numéro 5) sert essentiellement à lire le clavier, ou à envoyer des lettres, chiffres, etc., vers un périphérique. On trouve de 0 à &H1F les caractères de contrôle, et de &H20 à &H7F les caractères affichables.

D. LE LANGAGE-ASSEMBLEUR

Il n'est pas question ici de décrire la globalité de ce langage, d'autres livres ont été conçus pour cela et les tableaux numéro 8 et 9 sont complets à ce sujet. Nous nous contenterons donc pour l'instant d'explications générales ; le reste sera détaillé au fur et à mesure d'applications pratiques.

Le dernier paragraphe de cette première partie comportera enfin quelques explications sur la cartouche-assembleur de votre appareil, en complément du livre de référence qui l'accompagne.

1. Les mnémoniques :

Il serait fastidieux pour l'être humain de programmer avec des chiffres, fussent-ils hexadécimaux. Il apparaît donc plus raisonnable de travailler avec des instructions plus compréhensibles : les mnémoniques. Les mnémoniques sont des abréviations d'un langage, anglais bien sûr.

Exemple :

LOAD A (c'est-à-dire charge l'accumulateur A* avec une valeur X) s'écrit avec le mnémonique LDA, dont le code hexadécimal varie selon le type d'adressage (ou mode de chargement de l'accumulateur).

STORE A (c'est-à-dire range la valeur contenue dans "A", à une adresse donnée) s'écrit : STA, dont le code varie aussi selon le type d'adressage (c'est-à-dire ici la manière dont on va déposer la valeur contenue dans "A", à une adresse donnée).

2. L'opérande :

Le mnémonique était le code-opération. L'opérande représente quant à lui la valeur à ranger dans "A", ou l'adresse à laquelle "A" va déposer cette valeur, selon qu'on utilise l'instruction LOAD ou STORE.

Exemple :

LDA #$24

Par cette instruction, vous demandez au processeur de charger l'accumulateur A avec la valeur hexadécimale 24. Valeur, car l'adressage se fait avec un "#". On appelle ce type d'adressage : immé-

* L'accumulateur A est un registre ou case-mémoire interne du microprocesseur.

diat. "$" signifie hexadécimal en langage d'assembleur, et remplace le "&H" du basic.

3. L'adressage :

L'adressage est contenu dans la zone de l'opérande, séparé du code-opération appelé aussi OP-CODE par un certain espace. Le signe de l'adressage, quand il est inscrit, doit être « collé » à l'opérande.

Les types d'adressage sont très variés, nous les étudierons au fur et à mesure des programmes qui vont suivre.

Exemples :

: adressage immédiat. Il permet le chargement d'une valeur, « immédiatement », sans « intermédiaire ».

> : adressage étendu (le signe ">" n'est pas réellement obligatoire. Lorsqu'on le supprime, cela signifie la même chose). Il s'étend à toutes les adresses de la mémoire.

Exemple :

Instructions	Programme-objet = hexadécimal ou machine		Programme-source = assembleur		
n° 1	86	24		LDA	#$24
n° 2	B7	7FFF		STA	>$7FFF
			ou	STA	$7FFF

Le programme-objet est le même que le programme-source, mais écrit uniquement en hexadécimal, donc en langage-machine. A chaque mnémonique correspond un code (tableau numéro 8). "86" est le code hexadécimal ou machine de LDA en adressage immédiat, et "24" reste le code-opérande.

Instruction numéro 1 :

Demande est faite au processeur de charger « immédiatement » l'accumulateur A avec la valeur hexadécimale 24.

Instruction numéro 2 :

Demande est faite au processeur de déposer la valeur contenue dans l'accumulateur A à l'adresse : $7FFF.

Testons ce programme :

Pour ce faire nous allons travailler en basic, avec un petit programme

dont la structure reviendra toujours par la suite (seules les données en DATA et les adresses d'implantation du programme changeront).

```
10 FOR I= 32000 TO 32000+5
20 READ A$
30 POKE I,VAL("&H"+A$)
40 NEXT I
50 DATA 86,24,B7,7F,FF,39
```

Adresses d'implantation : de 32000 à 32000 + 5, ce qui fait six adresses puisqu'il y a six données en DATA.

Ce programme permet d'utiliser l'hexadécimal, par la concaténation de deux chaînes de caractères ("&H" +A$), en basic (A$ représentant la « valeur » d'une donnée en DATA).

Il faut noter que l'adresse $7FFF se code sur deux octets. Une donnée en DATA correspondant à un octet, il faudra deux données en DATA pour coder une adresse :

32003 contiendra : &H7F
32004 contiendra : &HFF

Le dernier code de la ligne 50 est 39 en hexadécimal ; il signifie RTS, c'est-à-dire : retour de sous-programme. RTS rend ici la main au basic et donc à l'utilisateur. On ne peut le négliger, sous peine de se « planter ».

Après avoir écrit ce programme en basic, faites : RUN. Le programme-machine, écrit en DATA, va s'implanter entre 32000 et 32005. Vérifions-le :

```
      *
? HEX$ (PEEK(32000))
```

Réponse : 86

"86" est bien le code hexadécimal ou machine de la 1re instruction ou mnémonique : LDA en adressage immédiat, etc..

Maintenant, vous pouvez taper : NEW. Le programme-machine reste bien implanté, invisible. Le listing est en effet blanc, mais les adresses à partir de 32000 contiennent toujours les codes hexadécimaux qui y ont été préalablement déposés.

* Sur MO5 : avec le D.O.S. seulement, sinon il vous faut faire une conversion « hexadécimal-décimal ».

Faites : EXEC 32000. L'exécution a eu lieu, vous recevez : O.K.. Que s'est-il passé ? *

Faites : `? HEX$ (PEEK( &H7FFF))`

Réponse : 24

Voilà, la valeur 24 en hexadécimal est arrivée à destination.

Cela eût pu être fait par POKE &H7FFF, &H24, mais ici l'implantation s'est faite en langage-machine.

Deux opérations ont donc été nécessaires en basic : RUN, représentant l'implantation du programme-machine, et EXEC pour l'exécution de celui-ci.

E. CARTOUCHES-ASSEMBLEUR, EN BREF

Ce paragraphe est surtout destiné à ceux qui possèdent une cartouche-assembleur de type TO TEK INTERNATIONAL pour TO7(70) et TO9, ou MO5. Il existe des différences pour le logiciel MO5 de type ODIN, nous en parlerons peu. Ceux qui ne possèdent pas cette cartouche doivent quand même lire ce qui suit avec intérêt.

Mettez votre cartouche-assembleur dans l'appareil et allumez-le.

Sur TO7(70) ou TO9 vous découvrez : 6809-langage modulé en 1. Appuyez sur 1, vous voilà au menu. Sur MO5 vous accédez directement au menu :

1 : éditeur } ces deux numéros nous intéressent plus particulièrement.
2 : moniteur }
3 : directory : pour savoir ce qu'il y a sur vos disquettes.
4 : copy : pour recopier un fichier.
5 : rename : pour changer le nom d'un fichier.
6 : kill : pour détruire un fichier.
7 : format : pour formater ou initialiser une disquette.
8 : printer columns : pour choisir le nombre de colonnes (40 ou 80) de l'imprimante.

* Sur MO5 : avec le D.O.S. seulement, sinon il vous faut faire une conversion « hexadécimal-décimal ».

Tapez : 1 : vous voyez un bandeau jaune apparaître, dans lequel vous pourrez plus tard écrire le nom d'un fichier à charger.
Faites ENTREE : la page est noire car l'éditeur est vide, mais dans le bas de cette page se trouve un carré bleu dans un bandeau jaune. Ce carré bleu signifie que vous êtes en Commande d'éditeur.

- *X = EXIT :*

Donnez la commande EXIT (sortie) en tapant X puis ENTREE. Vous voilà sorti vers le moniteur qui, lui, travaille en binaire, alors que l'éditeur travaille en assembleur. Pour revenir à l'éditeur, après le " # ", vous pourriez faire : X et ENTRÉE. Le " # " est inscrit à gauche, sur une ligne se trouvant au-dessous de celle des registres du processeur dont nous allons parler ensuite : [PC A B DP CC X Y U S].

- Q = QUIT :

Tapez Q et ENTRÉE : vous êtes revenu au menu. Tapez 2 puis ENTRÉE : vous voici de nouveau dans le moniteur. Essayez encore : Q et ENTRÉE, puis tapez 1 et ENTRÉE pour être dans l'éditeur.

Voilà quelques notions pratiques. Vous êtes donc maintenant dans la page-éditeur, où vous assemblerez votre premier programme. Vous irez, ensuite, le faire « tourner » dans la page-moniteur.

Faites CNT-C. Le carré bleu, de la page-éditeur, disparaît : vous pouvez écrire votre programme car vous êtes en édition et non plus en commande d'éditeur.

Il existe quatre zones dans la page-éditeur : zone-étiquette/zone opcode/zone opérande/zone des commentaires (= REM en basic).

Il est facile de passer d'une zone à une autre avec le curseur, en tapant sur la barre d'espacement, car il existe une tabulation automatique.

Essayons la barre d'espacement : tapez une fois et suivez le curseur. Tapez encore deux fois et suivez le curseur. Tapez enfin plusieurs fois : la tabulation automatique a disparu dans la zone-commentaire. Cette dernière zone n'est pas prise en compte par le programme ; elle sert à noter les remarques nécessaires à la bonne compréhension de celui-ci, plusieurs mois ou années après sa conception parfois. Il est important d'avoir quelques (seulement quelques) bons commentaires dans cette zone.

La directive ORG sert, au début d'un programme, à définir l'origine (= adresse d'implantation).
De même que l'on utilise maintenant "$" pour l'hexadécimal, on utilisera "&" pour le décimal, ou rien (le nombre étant alors implicitement pris pour un nombre décimal).

Faites ENTRÉE et remontez éventuellement d'un cran avec la flèche : ↑.

Nous allons écrire notre premier programme. Tapez une fois sur la barre d'espacement : vous êtes dans la zone du Code-opération. C'est ici que vous taperez : ORG.

Essayez donc de taper ce petit programme, en respectant scrupuleusement la tabulation (TO7(70) et TO9) :

```
        ORG     &32000      origine
DEBUT   LDA     #$24
        STA     $7FFF       mettre $24 en $7FFF
        SWI
        END
```

L'instruction SWI (SOFTWARE INTERRUPT) est une interruption logicielle, permettant de rendre la main au programmeur en langage-assembleur sur TO7 (70) et TO9, au lieu de &H39 en basic. Sur MO5, SWI a une fonction un peu différente : elle autorise, par l'intermédiaire d'un code, l'accès à une routine ou sous-programme du moniteur-système.

Exemple : (MO5)

- Cartouche TO TEK :

```
SWI
FCB     0
```

- Logiciel ODIN :

```
SWI   #0
```

"0" est le code de la routine en R.O.M. qui, sur MO5, permet de revenir à une page vierge comme lors d'une initialisation en basic, ou au menu en assembleur.

FCB signifie (sur TO7(70), MO5, TO9) : FORM CONSTANT BYTE, c'est-à-dire que cette directive permet d'implanter une constante de 8 bits (ici 0) à l'adresse où elle est programmée. BYTE signifie octet.

Sur MO5, nous n'utiliserons donc pas l'interruption logicielle SWI pour finir un programme, mais un « break » : STOP, qui n'existe pas sur

TO7(70) ou TO9. Votre premier programme, sur MO5, sera le même que pour les TO7(70) ou TO9, sauf pour SWI, remplacée par STOP.

En basic il faut, comme nous l'avons déjà vu, remplacer SWI ou STOP par RTS, dont le code est &H39.

Ceci a pour effet de faire « revenir le programme » à l'adresse qui suit immédiatement EXEC (en mode-programmation) et d'agir comme un JSR (JUMP TO SUBROUTINE). JSR est le GOSUB du langage-machine, et RTS son RETURN. Inutile donc d'insister sur l'absolue nécessité de RTS en basic, pour l'instant.

Note :

Lors d'une erreur de programmation il est peu recommandé, sauf cas de force majeure, d'initialiser l'appareil avec la cartouche-assembleur.

En cas de RESET, enlevez la ou les disquettes du ou des lecteurs, pour que le système ne détruise pas les programmes enregistrés sur celles-ci.

La directive END est nécessaire si vous voulez éviter que l'appareil ne vous signale une erreur lors de la phase d'assemblage. Cette directive ne compte pas plus que ORG, dans le programme-machine final issu de cet assemblage, et mentionne à la cartouche la fin du programme à assembler. Ce qui suit END ne sera donc pas pris en compte.

Vérifiez que votre programme est bien tapé, dans les bonnes zones. S'il existe un espace entre deux lignes horizontales, cela n'a pas d'importance ; vous pouvez d'ailleurs le supprimer en tapant CNT-X au tout début de cet espace. Attention, n'effacez cependant aucune ligne du programme, sinon il vous faudrait faire ENTRÉE sur la ligne précédente pour créer un espace et récrire la ligne effacée.

Tapez CNT-C : vous êtes revenu en Commande d'éditeur. Tapez : A (assemblage) et faites ENTRÉE. Le programme s'inscrit : à gauche et en jaune le programme-machine, et à droite le programme-assembleur en bleu. L'assemblage correspond au RUN du programme-basic précédent, permettant l'implantation des codes-machine dans la R.A.M..

En jaune, vous voyez les adresses en hexadécimal où sont implantés les codes et valeurs, eux-aussi en hexadécimal, du programme. Seule la première adresse de chaque ligne est mentionnée, même si cette ligne demande 2, 3, 4 ou 5 octets.

Vous avez donc ceci :

TO7(70) et TO9 :

```
(origine :         7D00                     ORG  &32000
ne compte pas)
(début réel)       7D00 86 24     DEBUT     LDA  #$24
                   7D02 B7 7FFF             STA  $7FFF
                   7D05 3F                  SWI
(ne compte pas)            0000             END
```

MO5 :

en $7D05 la ligne est différente :

```
7D05  BD  B000    STOP
```

Le STOP correspond à un branchement (BD) du programme en $B000, ce qui permet de rendre la main à l'utilisateur.

- TO7(70), TO9 et MO5 :

Le nombre d'erreurs est affiché au-dessous : en principe « 00000 TOTAL ERRORS ». En cas d'erreur, signalée en caractères rouges à l'assemblage du programme, il vous faudra effectuer les corrections nécessaires, puis assembler une nouvelle fois.

La Table des symboles (étiquettes) se trouve tout à la fin. Ici, une seule étiquette à signaler : DEBUT en $7D00.

Suivez les conseils inscrits dans le bandeau rouge. Par exemple « Press any key », c'est-à-dire : appuyez sur une touche quelconque.

Voilà, votre programme est implanté (comme RUN précédemment).

Faites X puis ENTRÉE. Vous sortez vers le moniteur. Derrière le "#" se trouve le curseur. Tapez : G & 32000 ou G $7D00 (G signifie GO, c'est-à-dire : départ ; c'est le EXEC du programme-basic précédent) et faites ENTRÉE. Rapidement arrive le message : "8 BRK 7D05". Ce message signifie : BREAK en $7D05, là où nous avons placé, sur TO7(70) et TO9, une interruption logicielle (c'est-à-dire une interruption programmée, par oppositon à une interruption matérielle), ou un break sur MO5.

Vérifions que $24 est bien en $7FFF, en tapant derrière le "#" où se trouve le curseur : N1 ou N et ENTRÉE (c'est-à-dire : demande d'une valeur numérique dans un octet). Après le "#" du dessous tapons : $7FFF ou 7FFF puis "/".

Nous voyons apparaître :

— en bleu : 7FFF/
— en jaune : 24

Faites ENTRÉE et tapez ID (INPUT DECIMAL) et ENTREE. Ceci va nous permettre de demander, derrière un dièse : &32000/ ou 32000/, et la réponse sera : 86. C'est bien la valeur implantée en $7D00, c'est-à-dire en 32000.

Faites encore ENTRÉE et tapez : 32767/. La réponse est : 24. C'est bien la valeur déposée en $7FFF, c'est-à-dire en 32767.

Faites ENTRÉE. Nous sommes toujours en "ID". Tapez : 32767= . La réponse est : 7FFF. L'ordinateur a converti la valeur décimale 32767 en valeur hexadécimale : $7FFF.

Faites ENTRÉE et revenez dans l'éditeur en tapant X derrière un " # ", puis ENTRÉE.

Quelques autres notions pratiques en commande d'éditeur :

Les instructions qui suivent demandent toutes une validation par la touche ENTRÉE.

— B vous donne la fin d'un listing (BOTTOM), et T le début (TOP).
— Pour effacer un programme-assembleur, il faut taper en commande d'éditeur : N (NEW). La question vous est posée : « ARE YOU SURE Y/N ? » (êtes-vous sûr ?). Tapez alors Y (YES) pour que le programme s'efface, ou N (NO) dans le cas contraire.

— Pour sauvegarder, ou charger, sur cassette ou disquette un programme-assembleur, il faut faire ceci :

1. Sur cassette :

SC : Nom du programme. ASM
c'est-à-dire : S = SAVE ; C = CASSETTE ; ASM = ASSEMBLEUR.
ou LC : Nom du programme. ASM
c'est-à-dire : L = LOAD, donc chargement.

Pour la cassette, la question vous est posée : « CASSETTE READY Y/N ? » (cassette prête ?).

Répondez : Y (YES) ou N (NO).

2. Sur disquette (sauf sur nano-réseau) :

SO : nom du programme. ASM
ou LO : nom du programme. ASM
où O représente le numéro du lecteur concerné.

(Il est possible d'effectuer ces mêmes opérations dans le moniteur-binaire, pour des programmes-binaire (suffixe .BIN), derrière un dièse).

– Pour sauvegarder un programme en binaire, en même temps que l'assemblage (vérifiez qu'il n'y a pas d'erreur à l'assemblage auparavant) faites :

1. Sur cassette :

AC : nom du programme. BIN
où A signifie ASSEMBLE, C : SUR CASSETTE, et BIN : BINAIRE.

En effet, l'assemblage génère un programme-objet, en binaire, c'est-à-dire en langage-machine.

2. Sur disquette (sauf sur nano-réseau) :

A0 : nom du programme. BIN

Les programmes en binaire, et non ceux en assembleur bien sûr, peuvent être réutilisés et implantés aux mêmes adresses, en basic, par un LOADM ''Nom de programme. BIN'', 0. Ceci est particulièrement intéressant pour ceux qui travaillent sur nano-réseau. Ils pourront en effet, à partir d'un programme binaire sauvegardé sur cassette, le réimplanter en mémoire (LOADM''CASS:) et le sauvegarder ensuite sur la disquette du réseau (SAVEM''Nom du programme. BIN'', adresse de début, adresse de fin, 0).

Pour charger un programme directement au menu :

A/LP. On peut éviter l'impression de la table des symboles, par l'option NS (NO SYMBOL) : A/LP/NS.

– Pour charger un programme directement au menu :

1. Programme-assembleur :

Tapez, dans le bandeau jaune de l'éditeur : ''0 : nom du programme. ASM'', sans L, pour une disquette sur lecteur n° 0, et ''C : nom du programme. ASM'', sans L, pour une cassette.

2. Programme-binaire :

Tapez, dans le bandeau jaune du moniteur-binaire : ''0 : nom du programme. BIN'', sans L, pour une disquette sur lecteur n° 0, et ''C : nom du programme. BIN'', sans L, pour une cassette.

Voilà quelques notions, parmi les plus utiles à connaître, sur les cartouches-assembleur de type TO TEK. Pour le reste, reportez-vous à votre manuel de référence.