*Table des matières*

*Note de l'auteur*

*Avertissement*

**I. PARTIE: INITIATION AU SATURN**

**A. Présentation**

1. Introduction

2. Fonctionnement

3. Les registres du SATURN

**B. Les registres de calcul**

1. Les champs

2. Affectation d'une valeur

3. Opérations des registres de calcul

**C. Les registres de sauvegarde:**

1. Copie d'un registre de travail vers un registre de sauvegarde

2. Copie d'un registre de sauvegarde vers un registre de travail

3. Copie d'un champ d'un registre de travail vers un registre de sauvegarde

4. Copie d'un champ d'un registre de sauvegarde vers le champ d'un registre de travail

5. Echange du contenu d'un registre de travail avec le contenu d'un registre de sauvegarde

6. Echange d'un champ d'un registre de travail avec le même champ d'un registre de sauvegarde

**D. Les registres pointeurs**

1. Pointage et lecture à une adresse en mémoire

2. Déplacement à gauche et à droite d'un pointeur

3. écriture à une adresse en mémoire

4. Echanges entre registres pointeurs et registres de travail

**E. Les sauts et tests**

1. Les sauts

2. Les tests

**II. PARTIE: NOTIONS DE BASE**

**A. Les routines en mémoire morte :**

**B. Adresses utiles de la mémoire vive :**

**C. Qu'est-ce qu'un programme SOURCE ?**

**D. Comment le rendre exécutable ?**

**E. Structure de base d'un programme :**

**F. Comment tester une touche ?**

1. Les couples OUT/IN:

**G. Squelette d'un jeu**

1. Avertissement

2. Mise en forme

**H. Comment émettre un son ?**

1. Procédons par étape successives:

**I. Comment afficher un GROB ?**

1. Pour modifier une donnée à l'écran:

2. Pour écrire une donnée à l'écran:

3. Pour afficher un GROB sur tout l'écran

**J. Gérer les sprites**

1. Comment afficher un sprite ?

2. Comment déplacer un sprite ?

**K. Un petit jeu !**

**III. POUR LE PROGRAMMEUR CONFIRME:**

1. Avertissement

**B. Déplacer un sprite II**

**C. Afficher un compteur**

**D. Le double-écran**

**E. Les niveaux de gris**

**IV. APPLICATIONS LUDIQUES**

**A. Création de Mariohp:**

**B. Création de Méganoid:**

**V. Lexique:**

*Note de l'auteur*

*Parce les ouvrages consacrés à l'assembleur sont de plus en plus nombreux et que la plupart rebutent le programmeur par leur complexité, j'ai décidé moi, programmeur amateur de jeu sur HP48, de publier dans le présent ouvrage, tout mes conseils et astuces concernant l'élaboration d'un jeu en assembleur sur HP48. Ce livre ne prétend nullement remplacer les précieux ouvrages existants sur l'assembleur, mais d'apporter au programmeur débutant, tout le savoir faire acquis au prix de nombreuses heures de programmation et d'études de programmes. Ayant eut la chance d'aborder l'assembleur très tôt (en classe de seconde exactement) je pense que mes explications seront d'avantages compréhensibles par tous, car je connaît les réelles difficultés du programmeur débutant.*

*Bon courage à tous, et n'oubliez pas que tous savoir ne doit jamais rester secret, que ce soit en programmation ou ailleurs.*

*Julien Meyer.*

*Avertissement*

La programmation en assembleur est difficile car elle nécessite une parfaite connaissance d'une foule de notions, que nous allons étudier dans cet ouvrage. Elle offre en contrepartie les meilleurs jeux qui soient, fluides et rapides, le tout accompagné d'un gain relatif de l'espace mémoire. Pour programmer en assembleur d'une manière plus commode, le programmeur se devra dans l'obligation d'utiliser un éditeur de texte, ainsi qu'un compilateur (ou assembleur), fournit avec la disquette du présent ouvrage. Le savoir faire de la programmation en assembleur se gagne plus par la persévérance et l'enthousiasme que dans l'apprentissage fastidieux d'un tableau de mnémoniques. Les ouvrages sur l'assembleur existent pour vous faciliter la tache, mais ils ne peuvent vous apporter l'inspiration et la passion de la programmation. C'est pourquoi n'ayez pas peur de passer plusieurs nuits blanches pour seulement afficher un GROB, tout les programmeurs sont passés par là !

# PARTIE: INITIATION AU SATURN

## Présentation

### Introduction

Le SATURN, c'est le nom que l'on donne au microprocesseur de la HP48. Cette 'puce' est celle qui équipe de nombreux modèles de HP.

Le SATURN est très performant: Pour comparer avec votre console de jeux vidéo, le SATURN est dit 4 bits (il permet même parfois de travailler avec 64 bits!). Pour comparer avec votre ordinateur personnel, Le SATURN est cadencé par une horloge interne de 2 Mhz pour les HP48S(X), et 4 Mhz pour les HP48G(X), ce qui permet de réaliser tout plein de programmes rapides et performants! Mais avant de se lancer dans la programmation en assembleur, il nous faut étudier le fonctionnement interne du SATURN.

### Fonctionnement

Le SATURN est avant tout un microprocesseur. Comme la plupart des microprocesseur, il ne sait 'lire' qu'une suite de 0 et de 1. On appelle cela le binaire. En binaire, on ne connaît que les 0 et les 1. Il est inconcevable de programmer en binaire. On aura donc recourt à l'hexadécimal. En hexadécimal, on à 16 chiffre à notre disposition, (0123456789ABCDEF) contrairement au binaire, ou on en a 2, (01) et au décimal, ou on en a 10 (0123456789). L'emploi de la base Hexadécimal n'est pas une fantaisie des programmeurs, mais permet une meilleur lecture de ces 0 et 1. En effet, quatre chiffres binaires, (ex: #0101b) peuvent être regroupés dans un seul chiffre hexadécimal, (ex #5h). Un chiffre binaire (0 ou 1) est aussi appelé: bit. On appelle une suite de 4 bits: un quartet. (Un octet étant une suite de 8 bits).

Pour nous résumer:

*Un fragment de programme en binaire ressemblerai à:*

00110101011100101111010101001100

*Découpons-le en 8 quartets:*

0011 0101 0111 0010 1111 0101 0100 1100

*Transposons ces 8 quartets en 8 chiffres Hexadécimaux:*

(Remarquons que 0111 = #1110h).

C A E 4 F A 2 3

*Ce qui nous donne:*

CAE4FA23

Ce qui est tout de même plus lisible, mais encore incompréhensible !

Une instruction du SATURN correspond à une suite de quelques chiffres hexadécimaux. Par exemple, l'instruction qui éteint l'écran de la HP est codé par: 8FDBB10. L'assembleur est né lorsqu'on s'est dit qu'il serait plus commode d'associer un nom de fonction à chaque instruction du SATURN. En assembleur, on se content d'écrire le nom de la fonction (appelée aussi: mnémonique) dans notre programme. Un outils appelé: ASSEMBLEUR permet de remplacer ces noms de fonction par les codes hexadécimaux correspondants. C'est ainsi que le langage d'assembleur est né. A noter qu'il existe une table ou sont répertoriées toutes les fonctions du SATURN, ainsi que les codes correspondants, on appelle cette table: Table des mnémoniques.

Mais prenons un exemple simple:

Le nom de l'instruction qui permet d'éteindre l'écran est:

GOSBVL 01BBD

Ce qui correspond en fait à 8FDBB10 en langage machine, ou plutôt à:

1000111111011011101100010000 en binaire..

On remarquera que les mnémoniques sont tout de même très utiles !

Ainsi un programme en assembleur n'est rien d'autre qu'une suite de mnémoniques, que l'on doit assembler à l'aide d'un outils appelé 'assembleur', pour pouvoir exécuter ce programme. Mais voyons toutes les fonctions que peut réaliser le SATURN.

### Les registres du SATURN

#### Rappels

Le SATURN calcul en hexadécimal, c'est-à-dire suivante une base de 16 chiffres qui sont: 0123456789ABCDEF. Ainsi, pour écrire #14d dans un programme, nous devrons écrire: #Eh.

Voici le tableau de correspondance entre ces deux bases:

|  |  |
| --- | --- |
| **Base 10** | **Base 16** |
| 012345678910111213141516100255 | 0123456789ABCDEF1064FF |

#### Ses registres

On regroupe les spécificité du SATURN selon plusieurs registres. La notion de registres sera employée à de nombreuses reprises, ainsi soyez attentif.

*Il existe plusieurs sortes de registres:*

Les registres de calcul: A,B,C,D

Les registres de sauvegarde: R0,R1,R2,R3,R4

Les registres pointeurs: D0,D1

Le compteur ordinal: PC

Les entrées et sorties: OUT/IN

La pile des retours: RSTK

Les drapeaux: ST, HST

Le pointeur de champs: P

Analysons chacun de ces registres en détail.

## Les registres de calcul

### Les champs

Il existe 4 registres de calculs: A,B,C et D. S'il peuvent paraître similaire, sachez que les registres: A et C sont les plus importants. Tout les calculs de votre HP passent par ces registres. Il est donc nécessaire de bien comprendre leur fonctionnement. Ces registres peuvent accueillir 64 bits, soit 16 quartets. On se sert des registres A et C pour lire et écrire en mémoire. Seul le contenu des registres A,B,C et D peuvent faire l'objet d'un traitement. L'un des atouts majeur du SATURN nous autorise à limiter ce traitement à une portion donnée de ce registre de traitement, et ce selon le tableau suivant:

|  |
| --- |
| **Division en champs des 16 quartets d'un registre de travail (A,B,C et D)** |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| W |
| S | M | X |
|  | A |
|  | XS | B |

Ayez bien ce tableau à l'esprit, il vous sera très utile par la suite.

#### Le champ W:

Il s'applique à tout les quartets (64 bits), et s'étend donc du quartet 0 au quartet 15. Ce champs est très utile lorsqu'on veut afficher rapidement une image à l'écran, ou faire un échange rapide entre deux zones de la mémoire.

#### Le champ S:

Il ne comprend que le quartet 15.

Il est assez peu utilisé.

#### Le champ M:

Ce champ ne comprend que 12 quartets, et s'étend du quartet 3 au quartet 14. Il est assez peu utilisé.

#### Le champ X:

Ce champ ne comprend que 3 quartets. Il est très utile, notamment lors d'un test du clavier, ou lors de l'activation du buzzer. Ce champ s'étend du quartet 0 au quartet 2.

#### Le champ XS:

Le champ XS est une portion du champ X; qui ne comprend que le quartet 2. Il est très peu utilisé.

#### Le champ B:

Le champ B comprend 2 quartets qui s'étendent du quartet 0 au quartet. Il est très utile lorsqu'on utilise des compteurs, ou des affichages de sprites. Mais il est essentiellement utilisé pour gérer les caractères.

#### Le champ A:

Le champ A comprend 5 quartets qui s'étend du quartet 0 au quartet 4. C'est le champ le plus employé et de loin, en effet, il est essentiellement utilisé lors de la lecture ou le pointage d'adresses en mémoire. Une adresse est codée sur 5 quartets.

Attention à ne pas confondre les champs A et B avec les registres de calcul A et B.

### Affectation d'une valeur

*Que peut-on faire avec ces registres de travail ?*

#### Charger une valeur dans l'un des registres A ou C

LC 01235

LA 14

A noter que l'on ne peut charger que 16 quartets maximum, et que ces nombres sont des nombres hexadécimaux. (#12d = # Ch).

Ex: LC 0123456789ABCDEF

|  |
| --- |
| **Mnémoniques** |
| LA nLC n |

#### Mettre à zéro le champ d'un registre de travail

Le champ devra être spécifié par 'ch'. Le contenu du champ du registre est remplacé par des 0. Pour mettre à 0 l'ensemble du registre, il faut spécifier le champ W (les 16 quartets).

|  |
| --- |
| **Mnémoniques** |
| A=0 chB=0 chC=0 chD=0 ch |

#### Manipuler un bit isolé

On peut armer (allumer) ou désarmer (éteindre) un bit (ou pixel) d'un registre de travail, avec l'instruction ABIT et CBIT. ABIT concerne le registre A, CBIT le registre C.

Ex: ABIT=1 4

(on arme le bit 4 du registre A)

Ex: CBIT=0 6

(On désarme le bit 6 du registre C)

Seulement les 16 premiers bits sont modifiables.

|  |
| --- |
| **Mnémoniques** |
| ABIT=0 nABIT=1 nCBIT=0 nCBIT=1 n |

#### Effectuer un échange des champs entre 2 registres

Vous spécifiez les champs concernés par cet échange avec: 'ch'. Pour échanger la totalité de deux registres, il faut employer le champ W.

Ex: ABEX A

(les contenus des champs A des registres A et B sont échangés)

|  |
| --- |
| **Mnémoniques** |
| ABEX chBCEX chCAEX chDCEX ch |

#### Copie entre deux registres

La copie d'un champ entre 2 registres ne concerne que l'un des deux registres, contrairement à l'échange, ou les 2 registres sont concernés. Pour copier l'ensemble d'un registre vers un autre, il faut employer le champ W.

Ex: A=C A

(le champ A du registre C est copié sur le champ A du registre A).

La copie s'effectue toujours de droite à gauche.

|  |
| --- |
| **Mnémoniques** |
| A=B chB=A chB=C chC=B chC=A chA=C chD=C chC=D ch |

### Opérations des registres de calcul

Les registres de calcul A,B,C et D nous permettent d'effectuer des opérations simples, comme des additions, des soustractions, des opérations logiques comme le "&" et le "!" logique.

#### Les additions

Il nous est possible d'additionner deux registres entre eux et d'ajouter une valeur quelconque à un registre.

##### Addition des champ de deux registres

Le champ devra être spécifié par: 'ch'.

Ex: A=A+C A

(les champs A des registres A et C sont additionnés, le résultat est copié dans le champ A du registre A).

|  |
| --- |
| **Mnémoniques** |
| A=A+B chB=A+B chB=B+C chC=B+C chC=C+A chA=C+A chD=D+C chC=D+C ch |

Les mnémoniques suivantes permettent de doubler (\*2) les valeurs des champs des registres.

|  |
| --- |
| **Mnémoniques** |
| A=A+A chB=B+B chC=C+C chD=D+D ch |

##### Addition d'une constante à un champ d'un registre

Il nous est possible d'ajouter une valeur comprise entre 0 et 17 (0<n<17) à un champ spécifié par 'ch' d'un registre.

Ex: A=A+16 A

(on ajoute 16 au champ A du registre A).

|  |
| --- |
| **Mnémoniques** |
| A=A+n chB=B+n chC=C+n chD=D+n ch |

#### Les soustractions

Il nous est possible de soustraire deux registres entre eux et de soustraire une valeur quelconque à un registre.

##### Soustraction des champ de deux registres

Le champ devra être spécifié par: 'ch'.

Ex: A=A-C A

(les champs A des registres A et C sont soustrait, le résultat est copié dans le champ A du registre A).

|  |
| --- |
| **Mnémoniques** |
| A=A-B chB=B-C chC=C-A chD=D-C chB=B-A chC=C-B chA=A-C chC=C-D chA=B-A chB=C-B chC=A-C chD=C-D ch |

##### Soustraction d'une constante à un champ d'un registre

Il nous est possible de soustraire une valeur comprise entre 0 et 17 (0<n<17) à un champ spécifié par 'ch' d'un registre.

Ex: A=A-16 A

(on soustrait 16 au champ A du registre A).

|  |
| --- |
| **Mnémoniques** |
| A=A-n chB=B-n chC=C-n chD=D-n ch |

##### Complément à 2

|  |
| --- |
| **Mnémoniques** |
| A=-A chB=-B chC=-C chD=-D ch |

##### Complément à 1

Le complément à 1 est utilisé pour inverser le contenu d'un champ d'un registre.

Ex: A=-A A

(on inverse la valeur contenu dans le champ A du registre A)

|  |
| --- |
| **Mnémoniques** |
| A=-A-1 chB=-B-1 chC=-C-1 chD=-D-1 ch |

#### Les opérations logiques

##### Le "!" logique

Représentation du "ou" logique:

|  |  |  |
| --- | --- | --- |
| n1 | n2 | résultat |
| 0011 | 0101 | 0111 |

On utilise beaucoup cette méthode dit du "ou" logique lorsqu'on affiche un sprite par 'transparence'. Nous étudierons cette méthode plus tard.

|  |
| --- |
| **Mnémoniques** |
| A=A!B chB=B!C chC=C!A chD=D!C chB=A!B chC=B!C chA=C!A chC=D!C ch |

##### Le "&" logique

Représentation du "et" logique:

|  |  |  |
| --- | --- | --- |
| n1 | n2 | résultat |
| 0011 | 0101 | 0001 |

On emploi le plus souvent cette méthode dite du "et" logique lorsque l'on test une touche du clavier. Nous étudierons cela plus tard.

|  |
| --- |
| **Mnémonique** |
| A=A&B chB=B&C chC=C&A chD=D&C chB=A&B chC=B&C chA=C&A chC=D&C ch |

#### Rotations et décalages

##### Décalage d'un quartet à droite et à gauche

On spécifie le champ concerné par: 'ch'.

Ex: ASL A

(on décale le champ A du registre A d'un quartet vers la gauche)

Ex: CSR W

(on décale le champ W du registre C d'un quartet vers la droite)

|  |
| --- |
| Mnémoniques |
| ASL chBSL chCSL chDSL chASR chBSR chCSR chDSR ch |

Lors d'un décalage à gauche, le quartet à gauche du champ spécifié est perdu, le quartet à droite devient 0.

Lors d'un décalage à droite, le quartet à droite du champ spécifié est perdu, le quartet à gauche devient 0.

##### Rotation d'un quartet à droite ou à gauche

Tout les quartets sont concernés par la rotation.

|  |
| --- |
| **Mnémoniques** |
| ASLCBSLCCSLCDSLCASRCBSRCCSRCDSRC |

Lors d'un décalage à droite, le quartet à droite prend la place du quartet perdu à gauche.

Lors d'un décalage à gauche, le quartet à gauche prend la place du quartet perdu à droite.

##### Décalage d'un bit vers la droite

Tout les bits du registre sont concernés.

Ex: ASRB

(on décale tout les bits du registre A vers la droite)

|  |
| --- |
| **Mnémoniques** |
| ASRBBSRBCSRBDSRB |

Lors d'un décalage à droite, le bit le plus à droite disparaît, le bit le plus à gauche devient 0.

##### Décalage d'un bit vers la droite limité à un champ

On spécifie le champ concerné par: 'ch'.

|  |
| --- |
| **Mnémoniques** |
| ASRB.F chBSRB.F chCSRB.F chDSRB.F ch |

##### Décalage d'un bit vers la gauche

Il n'existe pas d'instruction similaire pour décaler un bit vers la gauche, mais on peut employer l'instruction: A=A+A ch pour effectuer ce décalage à gauche.

Ex: C=C+C W

(on décale tout les bits du registre C vers la gauche)

|  |
| --- |
| **Mnémoniques** |
| A=A+A chB=B+B chC=C+C chD=D+D ch |

## Les registres de sauvegarde:

Il existe 5 registres de sauvegarde qui sont respectivement: R0,R1,R2,R3 et R4. Ces registres sont identiques et permettent de sauver le contenu des registres de travail A et C.

### Copie d'un registre de travail vers un registre de sauvegarde

Tout les champs sont concernés par cette opération.

|  |
| --- |
| **Mnémoniques** |
| R0=AR1=AR2=AR3=AR4=AR0=CR1=CR2=CR3=CR4=C |

### Copie d'un registre de sauvegarde vers un registre de travail

Tout les champs sont concernés par cette opération.

|  |
| --- |
| **Mnémoniques** |
| **A=R0****A=R1****A=R2****A=R3****A=R4****C=R0****C=R1****C=R2****C=R3****C=R4** |

### Copie d'un champ d'un registre de travail vers un registre de sauvegarde

Cette opération concerne que le champ spécifié (ch) et n'a aucun effet sur les autres champs. (A noter que ' .F' signifie Field en Anglais, Champ en Français).

|  |
| --- |
| Mnémoniques |
| R0=A.F chR1=A.F chR2=A.F chR3=A.F chR4=A.F chR0=C.F chR1=C.F chR2=C.F chR3=C.F chR4=C.F ch |

### Copie d'un champ d'un registre de sauvegarde vers le champ d'un registre de travail

Cette opération concerne que le champ spécifié (ch) et n'a aucun effet sur les autres champs. (A noter que ' .F' signifie Field en Anglais, Champ en Français).

|  |
| --- |
| Mnémoniques |
| A=R0.F chA=R0.F chA=R2.F chA=R3.F chA=R4.F chC=R0.F chC=R1.F chC=R2.F chC=R3.F chC=R4.F ch |

### Echange du contenu d'un registre de travail avec le contenu d'un registre de sauvegarde

Cette opération concerne tout les champs.

|  |
| --- |
| Mnémoniques |
| AR0EXAR1EXAR2EXAR3EXAR4EXCR0EXCR1EXCR2EXCR3EXCR4EX |

A noter que l'instruction: AR0EX est la même que celle-ci: R0AEX, qui n'est pratiquement pas employée des programmeurs.

### Echange d'un champ d'un registre de travail avec le même champ d'un registre de sauvegarde

Cette opération ne concerne que le champ spécifié par 'ch'.

|  |
| --- |
| Mnémoniques |
| AR0EX.F chAR1EX.F chAR2EX.F chAR3EX.F chAR4EX.F chCR0EX.F chCR1EX.F chCR2EX.F chCR3EX.F chCR4EX.F ch |

## Les registres pointeurs

Il existe 2 registres identiques dit 'pointeur' sur HP48. Ils sont: D0 et D1. Ces registres remplissent plusieurs fonctions majeures:

Il sont capable de lire et d'écrire en mémoire, d'échanger leurs informations avec les registres de travail: A et C, de se déplacer d'une valeur comprise entre 0 et 17 (0<n<17).

### Pointage et lecture à une adresse en mémoire

Il y a deux méthodes pour pointer en mémoire. La première consiste à spécifier directement l'adresse, la deuxième méthode consiste à pointer sur le champ A (A comme adresse!) d'un registre de travail contenant cette adresse.

#### En spécifiant directement l'adresse

*On pointe en mémoire en pointant sur une adresse notée sur 2,4 ou 5 quartets.*

Ex: D0= 8068D

(on pointe sur l'adresse de l'écran)

On lit en mémoire en recopiant l'adresse pointée dans le registre A ou C.

Ex: A=DAT0 A

(Le champ A du registre A contient l'adresse pointée par D0)

|  |
| --- |
| **Mnémoniques** |
| A=DAT0 chC=DAT0 chA=DAT1 chC=DAT1 ch |

#### En pointant sur un registre contenant l'adresse

Le champ A du registre concerné devra contenir l'adresse à laquelle on veut pointer.

Ex :

D0= 8068D

A=DAT0 A

(Le champ A du registre A contient l'adresse de l'écran)

D1=A

(D1 pointe sur le début de l'écran, à l'adresse contenue dans le champ A du registre A).

Notez que 'n' correspond à une adresse de 2,4 ou 5 quartets.

|  |  |
| --- | --- |
| **Mnémoniques** | **Mnémoniques** |
| D0= nD1= n | D0=AD0=CD1=AD1=C |

On lit en mémoire en recopiant les données pointées par D1 dans un registre de travail.

Ex: A=DAT1 W

(Le champ W du registre A contient les 16 premiers quartets de l'écran).

|  |
| --- |
| **Mnémoniques** |
| A=DAT0 chC=DAT0 chA=DAT1 chC=DAT1 ch |

On peut aussi indiquer à la place du champ noté: 'ch', le nombre de quartets que l'on désire lire.

Ex: A=DAT1 4

(on lit seulement les 4 premiers quartets).

|  |
| --- |
| **Mnémoniques** |
| A=DAT0 nC=DAT0 nA=DAT1 nC=DAT1 n |

### Déplacement à gauche et à droite d'un pointeur

On peut déplacer un registre pointeur à gauche ou à droite d'une valeur de n comprise entre 0 et 17 (0<n<17). C'est très utile pour parcourir une zone en mémoire, ou pour afficher données à l'écran.

Ex: D1= 8068D

(on pointe à une adresse quelconque)

D1=D1+ 16

(on se déplace de 16 quartets vers la droite)

|  |
| --- |
| **Mnémoniques** |
| D0=D0+ nD0=D0- nD1=D1+ nD1=D1- n |

### écriture à une adresse en mémoire

On écrit en mémoire en recopiant les donnée du registre A ou C, à l'adresse pointée par D0 ou D1.

Ex:

D0= 8068D

A=DAT0 A

D0=A

(on pointe sur le début de l'écran)

LA 1234

(on charge une valeur quelconque dans le registre A)

DAT0=A A

(on recopie cette valeur à l'adresse pointée par D0)

|  |
| --- |
| **Mnémoniques** |
| DAT0=A chDAT0=C chDAT1=A chDAT1=C ch |

On peut aussi indiquer à la place du champ noté: 'ch', le nombre de quartets que l'on désire écrire.

Ex: DAT1=A 6

(on écrit seulement les 6 premiers quartets).

|  |
| --- |
| **Mnémoniques** |
| DAT0=A nDAT1=A nDAT0=C nDAT1=C n |

### Echanges entre registres pointeurs et registres de travail

Il est possible d'échanger le contenu d'un registre pointeur avec le contenu d'un registre de travail, si ce n'est que seul le champ A est concerné.

Ex: AD0EX

(On échange le contenu du champ A du registre A avec le contenu du registre D0).

|  |
| --- |
| **Mnémoniques** |
| AD0EXAD1EXCD0EXCD1EX |

## Les sauts et tests

### Les sauts

Normalement, un programme s'exécute du début à la fin, mais il est possible de bouleverser ce sens en effectuant des sauts. Il existe plusieurs type de sauts, les sauts courts (GOTO), les sauts longs (GOLONG), les appels de sous-programmes court (GOSUB) et les appels de sous-programmes longs (GOSUBL).

On effectue un saut de la manière suivante:

GOTO label

\*label

Vous pouvez mettre n'importe quel nom à la place de: 'label' si ce n'est que vous ne pouvez utiliser deux fois le même label. L'instruction GOLONG est la même que l'instruction GOTO, si ce n'est quelle a une plus grande portée:

GOTO (#800h quartets)

GOLONG (#8000 quartets)

Il existe une instruction dite absolu: GOVLNG

Après utilisation de cette instruction, le programme reprend à l'adresse spécifiée. C'est un saut dit: absolu. On utilise souvent cette instruction pour quitter un programme:

Ex: GOVLNG 05143

***Les sous-programmes***

On utilise les sous-programmes à l'aide des instructions: GOSUB et GOSUBL. Lorsqu'on utilise un sous-programme, l'adresse de départ est sauvée en vue d'un éventuel retour de sous-programme, noté le plus souvent: RTN.

Résumons-nous:

%début du prg

GOSUB label1

(l'adresse ci-dessous est sauvée, et on se rend au sous-programme nommé: label1.)

%(Après un retour de sous-programme, le programme reprend à l'adresse sauvée ci-dessous)

GOVLNG 05143 %(fin du prg)

(ici, c'est les sous-programmes).

\*label1

%(sous-programme: label)

RTN

(on revient à l'adresse initiale: c'est le retour de sous-programme).

Attention, on ne peut 'emboîter' que 6 niveaux de sous-programmes, ce qui n'est déjà pas si mal!

GOSUB (appel court: #800h)

GOSUBL (appel long: #8000h)

### Les tests

Un test n'est rien d'autre qu'une comparaison entre 2 registres de travail. Il est précédé par le signe: ?

Ex: ?A=B A

Il existe par ailleurs un bit appelé: Carry, de retenu, qui gère les tests. Un test est soit vrai, soit faux. Si le test est vrai, alors le bit Carry est armé (il vaut 1), si au contraire le test est faux, alors le bit Carry est désarmé (il vaut 0). Ce bit spécial nous sert pour gérer le résultat d'un test. En effet, de l'état de ce bit dépendent les instructions suivantes:

?A=0 A

GOYES label2

%si Carry = 1 (test vrai), on effectue un saut vers le label spécifié.

%sinon on continu le programme comme si de rien n'était.

De même On a:

?A=C A

GONC label2

%si Carry = 0 (test faux), on effectue un saut vers le label spécifié.

%sinon on continu le programme comme si de rien n'était.

Le bit Carry est automatiquement remit à 0 après une telle instruction. On peut comparer deux registres de travail, selon un champ en particulier:

Ex: ?A=B W

(on compare les champs W des registres A et B)

On peut tester par supériorité ou infériorité:

Ex: ?A<C B

Ex: ?B>C A

On peut tester si le champ d'un registre est nul ou non:

?A=0 M

On peut aussi tester l'état d'un bit:

?ABIT=1 3

?CBIT=0 6

(à gauche: l'état du bit (0 ou 1), à droite: le numéro du bit (0 à 15).

Voici le tableau des mnémoniques principales (ch signifie champ et n le numéro du bit à tester avec 0≤n≤16):

|  |  |  |
| --- | --- | --- |
| **Comparaison** | **Infériorité/Supériorité** | **Les bits** |
| ?A=0 ch?A≠0 ch?A=B ch?A≠B ch?B=0 ch?B≠0 ch?B=C ch?B≠C ch?C=0 ch?C≠0 ch?C=A ch?C≠A ch?D=0 ch?D≠0 ch?D=C ch?D≠C ch | ?A<B ch?A≤B ch?B<C ch?B≤C ch?C<A ch?C≤A ch?D<C ch?D≤C ch?A>B ch?A≥B ch?B>C ch?B≤C ch?C>A ch?C≤A ch?D>C ch?D≤C ch | ?ABIT=0 n?ABIT=1 n?CBIT=0 n?CBIT=1 n |

# PARTIE: NOTIONS DE BASE

## Les routines en mémoire morte :

Les routines sont des ‘portions’ de programmes en mémoire morte qui peuvent être comparées a des sous-programmes. Elle sont d’une extrême utilité pour le programmeur car elles exécutent des taches qui reviennent fréquemment dans un programme de jeu. Ces routines sont appelées par un simple GOSBVL xxxxx ou par GOVLNG xxxxx.

>A noter que ces routines peuvent affecter certains registres de calculs comme A,B,C ou D, il faudra en tenir compte lors de l’élaboration d’un programme.

|  |
| --- |
| Fonction :Sauvegarde des registres |
| Adresse S/SX :0679B | Appel :GOSBVL | Adresse G/GX :0679B |
| Cette routine est celle employée au début de tout les programmes en assembleur. Elle sauvegarde les registres D0,D1,B(A) et D(A) |

|  |
| --- |
| Fonction :Récupération des registres |
| Adresse S/SX :067D2 | Appel :GOSBVL | Adresse G/GX :067D2 |
| Cette routine complète la précédente. Elle restaure les registres D0,D1,B(A) et D(A) en vue d’un retour au rpl du programme. |

|  |
| --- |
| Fonction :Détection d’une pression sur une touche |
| Adresse S/SX :00E0B | Appel :GOSBVL | Adresse G/GX :00E0B |
| Apres l’exécution de la routine, C(A) contient la valeur d’entrée de la touche si une touche est pressée et 00000 si aucune touche est pressée. Le registre OUT est mis à #1FFh. |

|  |
| --- |
| Fonction :C=IN |
| Adresse S/SX :01160 | Appel :GOSBVL | Adresse G/GX :01160 |
| Place les 3 quartets du registre IN dans les 5 quartets du registre C. |

|  |
| --- |
| Fonction :Ecran ON |
| Adresse S/SX :01B8F | Appel :GOSBVL | Adresse G/GX :01B8F |
| Cette routine allume l'écran.Apres exécution, le registre D0 pointe à l'adresse #100h. |

|  |
| --- |
| Fonction :Ecran OFF |
| Adresse S/SX :01BBD | Appel :GOSBVL | Adresse G/GX :01BBD |
| Cette routine éteint l'écran.Après exécution, le registre D0 pointe à l'adresse #100h. |

|  |
| --- |
| Fonction :Conversion du Décimal en Hexadécimal du registre C |
| Adresse S/SX :0D62F | Appel :GOSBVL | Adresse G/GX :0D62F |
| Convertit le nombre décimal placé dans C(W) en hexadécimal. Après exécution de la routine, A(W), B(W), C(W) contiendrontle nombre convertit. |

|  |
| --- |
| Fonction :Conversion de l'hexadécimal en Décimal du registre A |
| Adresse S/SX :0DB91 | Appel :GOSBVL | Adresse G/GX :0DB91 |
| Convertit le nombre hexadécimal placé dans A(W) en Décimal.Après exécution de la routine, A(W), B(W), C(W) contiendrontle nombre convertit. |

|  |
| --- |
| Fonction :Interruptions OFF |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :01115 |
| Les interruptions sont interdites (ST=0 15). |

|  |
| --- |
| Fonction :Interruptions ON |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :010E5 |
| Autorise les interruptions (ST=1 15). |

|  |
| --- |
| Fonction :Reset (ON C) |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :01FBD |
| Effectue un redémarrage à chaud comme celui réalise par le doubleappui sur les touches ON et C. |

|  |
| --- |
| Fonction :Try to Recover Memory (ON A F) |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :015F0 |
| Même effet que le triple appui sur les touches ON A et F. |

|  |
| --- |
| Fonction :Beep (D et C) |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :017A6 |
| Même effet que BEEP.Le registre C(A) doit contenir la durée en millisecondes et D(A) la fréquence en Hz.  |

|  |
| --- |
| Fonction :Mem |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :06806 |
| Même effet que MEM.Apres exécution, C(A) contient la taille de la mémoire disponible.D0 et A(A) sont modifiés. |

|  |
| --- |
| Fonction :$0000...00 |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :05B7D |
| Crée une chaîne de caractères constituée de C(A) quartets.Apres exécution de la routine, D0 pointe sur le premier caractère.R0, C(A) et D1 sont modifiés. |

|  |
| --- |
| Fonction :Exit |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :05143 |
| Manière la plus sure de quitter un programme en assembleur. |

|  |
| --- |
| Fonction :D0= écran |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :01C31 |
| Apres exécution de la routine, D0 pointe sur l'adresse du débutde l'écran. A(A) correspond à D0. |

|  |
| --- |
| Fonction :ClrLCD |
| Adresse S/SX : | Appel :GOSBVL | Adresse G/GX :01BFF |
| Efface l'écran et les menus.Les registres A(W), C(A), et D0 sont modifiés. |

## Adresses utiles de la mémoire vive :

La mémoire vive (ou RAM I/O) des entrées et sorties est très utiles au programmeur désirant modifier certains paramètres de la HP48. En effet le programmeur aura la possibilité a l'aide de ces adresses de modifier le contraste de l'écran , d'allumer ou d'éteindre certain des indicateurs en haut de l'écran par exemple. La plage d'adresses utilisée par la RAM I/O s'étant a partir de l'adresse #00100h jusqu'à l'adresse #0013Fh.

La technique consiste a lire les données pointée a une adresse de la RAM I/O à l'aide d'un registre pointeur tel D0 ou D1:

ex: D0= xxxxx

De lire les données pointées a l'aide d'un registre de travail tel A ou C:

ex: A=DAT0 A (ou C=DAT0 A )

De modifier les données lues en armant ou désarmant les bits du registre A ou C:

ex: ABIT=0 2 (ou CBIT=1 3 )

Puis de réécrire la donnée ainsi modifiée a l'adresse spécifiée dans D0 ou D1:

ex: DAT0=A A (ou DAT0=C A )

Avant cela il faut avoir une parfaite connaissance de chaque adresse de la RAM I/O sous peine de pertes de données fréquentes !

Voici présentées ici les principales adresse de la RAM I/O:

|  |  |  |
| --- | --- | --- |
| #00101 | Bit n°0 à 4 | Contraste de l'écran |
| Le contraste peut prendre 32 valeurs (soit #20h.) codées sur 5 bits. Modifier la valeurs de ces 5 bits revient a modifier le contraste de l'écran. Utile lorsque l'on désire modifier le contraste durant un jeu. |

|  |  |  |
| --- | --- | --- |
| #0010B | Bit n°0 à 3 | Allumage des indicateurs |
| De l'état de ces 4 bits dépend l'allumage des indicateurs en haut de l'écran. Le bit 0 allume l'indicateur < quand il est armé, le bit 1 >, le bit 2 α et le bit 3 l'indicateur de sonnerie. |

|  |  |  |
| --- | --- | --- |
| #0010C | Bit n°0,1&3 | Allumage des indicateurs |
| Le bit 0 allume l'indicateur 'occupé' lorsqu'il est armé, le bit 1 l'indicateur de transmission. Le bit 3 autorise l'allumage des indicateurs lorsqu'il est armé, ou les interdits lorsqu'il est désarmé. Utile lorsque l'on désire éteindre l'indicateur occupé. |

|  |  |  |
| --- | --- | --- |
| #00120 | Bit n°0 à 3 | Adresse de l'écran |
| L'adresse du GROB en permanence affiché à l'écran est fixée à cette adresse. Vous ne pouvez qu'écrire une nouvelle adresse en #00120h pour modifier le grob de l'écran. |

|  |  |  |
| --- | --- | --- |
| #00128 | Bit n°0 à 3 | Balayage et hauteur menu |
| Ces 4 bits contrôlent en lecture, le numéro de la ligne en cours de rafraîchissement, en écriture, la hauteur des menus. (en écrivant #3Fh en #00128h on fait disparaître la barre des menus). Pour obtenir le numéro de la ligne en cours de rafraîchissement, on lit les 6°bits (ex: D0= 00128 A=DAT0 6). Le registre A vaut donc 0 lorsque la dernière ligne a été rafraîchie. L'écran est rafraîchi 64 fois par seconde. Utile pour l'élaboration d'un jeu en plusieurs niveaux de gris. |

|  |  |  |
| --- | --- | --- |
| #00130 | Bit n°0 à 3 | Adresse de la bitmap menu |
| L'écran est séparé en deux. L'adresse de la partie inférieure est fixée en #00130 et correspond à la bitmap de la barre des menus. |

## Qu'est-ce qu'un programme SOURCE ?

L'aspect technique ayant été abordé, passons à la programmation à proprement parler. Réaliser un programme dans n'importe quel langage que ce soit nécessite l'élaboration au préalable d'un fichier dit: 'source'. Tout programme est d’abord un fichier source. Il est donc représenté par la HP par une chaîne : " " , et doit impérativement se terminer par le caractère spécial : @. Ce qui donne, pour un programme réduit à sa plus simple expression :

"

@ "

## Comment le rendre exécutable ?

Nous avons vu précédemment qu'un programme source simple se résumait à:

"

@"

Si vous assemblez ce programme source, vous obtiendrez alors votre premier Code exécutable ! Mais qui ne réalisera rien bien entendu, car il n’y a encore rien dans le programme ! On obtient alors dans la pile de la HP48:

  

  

>Je vous conseil d’ailleurs de ne pas tenter d’évaluer ce Code sous risque de perte de données. Voyons maintenant ce que l’on peut mettre dans ce programme !

## Structure de base d'un programme :

Afin de réaliser un programme ordonné, il est indispensable d'avoir une certaine structure de base. En effet, la plupart des programmes de jeu reprenant la même structure, il est indispensable de bien la connaître. Cette structure est très simple: une boucle infini qui attend un événement pour en sortir. (Cet événement peut-être la pression sur une touche par exemple). En clair cela donne:



Ayez toujours ce schéma en tête, il vous sera très utile par la suite. L'événement qui marque la fin d'un jeu est souvent la détection de la pression sur une touche, c'est pourquoi il s'avère maintenant indispensable d'aborder les tests du clavier.

## Comment tester une touche ?

Pour tester une touche particulière du clavier , le programmeur aura recours aux couples OUT/IN:

OUT est le registre des sorties du clavier (3 quartets).

IN est le registre des entrées du clavier (4 quartets).

Avant tout, il faut mettre à 0 le registre P:

P= 0

La technique consiste d'abord à charger dans le registre de calcul C(X) la valeur de sortie de la touche en question à l'aide de l'instruction:

LC xxx

Puis de placer cette valeur dans le registre OUT avec l'instruction:

 OUT=C

On interroge alors le clavier. Pour connaître la réponse, il faut récupérer la valeur d'entrée de la touche à l'aide de l'instruction:

A=IN

(on aura plutôt recours a la routine GOSBVL 01160)

Il ne reste plus alors qu'a effectuer un "et logique" entre C et A:

C=C&A X

Si le registre C(X) est nul, la touche n'est pas pressé, si C(X) est non nul, la touche est pressée. Effectuons un test logique:

?C=0 X

GOYES xxxx

(action spécifique a l'action de la touche)

\*xxxx

En résumé, cela nous donne:

P= 0

LA xxx

LC xxx

OUT=C

GOSBVL 01160

C=C&A X

?C=0 X

GOYES xxxx

\*xxxx

### Les couples OUT/IN:

Il s'avère donc indispensable de connaître le code d'entrée et de sortie de chaque touches :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A**out: 002in: 010 | **B**out: 100in: 010 | **C**out: 100in: 008 | **D**out: 100in: 004 | **E**out: 100in: 002 | **F**out: 100in: 001 |
| **MTH**out: 004in: 010 | **PRG**out: 080in: 010 | **CST**out: 080in: 008 | **VAR**out: 080in: 004 | out: 080in: 002 | **NXT**out: 080in: 001 |
| **'**out: 001in: 010 | **STO**out: 040in: 010 | **EVAL**out: 040in: 008 | **←**out: 040in: 004 | **↓**out: 040in: 002 | **→**out: 040in: 001 |
| **SIN**out: 008in: 010 | **COS**out: 020in: 010 | **TAN**out: 020in: 008 | **√x**out: 020in: 004 | **yx**out: 020in: 002 | **1/x**out: 020in: 001 |
| **ENTER**out: 010in: 010 | **+/-**out: 010in: 008 | **EEX**out: 010in: 004 | **DEL**out: 010in: 002 | **⇐**out: 010in: 001 |
| **α**out: 008in: 020 | **7**out: 008in: 008 | **8**out: 008in: 004 | **9**out: 008in: 002 | **÷**out: 008in: 001 |
| **<**out: 004in: 020 | **4**out: 004in: 008 | **5**out: 004in: 004 | **6**out: 004in: 002 | **×**out: 004in: 001 |
| **>**out: 002in: 020 | **1**out: 002in: 008 | **2**out: 002in: 004 | **3**out: 002in: 002 | **-**out: 002in: 001 |
| **ON**out: 400in: 8000 | **0**out: 001in: 008 | **.**out: 001in: 004 | **SPC**out: 001in: 002 | **+**out: 001in: 001 |

## Squelette d'un jeu

### Avertissement

Maintenant que les tests du clavier ont été abordés, nous pouvons mettre en place le squelette d'un jeu. Nous allons donc réaliser un programme qui prendra fin lors de l'appui sur une touche du clavier. Pour cela, nous reprendrons la structure de base d'un jeu étudiée précédemment, mais tout d'abord un avertissement: ***TOUT programme quel qu'il soit doit TOUJOURS débuter par la routine:***

GOSBVL 0679B

Qui a pour but de sauvegarder les registres courants de la HP48. Puis INTOFF qui interdit le traitement du clavier. ***De même TOUT programme doit se terminer par la routine:***

GOSBVL 067D2

Qui restaure les registres courants de la HP48. Mais nous emploierons plutôt la routine plus performante:

GOSBVL 05143

Ainsi, votre premier programme autonome se présentera sous la forme:

"GOSBVL 0679B

INTOFF

% ici, c'est votre programme !!

GOSBVL 05143

@"

Il suffirait d'assembler ce mini source pour se rendre compte que ce programme fonctionne à merveille ! En effet, il se lance puis s'arrête automatiquement: C'est magique me diriez-vous, non, c'est simplement de l'ASSEMBLEUR !

### Mise en forme

Mais revenons plutôt a notre petit jeu. La touche à tester sera ENTER. Analysons le couple OUT/IN spécifique de la touche ENTER:

OUT: 010

IN: 010

Le test spécifique à cette touche sera donc:

P= 0

LA 010

LC 010

OUT=C

GOSBVL 01160

C=C&A X

?C≠0 X

GOYES xxxx

\*xxxx

Plaçons-le dans une boucle infinie:

\*label\_1

P= 0

LA 010

LC 010

OUT=C

GOSBVL 01160

C=C&A X

?C≠0 X

GOYES label\_2

GOTO label\_1

\*label\_2

Ajoutons à cela un début et une fin et voici notre programme:

"GOSBVL 0679B

INTOFF

\*label\_1

P= 0

LA 010

LC 010

OUT=C

GOSBVL 01160

C=C&A X

?C≠0 X

GOYES label\_2

GOTO label\_1

\*label\_2

GOVLNG 05143

@"

Il est important de signaler ici que, en assembleur, le programmeur a de multiples possibilités pour réaliser la même tache dans un programme. C'est pourquoi certain programmeurs de jeu sur HP48 'paraissent' très doués, il ne sont en fait que très ordonnés et efficace dans l'analyse de chaque fonction de leur jeu. Ainsi, un programmeur gagne à être clair et structuré dans l'élaboration de ses jeux.

Mais reprenons plutôt notre petit programme, Ici rien ne nous empêche de placer le test de la touche ENTER en sous-programme. On gagne alors en clarté:

"GOSBVL 0679B

INTOFF

\*label\_1

%(ici c'est le coeur du programme!)

GOSUB test\_1

GOTO label\_1

%(ici, c'est le sous programme)

\*test\_1

P= 0

LA 010

LC 010

OUT=C

GOSBVL 01160

C=C&A X

?C≠0 X

GOYES label\_2

RTN

\*label\_2

GOVLNG 05143

@"

Les sous-programmes devrons toujours se situer à la fin du programme. Pour ceux qui on suivi, rien n'empêche de rajouter des appels de sous-programmes dans le 'coeur' du jeu, puis d'écrire ces sous-programmes les uns à la suite des autres ! On gagne alors en clarté et efficacité.

Entraînez-vous à tester d'autres touches en modifiant le couple OUT/IN suivant le tableau précédent. A noter que si votre machine ne répond plus, vous pourrez à tout moment interrompre la procédure par le double appui sur les touches: ON\_C

Ce qui aura pour effet immédiat de réinitialiser la HP48.

## Comment émettre un son ?

Un jeu vidéo n'est rien d'autre qu'un renouvellement de graphiques, de quelques tests du clavier, le tout agrémenté de quelques sons. Nous allons donc étudier la manière de réaliser des sons. Mais méfiez-vous, ne vous attendez surtout pas à entendre votre HP48 vous jouez du Mozart ou recréer l'ambiance d'un jeu Mystique ! En fait, le buzzer qui équipe les modèles de HP48 est très limité. Il ne sait jouer que quelques fréquences aiguës, ne permettant pas de réaliser une bande son. Nous nous contenterons donc de réaliser des effets sonores simples, comme le son du rebond d'une balle contre un mur par exemple !

Pour réaliser un son, plusieurs méthodes s'ouvrent à nous.

Nous pouvons soit:

- avoir recourt à la routine BEEP qui prend comme argument la fréquence et la durée du son.

- Créer le sous-programme de nous même.

Dans le but de vous familiariser avec le langage assembleur, la seconde méthode s'avère préférable !

### Procédons par étape successives:

\* Un son est caractérisé par sa fréquence en Hertz et par sa durée en secondes.

\* Il est obtenu en activant, puis en désactivant le buzzer HP48, plus ou moins rapidement (la fréquence), plus ou moins longtemps (la durée).

Voyons comment faire vibrer le buzzer HP48:

Le buzzer est activé en plaçant #800h dans le registre OUT, et il est désactivé en plaçant #000h dans ce même registre. Une activation, suivie d'une désactivation provoque un CLIC:

LC 800

OUT=C

C=0 X

OUT=C

Cette bride de programme provoque un CLIC. Une suite de clic à fréquence régulière provoque un son. Voyons comment réaliser une fréquence: Il nous faut placer ce programme dans une première boucle, ou le registre A(A) sera le compteur. Ce sera la durée effective du son. Il nous faut placer ensuite une seconde boucle, ou B(A) sera le compteur. Ce sera la fréquence effective du son. Cela nous donne:

A=0 A

LA 1000

C=0 A

LC 20

B=C A

D=C A

\*label\_1

LC 800

OUT=C

C=0 X

OUT=C

\*label\_2

B=B-1 A

GONC label\_2

B=0 A

C=D A

B=C A

A=A-1 A

GONC label\_1

A=0 A

Modifier la valeur initiale du registre B(A) ou A(A) revient donc à modifier la fréquence ou la durée du son ainsi émit. Il nous suffit plus alors qu'a mettre un début et une fin à notre programme pour le rendre autonome.

Ce qui nous donne:

"GOSBVL 0679B

INTOFF

A=0 A

LA 1000

C=0 A

LC 20

B=C A

D=C A

\*label\_1

LC 800

OUT=C

C=0 X

OUT=C

\*label\_2

B=B-1 A

GONC label\_2

B=0 A

C=D A

B=C A

A=A-1 A

GONC label\_1

A=0 A

GOVLNG 05143

@"

Assemblez ce programme et écoutez votre HP produire son premier cri en assembleur !!

Encore une fois, rien ne nous empêche ici de placer la partie du programme correspondant au son, dans un sous-programme, et qui serait appelé par la pression sur une touche:

"GOSBVL 0679B

INTOFF

\*prg

GOSUBL son

P= 0

LA 010

LC 010

OUT=C

GOSBVL 01160

C=C&A X

?C=0 X

GOYES nofin

GOVLNG 05143

\*nofin

GOTO prg

\*son

A=0 A

LA 1000

C=0 A

LC 20

B=C A

D=C A

\*label\_1

LC 800

OUT=C

C=0 X

OUT=C

\*label\_2

B=B-1 A

GONC label\_2

B=0 A

C=D A

B=C A

A=A-1 A

GONC label\_1

A=0 A

RTN

@"

La touche ENTER quitte le programme. A noter qu'un maximum de 6 embriquement de sous-programme n'est possible, sinon gare au Memory Lost ! Nous constatons ici encore à quel point réaliser un programme avec une structure ordonnée est utile. En effet, le corps du programme est très bref et compréhensible. Imaginez que par la suite vous ayez une trentaine de sous-programmes les uns à la suite des autres, le programme deviendrais vite illisible, d'ou une certaine rigueur dès le début de l'écriture du programme.

## Comment afficher un GROB ?

Nous arrivons maintenant à la partie la plus importante mais aussi la plus délicate lors de la création d'un jeu vidéo, à savoir: comment afficher un GROB de taille quelconque à l'écran ? Tout d'abord, un petit discours: L'écran est un périphérique de la HP au même titre que le clavier. Il est en fait la représentation d'une portion de la mémoire, donc écrire sur l'écran revient à écrire en mémoire. Pour cela , le programmeur aura recourt à l'adresse du début du grob de l'écran, situé à: #8068Dh pour les GX et #7050Eh pour les SX.

La technique consiste donc à pointer à cette adresse à l'aide d'un des registres pointeurs tels D0 ou D1, ex: D0= 8068D de lire l'adresse ainsi pointée, ex: A=DAT0 A puis de pointer sur les données elles-mêmes à l'aide d'un des registres de travail tels A(A) ou C(A), ex: A=DAT0 A On peut ainsi modifier à souhait ces données: ex: A=0 A (on efface ) ou ABIT=1 3 (on arme le bit 3) et on réécrit la donné ainsi modifiée, ex: DAT0=A A

D0= 8068D

A=DAT0 A

D0=A

A=DAT0 W

ABIT=1 4 %(ou: A=0 W pour effacer)

DAT0=A A

Si l'on veut, on peut même charger une valeur et l'afficher à l'écran, ex: LC 55 DAT0=C B (#55h correspond à #0101010101b) Il s'affichera alors à l'écran la valeur #55h, ou plutôt les bits 1010101010 de l'écran.

Il nous suffira ensuite de parcourir la totalité (ou une partie) de l'écran pour afficher un GROB sur tout l'écran (ou une partie). On parcours les données de l'écran en faisant déplaçant le registre pointer en avant:

ex: D0=D0+ n (0<n<17)

ou D1=D1+ n (0<n<17)

ou en arrière si l'on est situé en bas de l'écran,

ex: D0=D0- n (0<n<17)

ou D1=D1- n (0<n<17)

Lorsque vous déplacez en avant le registre pointer, vous vous déplacez alors de gauche à droite. Lorsque vous dépassez l'écran (à droite), le registre pointe alors à gauche, une ligne plus bas.

Vous avez parcouru une ligne de l'écran (131 pixels), après avoir incrémenté votre registre pointer de 34 en avant, ou en arrière.

Attention: #34d => #22h

Ce qui donne avec le registre pointer D0:

D0=D0+ 16

D0=D0+ 16

D0=D0+ 2

Ou:

D0=D0- 16

D0=D0- 16

D0=D0- 2

Résumons-nous:

### Pour modifier une donnée à l'écran:

D0= 8068D

A=DAT0 A

D0=A

%(on pointe sur le début de l'écran)

A=DAT0 W

%(on récupère dans A(W) les données de l'écran: 16 quartets = 64bits)

A=0 W %(on efface) ou ABIT=1 8 (on arme le pixel 8)

DAT0=A W

%(on réécrit les données ainsi modifiées)

### Pour écrire une donnée à l'écran:

#### Par écrasement:

D0= 8068D

A=DAT0 A

D0=A

%(on pointe sur le début de l'écran)

C=0 A

LC 55555

%(chargement d'une valeur dans C(A))

DAT0=C A

%(on écrit la donnée contenue dans C(A) par dessus la donnée actuelle)

#### Par transparence:

D0= 8068D

A=DAT0 A

D0=A

%(on pointe sur le début de l'écran)

A=DAT0 A

%(on lit les données actuelles)

C=0 A

LC 55555

%(chargement d'une valeur dans C(A))

C=C&A A

%( "et" logique entre la donnée actuelle et la donnée chargée)

DAT0=C A

%(on écrit la donnée contenue dans C(A) par transparence)

### Pour afficher un GROB sur tout l'écran

Avant tout, rappelons ce qu'est réellement un GROB. Un GROB est un objet graphique obtenu à l'aide de l'environnement PICTURE de votre HP48. C'est dans cette objet que sont stockées toutes les données concernant le graphique. C'est donc en lisant ces données que l'on peut ensuite afficher un GROB à l'écran.

*Habituez-vous dès maintenant à manipuler les GROB:*

Dans l'environnement PICTURE, dessinez puis sauvegardez votre dessin à l'aide de la touche [STO].

Votre dessin est alors placé dans le 1° niveau de la pile, cela donne:

Grob 131 \* 64

Pour pouvoir l'insérer dans votre programme source, il faut ôter aux GROB leur préface et leur taille. Pour cela, vous devez rééditer le GROB de la manière suivante:

GROB 131 64 123454100 [...] 000

Devient ensuite:

$1234541000 [...] 000

Par suppression du prologue GROB 131 64 , inutile en assembleur.

*A remarquer:*

L'insertion à gauche du caractère $ qui caractérise, dans un programme source, une donnée quelconque.

Ainsi, votre GROB doit ressembler à celui-ci:

$12345100015343154135411531435411654153411531..

Avant de se lancer dans l'affichage d'un GROB sur l'écran entier, encore faut-il que vous sachiez comment récupérer l'adresse de ce GROB !

Pour ce faire, il existe plusieurs méthodes: la plus connues et employée de tous les programmeurs en assembleur est celle-ci:

#### Méthode 1

GOSUBL label\_1 %(on place le début du GROB dans RSTK)

$000000032251405231430 %(ici c'est le grob sans sa taille )

\*label\_1

C=RSTK %(ici on récupère dans C(A) l'adresse du GROB)

A la sortie, C(A) contient l'adresse du début du GROB.

On peut ensuite soit sauver cette adresse, (dans R0 par exemple), soit pointer directement sur celle-ci à l'aide d'un des registres pointers D0 ou D1.

(Ex: D0=C ou D1=C).

#### Méthode 2

La seconde méthode consiste à lire directement un GROB dans la pile, et de l'afficher à l'écran. Pour cela, Il faut dès le début du programme lire l'adresse que contient le registre D1, car habituellement, c'est dans ce registre que se trouve l'adresse de l'objet contenu dans le premier niveau de la pile. On lit cette adresse à l'aide de l'instruction:

A=DAT1 A %(ou C=DAT1 A)

Si cette adresse vaut 0, donc si A(A) est nul, il n'y a aucun objets dans la pile.

On test alors avec:

?A≠0 A

GOYES label\_1

GOVLNG 05143

\*label\_1

A ce niveau, on pourrait tester si l'objet dans la pile est bien un objet graphique :

C=0 W

LC 02B1E

?A=C A

GOYES label\_2

GOVLNG 05143

\*label\_2

Il nous reste plus qu'a pointer sur ce GROB à l'aide de l'instruction:

D1=A

A ce stade, il reste plus qu'a effectuer la copie des données du GROB sur les données de l'écran. Voici le programme complet qui affiche un GROB de 131 \* 56 à l'écran:

"GOSBVL 0679B

INTOFF

GOSUBL label\_1

$123454115341865346854186 [...] 012

\*label\_1

C=RSTK

D1=C

%(D1 pointe sur le GROB à afficher)

D0= 8068D

A=DAT0 A

D0=A

%(D0 pointe sur le début de l'écran)

C=0 W

LC 76F

%(il y a 56 lignes à afficher: 56 \* 136= 7616 pixels, soit 1904 quartets, ou encore #770h. #770 - 1 = #76Fh.)

\*label\_2

A=DAT1 A %(on recopie la donnée du GROB pointée par le registre pointeur D1, dans A(A) )

DAT0=A 1 %(on copie un quartets par un, le GROB à l'écran, à l'adresse contenue dans le registre pointeur D0)

D0=D0+ 1 %(on se déplace d'un quartet à droite sur l'écran)

D1=D1+ 1 %(on se déplace d'un quartet à droite sur le GROB)

C=C-1 A

GONC label\_2

%(La boucle prend fin quand #770h quartet sont copiés à l'écran. Soit un GROB de 131 \* 56 pixels)

C=0 W

GOVLNG 05143

@"

Rien ne nous empêche de recopier les données du GROB à l'écran, par plusieurs quartets à la fois.

Il suffit alors d'utiliser des champs plus grands (ex: champs B,X,W)

de la manière suivante:

A=DAT1 B

DAT0=A B

Ici, le champs B regroupe 2 quartets, l'affichage va s'effectuer 2 fois plus vite, à la condition d'ajuster le compteur de boucle:

( #770h ÷ 2 ) -1 = #3B7h

Sinon, la boucle est trop longue et on écrit dans la mémoire vive (après la dernière ligne de l'écran), ce qui est formellement interdit sous peine de Mémory-lost !

De la même manière, si l'on utilise les champs W (16 quartets), il faut donc une boucle de valeur: ( #770h ÷ #10h ) - 1 = #76h

Ainsi, c'est en ajustant au mieux les champs de nos registres, lors de l'affichage d'un GROB à l'écran, que l'on peut gagner beaucoup de vitesse d'exécution du jeu.

Mais cette méthode n'est valable que pour l'affichage de GROB de 131 pixels de largeur, sinon il faut opérer de manière différente.

## Gérer les sprites

### Comment afficher un sprite ?

Tout d'abord, il faut signaler qu'en assembleur vos sprites devront avoir une largeur qui se compte en quartets, par groupe de 4 pixels. C'est à dire: 4, 8, 12, 16, 20, 24 ... pour une raison simple: cela correspond à la taille des champs des registres de travaille! Ainsi, pour un sprite de 8 \*12 pixels, on emploiera le champs B pour afficher une ligne de ce sprite car le champs B peut contenir 8 pixels. Vous devrez en tenir compte lorsque vous réaliserez vos sprites!

Les sprites sont dessinés dans l'environnent PICTURE de votre HP48.

Pour récupérer une portion d'écran (une balle par exemple) de l'environnement PICTURE, placez le curseur en haut à gauche de ce sprite et appuyez sur la touche [\*] pour placer une marque, déplacer vous ensuite en bas à droite de ce sprite et actionnez la commande [sub] pour extraire votre sprite. Ce sprite est alors placé dans la pile. Prenons l'exemple d'un sprite de 8 \* 12 pixels, Dessinez un rectangle noir de taille 8 \* 12 pixels, puis extrayez le, cela donne dans la pile:

Grob 8 \* 12

Si vous l'éditez:

GROB 8 12 FFFFFFFFFFFF

Puis enlevez le prologue et ajoutez le caractère $:

$FFFFFFFFFFFF

C'est cette portion de graphique ou sprite que nous devons afficher à l'écran. La largeur étant égale à 8 pixels, nous emploierons le champs B pou effectuer l'affichage.

Pour ce faire, plusieurs méthodes.

#### Par écrasement

Nous allons réemployer ici la méthode dites "par écrasement", déjà abordée précédemment. Cela signifie que le sprite sera affiché par dessus les données actuelles de l'écran.

Pour afficher un sprite, il faut tout d'abord tenir compte de sa taille. Ici la largeur de notre sprite est égale à 8 pixels, nous emploierons donc le champs B qui est le mieux adapté à la situation. La technique consiste alors à afficher le sprite ligne à ligne, et ce à l'aide d'une boucle qui correspond à la hauteur (-1) en pixels du sprite. Ici, notre sprite à une hauteur effective de 12 pixels, soit #Ch - 1 = #Bh. Notre boucle d'affichage aura comme valeur initiale: #Bh.

Pour passer d'une ligne à la suivante:

A l'écran cela correspond a un décalage de 34 quartes (#22h), par contre, sur le sprite, tout dépend de sa largeur. Pour une largeur de 4 pixels, on se déplace de deux quartets:

0>4 pixels: 2 quartets

4>8 pixels: 2 quartets

8>12 pixels: 4 quartets

12>16 pixels: 4 quartets

16>20 pixels: 6 quartets

20>24 pixels: 6 quartets

24>28 pixels: 8 quartets

etc...

Ici la largeur du sprite à afficher étant égale à 8 pixels, on se déplacera de ligne en ligne: 2 quartets par 2. Nous afficherons notre sprite à partir du coin supérieur gauche de l'écran en récupérant l'adresse en: GOSBVL 01C31. Voici ce que cela donne:

"GOSBVL 0679B

INTOFF

GOSUBL sprite\_1

$FFFFFFFFFFFF

\*sprite\_1

C=RSTK %(on récupère l'adresse du sprite)

D1=C %(D1 pointe sur le sprite)

GOSBVL 01C31 %(avec cette routine, D0 pointe sur l'écran)

C=0 W

LC B %(il y a 12 ligne, 12-1=11 ou #Bh)

\*label\_1

A=DAT1 B %(on lit le sprite)

DAT0=A B %(on le recopie sur l'écran)

D0=D0+ 16

D0=D0+ 16

D0=D0+ 2 %(on passe à la ligne suivante à l'écran)

D1=D1+ 2 %(on passe à la ligne suivante du sprite)

C=C-1 B

GONC label\_1 %(c'est la boucle, 12 ligne à copier)

C=0 W

GOVLNG 05143 %(on quitte le programme!)

@"

#### Par transparence

Cette technique d'affichage ayant déjà été abordée précédemment, nous nous y attarderons pas. Sachez que la technique dite par transparence consiste a copier le sprite à l'écran par dessus les données de l'écran par transparence entre l'écran et le sprite. Pour cela, on utilisera l'instruction '!' appelée: "et" logique. Le champs B jouera le rôle du compteur de boucle. Voici notre programme:

"GOSBVL 0679B

INTOFF

GOSUBL sprite\_1

$FFFFFFFFFFFF

\*sprite\_1

C=RSTK %(on récupère l'adresse du sprite)

D1=C %(D1 pointe sur le sprite)

GOSBVL 01C31 %(avec cette routine, D0 pointe sur l'écran)

C=0 W

B=0 W

LC B %(il y a 12 ligne, 12-1=11 ou #Bh)

B=C B

\*label\_1

A=DAT1 B %(on lit le sprite)

C=DAT0 B %(on lit l'écran)

A=A!C B %(on 'mélange' le tout !)

DAT0=A B %(on recopie sur l'écran)

D0=D0+ 16

D0=D0+ 16

D0=D0+ 2 %(on passe à la ligne suivante à l'écran)

D1=D1+ 2 %(on passe à la ligne suivante du sprite)

B=B-1 B

GONC label\_1 %(c'est la boucle, 12 ligne à copier)

B=0 W

GOVLNG 05143 %(on quitte le programme!)

@"

On ne peut élaborer un jeu vidéo sans savoir gérer parfaitement les sprites. Ainsi il est nécessaire que vous maîtrisiez parfaitement cette partie du chapitre. Entraînez vous à manipuler les GROB, sprites et autres.

### Comment déplacer un sprite ?

#### Première approche:

Déplacer un sprite revient à déplacer sur l'écran sa position d'affichage. Cette position dépend de l'adresse sur laquelle on pointe à l'écran. Modifier cette adresse revient à modifier la position d'affichage du sprite, et donc de le déplacer. Pour cela, il est nécessaire au préalable d'effectuer une sauvegarde de l'adresse ou le sprite doit être affiché. On pourrait à titre d'exemple sauver cette adresse dans le registre de sauvegarde R0. Cela nous donnerai alors:

GOSBVL 01C31

AD0EX

R0=A

Il nous suffirai alors pour déplacer le sprite d'ajouter 1 a l'adresse sauvegardée en R0:

A=R0

A=A+1 A

R0=A

Notre sprite se déplacerai alors d'un quartet (4 pixels) à droite. Inversement, pour déplacer le sprite d'un quartet à gauche:

A=R0

A=A-1 A

R0=A

Pour déplacer le sprite d'une ligne de l'écran vers le bas, il faut ajouter 34 (#22h) à cette adresse:

A=R0

A=A+16 A

A=A+16 A

A=A+2 A

R0=A

Inversement, on déplace notre sprite d'une ligne de l'écran vers le haut en soustrayant 34 à cette adresse:

A=R0

A=A-16 A

A=A-16 A

A=A-2 A

R0=A

## Un petit jeu !

Nous allons réaliser dans ce chapitre, un petit jeu qui sera en fait: votre premier jeu en assembleur ! Que tout ceux qui ont brûlé des étapes se méfient: L'élaboration de ce petit jeu est la synthèse de tout ce que nous venons d'étudier. Il serait donc préférable d'avoir lu les précédents chapitres avant d'aborder la création de ce jeu! Ne vous méprenez pas, ce jeu sera simple et par conséquent, assez banal, mais permettra néanmoins de vous familiariser au langage assembleur.

*Voici donc les caractéristiques du jeu:*

Il s'agit en fait d'un sprite quelconque que l'on déplace à l'écran à l'aide des touches fléchées. Ce sprite ne sort jamais de l'écran. On quitte le jeu en actionnant la touche [ENTER].

ATTENTION :

votre sprite ne devra jamais dépasser de l'écran, ni en haut, ni en bas, sous peine de Memory Clear !!

En effet, dépasser l'écran revient à écrire sur la mémoire de la HP. Afin d'éviter ceci, le programmeur aura recours à plusieurs procédés. La méthode la plus courante consiste à utiliser deux compteur implantés en mémoire, l'un pour les abscisses du sprite, l'autre pour les ordonnées. Les compteurs seront implantés dans la mémoire 'tampon' qui débute à l'adresse: #800.. mais dans le cadre de cet exemple, nous emploierons les registres B(A) et D(A).

*Voici le programme source:*

"GOSBVL 0679B

INTOFF

C=0 B

LC 3F

D0= 00128

DAT0=C B

%(on enlève les menus de l'écran)

GOSBVL 01C31

%(on récupère l'adresse écran)

C=0 W

LC 43F

A=0 B

\*clear

DAT0=A B

D0=D0+ 2

C=C-1 X

?C=0 X

GONC clear

C=0 A

%(on efface tout l'écran)

D0= 8068D

A=DAT0 A

R0=A

A=0 W

%(position du sprite dans R0)

\*prog

%(début programme principal)

GOSUBL testquit

GOSUBL test\_d

GOSUBL test\_g

GOSUBL test\_h

GOSUBL test\_b

GOSUBL sprite

GOSUBL wait

GOSUBL clearsprite

%(fin programme principal)

GOLONG prog

%(sous-programmes)

\*testquit

P= 0

LA 010

LC 010

OUT=C

GOSBVL 01160

C=C&A X

?C=0 X

GOYES noquit

GOVLNG 05143

%(ici on quitte)

\*noquit

RTN

\*wait

%(boucle d'attente: on bride le prog. Sinon c'est trop rapide !!)

C=0 W

LC 2000

\*label\_1

C=C-1 A

GONC label\_1

C=0 W

RTN

\*test\_g

P= 0

LA 004

LC 040

OUT=C

GOSBVL 01160

C=C&A X

?C=0 X

GOYES no\_g

C=R0

C=C-1 A

R0=C

\*no\_g

RTN

\*test\_d

P= 0

LA 001

LC 040

OUT=C

GOSBVL 01160

C=C&A X

?C=0 X

GOYES no\_d

C=R0

C=C+1 A

R0=C

\*no\_d

RTN

\*test\_h

P= 0

LA 002

LC 080

OUT=C

GOSBVL 01160

C=C&A X

?C=0 X

GOYES no\_h

C=R0

C=C-16 A

C=C-16 A

C=C-2 A

R0=C

\*no\_h

RTN

\*test\_b

P= 0

LA 002

LC 040

OUT=C

GOSBVL 01160

C=C&A X

?C=0 X

GOYES no\_b

C=R0

C=C+16 A

C=C+16 A

C=C+2 A

R0=C

\*no\_b

RTN

\*sprite

A=0 W

C=0 W

C=R0

D1=C

%(D1 pointe sur la position du sprite)

GOSUB data

$FFFFFFFFFFFF

\*data

C=RSTK

D0=C

%(D0 pointe sur le sprite 8 \* 12 pixels)

C=0 W

LC B

%(12 lignes = #Ch -1 = #Bh)

\*loop

A=DAT1 B

DAT0=A B

D0=D0+ 16

D0=D0+ 16

D0=D0+ 2

%(on descend d'une ligne de l'écran: 34 quartets)

D1=D1+ 2

%(on descend d'une ligne du pixel: 2 quartets)

C=C-1 A

GONC loop

C=0 W

RTN

\*clearsprite

A=0 W

C=0 W

C=R0

D1=C

%(D1 pointe sur la position du sprite)

C=0 W

LC B

%(12 lignes = #Ch -1 = #Bh)

A=0 B

\*loop2

DAT0=A B

D0=D0+ 16

D0=D0+ 16

D0=D0+ 2

%(on descend d'une ligne de l'écran: 34 quartets)

C=C-1 A

GONC loop2

C=0 W

RTN

@"

#  POUR LE PROGRAMMEUR CONFIRME:

### Avertissement

Ce chapitre concerne d'avantage le programmeurs initié que le débutant. je dévoile ici tout mes 'secrets' de programmation, c'est le moyen pour moi de vous montrer que la programmation n'est pas quelque chose d'innée, mais tout un art qui se peaufine petit à petit, de programmes en programmes. Ce recueil d'astuces et de conseils vous apportera les éléments essentiels à la programmation de jeu évolués tel: MARIOhp ou encore: MEGANOID. De plus, je pense qu'il est toujours utile de dévoiler ses méthodes afin que d'autre personnes, plus compétente, s'en inspire et les améliorent.

## Déplacer un sprite II

## Afficher un compteur

## Le double-écran

Lorsqu'on déplace un sprite à l'écran, il faut distinguer plusieurs étapes essentielles qui sont: On afficher le sprite, on efface le sprite, on le déplace, on réaffiche le sprite à sa nouvelle position. On est forcé d'effacer le sprite afin d'éviter de laisser une traînée lors du déplacement, ce qui produit un certain clignotement jamais très apprécié ! Pour parer ce problème, la technique consiste à afficher et effacer les sprites dans une zone déterminée en mémoire, et d'effectuer un transfert entre cette zone de la mémoire et l'écran, au moment exacte ou tout les sprites sont affichés. En clair, le jeu se déroule dans une zone de la mémoire à un adresse notée sur 5 quartets (ex: #80100h), on effectue de temps à autre un transfert de ce qui se passe en mémoire, à l'écran (à l'adresse: #8068Dh) exactement lorsque tout les sprites sont à l'état 'affichés'. C'est magique, plus aucun clignotements !

## Les niveaux de gris

Parce que les présentations dignes des meilleurs programmeurs sont en plusieurs niveaux de gris, c'est-à-dire avec du blanc, du noir, mais aussi du gris clair et foncé (comme sur une console portable de chez n..). On appelle cela aussi, le Greyscale. Le principe est simple, la mise en oeuvre astucieuse: On affiche deux images très rapidement, de manière à n'en distinguer qu'une seule. Pour cela, l'astuce consiste à effectuer l'affichage des images, que lorsque le rafraîchissement de l'écran en est à la première lignes. Autrement dit, on affiche une image autant de fois que l'écran de votre HP48 est renouvelé (ou balayé), c'est-à-dire 64 fois par seconde ! L'effet est saisissant mais peu s'avérer dangereux pour votre HP48 si vous ne tenez pas compte des conseils qui suivent: Toute modification des données qui vont suivre peut entraîner le blocage du rafraîchissement écran de la HP48, ce qui se solde au pire par la dégradation irréversible de l'écran. Ne partez pas en courant, je suis là pour vous expliquer !

Tout d'abord, votre image Greyscale est représentée par un objet graphique de 131 pixels sur 128 (64 \* 2 car deux images !).

Ex: Grob 131 \* 128

Pour vous ôter la fastidieuse tache de dessiner vous même vos greyscales dans l'environnement PICTURE de votre HP48, vous pourrez utiliser pour ce faire, l'outils TC (ou true colors) d'ATLAS.

Cet outil prend comme argument un Grob de 131 \* 128.

#  APPLICATIONS LUDIQUES

## Création de Mariohp:

## Création de Méganoid:

# Lexique:

quartets

hexadécimal

décimal

base

registre

champs

pointeur

pixels

donnée

GROB

buzzer

pile

variable

greyscales

code