Les GUIDES

Performance CALCUL

ASSEMBLEUR *sur* HP 48

Aspects pratiques & Programmation

Guy Toublanc - Christophe Nguyen

Terminales Scientifiques
Classes préparatoires scientifiques
Universités et Grandes Écoles

POLE

ASSEMBLEUR

sur

HP 48

Aspects pratiques & programmation

Guy Toublanc - Christophe Nguyen

TABLE DES MATIERES

LES GUIDES

Performance CALCUL

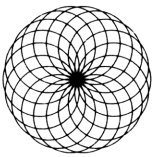
Au service des utilisateurs de calculatrices HP :

Le graphisme sur HP 48G/GX (en librairies, diffusion Belin)
Assembleur sur HP 48 (en librairies, diffusion Belin)
*La «Bible» de la HP 48G/GX (manuel avancé en français, 800 p)
*Disquette PERF3 (**toutes les sources de ce guide assembleur**)
*Disquettes PERF1 (S/SX) et PERF2 (S/SX/G/GX)
***Abonnement Formule Performance Calcul**
(chaque année, 6 numéros de 36 pages A4 + 2 guides de 64 pages)

Mathématiques pour lycéens et prépas

Les systèmes dynamiques (en librairies, diffusion Belin)
*Abonnement au magazine Tangente

**Les articles précédés d'un astérisque ne sont pas diffusés en librairies. Ils peuvent être commandés à : POLE, BP 87, 75622 Paris cedex 13.*
Prix : 195 FF («Bible» ou abonnement Perf Calcul), 48FF (disquettes PERF1 à PERF3) 200 FF (abonnement de 8 numéros à Tangente)



POLE

I.S.B.N. 2 - 909 737 - 14 - 4

INTRODUCTION

Parmi les calculatrices programmables, les HP48 se sont distinguées par des possibilités de créativité que l'on ne rencontre pas chez leurs concurrentes et cela, grâce à la possibilité d'utiliser trois langages :
- **le langage utilisateur** mis à la disposition par le constructeur.
- **le Rpl système**, plus riche et offrant des gains de rapidité très appéciables.
- **l'assembleur** reculant les limites de l'impossible par sa rapidité d'exécution et une gestion optimale de la mémoire.

En contrepartie, ces deux derniers langages nécessitent des compilateurs et plus d'attention de la part du programmeur.
L'objet de ce fascicule est d'assurer une initiation à l'assembleur dont les deux auteurs ont laissé entrevoir les richesses dans les revues Haute Performance (numéros 2 à 12, sous la signature de Christophe Nguyen) et dans Performance Calcul.

Grâce à ce guide, à vocation pratique, le lecteur pourra, nous l'espérons, très vite programmer en assembleur, en commençant par les programmes que nous lui offrons de refaire avec nous. Cela suppose évidemment qu'il connaisse les rudiments de l'utilisation courante de sa machine et de la programmation ordinaire.
Nous avons fait le choix de ne pas grouper les parties théoriques, mais de les disséminer sous forme de **fiches encadrées** au fur et à mesure des besoins des programmes. Si toutes les informations théoriques permettant la compréhension des programmes proposés sont fournies, il n'était pas possible en revanche d'avoir en 64 pages des ambitions d'exhaustivité. Nous vous renvoyons aux références bibliographiques de la page 64, signalées dans le texte par des crochets [...].

La disquette PERF3, proposée en option par les Editions POLE, contient :
-**les programmes** en assembleur de ce guide avec leurs fichiers sources ainsi que les programmes en langage utilisateur et les objets HP48 nécessaires à leur exécution.
- **des outils de développement sur HP48** et leur documentation (en anglais).
- **des outils de développement HP sur PC** et leur documentation (en anglais).
- **un désassembleur** de la Rom HP48 ainsi qu'**un décompilateur sur PC** d'objets HP48, programmes utilisant l'un des langages ou une combinaison de ceux-ci, bibliothèques, etc... Ces outils, très pratiques et utilisant les mnémoniques HP, sont l'œuvre de Jean-François Garnier.
- **un fichier "MS Word" de 90 pages contenant les nombreux descriptifs théoriques** qui n'ont pu figurer, faute de place, dans ce guide, et les principaux éléments de **la syntaxe de l'assembleur ASM Flash**, qui n'est pas le choix de ce fascicule, mais qui a de nombreux adeptes en France.

Si ce guide, par son aspect pratique, aide ses lecteurs à franchir les premiers obstacles rencontrés dans la programmation en assembleur par beaucoup de débutants, il aura atteint son but.

G.T. et C.N.

CALCULS ET MANIPULATIONS

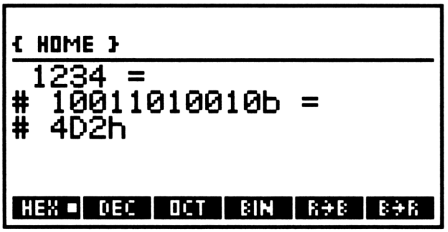
Dans ce fascicule :
✋ signale les calculs ou manipulations, au clavier avec la HP48.
☛ signale un point particulier ou des remarques d'ordre pratique permettant de mémoriser, faciliter les choix, mieux programmer.

UN PEU DE THÉORIE

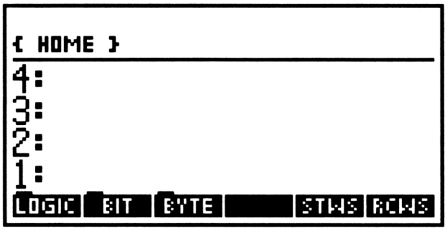
Nous nous efforcerons dans ce guide d'éviter les longs développements théoriques et d'aborder rapidement les aspects pratiques pour vous conduire plus vite vers la finalité : les programmes qui "font quelque chose". Quelques rappels sont néanmoins indispensables.

BASES DE NUMÉRATION

Les échanges entre numérations de **bases 10, 2** et **16** en particulier sont indispensables pour la suite. Grâce aux menus reproduits ci-dessous, la HP48 fournira une aide précieuse et un gain de temps appréciables dans les conversions.



menu MATH BASE 1ère page



menu MATH BASE 2ème page

Attention ! Les écrans ci-dessus sont ceux des HP48G/GX. Pour les HP48S/SX n'ayant que le menu MATH BASE, il faudra trouver la page correspondante.

Principe des bases:
- **base 10** (mode DECimal):
#123d = $\mathbf{1}*10^2+\mathbf{2}*10^1+\mathbf{3}*10^0$
- **base 2** (mode BINaire):
#10110b = $\mathbf{1}*2^4+\mathbf{0}*2^3+\mathbf{1}*2^2+\mathbf{1}*2^1+\mathbf{0}*2^0$
= #22d
- **base 16** (mode HEXadécimal):
1EAh = $\mathbf{1}*16^2+\mathbf{14}*16^1+\mathbf{10}*16^0$ = #490d

La représentation interne des nombres ou des données (se traduisant du point de vue physique par une suite d'états) se fait à l'aide de **digits** ou **bits** de valeur **0** ou **1**. La HP48 peut afficher en mode **BIN** jusqu'à 64 bits représentant la valeur 2^{64}**-1**. Cette suite de 64 bits constitue un **mot**.
Dans un mot, les bits extrêmes sont dits :
- **bit de poids faible** pour celui le plus à droite (de puissance 0)
- **bit de poids fort** pour celui le plus à gauche (de puissance la plus élevée)

☞ Faire **64 STWS DEC #12345 BIN** puis **14 STWS 8 STWS 4 STWS** . Constater l'effet : la disparition des *bits de poids fort* qui ne seront pas pris en compte dans les calculs.

Si l'expression d'un nombre en base 2 est la plus naturelle pour la machine, elle pose des problèmes de lisibilité pour le programmeur. Aussi la base 16 sera le plus souvent employée. Ceci se fera en regroupant les bits par 4 puisque 16=2^4 constituant des **quartets**, les caractères A à F désignant les valeurs décimales 10 à 15.

☞ faire **64 STWS DEC #12345 BIN** puis **HEX** pour obtenir :
 # 11 0000 0011 1001b → # 3039h
suivant le principe des bases.
474 #0 + → # 1DAh

LE MICROPROCESSEUR SATURN

Ce n'est qu'un microprocesseur 4 bits, Saturn, qui équipe la HP48 mais il est optimisé pour les opérations mathématiques de type DCB (décimal codé binaire) et pour une basse consommation d'énergie. On distinguera la **mémoire** où sont stockés tous les types d'objets HP48 sous forme de bits et les différents types de **registres** permettant de calculer, manipuler des données ou de contrôler certaines opérations.
Si pratiquement tous les types d'objets peuvent être importés de la mémoire, seuls ceux en mémoire Ram (mémoire vive) peuvent être modifiés. La Rom (mémoire morte) ne peut être altérée. Pour la suite la Rom contenant tout le code réalisé par HP sera identifiée par **Rom HP**.
Le travail du programmeur en assembleur consistera à manipuler des données déjà en mémoire ou à les créer. Ceci se fera à l'aide des **registres**.

LES REGISTRES

Afin de manipuler toute cette mémoire, le microprocesseur a besoin de conteneurs : des "variables" destinées à contenir des informations provenant de la mémoire ou même des adresses indiquant des positions particulières en mémoire. Ces conteneurs s'appellent des **Registres**. Ils font partie intégrante du microprocesseur et ne sont donc pas référencés dans les mémoire RAM ou ROM. Ce sont des emplacements mémoires particuliers destinés à être lus, modifiés, utilisés dans des calculs, etc... De tailles différentes, ils ont aussi différents usages.
Le microprocesseur **SATURN** est équipé de :

*** 4 registres de travail (ou de calcul) : A,B,C et D.**
Leur longueur est de 64 bits (16 quartets).
Ils sont essentiellement utilisés pour manipuler les données, et effectuer des calculs.
A et C sont spécialement solicités pour les opérations de Lecture/Ecriture avec la mémoire.

*** 5 registres de sauvegarde : R0,R1,R2,R3,R4**
Leur longueur est de 64 bits (16 quartets). Ils sont utilisés pour la sauvegarde des registres de calcul A,B,C et D. L'intérêt de sauvegarder des registres dans d'autres registres est qu'il n'y pas là d'échanges avec la mémoire : l'opération est une des moins couteuses pour le processeur.

*** 2 registres "pointeurs de données" : D0, D1**
Leur longueur est de 20 bits (5 quartets). Ils sont conçus pour contenir des adresses (rappel : une adresse tient sur 5 quartets = 20 bits). Les opérations de lecture/écriture s'effectuent au travers de l'adresse qu'ils contiennent. Ils sont en étroite relation avec les registres de calcul A et C, ces derniers servant à véhiculer l'information entre mémoire et registre.

*** 1 registre "pointeurs d'instruction" : PC (Program Counter)**
Sa longueur est de 20 bits (5 quartets). Comme on peut s'en douter, il contiendra une adresse, mais pas n'importe laquelle : à chaque instant, il contient l'adresse de la prochaine instruction à exécuter. Chaque opération réalisée par le microprocesseur modifie donc en conséquence la valeur de **PC**. L'utilisateur peut être amené à le modifier plus brutalement pour effectuer lui-même un saut inconditionnel à une adresse déterminée.

.../...

Les registres (suite)

- 1 registre "pointeur de champ" : P

Sa longueur est de 4 bits : 1 quartet. Il peut donc contenir un entier entre 0 et 15. En étroite relation avec les registres de travail A,B,C et D, son rôle est de désigner l'un des 16 quartets de ces registres. Il peut aussi être utilisé comme compteur de boucle (nombre d'itérations au plus 15). Les opérations d'incré-décré-mentation sur le registre P sont particulièrement rapides.

*** 2 Registres d'entrée/sortie : IN et OUT**

Ils sont dédiés aux opérations d'entrée/sortie du système. Le registre **IN** (registre des entrées) est long de 16 bits (4 quartets) et le registre **OUT** (registre des sorties) de 12 bits (3 quartets).

*** La pile des retours : RSTK**

Il ne s'agit pas d'un registre au même titre que les précédents, mais d'une pile permettant de mémoriser les adresses de retours des sous programmes en langage machine. Elle permet de stocker au maximum 8 adresses de 5 quartets chacune. C'est une pile de type LIFO : Last In First Out. La dernière adresse empilée sera la première dépilée.

*** Le bit CARRY**

Il s'agit d'un bit isolé au sein du microprocesseur "Saturn", dont le rôle consiste à signaler les retenues (par overflow : dépassement de capacité) survenant lors des opérations arithmétiques sur certains registres : une retenue finale "débordante" place le bit CARRY à 1.
Son rôle est aussi de fournir le résultat de tests logiques : si un test entre 2 registres donne la valeur VRAI, alors le bit CARRY est positionné à 1.
Lorsque le bit CARRY est mis à 1, on dit qu'il est **armé**.
Lorsque le bit CARRY est mis) 0, on dit qu'il est **désarmé**.

*** Le registre des "Status Bits" : ST**

Les "Status Bits" sont des drapeaux (flags) que le programmeur peut positionner selon ses besoins, pour indiquer et sauvegarder le résultat ou l'état de certaines "variables" qui lui sont propres. Ils sont au nombre de 16 et sont numérotés de 0 à 15. Les bits 12 à 15 sont réservés à l'usage interne de la HP48 : il est fortement déconseillé de les utiliser !
Ne pas confondre "Status Bits" et les Drapeaux Utilisateurs disponibles sur la HP48 en mode utilisateur : ils n'ont rien à voir entre eux.

*** Le registre des "Hardware Status Bits" : HS**

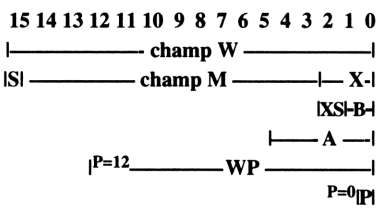
Il est constitué de 4 bits :
MP : (bit 3) "Module Pulled"
SR : (bit 2) "Service Request"
SB : (bit 1) "Sticky Bit"
XM : (bit 0) "External Module Missing"
Ces bits sont "armés" (en position "1") par des évènements matériels.
Seul le bit **SB** pourra nous être utile : il est armé lorsqu'une opération de décalage à droite (Right Shift) sur un registre de calcul se traduit par la sortie à l'extrémité droite de ce registre d'un bit à 1.

REGISTRES DE TRAVAIL : SUBDIVISION EN CHAMPS

Lorsque l'on désire utiliser le contenu d'un registre (lectures, écritures...), il n'est pas nécessaire d'opérer sur la totalité du registre. Pour cela, un registre (registre de travail ou registre de sauvegarde) peut être divisé en plusieurs parties, appelées **champs**, servant à référencer la partie du registre concernée pour un calcul donné.

Conséquence :
Un calcul ou une affectation par rapport à un champ ne se fait que pour les quartets concernés par ce champ. Les autres quartets restent inchangés. Les dépassements ou retenues surviennent à la limite du champ.

Descriptions des champs "usuels"
Notez la position du quartet 0, à l'extrême droite : quartet de poids le plus faible, et la position du quartet 15, à l'extrême gauche : quartet de poids le plus fort.



- Le champ **W** (word) : quartets 0 à 15 (totalité du registre)
- Le champ **S** (mantissa sign) : uniquement le quartet 15
- Le champ **M** (mantissa) : les 12 quartets numéros 3 à 14
- Le champ **X** (exponent) : les 3 quartets de poids le plus faible : 0,1,2
- Le champ **XS** (exponent sign) : uniquement le quartet no 2
- Le champ **B** (byte) : les 2 quartets de poids le plus faible 0 et 1
- Le champ **A** (adress) : les 5 quartets de poids le plus faible : 0,1,2,3,4

Les champs précédents sont fixes. Il existe encore 2 spécifications de champ variable, qui dépendent de la valeur en cours du registre **P** (dont on rapelle qu'il est long d'un quartet, et permet donc de représenter tous les entier de 0 à 15) :
- **Le champ P** : c'est le quartet dont le numéro est la valeur du registre P.
- **Le champ WP** : plage du quartet numéro 0 au quartet numéro P (inclus).
Ainsi, si P=15, WP équivaut au champ W, et si P=4, WP équivaut au champ A.

Remarques
☛ Ne pas confondre le champ A d'un registre et le registre de travail A.
☛ Le champ A est de loin le plus utilisé. De nombreuses instructions sont d'ailleurs codées de manière particulière lorsque ce champ est utilisé.
☛ Le champ B, contenant un octet (2 quartets) est très utilisé pour manipuler des caractères, ceux-ci étant codés sur 1 octet (les codes ASCII de 0 à 255).

Sélection du champ
La syntaxe du "Saturn" permet d'identifier, quand c'est nécessaire, le champ utilisé. Par défaut, c'est la totalité du registre (champ W) qui sera affectée.
Considérons par exemple l'instruction qui consiste à mettre à zéro le contenu d'un champ du registre de travail **C**. Dans le source en Assembleur (mnémoniques), l'instruction s'écrira par exemple : **C=0 A** *annule le champ A du registre C*

CRÉER UN PROGRAMME ASSEMBLEUR

La théorie, c'est bien, mais en pratique ? Ce chapitre vous indique la structure d'un programme assembleur, mais surtout vous donne des renseignements concrets quant à la façon de l'utiliser quand vous êtes parvenu(e) à l'écrire, ce qui ne saurait tarder...

STRUCTURE D'UN PROGRAMME ASSEMBLEUR

Comme pour réaliser un programme en langage utilisateur, en assembleur il faudra créer un texte avec les différentes instructions, à la différence près que cela nécessite un compilateur (n'existant pas en Rom HP) pour transformer ce texte en code exécutable. Ce texte se composera d'une suite d'instructions représentées par des **mnémoniques**.

Un listing de programme était généralement représenté en colonnes de la façon suivante :

(CODAGE) MNÉMONIQUE COMMENTAIRE

Depuis que le codage de l'instruction est automatiquement généré par l'assembleur utilisé, il n'apparaît plus dans la plupart des listings. Nous l'abandonnerons donc (sauf dans notre premier programme) au profit de la présentation :

MNÉMONIQUE COMMENTAIRE

- Les instructions proprement dites sont contenues dans la colonne mnémoniques.
- Le commentaire, précédé d'un astérisque, ne sera pas lu par le compilateur. Il sera en général écrit en italique dans ce fascicule.

Le mnémonique peut se diviser en 3 parties :

Exemple : **debut A=DAT1 A**

* **Le label** (facultatif), utilisé pour référencer une ligne de programme, appelée par exemple lors d'un saut (**GOTO**...).
* **L'instruction** elle-même.
* **Une 3ème colonne** (facultative), pouvant contenir soit un champ, soit une constante.

Notation :
Pour alléger les pages de rappels théorique, des notations, en usage depuis 1984 pour l'assembleur Saturn, seront introduites progressivement et utilisées.

☛ **fs** désigne de manière générale un **champ** (W, A etc...)

☛ **r,s** désigne un **couples ordonné** de **registres de travail** (A,B), (D,C) etc... pour lesquels les opérations sont possibles.

Sauvegarde et restauration des registres
Certains registres, comme D0, D1, B et D, sont utilisés de manière interne par la machine. Il convient de les sauvegarder avant usage dans un programme, et de les restaurer à la fin, ce qui correspond à la première et dernière instruction d'un programme propre.

L'ASSEMBLEUR ET LES AUTRES LANGAGES

L'assembleur n'est pas une fin en soi. Il peut être associé aux autres langages pour constituer un programme. Pourquoi vouloir refaire ce qui existe déjà en Rom HP ? Aussi les programmes en assembleur pourront-ils être combinés avec des programmes en Rpl utilisateur.

Les compilateurs que nous recommandons facilitent la programmation mixant assembleur et autres langages. Aussi un minimum de Rpl système complètera-t-il certains programmes nécessitant des arguments, ce qui assurera une sécurité tout en évitant d'allonger la partie assembleur pour vérifier la présence et le type des arguments.

Il faut savoir que même avant d'allumer l'affichage, la HP48 n'est pas inactive. Le programme principal, *le noyau*, attend l'appui sur la touche [ON]. Il se chargera de différentes tâches parmi lesquelles le rafraîchissement de l'affichage de la pile, la détection d'un appui de touche, l'exécution d'autres programmes avec reprise en main lors des retours au Rpl.

Un programme en assembleur est un objet HP48 commençant par une adresse préfixe (**prologue**) qui, à la fois caractérise le type d'objet Code et constitue un programme analysant la suite : longueur (hors prologue de l'objet Code) puis code exécutable.
On devra toujours avoir à l'esprit que l'objet CODE vient s'intercaler dans une suite d'actions s'enchaînant logiquement. Pour cela, *4 valeurs seront à sauvegarder en début de programme puis à restituer en fin de programme avant de redonner la main au Rpl.* Ce sont :
D0: pointeur d'instructions
D1: pointeur de données sur la pile
B[A]: pointeur de pile des retours
D[A]: (mémoire disponible)/5

D'où la structure suivante du texte d'un fichier source de programme:

```
CODE
    .....          opérations
    .....          avec la pile
    GOSBVL =SAVPTR
    .....          corps du
    .....          programme
    GOSBVL =GETPTR
    A=DAT0 A       lit le pointeur pour
                   l'objet Rpl suivant à exécuter
    D0=D0+ 5       avance le pointeur
                   d'instruction
    PC=(A)         branche à l'instruction
                   suivante
ENDCODE
```

CODE et **ENDCODE** indiquent au compilateur qu'il s'agit d'une séquence *assembleur* se terminant à ENDCODE.

Avant la sauvegarde des valeurs de **D0**, **D1**, **B[A]** et **D[A]** par appel au sous-programme en Rom HP de mnémonique **SAVPTR**, diverses opérations peuvent être effectuées. La sauvegarde doit permettre une restitution des valeurs nécessaires à la sortie du programme pour enchaîner avec les opérations suivantes.
La restitution des pointeurs se fera par appel au programme en Rom de mnémonique **GETPTR**.
GOSBVL correspond à un appel de sous-programme en Rom et donc d'adresse absolue (ex: SAVPTR adr. # 0679B).

☛ Il existe aussi un programme en Rom faisant l'équivalent des 4 lignes terminant le programme, et le mnémonique **GOVLNG** indiquant un saut absolu à ce programme.

Ecrire : **GOVLNG =GETPTRLOOP** permettra de raccourcir le programme.

ARITHMÉTIQUE GÉNÉRALE

Pour les additions, soustractions et complémentations, le résultat dépend du mode arithmétique imposé par les instructions **SETHEX** (valeur des quartets de 0 à F) et **SETDEC** (valeurs 0 à 9). D'autre part ces opérations peuvent affecter le **bit Carry** (bit de retenue) lors d'un dépassement de capacité ou de l'équivalent d'un changement de signe.

Une opération entre 2 registres se fait sur le même champ. Les différentes opérations sont :
- mise à zéro du champ d'un registre
- copie d'un registre dans un autre
- échange (EX) des valeurs de 2 registres
- doublement de valeur
- addition ou soustraction de 1
- addition ou soustraction d'une constante (CON) (valeur 1 à 16)
- suivant le mode, complément à 2 ou à 10 (r=-r)
- suivant le mode, complément à 1 ou à 9 (r=-r-1)
- somme de 2 registres.

Dans les exemples qui suivent,
(r,s) désignent (A,B) (A,C) (B,A) (B,C) (C,A) (C,B) (C,D) ou (D,C)
fs désigne A, P, WP, XS, X, S, M, B ou W.

r=0	**fs**	*exemple:* **B=0**	**W**
r=s	**fs**	*exemple:* **A=C**	**B**
s=r	**fs**	*exemple:* **C=A**	**A**
rsEX	**fs**	*exemple:* **ACEX**	**M**
r=r+r	**fs**	*exemple:***D=D+D**	**X**
r=r+1	**fs**	*exemple:***A=A+1**	**S**
r=r-1	**fs**	*exemple:***B=B-1**	**WP**
r=r+CON	**fs,d**	*exemple:***A=A+CON**	**A,6**
r=r-CON	**fs,d**	*exemple:***D=D-CON**	**X,12**
r=-r	**fs**	*exemple:***A=-A**	**XS**
r=-r-1	**fs**	*exemple:***C=-C-1**	**A**
r=r+s	**fs**	*exemple:***C=C+A**	**B**
s=r+s	**fs**	*exemple:***A=C+A**	**A**

☛ L'addition ou la soustraction d'une constante n'affecte pas le bit Carry pour les champs S, P, WP et XS ce qui peut entraîner une erreur : il n'y a en effet aucune possibilité de détecter un dépassement de capacité.

ARITHMÉTIQUE RESTREINTE

Il s'agit de la différence entre 2 registres.
(r,s) = (A,B) (A,C) ((B,C) (C,A) ou (D,C)
fs = A, P, WP, XS, X, S, M, B ou W.

r=r-s	**fs**	*exemple:***A=A-B**	**B**
r=s-r	**fs**	*exemple:***B=C-B**	**A**
s=r+s	**fs**	*exemple:***A=A-C**	**M**

☛ On peut simuler une partie des opérations ci-dessus avec les commandes du menu **BASE**, 2 niveaux de pile remplaçant 2 registres, en fixant les champs avec **n STWS** où n = 4*m, m étant le nombre de quartets : par exemple pour A, n = 4*5 =20.
. Expérimenter dans les 3 bases
. Constater l'effet des dépassements de capacité
. Retrouver le bit de poids fort disparu de l'affichage en augmentant n.
En faisant en mode BIN **# 11011010 DUP +** on constate le décalalage d'un rang à gauche de tous les bits si n > 8.

transferts mémoire ⇦⇨ registres

Le microprocesseur Saturn étant un 4 bits, la plus petite quantité d'information échangeable entre mémoire et registres de travail est le quartet.

La mémoire est composée d'une suite de quartets, chacun ayant une adresse dans l'étendue #00000h à # FFFFFh. Les Registres D0 et D1 (5 quartets) permettent de pointer à une adresse.
Voici nos conventions pour ce qui suit :
r = A ou C **dp** = D0 ou D1
n: expression de valeur 1à 16
nnnnn: expression de valeur 0 à # FFFFF

☛ **r[fs]** spécifiera le contenu du champ fs du registre r ou **r[n]** le contenu du quartet n ou **r[n1....n2]** le contenu des quartets n1 à n2.

Pour les pointeurs
rdpEX *échange r[A] avec dp. Ex*: **AD0EX**
rdpXS *échange r[0 à 3] avec dp. Ex*: **AD1XS**
dp=r *copie r[A] dans dp. Ex*: **D0=C**
dp=rS *copie r[0 à 3] dans dp. Ex:* **D0=AS**
dp=dp+ n *incrémente dp. Ex:* **D0=D0+ 16**
dp=dp – n *décrémente dp. Ex*: **D1=D1 – 16**
dp=(2) nnnnn *charge les quartets 0 et 1 de nnnnn dans dp. Ex*: **D1=(2) #F1**
dp=(4) nnnnn *fait de même avec les quartets 0 à 3 . Ex*: **D0=(4) #A3F1**
dp=(5) nnnnn *fait de même avec les quartets 0 à 4. Ex :* **D0=(5) #0011F** ou **D0=(5) =IRAM@**

☛ Une opération d'incrémentation ou décrémentation affecte le **bit Carry**, mis à un en cas de dépassement de capacité (mode HEX implicite).

Transferts de données pointées par **D0** ou **D1**, (données contenues dans le champ fs ou prenant n quartets).

mémoire → registres :

r=DAT0 fs ou n	*Ex :*	**A=DAT0**	**A**
	ou	**A=DAT0**	**5**
r=DAT1 fs ou n	*Ex:*	**C=DAT1**	**B**

registres → mémoire :

DAT0=r fs ou n	*Ex:*	**DAT0=A**	**S**
	ou	**DAT0=C**	**12**
DAT1=r fs ou n	*Ex:*	**DAT1=C**	**WP**

LE REGISTRE P : INSTRUCTIONS

n est une expression de valeur 0 à 15 (mode HEX implicite). Le **registre P** (1 quartet) est associé au **registre C** pour les échanges de valeurs. Les opérations arithmétiques affectent le **bit Carry**.

P = n	*P prend la valeur n*	**P = 4**
P=P+1	*incrémentation*	
P=P–1	*décrémentation*	
C+P+1	*ajoute la valeur P+1 à C[A]*	
CPEX n	*échange C[n] avec P. Ex:*	**CPEX 15**
P=C n	*P prend la valeur C[n]. Ex:*	**P=C 0**
C=P n	*C[n] prend la valeur de P.*	**C=P 2**

CHARGEMENT DE CONSTANTES

Ce chargement n'est possible qu'avec les **registres A** (LA...) et **C** (LC...) et à partir du quartet de **rang P** (vers la gauche). Nous utiliserons les conventions suivantes :
h chiffre hexadécimal, 16 chiffres possibles
i entier de 1 à 8,
nnnnnnnn expression de valeur en hexadécimal de 0 à FFFFFFFF
c caractère ASCII.

mnémoniques		**exemples**
LAHEX	h ....h	LAHEX. 2AC19
LA(i)	nnnnn	LA(3) 34*55
	équivalent de	LAHEX 74E
LCHEX	h ....h	LCHEX. C19
LC(i)	nnnnn	LC(4) 4096
	équivalent de	LCHEX 1000
LAASC	c.....c	LAASC 'ABC'
	équivalent de	LAHEX 414243
LCASC	c.....c	LCASC 'abc'
	équivalent de	LCHEX 616263

☛ Le chargement se fait de façon circulaire à partie de la position P:

P= 0 C=0 W LCHEX 123456
C contient: 0000000000123456
P= 13 C=0 W LCHEX 123456
C contient: 4560000000123

Cette particularité permettra des astuces de programmation.

PROGRAMMATION ET COMPATIBILITÉ

Actuellement les HP48 vont des versions A à R. Si la compatibilité est assurée en amont pour le langage utilisateur, il n'en est pas de même pour le Rpl système ou l'assembleur, d'où des programmes ou bibliothèques inutilisables d'une machine à l'autre. Pourtant, sauf dans de rares cas, il est possible d'assurer cette compatibilité.

Les problèmes d'adresses ...

De ce point de vue il faut distinguer les objets HP48 (programmes, nombres, chaînes etc..) existant en Rom HP et les zones de mémoire Ram où sont stockées des données (l'objet graphique de la pile ou du PICT par exemple).

Avec l'arrivée des HP48G/GX, ces zones de mémoire Ram ont été décalées rendant beaucoup de programmes pour HP48S/SX incompatibles.

* Pour ce qui existe en Rom, HP a donné une liste d'adresses garanties fixes et appelées points d'entrées supportés, par exemple SAVPTR, GETPTR.

* Pour les zones de données en Ram il existe des adresses fixes où sont stockées les adresses variables de zones de données. Par exemple pour déterminer l'adresse du premier quartet de l'objet graphique de la pile on pourra faire :

```
D0=(5)          #13299
A=DAT0 A
D0=A
A=DAT0 A
D0=A
```

mais aussi plus simplement:

```
GOSBVL          =D0->Row1
```

faisant pointer D0 sur le dit quartet.

... et les autres

Les HP48G/GX se distinguent aussi des S/SX par deux points trop souvent ignorés des auteurs :
- une plus grande rapidité d'exécution dont il faut tenir compte avec les boucles de temporisation.
- un affichage plus contrasté qui est une amélioration pour les images statiques mais pose des problèmes pour les animations graphiques qui laissent une impression d'images baveuses en fonction de la vitesse de déplacement. Il faut donc trouver un compromis entre vitesse et netteté.

Des programmes ou bibliothèques compatibles présentent les avantages suivants :
- ils évitent les "plantages" des machines par utilisation d'une version inadaptée.
- ils simplifient les problèmes d'archivage.
- ils simplifient les problèmes de publication de listage de programmes: une seule version.
- ils évitent la duplication des numéros d'identification des bibliothèques qui posent déjà beaucoup de problèmes.
- ils évitent de doubler les pages des listages de programmes d'un livre, par exemple *faites vos jeux en assembleur*, gonflé artificiellement en volume et en prix. et aurait permis aux possesseurs de HP48GX d'utiliser les 4 jeux et non un seul !
- ils permettent aux possesseurs des plus anciennes versions de HP48 de pouvoir bénéficier des créations pour les versions plus récentes et de ne pas se sentir abandonnés.

LES OUTILS DE L'ASSEMBLEUR

Deux possibilités sont offertes aux programmeurs sur HP48 : développer sur HP48 ou sur PC. Pour les progammes très importants la solution PC devient une nécessité pour des raisons de mémoire. Il existe, pour les deux types de développement, des outils assurant le confort d'utilisation souhaitable.

ASSEMBLAGE SUR HP48

HP48 et petite mémoire

Les pionniers de la programmation en assembleur ne disposaient que de HP48SX avec 30 Ko de Ram disponible et parfois sans extension mémoire. Dans ces conditions la première bibliothèque pour assembler, tenant compte de ces contraintes, présentait des différences avec la syntaxe HP des compilateurs pour Saturn existant depuis 1984 (pour le HP71). C'est le cas du célèbre compilateur français pour HP48, **ASM-FLASH,** qui fut le premier digne de ce nom. Aujourd'hui il est presque inconcevable de pratiquer l'assembleur sur HP48 sans extension mémoire par cartes verrouillables, pour des raisons de sécurité.

Un éditeur de chaînes comme STRING WRITER procurera un indéniable confort d'utilisation pour la création des fichiers sources, au moins pour les programmes longs.

HP48 et mémoire confortable

Certaines solutions ont l'avantage d'être compatibles avec les outils HP pour PC, permettant ainsi de produire des fichiers sources compréhensibles par les programmeurs de toute la planète.

* Ce fut d'abord le **'System-Rpl Development Toolkit for the HP48'** de Detlef Mueller & Raymond Hellstern, dont une version compatible HP48SX/GX se trouve dans le répertoire RPL48 de *Goodies Disc 9*. Il faut disposer de 53 Ko dans une carte en port 1 pour GX ou SX, ou port 2 pour SX. Un fichier de 64 Ko documente ces merveilleux outils.

* Puis, dans le même esprit et encore plus merveilleuse, la bibliothèque **JAZZ** de Mika Heiskanen intégre un débogueur de Jan Brittenson. Elle fournit un éditeur et un visualiseur spécifiques pour les fichiers sources intégrant une tabulation. On trouvera cette bibliothèque et la table des points d'entrée supportés dans le répertoire JAZZ de *Goodies Disc 10* (ou dans PERF3, la disquette en option de ce fascicule) avec d'autres utilités et une documentation. Les contraintes pour le stockage de ces outils sont les mêmes que précédemment et demandent encore un peu plus de place, 96 Ko. Mais quel confort !

Les programmes listés dans ce guide, qu'ils soient en assembleur pur ou mixés avec du Rpl système, peuvent être assemblés par les commandes de ces bibliothèques.

DESASSEMBLAGE SUR HP48

L'opération de désassemblage est utile pour explorer la Rom HP, pour décompiler des programmes faits par d'autres ou pour reconstituer les fichiers sources perdus de vos programmes. Les outils ci-dessus permettent ce désassemblage avec aisance et toujours suivant la syntaxe et les mnémoniques HP.

Il faut tout de même signaler que tout décompilateur a ses limites, en particulier lorsque les programmes contiennent des tables de données qui ne correspondent donc pas à des instructions. Là il faut désassembler par zones de mémoire et reconstituer le tout. Ce travail ne pourra se faire qu'avec une bonne compréhension de la programmation.

ASSEMBLAGE SUR PC

Les outils HP

On trouvera les outils d'assemblage sur PC sur la diquette *Goodies Disc 4* (ou sur PERF3), avec la documentation à la fois sur le Rpl système et l'assembleur. Cette documentation n'est pas complète, manque de qualités pédagogiques et a été quelquefois critiquée, mais elle a le mérite d'exister et de servir de référence pour les programmeurs du monde entier.

Voici une méthode d'utilisation très simple puisqu'elle se résumera à exécuter un fichier batch pour assembler des programmes tels que ceux présentés ici.

☛ créer l'arborescence :

HP48

↳ **INCLUDE**

↳ **BIN**

avec dans le répertoire **INCLUDE** les fichiers:
ENTRIES.O, ENTRIES.A et BINHD.O
et dans le répertoire **BIN**:
RPLCOMP.EXE, SASM.EXE,
SLOAD.EXE, A48.M, ASS.M, ASS48.BAT.

☛ ajouter dans le fichier **AUTOEXEC.BAT** la ligne suivante:
SET SASM_LIB=C:\HP48\BIN

☛ dans le répertoire INCLUDE le fichier **BINHD.O** sera optenu en créant dans le répertoire BIN le fichierle BINHD.S avec le texte:

```
NIBASC /HPHP48-A/
```

puis en compilant avec la séquence **SASM BINHD.S**. Le fichier BINHD.O sera copié dans le répertoire INCLUDE et disparaîtra du répertoire BIN avec BINHD.S.

Voici les fichiers pilotant la compilation :

ASS.M

```
OUTPUT ASS.O
LLIST ASS.LR
REL AS.O
END
```

A48.M

```
OUTPUT AS
LLIST AS.LR
SUPPRESS XREF
SEARCH C:\HP48\INCLUDE\ENTRIES.O
REL C:\HP48\INCLUDE\BINHD.O
REL ASS.O
END
```

ASS48.BAT

```
RPLCOMP AS.S AS.A
SASM AS
SASM.OPC
SLOAD ASS.M
SLOAD -H A48.M
```

Le fichier source à taper en simple ASCII portera toujours le nom **AS.S**.

Pour assembler le fichier source AS.S il suffit d'activer le fichier batch ASS48.BAT qui génèrera 7 fichiers dont :

- **AS.L** fichier où seront déclarées les erreurs lors de la première phase d'assemblage, donc à visualiser après assemblage pour contrôle.
- **AS.LR** fichier où l'on trouvera éventuellement les erreurs issues du fichier sources avec des points d'entrée non répertoriés dans le fichier ENTRIES.O. AS.LR sera donc aussi à visualiser pour contrôle.
- **AS** fichier binaire, résultat de la compilation, à exporter dans la HP48.

Avant toute nouvelle compilation il faudra détruire les fichiers générés, ce qui pourra se faire avec le fichier batch suivant :

DELAS.BAT:

```
DEL AS.A
DEL AS.L
DEL AS.O
DEL ASS.LR
DEL ASS.O
DEL AS.LR
DEL AS
```

Le fichier source **AS.S** et le fichier binaire **AS** du programme assemblé pourront être copiés dans un répertoire d'archivage et renommés.

En résumé,
☛ Taper le fichier source **AS.S**,
☛ Compiler en lançant **ASS48.BAT**.
Cette procédure est unique à la fois pour les programmes en assembleur ou mixtes: Rpl et assembleur.

Le fichier **ENTRIES.O** pourra être actualisé: il suffit d'ajouter des points d'entrées dans le fichier en ASCII **ENTRIES.A** et procéder comme ci-dessus pour le fichier BINHD.O.
On trouvera une version actualisée de ENTRIES.A avec les outils de décompilation de Jean-François Garnier (voir ci-dessous *désassemblage sur PC*)

Développer sur PC, les avantages :
- lecture plus aisée des listings et possibilité d'impression immédiate.
- pas de limite pour la longueur des listings et larges possibilités de commenter permettant de reprendre un travail sans devoir tout repenser.
- pas de problème de perte des fichiers sources

Développer sur PC, les inconvénients :
Le transfert entre PC et HP48 des programmes n'est pas immédiat et l'énergie des piles est mise à contribution.

DESASSEMBLAGE SUR PC

Comme sur HP48, il est possible de désassembler la Rom HP48 transférée sur PC ou de décompiler des objets HP48 sauvegardés sur PC. Jean-François Garnier a développé des outils très pratiques pour cela et disponibles dans la disquette PERF3. La documentation indique en particulier la méthode de transfert de la Rom HP48.

Les mnémoniques des points d'entrée soutenus par HP apparaissent dans les listages ainsi que ceux des points d'entrée qui sont restés fixes pour les versions S/SX ou G/GX et documentés sommairement par Mika Heiskanen mais constituant une liste impressionnante. Les listings sont ainsi nettement plus parlants.

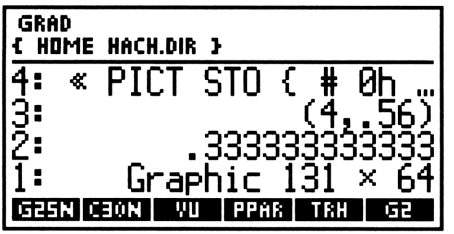
Avantages du désassemblage sur PC avec ces outils :
- il est possible d'avoir sur disque dur toutes les versions commercialisées de Rom HP48, permettant de contrôler les changements éventuels d'adresses évitant ainsi les programmes incompatibles entre versions HP48.
- profiter de l'écran d'un moniteur de PC pour avoir une vision beaucoup plus large des listings et donc une meilleure compréhension.
- rediriger la décompilation des objets HP48 vers un fichier texte donc imprimable sur papier et permettant une étude indépendante d'un PC.
Comme pour la décompilation sur HP48, tout n'est pas automatique : des programmes "biscornus" avec des tables ou des bibliothèques bien protégées par des astuces nécessiteront un désassemblage partiellement manuel.

Et les autres outils ?
Il a été fait allusion, au début, à ASM-FLASH qui fut un indispensable outil mais n'offrant pas le confort de ceux décrits ici. D'autres outils existent à la fois sur HP48 et sur PC mais ne respectant pas la syntaxe HP (assembleur et désassembleur intégrés au Méta Kernel et compatible ASM-FLASH) ou n'ayant pas été actualisés pour HP48 (par exemple AREUH pour PC). Il n'était pas possible de faire des fichiers sources utilisables par tous les compilateurs, aussi le choix s'est arrêté sur ceux d'utilisation universelle et capables de décompiler du Rpl et de l'assembleur.

PREMIERS PROGRAMMES

Nous voici en phase opérationnelle. L'essentiel du contenu des chapitres qui suivent sera consacré à des exemples de programmes. Et la théorie, direz-vous ? Elle fera l'objet d'encadrés qui seront souvent des rappels pour ceux qui ont suivi la rubrique d'assembleur de Haute Performance ou Performance Calcul. En route, et surtout, ne vous découragez pas si vous ne comprenez pas tout immédiatement. C'est normal, il vous faudra un temps d'adaptation. N'hésitez pas à revenir plus tard sur les passages qui vous paraissaient obscurs.

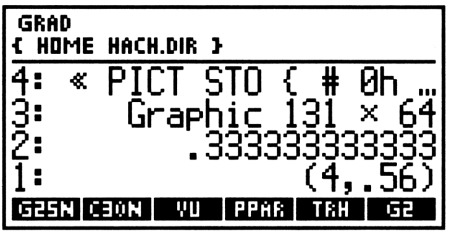
UNE MANIPULATION DE PILE



Sur cet écran de HP48, 4 objets sont affichés :
- un objet graphique rappelé sur la pile depuis la variable G2
- l'inverse de 3 obtenu en tapant **3 1/x**
- le nombre complexe tapé au clavier
- le programme rappelé sur la pile depuis la variable VU.

En fait, sur la pile il y a 4 adresses correspondant à 4 groupes de 5 quartets contenant l'adresse des objets affichés: pour les nombres (réel et complexe) ce sera dans la zone des objets temporaires, pour le graphique et le programme ce sera l'adresse des variables qui les contiennent. L'affichage est le résultat de la gestion du noyau pour le rendre compréhensible par l'utilisateur. Lors de l'entrée dans un programme assembleur nous auront la situation suivante:
- D1 pointe sur le niveau 1 de la pile, ici l'adresse ou l'on trouvera l'adresse de la variable contenant le graphique.
- P=0, mode HEX, D0 pointant sur la prochaine instruction Rpl à exécuter.

Voici, en haut de la page ci-contre, un petit programme échangeant les niveaux 1 et 3 de la pile pour obtenir l'écran :



Ce programme est présenté avec une colonne du code hexadécimal correspondant aux instructions et au type d'objet Code. On voit que le code généré lors de l'assemblage est de longueur variable en fonction des instructions et que le choix d'instructions pour arriver au même résultat n'est pas indifférent pour des raisons d'encombrement et de rapidité de programme. Si cette façon de présenter les programmes avec le code a été utilisée systématiquement dans [HtePerf] et [PerfCalc], elle n'est fournie ici qu'à titre d'exemple et sera abandonnée car elle ne se justifie plus avec l'utilisation de compilateurs, l'assemblage à la main ne se pratiquant plus.

Programme **SWAP13** (17,5 octets CRC # 24A3h)

```
CCD20  CODE                                  * prologue de l'objet Code: adresse # 02DCCh
E1000                                        * inversée comme sur cette deuxième ligne la
                                             * longueur de l'objet hors prologue
143            A=DAT1          A             * A(A):=adresse du graphique du niveau 1
179            D1=D1+          10            * D1 pointe le niveau 3 pour le nombre complexe
147            C=DAT1          A             * C(A):=adresse du complexe du niveau 3
141            DAT1=A          A             * charge l'adresse du graphique pour le niveau 3
1C9            D1=D1-          10            * D1 repointe sur le niveau 1
145            DAT1=C          A             * charge l'adresse du complexe pour le niveau 1
8D465D2        GOVLNG          =Loop         * retour au Rpl
       ENDCODE
```

GOVLNG =Loop remplace les 3 lignes:
A=DAT0 A D0=D0+ 5 PC=(A)
Les pointeurs n'ont pas pas été sauvegardés (pas d'appel à SAVPTR) car on ne les a pas modifiés, de même P=0 et mode HEX pour la sortie sont inchangés.

Pour un assemblage sur HP48, la chaîne contenant le texte ci-dessus étant sur la pile, activer la commande ASS de la bibliothèque JAZZ pour obtenir l'objet Code à stocker par exemple dans la variable SWAP13. Il ne reste plus qu'à activer cette nouvelle commande pour échanger les niveaux 1 et 3.

Attention: le programme ne vérifie pas s'il y a au moins 3 niveaux de pile, donc mettre au moins 3 objets sur la pile.

LES BOUCLES ET LES SAUTS

En programmation assembleur, les boucles jouent un grand rôle. Exemple classique de tâche : un quartet est transféré de la mémoire dans un registre de travail, il est modifié puis chargé à son adresse, on avance d'un quartet et on recommence le processus. Ces actions répétitives seront comptabilisées à l'aide d'un compteur.

Un type de structure de boucle

```
      LC(2)  20        pour faire 20 +1 tour
boucle ————————        label pour le saut en
       ————————        fin de boucle
       ————————        séquence d'opérations
       C=C-1  B        compteur décrémenté
       GONC boucle     si compteur ≥ 0 alors
                       branchement à boucle
```

GONC signifie saut au label si le Carry n'est pas posé par une opération ou à la suite d'un test positif. Ce **saut** est **de type court** : distance de saut (offset) 128 quartets vers adresses basses, 127 quartets vers adresses hautes.

instructions de saut en général

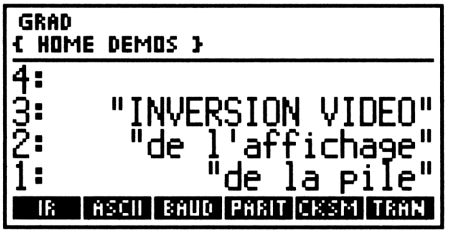
sauts relatifs à un label
GOTO sans condition offset :-2048 à 2047
GOYES après un test positif offset de type court dépendant du type de test le précédant
GOC saut si Carry posé offset: -128 à 127
GONC voir ci-dessus
GOLONG sans condition offset: -32768 à 32767

saut à une adresse absolue

GOVLNG sans condition

UN PROGRAMME D'INVERSION VIDÉO

Appliquons nos premières connaissances.à un programme de graphisme. Ce thème de programmation a été traité par C. N. dans [HtP] No 7 et adapté ici par G.T.



Ce que l'on veut faire : inverser l'affichage pour obtenir l'écran suivant :

L'affichage initial est divisé en 2 : la partie noire sur fond blanc (objet graphique de la pile: grob stack) et la partie sur fond noir de la barre des menus. Chacune correspond à une zone mémoire où est stocké l'objet graphique de ces 2 parties d'affichage. L'affichage est composé de 64 lignes de 131 pixels visibles, mais en réalité pour l'objet graphique chaque ligne est de longueur 136 points corespondant à 136 bits et donc 34 quartets. Un point est noir si le bit correspondant a la valeur 1 sinon 0.

Pour créer l'inversion graphique, il suffit d'inverser chaque bit.

avec **4 STWS** et en mode **BIN** pour travailler sur un quartet faisons : **#0 #0 13 +** , résultat : **# 0b** puis **− 1 −**

 # 1101b , résultat : # 0010b

les bits ont été inversés, c'est l'opération obtenue aussi en assembleur avec **r = − r − 1 fs**

Principe de programmation

On sauvegarde les pointeurs puis on prépare les données pour l'exécution de la boucle qui réalise l'inversion vidéo :

☛ On fait pointer D0 sur le premier quartet du grob stack.

☛ L'affichage de la pile correspond à 56 lignes de 34 quartets, soit (34*56/16) fois 16 quartets. Ce nombre est le point de départ d'un compteur dégressif qu'on diminue de 1 et qu'on stocke en C(B).

☛ La boucle prendra 16 quartets du grob stack, inversera les bits, replacera les 16 quartets, D0 pointera sur les 16 quartets suivants et le compteur sera décrémenté.

Programme **INVID** *(27,5 octets CRC: # 62AFh):*

```
CODE
        GOSBVL      =SAVPTR       * Sauvegarde des registres
        GOSBVL      =D0->Row1     * D0 pointe le 1er quartet du grob stack
        LC(2)       (34*56/16)-1  * Compteur pour 118+1 itérations

loop    A=DAT0      16            * Lecture de 16 quartets du grob stack
        A=-A-1      W             * Pour l'inversion vidéo
        DAT0=A      16            * Ecriture des 16 quartets transformés
        D0=D0+ 16                 * D0 pointe sur les 16 quartets suivants
        C=C-1  B                  * Décrémentation du compteur
        GONC   loop               * Si valeur compteur ≥ 0 alors encore un tour
        GOVLNG      =GETPTRLOOP   * Restaure les pointeurs et retour au Rpl
ENDCODE
```

Assembler et stocker ce programme dans la variable INVID puis pour voir l'effet taper le mini-programme: « INVID 7 FREEZE » pour geler l'affichage. Un appui sur une touche rétablit l'affichage normal.

On a vu que certains sauts sont assujettis au résultat d'un test. La prochaine utilisation de ces sauts nécessite la lecture de l'encadré ci-dessous.

LES TESTS

Comparaisons de valeurs de registres
On adoptera les conventions suivantes :
(r,s) = (A,B) (A,C) (B,A) (B,C) (C,A) (C,B) (C,D) ou (D,C)
fs = A, P, WP, XS, X, S, M, B ou W.

Un test sera suivi de **GOYES label** pour un saut ou **RTNYES** pour un retour de sous-programme si le test est positif.

Structures pour un test d'égalité :	**? r = s**	**fs**	**?r = 0**	**fs**			
Pour l'inégalité :	**? r # s**	**fs**	**? r # 0**	**fs**	**? r > s**	**fs**	
	? r < s	**fs**	**? r ≥ s**	**fs**	**? r ≤ s**	**fs**	

Exemple : **?C=D S**
GOYES Label1
ou **RTNYES** *pour retour de sous-programme*

Tests sur les bits des registres
Conventions : n expression de valeur 0 à 15 (pour tous les tests suivants), r = A ou C.

?rBIT=0 n **?rBIT=1 n**

Tests avec le pointeur P

?P= n **?P# n**

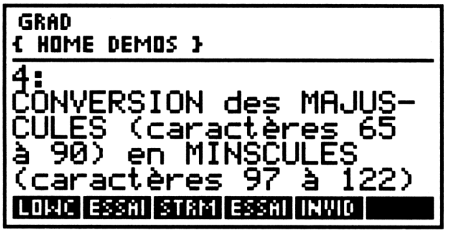
Tests avec les status bits
Ces "status bits", au nombre de 16 (0 à 15) se divisent en 2 groupes :
- de 0 à 11 ils composent le registre ST dont la valeur peut être échangée avec C(X).
- de 11 à 15 pour le système d'exploitation

?ST=0 n **?ST=1 n**
?ST#0 n **?ST#1 n**

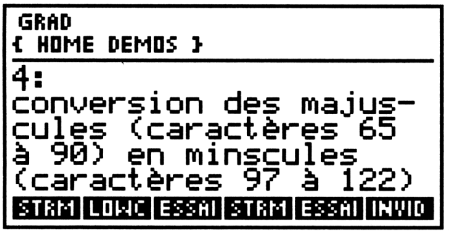
Tests avec les Hardware Status Bits
Seul le Sticky Bit (**SB**) nous intéresse. Il est armé par perte du bit de poids faible d'un registre lors d'une opération de décalage à droite. Il est désarmé par **SB=0**.

?SB=0

UNE MANIPULATION DE CARACTERES



Ce que l'on veut faire : convertir les lettres majuscules d'une chaîne de caractères (en haut) en minuscules (résultat ci-dessous).



La différence de valeur ASCII entre majuscules et minuscules est 32 (# 20h):
"A" NUM → 65 (#41h) "a" NUM → 97 (#61h)
"Z" NUM → 90 (#5Ah) "z" NUM → 122 (#7Ah

En mémoire, un caractère est réprésenté par sa valeur ASCII ***mais à l'envers*** :
'A' (#41h) → (adresses basses) **14** (adr hautes)
et après par exemple A=DAT1 A A(B):= 41
Le microprocesseur Saturn retourne les données en mémoire.

On convertira MAJUSCULES→minuscules :
☛ en vérifiant qu'un caractère a une valeur ASCII comprise entre 65 (#41h) et 97 (#61h).
☛ en ajoutant 32 (#20h) si la réponse est oui

Un programme avec arguments
Le programme précédent ne nécessitait aucun argument. Cette fois, il y en a un. On fera vérifier sa présence et son type par un minimum de Rpl système assurant la sécurité d'exécution et évitant des problèmes de perturbation de la mémoire : il faut savoir à quel niveau de la pile se situe l'argument, connaître la structure de celui-ci, s'il doit être modifié et dans ce cas recréer l'objet pour conserver l'original si nécessaire.

Structure minimale des programmes mixtes

La programmation mixte, qui ne présente aucune difficulté avec les compilateurs recommandés, sera limitée au strict minimum : le Rpl système n'est pas l'objet de ce fascicule. Voici la structure, avec les mnémoniques HP du Rpl système reconnus par les compilateurs choisis :

```
::                          début de programme Rpl
  CKnNOLASTWD  vérifie q'il y a n niveaux
  CK&DISPATCH1 vérifie le type des argu
               ments précisé par l'entier système n
  n            l'entier système
  CODE         l'objet Code
  ____
  ENDCODE
;                           fin de programme Rpl
```

n *dans* Ckn....*est à remplacer par 1,2,3 .. suivant le nombre d'arguments.*

Principe de programmation
La partie Rpl vérifie la présence d'un niveau de pile avec une chaîne de type 3, mnémonique de l'entier système 3: **str** ou **THREE** car il existe plusieurs mnémoniques pour certains points d'entrée et dont les noms sont plus ou moins évocateurs. La chaîne est recréée avec **TOTEMPOB** pour conserver l'original.
La partie Code prépare les données pour la boucle :
☛ L'appel à **GetStrLenStk** fait pointer D1 sur le 1er caractère de la chaîne et C(A) contient le nombre de caractères de la chaîne. Cette valeur est sauvée sur la pile des retours **RSTK**, nous reviendrons sur cette pile, pour l'instant il suf-

fit de savoir que les échanges sont possibles entre RSTK et C(A)

☛ La valeur ASCII de 'A' est chargée en B(B), celle de 'Z' en D(B), la différence 32 (#20h) en A(B).

☛ L'entrée dans la boucle se fait à *start* pour décrémenter le compteur et traiter le cas d'une chaîne vide par sortie sans rien faire.

La boucle fera les conversions des caractères compris dans l'intervalle 65 à 97.

La programmation d'une boucle doit se faire avec attention car elle peut être exécutée de très nombreuses fois et influera sur la rapidité d'un programme. Dans le programme ci-dessous, le maximum de constantes a été chargé dans les registres avant d'exécuter la boucle de manière que celle-ci exécute le minimum d'opérations.

Voici le programme **LOWCASE** (64 octets CRC: # 7C84h). Dans les listings, @ signifiera "**pointe**". Exemple: D1 @ le caractère.

```
::
 CK1NOLASTWD
 CK&DISPATCH1
 str
 ::
   TOTEMPOB
   CODE
        GOSBVL      =SAVPTR       * sauve les pointeurs
        GOSBVL      =GetStrLenStk * D1 @ 1er caract; C(A) nbre caract.
        RSTK=C                    * sauve nombre caract. sur pile RSTK
        LCASC       'A'           * valeur ASCII 1ère majuscule
        B=C         B             * sauve limite inférieure de l'intervalle
        LCASC       'Z'           * valeur ASCII dernière lettre majusc.
        D=C         B             * sauve limite supérieure de l'intervalle
        LA(2)       97-65         * différence majuscules minuscules
        GONC        start         * saut pour décrémenter le compteur
loop    RSTK=C                    * sauve le compteur de fin de boucle
        C=DAT1B                   * C(B):= caractère à tester
        ?C<B        B             * valeur ASCII inférieure à 65 ?
        GOYES       next          * si oui on passe au caractère suivant
        ?C>D        B             * valeur ASCII supérieure à 90 ?
        GOYES       next          * si oui on passe au caractère suivant
        C=C+A       B             * caractère dans l'intervalle: conversion
        DAT1=CB                   * charge en mémoire le caract. converti
next    D1=D1+ 2                  * D1 pointe le caractère suivant
start   C=RSTK                    * C(A):= compteur de caractères
        C=C-1       A             * décrémente le compteur de caractères
        GONC   loop               * encore 1 tour si valeur compteur ≥ 0
        GOVLNG      =GETPTRLOOP   * sinon retour au Rpl
   ENDCODE
  ;
;
```

CONVERSION DANS LES DEUX SENS AVEC 2 ARGUMENTS

Ce que l'on veut faire :
Le programme LOWCASE fait la conversion MAJUSCULES → minuscules d'une chaîne. Un deuxième argument d'option permettra de faire également la conversion dans le sens inverse.

L'option sera précisée par une chaîne : "U" pour le sens minuscules → MAJUSCULES, "L" pour le sens MAJUSCULES → minuscules.

Principe de programmation
Pour la partie Rpl système, 2 niveaux de pile seront vérifiés ainsi que le type chaîne avec l'entier système de valeurr #33h (51). L'option sera différenciée à l'aide de l'état du status bit 0, après avoir déterminé sa valeur, comme avec un flag. La conversion dans le sens minuscules → MAJUSCULES se fera avec une différence de valeur ASCII négative.
La boucle de conversion sera identique à celle du programme LOWCASE, les valeurs constantes étant préparées avant cette boucle.

Le programme **XCASE** ci-dessous
(92 octets CRC # 74DAh) nécessite :
- au niveau 2, la chaîne à convertir,
- au niveau 1, l'option.

Exemples :
"Les Clefs de l'Assembleur"
"U"
XCASE → "LES CLEFS DE L'ASSEMBLEUR

"Les CLEFS de l'ASSEMBLEUR"
"L"
XCASE → "les clefs de l'assembleur"

☛ Les caractères accentués ne sont pas convertis. A titre d'exercice, faire un programme qui les convertit aussi.

☛ La préparation des constantes permettant de tester si un caractère est dans l'intervalle des majuscules ou des minuscules, suivant l'option de conversion, ainsi que la différence ±32, avant la boucle, a permis de réduire au strict minimum les opérations effectuées par la boucle, en particulier de ne faire charger aucune constante par celle-ci et de ne plus avoir à opérer de tests pour l'option de conversion.

```
::
  CK2NOLASTWD
  CK&DISPATCH1
  FIFTYONE
  ::
    SWAP
    TOTEMPSWAP
    CODE
          GOSBVL        =PopASavptr     * A(A):= adresse de la chaîne option
*                                       * Drop sur le niveau 1, sauve pointeurs
          D1=A                          * D1 @ le prologue chaîne
          D1=D1+        10              * D1 saute prologue et longueur
          ST=0          0               * Pour option minuscules→majuscules
```

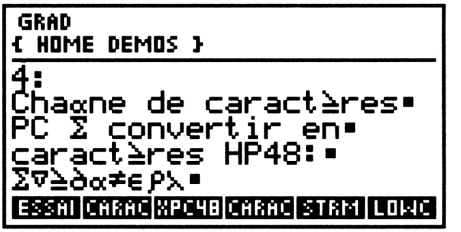
```
        LCASC       'U'          * C(B):= valeur ASCII de 'U'
        A=DAT1      B            * A(B):= valeur de l'option
        ?A=C        B            * option minuscules→majuscules ?
        GOYES       up           * si oui saut à up
        ST=1        0            * sinon arme le status bit 0
up      GOSBVL      =GETPTR      * restaure les pointeurs
        GOSBVL      =GetStrLenStk * D1 @ 1er caractère chaîne à modifier
*                                   C(A):= nombre de caractères
        RSTK=C                   * sauve ce nombre sur RSTK
        LCASC       'A'          * limite inférieure des majuscules
        B=C         B            * sauve cette limite
        LCASC       'Z'          * limite supérieure des majuscules
        D=C         B            * sauve cette limite
        LA(2)       97-65        * A(B):=différence de valeur ACII
*                                   entre minuscules et majuscules
        ?ST=1       0            * option MAJUSCULES → minuscules ?
        GOYES       start        * si oui on entre dans la boucle
        B=B+A       B            * sinon on change les limites pour les
*                                   minuscules en ajoutant 32
        ACEX        A            * C(B):=32
        D=D+C       B            * D(B):=valeur ASCII de 'z'
        ACEX        A            * restaure A(B) et C(B)
        A=-A        B            * -32 pour minuscules→majuscules
        GOC         start        * saut à start avec GOC car A=-A B
*                                   a armé le bit Carry
loop    RSTK=C                   * boucle identique à celle de
                                   LOWCASE, voir les commentaires
        C=DAT1      B            * de ce programme jusqu'à la fin
        ?C<B        B
        GOYES       next
        ?C>D        B
        GOYES       next
        C=C+A       B
        DAT1=C      B
next    D1=D1+      2
start   C=RSTK
        C=C-1       A
        GONC        loop
        GOVLNG      =GETPTRLOOP
    ENDCODE
  ;
;
```

CONVERSION UNIVERSELLE DE CARACTERES

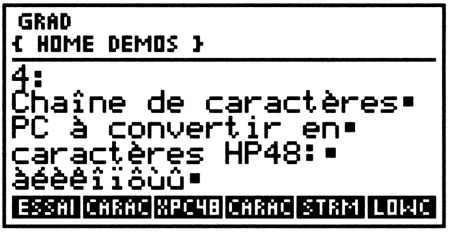
Nous avons vu comment faire appel à un peu de Rpl système pour vérifier le nombre et le type des arguments nécessaires au programme assembleur encapsulé dans la structure Rpl. Puis, connaissant la nature des objets HP48, il a été possible de modifier ceux-ci ou d'importer les données qu'ils contiennent. Les premiers programmes de conversion de caractères ont permis de se familiariser avec une programmation simple. Sur le même thème nous allons aborder des techniques un peu plus pointues. Il vous faudra auparavant lire l'encadré ci-dessous.

Ce que l'on veut faire

Convertir une partie des caractères d'une chaîne suivant une correspondance biunivoque entre les caractères de deux suites de caractères. Par exemple, la chaîne de l'écran ci-dessous a été importée d'un texte tapé sur PC. Elle comportait des caractères accentués, mais ces derniers apparaissent sous une forme différente car leur numéro ASCII ne correspond pas à celui de la table des codes des caractères HP48, même avec un décalage constant comme dans le cas des majuscules et des minuscules :



Il s'agit de convertir ces lettres pour obtenir l'affichage ci-dessous :



☛ Ne pas confondre le code d'un caractère avec l'image de ce caractère à l'écran. Les programmes d'affichage interpréteront le code

LES REGISTRES DE SAUVEGARDE

Les registres de travail A,B,C et D peuvent se révéler en nombre insuffisant pour contenir les valeurs nécessaires aux opérations. Si la pile des retours RSTK a été utilisée dans les programmes précédents, cela n'est pas toujours possible. Le registre P peut faire des échanges avec C(n) mais pour un seul quartet (voir *le registre P*). Cinq registres de 64 bits R0, R1, R2, R3 et R4 sont prévus pour faire des échanges avec les registres de travail A et C et pour les opérations suivantes de copie ou d'échange:

r = A ou C
ss = R0, R1, R2, R3 ou R4
fs = A,P, WP, XS, X, S, M, B ou W
lorsque **fs** (**.F**) n'est pas précisé la copie (**=**) ou l'échange (**EX**) se fait sur la totalité des 64 bits des registres.

r=ss		exemple: C=R0
ss=r		exemple: R3=A
rssEX		exemple: AR2EX
r=ss.F	fs	exemple: C=R1.F X
ss=r.F	fs	exemple: R0=C.F A
rssEX.F	fs	exemple: AR4EX.F S

Possibilités avec le registre ST (status bits)
pour les 12 bits de poids faible et C(X):

CSTEX	C=ST	ST=C

comme un numéro pour aller chercher l'image dans un jeu de caractères : la fonte.

La démarche :
On part de la liste **L** des codes ASCII de 0 à 255 où comme on l'a vu ci-dessus, chaque caractère correspond à un numéro: A → 65, a → 97, z → 122 etc...
Prenons l'exemple de la conversion de majuscules en minuscules : si dans **L** on remplace le numéro 65 de **A** par 97, lorsque le programme analysera le texte à convertir il consultera **L** et à la position 65 il trouvera 97 code de **a** et fera la conversion.

Principe de programmation:
Deux chaînes de **n** caractères sont fournies:
- **str1** chaîne des caractères à remplacer

$$c_1c_2c_3c_4c_5.....c_n$$

- **str2** chaîne des caractères de remplacement

$$C_1C_2C_3C_4C_5 \quad C_n$$

Ces deux chaînes doivent avoir leurs caractères en correspondance biunivoque: $c_1 \rightarrow C_1$.
Une chaîne vecteur **strv** des codes 0 à 255 sera créée et le code des caractères de **str1** y sera remplacé par celui des caractères de **str2** correspondants.
Puis le programme prendra successivement chaque caractère de la chaîne **str0** à convertir, recherchera à la position **k** le code (changé ou non) du caractère de code **k**, replacera le caractère dans la chaîne. Un code occupant 2 quartets la position k sera au quartet 2*k.

Le programme sera composé d'une partie en Rpl système pour vérifier 3 niveaux de chaînes et recréer la chaîne à convertir puis, après la partie Code, pour se débarrasser de str1 et str2.

Pour la partie assembleur on aura:
- création de la chaîne vecteur **strv** en réservant un espace mémoire de 512 quartets (256 codes de caractères) par la routine HP **MAKE$N** qui nécessite en C(A) le nombre de caractères de la chaîne, les pointeurs sauvés avant seront actualisés, en sortie D0 pointe sur le 1er caractère et R0(A) contient l'adresse de la chaîne créée.
- vérification que str0 n'est pas une chaîne vide, sinon émission du message *bad argument value* et arrêt du programme via la routine HP **GPErrJmpC** qui néccessite le numéro de message dans C(A), restaure les pointeurs puis saute à la routine créant le message puis le retour au Rpl avec arrêt du programme.
- vérification que str1 et str2 sont de même longueur non nulle, sinon déclenche l'ereur ci-dessus.
- modification de **strv** en fonction de **str1** et **str2**.
- boucle de conversion de **str0** et retour au Rpl

Il sera fait appel 3 fois à GetStrLenStk (vous l'avez déjà utilisée).
☛ la conversion pouvant opérer sur des milliers de caractères, la boucle de conversion sera active des milliers de fois, aussi celle-ci devra être la plus rapide possible. Ici, grâce à l'algorithme utilisé cette boucle ne contient pas de tests et est très courte (10 instructions).
☛ ici on utilisera encore la pile des retours RSTK pour sauvegarder des valeurs car nous n'aurons pas assez des 4 registres de travail ou lorsque le programme fera appel à des routines HP qui modifient ces registres. Ce type de sauvegarde est avantageux car peu gourmand en octets et rapide, en contrepartie il faut respecter l'odre de restauration des valeurs dans les registres de travail, suivant le principe du Rpn *dernier entré premier sorti*, et ne compter, de façon courante, que sur 6 niveaux d'empilage disponibles sans appels de routines.
☛ la boucle d'initialisation de strv aura pour compteur C(B) allant de 0 à 255 qui sera aussi la valeur des codes. Ici le compteur est incrémenté pour servir aussi de générateur de code des caractères, armant le Carry lors du passage de 255 à 0.

Programme **MAP**. (148.5 octets CRC: # 33FAh)

Syntaxe: niveau 3: **str0**, chaîne à traiter
niveau 2: **str1**, chaîne des caractères à remplacer
niveau 1: **str2**, chaîne des caractères de remplacement correspondant à str1

```
::
 CK3NOLASTWD CK&DISPATCH1 # 333
 ::
  ROT TOTEMPOB
  CODE
                                    * nous avons ici:
                                    *       niveau 3: str1
                                    *       niveau 2: str2
                                    *       niveau 1: str0 recréée
        GOSBVL      =SAVPTR       * sauve les pointeurs
        LC(5)       512           * longueur en quartets de chaîne à créer
        GOSBVL      =MAKE$N       * création de strv  D0 @ les données
        CD0EX
        R1=C                      * sauve l'adresse des données de strv
        D0=C                      * D0 @ les données de strv
        C=0         B             * pour initialiser strv

init    DAT0=C      B             * boucle d'initialisation de strv avec
*                                   les codes 0 à 255 (# 00h à # FFh)
        D0=D0+      2             * D0 @ le code suivant
        C=C+1       B             * C(B):= valeur du code k
        GONC        init          * saut à init si non passage à 00

        GOSBVL      =GETPTR       * restitue les pointeurs modifiés
        GOSBVL      =GetStrLenStk * C(A) := nombre de caractères de str0
        C=C-1       A
        GOC         bad0          * erreur si chaîne nulle
        RSTK=C                    * sauve (longueur - 1) de str0
        CD1EX                     * D1 pointait les données de str0
        RSTK=C                    * sauve l'adresse des données de str0
        GOSBVL      =GETPTR       * restitue les pointeurs
        D1=D1+      5             * D1 @ niveau 2 : str2
        GOSBVL      =GetStrLenStk * mêmes principes que pour str0
        RSTK=C                    * sauve le nombre de caractères de str2
        CD1EX                     * D1 pointait les données de str2
        RSTK=C                    * sauve l'adresse des données de str2
        GOSBVL      =GETPTR       * restitue les pointeurs
        D1=D1+      10            * D1 @ niveau 3 : str1
        GOSBVL      =GetStrLenStk * mêmes principes que pour str0, str2
        B=C         A             * sauve le nombre de caractères de str1
        C=RSTK                    * C(A):=adresse des données de str2
        D0=C                      * D0 @ les données de str2
```

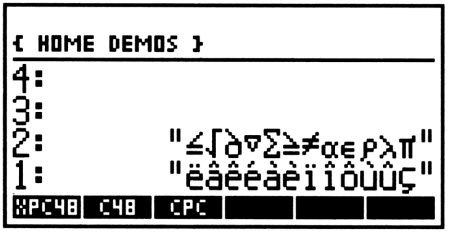
```
        C=RSTK                      * C(A):=nombre de caractères de str2
        A=C         A               * sauve nombre de caractères de str2
        C=R1                        * C(A):=adresse des données de strv
        D=C         A               * sauve cette adresse
        ?A=B        A               * longueur str2 = longueur str1 ?
        GOYES       ok              * si oui on continue
bad2    C=RSTK                      * si non on libère les 2 niveaux de pile
        C=RSTK                      * occupés
bad0    LCHEX       00203           * pour bad argument value
        GOVLNG      =GPErrjmpC      * sortie avec ce message
ok      B=B-1       A               * (nombre de caractères de str1) - 1
        GOC         bad2            * erreur si str1 et str2 de longueur nulle
tabl    C=0         A               * boucle de vectorisation
        C=DAT1      B               * valeur du code k du caractère j de str1
        C=C+C       X               * transforme en offset: 2*k
        C=C+D       A               * adresse des données de strv + 2*k =
*                                     adresse position du code du caract. j
        CD1EX                       * D1 @ position j dans table
        A=DAT0      B               * A(B):= code du caractère j de str2
        DAT1=A      B               * charge en position j dans strv
        CD1EX                       * restaure D1
        D1=D1+      2               * D1 @ caract. j + 1 de str1
        D0=D0+      2               * D0 @ caract. j + 1 de str2
decr    B=B-1       A               * compteur de caract.de str1 et str2
        GONC        tabl            * s'il reste encore des caractères de str1

        C=RSTK                      * sinon C(A):=adresse des données str0
        D1=C                        * D1 @ les données de str0
        C=RSTK                      * C(A):=nombre de caractères de str0-1
        A=C         A               * sauve ce nombre pour le compteur

loop    C=0         A               * boucle de conversion
        C=DAT1      B               * C(B):= code k du caractère j de str0
        C=C+C       X               * transforme en offset (2*k)
        C=C+D       A               * 2k + tadresse données de strv
        D0=C                        * D0 @ position du code k dans table
        C=DAT0      B               * C(B):=code k (changé ou non) du
*                                     caractère j à replacer dans str0
        DAT1=C      B               * charge le code dans str0
        D1=D1+      2               * D1 @ le caractère j + 1 de str0
        A=A-1       A               * actualise le compteur de caractères
        GONC        loop            * si encore des caractères saut à loop

        GOVLNG      =GETPTRLOOP     * restaure pointeurs et retour au RPL
ENDCODE
  UNROT2DROP                        * drop sur str1 et str2
 ;
;
```

Exemples d'applications de MAP

1. Conversion de caractères spéciaux

Comment utiliser MAP pour la conversion des caractères êâêéàèîîôùûç d'un texte importé d'un PC dans une HP48 en caractères HP48 et passer ainsi d'un affichage partiellement illisible à l'affichage de l'écran suivant ?

on crée une chaîne correspondant aux 12 caractères PC comme celle du niveau 2 ci-dessous :



Cela se fera avec le programme:

« { 137 131 136 130 133 138 139 140 147 151 150 135 } "" 1 12 FOR I OVER I GET CHR + NEXT SWAP DROP »

Cette chaîne est stockée dans la variable **CPC**.

On crée ensuite la chaîne des caractères HP48 (niveau 1 écran ci-dessus) soit avec le menu **CHARS** sur HP48G/GX ou sur HP48S/SX avec le programme précédent en remplaçant la liste par :

{ 235 226 234 224 232 239 238 244 249 251 231 }

Cette chaîne est stockée dans la variable **C48**.

Les conversions dans les 2 sens peuvent se faire avec le programme **XPC48** :

« C48 CPC "→PC: 0 →HP48: 1" "" INPUT OBJ→ IF THEN SWAP END MAP »

Une chaîne à convertir étant au niveau 1, on répond à l'invite du programme XPC48 par 0 pour une conversion HP48→PC ou par 1 pour la conversion inverse.

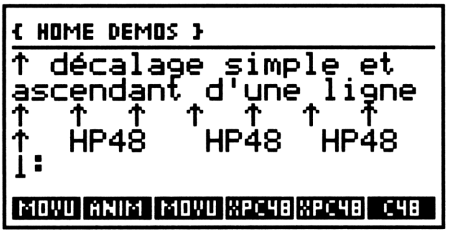
2. Autre application : le cryptage d'une chaîne

Le programme **MAP** permet également de crypter un texte puisqu'il suffit de créer une chaîne avec des caractères quelconques choisis dans la liste des codes 0 à 255 et devant remplacer une partie ou la totalité des caractères du texte à crypter.

PROGRAMMES GRAPHIQUES DE BASE

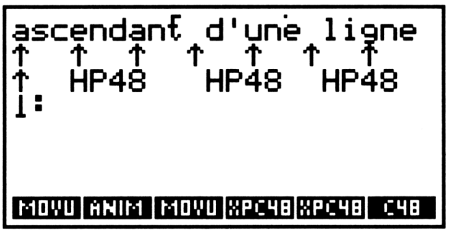
Le programme d'inversion vidéo du début du chapitre précédent est très simple car les quartets sont manipulés par blocs. Allons plus loin avec quelques petits programmes de graphisme inspirés de ceux de Christophe Nguyen dans Haute Performance mais traités avec d'autres techniques et illustrés par quelques applications.

DÉPLACEMENT ASCENDANT DE L'IMAGE DE LA PILE



Ce que l'on veut faire

Faire monter l'image située au-dessus de la barre des menus d'une ligne de pixels. Ce problème traité en Rpl serait d'exécution trop lente pour faire une animation graphique.



Sur cet écran, le progamme a été appelé plusieurs fois pour réaliser la remontée.

La démarche

Concernant l'affichage de la pile et le grob stack revoir les informations données pour l'inversion vidéo. Ici, seules, les adresses du contenu des lignes vont changer par décalage, vers les adresses basses, de 34 quartets.

Principe de programmation

Après appel à la routine **D0→Row1**, D0 pointe sur le premier quartet du contenu de la première ligne et A(A) contient l'adresse de ce quartet. Cette adresse + 34 donne l'adresse de la deuxième ligne. Si D0 pointe sur la 1ère ligne et D1 sur la deuxième ligne, décaler les 55 lignes (56-1), une par une, revient à décaler les 55*34=1870 quartets de ces 55 lignes de 34 quartets. 1870 c'est aussi 116*16+14 et ce décalage pourrait se faire avec la séquence:

```
        LC(2)   116-1
boucle  A=DAT1          16
        DAT0=A          16
        D1=D1+          16
        D0=D0+          16
        C=C-1           B
        GONC    boucle
        A=DAT1          14
        DAT0=A          14
```

La routine HP **MOVEDOWN** fera ce décalage de zone mémoire.

Au départ :

- D1 pointe le début de la zone à atteindre,
- D0 pointe le début de la zone à transférer,
- C(A) contient le nombre de quartets à décaler.

En sortie :

D1 et D0 pointent respectivement après chaque zone.

Le programme **MOVEUP** (29 octets CRC # 98D6h)

```
CODE
        GOSBVL      =SAVPTR          * Sauvegarde des registres
        GOSBVL      =D0→Row1         * D0 @ 1er quartet du grob stack A(A):=D0
        D1=A                         * D1 @ ce quartet ou début de 1ère ligne
        LC(5)       34               * C(A):=longueur en quartets d'une ligne
        A=A+C       A                * A(A):=adresse début de 2ième ligne
        D0=A                         * D0 @ début de 2ième ligne
        LC(3)       55*34            * nombre de quartets à déplacer pour 55 lignes
        GOSBVL      =MOVEDOWN        * déplacement ascendant d'1 ligne pour 55 lignes
        GOVLNG      =GETPTRLOOP      * restaure les pointeurs et retour au Rpl
ENDCODE
```

☛ La routine MOVEDOWN n'est utile que pour des déplacements de zones de plus de 16 quartets et nécessite une adresse à atteindre inférieure à celle de départ.

☛ Pour les échanges entre registres et mémoire des instructions peuvent avoir le même résultat mais avec des rapidités différentes:
- A=DAT1 A ou A=DAT1 B doivent être utilisées et non A=DAT1 5 ou A=DAT1 2, ces dernières étant moins rapides et codées sur 4 quartets au lieu de 3.
- de même pour DAT1=A A ou DAT1=A B
- mais préférer A=DAT1 16 à A=DAT1 W, cette dernière étant légèrement moins rapide.

Application

Le programme qui suit affiche 2 lignes de texte puis fait glisser l'image vers le haut jusqu'à disparition complète. Un appui sur une touche, avant disparition de l'image, stoppera le défilement comme représenté sur les écrans ci-dessus avec l'affichage de l'écran gelé. Pour obtenir un défilement de l'image du texte seul, débuter le programme avec CLLCD.

```
« "↑ décalage simple et
  ascendant d'une ligne" 3 DISP
  "↑ ↑ ↑ ↑ ↑ ↑ ↑" 5 DISP
  "↑ HP48  HP48  HP48" 6 DISP
  DO MOVEUP .05 WAIT
  UNTIL KEY END DROP 7 FREEZE »
```

Cette application n'est là que pour illustrer les effets de **MOVEUP**.

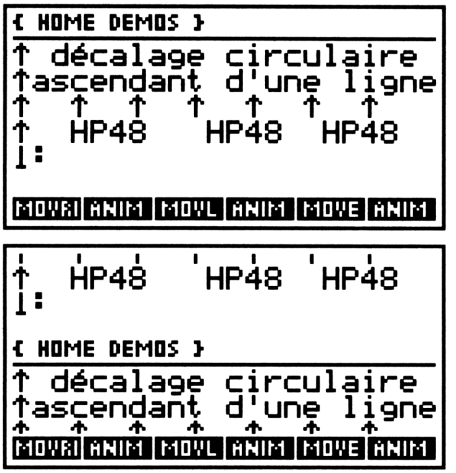
DÉPLACEMENT ASCENDANT CIRCULAIRE

Ce que l'on veut faire

Comme précédemment, un déplacement ascendant mais sans perdre la ligne 1 qu'on fera passer en ligne 56.

Principe de progammation

Avant de faire le déplacement avec la routine MOVEDOWN, les 34 quartets de la 1ère ligne seront sauvegardés dans R0, R1 et sur RSTK pour être chargés en mémoire à l'adresse correspondant à la 56ème ligne.



Application

Ce qui a été fait pour MOVEUP peut être répété Le défilement de l'image fera passer la ligne d'états (avec { HOME DEMOS }) en bas avec le programme:

« "↑ décalage circulaire
ascendant d'une ligne" 3 DISP
"↑ ↑ ↑ ↑ ↑ ↑ ↑" 5 DISP
"↑ HP48 HP48 HP48" 6 DISP
DO **MOVUPCIR** .05 WAIT
UNTIL KEY END DROP 7 FREEZE »

Le programme **MOVUPCIR** (50,5 octets CRC # E9CCh)

```
CODE
        GOSBVL      =SAVPTR          * comme MOVEUP
        GOSBVL      =D0→Row1         *
        D1=A                         * jusqu'ici
        A=DAT0      16               * lecture de la première ligne
        R0=A                         * sauvegarde des 16 premiers quartets
        D0=D0+      16               * D0 @ sur le 17ième quartet
        A=DAT0      16               * lecture des 16 quartets suivants
        R1=A                         * sauvegarde dans R1
        D0=D0+      16               * D0 @ sur le 33ième quartet
        C=DAT0      B                * lecture des 2 derniers quartets de la ligne 1
        RSTK=C                       * sauvegarde sur la pile RSTK
        D0=D0+      2                * D0 @ sur la 2ième ligne
        LC(5)       55*34            * comme pour MOVEUP
        GOSBVL      =MOVEDOWN        * après: D1 @ après la 55ième ligne
        A=R0                         * récupération des quartets de la 1ère ligne
        DAT1=A      16               * écriture en ligne 56 des 16 premiers quartets
        D1=D1+      16               * D1 @ le quartet 17 de la 56ième ligne
        A=R1                         * récupère les 16 quartets suivants
        DAT1=A      16               * écriture de ces quartets
        D1=D1+      16               * D1 @ sur le 33ième quartet de 56ème ligne
        C=RSTK                       * récupère les 2 derniers quartets
        DAT1=C      B                * écriture
        GOVLNG      =GETPTRLOOP
ENDCODE
```

INSTRUCTIONS DE DÉCALAGE

Les notations

r = A, B, C ou D **fs** = A, P, WP, Xs, X, M, B ou W

lorsque fs (ou **.F**) n'est pas précisé les décalages se font sur la totalité des 64 bits du registre.

Les instructions

rSRB décalage d'un bit à droite *Exemple:* **CSRB**

rSRB.F même chose sur le champ *Exemple:* **ASRB.F A**

dans ces décalages le bit de poids fort du registre ou du champ prend la valeur zéro

rSLC décalage circulaire à gauche d'un quartet, le quartet de poids fort devient celui de poids faible. *Exemple:* **DSLC**

rSRC décalage de même type à droite, le quartet de poids faible devient celui de poids fort *Exemple:* **BSRC**

rSL fs décalage d'un quartet à gauche

rSR fs décalage d'un quartet à droite

Remarques

☛ Du point de vue arithmétique, les décalages ci-dessus modifient les valeurs des registres pour les champs concernés. Exemples avec l'un des registres et pour le mode **HEX**

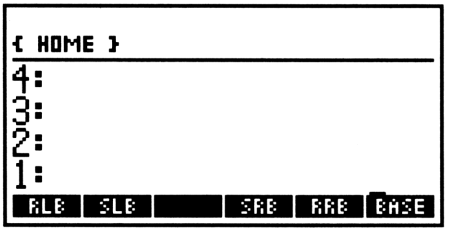
☛ **ASRB** ou **CSRB.F X** produisent une division entière par 2 de la valeur du registre ou du champ.

☛ Ces opérations arment le Stiky Bit (**SB**) si le bit de poids faible disparaissant était armé.

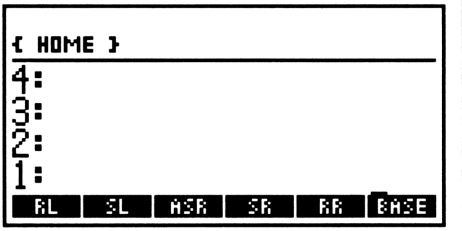
☛ **BSL M** produit une multiplication par 16 de la valeur du champ

☛ **CSR A** produit une division entière par 16 de la valeur du champ.

Avec les commandes des menus **BYTE** et **BIT** on peut émuler une partie des instructions ci-dessus avec leurs résultats arithmétiques. Il suffit de fixer la longueur du mot en bits avec **STWS** ainsi que le mode **HEX** pour les décalages en quartets, le mode **BIN** pour les décalages en bits. Voir le manuel utilisateur pour ces commandes avec les exemples (chapitre 14 ou 15 suivant le type de HP48). Lors de la création d'un programme assembleur, les commandes du menu BASES évitent certains calculs fastidieux.



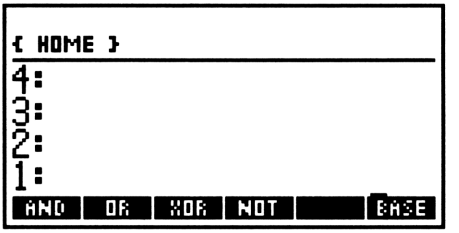
menu MATH BASE BYTE



menu MATH BASE BIT

OPÉRATEURS LOGIQUES

Pour en finir avec les instructions en relation avec le menu* BASES *faisons le point sur les opérations dites logiques correspondant à* AND *et* OR *du menu* LOGIC *ci-dessous.



Les notations
rs = (A,B), (A,C), (B,A), (B,C), (C,A), (C,B), (C,D), ou (D,C)
fs = A, P, WP, XS, X, S, M, B ou W

Les instructions

r=r&s	**fs**	'ET logique'	*Exemple:* **A=A&C P**
r=r!s	**fs**	'OU logique'	*Exemple:* **C=C!B A**

Remarques
☞ L'opération ***OU*** permet d'associer les bits armés de 2 valeurs de champ de registres, par exemple pour faire une surimpression d'images
☞ L'opération ET permet de désarmer les bits armés non à la fois dans 2 registres. Exemple pour trouver le reste de la division par 4, on fera en mode HEX (la valeur donnée n étant dans le registre A:
LCHEX 3 **C=A&C P** (P= 0)
C[0] aura pour valeur : **n modulo 4**
☞ L'instruction XOR n'existant pas on peut réaliser **A XOR C** avec:

B=A	**fs**	**sauve A*
B=B&C	**fs**	**A AND C*
A=A!C	**fs**	**A OR C*
A=A-B	**fs**	**(A OR C)–(A AND C)*

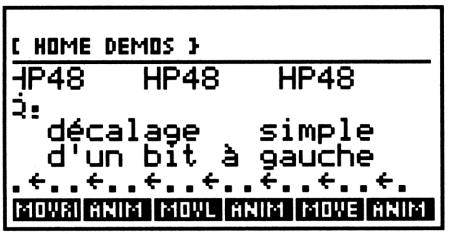
Comme pour les opérations de décalages on pourra s'aider de la HP48 avec les commandes du menu LOGIC et les exemples du manuel.

DÉPLACEMENT LATÉRAL GAUCHE

Les déplacements de lignes de l'écran ont été réalisés par manipulations de quartets entiers. Le décalage d'un pixel d'une ligne horizontalement est plus délicat : il demande de passer par la manipulation de bits. Vous pouvez, là encore, retrouver les deux thèmes qui suivent, mais traités différemment, dans la rubrique assembleur de [HtePerf] No 9 de C. N .

Ce que l'on veut faire

Décaler à gauche simultanément (en apparence) les 56 lignes du grob stack de la valeur d'un point. Par exemple, par appel continu au



programme, on obtiendra les images intermédiaires ci-dessus et ci-dessous.

La démarche

Soit une suite de 16 bits, en mémoire puis chargés dans un registre :

1001110101100101	mémoire
1010011010111001	registre

Saturn retournant les données lors des échanges mémoire←→registre, le bit de poids fort en mémoire devient le bit de poids faible dans le registre. *À un décalage à gauche en mémoire correspond un décalage à droite dans le registre.*

Principe de programmation

Une ligne de l'écran est contenue dans 34 quartets représentant 136 bits que l'on peut diviser en 64 + 64 + 8 bits pour les transferts dans les registres avec 16 + 16 + 2 quartets. Avec la numérotation par groupes,

00 01 ... 62 63 00 01 ... 62 63 00 01 ... 06 07

un décalage d'un bit à gauche donne:

01 ... 62 63 00 01 ... 62 63 00 01 ... 06 07 ??

Le bit de poids fort du premier groupe est perdu puis ceux des 2 autres groupes glissent dans le groupe précédent à gauche et le bit de poids faible du troisième groupe est remplacé par un bit non armé. Avec le retournement dans les registres on a :

* Avant décalage :

63 62 ... 01 00 63 62 ...01 00 07 06 ... 00 01

* Après décalage

00 63 62 ... 01 00 63 62 ... 01 ?? 07 06 ... 01

Le bit de poids faible d'un groupe remplace le bit de poids fort du groupe précédent et le bit de poids fort du dernier groupe est désarmé. Les décalages d'un bit à droite se font avec **rSRB**.

☛ La préparation des valeurs nécessaires à la boucle effectuant les décalages comprend non seulement l'adresse de la première ligne, le compteur de lignes, mais aussi la mise en réserve dans B(S) du bit de poids fort à ajouter suivant le principe ci-dessus. Avec 8 (#1000b) dans B(S) cela simplifiera la progammation de la boucle. Les registres A et C étant réservés aux échanges avec la mémoire, les registres B et D contiennent les autres valeurs (constante 8 et compteur).

La boucle

Le chargement dans les registres A et C de 2 groupes à la fois permettra de modifier le bit de poids fort du 1er groupe chargé et éviter de revenir en arrière (méthode C.N.), d'où une boucle plus courte et plus rapide. L'état du bit de poids faible sera testé avec **?rBIT=0 0** et non **?SB=0** après **rSRB**.

Programme **MOVELEFT** (63,5 octets CRC: # 42BAh)

```
CODE
        GOSBVL      =SAVPTR      * Sauvegarde des registres
        P=          15           * pour charger 8 (#1000b) dans C(S)
        LCHEX       8            * le bit de poids fort du registre
        P=          0            * restaure P=0 pour la suite et la sortie
        B=C         S            * sauve le bit de poids fort dans B(S)
        LC(2)       56-1         * compteur de lignes - 1
        D=C         A            * sauvegarde du compteur
        GOSBVL      =D0→Row1     * D0 @ la 1ère ligne  A(A):=cette adresse
loop    D0=A                     * D0 @ 1er groupe de 16 quartets ligne j
        D1=A
        D1=D1+      16           * D1 @ 2ième groupe de 16 quartets ligne j
        A=DAT0      16           * A(W):=16 premiers quartets 1er groupe
        ASRB                     * décalage d'un bit à droite du 1er groupe
        C=DAT1      16           * C(W):=16 quartets 2ème groupe ligne j
        ?CBIT=0     0            * bit de poids faible 2ème groupe non armé ?
        GOYES       nobit1       * si oui on ne fait rien
        A=A+B       S            * sinon le bit de fort du 1er groupe est armé
nobit1  CSRB                     * décalage d'un bit à droite du 2ème groupe
        DAT0=A      16           * écriture des 16 quartets du 1er groupe
        D0=D0+      16           * D0 @ le 2ème groupe de 16 quartets
        D1=D1+      16           * D1 @ le 3ième groupe de 2 quartets
        A=DAT1      B            * A(B):=les 2 quartets du 3ième groupe
        ?ABIT=0     0            * bit de poids faible du 3ème groupe non armé ?
        GOYES       nobit2       * si oui, alors on ne fait rien
        C=C+B       S            * sinon arme le bit de poids fort du 2ème groupe
nobit2  ASRB.F      B            * décalage d'un bit à droite du 3ème groupe
        DAT1=A      B            * écriture des 2 quartets du 3ème groupe
        DAT0=C      16           * écriture des 16 quartets du 2ième groupe
        D1=D1+      2            * D1 @ la ligne j+1
        AD1EX                    * A(A):=adresse ligne j+1 pour début de boucle
        D=D-1       B            * actualise le compteur de lignes
        GONC        loop         * si D(B):≥0 alors encore un tour de boucle
        GOVLNG      =GETPTRLOOP  * retour au Rpl
ENDCODE
```

Application

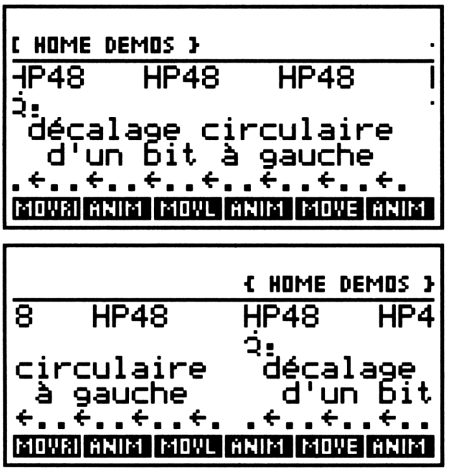
Le programme ci-contre fera défiler l'affichage représenté pour deux situations intermédiaires sur les écrans précédents. Une pression de touche avant la disparition complète de l'affichage gèlera celui-ci.

```
« "HP48  HP48  HP48" 3 DISP
" décalage  simple
  d'un bit à gauche" 3 DISP
← ← ← ← ← ← ← " 5 DISP
DO MOVELEFT .05 WAIT
  UNTIL KEY END DROP 7 FREEZE »
```

DÉPLACEMENT LATÉRAL CIRCULAIRE À GAUCHE

Encore une étape dans l'animation graphique assurant les bases de programmation pour une animation à venir, plus importante et plus divertissante.

Ce que l'on veut faire
Obtenir le même déplacement qu'avec le programme MOVELEFT mais en faisant réapparaître à droite l'image disparaissant à gauche.



La démarche
Le premier point de chaque ligne disparaissant à gauche sera replacé en dernière position à droite c'est à dire en 131ème position pour la partie visible.

Principe de programmation
Le programme MOVELEFT est complété ici par le repositionnement du bit de début de ligne.
Toujours avec la division du contenu d'une ligne en 3 groupes, le point 131 est contenu dans le dernier groupe de 2 quartets °°$\underline{°}$°°°°° ce qui donne par retournement dans le registre °°°°°$\underline{°}$°° le bit 2.

La boucle
L'état du bit faible (dans le registre) du premier groupe sera repéré par le status bit 0. En fin de boucle le bit 2 du 3ième groupe sera armé suivant l'état du status bit 0.
Pour le reste la programmation est identique à celle de MOVELEFT.

Application
Le programme ci-dessous fera défiler l'affichage représenté pour deux situations intermédiaires sur les écrans ci-contre. Une pression de touche stoppera l'affichage qui sera gelé.

```
«"HP48  HP48  HP48" 3 DISP
" décalage circulaire
  d'un bit à gauche" 3 DISP
← ← ← ← ← ← ← " 5 DISP
DO MOVLEFTC .05 WAIT
  UNTIL KEY END DROP 7 FREEZE »
```

Le programme **MOVLEFTC** (75 octets CRC: # 1061h)

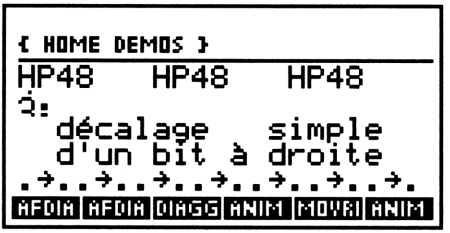
```
CODE
        GOSBVL      =SAVPTR       * même programmation
        P=          15            * que celle de MOVELEFT
        LCHEX       8             * jusqu'à la boucle
        P=          0             *
        B=C         S             *
        LC(2)       56-1          *
        D=C         A             *
        GOSBVL      =D0→Row1      *
loop    D0=A                      * début de boucle identique
        D1=A                      * à celui de MOVELEFT
        D1=D1+      16            *
        A=DAT0      16            *
        ST=0        0             * pour repérer l'état du bit faible du 1er groupe
        ?ABIT=0     0             * bit de poids faible du 1er groupe non armé ?
        GOYES       nodeb         * si oui on laisse ST=0 0
        ST=1        0             * sinon on arme ST(0)
nodeb   ASRB                      * on continue comme pour MOVELEFT
        C=DAT1      16            *
        ?CBIT=0     0             *
        GOYES       nobit1        *
        A=A+B       S             *
nobit1  CSRB                      *
        DAT0=A      16            *
        D0=D0+      16            *
        D1=D1+      16            *
        A=DAT1      B             *
        ?ABIT=0     0             *
        GOYES       nobit2        *
        C=C+B       S             *
nobit2  ASRB.F      B             *
        ?ST=0       0             * le bit faible du 1er groupe était non armé ?
        GOYES       noaj          * si oui on ne fait rien
        ABIT=1      2             * sinon on arme le bit 2 du dernier groupe
noaj    DAT1=A      B             * suite et fin identiques à MOVELEFT
        DAT0=C      16            *
        D1=D1+      2             *
        AD1EX                     *
        D=D-1       B             *
        GONC        loop          *
        GOVLNG      =GETPTRLOOP   *
ENDCODE
```

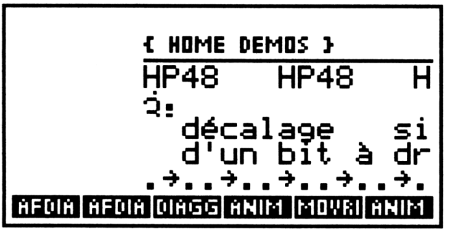
DÉPLACEMENTS LATÉRAUX À DROITE

Toujours pour assurer les bases d'une future animation graphique, voici le troisième groupe de déplacements d'un bloc de lignes d'écran.

1. DÉCALAGE SIMPLE

Ce que l'on veut faire
Obtenir à l'aide d'un programme MOVRIGHT le déplacement inverse de celui produit par le programme MOVELEFT comme sur les écrans ci-dessous.





La démarche
Le décalage d'un bit vers la droite du contenu d'une ligne se traduira dans les registres, avec la division en 3 groupes des bits d'une ligne, par un décalage à gauche et il faudra inverser ce qui a été fait avec MOVELEFT.

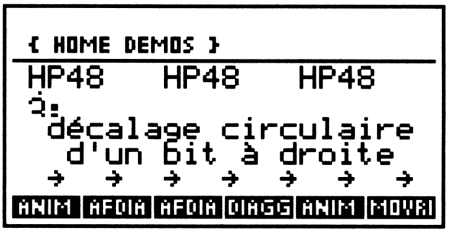
Principe de programmation
Le décalage à gauche d'un bit dans les registres s'obtiendra avec l'instruction **r=r+r fs**. Le débordement à gauche d'un bit armé sera détecté avec le bit Carry donc de façon simple.

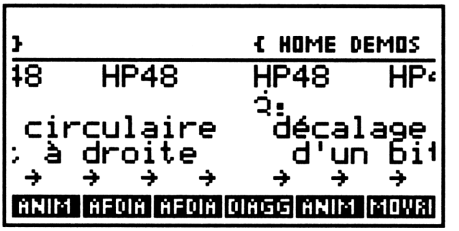
Application
Le programme ci-dessous fera défiler l'affichage représenté pour deux situations intermédiaires sur les écrans ci-contre. Une pression de touche avant la disparition complète de l'affichage gèlera celui-ci.

```
«"HP48  HP48  HP48" 3 DISP
" décalage  simple
 d'un bit à droite" 3 DISP
→ → → → → → → " 5 DISP
DO MOVRIGHT .05 WAIT
 UNTIL KEY END DROP 7 FREEZE »
```

2. DÉCALAGE CIRCULAIRE

Ce que l'on veut faire
Obtenir à l'aide d'un programme MOVRIGHTC le même déplacement qu'avec le programme MOVRIGHT mais en faisant réapparaître à gauche l'image disparaissant à droite.





La démarche
Le dernier point visible (131ème position) de chaque ligne disparaissant à droite sera replacé en première position à gauche.

Le programme **MOVRIGHT** (58 octets CRC: # B628h) de décalage simple :

```
CODE
        GOSBVL      =SAVPTR      * début identique au programme MOVELEFT
        LC(2)       56-1         * sauf la non nécessité de charger 8 dans B(S)
        D=C     A                *
        GOSBVL      =D0→Row1     *
loop    D0=A                     * D0 @ le 1er groupe de la ligne j
        D1=A                     *
        D1=D1+      16           * D1 @ le 2ème groupe de la ligne j
        A=DAT0      16           * A(W):=les 16 quartets du 1er groupe
        B=A         W            * sauve ces quartets
        C=DAT1      16           * C(W):=les 16 quartets du 2ième groupe
        D1=D1+      16           * D1 @ le 3ème groupe de la ligne j
        A=DAT1      B            * A(B):=les 2 quartets du 3ème groupe
        A=A+A       B            * décalage d'un bit à gauche du 3ème groupe
        C=C+C       W            * décalage d'un bit à gauche du 2ème groupe
        GONC        nodep        * si le bit de poids fort du 2e groupe n'est pas
*                                  armé on ne fait rien, sinon on arme le nouveau
        A=A+1       B            * bit de poids faible du 3ème groupe
nodep   DAT1=A      B            * charge en mémoire le 3ème groupe
        D1=D1-      16           * D1 @ le 2ème groupe
        B=B+B       W            * décalage d'un bit à gauche du 1er groupe
        GONC        nodep2       * si le bit de poids fort du 1er groupe n'était pas
*                                  armé on ne fait rien, sinon on arme le nouveau
        C=C+1       W            * bit de poids faible du 2ème groupe
nodep2  DAT1=C      16           * charge en mémoire le 2ème groupe
        A=B         W            * récupère le 1er groupe
        DAT0=A      16           * charge en mémoire le 1er groupe
        D1=D1+      16           *
        D1=D1+      2            * D1 @ la ligne j+1
        AD1EX                    * A(A):=adresse ligne j+1 pour début de boucle
        D=D-1       B            * actualise le compteur de lignes
        GONC        loop         * si D(B):≥0 alors encore un tour de boucle
fin     GOVLNG      =GETPTRLOOP  * retour au Rpl
ENDCODE
```

Principe de programmation

Le programme MOVRIGHTC, qui suit le principe du précédent, est complété par le repositionnement du bit de fin de ligne. Toujours avec la division du contenu d'une ligne en 3 groupes, le point 131 est contenu dans le dernier groupe de 2 quartets °°°°°°°° ce qui donne par retournement dans le registre °°°°°°°° le bit 2.

La boucle

L'état du bit 2 du troisième groupe sera testé et repéré par le status bit 0.. En fin de boucle le bit faible du premier groupe sera armé suivant l'état du status bit 0.

Pour le reste la programmation est identique à celle de MOVRIGHT. Le programme MOVRIGHTC est en page suivante.

Le programme **MOVRIGHTC** (68 octets CRC: # 38A4h)

```
CODE
        GOSBVL      =SAVPTR       * programmation identique à celle de
        LC(2)       55            * MOVRIGHT sauf pour les particularités
        D=C         A             * signalées dans la boucle
        GOSBVL      =D0→Row1      *
loop    D0=A                      *
        D1=A                      *
        D1=D1+      16            *
        A=DAT0      16            *
        B=A         W             * 1er groupe
        C=DAT1      16            * 2 ème groupe
        D1=D1+      16            *
        A=DAT1      B             * 3ème groupe
        ST=0        0             * pour repérer l'état du point 131
        ?ABIT=0     2             * bit 2 du 3ème groupe non armé ?
        GOYES       noreport      * si oui on ne fait rien
        ST=1        0             * sinon on repère cet état avec St=1 0
noreport A=A+A      B             *
        C=C+C       W             *
        GONC        nodep         *
        A=A+1       B             *
nodep   DAT1=A      B             *
        D1=D1-      16            *
        B=B+B       W             *
        GONC        nodep2        *
        C=C+1       W             *
nodep2  DAT1=C      16            *
        ?ST=0       0             * le point 131 était éteint ?
        GOYES       inchange      * si oui on ne fait rien
        B=B+1       A             * sinon on arme le nouveau bit de poids faible
*                                   du 1er groupe
inchange A=B        W             *
        DAT0=A      16            *
        D1=D1+      16            *
        D1=D1+      2             *
        AD1EX                     *
        D=D-1       B             *
        GONC        loop          *
        GOVLNG      =GETPTRLOOP   *
ENDCODE
```

Application

Le programme ci-contre fera défiler l'affichage représenté pour deux situations intermédiaires sur les écrans de la page 38. Une pression de touche stoppera et gèlera l'affichage.

```
«"HP48  HP48  HP48" 3 DISP
" décalage circulaire
 d'un bit à droite" 3 DISP
→ → → → → → → " 5 DISP
DO MOVRIGHTC .05 WAIT
 UNTIL KEY END DROP 7 FREEZE »
```

SOUS-PROGRAMMES

En langage utilisateur, un programme peut appeler un autre programme stocké dans une variable et qui joue le rôle de sous-programme. Un sous-programme en assembleur est une suite d'instructions se terminant par une instruction de retour après l'instruction d'appel. Contrairement au Rpl, un sous-programme peut être à l'intérieur d'un programme, celui-ci pouvant appeler plusieurs fois le sous-programme.

Structure d'un sous-programme

label
    instruction 1
    .....
    instruction n
    instruction de retour

Instruction d'appel

Elle dépend de la longueur du saut (**offset**) au label

GOSUB label *offset -2048 à +2047*
GOSUBL label *offset -32768 à +32767*
GOSBVL =Label *ici appel à une adresse absolue alors que les deux autres appels sont relatifs*

Instruction de retour

RTN *simple retour*
RTNSC *retour en armant le bit Carry*
RTNCC *retour en désarmant le bit Carry*
RTNC *retour si le bit Carry est armé*
RTNNC *retour si le bit Carry est désarmé*
RTNYES *retour après un test positif*

La pile des retours RSTK:

Lors d'un appel à un sous-programme, l'adresse de l'instruction après l'appel est empilée sur la pile des retours qui comprend 8 niveaux. Dans une séquence de programme, lorsqu'une instruction de retour est rencontrée, un niveau de pile des retours est récupéré pour se brancher à cette adresse après l'appel. De façon courante on ne dispose que de 6 niveaux, deux étant réservés lors des interruptions par la machine. L'empilage d'adresses, ainsi que de valeurs sur 5 quartets, et l'opération inverse se font avec C(A).

RSTK=C *pousse C(A) sur la pile*
C=RSTK *enlève un niveau de pile*

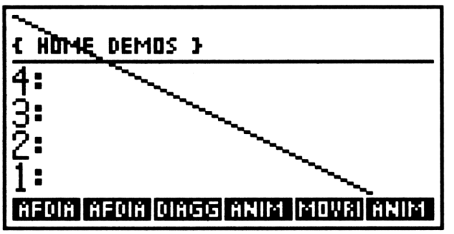
☛ Pour des valeurs sur 5 quartets, les niveaux de pile permettent les sauvegardes si on respecte l'ordre d'empilage et en étant attentif aux éventuels appels de sous programmes. Donc une possibilité, mais demandant de la vigilance.

AFFICHAGE D'UNE SÉRIE DE POINTS

Après les programmes de déplacements latéraux circulaires où nous avons dû manipuler des bits particuliers de données graphiques, le thème suivant portera sur l'allumage de points connaissant leurs coordonnées absolues. Il a été traité sous une forme différente dans [HtePerf] No 10.

Ce que l'on veut faire

Afficher une série de points suivant une oblique de direction donnée invariable. Ici, la direction y = x/2.



La démarche

On réalisera un sous-programme pour afficher un point. Ce sous-programme sera alors utilisé sans recourir à un algorithme général de tracé de droite : il suffira pour cela de construire la suite des points définis par leurs coordonnées absolues :

$$x_j = 2y_j \qquad x_{j+1} = 2y_j + 1$$

pour y allant de 0 à 55.

Principe de programmation

Le programme comportera trois parties.

Préparation de la boucle principale :
Adresse de la première ligne, compteur de lignes donnant aussi les valeurs des **y** de 55 à 0 (tracé du bas vers le haut).

Boucle principale :
Valeurs des coordonnées des points courants $\mathbf{P_j}$ et $\mathbf{P_{j+1}}$, sauvegarde de ces valeurs et appel (2 fois) du sous-programme **plot2** pour afficher ces points.

Sous-programme plot2 :
* Une ligne étant contenue dans 34 quartets, l'adresse du 1er quartet de la ligne j (0 à 55) est : (adresse début ligne 0) + 34*j soit **adrlj**.

* Un quartet contenant 4 bits, l'adresse du quartet contenant le point d'abscisse $\mathbf{x_j}$ (0 à 130) de la ligne j est **adrlj** + $\mathbf{x_j}$ **/4** (division entière).
Dans ce quartet le point $\mathbf{P_j}$ est le bit de rang égal à $\mathbf{x_j}$ modulo 4. Par exemple pour $\mathbf{x_j}$ = 6 le point sera dans le quartet en deuxième position à gauche °°°° et dans le registre en deuxième position à droite °°°°. $\mathbf{x_j}$ **/4** s'obtient avec 2 fois **rSRB.F fs** et modulo 4 avec AND (**r&s fs**), le registre s contenant 3 (# 0011b).
Le positionnement du bit du point s'obtenant par décalage à gauche dans le quartet 0001 suivant la valeur de $\mathbf{x_j}$ modulo 4 : 0010, 0100 ou 1000. Ce bit est injecté dans le quartet du point avec un OR (**r!s fs**).

Remarques

☛ Si le principe de base exposé ici est identique à celui de C.N. [HtePerf] No 10, les techniques mises en œuvre sont différentes. Il existe une 3ème méthode consistant à donner au pointeur P la valeur du modulo 16 et à charger une constante #.....24812481h, le quartet 0 aura la même valeur qu'avec la méthode ci-dessus.

☛ Dans le listing qui suit, on trouvera deux entrées de sous-programme :
* **plot** qui sert à vérifier la validité des valeurs x et y. Cette partie de programme ne sert pas dans le programme DIAGO car les x et y entrent dans les limites possibles (la "pente" de

la droite est 1/2). Mais de façon générale cette vérification est nécessaire sous peine de perturbation de la mémoire en cas de valeurs hors limites.

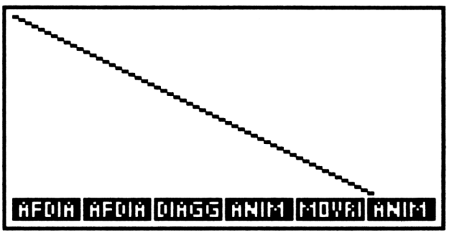
* **plot2** pour le tracé dans le cas de valeurs valables sans nécessité de contrôle.

☛ Un sous-programme doit faire le maximum d'opérations communes à plusieurs séquences de programmes pour réduire l'encombrement total du programme. Le sous-programme **plot2** aurait dû prendre en charge la sauvegarde des valeurs de x et y, cela n'a pas été fait pour assurer la compatibilité avec l'entrée **plot**.

Application

Il suffit juste de geler l'affichage après lancement du programme DIAGO pour obtenir l'écran ci-dessous.

« CLLCD **DIAGO** 7 FREEZE »



En supprimant CLLCD on obtient celui du début en surimpression.

Le programme **DIAGO** (72,5 octets CRC: # CC16h)

CODE

```
        GOSBVL      =SAVPTR       * Sauve les pointeurs
        GOSBVL      =D0→Row1      * D0 @ le début de la 1ère ligne
        C=0         A             * pour avoir des valeurs sur 5 quartets
        LC(2)       56-1          * nombre de lignes - 1 pour le compteur et yj
bclpt   RSTK=C                    * sauve le compteur et yj en valeur absolue
        D=C         A             * yj
        C=C+C       B             * xj = 2yj
        B=C         A             * sauve xj
        GOSUB       plot2         * appel du sous-programme de tracé du point Pj
        C=RSTK                    * C(A):= yj  (et aussi le compteur)
        RSTK=C                    * sauvegarde de ces valeurs
        D=C         A             * D(A):= yj
        C=C+C       B             * 2yj
        C=C+1       B             * xj+1 = 2yj + 1
        B=C         A             * sauve xj+1
        GOSUB       plot2         * appel de la routine de tracé du point Pj+1
        C=RSTK                    * C(A):= compteur de lignes
        C=C-1       B             * actualise le compteur
        GONC        bclpt         * si valeur C(B)≥0 alors encore un tour de boucle
        GOVLNG      =GETPTRLOOP   * retour au Rpl et fin de ce programme
```

(suite en page suivante)

```
* (suite du listing DIAGO)

plot
*                                        vérification de la validité des coordonnées
*       LC(5)       63          * C(A)=63 valeur maxi possible pour y
*       ?D>C        A           * y>63 ?
*       RTNYES                  * si oui retour de sous-programme sans rien faire
*       LC(2)       130         * C=130
*       ?A>C        A           * x>130 ?
*       RTNYES                  * si oui retour de sous-programme sans rien faire

plot2   AD0EX                   * A(A):= adresse début 1ère ligne (y = 0  x =  0)
        D0=A                    * restaure D0
        C=D         A           * C(A):= yj
        CSL         A           * 16 yj
        C=C+D       A           * 17yj
        C=C+C       A           * 34 yj = distance en quartets (ligne j - ligne 0)
        A=A+C       A           * A(A):= adresse début ligne j
        C=B         A           * C(A):= xj
        CSRB.F      A           * division entière: xj /2
        CSRB.F      A           * C(A):= quotient entier de xj /4
        A=A+C       A           * A(A):= adresse quartet contenant le point Pj
        D1=A                    * D1 @ ce quartet
        LCHEX       3           * C(0):=3 pour calculer xj modulo 4
        B=B&C       P           * B(0):= xj modulo 4 et valeur du compteur
        LCHEX       1           * C(0):= #0001b   bit 0 armé pour xj mod 4 = 0
        GONC        decr        * saut sur actualisation du compteur

bcl     C=C+C       P           * décalage des bits à gauche: 0010, 0100 ou 1000

decr    B=B-1       P           * actualise le compteur (reste de xj mod 4)
        GONC        bcl         * si B(0)≥0 alors encore un tour de boucle
        A=DAT1      1           * A(0):= quartet devant contenir le bit du point Pj
        A=A!C       P           * avec A OR C place le bit du point Pj dans A(0)
        DAT1=A      P           * replace en mémoire le quartet contenant Pj
        RTN                     * retour de sous-programme

ENDCODE
```

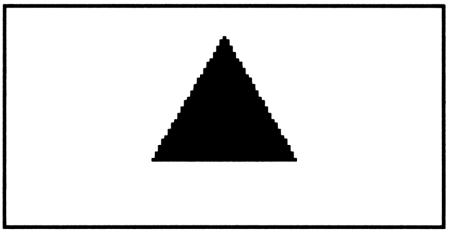
UN UTILITAIRE POUR HACHURER UNE ZONE

Les problèmes de graphisme traités jusqu'ici n'ont fait intervenir qu'un seul objet HP48 : le 'grob' de la pile. Le thème suivant mettra à profit les techniques et notions d'assembleur déjà utilisées mais pour construire un programme plus ambitieux nécessitant 4 objets de 2 types différents. Le résultat pourra s'assimiler à un petit utilitaire.

HACHURAGE PAR BALAYAGE HORIZONTAL

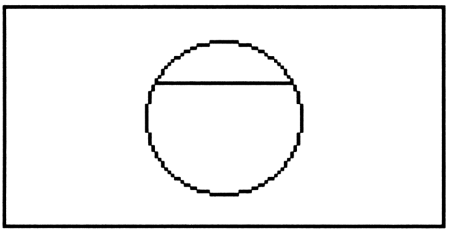
Ce que l'on veut faire

Hachurer la surface limitée par un contour comme sur l'écran ci-dessous. Le motif de hachurage est un objet graphique donné.



La démarche

Le motif de hachurage est un objet graphique 4x4. Le hachurage se fait par balayage suivant un segment sur la ligne **y**, entre $\mathbf{x_1}$ et $\mathbf{x_2}$, tel que sur l'écran ci-dessous. Le hachurage est effectué de façon cyclique dans le sens de la hauteur (**y** modulo 4) et par quartet horizontalement, le motif devant être conçu pour assurer l'homogénéité du hachurage sans ruptures.



Principe de programmation

Les arguments sont:
- le motif de hachurage: grob 4x4.
- les abscisses absolues $\mathbf{x_1}$ et $\mathbf{x_2}$, (2 réels).
- l'ordonnée absolue **y** du segment.

Les 3 coordonnées sont converties en entiers système (par **COERCE2** et **COERCE**). Après exécution du programme, le motif reste sur la pile, les 3 autres arguments sont éliminés.

Rappel de la structure d'un objet graphique :
Prologue, longueur, nombre de lignes, nombre de points par ligne puis les données.
Chaque ligne est contenue dans un multiple de 2 quartets. Pour un objet graphique 4x4 les données se présenteront de la manière suivante:

0000 0000
0000 0000
0000 0000
0000 0000

les bits soulignés sont les seules données utiles.

La détermination du quartet contenant le point $(\mathbf{x_1},\mathbf{x_2})$ se fera avec le même procédé que celui vu pour le programme **DIAGO**. La ligne du motif sera celle correspondant à **y** modulo 4.

Afin d'accélérer l'exécution, le segment sera décomposé en 3 parties, chacune étant traitée par une séquence de programme différente :

```
    oooo oooo oooo oooo ..... oooo oooo oooo
    ——— ——— ——— ——— ..... ——— ——— ———
x1 →      quartets entierss du segment      ← x2
```

- le premier quartet contenant le point x_1 sera traité du bit x_1 au dernier bit du quartet.
- les quartets complets du segment seront remplacés par la ligne du motif, donc très rapidement sans tests.
- le dernier quartet contenant le point x_2 sera traité du 1er bit du quartet au bit x_2 du segment donc de façon inversée par rapport au quartet contenant le point x_1.

Les explications détaillées concernant le traitement particulier de ces différentes parties sont développées dans le programme **HACH**.

Le programme **HACH** (212 octets CRC: # 83F7h)

```
:: COERCE2 ROT
COERCE
CODE
*                       niveau  4:      le motif (objet graphique 4x4)
*                       niveau  3:      x1 (entier système)
*                       niveau  2:      x2 (entier système)
*                       niveau  1:      y (entier système)

        GOSBVL          =POP#           * A(A):=y
        R0=A                            * sauve y en R0(A)
        GOSBVL          =POP2#          * A(A):= x1        C(A):= x2
        C=C-A           A               * ΔL = x2 - x1 = (longueur du segment) - 1
        R1=C                            * sauve ΔL en R1(A)
        AR0EX                           * sauve x1 en R0(A)  A(A):= y
        GOSBVL          =SAVPTR         * sauvegarde des pointeurs, D1 @ sur le niveau
*                                         du motif après drop des 3 premiers niveaux
        C=DAT1A                         * C(A):=adresse du motif
        D1=C                            * D1 @ le motif
        D1=D1+          15              * saute prologue,la longueur et dim 1
        D1=D1+          3               * D1 @ (adresse données du motif) -2
        B=A             A               * sauve y
        LCHEX           3               * pour modulo 4
        B=B&C           P               * B(0):= y modulo 4 = No de ligne du motif
bcl1    D1=D1+          2               * D1 @ la ligne j du motif
        B=B-1           P               * actualise le compteur de lignes du motif
        GONC            bcl1            * si B(0) ≥ 0 alors on va à la ligne suivante
        C=DAT1          1               * C(0):= données ligne j du motif
        D=C             P               * D(0):= données du motif pour la ligne y
        C=A             A               * C(A):= y pour libérer A
        GOSBVL          =D0→Row1        * D0 @ le contenu de la 1ère ligne de l'écran
        A=C             A               * A(C):= y
        ASL             A               * A(C):= 16y
        A=A+C           A               * A(C):= 17y
        A=A+A           A               * A(C):= 34y  conversion en quartets des y lignes
        CD0EX                           * C(A):=adresse de la 1ère ligne
        A=A+C           A               * A(A):=adresse de la ligne y
```

```
        D0=A                        * D0 @ le début de la ligne y
        LCHEX       3               * pour modulo 4
        A=R0                        * A(A):=x1
        B=A         A               * sauve x1
        C=C&B       P               * C(0):= x1 modulo 4
        ASRB.F      B               * x1/2
        ASRB.F      B               * A(A):= x1/4 (division entière)
        CD0EX                       * C(A):=adresse début de ligne y
        C=C+A       A               * C(A):=adresse du quartet du point x1 de ligne y
        CD0EX                       * D0 @ ce quartet  C(0) := x1 modulo 4
        P=C         0               * pointeur P:= x1 modulo 4
        C=R1                        * C(A):= ΔL = x2 - x1
        B=C         B               * sauve  ΔL = x2 - x1
        C=D         B               * C(0):=données du motif pour la ligne y
        A=DAT0      1               * A(0):=le quartet contenant le point x1
        ST=0        0               * pour repérer s'il reste des points du segment
        ?P=         3               * à partir d'ici on hachure depuis x1 ...
        GOYES       bit3            * ... jusqu'à la fin du 1er quartet du segment...
        ?P=         2               * ...suivant la valeur de P et de l'état du bit...
        GOYES       bit2            * ...correspondant de la ligne du motif
        ?P=         1               *
        GOYES       bit1            *
bit0    ?CBIT=0     0               * bit 0 de la ligne du motif non armé ?
        GOYES       d3              * si oui on passe au test suivant
        ABIT=1      0               * sinon arme le bit 0 du 1er quartet du segment
d3      B=B-1       B               * actualise le compteur de points du segment
        GOC         out1            * sortie des tests s'il n'y a plus de points du segm
bit1    ?CBIT=0     1               * suite de sequences semblables pour les points...
        GOYES       d2              * ...restants du 1er quartet du segment
        ABIT=1      1               *
d2      B=B-1       B               *
        GOC         out1            *
bit2    ?CBIT=0     2               *
        GOYES       d1              *
        ABIT=1      2               *
d1      B=B-1       B               *
        GOC         out1            *
bit3    ?CBIT=0     3               *
        GOYES       d0              *
        ABIT=1      3               *
d0      B=B-1       B               *
        GONC        load            *
out1     ST=1       0               * ici il ne reste plus de points du segment
load    P=          0               * pour le chargement de constantes
        DAT0=A      1               * replace le 1er quartet hachuré du segment
```

* (tourner la page pour la suite du programme HACH)

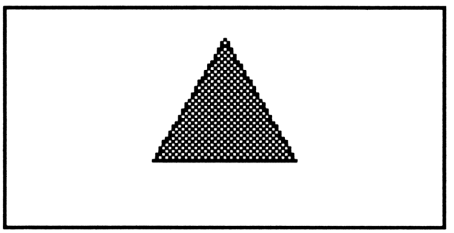
```
* (suite du programme HACH)
        ?ST=0           0               * encore des points du segment ?
        GOYES           cont            * oui on continue pour le ou les quartets suivants
        GOTO            out             * sinon sortie du hachurage
cont    D0=D0+          1               * D0 @ le quartet suivant
        B=B-1           B               * pour la répartition en quartets entiers
        GOC             excep           * 1 seul point restant (exception) alors saut (P=0)
        A=B             B               * reste longueur du segment
        ASRB.F          B               *
        ASRB.F          B               * A(A):= nombre de quartets entiers restants
        GONC            decr3           * saut au compteur de quartets
bcl3    C=DAT0          1               * C(0):= quartet k du segment
        C=C!D           P               * OR avec le quartet du motif
        DAT0=C          1               * replace le quartet k du segment
        D0=D0+          1               * D0 @ le quartet k+1 du segment
decr3   A=A-1           B               * actualise le compteur de quartets du segment
        GONC            bcl3            * si A(B):≥0 alors encore 1 tour de boucle
*                                         traitement du dernier quartet du segment
        LCHEX           3               * pour modulo 4
        C=C&B           P               * modulo 4
        P=C             0               * P prend la valeur de: (longueur restante) mod 4
excep   C=D             B               * C(0):= quartet du motif
        A=DAT0          1               * A(0):=quartet du segment contenant x2
        ?P=             0               * séquence de tests en fonction de P ayant pour...
        GOYES           p0              *..valeur (le nombre de points) - 1 pour atteindre
        ?P=             1               *... le point x2 de l'extrémité du segment
        GOYES           p1              *
        ?P=             2               *
        GOYES           p2              *
p3      ?CBIT=0         3               * bit 3 du quartet du motif non armé ?
        GOYES           p2              * oui alors saut au test suivant
        ABIT=1          3               * sinon arme le bit correspondant du segment
p2      ?CBIT=0         2               * même séquences avec les autre points
        GOYES           p1              *
        ABIT=1          2               *
p1      ?CBIT=0         1               *
        GOYES           p0              *
        ABIT=1          1               *
p0      ?CBIT=0         0               *
        GOYES           nobit2          *
        ABIT=1          0               *
nobit2  P=              0               * pour la sortie
        DAT0=A          1               * replacement du dernier quartet hachuré
out     GOVLNG          =GETPTRLOOP     * retour au Rpl avec P=0
ENDCODE
;
```

EXEMPLES DE HACHURAGES

Voici 10 motifs de hachurage contenus dans une liste et à stocker dans la variable **TRAMES**. Il suffira de rappeler cette liste sur la pile et de faire **n GET** pour obtenir le motif numéro n. La combinaison d'objets graphiques avec les commandes GOR et GXOR permet d'obtenir des images variées sans difficultés de programmation.

```
TRAMES
{ GROB 4 4 00000000        1: trame blanche
GROB 4 4 F0F0F0F0          2: trame noire
  GROB 4 4 50A050A0        3: trame T1
  GROB 4 4 00200080        4: trame T2
  GROB 4 4 40801020        5: trame T3
  GROB 4 4 20502000        6: trame T4
  GROB 4 4 20208080        7: trame T5
  GROB 4 4 003000C0        8: trame T6
  GROB 4 4 20108040        9: trame T7
  GROB 4 4 F0A0F0A0 }      10: trame T8
```

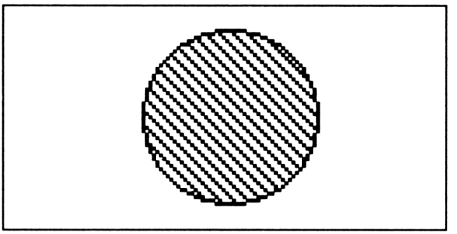
Exemple 1 : triangle hachuré
Il suffit de mettre sur la pile un motif et de lancer le programme **THACH**.



THACH

```
« ERASE { # 0h # 0h} PVIEW 37 0
   FOR Y Y 7 + 65 Y 3 √ 3 / * - LASTARG
     + CEIL HACH -1
  STEP { # 41h # 7h} { # 2Bh # 2Ch }
  LINE { # 2Bh # 2Ch} { # 57h # 2Ch }
  LINE { # 57h # 2Ch} { # 41h # 7h }
  LINE 7 FREEZE »
```

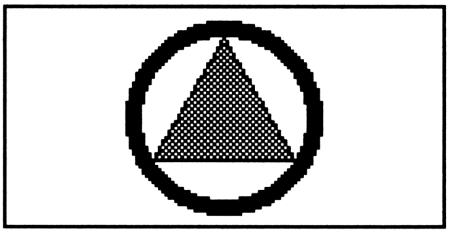
Exemple 2 : cercle hachuré
Lancer le programme CHACH avec au niveau 2 le motif et au niveau 1 le rayon du cercle.



CHACH

```
 « ERASE { # 0h # 0h} PVIEW 31 MIN
SWAP
  OVER DUP NEG
  FOR Y 32 Y - 65 4 PICK SQ Y SQ - √ -
    LASTARG + HACH -1
  STEP { # 41h # 20h } ROT R\->B 0
  360 ARC 7 FREEZE »
```

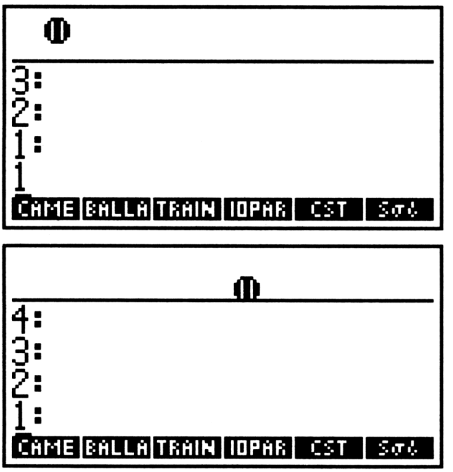
Voici pour terminer une combinaison de cercle et de triangle :



ANIMATION GRAPHIQUE ET SON

Les animations graphiques par déplacements répétitifs d'un bloc de lignes de l'écran étaient jusqu'à présent obtenues par des petits programmes en Rpl utilisateur faisant appel aux programmes en assembleur de déplacement par lignes ou par points. Cette fois toute l'animation sera faite en assembleur en y ajoutant un peu de son.

FAIRE REBONDIR UNE BALLE



Ce que l'on veut faire

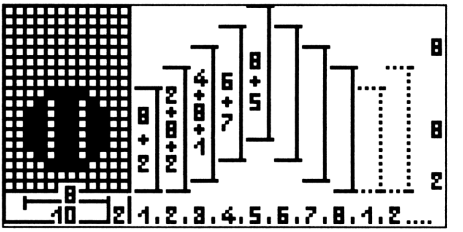
Créer une animation graphique dans la zone d'états de l'écran, sous forme d'une balle avançant vers la droite en rebondissant. Lors de la chute sur la ligne séparatrice, la balle s'aplatira et un son sera émis. Deux phases de cette animation sont visibles sur les écrans ci-dessus.

Les problèmes de compatiblité

Les différences de rapidité des HP48G/GX et S/SX ainsi que de contraste de l'écran posent des problèmes d'équivalence d'effets. Le programme **BALL** gèrera au mieux ces difficultés tant au niveau de l'animation que du son. Il faut signaler que si l'écran des HP48G/GX donne de meilleures images statiques par l'augmentation du contraste, l'écran des HP48S/SX est plus favorable aux images en mouvement et ne donne pas l'impression d'images 'baveuses'. De ce point de vue les mouvements doivent être ralentis sur une HP48G/GX pour limiter la dégradation des images.

La démarche

Il est possible de créer l'illusion du mouvement continu en opérant de façon discontinue sur des quartets et non des bits. Pour cela on part d'une image de la balle entourée de blancs, ceux-ci servant à masquer une partie de l'image précédente. Le déroulement s'effectue par cycles de 8 phases suivant le principe de la figure ci-dessous.



Le principe de programmation

Préparation de toutes les données nécessaires à l'exécution de la boucle principale gérant l'animation graphique. On détermine d'abord le type G ou S de la machine en doublant l'adresse du grob stack, cette opération armant le bit Carry pour un type G (adresse #8.....h) et le désarmant pour un type S (adresse #7....h). Le

type est repéré par le status bit 1. L'état de ST(1) déterminera la valeur du contraste, les compteurs de temporisation et le son suivant le type de machine.

Les données de l'image ci-dessus (balle entourée de blancs) sont incluses dans le programme et constituent une table de lignes (sur 3 quartets) à charger dans un registre pour y effectuer les décalages correspondant à l'avancement de la balle par cycles de 4 bits. L'évolution de celle-ci en hauteur sera obtenue en chargeant les lignes $\mathbf{h_1}$ à $\mathbf{h_2}$ de la table suivant les hauteurs indiquées sur le graphique. Le bloc de ces lignes se replacera à partir de la 2ième ligne de l'écran

Les cycles de 8 phases de déplacement de la balle seront gérés par la boucle principale appelant la routine de déplacement avec **C(A)** contenant le No de ligne de la table correspondant à $\mathbf{h_1}$ et **P** pour le compteur (par incrémentation) de lignes à charger.
Chaque partie du graphique correspondant à une phase de 1 à 8 sera repérée, dans les commentaires du progamme, par G1 à G8.

Utilisation d'une table de données
GOSUB dépose l'adresse après GOSUB sur la pile. Après la table cette adresse est récupérée permettant de faire pointer D1 ou D0 en début de table. Il suffit de se placer sur la ligne n désirée et de charger les quartets nécessaires.

```
        GOSUB datas
        NIBHEXnnn       début de la table
        ..................
        NIBHEXnnn
datas   C=RSTK          adresse de la table
        D1=C            D1 @ la table
```

Émission des sons
Une émission de sons peut se faire avec la routine **makebeep** qui présente le désavantage d'utiliser beaucoup de registres et oblige à faire des sauvegardes de ceux contenant des valeurs utiles. Ici on utilise une séquence de programme faisant usage du registre de sortie **OUT** (12 bits) et qui est commentée dans le sous-programme **sol** à partir de l'entrée **tone** du programme **BALL**.

Le programme BALL est listé ci-dessous et sur les trois pages suivantes. L'exécution du programme peut être stoppée par appui sur la touche [ATTN] ou [CANCEL].

Le programme **BALL** (249 octets CRC: # 2926h)

```
CODE
=CONTRAST    EQU    #00101
=Chk_attn    EQU    #04988
        GOSBVL       =SAVPTR     * Sauvegarde des registres
        CLRST                    * désarme les status bits 0 à 11
        D1=(5)       =CONTRAST
        C=DAT1       B           * valeur du contraste actuel
        RSTK=C                   * sauve la valeur du contraste sur la pile
        LC(2)        12          * valeur du contraste pour l'animation sur S/SX
        D0=(5)       #13299      * D0 @ l'adresse de l'adr. du début du grob stack
        A=DAT0       A           * A(A):=cette adresse
        D0=A                     * D0 @ l'adresse du début du grob stack
        A=A+A        A           * double l'adresse
        GONC         hp48s       * pas de dépassement de capacité donc type S/SX
```

* (à suivre sur trois pages)

```
* (suite du programme BALL)
        ST=1            1           * repère le type de machine G/GX
        LC(1)           15          * valeur du contraste pour l'animation sur G/GX
hp48s   DAT1=C          B           * charge la valeur du contraste
        A=DAT0          A           * A(A):=adresse du début du grob stack
        R2=A                        * sauve cette adresse en R2(A)
        D0=A                        * D0 @ le début des données du grob stack
        C=0             W           * pour charger les constantes suivantes
        P=              3
        CPEX            15          * C(S):=3 pour les calculs modulo 4.   P=0
        LC(2)           34-1        * pour le saut à la 2ième ligne - 1 quartet
        C=A+C           A           * C(A):=(adresse 2ème ligne) - 1
        D=C             W           * D(S) = 3 pour modulo 4  D(A):=adresse départ
        C=0             S
        C=C-1           S           * D(S):=#Fh ou -1 pour partir de 0 aprè (+ 1)
        R4=C                        * R4(S) = -1 = pour les cycles de décalage...
*                                     ... des bits modulo 4
        LC(5)           32          * avancement maximum en quartets
        D=D+C           A           * adresse limite pour l'avancement de la balle
        LC(2)           28          * compteur pour blanchiment de la zone d'états
        A=0             W           * pour charger en mémoire
loopcl  DAT0=A          16          * boucle de blanchiment de la zone d'états
        D0=D0+          16          * de la 2ème ligne à la 15ième, ligne séparatrice
        C=C-1           B           * conservée
        GONC            loopcl
        DAT0=A          12          * encore 12 quartets
*                                     ci-dessous les données du graphique de la balle
        GOSUB           grob        * le GOSUB dépose sur RSTK l'adresse suivante
        NIBHEX000000000000          * lignes 1 à 4 (haut du graphique général)
        NIBHEX000000000000          * lignes 5 à 8 du graphique général
        NIBHEX0F0861C63C63          * lignes 9 à 12
        NIBHEXC63C638610F0          * lignes 13 à 16 (bas du graphique général)
        NIBHEX000000                * lignes 17 et 18
grob    C=RSTK                      * C(A):= adresse du début des données
        R0=C                        * sauve adresse de cette table de données
loop    LC(5)           8*3         * pour un saut de 8 lignes dans la table
        P=              6           * pour le compteur de lignes de la table
        GOSUB           ball        * pour placer G1
        LC(2)           6*3         * puis phases suivantes 2,3, ...., 8
        P=              4           *
        GOSUB           ball        * placement de G2
        LC(1)           4*3         *
        GOSUB           ballp3      * placement de G3
        GOSUB           ballc6      * placement de G4
        P=              5           *
        GOSUB           sol         * placement de G5
```

```
        GOSUB     ballc6        * placement de G6
        LC(1)     4*3           *
        GOSUB     ballp3        * placement de G7
        LC(2)     6*3
        P=        4
        GOSUB     ball          * placement de G8
        ?ST=0     0             * position limite non atteinte ?
        GOYES     loop          * oui le cycle continue
fin     C=RSTK                  * récupère la valeur du contraste
        D0=(5)    =CONTRAST     * D0 @ la valeur du contraste
        DAT0=C    B             * charge la valeur initiale du contraste
        GOVLNG    =GETPTRLOOP   * retour au Rpl
*                                 ici différentes entrées du sous-programme
*                                 gérant les déplacements avec Gn (G1 à G8)
ballc6  LC(1)     2*3           * donnée commune à plusieurs appels
ballp3  P=        3             * même chose
ball    GOSUB     deplace       * déplace la balle
attn?   GOSBVL    =Chk_attn     * détecte un appui de touche ATTN ou CANCEL
        GONC      noattn        * si non appui alors on continue
        C=RSTK                  * sinon dépile l'adresse de retour de sous-prog.
        GOC       fin           * et saut pour la sortie
noattn  LCHEX     FD800         * pour boucle de temporisation HP48S/SX
        ?ST=0     1             * HP48S/SX ?
        GOYES     tempo         * oui on conserve le compteur de temporisation
        LCHEX     B600          * non alors temporisation HP48G/GX
tempo   C=C+1     A             * boucle de temporisation par incrémentation
        GONC      tempo         * et entre déplacements de la balle
        RTNCC                   * retour de sous-progr. et désarme le bit Carry
*                                 ici C(A):=valeur du saut à la ligne de la table
deplace A=R0                    * adresse de la table
        A=A+C     A             * adresse de la ligne de la table
        D1=A                    * D1 @ la 1ère ligne du bloc Gn
        C=R4                    * C(A):=adresse quartets à charger et
*                                 C(S):=valeur du cycle d'avancement des bits
        C=C+1     S             * actualise cette valeur
        C=C&D     S             * valeur modulo 4
        ?C#0      S             * si C(S):=0 alors un cycle complet de décalage
        GOYES     nolimit       * ...des bits est réalisé: décalage d'un quartet
        C=C+1     A             * avance d'un quartet
        ?C<D      A             * limite d'avancement en quartets non atteinte ?
        GOYES     nolimit       * oui on ne fait rien
        ST=1      0             * sinon repére l'état de limite atteinte
nolimit R4=C                    * sauve adresse du quartet actuel et la...
*                                 ... valeur du cycle de décalage des bits
        D0=C                    * @ 1er quartet à charger
```

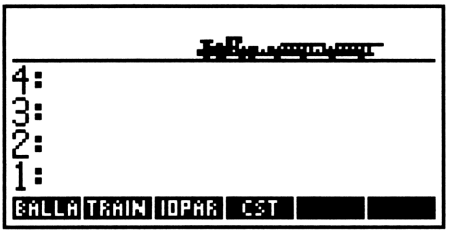
* (à suivre, tourner la page)

```
load    B=C         S       * compteur pour décalage
        A=0         A       * pour travailler sur 4 quartets
        A=DAT1      3       * C(A):=00ddd où ddd= ligne de la table
        ?A=0        A       * ligne blanche ?
        GOYES       fait    * il est inutile de faire des décalages de bits
        B=B-1       S       * actualise le comteur de décalage
        GOC         fait    * après un décalage d'1 quartet pas de bits à...
decal   A=A+A       A       * ...décaler. Ici décalage d'1 bit à gauche.
        B=B-1       S       * actualise le compteur de décalage de bits
        GONC        decal
fait    ?ST=0       0       * limite d'avancement atteinte ?
        GOYES       norac   * oui alors chargement normal
        DAT0=A      B       * sinon chargement de 2 quartets au lieu de 4
        GONC        cont    * saut après chargement
norac   DAT0=A      4       * chargement normal de 4 quartets
cont    D0=D0+      16      * pour passage à la ligne suivante de l'écran
        D0=D0+      16      * par avancement de 16+16+2=34 quartets
        D0=D0+      2       * D0 @ la ligne j+1 de l'écran
        D1=D1+      3       * D1 @ la ligne suivante du graphique
        P=P+1               * actualise le compteur par incrémentation
        GONC        load    * si non passage de p=15 à p=0 alors on réitère
        RTNCC               * retour de sous-programme, bit Carry désarmé
*                           * routine pour la chute sur le sol et bruit d'impact
sol     GOSUB       deplace * déplacement de la balle vers le sol
tone    LC(2)       4       * pour le nombres de cycles d'émission du son
        B=C         A       * sauve cette valeur
        LC(2)       40      * pour la durée d'émission du son
        ?ST=0       1       * HP48S/SX ?
        GOYES       48s2    * si oui on conserve les valeurs
        B=B+1       B       * sinon 1 cycle de plus pour HP48G/GX
        LC(2)       50      * et nombre d'itérations augmenté
48s2    A=C         A       * sauve cette valeur
loopt   LCHEX       800     * # 1000000000000b arme le bit 11 de C(X)
        OUT=C               * le bit 11 du registre OUT armé produit le déclic
        C=A         B       * pour la longueur d'émission
decr1   C=C-1       B       * boucle de longueur d'émission
        GONC        decr1   *
        C=0         A       * pour l'arrêt du déclic
        OUT=C               * désarme le bit 11 de OUT, le déclic s'arrête
        C=A         B       * pour la 2ème temporisation
decr2   C=C-1       B       * boucle de silence
        GONC        decr2   *
        B=B-1       B       * actualise le compteur de la boucle des cycles
*                             ...d'émission du déclic et des silences...
        GONC        loopt   * du son. Si B(B):≥0 encore 1 tour de boucle
        GOTO        attn?   * sinon teste de sortie et temporisation
ENDCODE
```

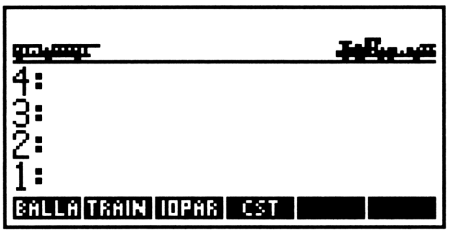
ANIMATION ET GESTION DE TOUCHE

L'animation graphique de la balle pouvait être stoppée par appui sur une touche particulière: [ATTN] ou [CANCEL], la détection d'appui étant confiée à la routine Chk_attn. L'animation suivante pourra évoluer à la suite de différents appuis de touches. Cela nous amène à de nouvelles notions.

SIMULATION DU MOUVEMENT D'UN TRAIN



Ce que l'on veut faire
un déplacement continu, dans la zone d'états de l'écran, de l'image d'un train rétro (écran du dessus). Le déplacement circulaire fera réapparaître le train à droite (écran ci-dessous).



Les actions suivantes seront provoquées par appuis sur les touches:
- **[STO]**: arrêt du déplacement de l'image tant que la touche est enfoncée.
- **[-]**: vitesse lente.
- **[+)**: vitesse rapide.
- **[➤]**: marche arrière. Un appui sur [STO] fait passer en marche avant.
- **[CST]**: rétablit la vitesse moyenne.
- **[ATTN]** ou **[CANCEL]**: sortie du programme.

La démarche
Les vitesses différentes de déplacement sont obtenues avec des temporisations plus ou moins longues entre générations d'images. Les glissements de celles-ci résultent des déplacements latéraux et circulaires du bloc des lignes 2 à 15 du haut de l'écran.

Principe de programmation
Comme précédemment, il faut préparer la boucle principale avec détection du type de machine, réglage du contraste, fournir l'adresse du grob stack et blanchir la zone d'états.

L'image du train est incluse dans le programme, il faut la charger en mémoire-écran dans la position de départ.

La boucle gère les interceptions de touches et appelle les sous-programmes pour les mouvements d'images.

Les sous-programmes :

- **leftcir** identique à MOVLEFTC mais pour un déplacement d'un bloc de 14 lignes contre 56.
-**movrc** identique à MOVRIGHTC mais pour un bloc de 14 lignes.
- **valpause** pour charger et sauvegarder les valeurs des constantes de temporisations en fonction du type de machine.

Les nouvelles notions du programme TRAIN ne concernent que la gestion des appuis de touches.

Les détections d'appuis de touches

Pour le programme BALL, l'émission d'un son a nécessité l'usage du registre OUT. La détection d'un appui de touche se fera à l'aide du couple des registres d'entrée et sortie **IN OUT**.

Le registe **IN** (4 quartets) s'utilise avec les instructions **A=IN** ou **C=IN** mais celles-ci étant à une adresse paire. Pour cela on fera appel aux routines en Rom **AINRTN**, **CINRTN** ou **OUTCIN** d'adresse # 01EECh contenant la séquence :

```
OUT=C          GOVLNG =CINRTN
```

Pour tester l'appui d'une touche on charge le registre **OUT** avec la valeur du tableau ci-dessous correspondant à la touche. En retour le registre IN aura le bit n armé, suivant la valeur du tableau , si la touche a été enfoncée.
exemple pour la touche [CST]:

```
LCHEX          080
GOSBVL         =OUTCIN
?CBIT=1              3
GOYES          appuiCST
```

car pour IN, #0008h=#....01000b donc le bit 3 armé. Pour la touche **[ON]** il suffit de savoir si le bit 15 du registre IN est armé sans charger de masque.

Pour des informations complémentaires, en particulier pour les combinaisons de touches, on pourra consulter les ouvrages [CourS], [CourG], [FerrS], [FerrG], [Kezir], [Nalec] ou [Donne].

Les interruptions seront inhibées avec **DisableIntr** et rétablies, après la fin des tests de touches, avec **AllowIntr**.

VALEURS DES MASQUES POUR LES COUPLES OUT/IN

Touche	A	B	C	D	E	F
OUT	002	100	100	100	100	100
IN	0010	0010	0008	0004	0002	0001
Touche	**MTH**	**PRG**	**CST**	**VAR**	**↑**	**NXT**
OUT	004	080	080	080	080	080
IN	0010	0010	0008	0004	0002	0001
Touche	**'**	**STO**	**EVAL**	**←**	**↓**	**→**
OUT	001	040	040	040	040	040
IN	0010	0010	0008	0004	0002	0001
Touche	**SIN**	**COS**	**TAN**	**√x**	**y^x**	**$1/x$**
OUT	008	020	020	020	020	020
IN	0010	0010	0008	0004	0002	0001
Touche	**ENTER**	**+/-**	**EEX**	**DEL**	**⇐**	
OUT	010	010	010	010	010	
IN	0010	0008	0004	0002	0001	
Touche	**α**	**7**	**8**	**9**	**÷**	
OUT	008	008	008	008	008	
IN	0020	0008	0004	0002	0001	
Touche	**↶**	**4**	**5**	**6**	**x**	
OUT	004	004	004	004	004	
IN	0020	0008	0004	0002	0001	
Touche	**↷**	**1**	**2**	**3**	**-**	
OUT	002	002	002	002	002	
IN	0020	0008	0004	0002	0001	
Touche	**ON**	**0**	**.**	**SPC**	**+**	
OUT	400	001	001	001	001	
IN	8000	0008	0004	0002	0001	

Le programme **TRAIN** (356 octets CRC: # 8D0h)

```
CODE
=OUTCIN         EQU     #01EEC          * voir ci-dessus
        GOSBVL          =SAVPTR         * Sauvegarde des registres
        GOSBVL          =DisableIntr    * interdit les interruptions
        D1=(5)          =CONTRAST       * D1 @ la valeur du contraste
        C=DAT1          B               * valeur du contraste à sauvegarder
        RSTK=C                          * sauvegarde sur la pile des retours
        LC(2)           12              * valeur du contraste pour HP48S/SX
        D0=(5)          #13299          * D0 @ l'adresse de l'adresse du grob stack
        A=DAT0          A               * A(A):=cette adresse
        D0=A                            * D0 @ l'adresse du début du grob stack
        ST=0            1               * pour repérer le type HP48S/SX
        A=A+A           A               * double cette adresse (# 7....h ou # 8...h)
        GONC            hp48s           * si #7....h alors type HP48S/SX, on ne fait rien
        ST=1            1               * sinon repère de type HP48G/GX
        LC(1)           15              * et valeur du contraste pour HP48G/GX
hp48s   DAT1=C          B               * charge la valeur du contraste
        GOSUB           valpause        * détermine, sauve la vitesse moyenne par défaut
        A=DAT0A                         * A(A)=adresse début données du grob stack
        D0=             A               * D0 @ la 1ère ligne
        D1=A                            * D1 @ la 1ère ligne pour le blanchiment
        R4=A                            * sauve l'adresse de la 1ère ligne en R4(A)
        A=0             W               * pour le blanchiment
        LC(2)           28              * 34*14=(28+1)*16 + 12 quartets à blanchir
bclclear DAT1=A         16              * boucle pour le blanchiment du haut de l'écran
        D1=D1+          16              * soit 29*16 quartets
        C=C-1           B               *
        GONC            bclclear        *
        DAT1=A          12              * et 12 quartets...
        GOSUB           load            * pour l'adresse de l'image du train ci-dessous
        NIBHEX00E300000000000           * les données de l'image du train: 7 lignes de 60
        NIBHEXE16300000000000           * ...points soit 7*15 quartets
        NIBHEXCC6100CFFF8FFF3           *
        NIBHEXCFF57CAAA8AAA20           *
        NIBHEXFF7F7DFFBAFFF20           *
        NIBHEXFFFFFFFFFFFF70            *
        NIBHEXCC1630303060C00           *
load    C=RSTK                          * C(A):=adresse de la 1ère ligne du train
        D1=C                            * D1 @ cette ligne
        AD0EX                           * A(A):= adresse de la 1ère ligne de l'écran
        LC(5)           34*7+19         * pour le 76ème point de la 8ième ligne
        A=A+C           A               * adresse du quartet contenant ce point
        D0=A                            * D0 @ ce quartet: départ de l'image du train
        P=              9               * pour charger les 16-9=7 lignes du train
```

```
bclload  A=DAT1      15        * boucle chargeant en mémoire écran les 7 lignes
         DAT0=A      15        * de l'image du train
         D1=D1+      15        * D1 @ la ligne suivante des données ci-dessus
         D0=D0+      16        * pour avancer sur le point 76 de la ligne
         D0=D0+      2         * suivante de l'écran
         D0=D0+      16        *
         P=P+1                 * actualise par incrémentation le compteur de
         GONC        bclload   * lignes. Si non passage de P=15 à P=0, boucle
******** ENTRÉE DANS LA BOUCLE GÉNÉRALE GÉRANT LES INTERCEPTIONS DE TOUCHES ********
bclleft  GOSUB       leftcir   * déplacement du train en marche avant
stop     ST=0        2         * pour le repère de marche avant
itstop   LCHEX       02800     * valeur pour augmenter ou diminuer la vitesse
         B=C         A         * sauve cette valeur
         C=R1                  * C(A):= valeur temporisation à vitesse actuelle
pause    C=C+1       A         * boucle de temporisation par incrémentation
         GONC        pause     * sortie de boucle si C(A):=0
         A=R0                  * A(A):= valeur temporisation à vitesse moyenne
         C=C+1       A         * 0 + 1 → 001 pour masque touche [+]
         GOSBVL      =OUTCIN   * voir principe de programmation
         ?CBIT=0     0         *  non appui sur la touche [+] ?
         GOYES       no+       * si oui saut au test suivant
         A=A+B       A         * sinon incrémente valeur tempo.: vitesse rapide
         R1=A                  * sauve la nouvelle valeur de temporisation
         GONC        nocst     * saute les tests pour les 2 autres vitesses
no+      LCHEX       002       * masque pour touche [-]
         GOSBVL      =OUTCIN   * voir détections d'appuis de touches
         ?CBIT=0     0         * non appui sur touche [-] ?
         GOYES       no-       * si oui  alors test suivant
         A=A-B       A         * sinon décrémente valeur de tempo: vitesse lente
         R1=A                  * sauve la nouvelle valeur de temporisation
no-      LCHEX       080       * masque pour touche [CST]
         GOSBVL      =OUTCIN   * voir détections d'appuis de touches
         ?CBIT=0     3         * non appui sur touche [CST] ?
         GOYES       nocst     * si oui saut au test suivant
         GOSUB       valpause  * rétablit la valeur de tempo. vitesse moyenne
nocst    LCHEX       040       * masque pour touche [STO]
         GOSBVL      =OUTCIN   * voir détections d'appuis de touches
         ?CBIT=1     15        * appui sur [ON] ? (masque non nécessaire
         GOYES       fin       * si oui sortie définitive
         ?CBIT=0     4         * non appui sur touche [STO] ?
         GOYES       nostop    * si oui saut au test suivant
         GOTO        stop      * sinon saut pour une temporisation
nostop   ?CBIT=0     0         * non appui sur touche [➢] ?
         GOYES       noarr     * si oui saut au test suivant
         ST=1        2         * sinon repère de marche arrière
         GONC        arr       * et saut à marche arrière
```

```
noarr   ?ST=1          2              * marche arrière ?
        GOYES          arr            * si oui saut à marche arrière
        GOTO           bclleft        * sinon saut en début de boucle à marche avant
arr     GOSUB          movrc          * marche arrière du train suivant vitesse
        GOTO           itstop         * saut en début de boucle pour temporisation
fin     C=RSTK                                * récupère le contraste d'origine
        D0=(5)         =CONTRAST      * D0 @ la valeur du contraste
        DAT0=C         B              * charge la valeur de l'ancien contraste
        GOSBVL         =AllowIntr     * autorise les interruptions
        GOVLNG         =GETPTRLOOP    * restaure les pointeurs et retour au Rpl
```

** sous-programme effectuant un décalage circulaire à gauche sur un bloc de 14 lignes d'écran*

```
leftcirC=0      W      nobit1CSRB
      P=        8            DAT0=A  16
      CPEX      15           D0=D0+  16
      B=C       S            D1=D1+  16
      LC(2)     13           A=DAT1  B
      D=C       A            ?ABIT=0 0
      A=R4                   GOYES nobit2
loop  D0=A                   C=C+B   S
      D1=A           nobit2ASRB.F   B
      D1=D1+  16           ?ST=0   0
      A=DAT0  16           GOYES   noaj
      ST=0    0            ABIT=1  2
      ?ABIT=0 0      noaj  DAT1=A  B
      GOYES nodeb          DAT0=C  16
      ST=1    0            D1=D1+  2
nodebASRB                  AD1EX
      C=DAT1  16           D=D-1   B
      ?CBIT=0 0            GONC    loop
      GOYES nobit1         RTN
      A=A+B   S
```

** sous-programme effectuant un décalage circulaire à droite sur un bloc de 14 lignes d'écran*

```
movrcLC(2)      13           A=A+1   B
      D=C       A      nodepDAT1=A   B
      A=R4                 D1=D1-  16
looprcD0=A                 B=B+B   W
      D1=A                 GONC    nodep2
      D1=D1+  16           C=C+1   W
      A=DAT0  16     nodep2 DAT1=C 16
      B=A     W            ?ST=0   0
      C=DAT1  16           GOYES inchange
      D1=D1+  16           B=B+1   A
      A=DAT1  B      inchange A=B  W
      ST=0    0            DAT0=A  16
      ?ABIT=0 2            D1=D1+  16
      GOYES noreport       D1=D1+  2
      ST=1    0            AD1EX
noreport A=A+A  B          D=D-1   B
      C=C+C   W            GONC    looprc
      GONC    nodep        RTNCC
```

```
* sous-programme chargeant et sauvegardant les valeurs de temporisation du type de HP48
valpause LCHEX         FB800          * valeur pour temporisation type HP48G/GX
         ?ST=1         1              * HP48G/GX ?
         GOYES         hp48g          * si oui saut pour sauvegardes sinon charge la
         LCHEX         D000           * valeur pour temporisation type HP48S/SX
hp48g    R0=C                         * sauvegarde pour vitesse moyenne par défaut
         R1=C                         * sauvegarde pour vitesse variable
         RTNCC                        * retour de sous-programme
ENDCODE
```

EN GUISE DE CONCLUSION

Ce modeste fascicule n'étant pas une "bible" de l'assembleur HP48, il n'est pas possible d'épuiser le sujet (même les gros ouvrages ne le font pas). Aussi, pour terminer, voici quelques éléments d'ordre pratique n'étant pas ou peu abordés dans la plupart des ouvrages.

CHOIX ET OPTIMISATION

Le choix des instructions

Au début du chapitre *premiers programmes* le listing du programme SWAP13 montre le code correspondant à chaque instruction. Les instructions du type r=DAT1 A ou DAT1=r A sont codées sur 3 quartets. Les instructions donnant le même résultat r=DAT1 5 ou DAT1=r 5 sont codées sur 4 quartets ce qui se traduit par 33% d'encombrement mémoire supplémentaire. D'autre part les temps d'exécution de ces dernières intructions sont plus importants que ceux des premières.

Il faut préférer le chargement des constantes avec le registre C et non A.

Pour les échanges entre registres de sauvegarde et de travail, il faut penser à la future utilisation des valeurs sauvegardées, par exemple:

Mauvais		**Bon**	
LC(5)	nnnnn	C=0	W
RO=C.F	A	LC(5)	nnnnn
LC(5)	mmmm	R0=C	
R1=C.F	A	LC(5)	mmm
.....		R1=C	
A=0	W	.....	
A=R0.F	A	A=R0	
C=0	W	C=R1	
C=R1.F	A	GOSBVL =MPY	
GOSBVL =MPY			

MPY faisant le produit des valeurs de A(W) et C(W). Les instructions r=ss et ss=r de droite (champ W implicite) sont codées sur 3 quartets, celles de gauche (champ spécifié) sont codées sur 6 quartets. Les instructions de gauche avec le champ A sont plus rapides mais en fin de compte la séquence de droite est plus courte en code et plus rapide.

Pour le code des différentes instructions on pourra consulter [CourS], [CourG], [FerrS], [FerrG], [Kezir] ou [CoubG]. Pour les temps en cycles machine les ouvrages [CourS] (3ème édition), [CourG] ou [CoubG] donnent les informations les plus complètes et actualisées.

Utiliser ou non les routines en Rom HP ?

Dès les premiers programmes, des routines HP ont été utilisées. Cela a permis d'obtenir des programmes plus compacts et de clarifier les listings. Cette pratique n'a pas toujours que des avantages. En effet certaines routines de type 'généraliste' peuvent être remplacées par des séquences de programme plus spécialisées et utilisant moins de registres donc nécessitant moins de sauvegardes. D'autre part il ne faut pas oublier que cette routine peut être appelée d'un niveau de sous-programme et comporter elle-même plusieurs niveaux de sous-programmes dépassant la capacité de la pile RSTK.

Enfin ces routines sont à utiliser en connaissance de cause: les valeurs en entrée, celles en sorties, les registres modifiés et le nombre de niveaux de sous-programmes utilisés. Pour cela on trouvera un échantillon de routines documentées dans [CourS], [CourG], [FerrS], [FerrG], [Kezir] ou [DONNE].

Travailler avec les champs et le pointeur P

Le temps pour effectuer des opérations ou des échanges avec les registres dépend de l'étendue des champs opérationnels. Le champ A est pri-

vilégié à la fois pour la longueur de codage des instructions et les temps d'exécution.
Le pointeur P et le champ WP peuvent permettre une exécution plus rapide pour des opérations sur un champ variable (voir § suivant). Il est recommandé de comparer le code et les temps dans les ouvrages cités au § *choix des instructions*.

Adapter les routines en Rom HP

Par exemple la routine **MPY** [# 53EE4h) effectue le produit de A(W) par C(W) et donne le résultat en A(W) et C(W):

```
MPY     D=C         W
        C=0         W
mpy1    SB=0
        ASRB
        ?SB=0
        GOYES       mpy2
        C=C+D       W
mpy2    D=D+D       W
        ?A#0        W
        GOYES       mpy1
        A=C         W
        RTNCC
```

À la fin, A(W) avait la valeur 0 avant A=C W. Cette dernière opération permet au programme appelant d'avoir indifféremment le résultat en A ou C mais avec une instruction de plus et la nécessité de faire A=0 W le cas échéant. D'autre part une utilisation intensive de MPY pour des produits sur un maximum de 6 quartets nécessiterait de travailler avec le champ WP défini par P= 6. Avec toutes ces conditions, MPY débutant avec P=5, en changeant le champ W pour WP et en supprimant A=C W inutile, on obtiendrait une meilleure rapidité (rétablir P=0 en fin de MPY).

Utiliser les bonnes instructions de retour

Par exemple:

```
        GOSUB  ssprog
        GONC   ok1
        .....
ok1     .....
        GOTO   fin
ssprog  .....
        ?A=C   A
        GOYES  retour
        A>B    A
        GOYES  retour
        A=A+A  A
        RTNC
        .....
retour  RTNCC
```

Le retour avec RTNCC permettra, après des opérations ou des tests de types totalement différents armant ou désarmant le bit Carry, de continuer une même séquence au retour du sous-programme alors que pour le débordement de A=A+A A, armant aussi le bit Carry, le retour de sous-programme sera traité autrement.

Préparer avec soins les boucles

La préparation soignée des boucles a déjà été signalée dans les principes de programmation marqués ☛. Une boucle, même courte mais réitérée des dizaines de milliers de fois, peut ralentir considérablement l'exécution d'un programme si elle n'a pas été soigneusement préparée et conçue. Le maximum d'opérations possibles doit être réalisé avant l'entrée dans la boucle pour décharger celle-ci de tâches répétitives inutiles.

Le bon choix des sauts

Il faut toujours choisir le type de saut le plus court possible (moins de code et plus rapide), donc préférer un GONC ou GOC à un GOTO suivant l'état du bit Carry. Par exemple, voir dans les programmes l'entrée dans une boucle au niveau du compteur permettant ainsi un test et un saut court avec GONC.

Le choix de l'algorithme

Avant de programmer il faut construire la solution du problème en faisant appel à un ou des algorithmes. D'eux dépendra la rapidité d'exécution du programme et un mauvais choix d'algorithme peut réduire à néant l'efficacité d'un bon codage.

DERNIERS CONSEILS

Au terme de ce guide, il est bon de rappeler certaines précautions qui évitent de gâcher une création ou de rendre inutilisable un long travail de programmation. D'autre part quelques recettes simples pour progresser pourront être profitables.

Attention : HP48 amnésique !
Tous les programmeurs, en langage assembleur ou Rpl-système, redoutent le message **Memory Clear** lors de l'exécution d'un programme qu'ils viennent de créer. Il est impératif de sauvegarder tout ce qui est en Ram et utile avant exécution d'une nouvelle création non en Rpl utilisateur. Pour ceux qui développent sur HP48, sans posséder de PC, il est absolument nécessaire d'avoir des extensions mémoire sous forme de cartes Ram verrouillables, pour tout sauvegarder, y compris le fichier source. Pour ceux développant sur PC, solution la plus confortable, il faut sauvegarder sur disque dur (ou cartes Ram verrouillées) ce qui existe en Ram HP48, le fichier source restant dans le PC.

Attention : programmeur amnésique !
Un programmeur débutant a tendance à brûler les étapes et à négliger de documenter ses programmes. Ce n'est pas un gain de temps, bien au contraire, car il sera souvent nécessaire de réutiliser des routines que l'on a créées ou de travailler d'une façon discontinue sur un programme et, dans ce cas, il sera difficile de s'y retrouver dans sa propre programmation. Il ne faut pas hésiter à documenter les fichiers sources même simplement sur papier si l'on ne possède pas de PC.

Analyser les programmes des autres
Que ce soit dans les ouvrages, les journaux des clubs ou les documentations sur support magnétique, les listings des programmes bien commentés constituent une aide précieuse pour acquérir des techniques de programmation. Il ne faut pas hésiter, sur le papier, à relier les instructions de sauts et les labels correspondants pour suivre la progression de la programmation qui n'a pas, pour l'assembleur, l'aspect structuré des autres langages.
L'étape suivante sera de désassembler des routines de la Rom HP. Il n'y a que l'embarras du choix. Les décompilateurs utilisant les mnémoniques HP apporteront un meilleur confort en traduisant les adresses des routines en mnémoniques. Une trace écrite de cette analyse devra être conservée avec les commentaires, les données en entrée, celles en sortie et les modifications des registres. Tout ceci permettra une utilisation ultérieure facile ou même une adaptation comme cela a été vu pour la routine MPY.
Toute analyse doit se faire avec un esprit critique et ne pas considérer qu'un programme existant est toujours le meilleur. Cela est valable pour ceux de ce fascicule comme ceux contenus dans les ouvrages. Pour la programmation de la Rom HP il faut tenir compte de la polyvalence de certaines routines entrainant une surabondance d'instructions qui permet des appels en différents points d'entrée.

Autres sujets
Si le graphisme et les animations graphiques, pouvant aboutir aux jeux, sont des sujets attractifs, d'autres thèmes méritent d'être traités en *assembleur*: petits utilitaires pour manipuler les nombreux types d'objets HP48, jeux de réflexion (le compte est bon [48sxtant] 34), calculs en virgule flottante ([PerfCalc] 1,.[48sxtant] 38), récursivité (PerfCalc] 2, [48Sxtant] 34)

Conclusion
Malgré l'âge de leur microprocesseur, les calculatrices HP48 restent de merveilleux outils de créativité. Elles ont déchaîné beaucoup de passions parmi les jeunes et les moins jeunes. Les créations en langage assembleur sont nombreuses et leur nombre devrait encore augmenter grâce aux nouveaux adeptes. Nous espérons que le lecteur viendra grossir leurs rangs et que ses créations figureront en bonne place dans l'œuvre collective de Performance Calcul.

LISTE D'ADRESSES UTILISEES

Voici la liste des adresses des routines HP utilisées dans les programmes de ce fascicule. Elles sont fournies pour les lecteurs qui utiliseraient encore ASM-FLASH + la bibliothèque DEV.LIB.

Mnémonique HP	Type	Adresse	1ère utilisation (page)
::	(Rpl système)	# 02D9Dh	20
;	(Rpl système)	# 0312Bh	20
AINRTN	(assembleur)	#0115Ah	56
AllowIntr	(assembleur)	# 010E5h	56
CK&DISPATCH1	(Rpl système)	# 18FB2h	20
CINRTN	(assembleur)	#01160h	56
CK1NOLASTWD	(Rpl système)	# 18AB2h	20
CODE	(Rpl système)	# 02DCCh	9
COERCE	(Rpl système)	# 18CEAh	46
COERCE2	(Rpl système)	# 194F7h	46
CONTRAST		#00101h	51
D0→Row1	(assembleur)	# 01C31h	12
DisableIntr	(assembleur)	# 01115h	56
FIFTYONE	(Rpl système)	# 64B58h	22
GETPTR	(assembleur)	# 067D2h	9
GetStrLenStk	(assembleur)	# 2FFB4h	20
GPErrJmpC	(assembleur)	# 10F40h	25
GETPTRLOOP	(assembleur)	# 05143h	9
Loop	(assembleur)	# 2D564h	17
MAKE$N	(assembleur)	# 05B7Dh	25
MOVEDOWN	(assembleur)	# 0670Ch	29
MPY	(assembleur)	#53EE4	60
POP#	(assembleur)	# 06641h	46
POP2#	(assembleur)	# 03F5Dh	46
PopASavPtr	(assembleur)	# 3251Ch	22
ROT	(Rpl système)	# 03295h	26
SAVPTR	(assembleur)	# 0679Bh	9
str (ou **THREE**)	(Rpl système)	# 0400Dh	20
SWAP	(Rpl système)	# 03223h	22
TOTEMPOB	(Rpl système)	# 06657h	20
TOTEMPSWAP	(Rpl système)	# 62C69h	22
UNROT2DROP	(Rpl système)	# 6112Ah	27

DOCUMENTATION ET BIBLIOGRAPHIE

Dans les pages de ce guide, la référence à une documentation, listée ci-dessous, pourra se faire simplement par la mention entre crochets, celle-ci se terminant par S ou G pour ce qui est spécifique aux HP48S/SX ou G/GX.

Ouvrages sur l'assembleur HP48 :

[CourS] Paul Courbis, Sébastien Lalande *Voyage au centre de la HP48S/SX* Editions Angkor.
[CourG] Paul Courbis *Voyage au centre de la HP48G/GX* Ed. Angkor. Actualisé pour G/GX.
[FerrS] Jean Michel Ferrard *Les secrets de la HP48* Editions D3I (1ère édition S/SX)
[FerrG] Jean Michel Ferrard *Les secrets de la HP48G/GX* Editions D3I
[Kezir] Philippe Kezirian *Assembleur sur HP48* Editions D3I
[CoubG] P. Courbis & Cyrille de Brébisson *Le compagnon de voyage de la HP48G/GX* Ed. Angkor.
[Nalec] Olivier Nachba, Cédric Lecourt *Initiation à l'assembleur* Editions Angkor.
[Donne] James Donnelly *An Introduction to HP48 System Rpl and Assembly Language Programming* Armstrong Publishing Company Albany, OR 97321 USA

Les fichiers documentaires

[Hpdoc] Ceux des outils de développement HP, en particulier les fichiers RPLMAN.DOC et SASM.DOC (en anglais).

Les ouvrages sur les mathématiques pour l'informatique:

Seymour Lipschutz Mathématiques pour informaticiens. Cours et problèmes 840 exercices résolus McGraw-Hill Série Schaum

Documentation sur support magnétique

Goodies Disc de Joseph Horn **Disquette PERF3** (toutes disponibles auprès de POLE)

Les journaux de liaison des clubs HP48

[HtePerf] Haute Performance **[PerfCalc]** Performance Calcul
[48Sxtant] 48 Sxtant

Les serveurs

36.15 RTEL **36.15 CALC**

Achevé d'imprimer en novembre 1996
par LOUIS JEAN - 05003 GAP
Dépôt légal : 801 Novembre 1996

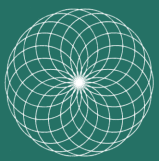
ASSEMBLEUR
sur
HP 48S/SX/G/GX

Christophe Nguyen et Guy Toublanc

La HP 48 est l'une des rares calculatrices permettant la programmation en «assembleur». Voici un guide conçu pour ceux qui veulent découvrir cette ressource de leur calculatrice sans y passer leurs nuits.

Après une exploration de l'environnement et des outils pratiques qui permettent le maniement de l'assembleur, les premiers principes de programmmation sont explicités au travers d'exemples concrets débouchant toujours sur des programmes réalisables par le lecteur. Déplacements de l'affichage, décryptage de code, hachurage de l'écran, et même animations comme le rebond d'une balle ou la simulation d'un train rétro, voilà les programmes qui sont prétextes à un apprentissage progressif et passionnant du langage assembleur.

La disquette PERF3 contenant les sources et les outils de ce guide, ainsi que 100 pages d'explications supplémentaires, est disponible en option auprès de l'éditeur.



POLE

I.S.B.N. 2-909737-14-4
40FF
Diffusion BELIN J016