LES *SECRETS* DE LA HP 48

TOME 1

J.-M. FERRARD

D3I DIFFUSION

930518

Pour Richard,
avec qui j'ai grand
plaisir à partager
tous ces "secrets".

J Michel FERRARI

Chez le même EDITEUR et du même AUTEUR, J.-Michel FERRARD :

- **La Maîtrise de la HP 28 S**
  TOME 1 : PROGRAMMATION ET EXERCICES

- **La Maîtrise de la HP 28 S**
  TOME 2 : PROGRAMMATION ET APPLICATIONS

- **La Maîtrise de la HP 48**
  TOME 1 : PROGRAMMATION ET EXERCICES

- **La Maîtrise de la HP 48**
  TOME 2 : PROGRAMMATION ET APPLICATIONS

- **Mastering your HP 48**
  PROGRAMMING AND APPLICATIONS
  (Version anglaise)

- **HP 48, NOCH EFFEKTIVER NUTZEN**
  ANWENDUNGSPROGRAMME FÜR FORTGESCHRITTENE
  (Version allemande)

D3I DIFFUSION
20 rue Hermès - 31526 Ramonville St Agne - France
Tél. : (33) 62 24 66 12 - Fax : (33) 62 19 03 77

LES *SECRETS* DE LA HP48

TOME 1

Jean-Michel FERRARD

Agrégé de Mathématiques
Professeur de Mathématiques Supérieures
au Lycée Déodat de Séverac, Toulouse

D3I
BP 49 - Tél. 62 24 66 20
31528 RAMONVILLE ST-AGNE CÉDEX (FRANCE)

Hewlett-Packard, HP 48, HP 48 S, HP 48 SX,
sont des marques déposées

Première édition : août 1992

© 1992 – D3I DIFFUSION
ISBN 2-908791-08-0

Tous les efforts ont été faits pour que les informations, programmes et schémas présentés dans ce livre soient aussi exacts et complets que possible. Ni les auteurs, ni l'éditeur ne pourront en aucun cas être tenus pour responsables des préjudices de quelque nature que ce soit, pouvant résulter de leur utilisation, tant dans un cadre privé, que commercial ou professionnel.

La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit, ou ayants cause, est illicite » (alinéa premier de l'article 40).

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal.

TABLE DES MATIERES DU TOME 1

DEUXIEME PARTIE: LES SYSEVALS

INTRODUCTION

Après les deux tomes de "*La Maîtrise de la HP48*", j'ai été atteint d'une maladie qui ne pardonne pas, et que connaissent malheureusement beaucoup trop d'utilisateurs des calculatrices HP: **La SYSEVALite aigüe** !

Les symptomes de ce mal sont terrifiants:

- Abandon des instructions standards de la machine au profit d'une seule commande (maudite): **SYSEVAL**.
- Recherche fébrile, au mépris de toute prudence, de ce que peuvent recéler les entrailles de la calculatrice.
- Vocabulaire réduit à quelques mots compréhensibles des seuls malheureux déchirés par ces tourments: *adresses, ROM cachée, quartets, RPL, D9D20, B2130, ...*
- Cauchemars où revient inlassablement, comme une malédiction, le message redouté "*Try to Recover Memory ?*"

Et pourtant la SYSEVALite est un mal délicieux, tout simplement parce que la HP48 recèle de véritables trésors, accessibles seulement à ceux qui veulent bien se donner la peine de les chercher:

- Ce sont des centaines d'instructions supplémentaires qui apparaissent, des dizaines d'effets spéciaux.
- C'est la possiblité de faire ce que vous voulez de votre calculatrice, d'épater les copains, de vous faire des frayeurs.
- C'est la satisfaction de comprendre (restons modestes tout de même) comment les ingénieurs de HP ont programmé cette machine géniale qu'est la HP48.
- C'est le plaisir bien connu de briser des tabous, d'arriver à faire avec votre machine ce que le constructeur a cru bon de vous défendre.

Bref, les satisfactions que procure ce mal valent bien quelques sacrifices ou les nuits blanches passées à chasser les SYSEVALs.

Mais voilà: L'épidémie se développe tous les jours un peu plus, fauchant toujours davantage d'éléments brillants de notre belle jeunesse. Le plus désolant est de constater le temps que passent les nouveaux malades à retrouver ce que leurs prédécesseurs ont pourtant trouvé avant eux.

J'ai pensé qu'il était de mon devoir d'enrayer la progression de la ***SYSEVALite***, la meilleure solution étant de livrer au public ce que j'ai pu obtenir, en un an de recherches personnelles et passionnées.

◆ Ce que sont "*les secrets de la HP48*":

Ce livre contient des milliers de SYSEVALs (c'est-à-dire de points d'entrées), inédits dans leur immense majorité.

Il est organisé en deux parties:

La première partie décrit les différents objets de la HP48 (même les plus mystérieux), et la façon dont ils sont représentés en mémoire. L'intérêt est évident pour tous ceux qui veulent comprendre comment la HP48 code et manipule les différents données qui lui sont utiles.

Dans cette partie, pour chacun des différents types d'objets, sont présentés tous les objets ayant ce type et qui figurent dans la ROM (mémoire morte) de la HP48.

Celà fait déjà plusieurs centaines de SYSEVALs que l'on pourraient qualifier de SYSEVALs "*inertes*", car dans leur grande majorité, ils permettent d'accéder à des objets qui ne sont que des données (listes, réels, entiers-système, chaînes de caractères, noms, etc....)

La deuxième partie du livre (de loin la plus fournie, et sans doute la plus intéressante) est consacrée aux SYSEVALs "*éxécutables*", c'est-à-dire qui désignent des procédures, des programmes.

On trouvera là des milliers d'adresses qui "*font quelque chose*". J'ai regroupé ces SYSEVALs par familles (ceux qui agissent sur la pile, ceux qui traitent du graphisme, etc...)

J'ai essayé de donner des explications suffisantes pour chacune de ces adresses. Chacune d'elle est associée à un mnémonique (qui dans les cas simples peut servir de commentaire du SYSEVAL).

J'ai mis un an à constituer cette énorme bibliothèque d'adresses. Pour celà, il m'a fallu forger des outils logiciels permettant d'étudier plus efficacement la ROM de la HP48.

En matière de SYSEVALs, et si l'on ne veut pas se décourager, on ne peut en effet partir "*à l'aveuglette*". Il faut absolument organiser sa recherche de manière rigoureuse et ordonnée, et cette phase est déjà passionnante.

La récompense vient ensuite, quand chaque nouvelle adresse élucidée conduit à plusieurs autres découvertes.

Au point où j'en suis, je pense avoir une connaissance très approfondie de la ROM de la HP48 (au point pratiquement d'être remonté au code "source" qu'ont du utiliser les ingénieurs de chez HP pour parvenir à ce code objet que les malheureux atteints de *SYSEVALite* essaient patiemment d'interpréter).

◆ Ce que sont les "*SYSEVALs*":

Tout d'abord une question de terminologie. J'appelerai indifféremment "*adresse*" ou "*SYSEVAL*" tout point d'entrée de la ROM (mémoire morte) de la HP48.

En un tel point d'entrée se trouve habituellement un objet qui peut être une donnée (une chaîne de caractères, par exemple) ou une procédure. "*Evaluer*" cet objet, c'est le déposer sur la pile (si c'est une donnée) ou l'éxécuter (si c'est une procédure).

Le meilleur moyen d'évaluer un tel objet est d'appliquer l'instruction **SYSEVAL** à l'adresse de cet objet (exprimée sous la forme d'un entier binaire).

Toutes les adresses figurant dans ce chapitre sont données en **hexadécimal** (base de numération comportant seize chiffres de 0 à 9, puis de A à F). C'est pourquoi je vous engage vivement, si ça n'est pas déjà fait, à sélectionner le mode "*hexa*" pour représenter les entiers binaires (actionnez la touche **HEX** dans le menu **MODES**)

Prenons trois exemples:

- L'instruction **#03FB3h SYSEVAL** place au niveau 1 de la pile l'entier-système <2D9Dh>
  Cet objet est d'un type que vous ne connaissez peut-être pas encore, mais qui est utilisé de manière essentielle par les routines de la HP48. Il fait partie des objets qui ne sont pas, a priori, accessibles à l'utilisateur.

- L'instruction **#4E2CFh SYSEVAL** efface le menu en cours (sans geler pour autant l'écran. Le menu réapparait quand le programme contenant cette instruction s'achève, à moins que ce programme ne se termine par l'instruction **4 FREEZE**)

- L'instruction **#4E347h SYSEVAL** affiche de nouveau le menu qui aurait été caché par l'instruction précédente.

- Essayez par exemple le programme suivant, qui fait clignoter le menu en cours, jusqu'à ce que vous appuyez sur une touche (et imaginez un programme qui fasse la même chose, mais sans utilisation de SYSEVALs):

```
« DO #4E2CFh SYSEVAL .1 WAIT
     #4E347h SYSEVAL .1 WAIT
  UNTIL KEY END DROP
»
```

◆ Avantages et inconvénients des "*SYSEVALs*":

L'utilisation des SYSEVALs offre de nombreux avantages:

- On accède aux formes internes des instructions standards de la HP48.
  Ces instructions sont en effet "protégées" par de nombreux tests sur la présence et la nature des arguments, la mémoire disponible, etc....
  Ces tests incessants pénalisent la vitesse d'éxécution des programmes. L'intérêt qu'il y a à les contourner est dès lors évident.

- Les SYSEVALs permettent d'utiliser des objets qui sont inaccessibles à l'utilisateur "*moyen*" (entiers-système, réels longs, etc....)

- De nombreux SYSEVALs constituent autant de nouvelles instructions pour votre calculatrice. Il existe par exemple une adresse permettant d'échanger deux colonnes d'un tableau. Utilisons-la sur un exemple.

  Le programme suivant transforme un réel en un entier-système. Appelons-le '**R→S**': **« #18CD7h SYSEVAL »**

  Le programme ci-dessous, baptisé '**SWAPC**', échange les colonnes de numéros m et n (réels) de la matrice M.

  Le schéma fonctionnel est: **3:M, 2:m, 1:n ⇒ 1:M'**

  ```
  « 1 2 START SWAP R→S NEXT
    #37500h SYSEVAL DROP2 »
  ```

  N.B: les deux programmes ci-dessus ne testent pas la bonne conformité des arguments. Soyez attentifs.

- Les SYSEVALs permettent des opérations à priori non permises par le constructeur. C'est un plaisir que de contourner ces interdictions...

Pour être honnête, voyons les inconvénients des SYSEVALs.

- Le risque d'erreur est important. le fait d'appliquer l'instruction SYSEVAL sur une adresse incorrecte peut avoir des effets imprévisibles. Au pire vous obtiendrez un "*Memory Clear*"...

- Le fait d'éxécuter un SYSEVAL avec des arguments absents ou incorrects peut entraîner les mêmes conséquences.

- Mais rassurez-vous: la machine en elle-même ne craint rien. Ma HP48, qui a subi des dizaines (voir des centaines) de "*Memory Clear*" sévères est là pour en témoigner.

◆ Les différentes versions de la HP48:

Hewlett-Packard a commercialisé jusqu'à aujourd'hui 5 versions de la HP48 (il ne s'agit pas des différences entre la 48S et la 48SX). Chacune d'elle est désignée par une lettre, de A à E.

Pour connaître la version de votre HP48, vous avez deux possibilités:

▸ Appuyez simultanément sur les touches **ON** et **D**, puis relachez les. Actionnez la touche ⇐ ("BackSpace"), puis la touche **EVAL**.
Vous devez voir apparaître un message du genre:

Version HP48-E
Copyright HP 1989

Revenez à la pile par un appui simultané sur les touches **ON** et **C**.
N.B: par cette méthode, vous perdez le contenu de la pile et celui de l'objet-graphique **PICT**.

▸ Tapez **#30794h SYSEVAL** (attention au mode hexadécimal!)
Vous obtenez au niveau 1 de la pile une chaîne du type: "**HPHP48-E**" (qui indique que la ROM porte le numéro de version E)

Fort heureusement, les différences entre les ROM successives sont très légères. C'est ce qui permet à ce livre d'exister: toutes les adresses que vous trouverez ici sont en effet communes aux différentes versions (j'ai tout de même dû renoncer à quelques dizaines de SYSEVALs)

◆ Mise en garde:

Ce livre contient des milliers d'adresses, en hexadécimal.
J'ai vérifié chacune d'elle, avec la plus grande attention, mais il m'est impossible de vous garantir qu'aucune erreur ne s'est glissée ici où là.
Je ne peux donc que vous recommander la plus grande attention, et vous conseiller d'être en mesure de sauvegarder le contenu de la mémoire centrale de votre calculatrice (carte RAM non fusionnée ou stockage, par la voie série, sur un micro-ordinateur)

D'autre part, l'utilisation que vous pourriez être amené à faire des SYSEVALs de ce livre est de votre seule et unique responsabilité. Ni mon éditeur ni moi-même ne pourrions être tenus pour responsables de l'usage (ou des conséquences de cet usage) des informations contenues dans ce livre.

RAPPELS ?

On se propose ici de rappeler quelques généralités sur la mémoire de la HP48, le microprocesseur, les adresses, les SYSEVALs, etc....

Ceux qui connaissent tout celà très bien passeront donc directement à la suite.

◆ LA MÉMOIRE:

Le coeur de votre HP48, c'est son **microprocesseur**. Il est du type "**Saturn**"; c'est un dérivé de celui qui a équipé initialement le HP71, puis la HP28. Il est cadencé à 2Mhz (pendant que nous parlons de vitesse, la HP28 pouvait être accélérée de manière logicielle; il n'en est plus de même pour la HP48, qui est "*Full Speed*")

Le microprocesseur éxécute les calculs élémentaires, et "*dialogue*" avec la mémoire qui l'environne:

- La mémoire morte, ou ROM (*Read Only Memory*). Elle n'est accesible qu'en lecture. Il peut s'agir de la ROM intégrée, ou de cartes protégées en écriture.
- La mémoire vive, ou RAM (*Random Access Memory*). On peut à la fois y lire et y écrire. Il peut s'agir de la mémoire vive intégrée (celle que vous utilisez pour stocker des objets, pour agir sur la pile, pour gérer des graphismes), ou du contenu d'une carte d'extension RAM non protégée en écriture.

Le "*Saturn*" est un microprocesseur "*4 Bits*", ce qui signifie que, pour lui, l'unité de lecture ou d'écriture est formée de la succession de 4 bits (chiffres égaux à 0 ou à 1).

Un telle succession de 4 bits est appelée un **quartet** (*Nibble* en anglais). Elle permet de coder tous les entiers de 0 à 15 = #Fh.

Le quartet **xyzt**, où x,y,z,t valent 0 ou 1, représente l'entier: n = 8x+4y+2z+t

La base de numération qui s'impose donc naturellement est la base hexadécimale.

Les quartets sont, dans la pratique, les "atomes" de la mémoire. Chacun d'eux est caractérisé par une certaine **adresse**. C'est cette adresse qui permet de localiser le quartet, pour le lire ou pour le modifier.

Tous les objets de la HP48 seront dès lors codés par une succession de quartets.

Deux quartets consécutifs forment un **octet** (*Byte* en anglais; 8 bits, qui permettent de représenter tous les entiers de 0 jusqu'à 2^8-1 = #FFh = 255).

Cinq quartets consécutifs forment un **mot** (*word*) ou encore une **adresse**.

Un "*mot*" est donc une succession de 20 bits et permet de coder tous les entiers de 0 à 2^20-1 = #FFFFFh = 1.048.575

RAPPELS ?

Le microprocesseur "Saturn" accède à la mémoire par un registre de 20 bits, c'est-à-dire de 5 quartets. La conséquence est qu'il peut "localiser" #FFFFFh adresses, de l'adresse #0h à l'adresse #FFFFFh (ce qui en fait 2^20).

A chacune de ces adresses se trouvent un quartet.

On a souvent l'habitude de mesurer l'étendue d'une mémoire, en informatique, en termes de "*kilo-octets*" (Ko en abrégé).

Un kilo-octet représente: 1024 octets (1024 = 2^10 = #400h).

De l'adresse #0h à l'adresse #FFFFFh, il y a 2^20 quartets et donc 2^19 octets.

La mémoire adressable par le "*Saturn*" est donc longue de 2^9 = 512 kilo-octets (ou encore, pour prendre une autre mesure courante, un demi méga-octet).

◆ LE PLAN DE LA MÉMOIRE:

La mémoire de la HP48, dont nous venons de mesurer l'étendue, est organisée en plusieurs zones distinctes:

- De l'adresse #0h à l'adresse #7FFFFh (256 Ko): C'est là que se trouve la mémoire morte intégrée (le système) de la HP48. Elle contient le codage de toutes les instructions de la HP48 (ainsi que de toutes les caractéristiques de fonctionnement de la machine). Tout celà donne un nombre hallucinant de sous programmes. Seule une partie d'entre eux (quelques milliers, sans plus) seront présentés dans ce livre.

- De l'adresse #70000h à l'adresse #7FFFFh (32Ko): C'est la mémoire vive, disponible pour l'utilisateur (mais une partie est néanmoins réservée pour le système de la HP48, pour y stocker les indicateurs décrivant les paramètres de fonctionnement de la calculatrice).

  On remarque que cette zone de mémoire est commune aux 32 derniers kilo-octets de la mémoire morte intégrée. La HP48 bascule de façon permanente, et sans que l'utilisateur s'en rende compte, d'un de ces deux bancs de 32Ko à l'autre (technique connue sous le nom de "*Bank Switching*").

  Quand on fait référence, au moyen de l'instruction SYSEVAL, à une adresse comprise dans cette zone, il s'agit toujours d'une adresse de la mémoire vive. C'est pourquoi on désigne souvent les 32 derniers Ko de la mémoire morte intégrée comme étant la **ROM cachée** (mais pour nous, elle acceptera bien vite de se dévoiler).

- De l'adresse #80000h à l'adresse #BFFFFh (128Ko): C'est la zone occupée par une carte d'extension dans le port 1 (si c'est une carte 128K, elle occupe toute cette zone; si c'est une carte 32K, elle en occupe le premier quart).

- De l'adresse #C0000h à l'adresse #FFFFFh (128Ko): C'est la zone occupée par une carte d'extension dans le port 2 (comme ci-dessus).

◆ LES SYSEVALS:

L'immense majorité des adresses qui apparaîtront dans ce livre sont relatives à la zone #00000h→#6FFFFh de la mémoire morte (toute la ROM donc, à l'exception de la ROM cachée, qui n'est pas accessible de façon immédiate par l'instruction SYSEVAL).

La syntaxe de l'instruction SYSEVAL est: ***adr*** **SYSEVAL**, où "*adr*" est un entier binaire désignant une adresse (dans l'intervalle #0h→#FFFFFh, mais de toute façon seuls les cinq derniers chiffres hexadécimaux de "*adr*" sont pris en compte).
L'entier binaire "*adr*" peut être exprimé dans une base quelconque de numération, mais la numération hexadécimale est recommandée (elle sera la règle dans ce livre).

La conséquence de cette instruction est l'évaluation (l'éxécution) du programme débutant à l'adresse "*adr*". Bien entendu, ce que la HP48 doit trouver à cette adresse doit être cohérent. L'éxécution de l'instruction SYSEVAL à une adresse choisie au hasard (ou à une adresse incorrecte) est imprévisible et peut déboucher sur une perte de données.

Il ne suffit pas que l'adresse utilisée soit correcte pour que l'instruction SYSEVAL fonctionne sans problème. Le programme ainsi éxécuté par son adresse peut requérir un certain nombre d'arguments (ceux-ci étant d'une certaine nature); il importe alors de respecter ces conditions...

Un exemple très très moyen:

Chacune des instructions standards de la HP48 est implantée quelque part dans la ROM intégrée, et est donc éxécutable au moyen d'un SYSEVAL.

Considérons le programme suivant: « **ROT * + »**
- L'adresse de l'instruction **ROT** est: **#1FC0Eh**
- L'adresse de l'instruction * est: **#1ADEEh**
- L'adresse de l'instruction + est: **#1AB67h**

Ce programme est donc équivalent à:
« #1FC0Eh SYSEVAL #1ADEEh SYSEVAL #1AB67h SYSEVAL »

Cet exemple donne une bien piêtre idée de ce que peuvent apporter les SYSEVALs. Le programme obtenu ici en les utilisant est plus long, il est beaucoup moins lisible, et il peut être une source d'erreurs ! Evidemment, tel n'est pas le rôle des SYSEVALs que de remplacer les instructions standards de la HP48...

Un exemple un peu meilleur:

Considérons le programme: **« 3 ROLLD SWAP ROT »**
Son rôle est d'échanger les objets placés aux niveaux 2 et 3 de la pile.
Il peut être remplacé par le programme: **« #60EE7h SYSEVAL »**
Un seul SYSEVAL peut donc remplacer plusieurs instructions standards...

Un bon exemple:

Imaginons un programme qui donne la liste de tous les noms figurant dans une expression donnée, la liste obtenue ne devant pas contenir de répétitions (même si un nom apparaît plusieurs fois dans l'expression initiale).

On doit par exemple obtenir:

'X+SIN(Y+X/Z)*Y' ═══> { X Y Z }

Ecrire un tel programme n'est pas chose facile !

Mais à quoi bon se creuser la cervelle sur ce problème si on sait que la séquence **#353ABh SYSEVAL** réalise exactement l'opération souhaitée ?

'X+SIN(Y+X/Z)*Y' ═══[#353ABh SYSEVAL]═══> { X Y Z }

Pour la petite histoire, sachez que ce SYSEVAL est utilisé par la HP48 pour produire le menu correct (à partir du contenu de la variable '**EQ**') quand on entre dans l'environnement **SOLVR**.

On voit sur cet exemple qu'il existe des SYSEVALs réalisant des opérations complexes, non implantées dans le langage standard; certaines de ces opérations restent programmables avec les instructions de base, mais au prix de bien des difficultés; d'autres SYSEVALs réalisent des opérations qu'il est impossible de programmer avec "*les moyens du bord*".

◆ LES DIFFÉRENTS NIVEAUX DE PROGRAMMATION:

Programmer notre HP48 c'est d'abord concevoir des séquences d'instructions qu'elle est capable de comprendre, puis c'est éxécuter ces séquences.

Le problème est que notre calculatrice, sans que nous nous en doutions, comprend plusieurs langues. Elles les connaît parfaitement bien: tout le problème vient de nous, qui maîtrisons à peine le langage "*usuel*".

On peut distinguer plusieurs niveaux de programmation, rangées dans un ordre de difficulté décroissante.

▸ Le langage "machine":

Le microprocesseur de la HP48 n'est jamais au repos. Il possède un registre (nommé **PC** comme "*Program Counter*") d'une taille de 20 bits et qui désigne en permanence une adresse.

A cette adresse figure un quartet (ou débute une succession de quartets) que le microprocesseur interprète comme une instruction. Il l'éxécute puis modifie le contenu du registre PC, pour lire et éxécuter les instructions suivantes...

Chacune de ces "*instructions*" est codée par un quartet ou par une succession de quartets. Il en existe un très grand nombre, et les retenir est impossible. On peut cependant imaginer, dans l'absolu, un programme écrit de cette manière.

▸ L'assembleur:

Le "langage machine" permet d'aboutir aux algorithmes les plus rapides. Il a contre lui de nécessiter l'apprentissage de dizaines de codes obscurs; les programmes ainsi créés sont plutôt illisibles et on peut difficilement les "débugguer".
On est donc amené à imaginer une "*interface*" entre ce langage ultra-rapide et notre souci de produire des programmes "*lisibles*". La solution réside dans la notion d'assembleur.

Un **assembleur** est un programme qui permet tout d'abord de représenter chacune des instructions du langage-machine par un code lisible appelé "***Mnémonique***".
Comme son nom l'indique un mnémonique facilite la mémorisation des instructions du microprocesseur. Il est en général formé par une abréviation indiquant assez clairement la nature de cette instruction. Il peut exister plusieurs assembleurs différents, mais tous respectent des conventions dictées par le constructeur.
Un assembleur ne se contente pas de vous fournir les mnémoniques: il permet surtout de traduire le texte que vous avez écrit en les utilisant (et qu'on appelle le "***code source***"), en un programme écrit en langage machine, et donc directement éxécutable par le microprocesseur (ce programme est appelé le "***code objet***").

Le passage du code source au code objet est appelé l'**assemblage**. Si l'assembleur que vous utilisez est performant, il vous indiquera les erreurs de syntaxe que vous auriez pu commettre dans l'écriture du code source. Celui-ci étant relativement lisible, il est assez facile de le corriger.

Certaines instructions du microprocesseur sont des instructions de *saut* à une adresse donnée (calculée souvent de manière relative par rapport à l'adresse en cours). Le calcul de ces déplacements s'avère souvent pénible. Tout assembleur qui se respecte doit donc vous permettre de définir des "labels" , c'est-à-dire de nommer des points précis du corps de votre programme, de manière à simplifier l'écriture de ces instructions de *branchement*.

Inversement, l'opération de "*désassemblage*" consiste à traduire un programme en langage-machine vers le programme équivalent et utilisant les mnémoniques conventionnels (il s'agit donc d'un retour du code objet au code source).
L'intérêt d'un "*désassembleur*" est de vous permettre d'interpréter, par exemple, les routines écrites en langage-machine, et qui apparaissent tout au long de la ROM de la HP48.

On ne doit cependant pas sous-estimer la difficulté d'écrire des programmes en "*assembleur*". Outre la nécessité de connaître tous les mnémoniques, un grand nombre d'instructions sont souvent nécessaires à la réalisation de programmes relativement simples (tant chaque instruction est "élémentaire").

RAPPELS ?

▸ La programmation par "Externals":

On appelera "External" tout point d'entrée non officiel de la HP48 (c'est-à-dire qui ne soit pas l'adresse d'une instruction standard).
L'intérêt des "Externals" est qu'il désignent, le plus souvent, des routines "toutes prêtes". L'appel à ces programmes vous dispense dont totalement de taches fastidieuses, en même temps qu'il permet d'augmenter considérablement le langage de la HP48.

Il faut comprendre en effet que la mémoire morte de la HP48 contient un nombre fantastique de sous-programmes, dont seule un infime minorité est disponible pour l'utilisateur de base, sous la forme des instructions standards.
Les programmeurs qui ont réalisé la ROM de la HP48 l'ont en effet parsemée d'utilitaires communs à plusieurs instructions standards, dans un souci de produire un code compact et efficace (l'étude de ce travail est une leçon de programmation irremplaçable).
Ce livre vous permettra d'accéder à un très grand nombre de ces utilitaires. Les programmes que vous écrirez n'en seront que plus courts et plus rapides...

L'inconvénient des "*Externals*" est qu'ils vous obligent à agir avec beaucoup de prudence. Autant les instructions standards de la HP48 son programmées de manière à prévenir toute erreur de l'utilisateur, autant les "*Externals*" sont dépourvus de tels garde-fous. Il sera donc essentiel de les utiliser à bon escient, en particulier en plaçant sur la pile les arguments qu'ils requièrent.

Pour comprendre comment on peut programmer en "*Externals*", il faut savoir que n'importe quel programme écrit dans le langage usuel de la HP48, est en fait codé comme une succession d'adresses (au moins pour ce qui concerne les instructions standards, et pour les entiers de -9 à 9).

Le programme « **3 ROLLD SWAP ROT** », évoqué plus haut, est codé de la manière suivante:

D9D20 L'adresse #02D9Dh désigne le début d'un programme
E1632 L'adresse #2361Eh est celle de l'instruction «
3F2A2 L'adresse #2A2F3h est celle de l'entier 3
0DCF1 L'adresse #1FCD0h est celle de ROLLD
DBBF1 L'adresse #1FBBDh est celle de SWAP
E0CF1 L'adresse #1FC0Eh est celle de ROT
93632 L'adresse #23639h est celle de l'instruction »
B2130

Programmer en "Externals", c'est être en mesure de réaliser de telles constructions, mais en utilisant des adresses non standards. Il faut pour celà des outils (car le langage usuel de la HP48 ne le permet pas).
Il faut savoir enfin qu'un programme écrit en external ne peut être édité en ligne de commande; on peut juste l'y visualiser mais ce n'est souvent pas clair...

▸ La programmation standard:

Il n'y a pas grand chose à en dire, sinon qu'elle vous assure une sécurité maximum, mais que vous payez cette tranquillité par une vitesse d'éxécution parfois un peu *bridée*, et que votre curiosité doit se satisfaire des instructions de "*Monsieur tout le monde*".

C'est à vous de voir si vous vous contentez de ce qui est autorisé, ou si votre goût de l'aventure est suffisamment fort pour vous entraîner sur les rivages luxuriants du monde des SYSEVALs (mais attention aux récifs...)

◆ LIRE ET ÉCRIRE A L'ENVERS:

Vous l'avez sans doute remarqué sur le programme utilisé précédemment comme exemple: toutes les adresses figurant dans ce programme étaient écrites "*à l'envers*".

Il s'agit en fait d'une situation constante que nous rencontrerons bien des fois. La structure du microprocesseur "Saturn" est telle qu'il écrit et lit, dans la mémoire, toute succession de quartets dans l'ordre inverse de l'ordre naturel.
Heureusement, celà ne va pas jusqu'à intervertir les 4 bits d'un même quartet.

Deux exemples permettent d'observer cette propriété essentielle.

▸ Les caractères:

Chaque caractère est codé sur un octet (8 bits) au moyen d'un code compris entre **0** et **255**. Jusqu'au code **127=#7Fh**, il s'agit de la correspondance classique connue sous le nom de **table ASCII**.
La caractère **Z** est ainsi représenté par le code **90=#5Ah**.
En mémoire, le caractère **A** sera codé de la manière suivante: **FB920 A5**
L'adresse **#029BFh** indique que l'octet suivant doit être interprété comme étant un "*Character*", et cet octet est écrit "*à l'envers*".

▸ Les entiers binaires:

Un entier binaire tel que **#123456789ABCDEF0** est codé de la façon suivante:
E4A20 51000 0FEDCBA987654321
Dans cette représentation, l'adresse **#02A4Eh** signale au micorprocesseur la présence d'un entier binaire et se charge d'interpréter les 21 quartets qui suivent. Le mot **#00015h** représente ici la longueur totale de l'objet (**#15h=21** quartets), à l'exception de l'adresse initiale **#02A4Eh**.
On voit bien, sur cet exemple, comment la HP48 renverse l'ordre des quartets, en fonction de la longueur de l'information à écrire ou à lire (les deux premières adresses sont inversées sur 5 quartets; le corps de l'entier binaire est inversé sur 16 quartets)

◆ EN CAS DE PROBLEME:

Si vous vous lancez sans retenue dans l'utilisation des SYSEVALs, vous risquez fort de vous trouver un jour ou l'autre devant le message "*Try to Recover Memory ?*". Il suffit pour celà de mal saisir une adresse, ou de ne pas respecter la présence et/ou la nature des arguements nécessités par ce SYSEVAL.

Rassurez-vous tout de même, la plupart des SYSEVALs erronés ne se terminent pas de cette manière. Il y a deux réactions courantes de la HP48 dans un tel cas:

- Ou bien la calculatrice entre dans une boucle infinie (l'appui sur la touche **ON** n'a aucun effet. Le mieux est ici d'éxécuter un "*reset à chaud*" par un appui simultané sur les deux touches **ON** et **C**.
  L'effet d'un tel redémarrage est de perdre le contenu de **PICT**, celui de la pile, puis d'afficher le menu **MTH** comme menu courant. Les flags ne sont pas affectés (vous restez par exemple en mode **HEX**, si vous y étiez au départ).

- Ou bien la HP48 éxécute elle même un tel redémarrage à chaud. Les conséquences ne sont pas bien graves, comme vous le constatez.

Revenons au message "*Try to Recover Memory ?*". Il vous est proposé de répondre par oui ou par non (le menu se réduit à deux alternatives, **YES** et **NO**).

- Si vous répondez non, on ne vous donne pas de deuxième chance. La calculatrice est complètement réinitialisée. Tous les flags reprennent leur valeur par défaut (en particulier l'écriture des entiers binaires revient au mode décimal).

- Si vous répondez oui, la calculatrice tente de reconstituer un maximum de données cohérentes (il est clair que si votre malencontreux SYSEVAL a nettoyé la mémoire aux endroits où sont codés vos répertoires, vous ne récupérerez pas grand-chose...)
  Il se peut que vous retrouviez vos données (même dans ce cas, les flags sont rétablis à leur valeur par défaut).
  Il se peut que vous ne retrouviez qu'une partie de vos variables, ou que la HP48 ait renommé vos répertoires avec des noms bizarres (**D.01**, **D.02**, etc...).
  Il se peut même que la séquence de "récupération" de la mémoire débouche en fait sur le message point trop agréable "*Memory Clear*".

Il faut envisager encore d'autres possibilités (c'est du vécu, ma HP48 a connu tous ces déboires bien souvent, mais apparemment elle ne m'en veut pas).
Il se peut par exemple que le clavier ne réponde plus du tout (et qu'il devienne impossible d'effecteur un "*reset*" par **ON-C** ou même de lancer le SOS qu'est le triple appui sur les touches **ON-A-F**.

Dans ce cas, plutôt que vous rabattre sur une calculatrice d'une autre marque (pouah! quelle déchéance...) vous avez deux possibilités:

- Retournez votre HP48 et sur la face arrière, retirer le patin de caoutchouc situé en haut à gauche. Vous deviez voir apparaître un petit **R**, et un orifice permettant le passage d'un objet effilé. Introduisez par exemple l'extrémité d'un trombone et observez en même temps votre écran.
  Quand la pointe atteint un contact prévu à cet effet, l'écran s'éteint. Il est prudent de ne pas trop insister. Rallumer alors votre machine par **ON**. Le prochain appui sur une touche devrait éxécuter un *reset*.
  Cette méthode s'applique également si votre écran affiche des choses bizarres (il m'est arrivé plusieurs fois que ma HP48 affiche deux fois les niveaux 1 et 2 de la pile, ce qui donne l'impression de deux écrans de HP28. Rien de grave, même si c'est impressionnant...)
  Après toutes ces émotions, il est fort possible que vous retrouviez l'intégralité de vos données (Inch'Allah).

- Pour les cas désespérés, retirez les piles, et attendez quelques minutes. Ne vous faites cependant aucune illusion sur vos chers programmes.

N.B: Si vous avez une HP48SX, et une carte RAM (de 32K ou de 128K), il est prudent de libérer la mémoire correspondante (instruction **FREE**) et d'archiver le contenu du répertoire **HOME** sur cette carte (instruction **ARCHIVE**), avant de protéger cette carte en écriture. De cette manière vous pourrez restorer le contenu de **HOME** depuis cette carte, en cas de problème (instruction **RESTORE**).

◆ ET MAINTENANT ?:

Si ce qui précède ne vous a pas découragé (mais il vous en faudrait beuacoup plus, vous pouvez vous lancer dans l'aventure des SYSEVALs.

Ce tome 1 est divisé en deux parties:

- Dans la première partie, on décrit les différents objets reconnus par la HP48 (même les plus exotiques), et on voit lesquels de ces objets figurent dans la ROM.

- Dans la seconde partie, on décrit les SYSEVALs éxécutables (c'est-à-dire qui renvoient à un sous-programme plutôt qu'à un objet inerte).

Je vous souhaite de très bons SYSEVALs...

PREMIERE PARTIE

LES OBJETS DE LA HP48

Plût au ciel que le lecteur, enhardi et devenu momentanément féroce comme ce qu'il lit, trouve, sans se désorienter, son chemin abrupt et sauvage, à travers les marécages désolés de ces pages sombres et pleines de poison; car, à moins qu'il n'apporte dans sa lecture une logique rigoureuse et une tension d'esprit égale au moins à sa défiance, les émanations mortelles de ce livre imbiberont son âme comme l'eau le sucre. Il n'est pas bon que tout le monde lise les pages qui vont suivre: quelques-uns seuls savoureront ce fruit amer sans danger. Par conséquent, âme timide, avant de pénétrer plus loin dans de pareilles landes inexplorées, dirige tes talons en arrière et non en avant.

Isidore Ducasse, Comte de Lautréamont,
LES CHANTS DE MALDOROR

◆ LA NOTION D'OBJET:

On le sait, le langage de la HP48 est articulé autour de la notion d'objet. Pour peu que l'on ait une certaine pratique de la programmation sur cette calculatrice, c'est un concept qui devient vite familier.

On sait qu'il existe différents types d'objets: des réels, des tableaux, des listes, etc...., mais on ne se préoccupe pas de chercher une définition globale.

Pourtant, si l'on étudie de près le contenu de la ROM de la HP48, ou bien la façon dont sont codés les programmes que nous écrivons, un certain nombre de principes unificateurs apparaissent vite.

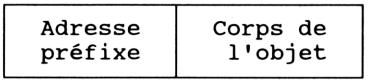
On se rend ainsi compte que la notion d'objet, pour vague qu'elle soit, permet une approche formalisée simple et efficace du langage de la calculatrice.

Si l'on veut se risquer à donner des définitions générales des "*objets*", on doit envisager deux aspects: la façon dont on les utilise "*couramment*" (comment ils sont affichés, comment ils réagissent aux différentes instructions, comment ils interagissent entre eux), et la façon dont ils sont représentés en mémoire (leur "*codage*").

Heureusement, il se trouve que ces deux points de vue sont très proches:

- Du point de vue de l'utilisation habituelle de la HP48, on peut envisager la définition suivante: un objet est défini par un **type**, et, dans ce type, par un **contenu** (une **valeur**).
  Ainsi les objets 123.45678 et 1789 sont tous deux de type réel, et ce sont leurs contenus qui les différencient (c'est banal, je sais...)

- Du point de vue du codage en mémoire, un "*objet*" est le couple formé par une adresse, dite "***adresse-préfixe***" (on dit aussi "adresse-prologue"; elle est codée sur 5 quartets) et par le "***corps***" de l'objet (celui-ci suivant immédiatement l'adresse-préfixe en mémoire).



→ Sens des adresses croissantes →

Il y a autant d'adresses-préfixes différentes que de types d'objets. La première utilité d'une adresse-préfixe est donc celle d'**identificateur de type**.

L'adresse-préfixe et le corps de l'objet doivent être interprétés en liaison avec la notion d'évaluation. Quand la HP48 "*rencontre*" un tel objet, elle l'**évalue**. La charge de cette évaluation revient au programme situé à l'adresse désignée par cette adresse-préfixe.

On voit donc que les adresses-préfixes ont un deuxième rôle essentiel: celui d'**interpréter** et d'**évaluer** le corps de l'objet qu'elles précèdent.

◆ LES DIFFÉRENTS TYPES D'OBJETS:

Voici toutes les catégories d'objets reconnues par la HP48.

Le tableau ci-dessous contient, pour chacun des différents types, son nom, l'adresse-préfixe qui lui correspond, et le résultat renvoyé par l'instruction **TYPE** si on l'applique à un objet de cette catégorie.

Nom du type	Adresse	type
Nombre réel	#02933h	0
Nombre complexe	#02977h	1
Chaîne de caractères	#02A2Ch	2
Tableau réel	#029E8h	3
Tableau complexe	#029E8h	4
Liste	#02A74h	5
Nom global	#02E48h	6
Nom local	#02E6Dh	7
Programme	#02D9Dh	8
Expressions	#02AB8h	9
Entier binaire	#02A4Eh	10
Objet-graphique	#02B1Eh	11
Objet-taggué	#02AFCh	12
Objet-unité	#02ADAh	13
Nom XLIB	#02E92h	14
Répertoire	#02A96h	15
Librairie	#02B40h	16
Objet-Backup	#02B62h	17
Fonction intégrée	#02D9Dh	18
Commande intégrée	#02D9Dh	19
Entier-système	#02911h	20
Réel long	#02955h	21
Complexe long	#0299Dh	22
Linked Array	#02A0Ah	23
Character	#029BFh	24
Code Object	#02DCCh	25
Library data	#02B88h	26
External	.....	27

Le tableau précédent appelle certaines remarques:

- Les objets "*Programmes*", "*Fonction intégrée*" et "*Procédure intégrée*" correspondent à la même adresse-préfixe. Il s'agit en fait, de manière interne, du même type d'objet.
  Ce n'est que l'instruction **TYPE** qui établit une différence en vérifiant si le programme placé au niveau 1 est ou n'est pas le corps d'une des instructions intégrées de la HP48.

- Il en est de même des types "*tableau réel*" et "*tableau complexe*", qui ne sont différenciés que par l'instruction **TYPE**.
  En fait, en mémoire, il n'y a qu'un seul type "*tableau*".

- Le tableau précédent est divisé en deux parties. La première partie décrit les objets accessibles à l'utilisateur en utilisation normale.
  Seul, dans cette catégorie, le type "*Nom **XLIB***" est relativement mystérieux. Il sera décrit en détail dans ce livre, mais disons simplement que les noms **XLIB** sont au librairies ce que les noms de variables sont aux répertoires.

- Dans la deuxième partie du tableau précédent, on trouve les objets qu'utilise la calculatrice pour son usage "*personnel*", c'est-à-dire pour traiter de manière interne les instructions que vous lui communiquez.
  En principe, vous ne verrez jamais apparaître de tels objets si vous continuez à utiliser votre calculatrice comme "*Monsieur tout le monde*".
  Mais vous n'êtes pas de cette eau là, bien sûr...

- Les "*Externals*" n'ont aucune adresse-préfixe, pour la simple et bonne raison qu'il ne leur en correspond aucune de façon constante.
  La HP48 semble désigner sous le terme générique d'"*External*" tout objet ne débutant pas par une adresse-préfixe répertoriée.
  Dans la pratique, cependant, les Externals sont des "*Primitive Code Objects*" espèce méconnue mais très courante dans la ROM de la HP48, et dont nous reparlerons un peu plus loin.

Les différents objets de la HP48 peuvent être regroupés en plusieurs catégories distinctes.

- Les objets de type "*donnée*": l'évaluation de ces objets a pour seul effet de les placer sur la pile.
  Par évaluation, on doit comprendre l'évaluation réalisée par la HP48 quand elle rencontre un tel objet dans un programme (et pas l'évaluation obtenue par la touche **EVAL**, ce qui fait une différence pour les listes, par exemple).
  La plupart des objets sont de type "*donnée*": nombres, tableaux, listes, expressions, etc....

- Les objets de type "*exécutable*" dans lesquels peut passer le flot d'instructions.
  On y trouve les structures Rpl, les "*Code Objects*" et les "*Primitive Code Objects*" (voir ci-après).

- Les objets de type "*identificateur*":
  Ce sont les noms globaux et locaux, ainsi que les noms **XLIB**.
  Ils servent à référencer un objet (sauf pour les noms globaux quand ils ne désignent aucune variable), et l'évaluation de ces noms signifie l'évaluation de l'objet qu'ils référencent.

Il y a un autre critère très important qui permet d'opérer des regroupements entre types différents. On peut en effet considérer les objets "***atomes***" et les objets "***composés***".

- Les objets "*atomes*" sont ceux qui ne sont pas dissociables en sous-objets distincts. C'est le cas le plus général. On comprend bien que les réels, par exemples, soient de ce type.
  Il en est de même pour les tableaux, ce qui peut surprendre à priori. En effet, de façon interne, les différents coefficients qui forment un tableau n'apparaissent pas de façon autonome, complète. Seuls les contenus de ces coefficients sont rangés ensemble, séparés de leur adresse-préfixe (qui n'apparaît qu'une seule fois au début du codage du tableau, ce qui se comprend: cette adresse-préfixe identifie une fois pour toutes le type de tous les coefficients).

- Les objets "*composés*" qui sont par nature des groupements d'objets quelconques. On trouve dans cette catégorie les listes, les expressions algébriques, les "structures Rpl".

◆ LES "STRUCTURES RPL":

L'adresse préfixe #02D9Dh débute les objets de type "*programme*" (au sens où nous l'entendons habituellement, les programmes sont des séquences d'instructions encadrées par les délimiteurs « et »).

Cette adresse-préfixe correspond aussi aux "*fonctions intégrées*" et aux "*commandes intégrées*".

En fait ces objets sont des cas particuliers d'une structure utilisée intensivement, de façon interne, par la HP48. C'est dans cette structure qu'apparaissent de façon particulièrement marquée les caractéristiques du "*système d'exploitation*" de la HP48, basé sur la **notation polonaise inversée**, et sur l'évaluation directe et indirecte de séquences d'objets quelconques.

Pour cette raison, et parce que le terme "programme" pouvait prêter à confusion, j'ai désigné, tout au long de ce livre, ces structures sous le nom de "***structures RPL***" (ou plus simplement encore, par l'abréviation **Rpl** d'après les seules initiales de ***Reverse Polish Language***).

Dans la mémoire de la calculatrice, les "*Structures RPL*" sont codées de la manière suivante:

D9D20	Adresse-préfixe #02D9Dh
Elément_1	Première "instruction"
Elément_2	Deuxième "instruction"
.....	.....
Elément_n	Dernière "instruction"
B2130	Adresse-suffixe #0312Bh

Une structure Rpl débute donc par l'adresse-préfixe #02D9Dh qui renvoie à un programme dont le but est précisément d'assurer l'évaluation de cette structure.

Elle se termine par une "*adresse suffixe*" (#0312Bh) dont le rôle est double: marquer la fin de la structure Rpl et assurer le retour au programme "*appelant*" (s'il y en a un) ou au mode d'éxécution directe de la HP48 (où celle-ci scrute le clavier en attendant votre bon vouloir).

Les éléments qui composent une structure Rpl peuvent être de deux types différents:

- Des objets quelconques: Ils sont formés, comme on l'a vu, d'une adresse-préfixe (chargée d'évaluer l'objet, et de transmettre, en principe, le flambeau à l'instruction suivante).
  Evaluer un tel objet, c'est éxécuter le programme désigné par l'adresse-préfixe.
  Dans ce cas (évaluation d'un objet présent "*physiquement*" dans la structure Rpl), on parle d'**évaluation directe**.

- Des pointeurs, c'est-à-dire des adresses. Ces adresses désignent des objets de la HP48 (en mémoire morte, en général). Evaluer un pointeur, c'est évaluer l'objet qu'il désigne.
  Dans ce cas (évaluation d'un objet désigné par son adresse) on parle d'**évaluation indirecte**.

Très souvent, les objets ainsi évalués (de façon directe ou indirecte) tout au long de l'éxécution d'une structure Rpl, sont eux-mêmes des structures Rpl.

Pour assurer la cohérence de ce système, la HP48 gère un pointeur (pour désigner l'adresse où se situe la prochaine instruction à éxécuter), et une pile (distincte de la pile-utilisateur) où stocker les adresses de retours (dans le cas d'appels répétés à des structures Rpl imbriquées les unes dans les autres).

Sans entrer dans le détail, disons simplement que le registre **D0** du microprocesseur pointe en permanence sur l'adresse où se trouve la prochaine instruction a éxécuter (cette instruction est de toute façon représentée par un pointeur, qu'il s'agisse de désigner indirectement un objet à évaluer, ou qu'il s'agisse de l'adresse-préfixe débutant un objet présent physiquement dans la structure Rpl).

De même, c'est le registre B du "*Saturn*" qui se voit confier le rôle de pointer sur le sommet de la pile des retours de sous-programmes Rpl.

L'évaluation d'une structure Rpl ne se passe pas dans la monotonie (une instruction après l'autre, gentiment).

Tout comme dans le langage de programmation qui nous est accessible, il existe (de façon presque souterraine) un langage de programmation extrêmemnent puissant et varié.

En particulier, un grand nombre de pointeurs désignent des instructions permettant d'agir sur le déroulement de la structure Rpl qui les contient (schémas répétitifs, conditionnels, etc...).

Un des outils essentiels utilisés par la HP48 est la possibilité de retarder l'éxécution d'une instruction, en plaçant l'objet désigné sur la pile utilisateur, sans l'évaluer.

Pour qu'une telle instruction soit possible, et notamment quand il s'agit de l'évaluation directe d'un objet réellement présent dans la structure Rpl, il faut que la taille de cet objet soit connue (ne serait-ce que pour accéder à l'objet suivant dans le Rpl en cours).

On se trouvera alors en présence de plusieurs situations:

- Ou bien le type de l'objet est tel que sa taille soit toujours la même. C'est le cas par exemple pour les nombres réels: un réel est toujours codé sur 16 quartets (sans compter l'adresse-préfixe **#02933h**).
  La routine éxécutée par l'adresse-préfixe contient alors une instruction d'incrémentation du registre **D0**, lui permettant de pointer sur le début de l'objet suivant.

- Ou bien l'objet considéré est d'un type tel que la longueur de l'objet soit variable, comme les tableaux, les chaînes de caractères, les noms, etc...
  Dans ce cas, le corps de l'objet lui même contiendra une indication permettant d'en connaître la longueur. Cette indication suit d'ailleurs en général immédiatement l'adresse-préfixe (et est lue par cette dernière pour incrémenter correctement le registre D0).

- Ou bien l'objet considéré est un objet-composé (dans la pratique, il s'agira d'une structure Rpl, d'une liste, ou d'une expression).
  Dans ce cas, l'objet étant-lui même une succession ordonnée d'objets et/ou de pointeurs, la longueur de chacun de ses composants est connue ou calculable (comme il vient d'être dit) et la fin de la structure composée est déterminée par la présence d'un pointeur particulier (adresse-suffixe) qui est toujours l'adresse **#0312Bh**.

La ROM de la HP48 contient un nombre énorme de structures RPL (à commencer par celles qui représentent les instructions standards de la calculatrice).
Chacune de ces structures est généralement constituée d'une succession d'adresses, qui renvoient à des objets de type "*données*" (des noms, des chaînes, des réels, des entiers-système, etc...), à de nouvelles structures Rpl (c'est très courant), ou à des "*objets*" du type "*Primitive code*" (voir plus loin).

L'utilisation imbriquée de structures Rpl (chacune d'elles en appelant d'autres, uniquement par la donnée de leurs adresses) donne à la ROM une programmation très structurée (ce qui facilite la tâche à celui qui cherche à la décrypter).

◆ LES "Primitive Code Objects":

A un moment ou à un autre, dans les structures Rpl, il faut bien qu'apparaissent ici ou là des pointeurs vers des séquences d'instructions en langage-machine (le seul langage que comprenne réellement le micro-processeur).

Ces objets sont appelés "*Primitive Code Objects*". Ils ne sont pourtant pas des objets au plein sens du terme, dans la mesure où leur longueur n'est pas déterminée à l'avance, ni calculable à priori.

De tels objets ne peuvent donc qu'être éxécutés de manière indirecte (par la donnée de leur adresse).

Il est impossible de les faire figurer tels quels dans une structure Rpl (d'ailleurs leur définition exige qu'ils soient placés à une adresse bien précise de la mémoire: ces objets ne sont pas "*relogeables*").

Les "*Primitive Code Objects*" sont très répandus dans la ROM de la HP48. Ils sont formés d'une adresse-préfixe puis d'une séquence écrite en langage machine, l'adresse-préfixe pointant sur le début de cette séquence. Cette définition n'est pas d'une clarté aveuglante, et un exemple n'est pas inutile:

Imaginons qu'à l'adresse **#1EF81h** de la ROM de la HP48, on trouve un tel objet.

Son contenu pourrait être, par exemple, **68FE1 1C4141142164808C**, ce qui correspondrait au code suivant:

Adresse	Contenu	Mnémonique
#1EF81h	**68FE1**	
#1EF86h	**1C4**	**D1=D1-5**
#1EF89h	**141**	**DAT1=A A**
#1EF8Ch	**142**	**A=DAT0 A**
#1EF8Fh	**164**	**D0=D0+5**
#1EF92h	**808C**	**PC=(A)**

Evaluer cet objet, c'est charger le compteur de programme **PC** du microprocesseur avec l'adresse placée au début de l'objet, c'est-à-dire l'adresse **#1EF86h**. La séquence d'instructions débutant à cette adresse est éxécutée.

On sait que le registre **D0** contient l'adresse où trouver un pointeur indiquant la prochaîne instruction à évaluer.

Le retour au programme "appelant" (pour en poursuivre l'éxécution) s'effectue de la manière suivante:

- On place dans le registre **A** du microprocesseur l'adresse se trouvant à l'endroit pointé par **D0** (instruction **A=DAT0 A**)
- On incrémente d'une adresse (5 quartets) le contenu du registre **D0** (qui pointe donc sur la prochaine adresse à évaluer).
- On charge le compteur de programme **PC** avec l'adresse se trouvant à l'endroit pointé par **A**.

Un dernier mot sur les "*Primitive Code Object*". Toutes les adresses-préfixes correspondant aux différents objets reconnus par la HP48 pointent sur des "*Primitive Code Objects*".

◆ "RPL SYSTEME" ET "RPL UTILISATEUR":

Le langage de programmation que nous employons couramment avec la HP48 est en fait un sous-ensemble du langage qu'utilise la calculatrice pour traiter les instructions que nous lui soumettons.

Si le "*langage-utilisateur*" nous apparaît plus simple (alors qu'il n'utilise réellement que des adresses en mémoire morte), c'est parce que la HP48 traduit, à l'écran, ces adresses par des mots.

Si nous plaçons sur la pile un programme contenant l'instruction **DUP**, le mot **DUP** ne fait pas partie du programme, mais de l'image que la HP48 renvoie de celui-ci. C'est en fait l'adresse où se trouve implantée l'instruction **DUP** qui fait réellement partie du programme...

Un dernier point. Vous avez vu que la HP48 reconnait et utilise un certain nombre d'objets qui ne sont pas accessibles dans l'utilisation normale.

Elle se sert par exemple des réels et des complexes longs pour accroître la précision des calculs intermédiaires, de façon à assurer la précision du résultat final.

Elle utilise les entiers-système (de façon intensive) pour disposer, par exemple, d'une arithmétique rapide sur les nombres entiers.

Pendant que la HP48 éxécute une instruction standard, elle emploie (et en particulier elle place sur la pile) un grand nombre de ces objets mystérieux. Comment se fait-il qu'aucun d'eux ne reste sur la pile, si l'instruction est interrompue par une erreur ? L'explication est la suivante:

- Au moment de démarrer, chaque instruction standard "*marque*" le contenu de la pile, et les arguments qu'elle nécessite (après avoir vérifié la présence et la nature de ceux-ci).
- Pendant l'éxécution de l'instruction, le contenu de la pile au dessus du point "*marqué*" (c'est-à-dire au dessus des arguments utilisés au départ) n'est pas modifié. Par contre, le déroulement de l'instruction s'accompagne de mouvements d'objets quelconques (en particulier d'objets de type > 19) sur la pile.
- Si une erreur survient, la pile est nettoyée des éléments en deçà du niveau "*marqué*". Les arguments utilisés initialement sont alors rétablis sur la pile si la fonction **LASTARG** est active. C'est seulement à ce moment là que le message d'erreur est généré.

Je vous invite maintenant à prendre connaissance de tous les objets reconnus par la HP48, même les plus étranges. Suivez le guide...

NOMBRES RÉELS

Sur la HP48, la valeur absolue d'un réel non nul est comprise entre:
MINR = 10^-499 et **MAXR** = 9.99999999999*10^499

Un nombre réel peut être affiché au format *standard*, *scientifique*, "*ingénieur*", ou en *format fixe*. Il est cependant toujours représenté sous forme scientifique $\mathbf{r=\epsilon*m\hat{}\alpha}$ en mémoire interne, où, dans l'ordre des adresses croissantes:

- α est l'exposant codé sur trois octets successifs **XYZ**, en mode **BCD** ("binaire codé décimal") ce qui signifie que le contenu des trois quartets X,Y,Z est compris entre 0 et 9.
  Dans le calcul de l'exposant, on doit tenir compte du fait que la HP48 "*retourne*" les adresses.

  Le signe de l'exposant est codé dans le quartet Z:
  Si Z est compris entre 0 et 4, l'exposant est positif ou nul.
  Si Z est compris entre 5 et 9, l'exposant est négatif.

  La formule donnant l'exposant est donnée par:
  α = 100*Z + 10*Y + X - 1000*(Z div 5).

  Autrement dit:
  Si $0 \le Z \le 4$, α = 100*Z + 10*Y + X (donc $0 \le \alpha \le 499$).
  Si $5 \le Z \le 9$, α = 100*Z + 10*Y + X - 1000 (donc $-499 < \alpha \le -1$).

- **m** est la ***mantisse*** ($1 \le m < 10$, sauf $m=0$ si le réel est nul), écrite en "*binaire codé décimal*", et à lire à l'envers, comme d'habitude.

- ϵ est le signe $\pm$ (codé sur un quartet à 0 si $r \ge 0$, à 9 sinon)

Le codage en mémoire débute par l'adresse préfixe **#02933h** (retournée comme il se doit), suivie de 16 quartets (3 pour l'exposant, 12 pour la mantisse, et 1 pour le signe).

Un réel occupe donc 21 quartets en mémoire....

Prenons quelques exemples:

e=2.71828182846 s'écrit:	33920	000	648281828172	0
Le nombre 0 est codé:	33920	000	000000000000	0
Le nombre 1 est codé:	33920	000	000000000001	0
Le nombre -1 est codé:	33920	000	000000000001	9
MAXR=9.99999999999E499 est codé	33920	994	999999999999	0
-MAXR=-9.99999999999E499 est codé	33920	994	999999999999	9
MINR=1E-499 est codé	33920	105	000000000001	0
-MINR=-1E-499 est codé	33920	105	000000000001	9

Nous allons maintenant examiner les réels qui figurent dans la mémoire morte de la HP48. Mais une particularité des entiers compris entre -9 et 9 doit tout d'abord être signalée.

Quand vous entrez un réel sur la pile ou dans un programme, la HP48 le crée en mémoire vive. Les réels entiers compris entre -9 et 9 font exception à cette règle.

Pour des raisons de rapidité d'accès aux réels les plus employés, les entiers compris entre -9 et +9 sont directement désignés par leur adresse en ROM, notée dans le tableau ci-dessous.

Faites-en l'expérience en appliquant l'instruction **BYTES** au réel 10 (on trouve 10.5 octets, c'est-à-dire les 21 quartets règlementaires) et au réel 9 (on ne trouve plus que 2.5 octets: les 5 quartets correspondant à l'adresse de 9 dans la mémoire morte).

Adresse des réels entiers de -9 à +9							
#2A2B4h	0	#2A31Dh	5	#2A386h	-1	#2A3EFh	-6
#2A2C9h	1	#2A332h	6	#2A39Bh	-2	#2A404h	-7
#2A2DEh	2	#2A347h	7	#2A3B0h	-3	#2A419h	-8
#2A2F3h	3	#2A35Ch	8	#2A3C5h	-4	#2A42Eh	-9
#2A308h	4	#2A371h	9	#2A3DAh	-5		

Les entiers du tableau ci-dessous qui apparaissent pour la plupart dans le corps de l'instruction **TYPE**, n'ont pas la propriété d'être reconnus comme objets intégrés à la HP48 (comme le sont -9,-8,...,8,9).

Réels entiers de 10 à 27				#1CDF2h	18	#1CCE2h	23
#650E7h	10	#1CC51h	14	#1CE07h	19	#1CD01h	24
#1CC03h	11	#1CC85h	15	#1CC6Bh	20	#1CD20h	25
#1CC1Dh	12	#1CD3Ah	16	#1CCA4h	21	#1CD73h	26
#1CC37h	13	#1CD54h	17	#1CCC3h	22	#1CD8Dh	27

Dans le tableau ci-dessous, la première colonne contient les différentes valeurs de vitesse de transmission **I/O**, en bauds. La seconde colonne contient le nombre de ticks d'horloge contenus respectivement dans une seconde, une minute, une heure, une journée, une semaine....

Les réels **π**, **e**, **MAXR**, et **MINR** sont des objets intégrés de la HP48, ce qui signifie que la calculatrice les référence directement par l'adresse du tableau. Remarquez cependant qu'une opération telle que **NEG** appliquée à **MAXR** conduit à la création de **-MAXR** en RAM (et non à une référence à l'adresse ci-dessous), comme le prouve l'utilisation de l'instruction **BYTES**.

		Autres réels utiles		#2A443h	π
				#2A472h	MAXR
		#0EFEEh	8192	#2A487h	-MAXR
#22352h	1200	#0F003h	491520	#2A49Ch	1E-499
#22367h	2400	#0F018h	29491200	#2A4B1h	-1E-499
#2237Ch	4800	#0F02Dh	707788800	#514EBh	2π
#22391h	9600	#0F042h	4954521600	#650A8h	e

Le reste					
		#415F1h	100	#650BDh	.5
#1A223h	2.5	#49161h	40	#650D2h	-.5
#2B0CEh	260	#494B4h	.1	#650FCh	180
#2B139h	-260	#49618h	.15	#65111h	200
#31F4Fh	1.8	#4C035h	499	#65126h	360
#320B1h	80	#52C72h	1E-12	#6513Bh	400

NOMBRES COMPLEXES

Un nombre complexe est représenté en mémoire par:

- L'adresse préfixe **#02977h**
- Le code sur 16 quartets de la partie réelle.
- Le code sur 16 quartets de la partie imaginaire.

Les deux réels constituant les parties réelle et imaginaire du nombre complexe sont codés comme il a été dit dans le chapitre précédent, sauf qu'ils ne sont pas précédés de l'adresse préfixe #02933h spécifique aux nombres réels.

- **Exemples:**

(0,0) est codé	77920 000 0000000000000 000 0000000000000
(0.56,-12.34) est codé	77920 999 0000000000650 200 0000000043219

Voici les seuls nombres complexes utiles qui apparaissent dans la ROM de la HP48:

#5196Ah #5267Fh	(-1,0) (0,1)	#524AFh #526AEh	(0,0) (0,-1)	#524F7h	(1,0)

Remarques:

- Seul i=(0,1) est reconnu par la calculatrice comme objet intégré, à l'adresse #5267Fh.
- Il est d'autre part intéressant de noter que la taille du programme « **(0,1)** » est de 28.5 octets, alors que la taille du programme équivalent « **i →NUM** » n'est que de 15 octets.
- Autrement dit, le nombre complexe i=(0,1) doit être entré sous la forme "**i**" (quitte à faire **→NUM** ensuite) pour être directement référencé en ROM, sans quoi la HP48 le recrée en mémoire vive.
- Notons enfin que le nombre complexe nul (0,0) apparait également à l'adresse #4AB2Ah.

Toujours au chapitre des réductions faciles d'octets, méditez l'exemple suivant:
Le programme « **[(1,0) (0,1)]** » occupe 54.5 octets, alors que le programme équivalent « **1 i →NUM 2 →ARRY** » n'en occupe que 22.5.

RÉELS LONGS

Pour accroître la précision des calculs mathématiques intermédiaires et assurer ainsi la précision du résultat final, la HP48 utilise les "***réels longs***".

La valeur absolue d'un réel long non nul est comprise entre:

10^-49999 et 9.999999999999999*10^49999

Le codage d'un réel long ressemble à celui d'un réel normal, notamment dans la succession:

1)adresse-préfixe, 2)exposant, 3)mantisse, 4)signe,

et dans l'emploi de la notation **BCD** (binaire codé décimal):

Cependant:

- L'adresse préfixe est #02955h.
- L'exposant est codé sur 5 quartets successifs VWXYZ.
- Avec ces notations, la valeur de l'exposant α est:
  $\alpha = 10000*Z+1000*Y+100*X+10*W+V$ si $0 \le Z \le 4$ (donc $0 \le \alpha \le 49999$)
  $\alpha = 10000*Z+1000*Y+100*X+10*W+V-100000$ si $5 \le Z \le 9$ ($-49999 \le \alpha \le -1$)
- La mantisse est codée sur 15 quartets (c-à-d 15 chiffres).
- Comme pour les réels, le signe est codé sur le dernier quartet, porté à 0 pour un réel long positif ou nul, et à 9 pour un réel long négatif.

Un réel long est donc codé sur 26 quartets (13 octets).
L'instruction **TYPE** appliquée à un réel long renvoie la valeur 21.

Exemples:

- La valeur de $\pi/180$ est 0.0174532925199433. Codée au format réel long, celà donne

55920	89999	334991529235471	0

- Le réel long égal à -495.920119017593 est codé:

55920	20000	395710911029594	9

RÉELS LONGS

La HP48 ne sait pas afficher les réels longs (contrairement aux entiers-système qui, bien qu'eux aussi inaccessibles en utilisation normale, n'en sont pas moins affichables "en clair").

La calculatrice se contente d'indiquer "***Long Real***" sur la pile pour signaler la présence d'un tel objet.

Les opérations mathématiques usuelles ne s'appliquent évidemment pas aux réels longs, de même qu'il est impossible de valider une ligne de commande contenant un tel objet.

Dans quelques chapitres, vous connaîtrez toutes les adresses utiles pour effectuer des calculs sur les réels longs.

Voici la liste des réels longs apparaissant dans la mémoire morte de la HP48.

Certains d'entre eux ont des valeurs simples. D'autres sont plus "bizarres", mais sont en fait utilisés par la HP48 pour calculer les valeurs (par exemple) de la fonction Gamma d'Euler en un point réel (la fonction **Gamma** est utilisée par l'instruction "*Factorielle*").

Réels longs en ROM de la HP48

Adresse	Valeur	Adresse	Valeur	Adresse	Valeur
#2A4C6h	0	#2C1C5h	100	#10E9Ch	273.15
#2A4E0h	1	#2A562h	.1	#10EB6h	459.67
#2A4FAh	2	#2A57Ch	.5	#2B0F2h	1E10000
#2A514h	3	#2B3DDh	.4	#2B1BCh	1E-10000
#2A52Eh	4	#52A2Fh	.7	#2B1D6h	-1E-10000
#2A548h	5	#10E68h	5/9	#2B31Fh	-495.920119017593
#2B1FFh	7	#0F688h	2π	#2B343h	30.3479606073615
#2A596h	10	#10ED0h	1/4π	#2B36Ch	-76.5594818140208
#2B2DCh	12	#2A458h	π	#2B390h	9.33584905660377
#0F547h	32	#2A62Ch	π/180	#2B3B9h	-1.21142857142857
#2B300h	60	#5230Fh	ln(10)	#2B410h	.918938533204673

NB: le réel long égal à 1 apparaît aussi à l'adresse #10E82h.

COMPLEXES LONGS

Pour la HP48, les complexes longs sont formés par la combinaison de deux réels longs (la partie réelle et la partie imaginaire).

Ils sont ainsi codés sur 47 quartets (23.5 octets):

- 5 quartets pour l'adresse préfixe, égale à **#0299Dh**.
- 21 quartets pour la partie réelle.
- 21 quartets pour la partie imaginaire.

- Un complexe long tel que $\mathbf{z = 32\text{-}i\pi}$ sera codé:

D9920	10000	000000000000023	0	00000	979853562951413	9

- Le seul complexe long présent en ROM est égal à (1,0), et il apparaît à l'adresse #5193Bh. Il est donc codé:

D9920	00000	000000000000001	0	00000	000000000000000	0

La HP48 ne sait pas plus afficher les complexes longs que les réels longs. Elle signale leur présence par le terme générique "*Long Complex*".

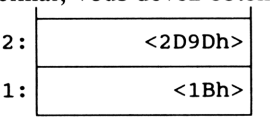
L'instruction **TYPE** appliquée à un complexe long donne le résultat 22.

ENTIERS-SYSTEME

Nous allons maintenant faire connaissance avec un des objets les plus utilisés par les routines internes de la HP48. J'ai nommé l'***entier-système***, ou pour utiliser un jargon anglicisant, le "***system binary***", encore appelé "***short integer***".

Tapez par exemple #03FB3h SYSEVAL, puis #040FDh SYSEVAL

Si vous êtes en mode hexadécimal, vous devez obtenir:



Un tel entier s'écrit en fonction de la base de numération en cours. L'entier-système <1Bh> s'écrira: <27d>, <33o>, ou <11011b>, selon que vous serez en mode "*décimal*", "*octal*" ou "*binaire*".

Un entier-système représente une valeur entière positive ou nulle codée sur 5 quartets, et donc comprise entre 0 et 2^20-1=#FFFFFh.

Il est utilisé par la HP48 de bien des manières, mais entre autres:

- Comme indice de boucle.
- Comme référence à un niveau particulier de la pile.
- Comme référence à une adresse de la ROM. C'est le cas de <2D9Dh> vu plus haut, qui désigne l'adresse #02D9Dh qui n'est autre que l'adresse-préfixe des routines RPL.
- Comme identificateur d'objet.

Un entier-système peut être manipulé comme n'importe quel objet sur la pile (vous pouvez ainsi éxécuter **2 →LIST** avec la pile ci-dessus et vous obtiendrez la liste **{ <2D9Dh> <1Bh> }** au niveau 1).

Il y a tout de même un certain nombre de choses que vous ne pouvez pas faire avec les entiers-système, comme par exemple les ajouter avec l'opérateur + (mais je vous donnerai l'adresse où se trouve implantée leur addition...patience...), mais surtout (comme pour tout objet non-standard) vous ne pouvez pas valider une ligne de commande qui contiendrait de tels objets.

En mémoire, un entier système est codé sur 10 quartets consécutifs:

- Une adresse-préfixe, égale à **#02911h** (retournée en 11920)
- Les 5 quartets correspondant au contenu de l'entier, retournés de la même manière.

Ainsi, l'entier-système <02D9Dh> est codé: 11920D9D20

Type d'un entier-système: L'instruction **TYPE** leur attribue la valeur 20.

Voici les adresses où trouver les entiers-système les plus utiles. Commençons par les "*petits*", les plus utilisés.

Dans les tables qui suivent, les entiers-système sont classés par valeurs croissantes. De nombreuses adresses se suivent à intervalles réguliers de 10 quartets.

Quand un entier-système figure plusieurs fois dans la ROM de la HP48, je n'ai retenu qu'une seule des adresses correspondantes (en privilégiant les occurences dans les quelques tables d'entiers-systèmes consécutifs qu'on trouve en mémoire).

Entiers-système de <0h> à <3Fh>

Adresse		Adresse		Adresse		Adresse	
#03FEFh	<0h>	#0408Fh	<10h>	#0412Fh	<20h>	#64B3Ah	<30h>
#03FF9h	<1h>	#04099h	<11h>	#04139h	<21h>	#64B44h	<31h>
#04003h	<2h>	#040A3h	<12h>	#04143h	<22h>	#64B4Eh	<32h>
#0400Dh	<3h>	#040ADh	<13h>	#0414Dh	<23h>	#64B58h	<33h>
#04017h	<4h>	#040B7h	<14h>	#04157h	<24h>	#64B62h	<34h>
#04021h	<5h>	#040C1h	<15h>	#04161h	<25h>	#64B6Ch	<35h>
#0402Bh	<6h>	#040CBh	<16h>	#0416Bh	<26h>	#64B76h	<36h>
#04035h	<7h>	#040D5h	<17h>	#04175h	<27h>	#64B80h	<37h>
#0403Fh	<8h>	#040DFh	<18h>	#0417Fh	<28h>	#64B8Ah	<38h>
#04049h	<9h>	#040E9h	<19h>	#04189h	<29h>	#64B94h	<39h>
#04053h	<Ah>	#040F3h	<1Ah>	#04193h	<2Ah>	#64B9Eh	<3Ah>
#0405Dh	<Bh>	#040FDh	<1Bh>	#0419Dh	<2Bh>	#64BA8h	<3Bh>
#04067h	<Ch>	#04107h	<1Ch>	#64B12h	<2Ch>	#64BB2h	<3Ch>
#04071h	<Dh>	#04111h	<1Dh>	#64B1Ch	<2Dh>	#64BBCh	<3Dh>
#0407Bh	<Eh>	#0411Bh	<1Eh>	#64B26h	<2Eh>	#64BC6h	<3Eh>
#04085h	<Fh>	#04125h	<1Fh>	#64B30h	<2Fh>	#64BD0h	<3Fh>

Entiers-système compris entre <40h> et <FFh>

Adresse		Adresse		Adresse		Adresse	
#64BDAh	<40h>	#3A215h	<58h>	#64D10h	<80h>	#64DA6h	<A9h>
#64BE4h	<41h>	#1CCD8h	<5Fh>	#64D1Ah	<82h>	#64DB0h	<AAh>
#64BEEh	<42h>	#64C8Eh	<60h>	#64D24h	<83h>	#64DBAh	<AEh>
#64BF8h	<43h>	#64C98h	<61h>	#46BE3h	<84h>	#1CD69h	<AFh>
#64C02h	<44h>	#64CA2h	<62h>	#2F4A2h	<86h>	#64DC4h	<B1h>
#64C16h	<46h>	#64CACh	<64h>	#64D2Eh	<8Fh>	#17F4Ah	<B8h>
#64C20h	<4Ah>	#64CB6h	<65h>	#64D38h	<91h>	#64DCEh	<BBh>
#2D96Ah	<4Bh>	#6D98Ch	<68h>	#64D42h	<92h>	#64DD8h	<C0h>
#6B5C4h	<4Dh>	#3A20Bh	<6Eh>	#6C68Bh	<93h>	#20D2Ch	<C8h>
#64C2Ah	<4Fh>	#64CC0h	<6Fh>	#64D4Ch	<9Ah>	#64DE2h	<CCh>
#64C34h	<50h>	#64CCAh	<70h>	#64D56h	<9Eh>	#64DECh	<D0h>
#64C3Eh	<51h>	#64CD4h	<71h>	#64D60h	<9Fh>	#64DF6h	<E1h>
#64C48h	<52h>	#64CDEh	<72h>	#64D6Ah	<A0h>	#64E00h	<EAh>
#64C52h	<53h>	#64CE8h	<73h>	#64D74h	<A1h>	#64E0Ah	<EEh>
#64C5Ch	<54h>	#64CF2h	<74h>	#64D7Eh	<A2h>	#64E14h	<F0h>
#64C66h	<55h>	#64CFCh	<75h>	#64D88h	<A5h>	#64E1Eh	<FDh>
#64C70h	<56h>	#64D06h	<7Ah>	#64D92h	<A6h>	#64E28h	<FFh>
#64C7Ah	<57h>	#1CD16h	<7Fh>	#64D9Ch	<A7h>		

ENTIERS-SYSTEME

Voici maintenant les entiers-système "à 3 chiffres" surtout utilisés par la HP48 pour tester la nature des objets présents aux trois premiers niveaux de la pile.

Entiers-système compris entre <100h> et <FFFh>					
#64E32h	<100h>	#1C93Fh	<515h>	#1E49Ch	<85Ch>
#64E3Ch	<102h>	#64F72h	<550h>	#64F9Ah	<861h>
#50E45h	<104h>	#50E4Fh	<602h>	#64FA4h	<862h>
#64E46h	<106h>	#4A320h	<605h>	#64FAEh	<865h>
#64E50h	<107h>	#4A32Ah	<606h>	#64FB8h	<86Eh>
#64E5Ah	<110h>	#4A334h	<607h>	#20D4Ah	<8F1h>
#64E64h	<111h>	#4A33Eh	<608h>	#20D3Bh	<9F1h>
#15E0Bh	<112h>	#4A348h	<609h>	#33CBFh	<A01h>
#15D6Fh	<117h>	#4A352h	<60Ah>	#33CD3h	<A02h>
#15DABh	<118h>	#4A35Ch	<60Bh>	#64FC2h	<A03h>
#64E6Eh	<123h>	#4A366h	<60Ch>	#33D91h	<A04h>
#64E78h	<124h>	#4A370h	<60Dh>	#33C29h	<A05h>
#31C5Eh	<127h>	#4A37Ah	<60Eh>	#33C83h	<A06h>
#33CA1h	<12Fh>	#4A384h	<60Fh>	#64FCCh	<A11h>
#64E82h	<131h>	#4A38Eh	<610h>	#64FD6h	<A12h>
#64E8Ch	<132h>	#4A398h	<611h>	#64FE0h	<A1Ah>
#64E96h	<133h>	#4A3A2h	<612h>	#64FEAh	<A21h>
#64EA0h	<134h>	#4A3ACh	<613h>	#64FF4h	<A22h>
#64EAAh	<135h>	#4A3B6h	<614h>	#64FFEh	<A2Ah>
#64EB4h	<136h>	#4A3C0h	<615h>	#65008h	<A61h>
#64EBEh	<137h>	#4A3CAh	<616h>	#65012h	<A62h>
#64EC8h	<138h>	#4A3D4h	<617h>	#6501Ch	<A65h>
#64ED2h	<139h>	#4A3DEh	<618h>	#65026h	<A6Eh>
#64EDCh	<13Ah>	#4A3E8h	<619h>	#65030h	<AA1h>
#64EE6h	<13Bh>	#4A3F2h	<61Ah>	#6503Ah	<AA2h>
#64EF0h	<13Dh>	#4A3FCh	<61Bh>	#65044h	<AAAh>
#64EFAh	<13Eh>	#4A406h	<61Ch>	#28AE7h	<B01h>
#64F04h	<151h>	#4A410h	<61Dh>	#2D5C3h	<C02h>
#64F0Eh	<200h>	#4A41Ah	<61Eh>	#6504Eh	<C06h>
#4ECFDh	<201h>	#4A424h	<61Fh>	#65058h	<C07h>
#4ED25h	<202h>	#4A42Eh	<620h>	#65062h	<C08h>
#1CEAFh	<204h>	#4A438h	<621h>	#6506Ch	<C0Ah>
#64F18h	<205h>	#4A442h	<622h>	#65076h	<C0Bh>
#6737Bh	<206h>	#4A44Ch	<623h>	#2DA7Ch	<C0Ch>
#1B147h	<304h>	#4A456h	<624h>	#2F0E0h	<C0Dh>
#64F22h	<311h>	#4A460h	<628h>	#2F8EEh	<C0Eh>
#1C930h	<313h>	#4A46Ah	<629h>	#2FA9Ch	<C0Fh>
#64F2Ch	<411h>	#4A474h	<62Ah>	#2F04Ah	<C10h>
#64F36h	<412h>	#4A47Eh	<62Bh>	#2DD6Fh	<C11h>
#64F40h	<444h>	#4A488h	<62Ch>	#2EC39h	<C12h>
#64F4Ah	<451h>	#4A492h	<62Dh>	#3182Ch	<C15h>
#64F54h	<452h>	#4A49Ch	<62Eh>	#31BF1h	<C16h>
#37DE7h	<501h>	#20496h	<644h>	#67385h	<C17h>
#472C8h	<502h>	#64F86h	<650h>	#1C889h	<C22h>
#472D2h	<503h>	#64F90h	<700h>	#1C903h	<C2Ch>
#472DCh	<504h>	#1D448h	<710h>	#1C87Ah	<C55h>
#472E6h	<505h>	#1D42Fh	<750h>	#1E460h	<C5Ch>
#472F0h	<506h>	#08E14h	<7FFh>	#26289h	<CFFh>
#64F5Eh	<510h>	#4F390h	<800h>	#65080h	<DFFh>
#64F68h	<511h>	#1E4BAh	<82Ch>	#6508Ah	<E00h>
		#1C898h	<855h>		

Passons enfin aux entiers-système supérieurs ou égaux à <1000h>, et qui sont surtout utilisés pour référencer des adresses.

Tout d'abord, les entiers-système qui renvoient à l'adresse-préfixe de certains des types d'objets connus de la HP48.

Adresse	Contenu	Référence à	Adresse	Contenu	Référence à
#03F8Bh	<2933h>	réels	#03FC7h	<2A96h>	répertoires
#03F95h	<2977h>	complexes	#03FD1h	<2E6Dh>	noms locaux
#03F9Fh	<2A74h>	listes	#03FDBh	<2955h>	réels longs
#03FA9h	<2E48h>	noms globaux	#03FE5h	<2ADAh>	objets-unité
#03FB3h	<2D9Dh>	routines RPL	#25C37h	<2B1Eh>	graphiques
#03FBDh	<2AB8h>	expressions			

Sans plus de cérémonies, voici les adresses des autres entiers-système, qui pour la plupart renvoient à des adresses, mais dont certains pourront vous être utiles (<FFFFFh>, <80000h> par exemple).

#04CDCh <72000h>	#05176h <FFFFFh>	#16AD6h <4000h>	#16AE5h <5000h>
#16AF4h <8000h>	#16B03h <9000h>	#16B12h <E000h>	#16B21h <D000h>
#16B30h <2F000h>	#1A48Ah <7DAC5h>	#1B2C8h <7D2D2h>	#1B361h <7D877h>
#1B3BFh <7D58Eh>	#1B471h <7D50Ch>	#1B4F2h <7C5D3h>	#1B54Bh <7C768h>
#1B5A4h <7C844h>	#1B5F3h <7CC18h>	#1B642h <7CDCBh>	#1B691h <7CF1Fh>
#1B6E0h <7CA2Eh>	#1B76Bh <7CACEh>	#1B7D8h <7CB6Eh>	#1B98Bh <7D1AAh>
#1BA02h <7D231h>	#1BA79h <7D11Dh>	#1BAEFh <7D0A0h>	#1E5DCh <2111h>
#1E5EBh <5B11h>	#1F024h <7D9DFh>	#1F02Eh <7D8EAh>	#1F25Ch <7DA20h>
#1F26Bh <7DF4Dh>	#1F2D3h <A110h>	#1F2E2h <AA10h>	#1F2F1h <A1A0h>
#1F300h <AAA0h>	#1F334h <7DA6Bh>	#1F533h <7DB15h>	#1F6CCh <7DB42h>
#1F6D6h <7DD13h>	#21744h <71541h>	#21DE7h <70000h>	#22647h <7448Ah>
#22DFEh <7427Ch>	#2C7ECh <11118h>	#3974Dh <3039h>	#3A73Fh <FFFFAh>
#3F103h <7DBBFh>	#3F10Dh <7DC05h>	#3F117h <7DC5Ah>	#3F121h <7DC82h>
#3F12Bh <7DCA0h>	#3F135h <7DCBEh>	#3F13Fh <7DCCDh>	#3F149h <7DCE1h>
#3F153h <7DCF0h>	#3F15Dh <7DD04h>	#3FB1Fh <7B4E4h>	#41125h <FFFFBh>
#4C908h <FFFFDh>	#4F4D0h <80000h>	#56B87h <7BA3Eh>	#56B91h <7BAC0h>
#56B9Bh <7BB1Fh>	#56BA5h <7BB83h>	#56C13h <7BC05h>	#56C1Dh <7BC32h>
#56C40h <7BCE1h>	#56C4Ah <7BD36h>	#56C54h <7BDA4h>	#56C5Eh <7BDF4h>
#56C68h <7BE58h>	#56C72h <7BEA3h>	#56C7Ch <7BEF8h>	#56C86h <7BF3Eh>
#56CCCh <7BF8Eh>	#56CD6h <7BFB1h>	#56CE0h <7BFE3h>	#56CEAh <7C010h>
#56CF4h <7C04Ch>	#56CFEh <7C06Fh>	#56D08h <7C0ABh>	#56D26h <7C0ECh>
#56D30h <7C146h>	#56D3Ah <7C164h>	#56D44h <7C191h>	#56D4Eh <7C1C8h>
#56D58h <7C213h>	#56D62h <7C254h>	#56D6Ch <7C295h>	#56D7Bh <7C303h>
#56D99h <7C358h>	#56F36h <7C3A3h>	#5727Ah <7C3CBh>	#57284h <7C411h>
#5728Eh <7C547h>	#57298h <7C5A6h>	#57455h <7B5F2h>	#5745Fh <7B638h>
#57469h <7B6A6h>	#57473h <7B705h>	#5747Dh <7B773h>	#5748Ch <7B7EBh>
#574CDh <7B8CCh>	#574EBh <7B8F4h>	#574F5h <7B917h>	#574FFh <7B93Fh>
#57509h <7B962h>	#57513h <7B985h>	#5751Dh <7B9ADh>	#575F9h <7B9D0h>
#57617h <7BA16h>	#65094h <70000h>	#6509Eh <FFFFFh>	#653C4h <726A5h>
#653CEh <72704h>	#653D8h <72DCFh>	#653E2h <72F1Eh>	#653ECh <736F9h>
#653F6h <7232Ch>	#65400h <7260Ah>	#6540Ah <72281h>	#65414h <72FE6h>
#67646h <80000h>	#67D12h <80000h>	#6866Eh <80000h>	#6B017h <80000h>
#6B035h <80000h>	#7C745h <7C7FEh>	#7C948h <7C66Eh>	#7CEB6h <7CE5Ch>
#7CEFCh <7CD85h>	#7CFBAh <7CCAEh>	#7D485h <7D421h>	#7D6ABh <7D642h>
#7D782h <7D70Ah>			

ENTIERS BINAIRES

Le codage d'un entier binaire en mémoire est totalement indépendant du choix de la base de numération, et du nombre de bits utiles spécifié par l'instruction **STWS**. Il se présente de la manière suivante:

- L'adresse préfixe **#02A4Eh** sur 5 quartets.
- La longueur totale de l'objet (exprimée en quartets, à l'exception de l'adresse-préfixe elle-même). Cette longueur est évidemment écrite à l'envers (on commence à en avoir l'habitude).
- Les différents quartets formant l'entier binaire (chaque quartet correspond à un chiffre de 0 à F dans l'écriture hexadécimale).
  Là encore, il y a un retournement général: les quartets les moins significatifs sont au début, les plus significatifs à la fin.

Exemple:

- L'entier binaire #123456789ABCDEF0h, entré au clavier, est codé:

E4A20	51000	0FEDCBA987654321

(L'adresse préfixe est #02A4Eh, et la longueur en quartets est #15h=21 car il y a 16 chiffres et il faut y ajouter les 5 quartets sur lesquels est codée cette longueur !)

Quand vous entrez un entier binaire en ligne de commande, il est toujours codé sur 64 bits, c'est-à-dire 16 quartets (donc 16 chiffres dans la numération hexadécimale).

On peut alors se demander à quoi sert le champ de 5 quartets contenant la longueur (hors adresse-préfixe) si cette longueur est constante. La réponse est simple, mais elle n'est pas donnée dans les manuels de la HP48: Il est en effet possible de créer des entiers binaires de longueur quelconque.

Par exemple, on trouve même un *binaire vide*, codé E4A2050000, à l'adresse #055D5h. On trouve un *binaire nul*, codé E4A20900000000 (donc n'ayant à priori que trois chiffres en hexadécimal) à l'adresse #1A215h. Inversement, la ROM de la HP48 contient des entiers binaires très longs, utilisés comme tables de conversion.

- Effectuez par exemple: #22651h SYSEVAL.
- Vous devez voir affiché: #1F6F56F6F3BF6F20h.
- Editez ce binaire: Vous verrez apparaître: C# 1925 02F6FB3F6F65F.....
  Autrement dit c'est un nombre binaire de **1925 chiffres** !

C'est ainsi qu'apparaissent les binaires de longueur supérieure à 16 chiffres hexadécimaux quand ils sont édités: ils sont précédés de **C#** et du nombre de leurs chiffres en hexadécimal, à la manière des chaînes comptées (précédées de **C$**).

C'est pourquoi on peut parler d'**entiers binaires comptés** pour désigner des entiers binaires ayant un nombre de chiffres quelconque (donc, à priori, différent de 16).

N.B: Il est possible d'entrer au clavier une *chaîne comptée*. Il est malheureusement impossible d'entrer un entier *binaire compté* en utilisant le préfixe **C#**. Dommage... Vous verrez cependant plus loin que quelques SYSEVALs vous permettont beaucoup mieux !

Voici maintenant la liste des binaires en ROM de la HP48. On s'aperçoit qu'ils ont presque toujours une longueur atypique.

Adresse	Codage en mémoire	Valeur
#055D5h	E4A20 50000	Binaire vide
#10E34h	E4A20 51000 0000000000000000	#0h
#10E4Eh	E4A20 51000 0000000010000000	#100000000h
#1A215h	E4A20 90000 0000	#0h
#1A471h	E4A20 F0000 014062625	#526260410h
#1A9F9h	E4A20 90000 0108	#8010h
#1AC75h	E4A20 A0000 70107	#70107h
#1AEDEh	E4A20 A0000 80108	#80108h
#1B013h	E4A20 C0000 8014050	#504108h
#1B104h	E4A20 A0000 90109	#90109h
#1B113h	E4A20 C0000 9014050	#504109h
#1B3E7h	E4A20 90000 010C	#C010h
#1BB2Ah	E4A20 90000 A010	#10Ah
#1BE84h	E4A20 A0000 80108	#80108h
#1E7CEh	E4A20 A0000 50105	#50105h
#1E854h	E4A20 A0000 40104	#40104h
#1E8CBh	E4A20 90000 0105	#5010h
#1EA21h	E4A20 A0000 60106	#60106h
#1F00Eh	E4A20 C0000 0134250	#524310h
#1F241h	E4A20 11000 014060626350	#53626060410h
#1F319h	E4A20 11000 014370606250	#52606073410h
#1F40Ch	E4A20 F0000 2214370B50	#5B0734122h
#1F523h	E4A20 B0000 014250	#52410h
#1F5D9h	E4A20 E0000 014360950	#59063410h
#1FB1Dh	E4A20 A0000 90127	#72109h
#22651h	E4A20 A8700	1925 chiffres
#22E08h	E4A20 69000	145 chiffres
#258D0h	E4A20 50000	Binaire vide
#25D3Ah	E4A20 50100	256 chiffres
#25E44h	E4A20 50100	256 chiffres
#25F4Eh	E4A20 50100	256 chiffres
#26058h	E4A20 50100	256 chiffres
#26DD2h	E4A20 C0000 0140950	#590410h
#26DE3h	E4A20 B0000 014050	#50410h
#26DF3h	E4A20 D0000 01406050	#5060410h
#26E05h	E4A20 F0000 0140606050	#506060410h
#26E19h	E4A20 11000 014060606050	#50606060410h
#26E2Fh	E4A20 31000 01406060606050	#5060606060410h
#26E47h	E4A20 A0000 00700	#700h
#3834Fh	E4A20 53200	560 chiffres
#737ECh	E4A20 73A00	2610 chiffres
#74228h	E4A20 F4000	74 chiffres
#7427Ch	E4A20 90200	516 chiffres
#7448Ah	E4A20 F0B10	6922 chiffres

CHAINES DE CARACTERES

Comme pour tous les objets de la HP48 le codage des chaînes de caractères débute par une adresse préfixe qui est ici **#02A2Ch**.

Elle est suivie par un groupe de 5 quartets précisant la taille de l'objet chaîne (taille exprimées en quartets, et incluant tout l'objet à l'exception de l'adresse préfixe).

Suivent ensuite les caractères formant la chaîne, dans l'ordre naturel, chacun d'eux étant codé sur deux quartets (entre 0 et 255: voir instructions **CHR** et **NUM**)

L'adresse-préfixe, la taille (toutes deux sur 5 quartets) et les octets codant chaque caractère sont "retournés" en mémoire.

Exemple: le codage de la chaîne "HALT" est:

C2A20	D0000	8414C445

- Ici la taille de l'objet (hors adresse-préfixe) est #Dh=13=5+2*4 (4 caractères).
- On reconnait les codes #48h, #41h, #4Ch, et #54h, des caractères **H**, **A**, **L**, et **T**.

N.B: La chaîne vide "" est codée: C2A2050000

Il y a une grande analogie dans le codage des chaînes et des entiers binaires:

- Une adresse-préfixe sur 5 quartets,
- Puis la taille sur 5 quartets (correspondant au nombre de quartets de l'objet, à l'exception de l'adresse-préfixe)
- Enfin le contenu effectif de l'objet.

Voici la liste exhaustive des chaînes de caractères de la ROM, même si l'intérêt d'une telle entreprise est relativement moyen (en regard des petits trésors sysevalesques que renferme ce modeste ouvrage...)

N.B: dans tous les tableaux ci-dessous, certains caractères sont remplacés par leur code (au format HP48). Par exemple, \215 remplace le caractère obtenu par 215 CHR)

Adresse	Chaîne	Adresse	Chaîne	Adresse	Chaîne
#055DFh	""	#0E524h	""	#0E5C6h	""
#0FA69h	"g"	#0FA8Eh	"m"	#0FAAEh	"A"
#0FACEh	"s"	#0FAEEh	"K"	#0FB0Eh	"cd"
#0FB30h	"mol"	#0FB54h	"?"	#11231h	"1E"
#15331h	"NXEQ"	#15442h	": "	#1585Fh	"EXPR="
#158B9h	"LEFT"	#158E4h	"RIGHT"	#15911h	"EXPR"
#15DF6h	"\215"	#15E47h	"GROB"	#15F23h	"C$"
#15FB5h	"C#"	#161B2h	"XLIB"	#16BB2h	"{"
#16BEBh	"}"	#16C42h	"DIR"	#16CEDh	"END"
#16D25h	":"	#16D25h	":"	#17DB4h	"PICT"
#183C9h	"0:"	#19AABh	""	#19F4Fh	"Rpt="
#19F70h	" week(s)"	#19F8Ah	" day(s)"	#19FA2h	" hour(s)"
#19FBCh	" minute(s)"	#19FDAh	" second(s)"	#19FF8h	" ticks"
#1FF34h	"Intercept"	#1FF50h	"Slope"	#20E4Bh	" "
#2127Dh	"IO"	#2189Dh	"ROM"	#218C6h	"SYSRAM"

Adresse	Chaîne	Adresse	Chaîne	Adresse	Chaîne
#2212Dh	"IR"	#2213Bh	"wire"	#2216Bh	"binary"
#22181h	"ASCII"	#221F9h	"none"	#2220Dh	"odd"
#2221Fh	"even"	#22233h	"mark"	#2226Ah	"spc"
#22290h	"invalid"	#22ED7h	"IF-prompt"	#2539Ah	"TO"
#253A8h	"DIR"	#253B8h	":"	#253C4h	"ELSE"
#253D6h	"END"	#253E6h	"UNTIL"	#253FAh	"REPEAT"
#25410h	"NEXT"	#25422h	"STEP"	#25434h	"THEN"
#25446h	"→"	#25678h	"bodh"	#25CF5h	" "
#28A08h	"SQRT"	#28A1Ah	"SQ"	#28A28h	"INV"
#2D37Ch	"p% @-# 1"	#2DF88h	":\10"	#2E4F0h	"\13\10"
#2E87Bh	"F"	#2E887h	"G"	#2E8B1h	"R"
#2F162h	"%%HP:"	#30B22h	"^* @-#Y"	#31D26h	" "
#32341h	"\4"	#34105h	"+ "	#34115h	"- "
#3412Fh	"?"	#3971Dh	"HALT"	#397D9h	"1USR"
#397EBh	"USER"	#3982Ah	"ALG"	#39862h	"PRG"
#3998Ah	" }"	#39F09h	"1: "	#39F28h	" "
#39FF2h	"\31"	#3B29Dh	"PARTS"	#3B2C0h	"PROB"
#3B2E1h	"HYP"	#3B300h	"MATRX"	#3B323h	"VECTR"
#3B346h	"BASE"	#3B55Bh	"STK"	#3B57Ah	"OBJ"
#3B599h	"DSPL"	#3B5BAh	"CTRL"	#3B5DBh	"BRCH"
#3B5FCh	"TEST"	#3B65Eh	"DRPN"	#3B7ECh	"DBUG"
#3B80Dh	"SST"	#3B82Ch	"SST↓"	#3B84Dh	"NEXT"
#3BA12h	"IR/W"	#3BA33h	"ASCII"	#3BB6Eh	"SYM"
#3BBB5h	"BEEP"	#3BC0Dh	"CNCT"	#3BE3Bh	"SOLVR"
#3BED1h	"PLOTR"	#3BF30h	"PTYPE"	#3BF76h	"NEW"
#3BFA4h	"EDEQ"	#3C001h	"CAT"	#3C066h	"HIST"
#3C366h	"RESET"	#3C4E2h	"SET"	#3C529h	"ADJST"
#3C574h	"ALRM"	#3C67Bh	"HR+"	#3C6A9h	"HR-"
#3C6D7h	"MIN+"	#3C707h	"MIN-"	#3C737h	"SEC+"
#3C767h	"SEC-"	#3C7ABh	">DATE"	#3C7CEh	">TIME"
#3C7F1h	"A/PM"	#3C812h	"EXEC"	#3C851h	"RPT"
#3C88Eh	"SET"	#3C8DFh	"WEEK"	#3C900h	"DAY"
#3C91Fh	"HOUR"	#3C940h	"MIN"	#3C95Fh	"SEC"
#3C97Eh	"NONE"	#3C9C2h	"→DATE"	#3C9E5h	"→TIME"
#3CA08h	"A/PM"	#3CA3Dh	"12/24"	#3CA74h	"M/D"
#3CB4Ch	"NEW"	#3CDA5h	"LIN"	#3CDC4h	"LOG"
#3CDE3h	"EXP"	#3CE02h	"PWR"	#3CE21h	"BEST"
#3CE7Eh	"LENG"	#3CE9Fh	"AREA"	#3CEC0h	"VOL"
#3CEDFh	"TIME"	#3CF00h	"SPEED"	#3CF23h	"MASS"
#3CF44h	"FORCE"	#3CF67h	"ENRG"	#3CF88h	"POWR"
#3CFA9h	"PRESS"	#3CFCCh	"TEMP"	#3CFEDh	"ELEC"
#3D00Eh	"ANGL"	#3D02Fh	"LIGHT"	#3D066h	"VISC"
#3D09Bh	"m"	#3D0A7h	"cm"	#3D0B5h	"mm"
#3D0C3h	"yd"	#3D0D1h	"ft"	#3D0DFh	"in"
#3D0EDh	"Mpc"	#3D0FDh	"pc"	#3D10Bh	"lyr"
#3D11Bh	"au"	#3D129h	"km"	#3D137h	"mi"
#3D145h	"nmi"	#3D155h	"miUS"	#3D167h	"chain"
#3D17Bh	"rd"	#3D189h	"fath"	#3D19Bh	"ftUS"
#3D1ADh	"mil"	#3D1BDh	"μ"	#3D1C9h	"\182"
#3D1D5h	"fermi"	#3D202h	"m^2"	#3D212h	"cm^2"
#3D224h	"b"	#3D230h	"yd^2"	#3D242h	"ft^2"
#3D254h	"in^2"	#3D266h	"km^2"	#3D278h	"ha"
#3D286h	"a"	#3D292h	"mi^2"	#3D2A4h	"miUS^2"
#3D2BAh	"acre"	#3D2E5h	"m^3"	#3D2F5h	"st"
#3D303h	"cm^3"	#3D315h	"yd^3"	#3D327h	"ft^3"

Adresse	Chaîne	Adresse	Chaîne	Adresse	Chaîne
#3D339h	"in^3"	#3D357h	"galUK"	#3D36Bh	"galC"
#3D37Dh	"gal"	#3D38Dh	"qt"	#3D39Bh	"pt"
#3D3A9h	"ml"	#3D3B7h	"cu"	#3D3C5h	"ozfl"
#3D3D7h	"ozUK"	#3D3E9h	"tbsp"	#3D3FBh	"tsp"
#3D40Bh	"bbl"	#3D41Bh	"bu"	#3D429h	"pk"
#3D437h	"fbm"	#3D460h	"yr"	#3D46Eh	"d"
#3D47Ah	"h"	#3D486h	"min"	#3D496h	"s"
#3D4A2h	"Hz"	#3D4C9h	"m/s"	#3D4D9h	"cm/s"
#3D4EBh	"ft/s"	#3D4FDh	"kph"	#3D50Dh	"mph"
#3D51Dh	"knot"	#3D52Fh	"c"	#3D53Bh	"ga"
#3D562h	"kg"	#3D570h	"g"	#3D57Ch	"lb"
#3D58Ah	"oz"	#3D598h	"slug"	#3D5AAh	"lbt"
#3D5BAh	"ton"	#3D5CAh	"tonUK"	#3D5DEh	"t"
#3D5EAh	"ozt"	#3D5FAh	"ct"	#3D608h	"grain"
#3D61Ch	"u"	#3D628h	"mol"	#3D651h	"N"
#3D65Dh	"dyn"	#3D66Dh	"gf"	#3D67Bh	"kip"
#3D68Bh	"lbf"	#3D69Bh	"pdl"	#3D6C4h	"J"
#3D6D0h	"erg"	#3D6E0h	"Kcal"	#3D6F2h	"cal"
#3D702h	"Btu"	#3D712h	"ft*lbf"	#3D728h	"therm"
#3D73Ch	"MeV"	#3D74Ch	"eV"	#3D773h	"W"
#3D77Fh	"hp"	#3D7B4h	"atm"	#3D7C4h	"bar"
#3D7D4h	"psi"	#3D7E4h	"torr"	#3D7F6h	"mmHg"
#3D808h	"inHg"	#3D81Ah	"inH2O"	#3D847h	"°C"
#3D855h	"°F"	#3D863h	"K"	#3D86Fh	"°R"
#3D896h	"V"	#3D8A2h	"A"	#3D8AEh	"C"
#3D8BAh	"Ω"	#3D8C6h	"F"	#3D8D2h	"W"
#3D8DEh	"Fdy"	#3D8EEh	"H"	#3D8FAh	"mho"
#3D90Ah	"S"	#3D916h	"T"	#3D922h	"Wb"
#3D949h	"°"	#3D955h	"r"	#3D961h	"grad"
#3D973h	"arcmin"	#3D989h	"arcs"	#3D99Bh	"sr"
#3D9C2h	"fc"	#3D9D0h	"flam"	#3D9E2h	"lx"
#3D9F0h	"ph"	#3D9FEh	"sb"	#3DA0Ch	"lm"
#3DA1Ah	"cd"	#3DA28h	"lam"	#3DA51h	"Gy"
#3DA5Fh	"rad"	#3DA6Fh	"rem"	#3DA7Fh	"Sv"
#3DA8Dh	"Bq"	#3DA9Bh	"Ci"	#3DAA9h	"R"
#3DACEh	"P"	#3DADAh	"St"	#3DB6Ah	"HEX"
#3DBACh	"DEC"	#3DBEEh	"OCT"	#3DC30h	"BIN"
#3DECAh	"FCN"	#3DF43h	"ROOT"	#3DF64h	"ISECT"
#3DF87h	"SLOPE"	#3DFAAh	"AREA"	#3DFCBh	"EXTR"
#3E01Eh	"F(X)"	#3E03Fh	"F'"	#3E05Ch	"NXEQ"
#3E0A0h	"+/-"	#3E0C9h	"REPL"	#3E0EAh	"SUB"
#3E109h	"DEL"	#3E128h	"COORD"	#3E17Dh	"ZOOM"
#3E1A3h	"Keys"	#3E1C4h	"MARK"	#3E221h	"CIRCLE"
#3E246h	"Z-BOX"	#3E282h	"DOT+"	#3E2B7h	"DOT-"
#3E2E2h	"←SKIP"	#3E364h	"SKIP→"	#3E3E6h	"←DEL"
#3E4CFh	"DEL→"	#3E595h	"INS"	#3E5D2h	"↑STK"
#3E729h	"SETUP"	#3E783h	"STD"	#3E7C5h	"FIX"
#3E7F8h	"SCI"	#3E82Bh	"ENG"	#3E85Eh	"ML"
#3E89Eh	"DEG"	#3EA5Ch	"CMD"	#3EAADh	"STK"
#3EB2Bh	"ARG"	#3EB77h	"FM,"	#3EBBEh	"CLK"
#3EC05h	"MODL"	#3F40Ch	"PORT0"	#3F461h	"PORT1"
#3F4BBh	"PORT2"	#414BDh	": "	#415A7h	": "
#43D2Eh	"VIEW"	#43DC7h	"DRPN"	#43DE8h	"KEEP"
#43E09h	"LEVEL"	#44228h	"\134"	#44243h	":"
#46017h	"EDIT"	#4604Ch	"←WID"	#4606Dh	"WID→"

Adresse	Chaîne	Adresse	Chaîne	Adresse	Chaîne
#460B6h	"+ROW"	#460D7h	"-ROW"	#460F8h	"+COL"
#46119h	"-COL"	#4613Ah	"→STK"	#4615Bh	"↑STK"
#461C2h	"GO→"	#461F0h	"GO↓"	#4621Eh	"VEC"
#47660h	"Indep:"	#476BCh	"Depnd:"	#476F0h	"x:"
#4771Ch	"y:"	#47900h	".EQ"	#47910h	",EQ"
#47F6Fh	"["	#47F8Fh	"x"	#47FAFh	"]"
#47FD4h	" dir"	#48009h	":"	#48042h	"\134"
#485C1h	"1-VAR"	#48657h	"PLOT"	#486B9h	"2-VAR"
#4871Dh	"EDIT"	#48766h	"SOLVR"	#487CAh	"PLOTR"
#48810h	"EQ+"	#4891Fh	"EDIT"	#48995h	"PURGE"
#48A76h	"→STK"	#48ACEh	"undefined"	#48B44h	"VIEW"
#48C69h	"FAST"	#48CBCh	"ORDER"	#48D25h	"EDIT"
#48D92h	"PURGE"	#48DECh	"EXECS"	#4971Dh	": "
#49825h	"Slope"	#49870h	"Root"	#4991Dh	"Area"
#499FCh	"I-sect"	#49A4Eh	"Extrm"	#49A8Fh	"F(x)"
#4A605h	")="	#4A677h	"Xcol:"	#4A69Ah	" Ycol:"
#4A6C4h	" Modl:"	#4EE51h	"XAUTO"	#4EE92h	"X"
#4EEC6h	"Y"	#4EEFAh	"XY"	#596B5h	"COLCT"
#59701h	"DNEG"	#5974Fh	"DINV"	#5979Dh	"*1"
#59823h	"^1"	#5986Dh	"/1"	#598F3h	"+1-1"
#59955h	"←→"	#59981h	"←A"	#599D0h	"A→"
#59A1Fh	"←T"	#59A6Eh	"T→"	#59ABDh	"(←"
#59B0Ch	"→)"	#59B5Bh	"(())"	#59B8Bh	"AF"
#59BB7h	"←M"	#59C06h	"M→"	#59C55h	"-()"
#59C83h	"1/()"	#59CB3h	"E()"	#59CE1h	"L()"
#59D0Fh	"L*"	#59D3Bh	"E^"	#59D67h	"→()"
#59DB8h	"←D"	#59E07h	"D→"	#59E56h	"→TRG"
#59E86h	"→()"	#59ED7h	"→DEF"	#59F07h	"TRG*"
#65150h	"]"	#6515Ch	"["	#6516Ah	"["
#65176h	"{"	#65182h	"}"	#6518Eh	"#"
#6519Ah	" "	#651A6h	"$"	#651B2h	"&"
#651BEh	" "	#651CAh	"»"	#651D6h	"«"
#651E2h	"E"	#651EEh	"\128"	#651FAh	"Σ"
#65206h	"|"	#65212h	14 espaces	#65238h	"\10"
#65244h	"der"	#65254h	" "	#65260h	"UNKNOWN"
#65278h	"""	#65284h	"'"	#65290h	","
#6529Ch	"."	#652A8h	";"	#652B4h	"("
#652C0h	")"	#652CCh	"^"	#652D8h	"*"
#652E4h	"/"	#652F0h	"+"	#652FCh	"-"
#65308h	"="	#65314h	"√"	#65320h	"\136"
#6532Ch	"GROB"	#6533Eh	"C$"	#6534Ch	"0"
#65358h	"1"	#65364h	"2"	#65370h	"3"
#6537Ch	"4"	#65388h	"5"	#65394h	"6"
#653A0h	"7"	#653ACh	"8"	#653B8h	"9"
#656C5h	"R\128\128"	#656D5h	"R\128Z"	#656E5h	"XYZ"
#656F5h	"«»"	#65703h	"{}"	#65711h	"[]"
#6571Fh	"''"	#6572Dh	"::"	#6573Bh	"()"
#65749h	""""	#65757h	"ECHO"	#65769h	"EXIT"
#6577Bh	"Undefined"	#65797h	"RAD"	#657A7h	"GRAD"
#67365h	"d"	#69692h	" RATIO"	#6A577h	"RULES"
#6A59Fh	"EDIT"	#6A5C0h	"EXPR"	#6A5E1h	"SUB"
#6A600h	"REPL"	#6B413h	"NOT"	#6D1C1h	" "
#7A953h	"=="	#2970Ah	"Invalid Expression"		

CARACTERES

Les "*caractères*" sont des objets auquels l'utilisateur n'a en principe pas accès, mais qui sont utilisés de manière assez fréquente par la HP48. Ils représentent un caractère particulier de la calculatrice. Ils ne sont pas affichables "*en clair*" à l'écran. La présence d'un tel objet sur la pile se traduit par le mot générique **Character**. L'instruction **TYPE**, appliquée à un "*caractère*", donne 24.

Les caractères sont codés sur 7 quartets:
- 5 quartets pour l'adresse-préfixe **#029BFh**
- 2 quartets (donc un octet) pour le code de ce caractère, compris entre 0 et 255.

Le caractère de code #2Ah=42 (c-à-d le signe *) est donc codé: **FB920A2**
N.B: Un "caractère" = 7 quartets; une chaîne de 1 caractère = 12 quartets.

Dans le tableau ci-dessous, les caractères ont été représentés par:
- Ou bien $ suivi de l'affichage "en clair" du caractère concerné.
- Ou bien \ suivi du code (en décimal) de ce caractère.
- Ou bien $spc pour désigner le caractère "espace".

Attention: n'utilisez pas SYSEVAL pour les adresses > #70000h.

#0D318h	$spc	#0D333h	$spc	#0FA62h	$k	#450BEh	\183	#45174h	$-
#6541Eh	\0	#65425h	\31						
#6542Ch	$"	#65433h	$#	#6543Ah	$*	#65441h	$+	#65448h	$,
#6544Fh	$-	#65456h	$.	#6545Dh	$/	#65464h	$0	#6546Bh	$1
#65472h	$2	#65479h	$3	#65480h	$4	#65487h	$5	#6548Eh	$6
#65495h	$7	#6549Ch	$8	#654A3h	$9	#654AAh	$:	#654B1h	$;
#654B8h	$<	#654BFh	$=	#654C6h	$>	#654CDh	$A	#654D4h	$B
#654DBh	$C	#654E2h	$D	#654E9h	$E	#654F0h	$F	#654F7h	$G
#654FEh	$H	#65505h	$I	#6550Ch	$J	#65513h	$K	#6551Ah	$L
#65521h	$M	#65528h	$N	#6552Fh	$O	#65536h	$P	#6553Dh	$Q
#65544h	$R	#6554Bh	$S	#65552h	$T	#65559h	$U	#65560h	$V
#65567h	$W	#6556Eh	$X	#65575h	$Y	#6557Ch	$Z	#65583h	$a
#6558Ah	$b	#65591h	$c	#65598h	$d	#6559Fh	$e	#655A6h	$f
#655ADh	$g	#655B4h	$h	#655BBh	$i	#655C2h	$j	#655C9h	$k
#655D0h	$l	#655D7h	$m	#655DEh	$n	#655E5h	$o	#655ECh	$p
#655F3h	$q	#655FAh	$r	#65601h	$s	#65608h	$t	#6560Fh	$u
#65616h	$v	#6561Dh	$w	#65624h	$x	#6562Bh	$y	#65632h	$z
#65639h	\141	#65640h	\171	#65647h	\187	#6564Eh	\128	#65655h	\136
#6565Ch	\132	#65663h	$(	#6566Ah	\10	#65671h	\135	#65678h	$)
#6567Fh	\133	#65686h	$spc	#6568Dh	$_	#65694h	$[	#6569Bh	$]
#656A2h	${	#656A9h	$}	#656B0h	\137	#656B7h	\138	#656BEh	\139
#6947Ch	$'	#69483h	$@	#6D2B1h	$?	#7A929h	\140	#7A930h	$%
#7A937h	$'	#7A93Eh	\137	#7A945h	\138	#7A94Ch	\139	#7A961h	$^
#7A968h	\223	#7A96Fh	\155	#7A976h	\146	#7A97Dh	\147	#7A984h	\149
#7A98Bh	\145	#7A992h	\148	#7A999h	\161	#7A9A0h	\159	#7A9A7h	$|
#7A9AEh	\144	#7A9B5h	\150	#7A9BCh	\181	#7A9C3h	\134	#7A9CAh	\157
#7A9D1h	\142	#7A9D8h	\143	#7A9DFh	\151	#7A9E6h	\152	#7A9EDh	\153
#7A9F4h	\191	#7A9FBh	\154	#7AA02h	\129	#7AA09h	\177	#7AA10h	\156
#7AA17h	\162	#7AA1Eh	\163	#7AA25h	\165	#7AA2Ch	\164	#7AA33h	\176
#7AA3Ah	$!	#7AA41h	$?	#7AA48h	$&	#7AA4Fh	$@	#7AA56h	$$

TABLEAUX

Le codage d'un tableau dans la HP48 prend la forme suivante, où les 5 premiers quartets représentent (sous la forme retournée habituelle), l'adresse-préfixe **#029E8h**:

8E920	Taille	Type	Nidx	Idx1	Idx2	....	IdxN	Éléments

, où:

- "*Taille*", "*Type*", ... ,"*IdxN*" sont des champs sur 5 octets, donnant les caractéristiques du tableau. Ils sont codés en hexadécimal, leur valeur étant écrite "à l'envers" (celà vous étonne-t-il encore ?):
- "*Taille*" est la taille totale de l'objet, exprimée en quartets, l'adresse préfixe exceptée.
- "*Type*" est l'adresse-préfixe correspondant aux objets du tableau. C'est le même type pour tous les éléments de celui-ci.
- "*Ndix*" est le nombre d'indices du tableau.
- "*Idx1*" est la valeur maximum du premier indice.

.....

- "*IdxN*" est la valeur maximum du N-ème et dernier indice.
- "*Éléments*" est la séquence des objets figurant dans le tableau. Ces éléments sont rangés dans l'ordre lexicographique croissant des N indices. Ils apparaissent codés sans leur adresse-préfixe (qui est donnée une fois pour toutes au début de la structure).

  N.B: L'ordre lexicographique croissant est l'ordre du dictionnaire, si on imagine les valeurs des indices successifs comme formant un "*mot*".

Remarque:

La taille d'un tableau, telle que donnée par l'instruction **BYTES**, est exprimée en octets, et elle prend en compte les 5 quartets de l'adresse-préfixe #029E8h du tableau.

Si on reprend les notations ci-dessus, et en appelant **Size** le résultat donné par l'instruction **BYTES** (Rappelons que "*Taille*" est exprimé en quartets, et que "*Size*" l'est en octets):

- Size = (5+Taille)/2
- Taille = 5*(N+3) + (Idx1*Idx2*...*IdxN)*T, où T est la taille, exprimée en quartets (sans tenir compte de l'adresse-préfixe), d'un élément du tableau.

Exemple:

Considérons une matrice de réels, de dimension 3*4:

N=2 (2 dimensions), T=16 (16 quartets pour coder un réel), Idx1=3, et Idx2=4.

Donc: Taille=5*5+3*4*16=217 quartets, et Size = (5+217)/2 = 111 octets (c'est bien ce que l'on obtient en éxécutant **BYTES** sur une telle matrice réelle).

Le codage d'une telle matrice en mémoire commencerait par:

8E920	9D000	33920	20000	30000	40000	....(#D9h=217 quartets)

En utilisation "*normale*" de la HP48, on ne rencontre que des tableaux de nombres réels ou complexes, à un indice (*vecteurs*) ou deux indices (*matrices*).

En fait, la structure d'un tableau en mémoire est telle qu'on peut (en théorie) y placer des objets d'un type quelconque (on peut ainsi imaginer des tableaux d'entiers-systèmes, ou de chaînes de caractères), et que le nombre de dimensions du tableau peut être supérieur à 2.

Les seuls tableaux présents dans la ROM de la HP48 sont situés dans la ROM "*cachée*", c'est-à-dire entre les adresses #70000h et #7FFFFh.

Ce ne sont que des vecteurs de chaînes de caractères (messages divers de la HP48) ou d'entiers-système (adresses de traitement des touches du clavier).

◆ Une expérience étonnante:

- Placez l'adresse #72000h sur la pile.
- Faites #5A03h SYSEVAL pour obtenir l'entier système <72000h>. C'est l'adresse du tableau des messages d'erreur N°1 à 16.
- Faites #C612h SYSEVAL pour placer le contenu de ce tableau sur la pile (on est en ROM cachée: n'appliquez SYSEVAL à une adresse en binaire supérieure ou égale à #70000h !!).
- Vous devez alors observer, au niveau 1:

```
1: | Array of String |        ETONNANT, NON ?
```

- Encouragé par cette situation, faites 1 GET: Vous devez trouver: (-6.675736649E2;,-2.07466656963E-304) .....!

Il est donc possible, en rusant un peu, de placer un tableau non *orthodoxe* sur la pile. Mais l'instruction **GET** ne semble que reconnaître les tableaux réels ou complexes (cette instruction lit l'adresse-préfixe qui, dans le tableau, détermine le type des objets: si elle ne trouve pas l'adresse-préfixe des réels, elle en conclut sans autre vérification qu'il s'agit d'un tableau complexe !).

Remarque: en remplaçant ci-dessus #72000h par #7B4E4h, on trouve:

```
1: | Array of System Binary |
```

Un conseil: N'essayez pas d'éditer ce tableau dans **Matrix-Writer**...

Avec ce tableau d'entiers-système sur la pile, l'instruction **SIZE** renvoie le résultat { 49 }, nous apprenant donc qu'il s'agit d'un vecteur de 49 entiers-système (mais toujours pas moyen d'utiliser **GET** (Faites **OBJ→** et vous verrez apparaître 49 nombres complexes idiots....).

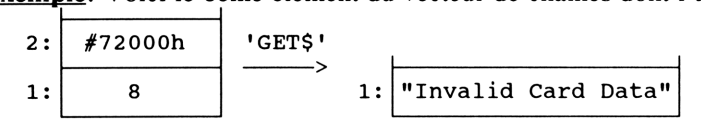
Mais puisque c'est vous, je vais vous livrer un programme permettant d'accéder à un élément donné d'un de ces tableaux mystérieux:

Le programme '**GET$**' attend:
- Un nombre binaire au niveau 2, égal à l'adresse du tableau.
- Un réel au niveau 1 égal à la position de l'élément cherché.

Il renvoie l'élément du tableau.

'**GET$**' : (Checksum #DAABh; Taille 72.5 octets)
```
« #2EC11h SYSEVAL SWAP #5A03h SYSEVAL
    #C4ECh SYSEVAL DROP NEWOB
»
```

Exemple: Voici le 8ème élément du vecteur de chaînes dont l'adresse est #72000h:



◆ LES VECTEURS DE MESSAGES:

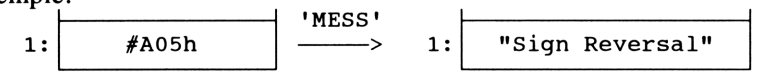
Voici maintenant les différents vecteurs de chaînes de caractères qui apparaissent dans la ROM de la HP48. Chacun contient les messages relatifs à un sujet particulier.

Chaque message de la HP48 a un numéro binaire (précisé dans les tableaux ci-dessous). Les messages d'un même vecteur ont des numéros binaires consécutifs (une liste partielle de ces messages est donnée dans le volume 2 du manuel de l'utilisateur, pages 703 et suivantes).

N.B: Le programme suivant donne la chaîne correspondant à un message précisé par son numéro binaire au niveau 1 de la pile:

'**MESS**': (Checksum #CCFCEh; Taille 49.5 octets)
```
« #5A03h SYSEVAL #4D64h SYSEVAL »
```

Par exemple:



Adresse	Nombre	Contenu
#72000h	16	Messages N° #001h à #010h

1	#001h	"Insufficient Memory"	9	#009h	"Object In Use"
2	#002h	"Directory Recursion"	10	#00Ah	"Port Not Available"
3	#003h	"Undefined Local Name"	11	#00Bh	"No Room in Port"
4	#004h	"Undefined XLIB Name"	12	#00Ch	"Object Not in Port"
5	#005h	"Memory Clear"	13	#00Dh	"Recovering Memory"
6	#006h	"Power Lost"	14	#00Eh	"Try To Recover Memory?"
7	#007h	"Warning:"	15	#00Fh	"Replace RAM, Press ON"
8	#008h	"Invalid Card Data"	16	#010h	"No Mem To Config All"

Adresse	Nombre	Contenu
#72281h	6	Messages N° #A01h à #A06h

1	#A01h	"Bad Guess(es)"	4	#A04h	"Zero"
2	#A02h	"Constant?"	5	#A05h	"Sign Reversal"
3	#A03h	"Interrupted"	6	#A06h	"Extremum"

Adresse	Nombre	Contenu
#7260Ah	4	Messages N° #D01h à #D04h

1	#D01h	"Invalid Date"	3	#D03h	"Invalid Repeat"
2	#D02h	"Invalid Time"	4	#D04h	"Nonexistent Alarm"

Adresse	Nombre	Contenu
#726A5h	2	Messages N° #B01h à #B02h

1	#B01h	"Invalid Unit"	2	#B02h	"Inconsistent Units"

Adresse	Nombre	Contenu
#7232Ch	23	Messages N° #C01h à #C17h

1	#C01h	"Bad Packet Block Check"	13	#C0Dh	"Sending"
2	#C02h	"Timeout"	14	#C0Eh	"Receiving"
3	#C03h	"Receive Error"	15	#C0Fh	"Object Discarded"
4	#C04h	"Receive Buffer Overrun"	16	#C10h	"Packet #"
5	#C05h	"Parity Error"	17	#C11h	"Processing Command"
6	#C06h	"Transfer Failed"	18	#C12h	"Invalid IOPAR"
7	#C07h	"Protocol Error"	19	#C13h	"Invalid PRTPAR"
8	#C08h	"Invalid Server Cmd."	20	#C14h	"Low Battery"
9	#C09h	"Port Closed"	21	#C15h	"Empty Stack"
10	#C0Ah	"Connecting"	22	#C16h	"Row"
11	#C0Bh	"Retry #"	23	#C17h	"Invalid Name"
12	#C0Ch	"Awaiting Server Cmd."			

Adresse	Nombre	Contenu
#72DCFh	8	Messages N° #201h à #208h

1	#201h	"Too Few Arguments"	5	#205h	"LASTARG Disabled"
2	#202h	"Bad Argument Type"	6	#206h	"Incomplete Subexpression"
3	#203h	"Bad Argument Value"	7	#207h	"Implicit () off"
4	#204h	"Undefined Name"	8	#208h	"Implicit () on"

Adresse	Nombre	Contenu
#72704h	62	Messages N° #101h à #13Dh

1	#101h	"No Room to Save Stack"	32	#120h	"Code"
2	#102h	"Can't Edit Null Char."	33	#121h	"Library Data"
3	#103h	"Invalid User Function"	34	#122h	"External"
4	#104h	"No Current Equation"	35	#123h	""
5	#105h	""	36	#124h	"LAST STACK Disabled"
6	#106h	"Invalid Syntax"	37	#125h	"LAST CMD Disabled"
7	#107h	"Real Number"	38	#126h	"HALT Not Allowed"
8	#108h	"Complex Number"	39	#127h	"Array"
9	#109h	"String"	40	#128h	"Wrong Argument Count"
10	#10Ah	"Real Array"	41	#129h	"Circular Reference"
11	#10Bh	"Complex Array"	42	#12Ah	"Directory Not Allowed
12	#10Ch	"List"	43	#12Bh	"Non-Empty Directory"
13	#10Dh	"Global Name"	44	#12Ch	"Invalid Definition"
14	#10Eh	"Local Name"	45	#12Dh	"Missing Library"
15	#10Fh	"Program"	46	#12Dh	"Invalid PPAR"
16	#110h	"Algebraic"	47	#12Eh	"Non-Real Result"
17	#111h	"Binary Integer"	48	#12Fh	"Unable to Isolate"
18	#112h	"Graphic"	49	#130h	"No Room to Show Sack"
19	#113h	"Tagged"	50	#131h	"Warning:"
20	#114h	"Unit"	51	#132h	"Error:"
21	#115h	"XLIB Name"	52	#133h	"Purge?"
22	#116h	"Directory"	53	#134h	"Out of Memory"
23	#117h	"Library"	54	#135h	"Stack"
24	#118h	"Backup"	55	#136h	"Last Stack"
25	#119h	"Function"	56	#137h	"Last Commands"
26	#11Ah	"Command"	57	#138h	"Key Assignments"
27	#11Bh	"System Binary"	58	#139h	"Alarms"
28	#11Ch	"Long Real"	59	#13Ah	"Last Arguments"
29	#11Dh	"Long Complex"	60	#13Bh	"Name Conflict"
30	#11Eh	"Linked Array"	61	#13Ch	"Command Line"
31	#11Fh	"Character"	62	#13Dh	""

Adresse	Nombre	Contenu
#72F1Eh	5	Messages N° #301h à #305h

1	#301h	"Positive Underflow"	3	#303h	"Overflow"
2	#302h	"Negative Underflow"	4	#304h	"Undefined Result"
			5	#305h	"Infinite Result"

Adresse	Nombre	Contenu
#736F9h	6	Messages N° #501h à #506h

1	#501h	"Invalid Dimension"	4	#504h	"Deleting Column"
2	#502h	"Invalid Array Element"	5	#505h	"Inserting Row"
3	#503h	"Deleting Row"	6	#506h	"Inserting Column"

Adresse	Nombre	Contenu
#72FE6h	46	Messages N° #601h à #62Eh

1	#601h	"Invalid Σ Data"
2	#602h	"Nonexistent ΣDAT"
3	#603h	"Insufficient ΣData"
4	#604h	"Invalid ΣPAR"
5	#605h	"Invalid ΣData LN(Neg)"
6	#606h	"Invalid ΣData LN(0)"
7	#607h	"Invalid EQ"
8	#608h	"Current equation:"
9	#609h	"No current equation."
10	#60Ah	"Enter eqn, press NEW"
11	#60Bh	"Name the equation, press ENTER"
12	#60Ch	"Select plot type"
13	#60Dh	"Empty catalog"
14	#60Eh	"undefined"
15	#60Fh	"No stat data to plot"
16	#610h	"Autoscaling"
17	#611h	"Solving for"
18	#612h	"No current data. Enter"
19	#613h	"data point, press Σ+"
20	#614h	"Select a model"
21	#615h	"No alarms pending."
22	#616h	"Press ALRM to create"
23	#617h	"Next alarm:"
24	#618h	"Past due alarm:"
25	#619h	"Acknowledged"
26	#61Ah	"Enter alarm, press SET"
27	#61Bh	"Select repeat interval"
28	#61Ch	"I/O setup menu"
29	#61Dh	"Plot type:"
30	#61Eh	""
31	#61Fh	"(OFF SCREEN)"
32	#620h	"Invalid PTYPE"
33	#621h	"Name the stat data, press ENTER"
34	#622h	"Enter value (zoom out if >1), press ENTER"
35	#623h	"Copied to stack"
36	#624h	"x axis zoom w/AUTO."
37	#625h	"x axis zoom."
38	#626h	"y axis zoom."
39	#627h	"x and y axis zoom."
40	#628h	"IR/wire: "
41	#629h	"ASCII/binary: "
42	#62Ah	"baud: "
43	#62Bh	"parity: "
44	#62Ch	"checksum type: "
45	#62Dh	"translate code:"
46	#62Eh	"Enter matrix, then NEW"

◆ LES VECTEURS D'ENTIERS-SYSTEME:

Venons-en enfin aux vecteurs d'entiers-système qui figurent dans la ROM (cachée) de la HP48. Il y a 50 tels tableaux, et 50=1+49, comme nous allons le voir.

Le tableau situé à l'adresse **#7B4E4h** à une importance considérable dans le fonctionnement de la calculatrice. Il contient les 49 adresses des 49 autres tableaux.

Chacun de ces 49 tableaux correspond à une des 49 touches de la HP48: il contient les six adresses renvoyant aux 6 fonctions possibles d'une même touche, selon que cette touche est actionnée (dans cet ordre):

1) non shiftée (plan 1)
2) après <¬ (plan 2)
3) après ⌐> (plan 3)
4) après α (plan 4)
5) après α <¬ (plan 5)
6) après α ⌐> (plan 6)

Adresse: #7B4E4h. Vecteur de 49 entiers-système.

<7AA5Dh> <7AA94h> <7AACBh> <7AB02h> <7AB39h> <7AB70h> <7ABA7h>
<7ABDEh> <7AC15h> <7AC4Ch> <7AC83h> <7ACBAh> <7ACF1h> <7AD28h>
<7AD5Fh> <7AD96h> <7ADCDh> <7AE04h> <7AE3Bh> <7AE72h> <7AEA9h>
<7AEE0h> <7AF17h> <7AF4Eh> <7AF85h> <7AFBCh> <7AFF3h> <7B02Ah>
<7B061h> <7B098h> <7B0CFh> <7B106h> <7B13Dh> <7B174h> <7B1ABh>
<7B1E2h> <7B219h> <7B250h> <7B287h> <7B2BEh> <7B2F5h> <7B32Ch>
<7B363h> <7B39Ah> <7B3D1h> <7B408h> <7B43Fh> <7B476h> <7B4ADh>

Il faut lire le tableau ci-dessus de la manière suivante:

- Le premier entier-système **<7AA5Dh>** pointe sur le tableau contenant les 6 adresses de traitement associées à la touche N°1 du clavier (touche "A" du menu).
- Le deuxième entier-système **<7AA94h>** pointe sur le tableau associé aux 6 fonctions de la touche N°2 (touche "B" du menu).
- Enfin le dernier entier-système **<7B4ADh>** pointe sur le tableau qui correspond aux 6 fonctions de la touche N°49 (touche "+").

Nous allons maintenant passer en revue les 49 tableaux évoqués ci-dessus, dans l'ordre naturel des touches de la HP48 (sur une ligne de gauche à droite, et de la ligne la plus haute à la ligne la plus basse).

Il convient de savoir que cet ordre n'est pas celui utilisé par l'instruction **KEY** (qui identifie une touche de la ligne L, colonne C, par l'entier 10*L+C). Par exemple, **KEY** associe désigne "7" avec l'entier 62 (6-ème ligne et 2-ème colonne) alors que nous considérons ici que "7" est la 32-ème touche du clavier.

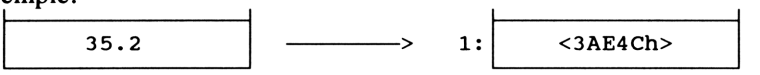
Avant cette énumération de 49 tableaux (qui est une véritable mine d'adresses passionnantes pour les chercheurs de SYSEVALs), je vous donne le listing du programme '**GETK**' qui:

- Prend au niveau 1 le nombre décimal permettant de désigner une touche et ses shifts éventuels (l'instruction **0 WAIT** attend qu'une touche soit appuyée et donne une tel nombre).
  Par exemple la séquence α <¬ 7 correspond au nombre 62.5
- Place sur la pile l'entier-système correspondant à l'adresse de traitement de cette touche.

'**GETK**': (Cheksum: #9940h, Taille 95.5 octets)

```
« #41CA2h SYSEVAL SWAP #7B4E4h #5A03h SYSEVAL
  1 2 START #C4ECh SYSEVAL DROP NEWOB NEXT
»
```

Exemple:



On trouve ainsi l'adresse de traitement de **REVIEW** (<¬ ∇)

Dans le tableau ci-dessous figurent, ligne par ligne:

- L'adresse du tableau de 6 entiers-système correspondant à une touche donnée.
- Le numéro de cette touche sur le clavier (de 1 à 49).
- Les 6 adresses renvoyant au traitement de cette touche, sur les 6 différents plans du clavier (du plan 1: non shifté, au plan 6 qui correspond au shift α ┌>).

Adresse	Key	Adresses associées à la touche N° Key					
#7AA5Dh	1	<3FFDBh>	<3FFF4h>	<4000Dh>	<654CDh>	<65583h>	<7A929h>
#7AA94h	2	<4003Fh>	<40026h>	<40058h>	<654D4h>	<6558Ah>	<7A968h>
#7AACBh	3	<40071h>	<4008Ah>	<400A3h>	<654DBh>	<65591h>	<7A96Fh>
#7AB02h	4	<400BCh>	<400D5h>	<400EEh>	<654E2h>	<65598h>	<7A976h>
#7AB39h	5	<40107h>	<40120h>	<40139h>	<654E9h>	<6559Fh>	<7A97Dh>
#7AB70h	6	<40152h>	<4016Bh>	<40184h>	<654F0h>	<655A6h>	<7A984h>
#7ABA7h	7	<3AD57h>	<3AE33h>	<1EE53h>	<654F7h>	<655ADh>	<7A98Bh>
#7ABDEh	8	<3AE1Ah>	<3A6E5h>	<21FD1h>	<654FEh>	<655B4h>	<7A992h>
#7AC15h	9	<3AB72h>	<3ADA2h>	<3ADBBh>	<65505h>	<655BBh>	<7A9A0h>
#7AC4Ch	10	<3AF37h>	<3AD70h>	<3AD89h>	<6550Ch>	<655C2h>	<7A9A7h>
#7AC83h	11	<3A93Dh>	<3AE6Fh>	<3B00Eh>	<65513h>	<655C9h>	<7A9AEh>
#7ACBAh	12	<3A71Ch>	<3A735h>	<3B211h>	<6551Ah>	<655D0h>	<7A9B5h>
#7ACF1h	13	<3A69Ah>	<1A15Bh>	<1A140h>	<65521h>	<655D7h>	<7A937h>
#7AD28h	14	<3A992h>	<20D65h>	<20B40h>	<65528h>	<655DEh>	<7A9BCh>
#7AD5Fh	15	<1A3BEh>	<1F9C4h>	<1A5E4h>	<6552Fh>	<655E5h>	<7A9CAh>
#7AD96h	16	<3A834h>	<1E2BAh>	<3AFE6h>	<65536h>	<655ECh>	<7A9D1h>
#7ADCDh	17	<3A645h>	<3AE4Ch>	<3AF69h>	<6553Dh>	<655F3h>	<7A9D8h>
#7AE04h	18	<3A8DEh>	<1FBBDh>	<3A80Ch>	<65544h>	<655FAh>	<7A9DFh>
#7AE3Bh	19	<1B4ACh>	<1B6A4h>	<1EF7Eh>	<6554Bh>	<65601h>	<7A9E6h>
#7AE72h	20	<1B505h>	<1B72Fh>	<1F1D4h>	<65552h>	<65608h>	<7A9EDh>
#7AEA9h	21	<1B55Eh>	<1B79Ch>	<1F2C9h>	<65559h>	<6560Fh>	<7A930h>
#7AEE0h	22	<1B374h>	<1B426h>	<1B185h>	<65560h>	<65616h>	<7A961h>
#7AF17h	23	<1B02Dh>	<1BA3Dh>	<1B9C6h>	<65567h>	<6561Dh>	<7A9FBh>
#7AF4Eh	24	<1B278h>	<1B905h>	<1B94Fh>	<6556Eh>	<65624h>	<7AA02h>
#7AF85h	25	<3A7F3h>	<3AF50h>	<3B068h>	<3A7F3h>	<7AA48h>	<7AA4Fh>
#7AFBCh	26	<3AA82h>	<3A7C6h>	<3B12Bh>	<65575h>	<6562Bh>	<7AA09h>
#7AFF3h	27	<3AC3Ah>	<3B15Dh>	<3B18Fh>	<6557Ch>	<65632h>	<7AA10h>
#7B02Ah	28	<3AB09h>	<3B1DFh>	<3AAEBh>	<3AB09h>	<7AA3Ah>	<7A999h>
#7B061h	29	<3A5F0h>	<1FBD8h>	<1FCEBh>	<3A5F0h>	<7AA41h>	<7A9F4h>
#7B098h	30	<3AA0Ah>	<3B0DBh>	<3B081h>	<3B036h>	<3E5EEh>	<3E5AFh>
#7B0CFh	31	<65495h>	<3AE88h>	<3BE54h>	<65495h>	<3F167h>	<3F17Bh>
#7B106h	32	<6549Ch>	<3ADEDh>	<48FD6h>	<6549Ch>	<3F18Fh>	<3F1A3h>
#7B13Dh	33	<654A3h>	<3AB59h>	<47B5Ah>	<654A3h>	<3F1B7h>	<3F1CBh>
#7B174h	34	<1AF05h>	<3A6CCh>	<3AC08h>	<6545Dh>	<3A6CCh>	<65433h>
#7B1ABh	35	<3A893h>	<3A8ACh>	<3A893h>	<3AA37h>	<3AA50h>	<3AA37h>
#7B1E2h	36	<65480h>	<3A7A3h>	<47AE7h>	<65480h>	<7AA56h>	<7AA17h>
#7B219h	37	<65487h>	<3AEB5h>	<3AEE2h>	<65487h>	<7AA1Eh>	<7AA25h>
#7B250h	38	<6548Eh>	<3AF05h>	<3AF1Eh>	<6548Eh>	<7AA2Ch>	<7AA33h>
#7B287h	39	<1ADEEh>	<3ABA4h>	<3A6B3h>	<6543Ah>	<3ABA4h>	<6568Dh>
#7B2BEh	40	<3A8C5h>	<3A8C5h>	<3A8ACh>	<3AA69h>	<3AA69h>	<3AA50h>
#7B2F5h	41	<6546Bh>	<3B1CBh>	<3B1B7h>	<6546Bh>	<7A953h>	<7A94Ch>
#7B32Ch	42	<65472h>	<3ACF8h>	<1A604h>	<65472h>	<654B8h>	<654C6h>
#7B363h	43	<65479h>	<3ACBCh>	<3A6FEh>	<65479h>	<7A93Eh>	<7A945h>
#7B39Ah	44	<1AD09h>	<3ABD6h>	<3AC21h>	<6544Fh>	<3ABBDh>	<6542Ch>
#7B3D1h	45	<3A5CDh>	<1A8BBh>	<3A9CEh>	<3A5CDh>	<1A8BBh>	<3A9CEh>
#7B408h	46	<65464h>	<1A8D8h>	<22FEBh>	<65464h>	<654BFh>	<65639h>
#7B43Fh	47	<3A753h>	<3A77Bh>	<3ADD4h>	<3A753h>	<3A77Bh>	<3ADD4h>
#7B476h	48	<65686h>	<1AABDh>	<6564Eh>	<65686h>	<65671h>	<6564Eh>
#7B4ADh	49	<1AB67h>	<3AB8Bh>	<3ABEFh>	<65441h>	<3AB8Bh>	<654AAh>

NOMS GLOBAUX ET NOMS LOCAUX

La HP48 connait deux sortes de noms: les *noms **globaux*** et les *noms **locaux***.

Les noms globaux sont ceux par lesquels l'utilisateur référence les objets qu'il stocke dans un menu **VAR** (ou dans le menu **CST**, ou dans les PORTS 0, 1, ou 2....).
Ce sont dans ce cas des noms de variables, et leur évaluation provoque l'évaluation du contenu de cette variable.

Les noms globaux servent également de *variable formelle* dans les expressions algébriques (ils ne référencent alors aucun contenu).

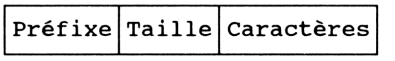
Quand vous entrez un nom au clavier, et que ce nom n'existe pas, il est créé comme nom global et placé (entre deux cotes ') sur la pile.

Les noms locaux, au contraire, référencent toujours un contenu. Ce sont donc toujours des noms de variables locales.
Ces variables locales peuvent être créées par l'utilisateur au début d'une structure locale, et leur durée de vie est limitée à l'éxécution de cette structure.

Rappelons que l'instruction **TYPE** attribue l'entier 6 aux noms globaux et 7 aux noms locaux.

Dans la mémoire de la HP48, les noms globaux et locaux de la HP48 sont codés de la même manière, à l'exception des adresses-préfixes, qui sont différentes:

- L'adresse-préfixe des noms globaux est: **#02E48h**
- L'adresse-préfixe des noms globaux est: **#02E6Dh**
- La structure commune est la suivante:



"Taille" est le nombre de caractères que possède le nom (de 0 à 255). Ce nombre est codé sur 1 octet.
Les caractères apparaissent ensuite, dans l'ordre naturel du nom. Ils sont représentés par leur code (de 0 à 255=#FFh), qui est le code **ASCII** pour les caractères de code<128, mais qui est un code parfois spécifique à la HP48 au-delà (voir instructions **CHR** et **NUM**).
Ces codes sont écrits sur un octet.

Evidemment, ces différents octets (la taille puis le code de chacun des caractères) sont écrits à l'envers.

Ceux qui ne s'en doutaient pas viennent sans d'outre d'ouvrir ce livre pour la première fois, au hasard, et à cette page.

Exemples:

- Le codage du nom global **ΣPAR** est: **84E20 40 58051425**
  L'adresse préfixe est #02E48h.
  Il y a 4 caractères.
  Les codes successifs de ces caractères sont:
  #85h=133 (Σ), #50h=80 (P), #41h=65 (A), et #52h=82 (R).

  On note ici la différence qui existe pour le caractère Σ, entre le code spécifique à HP (code=133) et le code de ce caractère dans la table IBM habituelle (code=228).

- Le codage du nom local **'PKNO** (utilisé de façon interne par la HP48 pour mesurer le numéro de "*Packet*" dans un échange par la voie série est:
  D6E20 50 7205B4E4F4
  Le caractère qui débute **'PKNO** fait partie intégrante du nom.

  De tels noms sont en principe impossibles à créer pour l'utilisateur. Cette "astuce" est utilisée souvent dans la ROM de la HP48, certainement pour éviter des "*collisions*" entre les noms créés par l'utilisateur et ceux qu'emploie la calculatrice.

- Il est possible également de considérer le *nom global vide*, codé **84E2000**, et le *nom local vide*, codé **D6E2000**. Tous deux sont d'une grande importance pour la HP48.

- Le nom global vide est utilisé dans le répertoire **HOME** pour référencer un répertoire caché dont nous reparlerons.
  Ce même nom global peut être utilisé dans d'autres répertoires que **HOME**, pour masquer tout ou partie du contenu de ce répertoire, les entrées masquées étant encore connues par leur nom.

- Le nom local vide est utilisé de manière intensive par la HP48 pour stocker provisoirement des objets.
  C'est en effet un moyen rapide (et économe en mémoire) de désigner une ou plusieurs variables locales dont le nom importe peu (on découvrira bientôt que, de façon interne, la HP48 créée souvent des groupes de variables locales aux noms vides, et qu'elle accède à ces variables par leur numéro d'ordre)

◆ LISTE DES NOMS GLOBAUX:

Voici la liste des noms globaux présents dans la ROM de la HP48. Vous constaterez que certains noms apparaissent plusieurs fois (notamment **X**).

Les noms sont donnés sans les cotes '. Le nom '' désigne ici le nom global vide; le nom **'symb** commence bien par le caractère '.

Pour chacun de ces noms, un Syseval à l'adresse correspondante a pour conséquence de l'évaluer.

Il en résulte un problème pour le nom **HOMEDIR** situé à une adresse supérieure à #70000h. Pour placer le nom '**HOMEDIR**' sur la pile, il faut éxécuter la séquence:

#7F9B5h #05A03h SYSEVAL #0C612h SYSEVAL

Plus généralement le programme suivant prend une adresse sur la pile (en binaire) et place l'objet correspondant, au niveau 1, sans l'évaluer.

'**RCLAD**' (Checksum: #DD21h; Taille 50.5 octets)

« #05A03h SYSEVAL #0C612h SYSEVAL »

Adresse	Nom Global	Adresse	Nom Global	Adresse	Nom Global
#0DF01h	Alarms	#0DF28h	Alarms	#1576Ch	EQ
#15781h	''	#19A72h	ALARMDAT	#19B1Fh	ALARMDAT
#19DBEh	ALARMDAT	#211B4h	CST	#225A4h	S
#2C1FDh	ΣDAT	#2C738h	ΣPAR	#2E9D5h	IOPAR
#2EA59h	IOPAR	#31F87h	PRTPAR	#31FB8h	PRTPAR
#34DBBh	'symb	#3FACFh	SKEY	#4093Bh	αENTER
#409DFh	ßENTER	#41A43h	UserKeys	#41A69h	UserKeys.CRC
#41BD7h	S	#4353Eh	ALG	#4358Ah	α
#435CEh	V	#47459h	X	#48D4Bh	ALARMDAT
#4A145h	X	#4A19Eh	X	#4A1DEh	X
#4A22Dh	X	#4A25Eh	X	#4AB1Ch	X
#4AB59h	Y	#50FCEh	X	#50FE6h	Y
#51288h	PPAR	#51436h	s1	#56859h	IERR
#5793Fh	n0	#5795Dh	s0	#59304h	s1
#7F9B5h	HOMEDIR				

◆ LISTE DES NOMS LOCAUX:

Voici maintenant la liste des noms locaux de la ROM. Là encore certains noms apparaissent plusieurs fois. Les caractères ' au début d'un nom font partie intégrante de celui-ci (Exception: les noms locaux vides, représentés ici par '').

Un SYSEVAL à chaque adresse a pour effet d'évaluer le nom qui lui correspond, le résultat étant une erreur "*Undefined Local Name*" si ce nom local n'existe pas déjà.

Adr	Nom local	Adr	Nom local	Adr	Nom local	Adr	Nom local
#0E47Ah	M	#0E483h	N	#0E4A0h	M	#0E4AEh	N
#0E4C1h	M	#1439Bh	'halt	#14483h	'nohalt	#1F96Fh	'num
#1F97Eh	'fcn	#2372Eh	'stop	#2373Fh	'noname	#23908h	st
#23913h	ofs	#23920h	tok	#2394Bh	st	#23956h	ofs
#23963h	tok	#24A2Dh	i	#24A36h	j	#24A5Dh	i
#24A6Bh	j	#24B0Ah	j	#24B1Dh	i	#24B30h	i
#24BB6h	j	#24BD3h	i	#24BE1h	i	#25A0Bh	'1
#25A16h	'2	#25A21h	'3	#25A3Bh	'1	#25A46h	'2
#25A51h	'3	#272CDh	'ttt	#272DCh	'str	#272EBh	'ofs
#272FAh	'tok	#27309h	'rbv	#27318h	'idfflg	#2732Dh	'tmpop
#27340h	'tmppdat	#27357h	'ploc	#27368h	'bv	#27375h	'unbound
#2D3A0h	'PKNO	#2D3B1h	'PACKET	#2D3C6h	'RETRY	#2D3D9h	'ERRMSG
#2D3EEh	'KP	#2D3FBh	'LNAME	#2D40Eh	'OBJ	#2D41Dh	'OPOS
#2D42Eh	'EXCHP	#2D45Ah	'KLIST	#2D46Dh	'KMODE	#2D480h	'KPTRN
#2D493h	'KRM	#2D4A2h	'MaxR	#2F211h	'KML	#2F46Eh	'KEOF
#31C37h	'IWrap	#34D30h	''	#36BF6h	#a	#36C01h	#b
#36C2Fh	#b	#36C3Fh	#a	#36CEFh	#b	#36D18h	#b
#38A3Eh	'SavedUI	#3FAE8h	SKEY	#41BEAh	S	#43555h	ALG
#4359Dh	α	#435E1h	V	#4C944h	'xmax	#4C955h	'N
#4CF55h	'EnvOK	#4CF68h	'EXITFCN	#4D30Dh	'EnvOK	#4D352h	'EnvOK
#4D36Fh	'EnvOK	#4FF9Dh	'xe	#4FFAAh	'ye	#4FFB7h	'x
#4FFC2h	'y	#4FFCDh	'xc	#4FFDAh	'yc	#4FFE7h	'r2
#4FFF4h	'left	#50005h	'up	#50012h	'exit	#50D3Eh	'PlotEnv
#5190Bh	'PlotEnv	#5456Fh	'tcls	#54580h	'fcls	#5460Eh	'tcls
#54624h	'fcls	#5465Dh	'tcls	#5466Eh	'fcls	#549DBh	'dvar
#54DD0h	'xSYMfcn	#54DE7h	'xfcn	#5566Bh	'oth	#55783h	'scl
#55792h	'xSYMfcn	#557A9h	'xfcn	#55800h	'xSYMfcn	#55817h	'xfcn
#56976h	'sumexpr	#5698Dh	'sumvar	#56F0Bh	'dv	#5720Bh	'nm
#57218h	'op	#578E2h	'ni	#578EFh	'ns	#57EF3h	'*s
#58149h	'+s	#58DB6h	'fl	#59115h	'nmls	#592C0h	'c
#592CBh	'b	#592D6h	'a	#59517h	'n	#59522h	'prog
#59646h	'n	#5A614h	'piflag	#5A665h	'd	#5A670h	'r
#5A761h	'd	#5A76Ch	'R	#5A777h	'est	#5A786h	'X
#5A791h	'T	#5AAE5h	'bnds	#5D67Dh	'which	#5D690h	'op1
#5D69Fh	'op2	#5FDC1h	'ct	#5FDCEh	'pp	#5FDDBh	'ep
#6080Bh	'reg	#6081Ah	'sur	#60829h	'cts	#60838h	'sun
#60847h	'mlg	#60856h	'ckd	#60865h	'prd	#60874h	'prp
#60883h	'rhs	#60C04h	'compos	#60C19h	'varls	#60C6Ah	'compos
#60CCAh	'compos	#60D7Fh	'varls	#60E8Ch	&1	#60E97h	&2
#60EA2h	&3	#60EADh	&4	#61D3Ah	''	#69A97h	'Radix
#69AAAh	'KeysOK?	#69AC1h	'ExprLit	#69AD8h	'BuffW	#69AEBh	'BuffH
#69B19h	'ManOp	#69B2Ch	'nohalt	#69B41h	'AppMode	#69B58h	'NameGrob
#69B71h	'EXITFCN	#69BA3h	'LE	#69BB0h	'LB	#69BBDh	'TE
#69BE5h	'prow	#69BF6h	'pcol	#69C07h	'cursy	#69C1Ah	'cursx
#69C2Dh	'ttt	#69C3Ch	'source	#69C51h	'ofs	#69C60h	'tok
#69C6Fh	'rbv	#69C7Eh	'idfflg	#69C93h	'tmpop	#69CA6h	'tmppdat
#69CBDh	'ploc	#69CCEh	'bv	#69CDBh	'unbound	#7FED7h	'A
#2387Eh	'ioinprogress	#60BE9h	'patternls	#60C4Fh	'patternls		
#69AFEh	'SaveBlank	#69B88h	'FontGauge	#69BCAh	'FormEnvOK		

LE RÉPERTOIRE "CACHÉ"

La HP48 gère, de façon "transparente" pour l'utilisateur, un répertoire caché où elle stocke la liste des alarmes et celle des touches *réassignées* (par l'instruction **ASN**).

Le nom de ce répertoire caché est le *nom global vide*, dont l'adresse a été donnée dans le chapitre précédent (**#15781h**).

L'instruction **#15781h SYSEVAL** a donc pour effet d'évaluer le nom global vide, et ainsi de faire du répertoire caché le répertoire courant.

On constate alors un certain nombre de choses, tant que l'on reste dans ce répertoire "*caché*":

- Le chemin courant est **HOME**.

- Il apparait trois variables dont les noms sont:
  'Alarms', **'UserKeys'**, et **'UserKeys.CRC'**.

- Si on passe dans un des menus intégrés à la HP48, un appui sur la touche **VAR** nous ramène aux trois entrées du menu caché.

- La HP48 ne reconnait plus les noms quand ils sont passés via la ligne de commande: les variables de votre répertoire **HOME** sont inconnues. Même les noms des fonctions ou commandes intégrées à la HP48 semblent ignorées!
  Si on tape par exemple **4 INV** (puis **ENTER**) on trouve **4** au niveau 2 et le nom **INV** au niveau 1....
  On peut cependant encore utiliser les fonctions et commandes intégrées de la HP48 à condition d'y accéder par la touche du clavier qui leur correspond ou par les entrées des différents menus intégrés.

- On peut créer, à l'intérieur de ce répertoire caché, des variables par **STO** et les effacer par **PURGE** (mais ne touchez cependant pas au contenu des 3 variables **'Alarms'**, **'UserKeys'**, et **'UserKeys.CRC'**.....)

- On peut même créer un menu personnalisé par **CST** (qui reste attaché au *répertoire caché*), ou encore créer des sous-répertoires du *répertoire caché*....

- Pour sortir du *répertoire caché*, il faut par exemple actionner la touche **HOME** (ou la touche **UPDIR**).

Après cet état des lieux, intéressons-nous au contenu des trois variables **'Alarms'**, **'UserKeys'**, et **'UserKeys.CRC'**.

◆ LA VARIABLE 'Alarms':

Elle contient la liste des alarmes à venir (ou déjà passées) mais dont on n'a pas accusé réception (qui n'ont pas été "*Acknowledged*"), ce que l'on appelle les "*Past due Alarms*").

Si aucune alarme n'est en attente, la liste est donc vide. Il en est ainsi après toute réinitialisation de la calculatrice.

Dans le cas où une ou plusieurs alarmes sont en attente, le contenu de '**Alarms**' peut être schématisé de la façon suivante:

{ Alarm1 Alarm2 ..... AlarmN },

où *Alarm1*, *Alarm2*, ... *AlarmN* sont des listes décrivant, dans l'ordre chronologique, les différentes alarmes (passées et non "honorées" ou qui restent à venir).

La liste correspondant à une alarme particulière est de la forme **{ Binaire Objet }** où "*Binaire*" est un nombre binaire de longueur 24 quartets (c'est-à-dire 96 bits, alors que la longueur habituelle des binaires est de 16 quartets, c'est-à-dire 64 bits) et "*Objet*" est l'objet que cette alarme doit éxécuter (une chaîne éventuellement vide ou un programme).

Cet entier binaire contient les différentes informations concernant le "*timing*" de l'alarme, à savoir la date et l'heure précise de déclenchement, et la périodicité éventuelle.

Exemple: une alarme devant se déclencher le 08/11/1992 à 16h34mn48s, avec une périodicité de 3 jours définit un entier binaire égal à: #00007E9000001D48E37B50000h.

En mémoire: il est codé

E4420	D1000	00005B73E84D1	000009E7000

On peut décomposer cet entier binaire en deux champs:

- Le champ **00005B73E84D1** définit l'entier binaire **#1D48E37B50000h** égal au contenu (en nombre de ticks) de l'horloge du système au moment précis où doit se déclencher l'alarme (à ce moment là, c'est l'entier binaire que donnerait l'instruction **TICKS**).
- Le champ **000009E7000** des 11 derniers quartets définit **#7E900000h**, égal à 2123366400 (ticks d'horloge entre deux déclenchements successifs de l'alarme).
- En effet, avec notre exemple:
  2123366400 = 3*707788800 = 3*24*60*60*8192, ce qui nous donne bien trois jours à raison de 8192 ticks par seconde.

◆ LA VARIABLE 'UserKeys':

La variable '**UserKeys**' du *répertoire caché* contient la liste de toutes les assignations de touches que vous avez pu effectuer (à l'aide des instructions **ASN** ou **STOKEYS**).

Si vous n'avez effectué aucune assignation, ou si vous les avez toutes annulées par la séquence **0 DELKEYS**, cette liste est vide.
Sinon cette liste est composée de 49 sous-listes de 6 objets.

Chacune de ces listes traite le cas d'une des 49 touches de la HP48 (l'ordre des 49 listes correspondant à l'ordre des touches sur le clavier quand on parcourt celui-ci de gauche à droite et de haut en bas).

Une telle liste s'écrit:
{ Objet1 Objet2 Objet3 Objet4 Objet5 Objet6 },
Où Objet1,...,Objet6 sont les objets devant être évalués en mode **USER** selon que cette touche est actionnée dans l'un des 6 plans possibles du clavier, qu'il n'est pas inutile de rappeler encore:

Les 6 différents plans du clavier	
Plan 1: Touche non shiftée	Plan 2: Touche shiftée par <┐
Plan 3: Touche shiftée par ┌>	Plan 4: Touche shiftée par α
Plan 5: Touche shiftée par α <┐	Plan 6: Touche shiftée par α ┌>

Quand une touche n'a pas été réassignée sur un ou plusieurs de ces 6 plans, l'objet correspondant est une liste vide.

Considérons un exemple:

- On veut que la séquence **α DEL** détruise l'élément placé au Nème niveau de la pile (N étant l'entier présent au niveau 1 à ce moment là, à la manière des instructions **DROPN**, **PICK**, **ROLL**, et **ROLLD**).
- On éxécute pour celà la séquence **« ROLL DROP » 54.4 ASN** (car la touche **DEL** est en 5ème colonne, 4ème ligne, et le plan **α** du clavier est le plan N°4).
- La HP48 place alors en 28ème position dans la liste contenue dans '**UserKeys**' la sous-liste:
  { { } { } { } « ROLL DROP » { } { } }.
- La touche **DEL** est en effet la 28ème touche du clavier, et les sous-listes vides dans cet exemple le sont si vous n'avez pas réassigné la touche **DEL** sur d'autres plans du clavier....

◆ LA VARIABLE 'UserKeys.CRC':

Elle contient le *Checksum* (code de redondance cyclique) correspondant à la liste contenue dans la variable '**UserKeys**'. C'est le même entier binaire que produirait l'instruction **BYTES** appliquée à cette liste.

Le contenu de '**UserKeys.CRC**' est donc actualisé à chaque assignation de touche (ou à chaque annulation d'une telle assignation).

Quand le contenu de '**UserKeys**' est vide, par exemple, le contenu de la variable '**UserKeys.CRC**' est **#75E6h**.

Remarque importante:

Il est prudent de ne pas modifier inconsidérément le contenu des trois variables '*Alarms*', '*UserKeys*', et '*UserKeys.CRC*', surtout pour les deux dernières, qui se contrôlent mutuellement.

La HP48 effectue en effet des contrôles sur ces contenus et elle peut enclencher la séquence "*Try To Recover Memory?*" si elle s'aperçoit que le contenu d'une de ces variables n'est pas conforme.

◆ MASQUER DES ENTRÉES DE MENU:

Tout d'abord deux SYSEVALS seront ici essentiels:
Pour évaluer le nom global vide: **#15781h SYSEVAL**.
Pour placer le nom vide sur la pile: **#15777h SYSEVAL**.

Le nom global vide réfère, dans **HOME**, au *répertoire caché*. Si on ne veut pas de "*Memory Clear*" à répétition, on ne "jouera" donc pas trop avec ce nom vide dans **HOME**. N'essayez pas par exemple de purger (avec **PGDIR**) le répertoire caché. Vous lanceriez invariablement la séquence redoutée "*Try To Recover Memory*" (qui ne se traduit pas forcément par une perte de données, mais tout de même...).

En revanche, on peut créer, dans chacun des sous-répertoires de **HOME**, une variable ou même un sous-répertoire dont le nom est le nom vide. La principale conséquence est la possiblité qui en résulte de masquer tout ou partie des entrées de ce sous-répertoire.

Quelques explications simples (sans vouloir rentrer dans les détails) suffiront à vous faire comprendre comment celà fonctionne.

Le contenu d'un répertoire donné est un objet particulier de la HP48. Dans cet objet figurent en séquence les noms et contenus des différentes variables de ce répertoire (l'ordre y est le contraire de celui des entrées dans le menu **VAR**, ou encore de l'ordre dans lequel les noms apparaissent dans la liste donnée par l'instruction **VARS**).

Si l'un de ces noms est le nom vide, alors toutes les variables à partir de celle-ci n'ont plus droit à une entrée dans le menu **VAR** (et l'instruction **VARS** donne une liste où ces noms ne figurent pas).

Cependant, et c'est là tout l'intérêt, toutes les variables du répertoire (y compris celle dont le nom est caché) sont encore connues de la HP48 et peuvent être évaluées (en entrant leur nom en ligne de commande par exemple, ou en utilisant ce nom dans un programme...)

Nous allons mettre ces idées en pratique. Rappelons toutefois que les programmes qui suivent ne peuvent et ne doivent pas être utilisés dans **HOME**.
On pourra les faire débuter par un test du type **IF PATH SIZE > 1 THEN** pour éviter toute utilisation maladroite. Néanmoins, vous pouvez très bien stocker ces programmes dans **HOME**, ce qui les rend visibles dans toute l'arborescence de votre HP48.

Le programme '**CACHE**' permet de cacher les variables dont le nom figure dans la liste placée au niveau 1 (les noms éventuellement erronés sont simplement ignorés). Dans le cas d'une seule variable, il n'est pas nécessaire d'inclure le nom dans une liste.

Les masquages opérés par '**CACHE**' ne sont pas *cumulatifs*. Seuls sont effectivement cachés les noms précisés lors du dernier appel de '**CACHE**'.

La séquence **VARS CACHE** masque donc la totalité du répertoire.
De même la séquence **{ } CACHE** redonne l'intégralité du répertoire (en y laissant cependant la variable de nom vide, invisible à la fin du répertoire).

'**CACHE**': (Checksum: #C3CFh; Taille: 134.5 octets)

```
« { } + → h
   « #15777h SYSEVAL PURGE VARS OBJ→ { }
      1 ROT START
         IF h 3 PICK POS
         THEN SWAP DROP
            ELSE +
         END
      NEXT
      0 #15777h SYSEVAL STO ORDER
   »
»
```

Le programme '**DECACHE**' fait réapparaître le nom figurant au niveau 1 (ou les noms de la liste au niveau 1). Les noms réapparaissent en fin de menu, ce qui n'était pas forcément leur position avant d'être cachés...

'**DECACHE**': (Checksum: #319Dh; Taille: 31.5 octets)

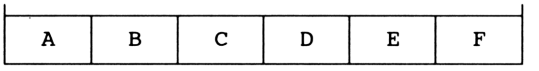
```
   « VARS SWAP + ORDER »
```

Enfin le programme '**NOCACHE**' élimine les variables éventuellement cachées dans le répertoire en cours (il ne fait que purger le nom vide):

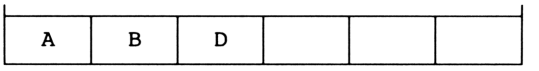
'**NOCACHE**': (Checksum: #E28Dh; Taille: 39.5 octets)
« #15777h SYSEVAL PURGE »

Exemple:

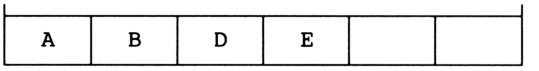
- Supposons que le répertoire en cours contienne les entrées '**A**','**B**','**C**','**D**','**E**','**F**', et que les programmes '**CACHE**', '**DECACHE**' et '**NOCACHE**' soient visibles dans ce répertoire (par exemple s'ils sont stockés dans **HOME**).
- Le menu **VAR** a au départ l'apparence suivante:



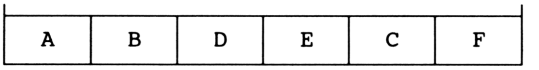
- Après **{ C E F } CACHE**, voici le nouveau menu:



- La séquence **'E' DECACHE** permet de revoir '**E**':



- Enfin **NOCACHE** démasque les variables encore cachées:



RÉPERTOIRES

Il y a deux sortes de répertoires: **HOME** et les sous-répertoires de **HOME**. Voici sur quels points ils diffèrent:

- **HOME** n'est pas nommé par un "*nom-utilisateur*".

- On ne peut rappeler le contenu de **HOME** sur la pile (à moins de connaître le bon SYSEVAL!).
- Le contenu d'un sous-répertoire de **HOME** peut être rappelé sur la pile en appliquant **RCL** au nom de ce sous-répertoire.

- Un nombre quelconque de librairies peuvent être attachées à **HOME**, alors qu'un sous-répertoire de **HOME** ne peut se voir attacher qu'une seule librairie à la fois.

- Le répertoire **HOME** contient une entrée, dont le nom est le *nom global vide*, et qui permet d'accéder à un *sous-répertoire caché* qui contient les informations décrivant les touches réassignées et les alarmes en cours.

Le codage en mémoire de ces deux types de répertoires est cependant pratiquement le même. La principale différence tient au problème des librairies attachées.

Comme tous les objets de la HP48, les répertoires sont codés en deux parties: une adresse-préfixe d'abord, puis le corps de l'objet lui-même. L'adresse-préfixe de tous les répertoires est **#02A96h**

Voici comment on peut représenter, schématiquement, le codage en mémoire d'un répertoire (**HOME** ou un de ses sous-répertoires).

La toute première colonne sert uniquement à marquer quelques repères.

Par exemple "**b**" désigne ici l'adresse où commence "**Zone_Obj1**".

Les adresses vont en augmentant ($a < b < c < \ldots < \alpha$).

(La terminologie utilisée est expliquée plus loin)

	Codage en mémoire	Commentaires
	69A20	#02A96h = Adresse-préfixe des répertoires
	Info_Lib	Zone informant sur les librairies attachées
a	Offset_to_Last	Sur 5 quartets: a + offset = α
	00000	5 quartets qui précèdent le premier objet
b	Zone_Obj1	Zone du premier objet (le dernier dans VARS)
c	Longueur_Zone1	Longueur sur 5 quartets de l'intervalle [b,c[
d	Zone_Obj2	Zone 2ème objet (l'avant-dernier dans VARS)
e	Longueur_Zone2	Longueur sur 5 quartets de l'intervalle [d,e[
	....	.....
α	Zone_LastObj	Zone du dernier objet (le premier dans VARS)

Quelques explications s'imposent:

"Info_Lib": Cette zone renseigne sur les librairies attachées:

- Pour un sous-répertoire de **HOME**, elle est longue de 3 quartets: elle contient **#7FFh** si aucune librairie n'est attachée. Sinon elle contient le N° de la seule librairie attachée.

- Pour le répertoire **HOME** lui-même, les choses sont plus compliquées, car un nombre quelconque de librairies peuvent lui être attachées.
  La zone "**Info_Lib**" est alors organisée de la manière suivante:

N	Zone_Lib_1	Zone_Lib_2	...	Zone_Lib_N

  N désigne ici, sur 5 quartets, le nombre de librairies attachées.
  Suivent ensuite les zones (chacune de 13 quartets) donnant les informations utiles sur chaque librairie attachée.
  Ces zones sont placées dans le sens croissant des N°s de librairie.

- D'une façon plus précise, "**Zone_Lib_i**" est codée comme suit:

Id	Ad_Hash_Table	Ad_Mess_Table

  "**Id**" est le numéro identifiant la librairie (sur trois quartets).
  "**Ad_Hash_Table**" est l'adresse où on trouve un entier binaire long permettant d'accéder aux objets de la librairie (pour en savoir plus, voir le chapitre consacré aux librairies).
  "**Ad_Mess_Table**" est l'adresse où on trouve le vecteur des messages associés à cette librairie.
  Les deux champs précédents sont codés sur 5 quartets (comme toutes les adresses..). Le champ "*Ad_Mess_Table*" est à zéro si aucun message n'est associé à cette librairie.

"Zone_Obj"

Dans le schéma précédent, *Zone_Obj1*, *Zone_Obj2*, ..., *Zone_LastObj*, désignent les zones où sont codés les noms et les contenus des différents objets qui composent le répertoire.

Une telle zone est organisée de la manière suivante:

n	Name	n	Obj

Ici n désigne le nombre de caractères du nom de l'objet (sur deux quartets). On remarque que n est répété après "*Name*".
"**Name**" est la suite des caractères composant le nom de l'objet. (chaque caractère occupe un octet, c-à-d deux quartets).
"**Obj**" est le contenu de l'objet.

Remarque:

On constate que l'ordre dans lequel apparaissent les objets, dans le codage en mémoire du répertoire auquel ils appartiennent, est l'ordre inverse de celui des entrées du menu **VAR**, quand ce répertoire est actif.

En fait, quand un répertoire est sélectionné, la HP48 saute d'abord au dernier objet du répertoire (en utilisant le champ "*Offset_to_Last*" décrit précédemment).

Elle lit le nom correspondant, puis elle affiche ce nom comme label de la première entrée de menu.

En remontant vers le début du répertoire (et en utilisant pour celà les "*Longueur_Zone_i*" pour se placer d'un nom au nom précédent), elle trouve ainsi tous les noms du répertoire, qu'elle affiche aux entrées qui suivent dans le menu **VAR**.

Elle s'arrête quand elle arrive sur les 5 octets à zéro qui précèdent "*Zone_Obj1*".

◆ UN EXEMPLE:

Pour illustrer ce qui précède, nous allons considérer un répertoire particulier, et voir comment **HOME** et ce répertoire sont codés en mémoire.

Nous supposerons que **HOME** ne contient que ce sous-répertoire, nommé "**ESSAI**", et la variable **IOPAR** (en deuxième position dans **HOME**, et contenant la liste **{ 9600 0 0 0 3 1}**)

Nous supposerons en outre qu'une librairie, de numéro 800, est attachée à **HOME**, alors qu'une librairie, de numéro 1000, est attachée au sous-répertoire **ESSAI**.

Reste à définir le contenu du répertoire **ESSAI**:

Nous supposerons qu'il est constitué des trois programmes suivants:

'**PGCD**':
```
« → a b
   « b 'PGCD(b,a MOD b)' a IFTE »
»
```

'**SIMPF**':
```
« DUP2 PGCD ROT OVER / 3 ROLLD / »
```

'**ΣF**':
```
« 4 ROLL OVER * ROT 4 PICK
   * + 3 ROLLD * SIMPF
»
```

Le programme '**PGCD**' calcule, comme son nom l'indique, le plus grand diviseur commun de deux entiers A et B:

2:A, 1:B = = [PGCD] = = > 1:Pgcd(A,B)

Le programme '**SIMPF**' simplifie la fraction A/B de numérateur A et de dénominateur B (tous deux entiers), pour donner la fraction C/D:

2:A, 1:B = = [SIMPF] = = > 2:C 1:D

Le programme '**ΣF**' calcule et simplifie la somme de deux fractions à coefficients entiers, A/B+C/D, pour nous donner M/N.

4:A, 3:B, 2:C, 1:D = = [ΣF] = = > 2:M, 1:N

N.B: ce répertoire **ESSAI** sera à nouveau utilisé pour nous donner un exemple de librairie.

La page suivante montre la façon dont est codé ce répertoire **ESSAI** (à l'intérieur du répertoire **HOME**). Ensuite nous verrons comment est codé le répertoire **HOME** lui-même (dans les conditions qui ont été décrites ci-dessus).

Remarques:

Le contenu du répertoire '**ESSAI**' est formé des trois programmes nommés '**PGCD**', '**SIMPF**', et '**ΣF**', dans cet ordre. Mais comme il a été dit plus haut, l'ordre dans le menu **VAR** est alors le suivant: d'abord '**ΣF**', puis '**SIMPF**', et enfin '**PGCD**'.

On retrouve dans le contenu des 3 programmes des objets dont le codage est décrit dans d'autres chapitres de cette première partie du livre. Toutes les fonctions et commandes intégrées sont représentées par leur adresse en ROM (par exemple l'adresse **#1FC29h** pour l'instruction **OVER**).

Vous trouverez les explications sur l'instruction "**Apply**" dans le chapitre consacré aux expressions algébriques.

Le tableau contient trois colonnes:

- La toute première colonne (très étroite) sert uniquement à marquer des repères (a,b,c,d,e,f désignent les adresses où on trouverait le premier quartet de la deuxième colonne).

- La deuxième colonne contient les quartets qui constituent l'image du répertoire en mémoire.
  Notez qu'à son habitude, le microprocesseur "retourne" les unités d'informations (caractères, adresses, etc...)

- La troisième colonne contient des commentaires qui rendent le tout assez clair (I hope so!).

	Contenu du répertoire 'ESSAI'	Commentaires
	69A20	#02A96h = adresse-préfixe
	8E3	#3E8h = librairie 1000 attachée
a	D0100	a + #10Dh = f
	00000	Précède début de la première zone
b	40 05743444 40	Le nom "PGCD" a quatre caractères
	D9D20	Début Rpl
	E1632 1C432	« ->
	D6E20 10 16	le nom local a
	D6E20 10 26	le nom local b
	E1632	«
	D6E20 10 26	b
	8BA20	début expression
	D6E20 10 26 D6E20 10 16	b a
	D6E20 10 26 D4EB1	b MOD
	8BA20	début expression
	84E20 40 05743444	le nom global PGCD
	B2130	fin expression
	30040	l'entier-système <2h>
	046F1	Instruction "Apply"
	B2130	fin expression
	D6E20 10 16	a
	EF3A1 EF532	IFTE » (purge var. locales)
	93632	»
	B2130	Fin Rpl
c	5A000	#A5h = taille de la zone [b,c[
d	50 3594D40564 50	Le nom "SIMPF" a cinq caractères
	D9D20	Début Rpl
	E1632 2ABF1	« DUP2
	84E20 40 05743444	le nom global PGCD
	E0CF1 92CF1 50FA1	ROT OVER /
	3F2A2 0DCF1 50FA1	3 ROLLD /
	93632	»
	B2130	Fin Rpl
e	45000	#54h = taille de la zone [d,e[
f	20 5864 20	Le nom "ΣF" a deux caractères
	D9D20	Début Rpl
	E1632 803A2 5BCF1	« 4 ROLL
	92CF1 EEDA1 E0CF1	OVER * ROT
	803A2 A9CF1 EEDA1	4 PICK *
	76BA1 3F2A2 0DCF1 EEDA1	+ 3 ROLLD *
	84E20 50 3594D40564	le nom global SIMPF
	93632	»
	B2130	Fin Rpl

La page suivante contient la représentation en mémoire du contenu de **HOME**, dans les conditions qui ont été exposées précédemment.

	Contenu du répertoire HOME	Commentaires
	69A20	#02A96h = adresse-préfixe
	300	#003h = 3 librairies attachées
	200 74622 8D356	Librairie #002h
	023 19108 00000	Librairie #320h=800
	007 EFD22 00000	Librairie #700h
a	AC000	a + #CAh = m
	00000	Marque début première zone
b	00	C'est le nom vide
	69A20	Adresse-préfixe des répertoires
	FF7	#7FFh=pas de librairie attachée
c	C5000	c + #5Ch = h
	00000	marque début première zone
d	C0	le nom UserKeys.CRC possède
	55375627B4569737E2342534	#0Ch = 12 caractères
	C0	
	E4A20 90000 6E57	l'entier binaire #75E6h
e	A2000	#2Ah = taille zone [d,e[
f	80 55375627B4569737 80	le nom Userkeys a 8 caractères
	47A20 B2130	liste vide
g	E1000	#1Eh = taille zone [f,g[
h	60 14C61627D637 60	le nom Alarms a 6 caractères
	47A20 B2130	liste vide
i	08000	#80h = taille zone [b,i[
j	50 94F4051425 50	Le nom IOPAR (cinq caractères)
	47A20	{
	19322 4B2A2 4B2A2	9600 0 0
	4B2A2 3F2A2 9C2A2	0 3 1
	B2130	}
k	63000	#36h= taille zone [j,k[
m	50 5435351494 50	Le nom ESSAI (cinq caractères)
	69A20	Début du sous-répertoire ESSAI
	.....	voir page précédente

Remarques:

Les librairies de N° 2 et #700h sont les librairies fondamentales de la HP48. Elles sont toujours attachées à **HOME**. La librairie N°2 contient presque tous les objets intégrés à la HP48, tandis que la librairie #700h contient, entre autres, les objets du répertoire **PRG/BRCH**.

Les adresses de *Hash-Tables* ou de *Message-Tables* associées à ces deux librairies conduisent en fait à des entiers-système qui eux-mêmes désignent les véritables adresses, en ROM cachée.

La librairie (ici imaginaire) N°**320h**=800 n'a pas de table de message, et l'adresse **#80191h** de sa Hash-Table est dans le port 1 (ici il n'y a pas d'indirection).

Pour ce qui est du *répertoire caché* (bien visible ici) on se reportera aux explications fournies dans le chapitre précédent.

EXPRESSIONS ALGÉBRIQUES

◆ Rappels sur les expressions:

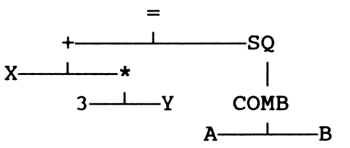
Dans le langage de la HP48, les expressions algébriques sont constituées d'opérateurs (les "*fonctions*") agissant sur des opérandes (les "*arguments*" de ces fonctions).

Les arguments peuvent eux-mêmes être constitués de fonctions s'appliquant à d'autres arguments (la définition des expressions algébriques est donc, par nature, *récursive*).

On peut représenter une expression algébrique par une structure arborescente dont les "*feuilles*" (les arguments ultimes) sont des noms, des nombres (réels ou complexes), ou des objets-unité.

Pour prendre un exemple, l'expression **'X+3*Y=SQ(COMB(A,B))'** peut être représentée par la structure arborescente ci-dessous.

L'opérateur = est appelé l'*opérateur racine* de cette expression.



Il est facile de transformer une expression en une séquence équivalente en notation polonaise inverse.

Il suffit de partir d'un l'opérateur racine '**op**' et d'appliquer la transformation de '**op(arg1,arg2,...,argn)**' en **arg1 arg2 ... argn op**

On applique alors récursivement cette transformation à chacun des arguments *arg1*, *arg2*, ... *argn* s'il est lui-même une expression, jusqu'à ce que tous les arguments obtenus soient des noms, des nombres, ou des objets-unité.

Ainsi l'expression ci-dessus deviendra successivement:

'X+3*Y' 'SQ(COMB(A,B))' =

Puis

'X' '3*Y' + 'COMB(A,B)' SQ =

Et enfin

'X' 3 'Y' * + 'A' 'B' COMB SQ =

◆ Le codage des expressions:

Dans la mémoire de la HP48, les expressions sont codées sous leur forme RPL. Un "*objet-expression*" est constitué de la manière suivante:

- 5 quartets pour l'adresse-préfixe **#02AB8h** (écrite 8BA20...).
- La traduction de l'expression en notation polonaise inverse.
- 5 quartets pour l'adresse-postfixe **#0312Bh** (écrite B2130...): Il faut bien en effet que la calculatrice comprenne quand se termine l'expression.

Remarque: l'adresse postfixe **#0312Bh** est commune aux routines RPL, aux expressions algébriques, aux listes, et aux objets-unités.

Dans la traduction en notation polonaise inverse, chaque objet de l'expression (fonction, argument) est codé d'une des manières suivantes:

- par son adresse sur 5 quartets s'il s'agit d'un objet répertorié en mémoire morte (les fonctions intégrées, les entiers de -9 à +9).
- par son contenu dans les autres cas.

Ainsi l'expression **'X+3*Y=SQ(COMB(A,B))'** sera-t-elle codée (ici les espaces et retours à la ligne sont non significatifs. Ils ne sont là que pour faciliter la lecture):

```
8BA20              L'adresse-préfixe est #02AB8h
    84E20 10 85        Le nom global 'X'
    3F2A2              L'adresse de l'entier 3 est #2A2F3h
    84E20 10 95        Le nom global 'Y'
    EEDA1              L'adresse de l'opération * est #1ADEEh
    76BA1              L'adresse de l'opération + est #1AB67h
    84E20 10 14        Le nom global 'A'
    84E20 10 24        Le nom global 'B'
    6F1C1              L'adresse de la fonction COMB est #1C1F6h
    624B1              L'adresse de la fonction SQ est #1B426h
    8D8A1              L'adresse de la fonction = est #1A8D8h
B2130              L'adresse-postfixe est #0312Bh
```

Voici maintenant le codage de l'expression **' ∫ (1,2,LN(X),X)'**:

```
8BA20              Début expression
    9C2A2 ED2A2                          Les réels 1 et 2
    8BA20 84E20 10 85 F49B1 B2130        Expression 'LN(X)'
    8BA20 84E20 10 85 B2130              Expression 'X'
    322F1                                Opérateur ∫
B2130              Fin de l'expression
```

◆ Cas des "fonctions-utilisateur":

Une particularité se présente lorsque vous créez des expressions incluant une *fonction-utilisateur*. Contrairement aux fonctions intégrées, la HP48 ne sait pas à priori combien d'arguments cette fonction nécessite.

Considérons par exemple l'expression **'W+ABC(X,Y,Z)'** (que la fonction **ABC** soit déjà créée ou non ne change rien: le problème se posera au moment de l'évaluation de cette expression).

Cette expression est codée:

```
8BA20          L'adresse-préfixe est #02AB8h
   84E20 10 75    Le nom global 'W'
   84E20 10 85    Le nom global 'X'
   84E20 10 95    Le nom global 'Y'
   84E20 10 A5    Le nom global 'Z'
   8BA20          Adresse-préfixe #02AB8h des expressions.
      84E20 30 142434  Le nom global 'ABC'
   B2130          Adresse-postfixe #0312Bh des expressions.
   D0040          L'adresse de l'entier-système <3h> est #0400Dh
   046F1          L'adresse de la routine Apply(_,ex,s) est #1F640h
   76BA1          L'adresse de l'opération + est #1AB67h
B2130          Fin de l'expression
```

On voit ici apparaître une routine non standard de la HP48, que j'ai baptisée avec le *mnémonique* **Apply(_,ex,s)** et dont le rôle est d'appliquer une expression réduite à un nom à un nombre d'opérandes précisé par un entier-système (ici **<3h>**).

Tout celà sera décrit dans la deuxième partie du livre (et dans le chapitre conscacré aux expressions).

Terminons par la liste des expressions présentes dans la ROM. Seule la première est dans la ROM non cachée. Pour toutes les autres, n'essayez pas de les évaluer par un SYSEVAL à leur adresse.

Utilisez plutôt la séquence suivante, à partir de leur adresse en binaire:

#5A03h SYSEVAL #C612h SYSEVAL

Remarque: Dans presque toutes les expressions ci-dessous, vous verrez apparaître les noms **&1** ou **&2**. Sachez que ces noms apparaissent dans ces expressions sous forme de nom local.

Une évaluation d'une de ces expressions sans avoir préalablement créé les variables locales **&1** ou **&2** se terminerait donc par une erreur "*Undefined Local Name*".

Expressions dans la ROM de la HP48

Adresse	Contenu de l'expression
#592FFh	's1'
#7C614h	'-COS(&1)'
#7C67Dh	'-INV(SIN(&1))'
#7C6E6h	'-INV(TAN(&1))'
#7C781h	'COS(&1)'
#7C7D1h	'INV(COS(&1)*SIN(&1))'
#7C85Dh	'TAN(&1)^2'
#7C8C1h	'TAN(&1)'
#7C91Bh	'INV(TAN(&1)*SIN(&1))'
#7C99Dh	'INV(COS(&1))'
#7C9F7h	'LN(SIN(&1))'
#7CA6Fh	'&1*ASIN(&1)+√(&-&1^2)'
#7CB0Fh	'&1*ACOS(&1)-√(&-&1^2)'
#7CC31h	'SINH(&1)'
#7CC81h	'INV(SINH(&1)*TANH(&1))'
#7CCEFh	'INV(SINH(&1)^2)'
#7CD58h	'INV(SINH(&1)*COSH(&1))'
#7CDE4h	'COSH(&1)'
#7CE34h	'INV(COSH(&1)^2)'
#7CE93h	'COSH(&1)^-2'
#7CF38h	'TANH(&1)'
#7CF8Dh	'INV(TANH(&1)*SINH(&1))'
#7D00Fh	'INV(COSH(&1))'
#7D069h	'LN(SINH(&1))'
#7D0E1h	'EXP(&1)-&1'
#7D15Eh	'.434294481904*ALOG(&1)'
#7D1EBh	'&1*LN(&1)-&1'
#7D2EBh	'INV(&1)'
#7D33Bh	'INV(&1^2)'
#7D39Ah	'INV(SQ(&1))'
#7D3F4h	'INV(1+&1^2)'
#7D458h	'INV(&1^2+1)'
#7D4DAh	'ATANH(&1)'
#7D54Dh	'&1^3/3'
#7D5E8h	'ASIN(&1)'
#7D651h	'ASINH(&1)'
#7D719h	'ACOSH(&1)'
#7D79Bh	'√&1'
#7D80Eh	'INV(√&1)'
#7D890h	'SIGN(&1)'
#7CBAFh	'&1*ATAN(&1)-LN(1+&1^2)/2'
#7D6C4h	'INV(√(1+&1))*INV(√(-1+&1))'
#7D272h	'.434294481904*(&1*LN(&1)-&1)'
#7C5ECh	'SIN(&1)'
#7C641h	'INV(SIN(&1)*TAN(&1))'
#7C6AFh	'INV(SIN(&1)^2)'
#7C718h	'INV(SIN(&1)*COS(&1))'
#7C7A9h	'SIN(&1)'
#7C80Dh	'LN(TAN(&1)'
#7C88Fh	'TAN(&1)-&1'
#7C8E9h	'LN(COS(&1))'
#7C961h	'TAN(&1)*INV(COS(&1))'
#7C9CAh	'INV(TAN(&1))'
#7CA47h	'ASIN(&1)'
#7CAE7h	'ACOS(&1)'
#7CB87h	'ATAN(&1)'
#7CC59h	'COSH(&1)'
#7CCBDh	'-INV(SINH(&1))'
#7CD26h	'-INV(TANH(&1))'
#7CD94h	'LN(TANH(&1))'
#7CE0Ch	'SINH(&1)'
#7CE6Bh	'TANH(&1)'
#7CECFh	'NV(COSH(&1)*SINH(&1))'
#7CF60h	'LN(COSH(&1))'
#7CFD3h	'TAN(&1)*INV(COSH(&1))'
#7D03Ch	'INV(TANH(&1))'
#7D0B9h	'EXPM(&1)'
#7D136h	'ALOG(&1)'
#7D1C3h	'LN(&1)'
#7D24Ah	'LOG(&1)'
#7D313h	'LN(&1)'
#7D36Dh	'-INV(&1)'
#7D3C7h	'-INV(&1)'
#7D430h	'ATAN(&1)'
#7D49Eh	'INV(1-&1^2)'
#7D525h	'SQ(&1)'
#7D5A7h	'INV(√(1-&1^2))'
#7D610h	'INV(√(1+&1^2))'
#7D679h	'INV(√(&1^2+1))'
#7D741h	'INV(√&1-1)*√(&1+1))'
#7D7C3h	'2*&1^(3/2)/3'
#7D83Bh	'2*√&1'
#7D8B8h	'ABS(&1)'

OBJETS-UNITÉS

Un ***objet-unité*** est la combinaison d'une partie scalaire et d'une partie spécifiant la ou les unités utilisées.
A l'affichage ces deux composantes sont séparées par le caractère _.

La *partie scalaire* est un réel.
La *partie unité* apparaît, à première vue, comme une expression combinant les différentes unités élémentaires utilisées.
Cette combinaison peut prendre la forme d'un quotient num/den, où "*num*" et "*den*" sont respectivement obtenus par produit ou exponentiation d'unités simples.

On peut ainsi former l' objet: **1.2345_N/(m^2*s)**

Voici la liste des seuls objets-unités présents dans la ROM de la HP48:

#0FA58h	1_kg	#0FA84h	1_m	#0FAA4h	1_A	#0FAC4h	1_s
#0FAE4h	1_K	#0FB04h	1_cd	#0FB26h	1_mol	#0FB4Ah	1_?

Quand on effectue, par exemple, **#0FA58h SYSEVAL**, on obtient l'objet unité **1_kg** au niveau 1 de la pile. L'instruction **BYTES** renvoie alors les résultats **#0h** et **2.5**, confirmant ainsi que l'objet est désigné par une adresse en mémoire morte.

Si on passe dans le menu **UNITS\MASS**, et si on actionne la touche **KG** après avoir placé **1** au niveau 1, on obtient toujours l'objet-unité **1_kg** au niveau 1 de la pile, mais cette fois l'instruction **BYTES** donne les résultats **#8Dh** et **22**.

Les objets_unités standards (apparaissant dans le tableau ci-dessus) sont donc recréés en mémoire vive, et non pas considérés comme objets intégrés à la HP48 (comme le sont par exemple les entiers de -9 à 9).

En mémoire, le codage d'un objet_unité est organisé de la manière suivante:

- Une adresse préfixe **#02ADAh**.
- La partie scalaire (représentée par une adresse si elle est entière et comprise entre -9 et 9).
- La description de la partie "unités".
- Une adresse postfixe **#0312Bh** (c'est la même que pour les listes, les expressions et les structures RPL).

- La partie "*unités*" est codée en notation polonaise inverse:

  Les noms d'unités sont sous forme de chaînes.

  Les opérations entre unités (seules *, /, et ^ sont possibles) sont codées par des adresses renvoyant à une liste vide.
  - L'adresse **#10B5Eh** correspond à l'opération *
  - L'adresse **#10B68h** correspond à l'opération /
  - L'adresse **#10B72h** correspond à l'opération ^
  - L'adresse **#10B7Ch** correspond à l'opération permettant de combiner un préfixe d'unité (voir le tome 1 du manuel de l'utilisateur, page 201) avec une unité.

- L'opération _ (qui marque la fin de la partie "unités") est codée par l'adresse **#10B86h**, désignant une liste vide.

Exemples de codages d'objets-unités:

- Le codage de **1_m** est:

```
ADA20   Adresse-préfixe
    9C2A2            L'adresse de 1 est #2A2C9h
    C2A20 70000 D6   La chaîne "m"
    68B01            #10B86h: liste vide remplaçant _
B2130   Adresse-postfixe
```

- Voici le codage de l'objet-unité 1.2345_N/(cm^2*s).

```
ADA20   Adresse-préfixe
    33920 000 000000054321 0  Le réel 1.2345
    C2A20 70000 E4            La chaîne "N"
    FB920 36                  Le caractère "c"
    C2A20 70000 D6            La chaîne "m"
    C7B01                     {} pour combiner "c" et "m"
    ED2A2                     L'adresse de 2 est #2A2DEh
    27B01                     #10B72h: liste vide remplaçant ^
    C2A20 70000 37            La chaîne "s"
    E5B01                     #10B5Eh: liste vide remplaçant *
    86B01                     #10B68h: liste vide remplaçant /
    68B01                     #10B86h: liste vide remplaçant _
B2130   Adresse-postfixe
```

LISTES

Les listes sont des *objets composés* (tout comme les structures RPL et les expressions algébriques). Elles permettent de grouper dans une même structure ordonnée une succession d'objets hétéroclites.

Le codage d'une liste dans la mémoire de la HP48, est le suivant:

- Une adresse-préfixe **#02A74h.**
- La description des objets figurant dans la liste. Un tel objet peut être décrit par son adresse en ROM (si elle est connue de la HP48) sur 5 quartets, ou par son contenu.
- Une adresse-préfixe **#0312Bh**, indiquant la fin de la liste.

Au risque de radoter, je rappelle que notre ami le *Saturn* lit et écrit les adresses à l'envers. Il faut s'y habituer.

Ainsi la liste **{ 1 'A' 15 { * + } "BASE" }** est codée de la manière suivante (les espaces et les retours à la ligne sont ici non significatifs et ne servent qu'à vous faciliter la lecture de l'objet).

```
47A20     L'adresse-préfixe des listes est #02A74h.
   9C2A2                      L'adresse de l'entier 1 est #2A2C9h
   84E20 10 14                Le nom global 'A'.
   33920 100 000000000051 0   Le réel 15.
   47A20                      Adresse-préfixe des listes.
      EEDA1                   L'adresse de * est #1ADEEh
      76BA1                   L'adresse de + est #1AB67
   B2130                      Adresse-postfixe des listes.
   C2A20 D0000 24143554       La chaîne "BASE"
B2130     L'adresse-postfixe des listes est #0312Bh
```

On remarque dans cette liste (nous l'avons déjà observé dans le chapitre consacré aux nombres réels) que l'entier **1** est reconnu comme un objet intégré à la HP48 (et est donc codé par une adresse en ROM).
Il n'en est pas de même de **15** qui est codé par son contenu.

Remarque: les listes, les structures RPL, les expressions, et les objets-unités ont la même adresse-préfixe.

Le nombre des listes figurant en ROM est particulièrement important. Nous allons plutôt en donner une liste "*thématique*".

◆ LISTES VIDES:

Une liste vide est codée **47A20B2130**. On en rencontre quelques-unes dans la ROM de la HP48.

La plus "*sérieuse*" semble être à l'adresse **#055E9h**, car on y trouve à coté d'autres objets "*vides*" (binaire, chaîne, expression, RPL).

Les 4 listes vides consécutives aux adresses **#10B5Eh** à **#10B86h** sont utilisées pour la formation des objets-unités.

Adresses des listes vides						
#055E9h	#0B5A8h	#101D5h	#10B5Eh	#10B68h	#10B72h	#10B7Ch
#10B86h	#23989h	#258EEh	#2807Ah	#547ABh	#600F3h	#68484h

◆ LISTES DE NOMS LOCAUX:

On trouve de nombreuses listes constituées exclusivement de noms locaux.

Ces listes sont particulièrement utilisées par la HP48 pour créer plusieurs variables locales simultanément.

Dans les tableaux ci-dessous, les crochets **[..]** indiquent que c'est l'adresse du nom qui apparaît dans la liste, pas le nom lui-même (en fait, çà ne change pas grand chose).

J'ai désigné par " le nom local vide (important) et par **{ 7['"] }** (par exemple) une liste contenant 7 adresses du nom local vide.

Adresse	Liste de noms locaux
#272C8h	{ 'ttt 'str 'ofs 'tok 'rbv 'idfflg 'tmpop 'tmppdat 'ploc 'bv 'unbound }
#2D5F5h	{ ['PACKET] ['RETRY] ['MaxR] }
#2D86Bh	{ ['LNAME] ['RETRY] ['KMODE] ['KRM] }
#2DF01h	{ ['LNAME] ['KMODE] ['KRM] }
#2E90Dh	{ ['LNAME] ['KMODE] ['KRM] }
#2EED3h	{ ['LNAME] ['OBJ] ['PACKET] ['KRM] }
#2F1FDh	{ [''] ['KLIST] ['OPOS] 'KML }
#4FF98h	{ 'xe 'ye 'x 'y 'xc 'yc 'r2 'left 'up 'exit }
#60806h	{ 'reg 'sur 'cts 'sun 'mlg 'ckd 'prd 'prp' 'rhs }
#60BE4h	{ 'patternls 'compos 'varls }
#69A92h	{ 'Radix 'KeysOK? 'ExprLit 'BuffW 'BuffH 'SaveBlank 'ManOp 'nohalt 'AppMode 'NameGRob 'EXITFCN 'FontGauge 'LE 'LB 'TE 'FormEnvOK 'prow 'pcol 'cursy 'cursx 'ttt 'source 'ofs 'tok 'rbv 'idfflg 'tmpop 'tmppdat 'ploc 'bv 'unbound }

Adresse	Liste de noms locaux	Adresse	Liste de noms locaux
#23754h	{ ['noname] ['stop] }	#23879h	{ 'ioinprogress }
#23903h	{ st ofs tok }	#2D839h	{ ['KP] ['PKNO] }
#2DD56h	{ ['ERRMSG] [''] }	#2E6CDh	{ ['KP] ['PKNO] }
#2E8E5h	{ ['KP] ['PKNO] }	#32F9Fh	{ ['nohalt] }
#36BF1h	{ #a #b [''] }	#38A39h	{ 'SavedUI }
#4C93Fh	{ 'xmax 'N }	#4CF50h	{ 'EnvOK 'EXITFCN }
#50D39h	{ 'PlotEnv }	#5456Ah	{ 'tcls 'fcls }
#54DCBh	{ 'xSYMfcn 'xfcn }	#5577Eh	{ 'scl 'xSYMfcn 'xfcn }
#557FBh	{ 'xSYMfcn 'xfcn }	#56971h	{ 'sumexpr 'sumvar }
#571F2h	{ ['dv] ['op] ['nm] }	#578CEh	{ ['ni] ['ns] }
#592BBh	{ 'c 'b 'a }	#59512h	{ 'n 'prog }
#5A75Ch	{ 'd 'R 'est 'X 'T }	#5AAE0h	{ 'bnds ['dvar] }
#5D678h	{ 'which 'op1 'op2 }	#5FDBCh	{ 'ct 'pp 'ep }

Adresse	Liste	Adresse	Liste	Adresse	Liste
#0E475h	{ M N }	#14396h	{ 'halt }	#155EFh	{ ['nohalt] }
#1F960h	{ ['num] }	#24A28h	{ i j }	#25A06h	{ '1 '2 '3 }
#2D4DBh	{ [''] }	#2E9F0h	{ [''] }	#2F6F0h	{ ['RETRY] }
#31C32h	{ 'IWrap }	#3306Ch	{ 7[''] }	#34D2Bh	{ '' }
#36CEAh	{ #b [''] }	#36D5Ah	{ 2[''] }	#36D82h	{ 3[''] }
#52D26h	{ 4[''] }	#55666h	{ 'oth }	#549CCh	{ ['dvar] }
#56EFCh	{ ['dv] }	#57EA3h	{ ['*s] }	#580F9h	{ ['+s] }
#58DB1h	{ 'fl }	#59110h	{ 'nmls }	#59641h	{ 'n }
#5A60Fh	{ 'piflag }	#5A660h	{ 'd 'r }		

◆ LISTES DIVERSES:

Quand un objet d'une liste n'est pas représenté par son contenu mais par son adresse, je l'ai placé entre crochets **[ ]** (aucune confusion possible avec les vecteurs de réels, absents de la ROM)

#19A91h	{0 "" 0}	#223C9h	{[1] [2] [3]}
#22400h	{[0] [1] [2] [3] [4]}	#22441h	{[0] [1] [2] [3]}
#2234Dh	{1200 2400 4800 9600}	#28BB4h	{[{}]}
#31CA9h	{[1]}	#31F4Ah	{1.8 [""] [80] ["▪▪"]}
#3304Eh	{[<Eh>]}	#36796h	{[<3h>]}
#368CCh	{[3]}	#45716h	{[<1h>] [<1h>]}
#47B73h	{[9] [15]}	#47BBEh	{[3] [15]}

#0FA53h	{1_kg 1_m 1_A 1_s 1_K 1_cd 1_mol 1_?}
#19EFAh	{[4954521600] [707788800] [29491200] [491520] [8192] [1]}
#25699h	{[<2h>] [<8h>] [<Ah>] [<10h>]}
#2C756h	{[1] [2] [0] [0] [LINFIT]}
#2E99Eh	{[9600] [0] [0] [0] [3] [1]}
#433DBh	{[""] [<0h>] [<1h>] [<1h>] [<2h>] [<0h>]}
#4A7CFh	{[LINFIT] [LOGFIT] [EXPFIT] [PWRFIT]}
#60E73h	{[{}] [{}] [{}] [{}] &1 &2 &3 &4}
#6419Ah	{[<13Eh>] [<123h>] [<DFFh>]}

◆ LISTES DE CHAINES DE CARACTERES:

La plupart des listes de chaînes de caractères en ROM contiennent les chaînes représentant les unités intégrées de la HP48.

Adresse	Contenu de la liste de chaînes de caractères
#19F6Bh	{ " week(s)" " day(s)" " hour(s)" " minute(s)" " second(s)" " ticks" }
#1FF2Fh	{ " intercept" " slope" }
#221F4h	{ "none " "odd " "even " "mark " }
#3D096h	{ "m" "cm" "mm" "yd" "ft" "in" "Mpc" "pc" "lyr" "au" "km" "mi" "nmi" "miUS" "chain" "rd" "fath" "ftUS" "mil" "μ" "A" "fermi" }
#3D1FDh	{ "m^2" "cm^2" "b" "yd^2" "ft^2" "in^2" "km^2" "ha" "a" "mi^2" "miUS^2" "acre" }
#3D2E0h	{ "m^3" "st" "cm^3" "yd^3" "ft^3" "in^3" "l" "galUK" "galC" "gal" "qt" "pt" "ml" "cu" "ozfl" "ozUK" "tbsp" "tsp" "bbl" "bu" "pk" "fbm" }
#3D45Bh	{ "yr" "d" "h" "min" "s" "Hz" }
#3D4C4h	{ "m/s" "cm/s" "ft/s" "kph" "mph" "knot" "c" "ga" }
#3D55Dh	{ "kg" "g" "lb" "oz" "slug" "lbt" "ton" "tonUK" "t" "ozt" "ct" "grain" "u" "mol" }
#3D64Ch	{ "N" "dyn" "gf" "kip" "lbf" "pdl" }
#3D6BFh	{ "J" "erg" "Kcal" "cal" "Btu" "ft*lbf" "therm" "MeV" "eV" }
#3D76Eh	{ "W" "hp" }
#3D7A1h	{ "Pa" "atm" "bar" "psi" "torr" "mmHg" "inHg" "inH2O" }
#3D842h	{ "°C" "°F" "°K" "°R" }
#3D891h	{ "V" "A" "C" "Ω" "F" "W" "Fdy" "H" "mho" "S" "T" "Wb" }
#3D944h	{ "°" "r" "grad" "arcmin" "arcs" "sr" }
#3D9BDh	{ "fc" "flam" "lx" "ph" "sb" "lm" "cd" "lam" }
#3DA4Ch	{ "Gy" "rad" "rem" "Sv" "Bq" "Ci" "R" }
#3DAC9h	{ "P" "St" }
#4132Dh	{ ["0"] ["1"] ["2"] ["&"] } (liste de 4 adresses)

◆ LISTES "ÉXÉCUTABLES":

J'ai regroupé des listes dont la nature est d'être *évaluée comme un programme*. Par exemple, la première d'entre elles, **{ [i] [*] [+] }**, évaluée "sous" les noms '**X**' (au niveau 3) et '**Y**' (au niveau 2) donne les résultat '**X+Y*i**'. Là encore les crochets **[** et **]** signifient que ce sont les adresses qui figurent dans la liste (çà ne change rien).

Attention:

Certaines listes sont à des adresses > #70000h (et donc en ROM *cachée*). Ne tentez pas de les placer sur la liste par un simple SYSEVAL à leur adresse.

Utilisez plutôt, à partie de leur adresse en binaire, le programme:

« #05A03h SYSEVAL #0C612h SYSEVAL »

Adresse	Liste "Éxécutable"
#281D3h	{ [i] [*] [+] }
#56E43h	{ [π] [180] [/] [*] }
#56EA2h	{ [180] [π] [/] [*] }
#576F8h	{ [π] [i] [*] }
#5DD42h	{ [2] [^] [-] [√] [i] }
#7B9E9h	{ [10] [LN] [/] [+] }
#7BCB4h	{ [ABS] [2] [*] [/] }
#7BE6Ch	{ [2] [^] [-] [√] [INV] }
#7BF0Ch	{ [2] [^] [+] [INV] }
#7C083h	{ [1] [+] [/] }
#7C1EBh	{ [√] [*] [/] }
#7C268h	{ [COSH] [2] [^] [/] }
#7EF33h	{ [-] [2] [I] [*] [/] }
#7EFBAh	{ [2] [*] [EXP] [1] [-] }
#7F023h	{ [I] [NEG] [1] }
#7F091h	{ [PI] [2] [/] [I] [1] }
#7F10Eh	{ [I] [NEG] [1] [I] }

Adresse	Liste "Éxécutable"
#56BCDh	{ [2] [^] [/] }
#56E61h	{ [π] [200] [/] [*] }
#56EC0h	{ [200] [π] [/] [*] }
#5772Fh	{ [2] [π] [*] [i] [*] }
#5DE1Eh	{ [π] [/] [*] }
#7BC8Ch	{ [CONJ] [*] [+] }
#7BD4Ah	{ [1] [-] [√] }
#7BEC6h	{ [2] [^] [+] [√] [/] }
#7BF61h	{ [2] [^] [-] [/] }
#7C0BFh	{ [10] [LN] [*] [/] }
#7C227h	{ [TAN] [2] [^] [+] }
#7C330h	{ [1] [-] [^] [*] }
#7EF88h	{ [+] [2] [/] }
#7EFF1h	{ [I] [*] [/] }
#7F055h	{ [*] [+] [LN] [*] }
#7F0CDh	{ [*] [+] [LN] [*] [+] }
#7F136h	{ [*] [+] [1] }

#7BCFAh	{ [2] [^] [-] [√] [INV] [NEG] }
#7BE1Ch	{ [CONJ] [-] [2] [I] [*] [/] }
#7F159h	{ [2] [^] [+] [√] [/] [LN] [*] }
#7F1A4h	{ [I] [*] [SIN] [I] [*] [NEG] }
#7F1E5h	{ [I] [*] [COS] }
#7F217h	{ [I] [*] [TAN] [I] [NEG] [*] }
#7F262h	{ [2] [^] [+] [√] }
#7F28Fh	{ [-] [LN] [NEG] }
#7F2C1h	{ [PI] [2] [/] [I] [1] }
#7F2FDh	{ [*] [+] [LN] [*] [+] [2] [^] [NEG] [√] }
#7F370h	{ [2] [^] [-] [√] [/] [LN] [NEG] }

◆ D'AUTRES LISTES EN ROM CACHÉE:

Les listes suivantes sont formées de sous-listes, la première d'entre elles étant représentée par son adresse **#37E06h** (voir plus haut dans le paragraphe "*listes diverses*").

Je n'ai représenté les autres sous-listes que par leur adresse.

Il est difficile de donner intégralement le contenu de ces listes qui contiennent des expressions et des routines RPL.

Sachez que l'on y trouve les schémas qui permettent à la HP48 d'effectuer des dérivations et des intégrations symboliques.

N.B: utilisez « **#05A03h SYSEVAL #0C612h SYSEVAL** » pour placer sur la pile les listes à partir de leur adresse en binaire.

Adresse	Contenu de la liste (adresses de ses sous-listes)
#7C5DDh	{ [#37E06h] #7C5E7h #7C63Ch #7C6AAh #7C713h }
#7C772h	{ [#37E06h] #7C77Ch #7C7CCh }
#7C84Eh	{ [#37E06h] #7C858h #7C8BCh #7C916h #7C95Ch #7C9C5h }
#7CA38h	{ [#37E06h] #7CA42h }
#7CAD8h	{ [#37E06h] #7CAE2h }
#7CB78h	{ [#37E06h] #7CB82h }
#7CC22h	{ [#37E06h] #7CC2Ch #7CC7Ch #7CCEAh #7CD53h }
#7CDD5h	{ [#37E06h] #7CDDFh #7CE2Fh #7CE8Eh #7CECAh }
#7CF29h	{ [#37E06h] #7CF33h #7CF88h #7CFCEh #7D037h }
#7D0AAh	{ [#37E06h] #7D0B4h }
#7D127h	{ [#37E06h] #7D131h }
#7D1B4h	{ [#37E06h] #7D1BEh }
#7D23Bh	{ [#37E06h] #7D245h }
#7D2DCh	{ [#37E06h] #7D2E6h #7D336h #7D395h #7D3EFh #7D453h #7D499h }
#7D516h	{ [#37E06h] #7D520h }
#7D598h	{ [#37E06h] #7D5A2h #7D60Bh #7D674h #7D6BFh #7D73Ch #7D796h #7D809h }
#7D881h	{ [#37E06h] #7D88Bh }

◆ LISTES DES ENTRÉES DE MENU:

Nous allons maintenant voir les listes les plus nombreuses (et les plus passionnantes du point de vue de la recherche de SYSEVALs) que sont les listes représentant des entrées de menu.

Vous n'êtes pas sans savoir que la HP48 permet de créer des menus personnalisés grâce aux instructions **MENU** et **TMENU**.

Pour **MENU** et **TMENU**, un *menu-utilisateur* de n entrées est défini par une liste ayant le format: **{ Label_1 Label_2 ..... Label_n }**, où *Label_1... Label_n* sont des objets définissant les différentes entrées de ce menu.

Prenons l'exemple de ***Label_1***.

- ***Label_1*** peut-être une liste ou un autre objet (Dans ce dernier cas l'appui sur la touche de menu évalue cet objet).

- Dans le cas où ***Label_1*** est une liste, elle a le format suivant:
  { "Libellé" Objet1 }
  ou **{ "Libellé" { Objet1 Objet2 } }**
  ou **{ "Libellé" { Objet1 Objet2 Objet3 } }**

- "***Libellé***" est le nom donné à cette entrée de menu.

- L'appui sur la touche de menu se traduit alors par l'évaluation de: *Objet1* , *Objet2*, *Objet3*, suivant que la touche est non shiftée, shiftée par <¬ , ou est shiftée par ┌−>.

Dans la ROM de la HP48, de nombreuses entrées de menu sont définies par ce modèle.

Prenons par exemple le cas du menu **MEMORY**. La liste définissant ce menu est en **#3BCE7h**. Elle est constituée des 17 adresses correspondant à chacune des 17 entrées (de **MEM** à **PGDIR**).

Si vous faites **#3BCE7h SYSEVAL**, vous placez cette liste au niveau 1.

Vous pouvez alors éxécuter **MENU**, ce qui a pour effet de définir le menu **'CST'** à l'identique du menu **MEMORY**).

Les choses ne sont pas toujours aussi simples !

Prenons l'exemple du menu **GRAPH**. La liste qui contient les différentes entrées de ce menu débute en **#3DE9Dh**.

Ne croyez pas cependant pouvoir avec cette liste faire ce que vous venez de faire avec la liste définissant le menu **MEMORY**: vous risquez de "*planter*" votre HP48.

L'explication est la suivante: avant que d'afficher le menu **GRAPH** (à l'issue d'un appui sur la touche correspondante du clavier, ou parce que l'instruction **GRAPH** a été rencontrée dans un programme), la HP48 "*prépare le terrain*" et en particulier crée tout un tas de variables locales (ne serait-ce que pour la position du curseur...)

Vous ne pouvez pas "*shunter*" ces préparatifs.

Je vous donne dans les tableaux suivants toutes les listes correspondant à des entrées de menu.

Cependant soyez prudent dans leur utilisation et gardez la remarque précédente à l'esprit, notamment si vous voulez utiliser les listes correspondant à des entrées de menu dans des *environnements complexes* tels que: **GRAPH**, **Pile Interactive**, **Equation-Writer**, **Matrix-Writer**.

Ces listes n'en demeurent pas moins des trésors à SYSEVALs.

Notations:

Une liste dont le contenu se termine par le caractère \ désigne les différentes entrées d'un menu. Dans le cas contraire, elle désigne une entrée particulière d'un menu.

Ainsi **{MTH\HYP}** désigne l'entrée **"HYP"** du menu **MTH**, et **{MTH\HYP\}** désigne la liste des différentes entrées du menu **MTH\HYP**.

Pour des raisons d'encombrement, j'ai parfois réduit le chemin permettant de désigner telle ou telle entrée de menu.

Par exemple, l'entrée **{TIME\ALRM\RPT\WEEK}** est devenue **{RPT\WEEK}**.

L'adresse **#3EC71h** est celle d'une liste permettant de définir une *entrée vide* dans un menu (avec un *bip* si on cherche à évaluer cette entrée). De telles entrées vides sont parfois utilisées dans les différents menus intégrés à la HP48.

Adresse	Menu,Entrée	Adresse	Menu,Entrée	Adresse	Menu,Entrée
#3B293h	{MTH\}	#3B298h	{MTH\PARTS}	#3B2BBh	{MTH\PROB}
#3B2DCh	{MTH\HYP}	#3B2FBh	{MTH\MATRX}	#3B31Eh	{MTH\VECTR}
#3B341h	{MTH\BASE}	#3B36Ch	{MTH\PARTS\}	#3B3E4h	{MTH\PROB\}
#3B420h	{MTH\HYP\}	#3B452h	{MATH\MATRX\}	#3B489h	{MTH\VECTR\}
#3B4CAh	{MTH\BASE\}	#3B551h	{PRG\}	#3B556h	{PRG\STK}
#3B575h	{PRG\OBJ}	#3B594h	{PRG\DSPL}	#3B5B5h	{PRG\CTRL}
#3B5D6h	{PRG\BRCH}	#3B5F7h	{PRG\TEST}	#3B622h	{PRG\STK\}
#3B67Fh	{PRG\OBJ\}	#3B6F7h	{PRG\DSPL\}	#3B7E2h	{PRG\CTRL\}
#3B8B4h	{PRG\BRCH\}	#3B90Eh	{PRG\TEST\}	#3B972h	{PRINT\}
#3B9A4h	{I/O\}	#3BA08h	{I/O\SETUP\}	#3BA0Dh	{SETUP\IR/W}
#3BA2Eh	{SETUP\ASCII}	#3BA51h	{SETUP\BAUD}	#3BA7Eh	{SETUP\PARIT}
#3BAABh	{SETUP\CKSM}	#3BAD8h	{SETUP\TRAN}	#3BB46h	{MODES\}
#3BB5Fh	{MODES\SYM}	#3BBA6h	{MODES\BEEP}	#3BBFEh	{MODES\CNCT}
#3BC8Dh	{CUSTOM\}	#3BCE7h	{MEMORY\}	#3BD46h	{STORE\}
#3BDFAh	{EDIT\}	#3BE27h	{SOLVE\}	#3BE2Ch	{SOLVE\SOLVR}
#1532Ch	{SOLVR\NXEQ}	#1585Ah	{SOLVR\EXPR=}	#3BEBDh	{PLOT\}
#3BEC2h	{PLOT\PLOTR}	#3BF21h	{PLOT\PTYPE}	#3BF71h	{PLOT\NEW}
#3BF9Fh	{PLOT\EDEQ}	#3BFC0h	{PLOT\STEQ}	#3BFFCh	{PLOT\CAT}
#3C03Eh	{PLOT\PTYPE\}	#3C0B4h	{PLOT\PLOTR\}	#3C0B9h	{PLOTR\ERASE}
#3C0F0h	{PLOTR\DRAW}	#3C145h	{PLOTR\AUTO}	#3C18Bh	{PLOTR\XRNG}
#3C1D6h	{PLOTR\YRNG}	#3C221h	{PLOTR\INDEP}	#3C26Ch	{PLOTR\DEPND}
#3C2ADh	{PLOTR\RES}	#3C2E9h	{PLOTR\CENTR}	#3C325h	{PLOTR\SCALE}
#3C361h	{PLOTR\RESET}	#3C384h	{PLOTR\AXES}	#3C3CAh	{PLOTR*H}
#3C3F2h	{PLOTR*W}	#3C41Ah	{PLOTR\PDIM}	#3C483h	{ALGEBRA\}
#3C4CEh	{TIME\}	#3C4D3h	{TIME\SET}	#3C51Ah	{TIME\ADJST}
#3C565h	{TIME\ALRM}	#3C5B8h	{TIME\ACK}	#3C5E5h	{TIME\ACKALL}
#3C612h	{TIME\CAT}	#3C671h	{TIME\ADJST\}	#3C676h	{ADJST\HR+}
#3C6A4h	{ADJST\HR-}	#3C6D2h	{ADJST\MIN+}	#3C702h	{ADJST\MIN-}
#3C732h	{ADJST\SEC+}	#3C762h	{ADJST\SEC-}	#3C7A1h	{TIME\ALRM\}
#3C7A6h	{ALRM\>DATE}	#3C7C9h	{ALRM\>TIME}	#3C7ECh	{ALRM\A/PM}
#3C80Dh	{ALRM\EXEC}	#3C842h	{ALRM\RPT}	#3C889h	{ALRM\SET}
#3C8D5h	{ALRM\RPT\}	#3C8DAh	{RPT\WEEK}	#3C8FBh	{RPT\DAY}
#3C91Ah	{RPT\HOUR}	#3C93Bh	{RPT\MIN}	#3C95Ah	{RPT\SEC}
#3C979h	{RPT\NONE}	#3C9B8h	{TIME\SET\}	#3C9BDh	{SET\->DAT}
#3C9E0h	{SET\->TIM}	#3CA03h	{SET\A/PM}	#3CA38h	{SET\12/24}
#3CA6Fh	{SET\M/D}	#3CAACh	{ΣDAT\}	#3CAB1h	{ΣDAT\Σ+}
#3CB1Ah	{ΣDAT\CLΣ}	#3CB47h	{ΣDAT\NEW}	#3CB75h	{ΣDAT\EDITΣ}
#3CB98h	{ΣDAT\STOΣ}	#3CBD4h	{ΣDAT\CAT}	#3CC06h	{ΣDAT\XCOL}
#3CC47h	{ΣDAT\YCOL}	#3CC88h	{ΣDAT\BARPL}	#3CCBFh	{ΣDAT\HISTP}
#3CCECh	{ΣDAT\SCATR}	#3CD9Bh	{ΣDAT\MODL\}	#3CDA0h	{MODL\LIN}
#3CDBFh	{MODL\LOG}	#3CDDEh	{MODL\EXP}	#3CDFDh	{MODL\PWR}
#3CE1Ch	{MODL\BEST}	#3CE74h	{UNIT\}	#3CE79h	{UNIT\LENG}
#3CE9Ah	{UNIT\AREA}	#3CEBBh	{UNIT\VOL}	#3CEDAh	{UNIT\TIME}
#3CEFBh	{UNIT\SPEED}	#3CF1Eh	{UNIT\MASS}	#3CF3Fh	{UNIT\FORCE}
#3CF62h	{UNIT\ENRG}	#3CF83h	{UNIT\PWR}	#3CFA4h	{UNIT\PRESS}
#3CFC7h	{UNIT\TEMP}	#3CFE8h	{UNIT\ELEC}	#3D009h	{UNIT\ANGL}
#3D02Ah	{UNIT\LIGHT}	#3D04Dh	{UNIT\RAD}	#3D061h	{UNIT\VISC}
#3DAF2h	{CONVERT\}	#3DB5Bh	{MODES\HEX}	#3DB9Dh	{MODES\DEC}
#3DBDFh	{MODES\OCT}	#3DC21h	{MODES\BIN}	#3DC63h	{PRG\BRCH\IF}
#3DCC2h	{BRCH\CASE}	#3DD1Ch	{BRCH\START}	#3DD6Ch	{BRCH\FOR}
#3DDBCh	{BRCH\DO}	#3DDF8h	{BRCH\WHILE}	#3DE34h	{BRCH\IFERR}
#3DE9Dh	{GRAPH\}	#3DEBBh	{GRAPH\FCN}	#3DF39h	{GRAPH\FCN\}

Adresse	Menu,Entrée	Adresse	Menu,Entrée	Adresse	Menu,Entrée
#3DF3Eh	{FCN\ROOT}	#3DF5Fh	{FCN\ISECT}	#3DF82h	{FCN\SLOPE}
#3DFA5h	{FCN\AREA}	#3DFC6h	{FCN\EXTR}	#3DFE7h	{FCN\EXIT}
#3E019h	{FCN\F(X)}	#3E03Ah	{FCN\F'}	#3E057h	{FCN\NXEQ}
#3E091h	{GRAPH\+/-}	#3E0C4h	{GRAPH\REPL}	#3E0E5h	{GRAPH\SUB}
#3E104h	{GRAPH\DEL}	#3E123h	{GRAPH\COORD}	#3E146h	{GRAPH\LABEL}
#3E15Ah	{GRAPH\CENTR}	#3E16Eh	{GRAPH\ZOOM}	#3E19Eh	{GRAPH\Keys}
#3E1BFh	{GRAPH\MARK}	#3E1E0h	{GRAPH\LINE}	#3E1F4h	{GRAPH\TLINE}
#3E208h	{GRAPH\BOX}	#3E21Ch	{GRAPH\CIRCL}	#3E241h	{GRAPH\Z-BOX}
#3E273h	{GRAPH\DOT+}	#3E2A8h	{GRAPH\DOT-}	#3E2DDh	{EDIT\<-SKIP}
#3E35Fh	{EDIT\SKIP->}	#3E3E1h	{EDIT\<-DEL}	#3E4CAh	{EDIT\DEL->}
#3E586h	{EDIT\INS}	#3E5CDh	{EDIT\\|STK}	#3E71Ah	{I/O\SETUP}
#3E774h	{MODES\STD}	#3E7B6h	{MODES\FIX}	#3E7E9h	{MODES\SCI}
#3E81Ch	{MODES\ENG}	#3E84Fh	{MODES\ML}	#3E88Fh	{MODES\DEG}
#3E8D1h	{MODES\RAD}	#3E908h	{MODES\GRAD}	#3E94Eh	{MODES\XYZ}
#3E98Ah	{MODES\R_Z}	#3E9EEh	{MODES\R__}	#3EA4Dh	{MODES\CMD}
#3EA9Eh	{MODES\STK}	#3EB1Ch	{MODES\ARG}	#3EB68h	{MODES\FM,}
#3EBAFh	{MODES\CLK}	#3EBF6h	{STAT\MODL}	#3EC71h	{Entrée Vide}
#3F0D6h	{RULES\}	#3F407h	{LIBR\PORT0}	#3F45Ch	{LIBR\PORT1}
#3F4B6h	{LIBR\PORT2}	#43D10h	{STACK\}	#43D15h	{STACK\ECHO}
#43D29h	{STACK\VIEW}	#43D5Eh	{STACK\PICK}	#43D72h	{STACK\ROLL}
#43D86h	{STACK\ROLLD}	#43D9Ah	{STCK\->LIST}	#43DAEh	{STACK\DUPN}
#43DC2h	{STACK\DRPN}	#43DE3h	{STACK\KEEP}	#43E04h	{STACK\LEVEL}
#43E31h	{ECHO}	#4600Dh	{MATRIX\}	#46012h	{MATRIX\EDIT}
#46033h	{MATRIX\VEC}	#46047h	{MATRX\<-WID}	#46068h	{MATRX\WID->}
#46089h	{MATRIX\GO->}	#4609Dh	{MATRIX\GO|}	#460B1h	{MATRIX\+ROW}
#460D2h	{MATRIX\-ROW}	#460F3h	{MATRIX\+COL}	#46114h	{MATRIX\-COL}
#46135h	{MATRX\->STK}	#46156h	{MATRIX\\|STK}	#48508h	{EQ_CAT\}
#4854Eh	{STAT_CAT\}	#4858Ah	{ALRM_CAT\}	#485BCh	{ST_CAT\1VAR}
#48652h	{ST_CAT\PLOT}	#486B4h	{ST_CAT\2VAR}	#48718h	{ST_CAT\EDIT}
#48761h	{EQ_CAT\SLVR}	#487C5h	{EQ_CAT\PLTR}	#4880Bh	{EQ_CAT\EQ+}
#4891Ah	{EQ_CAT\EDIT}	#48990h	{CAT\PURGE}	#48A71h	{CAT\->STK}
#48B3Fh	{CAT\VIEW}	#48C5Ah	{EQ_CAT\FAST}	#48CB7h	{CAT\ORDER}
#48D20h	{AL_CAT\EDIT}	#48D8Dh	{AL_CAT\PURG}	#48DDDh	{AL_CAT\EXEC}
#4EE47h	{GRAPH\ZOOM\}	#4EE4Ch	{ZOOM\XAUTO}	#4EE8Dh	{ZOOM\X}
#4EEC1h	{ZOOM\Y}	#4EEF5h	{ZOOM\XY}	#596B0h	{RULES\COLCT}
#596FCh	{RULES\DNEG}	#5974Ah	{RULES\DINV}	#59798h	{RULES*1}
#5981Eh	{RULES\^1}	#59868h	{RULES\/1}	#598EEh	{RULES\+1-1}
#59950h	{RULES\<-->}	#5997Ch	{RULES\<-A}	#599CBh	{RULES\A->}
#59A1Ah	{RULES\<-T}	#59A69h	{RULES\T->}	#59AB8h	{RULES\(<-}
#59B07h	{RULES\->)}	#59B56h	{RULES\(())}	#59B86h	{RULES\AF}
#59BB2h	{RULES\<-M}	#59C01h	{RULES\M->}	#59C50h	{RULES\-()}
#59C7Eh	{RULES\1/()}	#59CAEh	{RULES\E()}	#59CDCh	{RULES\L()}
#59D0Ah	{RULES\L*}	#59D36h	{RULES\E^}	#59D62h	{RULES\->()}
#59DB3h	{RULES\<-D}	#59E02h	{RULES\D->}	#59E51h	{RULES\->TRG}
#59E81h	{RULES\->()}	#59ED2h	{RULES\->DEF}	#59F02h	{RULES\TRG*}
#6935Fh	{GRAPH?}	#6A563h	{EQUATION\}	#6A568h	{EQUAT\RULES}
#6A59Ah	{EQUAT\EDIT}	#6A5BBh	{EQUAT\EXPR}	#6A5DCh	{EQUATION\SUB}
#6A5FBh	{EQUAT\REPL}	#6A61Ch	{EQUAT\EXIT}		

◆ MENUS "RULES":

Quand on édite une expression dans **Equation-Writer**, et que l'on place une sous-expression en *sur-brillance* (on commence par un appui sur la touche "*Flèche-gauche*"), il apparaît un menu dont la première entrée s'appelle **RULES** et qui conduit à un nouveau menu.

Les entrées qui composent ce menu **RULES** dépendent en fait de la sous-expression que vous avez mis en surbrillance.

Voici des listes (toutes placées en ROM cachée) qui constituent les différentes compositions du menu **RULES**. Pour être plus clair, j'ai indiqué quelles étaient ces entrées (entre crochets, car une telle entrée est représentée par son adresse dans la liste).

Dans les deux cas où la liste est écrite sur plusieurs lignes, chaque ligne représente une page du menu **RULES** correspondant.

Adresse	Composition du menu RULES correspondant
#7DBBFh	{ [<-T] [T->] [<-M] [M->] [AF] [<-->] [(<-] [->)] [<-A] [A->] [(())] [-()] }
#7DC05h	{ [<-T] [T->] [<-M] [M->] [<-D] [D->] [(<-] [->)] [<-A] [A->] [<-->] [(())] [-()] [1/()] [L()] }
#7DC5Ah	{ [1/()] [E()] [<-D] [D->] [<-A] [A->] }
#7DC82h	{ [1/()] [E^] [->TRG] [D->] }
#7DCA0h	{ [-()] [L*] [] [D->] }
#7DCBEh	{ [->()] }
#7DCCDh	{ [<-T] [T->] }
#7DCE1h	{ [->()] }
#7DCF0h	{ [->DEF] [TRG*] }
#7DD04h	{ [->DEF] }

STRUCTURES RPL

Quand vous créez un programme sur votre HP48, vous créez un objet placé entre les "*caractères*" « et ».

Cet objet est une **structure RPL** ("*Reverse Polish Language*"), mais ce n'est qu'un cas particulier d'une situation plus large.

La ROM de la HP48 contient un nombre astronomique de tels objets, et une grande partie de ce livre est consacrée à la découverte des plus intéressants d'entre eux.

Nous appelerons ici *structures RPL* les objets-programmes (constitués d'instructions éxécutées en séquence, mais pouvant impliquer çà et là des structures conditionnelles ou répétitives).

L'instruction **TYPE**, appliquée à un tel objet, renverra toujours la valeur 8.

L'adresse-préfixe des RPL est **#02D9Dh**.

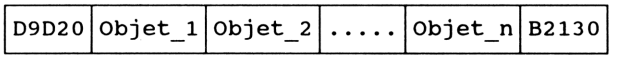
Leur adresse-postfixe est **#0312Bh** (tout comme les listes , les objets-unités, et les expressions).

Entre ces deux extrémités, le RPL est composé d'objets qui seront *évalués en séquence* (sauf boucles, tests, qui peuvent bouleverser cet ordre un peu monotone).

Un tel objet peut être une adresse sur 5 quartets (auquel cas c'est l'objet situé à cette adresse qui est évalué), ou par son contenu (débutant par une adresse préfixe, qui est l'adresse du sous-programme chargé de l'évaluation de cet objet...)

Vous allez trouver que je radote, mais tant pis: les différentes adresses qui interviennent dans un RPL (à commencer par ses adresses préfixe et postfixe) sont codées à l'envers.....

On peut donc représenter, très schématiquement, une structure RPL de la manière suivante:



Le RPL le plus court (et qui ne fait rien) est donc codé **D9D20B2130**

Pour conserver une terminologie assez vague, nous dirons que *Objet_1*, ..., *Objet_n*, sont les "*instructions*" du RPL.

Les programmes que vous créez (dans les conditions habituelles) sont des structures RPL pour lesquelles, en reprenant les notations ci-dessus:

- ***Objet_1*** est l'adresse **#2361Eh**
- ***Objet_n*** est l'adresse **#23639h**

Ces adresses correspondent respectivement aux caractères « et » de début et de fin.

Mais il est très intéressant de constater que les programmes situés à ces adresses sont des RPL ne contenant qu'une seule instruction dont le seul rôle est (je simplifie) d'interrompre le programme si l'appui sur la touche **ON** est détecté.

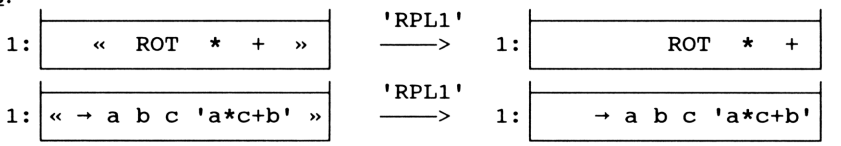
Les deux symboles « et » ont donc surtout un rôle *visuel*; ils permettent à l'utilisateur d'"*enfermer*" un groupe d'instructions dans un même programme, et ils sont importants dans le processus d'évaluation de la ligne de commande (leur rôle est alors de "*contenir*" et de retarder l'évaluation des objets qu'ils renferment, un peu à la manière d'une liste).

Conséquence:
En supprimant les adresses correspondant aux caractères « et », on ne modifie pas le fonctionnement d'un RPL.
On empêche seulement sa saisie et sa modification en ligne de commande (un appui sur **ENTER** se traduirait par l'éxécution immédiate du programme).

Application:
Nous allons voir un programme permettant de supprimer les caractères « et », qui débutent et terminent un programme (au sens habituel). Le programme '**RPL1**' ci-dessous prend un programme au niveau 1 de la pile, et lui enlève son premier objet (normalement «) et son dernier objet (normalement »)

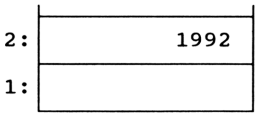
'**RPL1**': (Checksum: #6B81h; Taille 59.5 octets)
« 2 OVER #1CA3Ah SYSEVAL 1 − #1C8CFh SYSEVAL »

Exemples:



Les programmes ainsi obtenus peuvent être stockés dans des variables et fonctionneront de la même manière qu'auparavant.
Il est cependant impossible de les éditer en ligne de commande sans les éxécuter à l'appui sur **ENTER** (à moins de replacer "*à la main*" les caractères « et » qui ont été supprimés par '**RPL1**'.

On peut d'ailleurs appliquer '**RPL1**' au programme réduit à « ». On obtient alors sur la pile le *RPL vide*, qui est tout simplement invisible (mais qui occupe bel et bien un niveau sur la pile).
Faites par exemple: **CLEAR 1992 « » RPL1** et les niveaux 2 et 1 de la pile auront l'aspect suivant (de quoi épater ou inquiéter vos camarades moins bien informés que vous....)



◆ Exemples de structures RPL:

Voyons comment sont codés en mémoire, quelques programmes simples.
Les espaces et passages à la ligne ne servent ici qu'à rendre les listings plus lisibles.

▸ Le programme « **ROT * +** » est codé:

```
D9D20      Adresse-préfixe #02D9Dh des RPL
    E1632      L'adresse de « est #2361Eh
    E0CF1      L'adresse de ROT est #1FC0Eh
    EEDA1      L'adresse de * est #1ADEEh
    76BA1      L'adresse de + est #1AB67h
    93632      L'adresse de » est #23639h
B2130      Adresse-postfixe #0312Bh des RPL
```

Cet exemple montre comme l'écriture *polonaise inverse* conduit à des programmes compacts. Chaque instruction est ici reconnue par la HP48 comme un des ses objets intégrés et est référencée directement au moyen de son adresse en ROM.
Il est évident que ce système de codage impose que ces adresses restent invariables, et ceci quelque soit la version de la HP48: La transmission en binaire utilise en effet les quartets composant le code de l'objet transmis, et ceci sans chercher à interpréter ces quartets.
Si ces adresses changeaient selon la version de la HP48, seules les transmissions en ASCII seraient possibles....

▸ Le programme « **→ a b c 'b+c*a'** » (fonctionnellement identique au précédent) est codé:

```
D9D20  Adresse-préfixe #02D9Dh des RPL
    E1632      L'adresse de « est #2361Eh
    BEF22      L'adresse de → devant 'expr' est #22FEBh
    D6E20 10 16            Nom local 'a'
    D6E20 10 26            Nom local 'b'
    D6E20 10 36            Nom local 'c'
    8BA20      Adresse-préfixe #02AB8h des expressions
        D6E20 10 26        Nom local 'b'
        D6E20 10 36        Nom local 'c'
        D6E20 10 16        Nom local 'a'
        EEDA1              L'adresse de * est #1ADEEh
        76BA1              L'adresse de + est #1AB67h
    B2130      Adresse-postfixe #0312Bh des expressions
    93632      L'adresse de » est #23639h
B2130  Adresse-postfixe #0312Bh des RPL
```

▸ Le programme « → **a b c « b c a * + » »** (là encore identique dans son résultat aux précédents) est codé:

D9D20 Adresse-préfixe #02D9Dh des RPL
E1632 L'adresse de « est #2361Eh
1C432 L'adresse de → devant «...» est #234C1h
D6E20 10 16 Nom local 'a'
D6E20 10 26 Nom local 'b'
D6E20 10 36 Nom local 'c'
E1632 L'adresse de « est #2361Eh
D6E20 10 26 Nom local 'b'
D6E20 10 36 Nom local 'c'
D6E20 10 16 Nom local 'a'
EEDA1 L'adresse de * est #1ADEEh
76BA1 L'adresse de + est #1AB67h
EF532 Adresse de » (comme fin de structure locale)
93632 L'adresse de » est #23639h
B2130 Adresse-postfixe #0312Bh des RPL

Les comparaisons entre les deux programmes précédents révèlent quelques détails fort intéressants:

L'instruction → (qui précède les noms des variables à créer) est à l'adresse **#22FEBh** ou **#234C1h**, suivant que la structure locale de ces noms locaux est une expression ou un programme (la distinction s'établissant au moment de la création du programme en ligne de commande).

L'instruction » est à l'adresse **#235FEh** (et non pas en **#23639h**) quand elle termine une structure locale du type: **→ ... « ... »**.

Là encore tout le travail est effectué par la routine (essentielle) chargée de coder une ligne de commande validée par **ENTER**.

▸ Le programme « **IF DUP THEN INV END** » est codé:

D9D20 Adresse-préfixe #02D9Dh des RPL
E1632 L'adresse de « est #2361Eh
3CE22 L'adresse de IF est #22EC3h
78BF1 L'adresse de DUP est #1FB87h
AFE22 L'adresse de THEN est #22EFAh
872B1 L'adresse de INV est #1B278h
5DF22 L'adresse de END (pour IF) est #22FD5h
93632 L'adresse de » est #23639h
B2130 Adresse-postfixe #0312Bh des RPL

Un examen des routines correspondant à **IF** et **END** réserve une surprise. Ces deux routines font aussi peu de choses que celles qui traitent les "*instructions*" « et » (à savoir interrompre le programme si la touche **ON** est actionnée). Tout le travail revient en fait à la routine qui correspond à **THEN**.

Modifions légèrement le programme précédent.

► Le programme « **DUP IF THEN INV ELSE RAND 2 * END** » est codé:

D9D20 Adresse-préfixe #02D9Dh des RPL
E1632 L'adresse de « est #2361Eh
78BF1 L'adresse de DUP est #1FB87h
3CE22 L'adresse de IF est #22EC3h
AFE22 L'adresse de THEN est #22EFAh
872B1 L'adresse de INV est #1B278h
5BF22 L'adresse de ELSE est #22FB5h
D9D20 Adresse-préfixe #02D9Dh des RPL
9B1C1 L'adresse de RAND est #1C1B9h
ED2A2 L'adresse de 2 est #2A2DEh
EEDA1 L'adresse de * est #1ADEEh
B2130 Adresse-postfixe #0312Bh des RPL
5DF22 L'adresse de END (pour IF) est #22FD5h
93632 L'adresse de » est #23639h
B2130 Adresse-postfixe #0312Bh des RPL

On voit ici que la séquence **RAND 2 *** a été "*enfermée*" dans un RPL de manière à être traitée comme un objet unique (une "*macro-instruction*").

► Le programme: « **WHILE 0 > REPEAT LN END** » est codé:

D9D20 Adresse-préfixe #02D9Dh des RPL
E1632 L'adresse de « est #2361Eh
33032 L'adresse de WHILE est #23033h
4B2A2 L'adresse de 0 est #2A2B4h
D5CE1 L'adresse de > est #1EC5Dh
D5032 L'adresse de REPEAT est #2305Dh
F49B1 L'adresse de LN est #1B94Fh
49632 L'adresse de END (pour WHILE) est #23694h
93632 L'adresse de » est #23639h
B2130 Adresse-postfixe #0312Bh des RPL

Ce listing appelle quelques remarques:
L'instruction **END** qui termine une structure **WHILE..REPEAT..END** est codée à l'adresse **#23694h** (alors que le **END** de **IF..THEN..END** est codé à l'adresse **#22FD5h**).
Les deux instructions **WHILE** et **REPEAT** suffisent à délimiter la séquence **0 >** sans qu'il soit besoin de l'inclure dans un RPL.

Au contraire, si on remplace **LN** par **LN 1 +** , alors le codage du programme est modifié de la manière suivante:

la ligne: **F49B1 L'adresse de LN est #1B94Fh**

sera remplacée par:

D9D20 Adresse-préfixe #02D9Dh des RPL
F49B1 L'adresse de LN est #1B94Fh
9C2A2 L'adresse de 1 est #2A2C9h
76BA1 L'adresse de + est #1AB67h
B2130 Adresse-postfixe #0312Bh des RPL

◆ UN RPL PEU LISIBLE:

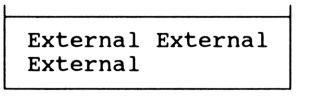
Effectuez la séquence: **{ DEPTH } OBJ→ DROP**

Vous placez ainsi le nom **DEPTH** au niveau 1 de la pile sans l'évaluer. Cet objet est un nom de commande intégrée. l'instruction **TYPE** appliquée à un tel objet renvoie la valeur 19.

Nous allons appliquer l'instruction **NEWOB** à cet objet, mais pas l'instruction **NEWOB** intégrée (qui précisément refuse de générer un nouvelle copie d'un objet référencé en ROM).

L'instruction "***Newob***" que nous allons utiliser est située à l'adresse **#06657h**, et elle s'applique à n'importe quel objet.

Le mot **DEPTH** étant toujours au niveau 1 de la pile, effectuez l'instruction **#6657h SYSEVAL**. Voilà ce que vous devez obtenir sur la pile, au niveau 1.



L'objet obtenu est un RPL (Type=8) qui est l'image de la routine **DEPTH**.

La HP48 nomme par le nom générique **External** tout objet qu'elle ne peut identifier comme étant un objet intégré (fonctions, commandes, les entiers de -9 à 9) ou comme objet identifié par un préfixe (réels, chaînes, tableaux, RPL, etc....).

Ce que signifie le résultat ci-dessus, c'est que la routine traitant l'instruction **DEPTH** est un RPL contenant trois adresses désignant des routines en langage-machine.

D'une façon plus précise, le codage de l'instruction **DEPTH** débute à l'adresse **#1FC44h** et il formé par:

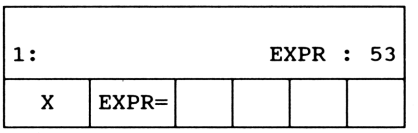
D9D20 Adresse-préfixe #02D9Dh des RPL
E1A81 Adresse #18A1Eh: Vérifie pile non saturée
C4130 Adresse #0314Ch: DEPTH en entier-système
FBD81 Adresse #18DBFh: entier-système vers réel
B2130 Adresse-postfixe #0312Bh des RPL

OBJETS TAGGUÉS

Un **objet taggué** se compose d'un "*tag*" appliqué à un objet quelconque pour en améliorer l'interprétation sur la pile.

La HP48 utilise d'elle même de tels objets taggués, notamment dans le sous-menu **FCN** du menu **GRAPH**, ou dans le menu **SOLVR**...

Si par exemple l'équation courante est **'X^3-3*X+1'**, alors, dans le menu **SOLVR**, un appui sur **EXPR=** (si la valeur de **X** est **4**) donne l'affichage ci dessous:



Remarque: Si, à partir de la pile ci-dessus, vous faites **'E' STO** pour stocker l'objet taggué **EXPR: 53** dans une variable **'E'**, vous constaterez que le contenu de **'E'** n'est que **53**.

L'instruction **STO** "*détaggue*" donc un objet avant de le stocker dans une variable.

Solution: Si vous tenez absolument à stocker un objet taggué dans une variable, il vous faut un petit SYSEVAL!. Plutôt que **STO**, faites donc **#18513h SYSEVAL**... Le tour est joué!

Le codage, en mémoire, d'un objet taggué est le suivant:

- L'adresse-préfixe est **#02AFCh**.
- Elle est suivie par la description du *tag* (d'abord le nombre de caractères qui le compose, codé sur un octet, puis chacun de ces caractères codé sur un octet).
- Le caractère "deux points" **:** (qui sépare le *tag* du *contenu*) n'apparaît qu'à l'affichage, et donc pas dans le codage en mémoire.
- On trouve ensuite la description du contenu de l'objet-taggué.

Le codage de l'objet taggué **EXPR: 53** est donc (comme d'habitude les espaces et passages à la ligne ne sont pas significatifs).

```
CFA20                       L'adresse préfixe #02AFCh.
40                          Le tag possède 4 caractères.
54 85 05 25                 #45h = code de E, ..., #52h = code de R.
33920 100 000000000035 0        Le réel 53.
```

Remarques:

On peut tagguer des objets qui le sont déjà.

La ROM de la HP48 ne contient aucun objet taggué.

OBJETS GRAPHIQUES

Le codage en mémoire d'un objet-graphique est le suivant:

- L'adresse préfixe est **#02B1Eh**
- Elle est suivie de la taille de l'objet, exprimée en quartets, et codée sur 5 quartets (sans compter l'adresse-préfixe).
- Puis vient le nombre de lignes, sur 5 quartets.
- Ensuite vient le nombre de colonnes, sur 5 quartets.
- On trouve enfin la descrition pixel par pixel. Cette description correspond à une lecture:
  1) sur une ligne: de gauche à droite par tranches de 4 pixels.
  2) de la ligne la plus haute à la ligne la plus basse.

Remarque:

A son habitude, la HP48 retourne les données en mémoire.

Il en est ainsi des données sur 5 quartets (adresse-préfixe, nombre de lignes, de colonnes, taille de l'objet). Il en est de même des quartets permettant de coder 4 pixels voisins sur une même ligne de l'objet graphique.

Les nombres de lignes et de colonnes qui sont indiqués dans la structure décrite correspondent aux dimensions exactes de l'objet graphique à l'écran.

On se demande alors pourquoi il est nécessaire de préciser la taille totale de l'objet, en quartets. C'est parce que chaque ligne de l'objet-graphique est en fait codée sur nombre pair de quartets 5 (et donc sur un nombre exact d'octets): les colonnes éventuellement éxédentaires (à la droite de chaque ligne) sont formées de 0 et non prises en compte à l'affichage.

Exemple 1:

Un objet-graphique consistant en une ligne verticale de 7 points tous "*allumés*", et dont la présence sur la pile est signalée par l'indication **Graphic 1x7** , serait codé:

```
E1B20          Adresse-préfixe #02B1Eh des objets-graphiques
D1000          Taille de l'objet, en quartets: #1Dh = 29 = 15 + 7*2
70000          Nombre de lignes: #7h
10000          Nombre de colonnes: #1h
10 10 10 10 10 10 10   Lignes N°1 à N°7.
```

Sur chaque ligne les quartets 1 et 0 définissent un octet égal à 00010000.

Comme il a été dit, le 1er quartet doit être lu "*à l'envers*", et donne donc un premier pixel à 1, les 3 autres étant à 0.

Le 2-ème quartet (lu à l'envers ou pas!) donne 4 pixels tous nuls.

Finalement, chaque ligne est formée de 1 pixel allumé, suivi de 7 pixels éteints. En fait comme l'indique le nombre de colonnes indiqué au début de la structure, seule la première colonne de pixels sera prise en considération à l'affichage.

Exemple 2:

la séquence **"S" 1 →GROB** produit l'image graphique du caractère "**S**", avec une taille minimum. Cet objet-graphique s'écrit **Graphic 6x8** sur la pile. Il a donc 6 colonnes et 8 lignes.

Il est codé:

E1B20	Adresse-préfixe #02B1Eh des objets-graphiques
F1000	Taille de l'objet, en quartets: #1Fh = 31 = 15 + 8*2
80000 60000	Il y a 8 lignes et 6 colonnes
E0 11 10 E0 01 11 E0 00	Les 8 lignes de l'objet

Ici, on constate que 2 quartets sont utiles pour coder une ligne. Il faut lire à l'envers chacun d'eux. On peut ainsi reconstituer le caractère "**S**".

```
Ligne N°1: #E0h = #11100000b      ==>      o▓▓▓ooo
Ligne N°2: #11h = #00010001b      ==>      ▓ooo▓ooo
Ligne N°3: #10h = #00010000b      ==>      ▓ooooooo
Ligne N°4: #E0h = #11100000b      ==>      o▓▓▓ooo
Ligne N°5: #01h = #00000001b      ==>      oooo▓ooo
Ligne N°6: #11h = #00010001b      ==>      ▓ooo▓ooo
Ligne N°7: #E0h = #11100000b      ==>      o▓▓▓ooo
Ligne N°8: #00h = #00000000b      ==>      ooooooo
```

◆ OBJETS-GRAPHIQUES DANS LA ROM:

Voici les adresses, les tailles (CxL = colonnesxLignes), ainsi qu'une description très sommaire, des objets-graphiques qui figurent en ROM. Faites un SYSEVAL à leur adresse et placez les dans **PICT** (parfois sur un fond noir) pour voir à quoi ils ressemblent vraiment).

Adresse	CxL	Description sommaire de l'objet-graphique
#13D8Ch	6x10	Curseur en mode insertion.
#13DB4h	6x10	Curseur en mode recouvrement.
#39B2Dh	131x2	Ligne noire sur ligne blanche.
#3A337h	21x8	Rectangle blanc, surmonté ligne noire.
#3A399h	21x8	Idem que précédent, plus petit carré noir.
#3A3FBh	21x8	Idem #3A337, plus petite barre pour répertoire.
#3A45Dh	21x8	Rectangle blanc, avec lignes noires sup. et inf.
#5053Ch	5x5	Réticule en croix (environnement GRAPH).
#5055Ah	5x5	Marque (environnement GRAPH)
#505B2h	0x0	Objet graphique vide: E1B20 F0000 00000 00000
#66EA5h	6x10	Rectangle blanc, contenant cadre noir 5x9
#66ECDh	6x8	Cadre noir 6x8, avec intérieur blanc.
#66EF1h	4x6	Cadre noir 4x6, avec intérieur blanc.
#66F11h	6x8	Rectangle blanc.
#66F35h	6x10	Rectangle blanc.
#66F5Dh	7x5	Symbole d'exponentiation, en vidéo inversée.
#66F7Dh	5x4	Symbole d'exponentiation, en vidéo inversée.

OBJETS "CODE"

Les "***Code Objects***" sont des objets particuliers de la HP48 destinés à recevoir des séquences écrites directement en langage machine.
La commande **TYPE**, appliquée à un tel objet, renvoie la valeur 25.

Les "***Codes Objects***" ne sont pas affichés "*en clair*" sur l'écran.
La présence d'un tel objet sur la pile est signalée par le terme générique **Code**.

La structure d'un "***Code Object***" est la suivante:
- Une adresse-préfixe **#02DCCh** (codée "*à l'envers*" sur 5 quartets).
- La longueur de l'objet (codée "*à l'envers*" sur 5 quartets).
- La séquence des codes hexadécimaux constituant la partie éxécutable de l'objet.

Un objet-code peut être évalué (éxécuté) comme un programme, mais il faut être prudent (ne lancez pas de telles évaluations au hasard si vous n'aimez pas trop les *Memory Clear*).
Vous pouvez éxécuter un objet-code en effectuant un SYSEVAL à son adresse (si vous êtes un *kamikaze*), et vous pouvez placer ces objets sur la pile en plaçant leur adresse en binaire sur la pile et en effectuant:

#5A03h SYSEVAL #C612h SYSEVAL

Exemple:

- L'objet-code situé à l'adresse #1A556h s'écrit:

CCD20	D1000	147174E7137174143135808C

- La longueur de cet objet est donc #1Dh=29 (5 quartets pour coder cette longueur et 24 quartets de langage-machine).
- La signification de cet objet-code est d'effectuer un SYSEVAL à l'adresse spécifiée au niveau 1 sous forme d'entier-système.
- Voici le détail commenté du contenu de cet objet-code.

Adresse	Code	Mnémonique	Interprétation
#1A560h	147	C=DAT1 A	C <-- Adresse du niveau 1 de la pile
#1A563h	174	D1=D1+5	D1 pointe sur prochain niveau de pile
#1A566h	E7	D=D+1 A	Mémoire libre augmente de 5 quartets.
#1A568h	137	CD1EX	Échange contenus de C et de D1.
#1A56Bh	174	D1=D1+5	D1 pointe sur contenu entier-système.
#1A56Eh	143	A=DAT1 A	A <-- contenu de l'entier-système.
#1A571h	135	D1=C	D1 repointe sur niveau 1 de la pile.
#1A574h	808C	PC=(A)	Saut à l'adresse lue à l'adresse contenue dans A

◆ Objets-Code en ROM:

Voici maintenant les adresses des différents "*Code Objects*" qui apparaissent dans la mémoire morte de la HP48.

j'ai indiqué les 10 premiers quartets de chaque objet. les quartets 5 à 10 permettent de connaître la longueur en quartets de l'objet (enlevez 5 à cette valeur pour connaître le nombre exact de quartets utilisés pour le langage-machine).

Adresse	Début de l'objet	Adresse	Début de l'objet
#021CCh	CCD20 31000 ...	#0BC6Fh	CCD20 B5000 ...
#0BCEDh	CCD20 A3000 ...	#0EB8Bh	CCD20 31000 ...
#0EE5Fh	CCD20 A1000 ...	#0EE92h	CCD20 43000 ...
#0EEDFh	CCD20 C5000 ...	#0EF72h	CCD20 86000 ...
#0F138h	CCD20 22000 ...	#10B9Fh	CCD20 72000 ...
#10BE4h	CCD20 72000 ...	#10C4Ch	CCD20 05000 ...
#128BAh	CCD20 42000 ...	#128E3h	CCD20 86000 ...
#1415Fh	CCD20 53000 ...	#14565h	CCD20 C3000 ...
#145BAh	CCD20 63000 ...	#15694h	CCD20 C4000 ...
#17AEAh	CCD20 29000 ...	#17F59h	CCD20 12000 ...
#1A556h	CCD20 D1000 ...	#1A7EDh	CCD20 A5000 ...
#21097h	CCD20 F4000 ...	#27D32h	CCD20 A4000 ...
#27D9Ah	CCD20 F6000 ...	#2C158h	CCD20 21000 ...
#2C17Eh	CCD20 21000 ...	#2C1A9h	CCD20 21000 ...
#2C4D7h	CCD20 45000 ...	#2D544h	CCD20 52000 ...
#2DE64h	CCD20 52000 ...	#2E117h	CCD20 07100 ...
#2E2E1h	CCD20 F6100 ...	#2EDB0h	CCD20 72000 ...
#3017Ah	CCD20 CE100 ...	#303B6h	CCD20 7B000 ...
#30486h	CCD20 03100 ...	#30634h	CCD20 2F000 ...
#308A7h	CCD20 36000 ...	#30928h	CCD20 AB000 ...
#30A0Ah	CCD20 FD000 ...	#30B80h	CCD20 43000 ...
#30D40h	CCD20 AF000 ...	#31449h	CCD20 E7000 ...
#315CBh	CCD20 F1000 ...	#31EFBh	CCD20 B3000 ...
#323B4h	CCD20 B3000 ...	#32B88h	CCD20 81100 ...
#33DF0h	CCD20 26000 ...	#33E7Fh	CCD20 92000 ...
#34961h	CCD20 B5000 ...	#34CF1h	CCD20 03000 ...
#3975Ch	CCD20 64000 ...	#52C8Ch	CCD20 09000 ...
#52D49h	CCD20 73000 ...	#52D8Ah	CCD20 C2000 ...
#52DC5h	CCD20 14000 ...	#52E2Eh	CCD20 5F000 ...
#52F2Dh	CCD20 F2000 ...	#52F7Fh	CCD20 BD000 ...
#5309Bh	CCD20 76100 ...	#5320Ch	CCD20 84200 ...
#5346Dh	CCD20 75000 ...	#534F1h	CCD20 B2000 ...
#53521h	CCD20 59000 ...	#5792Eh	CCD20 91000 ...
#5794Ch	CCD20 91000 ...	#648EFh	CCD20 91200 ...
#66FADh	CCD20 72000 ...		

NOMS XLIB

Au sein d'une même librairie, un objet est désigné par un numéro. Les ***noms XLIB*** sont les objets permettant d'identifier une commande appartenant à une librairie, par la donnée du numéro de cette librairie, puis du numéro de la commande.

La présence d'un ***nom XLIB*** sur la pile se traduit par le nom générique **XLIB**, suivi des deux nombres évoqués ci-dessus. Un ***nom XLIB*** peut être évalué comme n'importe quel nom (évaluation à faire avec prudence, car les "*Try To Recover Memory*" ne sont pas rares).

◆ LE CODAGE DES NOMS XLIB:

La commande **TYPE**, appliquée à un ***nom XLIB***, donne le résultat 14.

Les ***noms XLIB*** sont codés en mémoire de la façon suivante:

- Une adresse préfixe: **#29E20h** (codée "à l'envers" sur 5 quartets).
- Le numéro de librairie, puis le numéro d'objet dans la librairie (ces deux nombres étant codés "à l'envers" sur trois quartets).

Exemple:

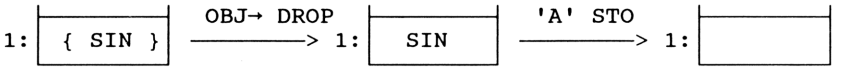
Le nom **XLIB 1000 73** est codé: **02E92 8E3 940** (1000=#3E8h, 73=#49h)

◆ Un "BUG" dans l'instruction BYTES ? :

Pour stocker une commande (ou une fonction) intégrée dans une variable, placez cette commande dans une liste et faites **OBJ→ DROP**.

De cette manière vous obtenez le nom de la commande au niveau 1.

La séquence ci-dessous place la fonction intégrée **SIN** dans la variable **'A'**.



Un examen attentif de la mémoire montre que le contenu de la variable **'A'** est: **29E20 200 440**.

Autrement dit le contenu de **'A'** est le ***nom XLIB*** correspondant au 68-ème objet de la 2-ème librairie (c'est bien la fonction **SIN**).

La taille de la variable **'A'** devrait être 11 octets, mais l'instruction **BYTES** annonce 33 octets...

Mieux: si on stocke la commande + dans la variable **'A'**, la séquence **'A' BYTES** donne une taille de 138 octets ! (au lieu des 11 octets réellement occupés par cette variable). Il apparaît que l'instruction **BYTES** prend ici en compte la taille de la routine gérant l'opération + en mémoire (et qui est effectivement assez longue).

Il est difficile de ne pas y voir une erreur de la commande **BYTES**.

◆ UNE ROUTINE POUR VOIR DES NOMS XLIB:

Le programme ci-dessous vous permet de placer sur la pile le ***nom XLIB*** figurant en **p**-ème position dans la librairie **L** (p et L doivent être donnés sous forme d'entiers).

Dans le cas où l'objet ainsi référencé est une fonction ou une commande intégrée à la HP48, c'est le nom usuel de cet objet qui est obtenu.

'XLB':(Checksum #A9BCh; Taille 69 octets)

```
« SWAP #2EC11h SYSEVAL SWAP
  #2EC11h SYSEVAL #7E50h SYSEVAL
»
```

Exemples:

- 2:458 1:25 ---[XLB]---> 1:XLIB 458 25
  (une tentative d'évaluation conduit à "*Undefined XLIB Name*")

- 2:2 1:96 ---[XLB]---> 1:ALOG
  Ce résultat signifie que **ALOG** est le 96-ème objet de la librairie N°2.

- 2:1792 1:5 ---[XLB]---> 1:WHILE
  Ce résultat signifie que **WHILE** est le 5-ème objet de la librairie N°1792=#700h.

◆ LA LIBRAIRIE 240:

Dans le domaine des ***noms XLIB***, l'exploration du contenu de la ROM de la HP48 réserve quelques surprises: On ne trouve d'abord aucun ***nom XLIB*** correspondant aux librairies #2h et #700h intégrées à la HP48.

Au contraire, tous les ***noms XLIB*** rencontrés se réfèrent à une librairie dont le numéro est 240. J'ai fini par m'apercevoir que ces ***noms XLIB*** servaient à accéder aux routines RPL de la ROM cachée.

La plupart de ces routines gèrent des opérations de réarrangement d'expresssions algébriques (telles que celles permises par le menu **RULES** dans l'environnement **EQUATION-WRITER**). D'autres permettent les échanges avec les ports 0, 1, et 2.

NOMS XLIB

Le tableau ci-après donne toutes les adresses où on trouve des ***noms XLIB*** de cette librairie. Ces noms permettent d'éxécuter des routines dans la ROM cachée.

J'ai donné là encore les adresses de ces routines (certaines adresses sont manquantes: la raison en est qu'elles ont changé entre la version A et la version E de la HP48).

Dans la deuxième partie de ce livre, nous verrons quelques SYSEVALs intéressants qui s'appliquent aux ***noms XLIB***.

Adresse	Nom XLIB	éxécute	Adresse	Nom XLIB	éxécute
#21558h	XLIB 240 93		#21572h	XLIB 240 93	
#2158Ch	XLIB 240 93		#215C9h	XLIB 240 95	
#2176Bh	XLIB 240 96		#21776h	XLIB 240 103	
#217D1h	XLIB 240 96		#217DCh	XLIB 240 107	
#217FBh	XLIB 240 96		#21806h	XLIB 240 109	
#21B39h	XLIB 240 99		#21B64h	XLIB 240 100	
#21B7Eh	XLIB 240 101		#21D68h	XLIB 240 102	
#21D9Bh	XLIB 240 111		#596DDh	XLIB 240 92	
#5BE92h	XLIB 240 5	#7E372h	#5BEA2h	XLIB 240 6	#7E38Bh
#5BEB2h	XLIB 240 7	#7E3B3h	#5BEC2h	XLIB 240 8	#7E3CCh
#5BEF3h	XLIB 240 9	#7E3F4h	#5BF03h	XLIB 240 10	#7E40Dh
#5BF27h	XLIB 240 11	#7E426h	#5BF37h	XLIB 240 12	#7E43Fh
#5BF47h	XLIB 240 13	#7E458h	#5BF78h	XLIB 240 14	#7E476h
#5BF88h	XLIB 240 15	#7E48Fh	#5BFACh	XLIB 240 16	#7E4A8h
#5BFBCh	XLIB 240 17	#7E4C1h	#5BFCCh	XLIB 240 18	#7E4C1h
#5BFFDh	XLIB 240 19	#7E4DAh	#5C00Dh	XLIB 240 20	#7E4DAh
#5C01Dh	XLIB 240 21	#7E502h	#5C02Dh	XLIB 240 22	#7E51Bh
#5C03Dh	XLIB 240 23	#7E534h	#5C04Dh	XLIB 240 23	#7E534h
#5C05Dh	XLIB 240 24	#7E54Dh	#5C06Dh	XLIB 240 24	#7E54Dh
#5C07Dh	XLIB 240 25	#7E566h	#5C08Dh	XLIB 240 25	#7E566h
#5C09Dh	XLIB 240 26	#7E57Fh	#5C0ADh	XLIB 240 26	#7E57Fh
#5C15Ch	XLIB 240 27	#7E598h	#5C16Ch	XLIB 240 28	#7E598h
#5C17Ch	XLIB 240 29	#7E5ACh	#5C18Ch	XLIB 240 29	#7E5ACh
#5C1B0h	XLIB 240 30	#7E5C5h	#5C1C0h	XLIB 240 31	#7E5C5h
#5C1D0h	XLIB 240 31	#7E5C5h	#5C215h	XLIB 240 32	#7E5EDh
#5C225h	XLIB 240 33	#7E606h	#5C235h	XLIB 240 34	#7E61Fh
#5C245h	XLIB 240 35	#7E61Fh	#5C255h	XLIB 240 35	#7E61Fh
#5C272h	XLIB 240 36	#7E647h	#5C282h	XLIB 240 36	#7E647h
#5C292h	XLIB 240 39	#7E66Fh	#5C2A2h	XLIB 240 40	#7E683h
#5C2B2h	XLIB 240 37	#7E65Bh	#5C2C2h	XLIB 240 38	#7E65Bh
#5C2DFh	XLIB 240 41	#7E697h	#5C2EFh	XLIB 240 41	#7E697h
#5C2FFh	XLIB 240 42	#7E6B0h	#5C30Fh	XLIB 240 42	#7E6B0h
#5C32Ch	XLIB 240 43	#7E6D3h	#5C33Ch	XLIB 240 43	#7E6D3h
#5C359h	XLIB 240 44	#7E6ECh	#5C369h	XLIB 240 44	#7E6ECh
#5C386h	XLIB 240 45	#7E705h	#5C396h	XLIB 240 45	#7E705h
#5C3A6h	XLIB 240 46	#7E723h	#5C3B6h	XLIB 240 46	#7E723h
#5C3D3h	XLIB 240 52	#7E82Bh	#5C3E3h	XLIB 240 53	#7E84Eh
#5C3F3h	XLIB 240 54	#7E871h	#5C403h	XLIB 240 54	#7E871h
#5C413h	XLIB 240 55	#7E88Ah	#5C423h	XLIB 240 55	#7E88Ah
#5C433h	XLIB 240 56	#7E8A3h	#5C443h	XLIB 240 56	#7E8A3h
#5C453h	XLIB 240 57	#7E8BCh	#5C463h	XLIB 240 57	#7E8BCh
#5C473h	XLIB 240 47	#7E74Bh	#5C483h	XLIB 240 48	#7E74Bh

Adresse	Nom XLIB	exécute	Adresse	Nom XLIB	exécute
#5C493h	XLIB 240 47	#7E74Bh	#5C4A3h	XLIB 240 48	#7E74Bh
#5C4B3h	XLIB 240 47	#7E74Bh	#5C4C3h	XLIB 240 48	#7E74Bh
#5C4E0h	XLIB 240 58	#7E8D5h	#5C4F0h	XLIB 240 58	#7E8D5h
#5C500h	XLIB 240 58	#7E8D5h	#5C510h	XLIB 240 58	#7E8D5h
#5C520h	XLIB 240 58	#7E8D5h	#5C530h	XLIB 240 58	#7E8D5h
#5C54Dh	XLIB 240 2	#7E272h	#5C55Dh	XLIB 240 2	#7E272h
#5C56Dh	XLIB 240 4	#7E327h	#5C57Dh	XLIB 240 4	#7E327h
#5C59Ah	XLIB 240 63	#7E9E8h	#5C5AAh	XLIB 240 63	#7E9E8h
#5C5BAh	XLIB 240 64	#7EAA1h	#5C5CAh	XLIB 240 64	#7EAA1h
#5C5E7h	XLIB 240 65	#7EB5Ah	#5C5F7h	XLIB 240 65	#7EB5Ah
#5C607h	XLIB 240 66	#7EC13h	#5C617h	XLIB 240 66	#7EC13h
#5C634h	XLIB 240 67	#7ECCCh	#5C644h	XLIB 240 67	#7ECCCh
#5C654h	XLIB 240 68	#7ED67h	#5C664h	XLIB 240 68	#7ED67h
#5C681h	XLIB 240 73	#7EEC0h	#5C6CDh	XLIB 240 0	#7E128h
#5C6F1h	XLIB 240 61	#7E998h	#5C701h	XLIB 240 61	#7E998h
#5C711h	XLIB 240 62	#7E9C0h	#5C721h	XLIB 240 62	#7E9C0h
#5C731h	XLIB 240 1	#7E1C3h	#5C755h	XLIB 240 69	#7EE02h
#5C765h	XLIB 240 69	#7EE02h	#5C775h	XLIB 240 69	#7EE02h
#5C785h	XLIB 240 69	#7EE02h	#5C795h	XLIB 240 69	#7EE02h
#5C7A5h	XLIB 240 69	#7EE02h	#5C7B5h	XLIB 240 69	#7EE02h
#5C7C5h	XLIB 240 69	#7EE02h	#5C7D5h	XLIB 240 70	#7EE02h
#5C7E5h	XLIB 240 70	#7EE02h	#5C7F5h	XLIB 240 70	#7EE02h
#5C805h	XLIB 240 71	#7EE02h	#5C815h	XLIB 240 71	#7EE02h
#5C825h	XLIB 240 71	#7EE02h	#5C85Dh	XLIB 240 74	#7EF0Bh
#5C86Dh	XLIB 240 75	#7EF60h	#5C87Dh	XLIB 240 76	#7EFABh
#5C88Dh	XLIB 240 80	#7F195h	#5C89Dh	XLIB 240 81	#7F1D6h
#5C8ADh	XLIB 240 82	#7F208h	#5C8BDh	XLIB 240 77	#7F014h
#5C8CDh	XLIB 240 78	#7F082h	#5C8DDh	XLIB 240 79	#7F0FFh
#5C8EDh	XLIB 240 83	#7F249h	#5C8FDh	XLIB 240 84	#7F2B2h
#5C90Dh	XLIB 240 85	#7F33Eh	#5C935h	XLIB 240 86	#7F3A7h
#5C945h	XLIB 240 86	#7F3A7h	#5C955h	XLIB 240 87	#7F41Fh
#5C965h	XLIB 240 87	#7F41Fh	#5C975h	XLIB 240 89	#7F528h
#5C985h	XLIB 240 89	#7F528h	#5C995h	XLIB 240 90	#7F5A0h
#5C9A5h	XLIB 240 90	#7F5A0h	#5C9B5h	XLIB 240 91	#7F618h
#5C9C5h	XLIB 240 91	#7F618h	#5C9D5h	XLIB 240 88	#7F4B5h
#5C9E5h	XLIB 240 88	#7F4B5h	#5D2BBh	XLIB 240 59	#7E8EEh
#5D532h	XLIB 240 60	#7E943h	#5D7A3h	XLIB 240 59	#7E8EEh
#5D7FEh	XLIB 240 60	#7E943h	#5DCF0h	XLIB 240 72	#7EE5Ch
#5DD1Eh	XLIB 240 72	#7EE5Ch	#5FC6Fh	XLIB 240 49	#7E787h
#5FC89h	XLIB 240 49	#7E787h	#7E29Ah	XLIB 240 3	#7E2CDh
#7E791h	XLIB 240 50	#7E7E0h	#7E7A6h	XLIB 240 51	#7E812h
#7E7B1h	XLIB 240 50	#7E7E0h	#7E7D0h	XLIB 240 50	#7E7E0h

LIBRAIRIES

Les ***librairies*** (ou "*bibliothèques*") sont des objets regroupant des objets nommés, pouvant être utilisés comme extension au jeu de commandes.

Il y a une grande analogie entre les librairies et les répertoires, en ceci au moins que les deux structures sont formées d'objets auxquels on a donné un nom. Il y a évidemment des différences:

- Un répertoire réside en mémoire vive (RAM) et peut être modifié en permanence.

- A contrario, une librairie est plutôt un "*package*" de commandes ou de fonctions parfaitement au point, et qu'il n'est plus nécessaire de modifier. La place logique d'une librairie est en mémoire morte (ROM).

- La recherche d'un objet, dans un répertoire, s'effectue par un parcours séquentiel sur les noms.

- Dans une librairie, les noms des objets sont purement descriptifs (ils n'apparaissent réellement que sur les touches de menus).
  Les objets y sont accessibles via une table d'adresses, ce qui permet de les localiser de manière indexée, et donc plus rapidement.

- Les répertoires peuvent contenir des sous-répertoires. Au contraire les librairies sont des objets "*plats*".
  La notion de sous-librairie n'a donc aucun sens.

Les librairies ne peuvent être créées par la HP48 elle-même. Pour créer de tels objets (fort complexes au demeurant) il faut utiliser des logiciels spécifiques sur micro (le programme **USRLIB.EXE**, sous copyright de **Hewlett-Packard**, est fait pour ça).

◆ Les numéros de librairies:

Une librairie est référencée par un numéro. La HP48 utilise *de facto* deux librairies de numéros #002h et #700h. Ces deux librairies (la première est de loin la plus volumineuse) contiennent les différentes commandes et fonctions intégrées à la HP48.

Les numéros de librairie sont compris entre #0 et #7FFh=2047.

C'est par son numéro qu'une librairie est identifiée (et non par son nom). Deux librairies différentes ne peuvent donc pas posséder le même numéro.

Les numéros de librairies (entre #0h et #700h) sont soumis à certaines obligations:

- Les N° 257 (#101h) à 768 (#300h) sont réservés aux applications HP.
- Les N° 769 (#301h) à 1536 (#600h) sont réservés aux applications non HP (mais recevant leur aval).
- Les N° 1537 (#601h) à 1791 (#6FFh) ne souffrent aucune restriction.
- Les N° 1792 (#700h) à 2047 (#7FFh) sont utilisés par la ligne de commande de la HP48.

◆ Attacher et détacher une librairie:

Une librairie peut être attachée à un répertoire quelconque (un sous répertoire de **HOME** ne peut se voir attacher qu'une seule librairie, mais **HOME** lui-même n'est pas soumis à cette restriction), au moyen de l'instruction **ATTACH**, et elle peut être détachée au moyen de l'instruction **DETACH**.

Pour être "*attachable*", une librairie doit être stockée dans un port quelconque (le port 0 en mémoire centrale, ou l'un des deux ports 1 ou 2 s'il s'agit d'une librairie placée sur une carte d'extension).

Le fait de stocker la librairie dans un port ne suffit pas. Il faut éteindre et allumer la HP48 pour que ce nouvel objet soit pris en compte (et pour que le numéro de la librairie apparaisse dans le menu **PORT0**, ou **PORT1**, ou **PORT2**).

◆ Le domaine de visibilité d'une librairie:

Si une librairie est attachée à un sous-répertoire de **HOME**, elle n'est réellement visible que dans ce sous-répertoire et dans toute sa "*descendance*". Si elle est attachée à **HOME**, elle est visible partout.

Quand elle est attachée à un répertoire, les objets (qui ne sont pas cachés: voir plus loin) dont elle est constituée viennent compléter le jeu d'instructions de la HP48.

Il est impossible d'accéder au contenu de ces objets. Il est également impossible de placer le nom d'un tel objet sur la pile sans l'évaluer.

Néanmoins, si une librairie est attachée à un répertoire et s'il se pose un problème d'homonymie (2 objets ayant le même nom, l'ancien dans le répertoire et le nouveau dans la librairie), c'est le nom du répertoire qui a priorité.

◆ Le nom d'une librairie:

Une librairie peut posséder un nom. Ce nom ne sert pas à "*appeler*" la librairie, mais le début de ce nom apparaît dans un label du menu **LIBRARY**. La totalité du nom apparaîtra si on éxécute **REVIEW** dans ce menu.

◆ L'objet de configuration:

Une librairie peut contenir un objet dit "*de configuration*".

Cet objet est évalué à chaque fois que la HP48 refait le tour de ses librairies, notamment à chaque **ON-C**, ou si vous allumez votre calculatrice après avoir modifié la configuration des ports 1 ou 2.

Le principal intérêt de cet objet est de permettre d'auto-attacher la librairie. Si celle-ci porte le numéro 1500 par exemple, on pourra placer le programme « **1500 ATTACH** » dans l'objet de configuration, rendant ainsi la librairie "*auto-attachable*" (elle n'en restera pas moins détachable).

Le contenu de l'objet de configuration ne peut cependant pas être quelconque. Il ne doit rien prendre sur la pile, rien y laisser, et il ne doit pas provoquer d'erreur (sinon c'est le "*crash*").

◆ La table des messages:

Une librairie peut être accompagnée d'une table de messages. Les différents messages constituant cette table doivent être placés dans un vecteur de chaînes de caractères (voir plus loin)

◆ Le codage des librairies:

L'adresse préfixe des librairies est **#02B40h**. Voici comment on peut représenter, schématiquement, le codage en mémoire d'une librairie.

La toute première colonne sert uniquement à marquer quelques repères. Par exemple "**j**" désigne ici l'adresse de l'objet de configuration. Les adresses vont en augmentant ($a < b < c < \ldots < \alpha$).

	Codage en mémoire	Commentaires
	40B20	#02B40h = Adresse-préfixe des librairies
a	Taille	Sur 5 quartets: a+Taille=k+4 (après checksum)
	Zone_du_nom	Voir plus loin
	Numéro	Numéro de la librairie, sur 3 quartets
b	Offset_To_Hash	Sur 5 quartets: b + Offset_To_Hash = g
c	Offset_To_Mess	Sur 5 quartets: c + Offset_To_Mess = h
d	Offset_To_Link	Sur 5 quartets: d + Offset_To_Link = i
e	Offset_To_Config	Sur 5 quartets: e + Offset_To_Config = j
f	Zone_des_Objets	L'ordre dans lequel apparaissent ces cinq zones est quelconque. Les offsets qui précèdent permettent de toutes façons d'accéder aux informations utiles.
g	Hash_Table	
h	Message_Table	
i	Link_Table	
j	Config_Object	
k	CheckSum	Somme de contrôle interne (4 quartets) calculée sur la zone [a,k+4]. Rien à voir avec le résultat de BYTES.

Entrons un peu plus dans le détail:

◆ La "Zone_du_Nom":

Si la librairie ne porte pas de nom (encore une fois, on identifie une librairie par un numéro; le nom éventuel apparaît comme label dans le menu **LIBRARY**), cette zone se réduit à 2 quartets nuls.

Si la librairie est nommée, cette zone s'écrit:

n	Name	n

Ici n désigne le nombre de caractères du nom de la librairie (sur deux quartets). On remarque que n est répété après "*Name*".

"*Name*" est la suite des caractères composant le nom de la librairie (chaque caractère occupe un octet, c-à-d deux quartets)

◆ Zone_des_Objets:

Chaque objet de la librairie occupe une zone du type:

Alg?	N_lib	N_obj	ObJ

- Le champ **Alg?** permet de distinguer les objets algébriques (c-à-d utilisables dans une expression).
  Alg? contient 000 (3 quartets nuls) si l'objet est algébrique.
  Alg? contient 8 si l'objet n'est pas algébrique.
- Le champ **N_lib** contient le numéro de la librairie, sur 3 quartets.
- Le champ **N_obj** contient le numéro de l'objet dans la librairie, codé sur trois quartets. Les 2 champs *N_lib*, *N_obj* indiquent donc les composantes du ***nom XLIB*** associé à cet objet.
- Le champ **OBJ** contient bien sûr le corps de l'objet.

Il ne me semble pas obligatoire que les zones correspondant aux différents objets de la librairie soient adjacentes (que ces objets soient cachés ou non). En effet, comme on va le voir dans un instant, on accède à l'adresse de chaque objet par un pointeur situé dans la **Link-Table** (elle-même accessible au moyen d'un offset vu précédemment).

◆ La Link-Table:

C'est elle qui permet de retrouver instantanément l'adresse où se trouve le corps d'un objet de la librairie, connaissant le numéro de cet objet. L'adresse obtenue est celle du corps de l'objet (celle du champ "**OBJ**" dans la terminologie précédente).

La "*Link-Table*" est en fait un entier binaire long (on sait que la taille d'un entier binaire n'est pas nécessairement limitée aux 16 chiffres hexadécimaux habituels).

La structure de la *Link-Table* est la suivante, en supposant que la librairie est formée de K+1 objets (cachés ou non), numérotés de 0 à K.

Dans la première colonne, α, ß, etc... désignent des adresses en mémoire (mais n'ont rien à voir avec le contenu de la *Link-Table*).

Le contenu de la *Link_Table* est représenté dans la deuxième colonne. Les "*Offsets*" sont des champs de cinq quartets, contenant le déplacement à appliquer à l'adresse courante pour accéder au corps d'un objet de la librairie (voir exemple plus loin !)

Adresses	Contenu	Commentaires
α	E4A20	#02A4E = adresse-préfixe des binaires
α+ 5	Taille	Sur 5 quartets: Taille = 5 + 5*(K+1)
ß = α+10	Offset_0	ß + Offset_0 = adresse du corps objet N°0
σ = ß+ 5	Offset_1	σ + Offset_1 = adresse du corps objet N°1
μ = ß+10	Offset_2	μ + Offset_2 = adresse du corps objet N°2
.....	.....	.....
δ = ß+5K	Offset_K	δ + Offset_K = adresse du corps objet N°K

◆ La table des messages:

La librairie peut très bien n'être accompagnée d'aucun message d'erreur particulier. Dans ce cas le champ "*Offset_To_Mess*" évoqué plus haut est réduit à cinq quartets nuls (il en est d'ailleurs de même du champ "*Offset_To_Config*" en l'absence de tout objet de configuration).

Dans le cas où la librairie est munie de messages d'erreurs particuliers, ces messages sont numérotés (à partir de 1).

Pour provoquer une erreur affichant le message N°**k**, il faudra éxécuter (dans un objet de la librairie, ou à l'extérieur pourvu que la librairie soit attachée) l'instruction **DOERR** avec l'argument **256*N+k**, où **N** est le numéro de librairie.

Par exemple, si la librairie porte le numéro 1515 (#5EBh), alors la séquence **#5EB01h DOERR** éxécute l'erreur N°1 de la librairie, avec le message correspondant.

Les messages d'erreur de la librairie sont groupés dans un vecteur de chaînes de caractères (comme nous en avons vus dans le chapitre consacré aux tableaux)

Voici quelle est donc la structure de la table des messages:

8E920	#02920h est l'adresse-préfixe des tableaux.
Taille	Taille du tableau, hors préfixe, sur 5 quartets.
C2A20	#02A2Ch est l'adresse-préfixe des chaînes.
10000	Il s'agit bien d'un vecteur (une seule dimension).
Nombre N	Nombre de messages, sur 5 quartets.
Size_1	La taille Size_k du message N°k, sur 5 quartets,
Message_1	mesure le nombre de quartets occupés par le
Size_2	message, y compris les 5 quartets de Size_k.
Message_2	Le message N°k est ensuite représenté par les codes
.....	de ses caractères (chaque caractère vaut 1 octet
Size_N	= 2 quartets, ces deux quartets étant inversés)
Message_N	

◆ La Hash-Table:

C'est la table la plus complexe de la librairie.

On a vu que la *Link_Table* permettait de trouver immédiatement le corps d'un objet de la librairie, dès lors que l'on connaissait le numéro de cet objet. Il s'agit alors d'éxécuter cet objet, et non pas d'en connaître le nom.

Tel est le cas dans les objets de la librairie, s'ils s'appellent entre eux (ils sont représentés, comme on le verra dans l'exemple qui suit, par leur ***nom XLIB***).

Pour que la *Link_Table* soit utile, il faut "lui donner" un numéro d'objet (c'est-à-dire un indentificateur **XLIB**: cf chapitre précédent).

Que se passe-t-il quand on utilise un nom désignant un objet de la librairie dans un programme, ou quand on l'entre en ligne de commande?

Ce nom est analysé et la HP48 vérifie s'il appartient au répertoire en cours (ou à l'ascendance de ce répertoire), puis s'il appartient aux différentes librairies.

Il faut donc à la HP48 un moyen rapide lui permettant de savoir si un nom donné représente ou pas un objet d'une librairie.

Inversement, dans la phase de décompilation d'un programme où vous avez inclus des objets d'une librairie (dans ce programme, ces objets sont codés sous forme de ***noms XLIB***) il est nécessaire de traduire rapidement un ***nom XLIB*** donné en un nom (au sens habituel) apparaissant en ligne de commande (si vous éditez le programme), ou sur la pile (dans la partie affichée du programme).

L'objet principal de la *Hash-Table* est donc la traduction réciproque entre identificateur **XLIB** et noms usuels.

Pour accélérer les opérations de recherche que nécessite cette traduction, nous allons voir que la HP48 classe les noms (usuels) suivant le nombre de leurs caractères (de 1 à 16 caractères).

Celà permet une recherche séquentielle indexée (alors que la recherche des noms dans les répertoires **VAR** est purement séquentielle). Il peut en résulter un gain de temps appréciable (notamment dans les librairies volumineuses).

Voici maintenant, schématiquement, la structure de la *Hash-Table*, qui comme la *Link-Table*, est un entier binaire long.

Je rappelle que la première colonne sert uniquement à fixer des repères en termes d'adresse en mémoire, et qu'elle n'a donc rien à voir avec le contenu de la *Hash-Table* (qui est représenté en deuxième colonne).

Adresses	Contenu	Commentaires
a	E4A20	#02A4E = adresse-préfixe des binaires
b = a+5	Taille_H	Sur 5 quartets: b + Taille_H = g
c_1	Offset_1	Les offsets sont codés sur 5 quartets.
c_2	Offset_2	L'adresse obtenue en ajoutant Offset_k
.....	.....	à l'adr. courante c_k est l'adresse
c_15	Offset_15	e_k de la zone des noms ayant k car.
c_16	Offset_16	Offset_k = 00000 si pas de tels noms.
d=a+18*5	Taille_N	Taille zone des noms: d+Taille_L = f_0
e_1=d+5	Zone_1	Zone des noms ayant 1 caractère
e_2	Zone_2	Zone des noms ayant 2 caractères
.....	.....	.....
e_15	Zone_15	Zone des noms ayant 15 caractères
e_16	Zone_16	Zone des noms ayant 16 caractères
f_0	Offs_0	Tous ces offsets sont sur 5 quartets.
f_1	Offs_1	L'adresse f_j - Offs_j est celle où on
.....	.....	trouve dans la zone [e_1,e_16[, le
f_K	Offs_K	champ occupé par le nom N°j
g	.....	Premier quartet après la Hash-Table

◆ Les "Zones de noms" dans la Hash-Table:

Dans le tableau précédent, je nomme ***Zone_1***, ***Zone_2***, etc... des zones où sont regroupés, selon leur longueur, les noms de la librairie.

Voilà comment les choses se présentent à l'intérieur de la ***Zone_k*** des noms ayant k caractères.

Chaque nom occupe un champ organisé comme suit:

k	Name	Number

Le nombre k est écrit sur deux quartets.
"*Name*" est la succession des caractères qui forment le nom (2 quartets par caractère).
"*Number*" est le N° de l'objet ayant le nom "*name*" dans la librairie.

Les champs correspondants aux différents noms ayant k caractères sont consécutifs, et classés dans l'ordre alphabétique croissant du nom.

Remarque: Si aucun nom de la librairie n'a exactement k caractères, alors la "*Zone_k*" est vide. On peut ainsi passer directement de "*Zone_2*" à "*Zone_5*" s'il n'y a aucun nom ayant 3 ou 4 caractères. De toute façon c'est le premier octet (noté k dans le champ décrit-ci dessus, qui indique quand on passe d'une zone de noms à une autre).

◆ Comment fonctionne la Hash-Table:

Comme je l'ai dit, le rôle de la *Hash-Table* est d'effectuer une traduction, dans les deux sens, entre identificateurs **XLIB** et identificateurs usuels (lisibles).

- Si on connaît l'identificateur **XLIB**, et que l'on cherche le nom "*lisible*" (c'est le cas quand on passe en ligne de commande un programme contenant des objets de la librairie): dans la *Hash-Table*, on accède à l'adresse "**d**" puis à l'adresse "**f_0**" (notations du tableau ci-dessus).

- Connaissant donc le numéro **k** de l'objet dans la librairie, on accède à l'adresse "**f_k**" convenable, où on trouve un offset nous ramenant en arrière à l'adresse où débute le champ occupé par le nom recherché, qu'il suffit donc de lire....

- Si on connait le nom "*lisible*" et que l'on cherche à le traduire en un identificateur **XLIB** de la librairie (c'est le cas quand la HP48 traite un programme en ligne de commande, et que cet objet contient un nom: il s'agit de savoir si ce nom correspond à un objet de la librairie).

- Le nom "*lisible*" étant connu, on connaît le nombre **n** de ses caractères.
  On accède alors, en reprenant les notations du tableau ci-dessus, à l'adresse "**e_n**" de la zone des noms ayant **n** caractères.

- On cherche alors dans cette zone (sans doute par dichotomie puisque les noms sont classés par ordre alphabétique) le nom en question.

  Si ce nom n'est pas trouvé, il ne correspond pas à un objet de la librairie (et on va par exemple explorer une autre librairie).

  Si ce nom est trouvé, il est suivi de son numéro d'objet dans la librairie, et le tour est joué....

◆ Et les noms cachés, dans tout ça ?:

On sait que certains noms peuvent être cachés dans une librairie. Ils y ont cependant un numéro **XLIB**, et ils servent en général de sous-programmes aux programmes principaux de la librairie.

Néanmoins, ces objets cachés ne peuvent être utilisés à l'extérieur de la librairie, et leur nom n'apparaît pas dans le menu de celle-ci.

La solution est la suivante, en reprenant les notations du tableau précédent.

Si les numéros d'objet de la librairie vont de **0** à **N**, la zone où se trouvent les offsets "***Offs_0***", "***Offs_1***", ... "***Offs_K***" ne permet d'accéder qu'aux objets de numéros **0** à **K**; et ce sont ces derniers, et eux seulement, qui remplissent les zones "***Zone_1***", "***Zone_2***", ...,"***Zone_16***".

Moralité:

Si K<N, les objets portant les numéros K+1 à N sont cachés (aucune traduction n'est possible entre leur nom **XLIB** et leur nom lisible qui n'apparaît d'ailleurs nulle part).

Bien que cachés, ces objets sont néanmoins utilisables à l'intérieur de la librairie, car leur adresse peut-être obtenue dans la *Link-Table*.

J'espère que ces explications ne vous ont pas trop rebutés. Prenez le temps de bien assimiler ces notions.

Elles n'ont peut-être pas un intérêt pratique extraordinaire, mais leur intérêt théorique est fondamental pour qui veut comprendre les dessous du système d'exploitation de la HP48.

Comparez bien les explications qui précèdent avec l'exemple qui suit, et qui montre une librairie "*grandeur nature*".

◆ UN EXEMPLE DE LIBRAIRIE:

Il m'a semblé indispensable de vous montrer une vraie librairie (elle a été créée au moyen du programme **USRLIB.EXE**, sous copyright de **HP**: je ne sais pas si ce programme est dans le domaine public)

La librairie choisie ici porte le numéro 1515 (#5EBh). Elle est auto-attachable (son objet de configuration est « **1515 ATTACH** »)

Elle s'inspire du répertoire utilisé comme exemple dans un chapitre précédent consacré au codage des répertoires (s'y reporter pour les explications sur les programmes **PGCD**, **SIMPF**, et **ΣF**)

La librairie est constituée de 5 objets:

- Le programme **PGCD** (écrit comme une fonction-utilisateur)
  « → a b « b 'PGCD(b,a MOD b)' a IFTE » »

- Le programme **SIMPF**:
  ```
  « IFERR DUP2 PGCD ROT OVER / 3 ROLLD /
    THEN #5EB01h DOERR END »
  ```

- Le programme **ΣF**:
  ```
  « INVITE
    IFERR 4 ROLL OVER * ROT 4 PICK
      * + 3 ROLLD * SIMPF
    THEN #5EB02h DOERR END
  »
  ```

- Le programme **INVITE** (appelé par **ΣF**)
  « "Entrez 4 entiers" "" INPUT OBJ→ »

- L'objet de configuration:
  « 1515 ATTACH »

Les deux derniers objets sont des *objets cachés*. Il n'apparaîtront donc pas dans le menu de la librairie (c'est au moment où on va créer la librairie avec le programme **USRLIB.EXE** qu'il faut préciser quels objets doivent être cachés)

La librairie est munie de deux messages d'erreurs:

Le message N°1: "*Erreur dans SIMPF*"

Le message N°2: "*Erreur dans ΣF*"

Quand la librairie est attachée, le message N°1 est obtenu par l'instruction: **#5EB01h DOERR** (#5EB01h = 1515*256 + 1) et le message N°2 est obtenu par l'instruction: **#5EB02h DOERR**.

On voit d'ailleurs que les programmes **SIMPF** et **ΣF** ci-dessus éxécutent l'erreur qui leur correspond si un problème survient pendant leur éxécution.

J'ai utilisé ci-dessous un système d'adresses relatives (la librairie débutant symboliquement à l'adresse #000h), pour vous permettre de bien comprendre la signification des différents offsets (décalages) qui entrent dans la composition d'une librairie.

Adresse Relative	Contenu librairie	Commentaires (précédés du signe @)
#000h	04B20	@ #02B40h = adresse-préfixe des librairies
#005h	5C300	@ Taille: #005h+#3C5h=#3CAh pointe après Checksum
#00Ah	21 5437371696024656 02C69626271696279656 21	
		@ Titre: "Essai de librairie": #12h=18 caractères
#032h	BE5	@ Le numéro de la librairie est #5EBh=1515
#035h	EF200	@ #35h+#2FEh = #333h, pointe sur Hash_Table
#03Ah	57200	@ #3Ah+#275h = #2AFh, pointe sur Message_Table
#03Fh	1D200	@ #3Fh+#2D1h = #310h, pointe sur Link_Table
#044h	79000	@ #44h+#097h = #0DBh, pointe sur Config_Obj
#049h	8	@ L'objet, suivant, SIMPF, est une commande
#04Ah	BE5 200	@ SIMPF est l'objet 2 de la librairie 1515=#5EBh
#050h	D9D20	@ Début du rpl SIMPF
#055h	E1632 FD332	@ « IFERR
#05Fh	D9D20	@ Début Rpl-risque
#064h	2ABF1 29E20 BE5 000 E0CF1	@ DUP2 XLIB 1515 000 ROT
#079h	92CF1 50FA1 3F2A2 0DCF1 50FA1	@ OVER / 3 ROLLD /
#092h	B2130	@ Fin Rpl-risque
#097h	F1732	@ THEN de iferr
#09Ch	D9D20	@ Début Rpl-traitement
#0A1h	E4A20 51000 10BE500000000000 933A1	@ #5EB01h DOERR
#0C0h	B2130	@ Fin Rpl-traitement
#0C5h	5DF22 93632	@ END de iferr, et »
#0CFh	B2130	@ Fin du rpl SIMPF
#0D4h	8	@ L'objet-suivant, Config_Object, est une commande
#0D5h	BE5 300	@ Conf_Object est l'objet 3 de la libr. 1515=#5EBh
#0DBh	D9D20	@ Début du rpl de configuration
#0E0h	E1632 33920 3000000000051510 84412 93632	@ « 1515 ATTACH »
#104h	B2130	@ Fin du rpl de configuration
#109h	8	@ L'objet-suivant, ΣF, est une commande
#10Ah	BE5 100	@ ΣF est l'objet 1 de la librairie 1515=#5EBh
#110h	D9D20	@ Début du rpl ΣF
#115h	E1632 29E20 BE5 400 FD332	@ « XLIB 1515 004 IFERR
#12Ah	D9D20	@ Début du Rpl-risque
#12Fh	803A2 5BCF1 92CF1 EEDA1 E0CF1	@ 4 ROLL OVER * ROT
#148h	803A2 A9CF1 EEDA1 76BA1 3F2A2	@ 4 PICK * + 3
#161h	0DCF1 EEDA1 29E20 BE5 200	@ ROLLD * XLIB 1515 002
#176h	B2130	@ Fin du Rpl-risque
#17Bh	F1732	@ THEN de iferr
#180h	D9D20	@ Début Rpl-traitement
#185h	E4A20 51000 20BE500000000000 933A1	@ #5EB02h DOERR
#1A4h	B2130	@ Fin Rpl-traitement
#1A9h	5DF22 93632	@ END de iferr, et »
#1B3h	B2130	@ Fin du rpl ΣF

(Suite du tableau précédent)

Adresse Relative	Contenu librairie	Commentaires (précédés du signe @)
#1B8h	8	@ L'objet-suivant, INVITE, est une commande
#1B9h	BE5 400	@ INVITE est l'objet 4 de la libr. 1515=#5EBh
#1BFh	D9D20	@ Début du rpl INVITE
#1C4h	E1632 C2A20 52000	@ « puis chaîne
#1D3h	54E6472756A702430256E64796562737	@ "Entrez 4 entiers"
#1F3h	C2A20 50000 AC422 B7FC1 93632	@ "" INPUT OBJ-> »
#20Ch	B2130	@ Fin du rpl INVITE
#211h	000	@ L'objet suivant, PGCD, est une fonction
#214h	BE5 000	@ PGCD est l'objet 0 de la librairie 1515=#5EBh
#21Ah	D9D20	@ Début du rpl PGCD)
#21Fh	E1632 1C432 D6E20 10 16 D6E20 10 26	@ « -> a b
#23Bh	E1632 D6E20 10 26	@ « b
#249h	8BA20	@ Début expression
#24Eh	D6E20 10 26 D6E20 10 16 D6E20 10 26 D4EB1	@ b a b MOD
#26Eh	8BA20 29E20 BE5 000 B2130	@ Expr. réduite à XLIB 1515 0
#283h	30040 046F1	@ <2h> APPLY
#28Dh	B2130	@ Fin Expression
#292h	D6E20 10 16 EF3A1 EF532 93632	@ a IFTE » »
#2AAh	B2130	@ Début du rpl PGCD)
#2AFh	@ Table des messages	
	8E920	@ C'est un tableau
#2B4h	C5000	@ Taille totale: #2B4h+#5Ch=#310h)
#2B9h	C2A20 10000 20000	@ Vecteur de 2 chaînes
#2C8h	72000 5425255455250244 14E435023594D40564	@"Erreur dans SIMPF"
#2EFh	12000 5425255455250244 14E435025864	@"Erreur dans ΣF"
#310h	@ Link-Table	
	E4A20	@ C'est un entier binaire
#315h	E1000	@ Taille #1Eh: #315h+#1Eh=#333h
#31Ah	00FFF	@ #31Ah+#FFF00h=#21Ah, pointe sur objet 0)
#31Fh	1FDFF	@ #31Fh+#FFDF1h=#110h, pointe sur objet 1)
#324h	C2DFF	@ #324h+#FFD2Ch=#050h, pointe sur objet 2)
#329h	2BDFF	@ #329h+#FFDB2h=#0DBh, pointe sur objet 3)
#32Eh	19EFF	@ #32Eh+#FFE91h=#1BFh, pointe sur objet 4)
#333h	@ Hash-Table	
	E4A20	@ C'est un entier binaire
#338h	E8000	@ Taille #8Eh: #338h+#8Eh=#3C6h
#33Dh	00000	@ Pas de nom ayant 1 caractère
#342h	05000	@ #342h+#50h=#392h, pointe sur liste noms de 2 car
#347h	00000	@ Pas de nom ayant 3 caractères
#34Ch	F4000	@ #34Ch+#4Fh=#39Bh, pointe sur liste noms de 4 car
#351h	75000	@ #351h+#57h=#3A8h, pointe sur liste noms de 5 car
#356h	00000 00000 00000 00000	@ Pas de nom ayant 6 à 9 caract.
#36Ah	00000 00000 00000 00000	@ Pas de nom ayant 10 à 13 caract.
#37Eh	00000 00000 00000	@ Pas de nom ayant 14 à 16 caract.
#38Dh	A2000	@ Taille liste des noms = #2Ah : #38Dh+#2Ah=#3B7h
#392h	20 5864 100	@ ΣF: 2 caractères. Objet N°1
#39Bh	40 05743444 000	@ PGCD: 4 caractères. Objet N°0
#3A8h	50 3594D40564 200	@ SIMPF: 5 caractères. Objet N°2
#3B7h	C1000	@ back_offset vers PGCD (obj N°0): #3B7h-#1Ch=#39Bh
#3BCh	A2000	@ back_offset vers ΣF (obj N°1): #3BCh-#2Ah=#392h
#3C1h	91000	@ back_offset vers SIMPF (obj N°2): #3C1h-#19h=#3A8h
#3C6h	0E8F	@ Checksum interne : #F8E0h
#3CAh	@ Premier quartet après la librairie	

"LIBRARY DATA"

Les objets de type "***Library Data***" sont créés et utilisés par les librairies, pour stocker des données nécessaires à leur fonctionnement. Ces objets sont en général créés en RAM, et peuvent être purgés à la sortie de l'application qui les a utilisés.

Un objet de type "***Library Data***" est (c'est ce qui ressort de sa structure) attaché à une librairie particulière, et pour cette librairie, il est caractérisé par un numéro particulier.

Un objet de type "***Library Data***" est une collection de sous-objets de type quelconque (au sens habituel, adresses-préfixes comprises).

Les "***Library Data***" ne sont pas affichables "*en clair*" à l'écran. Leur présence sur la pile est signalée par les mots génériques **Library Data**. L'instruction **TYPE** renvoie alors le résultat 26.

La structure d'un tel objet est la suivante:

88B20	L'adresse-préfixe est #02B88h
Taille	La taille totale, hors préfixe, sur 5 quartets
N°Lib	Numéro de librairie, sur trois quartets.
N°Data	Numéro de "Library Data", sur deux quartets.
Obj_1	Premier objet du "Library Data"
Obj_2	Deuxième objet du "Library Data"
.....	.....
Obj_N	Dernier objet du "Library Data"
B2130	Adresse postfixe #0312Bh, de fin de structure.

On remarque que les "***Library Data***" sont des objets composés (au même titre que les listes, les rpl, les expressions, les objets-unités), avec toutefois une restriction: Les deux premiers champs sur 5 quartets ne sont pas des sous-objets (au sens des éléments d'une liste par exemple) puisqu'ils désignent respectivement la longueur, puis l'identificateur de l'objet "***Libary Data***".

Il est étonnant qu'un tel objet comporte à la fois un marqueur de fin de structure (l'adresse-postfixe **#0312Bh**), et un champ indiquant la longueur totale de l'objet (hors préfixe).

◆ Un exemple de "Library Data":

Si vous possédez la carte "**EQ LIBRARY**", entrez dans le programme de jeu **MINEHUNT**, et actionnez la touche **STO**. Dans le menu **VAR** apparait une entrée '**MHpar**', dont le contenu est un objet de type "***Library Data***".

Effectuez **#54AFh SYSEVAL** sur cet objet pour le décomposer en ses différents éléments (cf le chapitre sur les listes et les objets-composés) et vous verrez qu'il est composé d'une liste, laquelle contient un objet-graphique, 3 entiers binaires égaux à <1h>, et une chaîne de caractères (les 2 Externals qui apparaissent sont la traduction des 2 premiers champs, indiquant la longueur et l'indentificateur de l'objet).

OBJETS-SAUVEGARDES

Les ***objets-sauvegardes***, ou "*Backup-Objects*" permettent de stocker des objets quelconques (et par exemple des répertoires) dans les ports 1 et 2 (pour la HP48SX uniquement, sur des cartes RAM d'extension mémoire) ou dans le port 0 (qui est une zone de la mémoire centrale prévue à cet effet, dont la taille s'adapte au nombre d'***objets-sauvegardes*** qui y sont stockés)

La façon dont on crée les ***objets-sauvegardes***, dont on les rappelle, dont on les évalue, est décrite dans les pages 670 et suivantes du manuel de l'utilisateur de la HP48.

Rappelons seulement qu'un ***objet-sauvegarde*** résulte du stockage d'un objet, sous un certain nom. L'objet stocké est le plus souvent le contenu d'une variable, ou celui d'un répertoire. Le nom de cette variable ou de ce répertoire n'intervient pas ici. Il n'a aucune raison d'être identique au nom sous lequel l'***objet-sauvegarde*** est stocké.

La structure des ***objets-sauvegardes*** est décrite schématiquement dans le tableau ci-dessous. On constate que l'***objet-sauvegarde*** se termine par un entier-système contenant une somme de contrôle interne.

26B20	L'adresse-préfixe est #02B62h
Taille	La taille totale, hors préfixe, sur 5 quartets
n	Nombre de caractères du nom (deux quartets)
Nom	Le nom de l'objet-sauvegarde (2*n quartets)
n	A nouveau le nombre de caractères du nom
Obj	L'objet sauvegardé (quelconque)
11920	Adresse préfixe #02911h des entiers-systèmes
CRC	Sur 5 quartets, somme de contrôle de l'objet

Exemple: Si on stocke le programme « **ROT * + »** sous la forme d'un objet-sauvegarde nommé **PRG**, l'objet obtenu s'écrit:

Contenu	Commentaires
26B20	#02B62h = adresse préfixe des objets backups
C3000	#3Ch = 60 = taille de l'objet, hors préfixe
30	Le nom de l'objet-sauvegarde a trois caractères
052574	Ce nom est PRG (05 => #50h, code de P)
30	Le nom de l'objet-sauvegarde a trois caractères
D9D20	Début de l'objet sauvegardé (c'est un Rpl)
E1632	«
E0CF1	ROT
EEDA1	*
76BA1	+
93632	»
B2130	Adresse suffixe des Rpl
11920	Adresse préfixe des entiers-système
05D41	#14D50 = Somme de contrôle interne

"LINKED ARRAYS"

Voici sans doute le type d'objet la plus mystérieux de la HP48. Il n'en existe aucun dans la ROM de la HP48. Ce chapitre devrait lever les mystères qui entourent cette structure.

On connait déjà le type "*Array*", et on a vu dans le chapitre qui lui est consacré que le type "*Array*", apparemment très simple, pouvait se prêter à des variations intéressantes (on peut par exemple considérer des tableaux ayant plus de deux dimensions, ou encore constitués de chaînes de caractères). Il en est de même avec les objets de type "***Linked-Array***".

Les objets de type "***Linked-Array***" sont des tableaux qui ont de nombreux points communs avec les objets de type "*Array*" (c'est-à-dire les tableaux au sens usuel):

- Les éléments qui le composent doivent être du même type.
- Le nombre de dimensions peut être quelconque.
- La longueur du tableau dans une direction donnée reste la même.

Il y a bien sûr une différence de taille:

Tous les éléments d'un tableau de type "*Array*" doivent être présents dans la structure (même s'ils sont tous égaux...)

Au contraire, certains éléments d'un "***Linked-Array***" peuvent être absents, et des éléments du "***Linked-Array***" dont on sait qu'il sont égaux n'ont pas besoin d'apparaître en plusieurs exemplaires dans la structure.

Comment celà est-il possible ? : tout simplement en accédant à tel ou tel élément du tableau via une table de ***pointeurs*** (d'*offsets*). Il suffit par exemple que plusieurs pointeurs désignent un même objet pour que cet objet (en un seul exemplaire pourtant) soit considéré comme figurant dans le "***Linked Array***" en différentes positions.

Voici donc la structure, schématiquement, d'un "***Linked Array***".

A0A20	#02A0Ah est l'adresse-préfixe de tout "Linked-Array"
Taille	Taille totale hors-préfixe, sur 5 quartets
Type	Adresse-préfixe => type des éléments (5 quartets)
Dims	Nombre N de dimensions du tableau, sur 5 quartets
Dim_1	Taille de la première dimension, sur 5 quartets
Dim_2	Taille de la seconde dimension, sur 5 quartets
.....	.....
Dim_N	Taille de la N-ème et dernière dimension, 5 quartets
Off_1	Offset sur le premier élément, sur 5 quartets.
Off_2	Offset sur le second élément, sur 5 quartets.
.....	.....
Off_M	Offset sur le M-ème et dernier élément du tableau.
.....	Zone des objets du tableau

Dans la zone des "*offsets*" (*décalages*), un offset à zéro (5 quartets nuls) signale l'absence de l'élément correspondant.

Dans cette zone, l'ordre implicite des éléments est l'ordre *lexicographique* habituel.

Par exemple, dans le cas d'un tableau à deux dimensions, le premier *offset* permet de trouver l'élément de position (1,1) , le second *offset* permet de trouver l'élément de position (1,2), etc....

Dans la zone des objets du tableau, l'ordre dans lequel sont disposés les objets n'a aucune importance (plusieurs offsets peuvent pointer indirectement sur le même objet). A la limite, il peut très bien n'y avoir qu'un seul objet (tous les offsets pointant dessus ou étant nuls), ou aucun objet (tous les offsets étant nuls).

◆ Un exemple de "Linked-Array":

Considérons le tableau (au sens usuel) de type { 2 3 }, dont les élements sont les nombres complexes suivants:

z = (1,3) pour les positions {1 1}, {2 2}, et

z = (2,-1) pour les autres positions du tableau.

Au format usuel, ce tableau s'écrit comme ci-dessous et sa taille donnée par BYTES est de 111 octets:

```
[[ (1,3)   (2,-1)   (2,-1) ]
 [ (2,-1)   (1,3)   (2,-1) ]]
```

Le même tableau, au format "***Linked-Array***", s'écrirait de la manière suivante. La première colonne indique l'adresse relative par rapport au début de l'objet (elle n'a aucun rapport avec le contenu du tableau, elle sert seulement à rendre plus compréhensible l'utilisation des offsets).

Adresse Relative	Contenu	Commentaires
#00h	A0A20	#02A0Ah = adresse-préfixe
#05h	77000	#77h = taille totale: #05h+#77h = #7Ch
#0Ah	77920	#02977h : les objets sont des complexes
#0Fh	20000	C'est un tableau a deux dimensions
#14h	20000	La première dimension vaut 2 (2 lignes)
#19h	30000	La 2-ième dimension vaut 3 (3 colonnes)
#1Eh	E3000	Pointe sur (1,3) : #1Eh + #3Eh = #5Ch
#23h	91000	Pointe sur (2,-1) : #23h + #19h = #3Ch
#28h	41000	Pointe sur (2,-1) : #28h + #14h = #3Ch
#2Dh	F0000	Pointe sur (2,-1) : #2Dh + #0Fh = #3Ch
#32h	A2000	Pointe sur (1,3) : #32h + #2Ah = #5Ch
#37h	50000	Pointe sur (2,-1) : #37h + #05h = #3Ch
#3Ch	000 0000000000020 000 0000000000019	(2,-1)
#5Ch	000 0000000000010 000 0000000000030	(1,3)
#7Ch	.....	Premier quartet après le Linked-Array

La taille de cet objet est de #7Ch quartets (62 octets) alors que le même tableau, au format "*Array*" traditionnel, occuperait 111 octets.

Il n'est déjà pas évident d'imaginer comment on pourrait construire un "***Linked Array***".

Mais même si l'on dispose d'un tel objet, les instructions **GET** et **PUT** (même leurs formes les plus internes) ne permettent ni d'y lire ni d'y écrire.

On se consolera en lisant les mots "**Linked Array**" par lesquels la HP48 signale la présence d'un tel objet sur la pile. Ce n'est déjà pas si mal.

FIN DU PREMIER ACTE....

DEUXIEME PARTIE

LES SYSEVALs

De toutes les écoles de la patience et de la lucidité, la création est la plus efficace. Elle est aussi le bouleversant témoignage de la seule dignité de l'homme: la révolte tenace contre sa condition, la persévérance dans un effort tenu pour stérile.[...]. Tout celà "pour rien", pour répéter et piétiner. Mais peut-être la grande oeuvre d'art a moins d'importance en elle-même que dans l'épreuve qu'elle exige d'un homme et l'occasion qu'elle lui fournit de surmonter ses fantômes et d'approcher d'un peu plus près sa réalité nue.

Albert CAMUS,
LE MYTHE DE SISYPHE

◆ SYSEVALs "à gogo":

Après avoir étudié les différents types d'objets, il est grand temps de découvrir comment ces objets sont utilisés dans les entrailles de la HP48.

Autant la première partie était consacrée au *présentations* et aux *descriptions*, autant cette deuxième partie est tournée vers l'***action***.

Toutes les adresses qui seront décrites ici pointent sur des "***structures Rpl***", ou des objets de type "***Primitive Code***". Dans tous les cas, il s'agira de SYSEVALs "*qui font quelque chose*".

Pour décrire un nombre d'adresses aussi grand, il faut un minimum d'organisation. Les SYSEVALs seront ici répartis en 38 chapitres successifs.

On trouvera ainsi les SYSEVALs s'appliquant à un type d'objet particulier (les entiers-système par exemple), ou bien encore agissant dans le cadre d'un environnement précis de la calculatrice (les alarmes par exemple).

Dans certains cas, il n'a pas été facile de choisir dans quel chapitre placer telle ou telle adresse (notamment si elle utilise des arguments de types différents).

Dans ce cas, la règle générale (en principe) est que le SYSEVAL est décrit dans le chapitre consacré à celui de ses arguments qui est le plus "*pointu*", le plus "*complexe*" (dans la pratique, les 38 chapitres de cette partie suivent plutôt l'ordre de complexité croissante).

Par exemple, un SYSEVAL qui nécessite un réel et un objet-graphique sera décrit dans le chapitre consacré aux objets-graphiques.

Très exceptionnellement, un SYSEVAL pourra être décrit dans deux chapitres distincts (mais le nombre de cas est très faible)

◆ SYSEVALs et Mnémoniques:

Il a bien fallu trouver un moyen rationnel de décrire individuellement l'action de toutes ces procédures, en évitant les longueurs et les répétitions.

Chaque SYSEVAL est tout d'abord associé à un ***mnémonique*** approprié. Celui-ci apparaît comme un code "*lisible*" renseignant sur la nature du SYSEVAL. Ce ***mnémonique*** contient une description abrégée des arguments éventuellement nécéssaires.

Dans chaque cas où le mnémonique ne suffit pas à caractériser parfaitement le SYSEVAL, des commentaires appropriés (ou un exemple d'utilisation) viendront décrire son fonctionnement.

Il sera beaucoup question de "*formes internes*" d'une instruction standard de la HP48. Chacune de ces instructions s'accompagne en effet de tests permettant d'éviter toute utilisation non prévue. Les *formes internes* de l'instruction sont les adresses des routines éxécutant cette instruction, mais en contournant ces tests préalables (qui sont souvent pénalisants en termes de vitesse d'éxécution).

◆ Notations

Voici quelques abréviations qui seront utilisées dans cette deuxième partie pour désigner les arguments nécessaires à l'éxécution d'un SYSEVAL. La plupart de ces abréviations se comprennent mieux à partir de la dénomination en langue anglaise (qui est rappelée):

Nombre réel:	r	real number
Nombre complexe:	c	complex number
Nombre (réel ou cplx):	nb	number
Chaîne de caractères:	st	string
Tableau réel:	ra	real array
Tableau complexe:	ca	complex array
Tableau (réel ou cplx):	ar	array
Liste:	li	list
Liste ou réel:	lr	
Tableau ou liste:	al	
Nom global:	gn	global name
Nom local :	ln	local name
Nom (global ou local):	nm	name
Programme:	pr	
Expression:	ex	
Entier binaire:	b	binary
Objet graphique:	gr	graphic object
Objet taggué:	to	tagged object
Objet unité:	uo	unit object
Nom Xlib:	xn	XLIB name
Répertoire:	di	directory
Librairie:	lb	
Objet backup:	bk	backup object
Fonction intégrée:	bf	built-in function
Commande intégrée:	bc	built-in command
Entier-système:	s	system binary
Réel long:	rr	
Complexe long:	cc	
Caractère:	ch	character
Code machine:	cd	code object
Booléen:	?	

D'autre part, le caractère _ (souligné) représentera les arguments dont le type est quelconque (il est parfois utile de signaler la présence d'un tel argument pour comprendre le fonctionnement du SYSEVAL).

Dans les mnémoniques, et quand il y a plusieurs arguments, ils sont ordonnés suivant leur position sur la pile (comme le seraient les arguments d'une fonction utilisateur).

Exemples: voici quelques exemples représentatifs de la façon dont sont décrits les SYSEVALs dans cette deuxième partie du livre.

Les adresses ci-dessous sont extraites du chapitre consacré aux chaînes de caractères (mais elles y figurent en différents endroits).

Adresse	Mnémonique	Type	Commentaires
#1ACA7h	+(st,_)	Rpl	obj1 d'abord transformé en chaîne
#1ACBBh	+(_,st)	Rpl	obj2 d'abord transformé en chaîne
#05733h	sub(st,2s)	LM	ex: "ABCDEFGH",<2h>,<4h> => "BCD"
#1C8BBh	sub(st,2r)	Rpl	ex: "ABCDEFGH", 2, 4 => "BCD"
#1CAD7h	pos(2st)	Rpl	ex: "ABCDECDF", "CD" => 3
#15EF6h	spos(2st)	Rpl	ex: "ABCDECDF", "CD" => <3h>
#645B1h	spos(2st,s)	LM	recherche débute au car. n°s de st3 ex: "ABCDECDF", "CD" , <2h> => <3h> ex: "ABCDECDF", "CD" , <4h> => <6h> ex: "ABCDECDF", "CD" , <7h> => <0h>
#645BDh	rspos(2t,s)	LM	"rspos" = "reverse system pos" recherche en amont, depuis car. n°s ex: "ABCDECDF", "CD" , <2h> => <0h> ex: "ABCDECDF", "CD" , <4h> => <3h> ex: "ABCDECDF", "CD" , <7h> => <6h>
#18873h	and(2st)	Rpl	pour ces 3 adresses, il y a erreur
#18887h	or(2st)	Rpl	"Bad Argument Value" si les chaînes
#1889Bh	xor(2st)	Rpl	ont des longueurs différentes
#188E6h	and(2st)!	LM	appelé par adresses ci-dessus.
#188F5h	or(2st)!	LM	A n'utiliser que si les 2 chaînes
#18904h	xor(2st)!	LM	ont la même longueur!
#188D2h	not(st)	Rpl	avec newob
#18961h	not(st)!	LM	sans newob

Le "***Type***" (Rpl ou LM) correspond à la nature de l'objet désigné par le SYSEVAL: ce peut être une structure Rpl ou une séquence en langage machine (LM).

Dans le cas d'un structure Rpl, on peut envisager une analyse de cette structure (des adresses qu'elle contient) pour aboutir à des SYSEVALs encore plus "*internes*" (c'est ainsi que j'ai procédé, à partir des adresses des instructions standards).

Dans le cas d'une séquence en langage machine, il ne vous reste plus, si vous voulez en savoir davantage, qu'à désassembler le code ainsi désigné (mais du point de vue des SYSEVALs, c'est la certitude qu'on ne peut aller plus loin).

Dans le tableau, ci-dessus, on voit comment sont décrits les arguments, notamment quand deux arguments de même type (et consécutifs sur la pile) sont nécessaires.

C'est ainsi que le *mnémonique* **sub(st,2s)** désigne l'opération qui consiste à extraire une sous-chaîne d'une chaîne "*st*" (placée au niveau 3), entre les caractères ayant les positions désignées par les deux entiers-système placés aux niveaux 2 et 1.

On voit sur les exemples ci-dessus qu'il faut parfois avoir recours à des artifices pour désigner les différentes versions d'une même instruction.

Prenons le cas de l'instruction standard **AND**.

A l'adresse **#18873h**, on trouve la forme interne de cette instruction, quand les deux arguments sont des chaînes de caractères (le mnémonique est **and(2st)**).

Cette "*forme interne*" effectue encore un test (débouchant éventuellement sur une erreur "*Bad Argument Value*") pour vérifier que les deux chaînes ont la même longueur.

Si on veut éviter ce test (par exemple si on sait que les deux chaînes ont vraiment la même longueur), on peut alors se tourner vers l'adresse **#188E6h** (mnémonique **and(2st)!**) pour trouver la forme interne ultime (écrite en langage-machine) de cette instruction **AND** pour deux chaînes de caractères.

Plus généralement, le point d'exclamation (utilisé ici dans le mnémonique **and(2st)!**) départagera des SYSEVALs à priori analogues, en désignant celui des deux qui est le plus *"interne"* (en général écrit en langage-machine) mais aussi, souvent, le plus *"risqué"*.

Il arrive souvent qu'un SYSEVAL puisse être décomposé en une succession de deux ou trois opérations élémentaires. Dans ce cas le mnémonique associé à ce SYSEVAL pourra être formé par les mnémoniques de ces composantes, séparés par le caractère "*point-virgule*" **;** .

C'est ainsi par exemple que le SYSEVAL de l'adresse **#62775h** et qui éxécute l'opération ***rot***, puis l'opération ***dup*** (les formes internes des instructions **ROT** et **DUP**) sera désigné par le mnémonique **rot;dup** .

Les mnémoniques utilisent en principe des caractères minuscules (pour les distinguer des instructions standards de la HP48: il faut par exemple différencier l'instruction **DUP** de sa forme interne **dup**). Quand le mnémonique est assez long et formé de plusieurs mots abrégés, des majuscules sont utilisées pour faire apparaître ces composantes.

Il en est ainsi, par exemple, pour l'adresse **#353ABh**, désignée par le mnémonique ***ListOfNames(ex)***, et dont le rôle est de produire la liste des noms figurant dans l'expression placée au niveau 1.

Bien d'autres "*ficelles*" seront utilisées pour décrire au mieux les centaines de SYSEVALs qui font l'essentiel de ce livre. Elles apparaîtront au fur et à mesure des besoins.

J'ajoute que l'utilité des mnémoniques est ici purement descriptive. Mais elle prendra une toute autre dimension dans le tome 2...

OPÉRATIONS SUR LA PILE

Dans ce chapitre, nous allons voir quelques adresses essentielles dans le monde des SYSEVALs: celles qui permettent d'agir sur la pile.

Nombre des opérations décrites ci-dessous sont des variantes des différentes commandes du menu **PRG\STK**, ou des combinaisons de ces commandes.

ATTENTION: souvent ces SYSEVALs ne vérifient pas l'état de la pile. C'est à vous de prendre vos responsabilités. Si vous essayez par exemple l'instruction ***drop*** (#03244h SYSEVAL) avec une pile vide, vous allez vers un "*Memory Clear*".

Adresse	Mnémonique	Type	Commentaires
#0314Ch	sdepth	LM	DEPTH sous forme d'entier-système
#03188h	dup	LM	
#031ACh	dup2	LM	
#03223h	swap	LM	
#03244h	drop	LM	No Comment
#03258h	drop2	LM	
#03295h	rot	LM	
#032C2h	over	LM	
#60FBBh	4roll	LM	Ces Sysevals permettent
#60FD8h	5roll	LM	d'éxécuter l'instruction
#61002h	6roll	LM	ROLL sur un niveau précis
#6106Bh	7roll	LM	de la pile (de 4 à 8)
#6103Ch	8roll	LM	
#60FACh	3rolld	LM	
#6109Eh	4rolld	LM	
#610C4h	5rolld	LM	
#610FAh	6rolld	LM	Idem avec ROLLD
#62BC4h	7rolld	Rpl	
#63119h	8rolld	Rpl	
#174F0h	9rolld	Rpl	
#6312Dh	10rolld	Rpl	
#611FEh	3pick	LM	
#6121Ch	4pick	LM	
#6123Ah	5pick	LM	
#6125Eh	6pick	LM	Idem avec PICK
#61282h	7pick	LM	
#612A9h	8pick	LM	
#60F4Bh	3dropn	LM	
#60F7Eh	4dropn	LM	
#60F72h	5dropn	LM	Idem avec DROPN
#60F66h	6dropn	LM	
#60F54h	7dropn	LM	

OPÉRATIONS SUR LA PILE

Adresse	Mnémonique	Type	Commentaires
#03325h	roll(s)	LM	
#612F3h	roll(s+1)	LM	
#61318h	roll(s+2)	LM	Ces Sysevals permettent d'appeler
#0339Eh	rolld(s)	LM	les commandes ROLL, ROLLD, PICK
#61353h	rolld(s+1)	LM	DUP, et DROPN, en utilisant
#61365h	rolld(s+2)	LM	l'entier-système s présent au
#032E2h	pick(s)	LM	niveau 1 de la pile.
#611A3h	pick(s+1)	LM	
#611BEh	pick(s+2)	LM	Dans certains cas, la valeur de
#611D2h	pick(s+3)	LM	s est augmentée au préalable,
#611E1h	pick(s+4)	LM	de 1,2,3, ou 4.
#031D9h	dupn(s)	LM	
#0326Eh	dropn(s)	LM	
#62F75h	dropn(s+1)	Rpl	
#32075h	dropn(s-4)	Rpl	4-(s);dropn
#62F89h	drop(s)	Rpl	Roll(s);drop
#4A842h	drop(2,4)	Rpl	
#4EB9Ah	drop(1-2,5)	Rpl	
#50154h	drop(1,5-6)	Rpl	Ces adresses permettent de
#60F21h	drop(3)	LM	supprimer un ou plusieurs
#60F9Bh	drop(2)	LM	objets de la pile.
#6112Ah	drop(2-3)	LM	drop(2,4) ôte les objets des
#6113Ch	drop(2-4)	LM	niveaux 2 et 4.
#62726h	drop(1,3)	LM	drop(2-4) ôte les objets des
#62864h	drop(4)	LM	niveaux 2 à 4.
#62880h	drop(5)	LM	
#64368h	drop(1,3)	Rpl	
#60EE7h	swap(2,3)	LM	Echange niveaux 2 et 3
#60F33h	swap(1,3)	LM	Echange niveaux 1 et 3
#62001h	swap2	LM	4:a,3:b,2:c,1:d => 4:c,3:d,2:a,1:b
#63C68h	pick(2-3)	Rpl	3pick;3pick
#63FBAh	pick(3-4)	Rpl	4pick;4pick

Adresse	Mnémonique	Type	Commentaires
#107D4h	dup;drop(s)	Rpl	┐ Ces adresses nécessitent
#35222h	dup;roll(s);swap	Rpl	│ un entier-système s au
#6119Eh	dup;pick(s+1)	LM	│ niveau 1 de la pile,
#61305h	dup;roll(s+2)	LM	├ puis effectuent une
#62D45h	roll(s);swap	Rpl	│ opération sur la pile,
#630DDh	dup;pick(s)	Rpl	│ fonction de la valeur
#630F1h	dup;roll(s)	Rpl	┘ de l'entier s.
#63105h	over;rolld(s+2)	Rpl	┐ Entier-système au niveau 2
#63FA6h	drop;dropn(s)	Rpl	┘
#61172h	-(2s);pick(s)	LM	
#612CCh	-(2s);roll(s)	LM	┐ Ces 5 adresses attendent
#612DEh	+(2s);roll(s)	LM	├ 2 entiers-système s2 et s1
#6132Ch	-(2s);rolld(s)	LM	│ et effectuent d'abord leur
#6133Eh	+(2s);rolld(s)	LM	┘ somme ou leur différence.

Les adresses du tableau ci-dessous donnent une idée du nombre d'utilitaires qu'on trouve un peu partout dans la ROM de la HP48. Il s'agit ici de SYSEVALs qui effectuent des opérations de réarrangement (plus ou moins sophistiquées) sur les niveaux inférieurs de la pile.

Ces opérations sont relativement fréquentes. C'est pourquoi les porgrammeurs de la HP48 en ont fait des sous-programmes, de manière à pouvoir les appeler à tout moment.

Adresse	Mnémonique	Type	Commentaires
#49523h	drop(1,3-4);dup	Rpl	4:a 3:b 2:c 1:d => 2:c 1:c
#2F29Dh	drop(4);rot	Rpl	4:a 3:b 2:c 1:d => 3:c 2:d 1:b
#35D08h	drop;3pick	Rpl	4:a 3:b 2:c 1:d => 4:a 3:b 2:c 1:a
#4E9F1h	drop(3-4);swap	Rpl	4:a 3:b 2:c 1:d => 2:d 1:c
#50A63h	swap(1,3);4roll	Rpl	4:a 3:b 2:c 1:d => 4:d 3:c 2:b 1:a
#60F0Eh	drop(3);swap	LM	3:a 2:b 1:c => 2:c 1:b
#61099h	dup;4rolld	LM	3:a 2:b 1:c => 4:c 3:a 2:b 1:c
#611F9h	dup;3pick	LM	2:a 1:b => 4:a 3:b 2:b 1:a
#61380h	swap;over	LM	2:a 1:b => 3:b 2:a 1:b
#6270Ch	drop;swap	LM	3:a 2:b 1:c => 2:b 1:a
#62747h	swap;dup	LM	2:a 1:b => 3:b 2:a 1:a
#62775h	rot;dup	LM	3:a 2:b 1:c => 4:b 3:c 2:a 1:a
#627A7h	drop;dup	LM	2:a 1:b => 2:a 1:a
#62830h	drop(2);dup	LM	2:a 1:b => 2:b 1:b
#6284Bh	3rolld;drop	LM	3:a 2:b 1:c => 2:c 1:a
#62C7Dh	rot;dup2	Rpl	3:a 2:b 1:c => 5:b 4:c 3:a 2:c 1:a
#62CA5h	rot;over	Rpl	3:a 2:b 1:c => 4:b 3:c 2:a 1:c
#62CB9h	dup;dup	Rpl	1:a => 3:a 2:a 1:a
#62CCDh	over;dup	Rpl	2:a 1:b => 4:a 3:b 2:a 1:a
#62CF5h	3rolld;dup	Rpl	3:a 2:b 1:c => 4:c 3:a 2:b 1:b
#62D09h	4rolld;dup	Rpl	4:a 3:b 2:c 1:d => 5:d 4:a 3:b 2:c 1:c
#62D31h	over;swap	Rpl	2:a 1:b => 3:a 2:a 1:b
#62ECBh	4roll;swap	Rpl	4:a 3:b 2:c 1:d => 4:b 3:c 2:a 1:d
#62EDFh	3pick;swap	Rpl	3:a 2:b 1:c => 4:a 3:b 2:a 1:c
#62EF3h	4pick;swap	Rpl	4:a 3:b 2:c 1:d => 5:a 4:b 3:c 2:a 1:d
#62FB1h	dup;rot	Rpl	2:a 1:b => 3:b 2:b 1:a
#62FC5h	drop;rot	Rpl	4:a 3:b 2:c 1:d => 3:b 2:c 1:a
#63001h	4roll;rot	Rpl	4:a 3:b 2:c 1:d => 4:b 3:d 2:a 1:c
#63015h	4rolld;rot	Rpl	4:a 3:b 2:c 1:d => 4:d 3:b 2:c 1:a
#63029h	drop;over	Rpl	3:a 2:b 1:c => 3:a 2:b 1:a
#6308Dh	3rolld;over	Rpl	3:a 2:b 1:c => 4:c 3:a 2:b 1:a
#630A1h	4roll;over	Rpl	4:a 3:b 2:c 1:d => 5:b 4:c 3:d 2:a 1:d
#630B5h	3pick;over	Rpl	3:a 2:b 1:c => 5:a 4:b 3:c 2:a 1:c
#630C9h	4pick;over	Rpl	a, b, c, d => a, b, c, d, a, d
#6386Ch	swap;dup2	Rpl	2:a 1:b => 4:b 3:a 2:b 1:a
#63C2Ch	swap;4roll	Rpl	4:a 3:b 2:c 1:d => 4:b 3:d 2:c 1:a
#63C40h	dup2;5roll	Rpl	3:a 2:b 1:c => 5:b 4:c 3:b 2:c 1:a
#63C54h	swap;3pick	Rpl	3:a 2:b 1:c => 4:a 3:c 2:b 1:a
#63C7Ch	swap;4pick	Rpl	4:a 3:b 2:c 1:d => 5:b 4:c 3:d 2:a 1:d
#63C90h	over;5pick	Rpl	a, b, c, d => a, b, c, d, c, a

Voici trois adresses plus généralistes, mais fort utiles:

- **#44197h pick(s)? (Rpl)**
  Le rôle de "***pick(s)?***" est d'aller chercher un objet à une hauteur donnée sur la pile sans risque d'erreur si la pile n'est pas suffisamment haute.
  Il tente d'effectuer pick(s). Le résultat est **1:false** si la pile trop courte ou **2:obj 1:true** sinon (***obj*** étant l'objet "*pické*")
  (les notions de booléens seront développées dans le chapitre suivant).

- "***SaveStack***" et "***RclSavedStack***" sont intéressants car ils donnent la possibilité de sauver la pile en cours à un moment donné de l'éxécution d'un programme, ou de rappeler la pile précédemment sauvée. Ce rappel se fait en remplaçant la pile en cours par la pile sauvée (c'est l'équivalent de **UNDO**, mais pendant l'éxécution du programme).

Adresse	Mnémonique	Type	Commentaires
#61D41h	SaveStack	LM	Sauve la pile en cours.
#61F8Fh	RclSavedStack	LM	Rappelle la pile sauvée.

Voici, pour terminer, quelques adresses "*en vrac*", assez quelconques:

- #10301h drop;1-(s);dup;roll(s)
- #1045Ah pick3(s-3) (Rpl).
  Effectue un "*pick*" sur les objets situés aux niveaux s-1, s-2, s-3, l'entier-système s restant au niveau 4.
  EX: G,F,E,D,C,B,A,<5h> => G,F,E,D,C,B,A,<5h>,D,C,B
- #113C2h SizeStack (LM).
  Donne l'entier-système s=Depth+1.
- #41E00h 1+(s);drop(3)
- #5F616h 3pick;pick(s+4);swap
- #61184h pick(s+1);drop(2) (LM)
- #6416Dh depth=>s (Rpl).
  Donne la taille de la pile, sous la forme d'un entier-système.

LES BOOLÉENS

Un **booléen** est une quantité pouvant prendre exclusivement deux valeurs: la valeur **VRAI** (***true***) et la valeur **FAUX** (***false***). Les booléens interviennent de façon essentielle dans tout langage de programmation, notamment au niveau des instructions conditionnelles.

Au plus haut niveau, le langage de la HP48 utilise déjà les booléens dans les structures:

- **IF (booléen)...THEN...ELSE...END**
- **WHILE (booléen)...REPEAT...END**
- **DO...UNTIL (booléen) END.**

Dans ce cas le booléen **VRAI** est tout réel non nul, alors que le booléen **FAUX** est le réel 0.

Au niveau du langage-machine, cette notion est également très fortement présente (considérer les différents branchements conditionnels).

Au niveau intermédiaire que constitue la programmation par SYSEVALs, on trouve encore des booléens. Plus précisément il existe deux adresses qui correspondent symboliquement aux valeurs "***true***" et "***false***".

Ce sont les adresses:

- **#03A81h** pour le booléen ***true***.
- **#03AC0h** pour le booléen ***false***.

Ces deux adresses pointent sur des "*Primitive Code Objects*"

Un examen des deux routines (en langage machine) qui correspondent à ces adresses montrent qu'elles sont identiques!

Appelées par un SYSEVAL, ou évaluées par le biais de leur adresse dans une structure Rpl, elles se contentent de déposer leur propre adresse sur la pile.

Cette adresse n'étant pas celle d'un Rpl, sa présence sur la pile est représentée par le symbole générique **External**.

On ne voit pas de prime abord ce qui permet d'attribuer les valeurs ***true*** et ***false*** à l'une ou l'autre de ces adresses. En fait les milliers de sous-programmes écrits dans la ROM de la HP48 ont fait ce choix une fois pour toutes:

Quand le résultat d'un test est un booléen: si le test est positif, il y a évaluation de l'adresse **#03A81h** (sinon de l'adresse **#03AC0h**).

Quand une instruction conditionnelle est éxécutée et qu'elle nécessite la présence d'un booléen sur la pile, alors la HP48 examine si l'adresse placée au niveau 1 de la pile est **#03A81h**. Si tel est le cas le booléen est considéré comme ayant la valeur ***true***.

LES BOOLÉENS

Les deux tableaux ci-dessous contiennent nombre de SYSEVALs "*de base*" dans la gestion des booléens.

Adresse	Mnémonique	Type	Commentaires
#03A81h	true	LM	Place true au niveau 1
#03AC0h	false	LM	Place false au niveau 1
#03AF2h	not(?)	LM	Négation d'un booléen
#03ADAh	xor(2?)	LM	┐
#03B46h	and(2?)	LM	├ Opérations sur deux booléens
#03B75h	or(2?)	LM	│
#635B0h	nor(2?)	Rpl	┘ nor = or, puis not.
#25966h	3true	Rpl	Ces adresses permettent de placer
#2F934h	2false	Rpl	plusieurs booléens sur la pile
#634F7h	true;false	Rpl	
#6350Bh	false;true	Rpl	Exemples:
#0BBEDh	2true	Rpl	=> 2:true 1:true
#0BC01h	false;true	Rpl	=> 2:false 1:true

Adresse	Mnémonique	Type	Commentaires
#097A7h	true;swap	Rpl	Ces adresses permettent de
#0F1FFh	drop(1-2,4);true	Rpl	placer un ou plusieurs
#10029h	4dropn;true	Rpl	booléens sur la pile et
#1781Ah	drop(2);false	Rpl	s'accompagnent d'appels
#2164Ch	swap;false	Rpl	à swap, dup, drop, etc..
#21660h	drop(2);true	Rpl	
#239CFh	drop(1-5,7);true	Rpl	Par exemple:
#24717h	4dropn;false;true	Rpl	a,b,c,d,e => a,false,true
#25A66h	drop2;2true	Rpl	a,b,c => a,true,true
#25BBFh	3dropn;false;true	Rpl	a,b,c,d => a,false,true
#25C0Ah	drop;false;true	Rpl	a,b => a,false,true
#265C5h	dup;true	Rpl	a => a,a,true
#27F58h	drop;false;true	Rpl	a,b => a,false,true
#2FDF7h	5dropn;false	Rpl	a,b,c,d,e,f => a,false
#4F1D8h	swap;true	Rpl	a,b => b,a,true
#4F840h	drop;rot;false	Rpl	a,b,c,d => b,c,a,false
#4F886h	drop(3);rot;false	Rpl	a,b,c,d => c,d,a,false
#5A496h	drop;false	Rpl	a,b => a,false
#57351h	3dropn;true	Rpl	a,b,c,d => a,true
#5F5E4h	4dropn;false	Rpl	a,b,c,d,e => a,false
#5F657h	3dropn;true	Rpl	a,b,c,d => a,true
#5F67Fh	4dropn;false	Rpl	a,b,c,d,e => a,false
#5F6B1h	5dropn;false	Rpl	a,b,c,d,e,f => a,false
#62103h	drop;true	LM	a,b => a,true
#6210Ch	drop;false	LM	a,b => a,false
#62B0Bh	drop2;false	Rpl	a,b,c => a,false
#62EB7h	drop(2-3);true	Rpl	a,b,c,d => a,d,true
#680B4h	drop;false;true	Rpl	a,b => a,false,true
#68736h	drop(2);true	Rpl	a,b,c => a,c,true
#6B08Ah	drop(2-3);true	Rpl	a,b,c,d => a,d,true
#6D414h	drop(4);true	Rpl	a,b,c,d,e => a,c,d,e,true

Il existe de nombreux SYSEVALs effectuant des tests et dont le résultat est un booléen. Une grande partie d'entre eux se trouve dispersée dans les différents chapitres de cette partie du livre.

C'est le cas si le test s'applique à des objets d'un type particulier faisant l'objet d'un chapitre à lui seul. Le SYSEVAL correspondant figure alors dans ce chapitre-là.

Exemple:

L'adresse **#03CE4h** (mnémonique **<(2s)?**) permet de tester si s2<s1, où *s2* et *s1* sont deux entiers-système placés respectivement aux niveaux 2 et 1 de la pile.

Il m'a semblé plus judicieux de placer cette adresse dans le chapitre traitant des SYSEVALs sur les entiers-système....

D'une façon générale, quand une adresse correspond à un test dont le résultat est un booléen, je propose des mnémoniques dont le nom se termine par le caractère **?**.

◆ SYSEVALs testant des égalités d'adresse:

Voici par exemple les SYSEVALs utilisant une instruction que j'appelle **=?** et qui teste si les deux adresses des niveaux 2 et 1 sont identiques.

Je vous rappelle à ce propos que la pile de l'utilisateur est constituée d'adresses. Ces adresses renvoient à des objets, et la HP48 affiche ces objets sur la pile (créant ainsi l'illusion que ces objets y sont réellement).

Quand deux adresses sont identiques, les objets référencés le sont également (c'est évident). Inversement, deux objets peuvent être identiques sans être placés à la même adresse.

Trois exemples permettront de mieux comprendre les différences entre égalités d'objets et égalités d'adresses.

- Quand vous faites **5 ENTER 5 ENTER**: les deux objets entrés sont identiques, et leurs adresses le sont également (car la HP48 considère les entiers de **-9** à **9** comme des objets intégrés et elle les référence directement par leur adresse en ROM.
- Quand vous faites **11 ENTER 11 ENTER**: les deux objets entrés sont identiques, mais leurs adresses sont différentes (11 n'est pas un objet intégré: une nouvelle copie de cet entier est créée à chaque fois en RAM).
- Quand vous faites **11 ENTER ENTER**: les objets des niveaux 1 et 2 sont identiques, et il en est de même pour leurs adresses (l'instruction **DUP** se contentant en fait de dupliquer l'adresse de l'objet au niveau 1)

Adresse	Mnémonique	Type	Commentaires
#03B2Eh	=?	LM	=? teste l'égalité des adresses
#6303Dh	=?;over	Rpl	(ce qui est une notion différente
#635D8h	dup2;=?	Rpl	de l'égalité des objets désignés
#63605h	=?;or(2?)	Rpl	par ces adresses)

◆ SYSEVALs testant des égalités d'objets:

Voici maintenant trois adresses testant ***l'égalité d'objets*** (ou leur différence).

Le principe est le même que pour l'instruction **SAME**: les deux objets sont dits égaux s'ils sont de même type et s'ils ont la même composition en mémoire (on ne confondra pas avec le résultat donné par == qui prend en compte la valeur évaluée des nombres, noms ou expressions).

Par exemple:

- Si vous faites **15 ENTER 15 ENTER**, les deux objets sont égaux.
- Si vous faites **15 ENTER '7+8' ENTER**, ils sont différents.
- Si vous faites **'8+7' ENTER '7+8' ENTER**, ils sont différents.

Adresse	Mnémonique	Type	Commentaires
#03B97h	same?	LM	same? teste l'égalité des objets
#635C4h	NoSame?	Rpl	NoSame? : contraire de "Same?"
#63619h	same?;or(2?)	Rpl	

◆ Un petit programme pour démêler le vrai du faux:

Quand vous avez un booléen sur la pile, il est remplacé à l'écran par le terme très vague **External**. Il n'est donc pas facile de savoir s'il s'agit du booléen ***false*** ou du booléen ***true***.

Le programme suivant utilise le niveau 1 de la pile et il répond: **"V"** s'il a reconnu le booléen ***true***, et **"F"** s'il a reconnu le booléen ***false***. Il répond **"?"** s'il n'a reconnu ni l'un ni l'autre.

'CHKB': (Checksum #8311h; Taille 89.5 octets)

```
«   #634F7h SYSEVAL 3 ROLLD
    IF OVER SAME THEN DROP2 "V"
    ELSE SAME "F" "?" IFTE  END
»
```

◆ SYSEVALs testant le type des objets:

Pour chacun des types d'objets reconnus par la HP48, il existe un SYSEVAL permettant de savoir si un objet est ou n'est pas de ce type.

Dans la première partie du tableau ci-dessous, l'objet testé est perdu dans le test. Dans la seconde partie, il est sauvegardé par un "*dup*" préalable.

Adresse	Mnémonique	Type	Commentaires
#62025h	ch?	LM	Type "Character" ?
#6203Ah	gn?	LM	Type "Global Name" ?
#6204Fh	uo?	LM	Type "Unit Object" ?
#6211Ah	ln?	LM	Type "Local Name" ?
#6212Fh	s?	LM	Type "System Binary" ?
#62144h	b?	LM	Type "Binary" ?
#62159h	st?	LM	Type "String" ?
#6216Eh	r?	LM	Type "Real" ?
#62183h	c?	LM	Type "Complex" ?
#62198h	ar?	LM	Type "Array" ?
#621ADh	xn?	LM	Type "XLIB Name" ?
#621C2h	di?	LM	Type "Directory" ?
#621D7h	ex?	LM	Type "Expression" ?
#621ECh	rpl?	LM	Type "Rpl" ?
#62201h	go?	LM	Type "Graphic Object" ?
#62216h	li?	LM	Type "Liste" ?
#6222Bh	to?	LM	Type "Tagged Object" ?
#6223Bh	ra?	LM	Type "Real Array" ?
#62256h	ca?	LM	Type "Complex Array" ?
#6CFB4h	ch?	Rpl	Type "Character" ?
#37AE0h	dup;cc?	Rpl	
#62020h	dup;ch?	LM	
#62035h	dup;gn?	LM	
#6204Ah	dup;uo?	LM	
#62115h	dup;ln?	LM	Tests identiques aux précédents
#6212Ah	dup;s?	LM	
#6213Fh	dup;b?	LM	mais avec
#62154h	dup;st?	LM	
#62169h	dup;r?	LM	sauvegarde de l'objet
#6217Eh	dup;c?	LM	
#62193h	dup;ar?	LM	testé
#621A8h	dup;xn?	LM	
#621BDh	dup;di?	LM	
#621D2h	dup;ex?	LM	
#621E7h	dup;rpl?	LM	
#621FCh	dup;go?	LM	
#62211h	dup;li?	LM	
#62226h	dup;to?	LM	
#0F1C8h	<>gn?	LM	Type différent de "Global Name" ?

LES BOOLÉENS

Comme d'habitude, il y a des adresses inclassables, qui n'ont pas un intérêt fantastique. En voici quelques unes (ce sont toutes des routines du type Rpl):

- #265D9h 1 + (s);true
- #27797h pick6-9;false;true |pick6-9 = pick sur levels 6 à 9.
- #277B0h <7h>;pick6-9;false;true
- #62C55h TrueFalse? |and(?2,not(?1))
- #62C91h rot;and(2?)
- #6351Fh <0h>;false
- #63533h <1h>;false
- #6F2CDh not(?3) |rot;not(?);3rolld
- #45897h -(2s);true
- #4B3BEh dup;st?;3pick;st?;5pick;c?
  |Vérifie que obj3 est un complexe, et que obj2, obj1
  |sont des chaînes. obj3,obj2,obj1 ne sont pas perdus.
  |Ce test est effectué dans le cadre de l'instruction **AXES**.
- #4B3E1h dup;r?;3pick;r?;5pick;gn?
  |vérifie que obj3 est un nom global et que obj2 et obj1
  |sont réels. obj3,obj2,obj1 ne sont pas perdus. Ce test
  |est effectué dans le cadre des instr. **DEPND** et **INDEP**.
- #4B490h dup;st?;3pick;st? |teste si obj2,obj1 sont des chaînes.
- #4B4A9h dup;r?;3pick;r? |teste si obj2,obj1 sont des réels.
- #50923h Reverse4;true | a,b,c,d => d,c,b,a,true.

ENTIERS-SYSTEME

Nous avons vu, dans un chapitre précédent, la nature et l'utilité des ***entiers-système***. Un très grand nombre de tels entiers sont déjà présents dans la ROM de la HP48, et je vous ai donné leurs adresses, qui forment autant de SYSEVALs utiles.

Nous allons voir ici les adresses qui permettent d'effectuer toutes sortes d'opérations sur ces objets.

Les ***entiers-système*** apparaissent de façon constante dans le fonctionnement interne de la HP48. Beaucoup de SYSEVALs où ils interviennent sont donc présentés dans d'autres chapitres que celui-ci.

Nous nous limiterons ici aux adresses portant de façon essentielle sur les ***entiers-système***. Voici tout d'abord les SYSEVALs où sont codées les opérations les plus courantes.

Adresse	Mnémonique	Type	Commentaires
#03DEFh	1+(s)	LM	
#03E0Eh	1-(s)	LM	
#03E2Dh	2+(s)	LM	Ces différentes adresses
#03E4Eh	2-(s)	LM	
#03E6Fh	2*(s)	LM	permettent d'effectuer
#03E8Eh	2/(s)	LM	
#45301h	10+(s)	Rpl	une opération simple sur
#6256Ah	3+(s)	LM	
#6257Ah	4+(s)	LM	l'entier-système présent
#6258Ah	5+(s)	LM	
#6259Ah	6+(s)	LM	au niveau 1 de la pile.
#625AAh	7+(s)	LM	
#625BAh	8+(s)	LM	
#625CAh	9+(s)	LM	Exemples:
#625DAh	10+(s)	LM	
#625EAh	12+(s)	LM	On ajoute 12,
#625FAh	3-(s)	LM	
#6260Ah	4-(s)	LM	
#6261Ah	5-(s)	LM	On retranche 5,
#6262Ah	6-(s)	LM	
#6264Eh	10*(s)	LM	
#62674h	8*(s)	LM	On multiplie par 8.
#62691h	6*(s)	LM	
#386ECh	and(s,<Eh>)	Rpl	On ne garde que les bits 1,2,3
#6B030h	<80000h>+(s)	Rpl	On ajoute <80000h>
#03DBCh	+(2s)	LM	
#03DE0h	-(2s)	LM	Il s'agit ici d'effectuer une
#03EB1h	and(2s)	LM	opération simple sur les deux
#03EC2h	*(2s)	LM	entiers-système apparaissant
#191B9h	*(2s)	LM	aux niveaux 1 et 2 de la pile.
#624BAh	min(2s)	LM	
#624C6h	max(2s)	LM	

Adresse	Mnémonique	Type	Commentaires
#03CA6h #03CC7h #5A473h #5A487h #62289h #6229Ah #622A7h #622B6h #636C8h #63385h #63673h #636B4h #636F0h	=0(s)? <>0(s)? =4(s)? =1(s)? =3(s)? =2(s)? =1(s)? <>1(s)? <>2(s)? =1(s)? <3(s)? =5(s)? >1(s)?	LM LM Rpl Rpl LM LM LM LM Rpl Rpl Rpl Rpl Rpl	On réalise ici un test sur l'entier-système s présent au niveau 1. Le résultat est le booléen true si le test est positif et false sinon. L'entier-système s est perdu à l'issue du test. Exemples: s est-il inférieur à 3? s est-il égal à 5? s est-il plus grand que 1?
#03CE4h #03D19h #03D4Eh #03D83h	<(2s)? =(2s)? <>(2s)? >(2s)?	LM LM LM LM	Idem avec 2 entiers-système s2 et s1 (perdus dans le test) Ex: s2 est-il différent de s1 ?
#62266h #622C5h #622D4h #63687h #6289Bh #628B5h #628D1h	dup;=0(s)? dup;=1(s)? dup;<>0(s)? dup;<7(s)? dup2;<(2s)? dup2;=(2s)? dup2;>(2s)?	LM LM LM Rpl LM LM LM	D'autres tests, sur un ou deux entiers-système, permettant de ne pas perdre le ou les entiers testés.
#5853Eh #620EBh #6365Fh #636DCh #6364Bh #5A4A5h #6362Dh	over;<(2s)? over;=(2s)? over;<(2s)? over;>(2s)? over;=0(s)? drop;=1(s)? *(2s);=0?	Rpl LM Rpl Rpl Rpl Rpl Rpl	Encore des tests sur un ou deux entiers-système.

Adresse	Mnémonique	Type	Commentaires
#1DABBh #28558h #107A7h #626F7h #62809h #6281Ah #628EBh #6292Fh #62E26h #62FD9h	1+(s);rot 1-(s);3rolld 1-(s);true dup;2+(s) 1+(s);dup 1-(s);dup dup;1+(s) dup;1-(s) 1+(s);swap 1-(s);rot	Rpl Rpl Rpl LM LM LM LM LM Rpl Rpl	Ces SYSEVALs requièrent la présence d'un entier-système au niveau 1 de la pile.
#51843h #51857h #62904h #637E0h #637F4h	1+(s2) 1-(s2) swap;1+(s) swap;1-(s) drop;1-(s)	Rpl Rpl LM Rpl Rpl	2:s 1:x => 2:(s+1) 1:x 2:s 1:x => 2:(s-1) 1:x 2:s 1:x => 2:x 1:(s+1) 2:s 1:x => 2:x 1:(s-1) 2:s 1:x => 1:(s-1)

Adresse	Mnémonique	Type	Commentaires
#2198Bh	rot;1+(s);swap	Rpl	Ces SYSEVALs nécessitent la
#5E8A2h	3pick;2+(s)	Rpl	présence d'un entier-système
#5E8C0h	3pick;1+(s)	Rpl	au niveau 3 de la pile.
#5FB76h	1+(s3)	LM	3:s 2:x 1:y => 3:(s+1) 2:x 1:y
#637B8h	rot;1+(s)	Rpl	
#41D42h	-(2s);5+(s)	Rpl	
#624FBh	-(2s);2/(s)	LM	Ces SYSEVALS nécessitent la
#62794h	swap;-(2s)	LM	
#627D5h	+(2s);dup	LM	
#627F8h	-(2s);dup	LM	présence de deux
#62DFEh	+(2s);swap	Rpl	
#62E12h	-(2s);swap	Rpl	
#63051h	+(2s);over	Rpl	entiers-système, aux
#63065h	-(2s);over	Rpl	
#63704h	dup2;+(2s)	Rpl	
#6372Ch	over;+(2s)	Rpl	niveaux 1 et 2 de la pile
#6377Ch	over;-(2s)	Rpl	
#637A4h	swap;over;-(2s)	Rpl	
#637CCh	-(2s);1+(s)	Rpl	
#63808h	+(2s);1-(s)	Rpl	
#6CE1Ah	2/(s);swap;2/(s)	Rpl	
#62DCCh	rot;+(2s);swap	Rpl	Ici les niveaux 3 et 1
#63718h	rot;+(2s)	Rpl	doivent contenir
#63740h	3pick;+(2s)	Rpl	des entiers-système
#63768h	rot;-(2s)	Rpl	
#62DE5h	4pick;+(2s);swap	Rpl	Même chose avec les
#63754h	4pick;+(2s)	Rpl	niveaux 4 et 1.

Adresse	Mnémonique	Type	Commentaires
#03EF7h	mdv(2s)	LM	Ici "mdv" signifie Mod,Div.
#04DD7h	mdv(s,256)	LM	2:s2 1:s1 => 2: s2 mod s1
#46855h	mdv(<14h>,s)	Rpl	1: s2 div s1

Voici quelques adresses dont l'intérêt m'échappe un peu....

```
#251ECh  6rolld;1+(s5)                (Rpl) s5: s au niveau 5
#25336h  drop(2);6rolld;1+(s5)        (Rpl)
#2F2E8h  drop(2-4);<FFFFFh>           (Rpl)
#39FE3h  begin<5h>end                 Rpl ne contenant que <5h>.
#39FFEh  begin<16h>end                Rpl ne contenant que <16h>.
#41DC9h  -(s,<18h>);<4h>;<5h>         (Rpl)
#4280Eh  3s=>(s3-s2+s1)               (Rpl)3:<m> 2:<n> 1:<p> → 1:<m-n+p>
#60035h  over;pick(s+3);2+(s)         (Rpl)
#60058h  over;pick(s+3);1+(s)         (Rpl)
#62E4Eh  1-(s);<1h>;swap              (Rpl)
#685C9h  dup;pick(s+1);+(2s);1+(s)    (Rpl)
```

ENTIERS-SYSTEME

Voici des adresses permettant de placer sur la pile, d'un coup, plusieurs entiers-système simples. Là encore, il s'agit en une seule adresse d'effecteur le travail de plusieurs autres (il n'y a pas de petit profit).

Adresse	Mnémonique	Type	Commentaires
#41DB5h	<0h>;<6h>	Rpl	
#63AC4h	<1h>;<1h>	Rpl	Ces Sysevals permettent de placer
#641FCh	2<0h>	LM	
#64209h	<0h>;<1h>	LM	simultanément plusieurs
#6427Ah	<0h>;<7h>	LM	
#6428Ah	<1h>;<1Bh>	LM	entiers-système sur la pile.
#6429Dh	<2h>;<1h>	LM	
#642AFh	2<2h>	LM	
#642BFh	<2h>;<4h>	LM	Exemples:
#642D1h	<3h>;<4h>	LM	
#642E3h	<5h>;<4h>	Rpl	=> 2:<5h> 1:<4h>
#642F7h	<5h>;<4h>	LM	
#64309h	3<0h>	Rpl	=> 3:<0h> 2:<0h> 1:<0h>
#6431Dh	2<0h>;<1h>	Rpl	
#64331h	2<0h>;<2h>	Rpl	=> 3:<0h> 2:<0h> 1:<2h>
#6CE88h	3<0h>;<7h>	Rpl	

Les adresses de ce dernier tableau sont un peu dans le style des précédentes, avec des mouvements sur la pile, en prime.

Adresse	Mnémonique	Type	Commentaires
#14314h	drop;<7h>	Rpl	
#16DBDh	Rep2(<FFFFFh>)	Rpl	Remplace objet2 par <FFFFFh>
#2530Eh	<0h>;5rolld	Rpl	
#364BCh	4pick;over;<2h>	Rpl	
#364D0h	<1h>;4pick;over	Rpl	
#45073h	swap;<0h>	Rpl	
#481C5h	drop2;<1h>	Rpl	
#51238h	drop;<83h>	Rpl	
#51260h	drop;<40h>	Rpl	
#5A734h	drop(2);<1h>	Rpl	drop(2) = drop sur level 2
#62535h	drop;<0h>	LM	
#6254Eh	drop2;2<0h>	LM	drop2 = drop sur levels 1 et 2
#62946h	drop;<1h>	LM	
#62E3Ah	<0h>;swap	Rpl	
#62E67h	<1h>;swap	Rpl	
#63079h	<0h>;over	Rpl	
#63A88h	dup;<0h>	Rpl	
#63A9Ch	dup;<1h>	Rpl	
#63AB0h	swap;<1h>	Rpl	
#63AD8h	dup;<2h>	Rpl	
#67E84h	<0h>;swap;<2h>	Rpl	

ENTIERS BINAIRES

Nous avons vu précédemment les adresses des différents ***entiers binaires*** présents dans la ROM de la HP48.

Ce ne sont pas des SYSEVALs aussi intéressants que ceux qui portent sur les ***entiers-système***, car les entiers binaires ainsi obtenus sont en général trop particuliers et donc difficilement réutilisables.

Je me contenterai de rappeler les deux adresses suivantes:

```
#055D5h   E4A20 50000    (Binaire vide)
#10E34h   E4A20 51000 0000000000000000 (Binaire nul: #0h)
```

Voyons tout d'abord les SYSEVALs qui permettent d'effectuer une opération simple sur un ou deux entiers-binaires.

Adresse	Mnémonique	Type	Commentaires
#53D4Eh	not(b)	LM	
#53D5Eh	sl(b)	LM	
#53D6Eh	slb(b)	LM	Opérations sur l'entier
#53D81h	sr(b)	LM	
#53D91h	srb(b)	LM	binaire placé au niveau 1
#53DA4h	rr(b)	LM	
#53DE1h	rrb(b)	LM	
#53E0Ch	rl(b)	LM	
#53E3Bh	rlb(b)	LM	
#53E65h	asr(b)	LM	
#53EC3h	neg(b)	LM	
#53D04h	and(2b)	LM	
#53D15h	or(2b)	LM	
#53D26h	xor(2b)	LM	
#53EA0h	+(2b)	LM	
#53EB0h	-(2b)	LM	Opérations sur les entiers
#53ED3h	*(2b)	LM	
#53F05h	/(2b)	LM	binaires des niveaux 1 et 2
#544D9h	==(2b)	LM	
#544ECh	<>(2b)	Rpl	Ces tests donnent le résultat
#54500h	>(2b)	LM	réel 0 ou 1, et non pas un
#5452Ch	≥(2b)	LM	résultat booléen.
#5453Fh	≤(2b)	LM	
#54552h	<(2b)	LM	
#5429Fh	/(r,b)	Rpl	
#542BDh	/(b,r)	Rpl	Opérations arithmétiques
#542D1h	*(r,b)	Rpl	faisant intervenir un réel et
#542EAh	*(b,r)	Rpl	un entier binaire:
#542FEh	-(r,b)	Rpl	
#5431Ch	-(b,r)	Rpl	Le résultat est un
#54330h	+(r,b)	Rpl	entier binaire.
#54349h	+(b,r)	Rpl	

◆ Remarques sur les adresses précédentes:

Rappelons que la taille d'un objet entier binaire entré au clavier est toujours égale à 26 quartets:

- 5 pour l'adresse-préfixe,
- 5 pour la longueur hors adresse-préfixe,
- 16 quartets représentant les différents chiffres hexadécimaux).

Il est cependant possible de créer des entiers binaires ayant moins ou plus de 16 chiffres hexadécimaux (c'est le cas pour la plupart des entiers-binaires présents dans la ROM de la HP48).

Malheureusement, les entiers-binaires résultant de l'utilisation des adresses précédentes sont toujours codés sur 26 quartets (indépendamment de la taille des entiers-binaires servant d'argument).

On est ainsi privé (je pense aux opérations arithmétiques) de ce qui aurait pu constituer à moindres frais une bibliothèque d'opérations sur les entiers en précision infinie....

◆ "Décortiquer" les entiers binaires:

Nous allons voir quelques adresses plus "hard" sur les entiers binaires. Il est possible de créer des entiers binaires de longueur quelconque, de les concaténer, d'en extraire une partie....

Adresse	Mnémonique	Type	Commentaires
#0EDE1h	makeb(s)	Rpl	L'instruction makeb(s) crée un un entier binaire nul, dont le nombre de chiffres hexadécimaux est s (s est un entier-système)
#0EDA5h	<Ah>;makeb(s)	Rpl	
#0EDB9h	<Dh>;makeb(s)	Rpl	
#0EDCDh	<18h>;makeb(s)	Rpl	
#61C1Ch	add0(b,s)	LM	Ajoute s zéros à l'entier-binaire b, pour obtenir ainsi un binaire de longueur longueur(b)+s. La valeur de b ne change pas.
#0518Ah	conc(2b)	LM	Concatène les deux binaires b2,b1
#05815h	sub(b,2s)	Rpl	Extrait le binaire formé par les quartets s2 à s1.

◆ utilisation des SYSEVALs précédents:

▸ **Concaténer deux entiers binaires:**

Entrez au clavier: #123456789h 'A' STO #AABBCCDDEEFF00h 'B' STO
L'entier binaire 'A' est codé: E4A20 51000 9876543210000000
L'entier binaire 'B' est codé: E4A20 51000 00FFEEDDCCBBAA00

Faites: **A B #518Ah SYSEVAL**

Vous devez voir apparaître l'entier binaire: #123456789h, et vous pouvez penser qu'il s'agit de l'entier binaire 'A'.Il n'en est rien comme on le constate en éditant l'objet-obtenu.
On obtient en ligne de commande:
C# 32 9876543210000000FFEEDDCCBBAA00,
ce qui prouve qu'on a obtenu un entier binaire de 32 chiffres hexadécimaux, par concaténation de 'A' et 'B'.
Cet entier binaire, est codé en mémoire:
E4A20 52000 9876543210000000FFEEDDCCBBAA00

▸ **Allonger la taille d'un entier binaire:**

Rien de plus simple avec l'adresse **#61C1Ch** (mnémonique ***add0(b,s)***).

Le programme suivant allonge un entier binaire (placé au niveau 2) en lui ajoutant **n** zéros (n étant un réel spécifié au niveau 1).

'**ADD0B**' (Checksum: #8271h; Taille 50..5 octets)
« #2EC11h SYSEVAL #61C1Ch SYSEVAL »

Faites #123456789h ' A' STO.
L'entier binaire 'A' est codé: E4A20 51000 9876543210000000

Faites ensuite: **A 10 ADD0B**
Vous devez encore voir apparaître #123456789h à l'écran, mais si éditez cet entier-binaire, vous obtenez: C# 26 98765432100000000000000000

On a ainsi allongé l'entier 'A' de 10 chiffres tous nuls (et celà ne change pas la valeur de l'entier), et le nouvel entier est codé:
E4A20 F1000 98765432100000000000000000

ENTIERS BINAIRES

- **Créer un entier binaire très long:**

  C'est possible avec l'adresse **#0EDE1h** (mnémonique ***makeb(s)***).

  Le programme suivant fabrique un entier binaire nul de longueur **L**, où **L** est un nombre réel spécifié au niveau 1 (par "*longueur*" on entend ici le nombre de chiffres hexadécimaux)

  '**DOB0**' (Checksum: #EB8Bh; Taille 49.5 octets)
  « #2EC11h SYSEVAL #EDE1h SYSEVAL »

  Faites par exemple: **234 DOB0**
  Vous obtenez #0h à l'écran, mais si vous éditez cet objet, vous voyez C# 234 00000000000000..... .Vous avez ainsi créé un entier binaire formé de 234 chiffres tous nuls.

- **Découpez une partie d'un entier binaire:**

  L'adresse **#05815h** (mnémonique ***sub(bi,2s)***) le permet. Elle attend un entier binaire **b** au niveau 2, et deux entiers-système **s2** et **s1** aux niveaux 2 et 1.
  Elle crée l'entier binaire **b'** dont les chiffres hexadécimaux sont les chiffres occupant les positions **s2** à **s1** dans **b**.

  Le programme suivant va nous permettre de tester ce SYSEVAL.
  Ici **s2** et **s1** sont remplacés par deux réels **n2** et **n1**.

  '**CUTB**' (Checksum #9D30h; Taille 70 octets)
  « SWAP #2EC11h SYSEVAL SWAP #2EC11h SYSEVAL
  #5815h SYSEVAL
  »

  Faites #123456789ABCDEFh 'A' STO.
  L'entier binaire 'A' est codé: E4A20 51000 FEDCBA9876543210
  Faites ensuite: **A 3 9 CUTB 'B' STO**
  L'entier 'B' est affiché #789ABCDh, et il est codé en mémoire:
  E4A20 C0000 DCBA987

  Il n'est donc plus formé que de 7 chiffres hexadécimaux, ceux qui occupaient les places 3 à 9 dans le codage de A.
  NB: L'ordre des chiffres en mémoire est l'inverse de l'ordre des chiffres affichés sur la pile...

NOMBRES RÉELS

Une bonne partie des SYSEVALs décrits ici correspondent aux adresses de traitement des fonctions de la HP48, dans le cas où leurs arguments sont des nombres réels. Voyons tout d'abord les adresses correspondant aux fonctions prenant un seul argument, dans le cas où celui-ci est réel.

Adresse	Mnémonique	Type	Adresse	Mnémonique	Type
#1B30Dh	arg(r)	Rpl	#1B3F5h	√(r)	Rpl
#1B47Bh	sq(r)	Rpl	#1B6EAh	asin(r)	Rpl
#1B775h	acos(r)	Rpl	#1B86Ch	acosh(r)	Rpl
#1B8DEh	atanh(r)	Rpl	#1B995h	ln(r)	Rpl
#1BA0Ch	log(r)	Rpl	#1C28Dh	sf(r)	Rpl
#1C2EEh	cf(r)	Rpl	#1C32Ch	fs?(r)	Rpl
#1C379h	fc?(r)	Rpl	#1C403h	fix(r)	Rpl
#1C437h	sci(r)	Rpl	#1C46Bh	eng(r)	Rpl
#1C4BAh	fs?c(r)	Rpl	#1C539h	fc?c(r)	Rpl
#1E8D9h	not(r)	Rpl	#2A622h	d->r(r)	Rpl
#2A655h	r->d(r)	Rpl	#2A8D7h	sign(r)	LM
#2A900h	abs(r)	LM	#2A920h	neg(r)	LM
#2A930h	mant(r)	LM	#2AAAFh	inv(r)	LM
#2AB2Fh	exp(r)	LM	#2AB42h	expm(r)	LM
#2ABA7h	lnp1(r)	LM	#2ABBAh	alog(r)	LM
#2ABEFh	sin(r)	LM	#2AC40h	cos(r)	LM
#2AC91h	tan(r)	LM	#2AD21h	atan(r)	LM
#2ADAEh	sinh(r)	LM	#2ADDAh	cosh(r)	LM
#2ADEDh	tanh(r)	LM	#2AE00h	asinh(r)	LM
#2AE39h	xpon(r)	LM	#2AF4Dh	fp(r)	LM
#2AF60h	ip(r)	LM	#2AF73h	ceil(r)	LM
#2AF86h	floor(r)	LM	#2B044h	rdz(r)	Rpl
#2B0C4h	fact(r)	Rpl			

Viennent ensuite les adresses où sont traitées les fonctions usuelles quand elles prennent deux arguments réels.

Adresse	Mnémonique	Type	Adresse	Mnémonique	Type
#1B124h	^(2r)	Rpl	#1C0B5h	%(2r)	Rpl
#1E7DDh	and(2r)	Rpl	#1E863h	or(2r)	Rpl
#1E946h	xor(2r)	Rpl	#1EC40h	<(2r)	Rpl
#1ECDFh	>(2r)	Rpl	#1ED7Eh	≤(2r)	Rpl
#1EE1Dh	≥(2r)	Rpl	#2A6F5h	max(2r)	Rpl
#2A70Eh	min(2r)	Rpl	#2A974h	+(2r)	LM
#2A981h	-(2r)	LM	#2A9BCh	*(2r)	LM
#2A9C9h	%(2r)	LM	#2A9FEh	/(2r)	LM
#2AA0Bh	%T(2r)	LM	#2AA30h	%CH(2r)	LM
#2AA81h	xroot(2r)	LM	#2ABDCh	mod(2r)	LM
#2AE62h	comb(2r)	LM	#2AE75h	perm(2r)	LM
#2AFC2h	rand	LM	#2B529h	rnd(2r)	Rpl
#2B53Dh	trunc(2r)	Rpl	#35804h	+(2r)	LM
#3581Eh	-(2r)	LM			

On trouve ensuite des adresses où sont éxécutées certaines fonctions de la HP48 prenant un ou deux paramètres réels, quand on sait que ce ou ces paramètres ont des propriétés particulières.

Utiliser ces adresses permet donc un gain de temps. Toutes les routines obtenues sont d'ailleurs écrites en langage machine.

Adresse	Mnémonique	Type	Commentaires
#2AA70h	^(2r)	LM	quand résultat réel
#2AB09h	√(r≥0)	LM	racine carrée d'un réel ≥ 0
#2AB6Eh	ln(r≥0)	LM	logarithme népérien d'un réel ≥ 0
#2AB81h	log(r≥0)	LM	logarithme décimal d'un réel ≥ 0
#2ACC1h	asin(-1≤r≤1)	LM	arc sinus d'un réel de [-1,1]
#2ACF1h	acos(-1≤r≤1)	LM	arc cosinus d'un réel de [-1,1]
#2AE13h	acosh(r≥1)	LM	arg Cosh d'un réel ≥ 1
#2AE26h	atanh(-1<r<1)	LM	arc tangente d'un réel de]-1,1[
#2AE4Ch	fact(r≥0)	LM	prend d'abord abs et ceil de r
#2B07Bh	rdz(r<>0)	LM	rdz appliquée à un réel non nul

Dans la première partie du tableau ci-dessous, vous trouverez quelques adresses, prenant pour arguments de un à trois réels, et qui peuvent se révèler utiles. L'utilité de la deuxième partie du tableau est laissée à votre appréciation souveraine....

Adresse	Mnémonique	Type	Commentaires
#2AD38h	arg(2r)	LM	argument de (x,y) avec 2:x, 1:y
#4B51Ch	sort(2r)	Rpl	2:x 1:y => 2:min(x,y) 1:max(x,y)
#503B1h	dist(2r)	Rpl	2:x 1:y => 1:abs(x-y)
#2AFACh	0rnd(r)	LM	arrondit à l'entier le + proche
#0E6D4h	0rnd(abs(r))	Rpl	(idem ci-dessus, mais d'abord abs)
#2B48Eh	r->p	Rpl	x,y=>r,θ (cartés. -> polaires)
#2B4BBh	p->r	Rpl	r,θ=>x,y (polaires -> cartés.)
#2B4F2h	sp->r	Rpl	r,θ,Φ=>x,y,z (sphér. -> cartés.)
#2EBC6h	4+(r)	Rpl	augmente un réel de 4
#50262h	1+(r)	Rpl	incrémente réel
#50276h	1-(r)	Rpl	décrémente réel
#62BF1h	10*(r)	Rpl	multiplie un réel par 10.
#51BE4h	+(2r);swap	Rpl	3:a 2:x 1:y => 2:x+y 1:a
#1CA0Dh	drop;1	Rpl	2:a 1:b => 2:a 1:1
#1F047h	drop2;0	Rpl	3:a 2:b 1:c => 2:a 1:0
#4949Bh	1;-1	Rpl	1:a => 3:a 2:1 1:-1
#5198Fh	drop;0	Rpl	2:a 1:b => 2:a 1:0
#50A3Bh	drop(2,3);0	Rpl	4:a 3:b 2:c 1:d => 3:a 2:d 1:0
#52C4Ah	4dropn;0;0	Rpl	5:a 4:b 3:c 2:d 1:e => 3:a 2:0 1:0
#554B3h	drop(2);1;swap	Rpl	3:a 2:b 1:c => 3:a 2:1 1:c
#56AFBh	4dropn;0	Rpl	5:a 4:b 3:c 2:d 1:e => 2:a 1:0
#56D12h	drop;0	Rpl	2:a 1:b => 2:a 1:0
#54CDBh	minr	Rpl	┐
#54D12h	maxr	Rpl	├ Comme instructions MINR, MAXR,
#54D35h	pi	Rpl	├ π, i, ou e, à ceci près que
#54D58h	i	Rpl	├ l'on ne vérifie pas que la
#54D7Bh	e	Rpl	┘ pile n'est pas saturée.

Ce dernier tableau présente des SYSEVALs effectuant des tests sur un ou deux réels, et renvoyant un résultat booléen (true ou false).

Adresse	Mnémonique	Type	Commentaires
#2A738h	<0(r)?	LM	le réel est-il strict. négatif ?
#2A76Bh	=0(r)?	LM	le réel est-il nul ?
#2A799h	>0(r)?	LM	le réel est-il strict. positif ?
#2A7CFh	<>0(r)?	Rpl	le réel est-il non nul ?
#2A7F7h	≥0(r)?	Rpl	le réel est-il positif ou nul ?
#2A871h	<(2r)?	LM	r2 est-il inférieur à r1 ?
#2A88Ah	>(2r)?	LM	r2 est-il supérieur à r1 ?
#2A8A0h	≥(2r)?	LM	r2 est-il supérieur ou égal à r1 ?
#2A8B6h	≤(2r)?	LM	r2 est-il inférieur ou égal à r1 ?
#2A8C1h	=(2r)?	LM	les deux réels sont-ils égaux ?
#2A8CCh	<>(2r)?	LM	les deux réels sont-ils distincts?

N.B: On ne confondra pas, par exemple, les adresses:

- **#1ECDFh** (mnémonique **>(2r)**) et
- **#2A88Ah** (mnémonique **>(2r)?**)
  qui testent toutes les deux si le réel au niveau 2 est supérieur au réel du niveau 1:

- La première adresse renvoie 1 si le test est positif, 0 sinon.
- La seconde adresse renvoie ***true*** si le test est positif, ***false*** sinon.

NOMBRES COMPLEXES

On a vu que la ROM de la HP48 ne contient que 5 nombres complexes:
(-1,0) à l'adresse #5196Ah, (0,0) à l'adresse #524AFh,
(1,0) à l'adresse #524F7h, (0,1) à l'adresse #5267Fh, et
(0,-1) à l'adresse #526AEh.

Voici les adresses de traitement de certaines fonctions usuelles, quand leurs arguments sont un ou deux nombres compexes, ou un réel et un complexe (dans ce cas l'ordre est important):

Adresse	Mnémonique	Type	Adresse	Mnémonique	Type
#1B48Fh	sq(c)	Rpl	#1DD29h	v->(c)	Rpl
#519A3h	re(c)	Rpl	#519B7h	im(c)	Rpl
#51B70h	neg(c)	LM	#51BB2h	conj(c)	LM
#51EFAh	inv(c)	Rpl	#52062h	abs(c)	Rpl
#52099h	arg(c)	Rpl	#520CBh	sign(c)	Rpl
#52107h	√(c)	Rpl	#52193h	exp(c)	Rpl
#521E3h	ln(c)	Rpl	#522BFh	log(c)	Rpl
#52305h	alog(c)	Rpl	#52530h	sin(c)	Rpl
#52571h	cos(c)	Rpl	#525B7h	tan(c)	Rpl
#5262Fh	sinh(c)	Rpl	#52648h	cosh(c)	Rpl
#5265Ch	tanh(c)	Rpl	#52675h	atan(c)	Rpl
#527EBh	atanh(c)	Rpl	#52804h	asin(c)	Rpl
#5281Dh	asinh(c)	Rpl	#52836h	acosh(c)	Rpl
#52863h	acos(c)	Rpl			
#37C20h	+(2c)	Rpl	#37CB1h	-(2c)	Rpl
#51C16h	+(2c)	Rpl	#51CFCh	-(2c)	Rpl
#51EC8h	/(2c)	Rpl	#52374h	^(2c)	Rpl
#1EA6Ch	==(r,c)	Rpl	#1EA76h	==(c,r)	Rpl
#1EB8Dh	<>(r,c)	Rpl	#1EB97h	<>(c,r)	Rpl
#35EC2h	rnd(c,r)	Rpl	#35F17h	trunc(c,r)	Rpl
#37C48h	+(r,c)	Rpl	#37C66h	+(c,r)	Rpl
#37CC5h	-(r,c)	Rpl	#37CD9h	-(c,r)	Rpl
#51BD0h	+(c,r)	Rpl	#51BF8h	+(r,c)	Rpl
#51CD4h	-(r,c)	Rpl	#51CE8h	-(c,r)	Rpl
#51D4Ch	*(c,r)	Rpl	#51D60h	*(r,c)	Rpl
#51D88h	*(2c)	Rpl	#51E19h	/(r,c)	Rpl
#51E64h	/(c,r)	Rpl	#52342h	^(r,c)	Rpl
#52360h	^(c,r)	Rpl			

Voici enfin quelques adresses, faciles à interpréter:

```
#05C27h   r->c            (LM)     2:x 1:y => 1:(x,y)
#05D2Ch   c->             (LM)     1:(x,y) => 2:x 1:y
#632A9h   swap;r->c       (Rpl)    2:x 1:y => 1:(y,x)
#4FE12h   0rnd(c);c->     (Rpl)    1:(x,y) => 1:(m,n) => 2:m 1:n
#51A4Ah   *(i,c)          (LM)     Multiplie par i=(0,1).
#51A5Fh   *(-i,c)         (LM)     Multiplie par -i=(0,-1).
#51B43h   =0(c)?          (LM)     Renvoie un résultat booléen
#51C6Bh   c->;rot;c->     (Rpl)    2:(a,b) 1:(c,d) => 4:c 3:d 2:a 1:b
```

RÉELS LONGS

La HP48 utilise les ***réels longs*** dans ses calculs mathématiques (fonctions usuelles, tableaux), pour assurer la précision du résultat final.

Il n'en reste pas moins que la plus grande partie des opérations usuelles sur les réels sont également implantées sur les réels longs, comme le montre le tableau suivant (les arguments sont des réels longs, et le résultat est également un réel long):

Adresse	Mnémonique	Type	Commentaires
#2A8F0h	abs(rr)	LM	valeur absolue d'un réel long
#2A910h	neg(rr)	LM	changement de signe
#2AA92h	inv(rr)	LM	inverse d'un réel long
#2AB1Ch	exp(rr)	LM	exponentielle d'un réel long
#2AB94h	lnp1(rr)	LM	LNP1 appliquée à un réel long
#2AC06h	sin(rr_)	LM	rr est ici dans le mode en cours
#2AC17h	sin(rr°)	LM	rr est ici en degrés
#2AC27h	sin(rr)	LM	rr est ici en radians
#2AC57h	cos(rr_)	LM	rr exprimé dans le mode en cours
#2AC68h	cos(rr°)	LM	rr est ici en degrés
#2AC78h	cos(rr)	LM	rr est exprimé en radians
#2ACA8h	tan(rr)	LM	rr est exprimé en radians
#2ACD8h	asin(rr)	LM	résultat en radians
#2AD08h	acos(rr)	LM	résultat en radians
#2AD95h	sinh(rr)	LM	sinus hyperbolique d'un réel long
#2ADC7h	cosh(rr)	LM	cosinus hyperbolique
#2AF27h	->hms(rr)	LM	transformation au format HMS
#2AF99h	floor(rr)	LM	plus grand réel long entier ≤ rr
#2A6DCh	max(2rr)	Rpl	maximum de deux réels longs
#2A943h	+(2rr)	LM	somme de deux réels longs
#2A94Fh	-(2rr)	LM	différence
#2A99Ah	*(2rr)	LM	produit
#2A9E8h	/(2rr)	LM	quotient

Les adresses qui suivent permettent d'effectuer un test sur un ou deux réels longs. Le résultat obtenu est un booléen (***true*** ou ***false***).

Adresse	Mnémonique	Type	Commentaires
#2A727h	<0(rr)?	LM	Le réel long est-il négatif ?
#2A75Ah	=0(rr)?	LM	Le réel long est-il nul ?
#2A788h	>0(rr)?	LM	Le réel long est-il positif ?
#2A7BBh	<>0(rr)?	Rpl	Le réel long est-il non nul ?
#2A7E3h	≥0(rr)?	Rpl	Est-il positif ou nul ?
#2A80Bh	≤0(rr)?	Rpl	Est-il négaitf ou nul ?
#2A81Fh	<(2rr)?	LM	rr2 est-il inférieur à rr1 ?
#2A87Fh	>(2rr)?	LM	rr2 est-il supérieur à rr1 ?
#2A895h	≥(2rr)?	LM	rr2 est-il ≥ à rr1 ?
#2A8ABh	≤(2rr)?	LM	rr2 est-il ≤ à rr1 ?

RÉELS LONGS

Voici d'autres adresses, sans doute moins utiles, qui prennent de un à quatre réels longs comme arguments, et qui renvoient un résultat sous la forme d'un ou de deux réels longs.

Adresse	Mnémonique	Type	Commentaires
#2AD4Fh	arg(2rr)m	LM	résultat dans le mode en cours
#2AD6Ch	arg(2rr)°	LM	résultat en rr degrés
#2AD7Ch	arg(2rr)	LM	résultat en rr radians
#2AA5Fh	^(2rr)	LM	quand résultat réel long
#2AAEAh	√(rr≥0)	LM	quand le réel long est ≥ 0
#2AB5Bh	ln(rr≥0)	LM	quand le réel long est ≥ 0
#2B1FAh	fact(rr.)	Rpl	quand rr n'est pas entier
#2B498h	rr->pp	Rpl	cartésiennes => polaires
#2B4C5h	pp->rr	Rpl	polaires => cartésiennes
#2C1DFh	1+(rr)	Rpl	incrémente un réel long
#51A94h	Σsq(2rr)	Rpl	somme des 2 carrés.
#51ADFh	πcplx(4rr)	Rpl	a,b,c,d => ac-bd,ad+bc

Adresse	Mnémonique	Type	Commentaires
#2A95Bh	-(2r)=>rr	LM	Ces SYSEVALs ont l'originalité
#2A9A6h	*(2r)=>rr	LM	de prendre un ou deux arguments
#2AA9Eh	inv(r)=>rr	LM	réels et de renvoyer un résultat
#2AAF6h	√(r≥0)=>rr	LM	réel long
#2AF3Ah	hms->(r)=>rr	LM	
#2AD5Bh	arg(2r)=>rr	LM	(résultat dans le mode en cours)

Comme d'habitude, il me reste quelques adresses à "solder". Elles correspondent toutes à des routines Rpl. Je vous laisse le soin de les décrypter. A vous de voir si elles peuvent être utiles.

```
#4B2D3h   (a,b)(c,d)=>d-b,c-a   |Résultats réels longs. Utilisé
             pour calculer la longueur du range en y et celle du
             range en x dans le cas où (a,b)=PMIN et (c,d)=PMAX.
#4F241h   r=>rr;rot;*(2rr);+(2rr);rr=>r
#4F264h   r=>rr;rot;*(2rr);-(2rr);rr=>r
#51DFBh   axy=>(x/a,y/a)      | a,x,y sont ici des réels longs
#520B2h   dup2;abs(2rr)       | xx,yy=>xx,yy,abs(xx,yy)
#521A7h   exp(,i)(2rr)        | calcule exp( rr2 + i*rr1 )
#5291Ch   drop2;%%0           | %%0 est ici le réel long nul
#529F3h   drop;<0(rr)?
#52A07h   drop(2);≥0(rr)?
#62EA3h   *(2rr);swap
#62FEDh   *(2rr);rot
#63B82h   /(2rr)=>r          |Division avec résultat réel
#63BBEh   swap;/(2rr)
#63C18h   *(2rr);3rolld
```

COMPLEXES LONGS

Les ***complexes longs*** sont (pour la HP48) aux nombres complexes ce que sont les réels longs aux nombres réels: un moyen de conserver une précision maximale pendant certains calculs intermédiaires, de façon à être sûr de la précision du résultat final.

Les SYSEVALs portant sur les complexes longs sont assez peu nombreux. D'ailleurs les voici...

Adresse	Mnémonique	Type	Commentaires
#51B91h	neg(cc)	Rpl	changement de signe
#51BC1h	conj(cc)	Rpl	conjugue un complexe long
#52080h	abs(cc)	Rpl	valeur absolue : un réel long
#51C3Eh	+(2cc)	Rpl	
#51D10h	-(2cc)	Rpl	Opérations entre deux nombres
#51DE2h	*(2cc)	Rpl	complexes longs
#51F13h	/(2cc)	Rpl	
#51C9Dh	+(cc,rr)	Rpl	
#51CB1h	+(rr,cc)	Rpl	La position respective du complexe
#51D24h	-(rr,cc)	Rpl	
#51D38h	-(cc,rr)	Rpl	long et du réel long sont
#51DABh	*(cc,rr)	Rpl	
#51DBFh	*(rr,cc)	Rpl	importantes
#51F3Bh	/(rr,cc)	Rpl	
#51F7Ch	/(cc,rr)	Rpl	
#05DBCh	cc->	LM	1:(xx,yy) => 2:xx 1:yy
#35F6Ch	drop;re(cc)	Rpl	2:(x,y) 1:a => 1:x
#36DEBh	neg&conj(cc)	Rpl	1:(x,y) => 1:(-x,y)
#51B2Ah	=0(cc)?	LM	Donne un résultat booléen
#51C84h	cc->;rot;cc->	Rpl	2:(x,y) 1:(z,t) => 4:z 3:t 2:x 1:y

Profitant que la récolte en SYSEVALS est maigre, voici quelques autres adresses, s'appliquant à un nombre (noté ici "***nb***"), qu'il soit réel ou complexe, *normal* ou *long*.

Adresse	Mnémonique	Type	Commentaires
#37BE9h	+(2nb)	Rpl	┐ Quelque soit la nature du nombre
#37C7Ah	-(2nb)	Rpl	├ (réel ou complexe, long ou pas)
#37D5Bh	abs(nb)	Rpl	│ Les deux arguments doivent être de
#37D7Eh	neg(nb)	Rpl	┘ même type
#37DA1h	=0?(nb)	Rpl	Le résultat est un booléen
#3675Ah	ToNorm(nb)	Rpl	Réduit à la forme "normale"
#37DC4h	ToLong(nb)	Rpl	Passe à la forme "longue"

LES TABLEAUX

Le nombre de SYSEVALs qui s'appliquent aux tableaux est assez conséquent, non seulement parce que le nombre des opérations possibles est important, mais encore parce qu'il y a parfois plusieurs combinaisons possibles d'arguments pour une fonction donnée.

Commençons par quelques adresses simples (toutes correspondent à des routines Rpl, et les mnémoniques proposés sont suffisamment explicites pour éviter tout commentaire). Notez simplement que l'ordre des arguments est important.

Rappels des notations utilisées pour les arguments:

ar = array (tableau réel ou complexe)
li = liste (ici de un ou deux réels)
r = réel **gn** = nom global ***ln*** = nom local
nb = nombre (réel ou complexe) ***c*** = nombre complexe
lr = nombre réel ou liste de un ou deux réels
2r = deux réels, ***2ar*** = deux tableaux, ***3ar***=.....

N.B: dans toutes les adresses ci-dessous, si un tableau "***ar***" utilisé est le contenu d'une variable, la modification du tableau ne s'effectue pas dans le contenu de la variable (il y a donc un **NEWOB**. A suivre....)

Adresse	Mnémonique	Adresse	Mnémonique
#1CA4Eh	size(ar)	#1D02Ch	->arry(r)
#1D040h	->arry(li)	#1D0ABh	obj->(ar)
#1D10Ch	rdm(ar,li)	#1D125h	rdm(gn,li)
#1D152h	rdm(ln,li)	#1D1EAh	con(li,nb)
#1D221h	con(ar,c)	#1D23Fh	con(gn,r)
#1D262h	con(gn,c)	#1D28Ah	con(ln,r)
#1D2ADh	con(ln,c)	#1D313h	idn(r)
#1D34Ah	idn(gn)	#1D36Dh	idn(ln)
#1D3BFh	trn(gn)	#1D3E2h	trn(ln)
#1D4DEh	put(ar,lr,nb)	#1D6B6h	puti(ar,lr,nb)
#1D86Bh	get(ar,lr)	#1D96Ch	geti(ar,lr)
#1DD3Dh	v->(ar)	#1DE7Fh	->v2(2r)
#1DEDBh	->v3(3r)	#35CAEh	con(ar,nb)
#35D35h	idn(ar)	#35DEBh	neg(ar)
#35E2Ch	rnd(ar,r)	#35EA9h	trunc(ar,r)
#35F30h	conj(ar)	#35F8Fh	re(ar)
#35FEEh	im(ar)	#36039h	r->c(2ar)
#360B6h	c->r(ar)	#36115h	+(2ar)
#36278h	-(2ar)	#362DCh	*(ar,nb)
#362DCh	*(nb,ar)	#363CCh	/(ar,nb)
#36435h	sq(ar)	#3643Fh	*(2ar)
#365ACh	rsd(3ar)	#366F6h	dot(2ar)
#36782h	cross(2ar)	#368E5h	rnrm(ar)
#368F9h	cnrm(ar)	#369CBh	abs(ar)
#36A2Ah	det(ar)	#36B0Bh	inv(ar)
#36B60h	/(2ar)	#37AFEh	r->c(ar)
#3811Fh	trn(ar)		

Voici quelques adresses utiles: (*s*, *sc*, *sl* sont des entiers-système: *sl* est un nombre de lignes, *sc* un nombre de colonnes)

Adresse	Mnémonique	Type	Commentaires
#35FA3h	dup;0con(ar)	Rpl	0con(ar) = tab. nul de même format
#36494h	dup2;*(2ar)	Rpl	2:M 1:N => 3:M 2:N 1:M*N
#03562h	length(ar)	LM	1:ar => 1:s (s = nombre d'élts)
#035A9h	LSsize(ar)	LM	1:ar => 1:{s} ou 1:{sl sc}
#1CDB1h	dup;type(ar)	Rpl	=> 3 si tab. réel, 4 si complexe
#62F9Dh	Ssize(ar)	Rpl	=> 2:sl, 1:sc ou 1:s
#63141h	over;length(ar)	Rpl	2:M 1:α => 3:M 2:α 1:s

◆ NEWOB ou pas NEWOB ?:

Imaginons que nous stockions une matrice dans une variable **'T'**.
Plaçons cette matrice sur la pile (en évaluant la variable **'T'**).
Je vous rappelle que le niveau 1 de la pile contient l'adresse de ce tableau en mémoire (et pas le tableau lui-même).

Si nous éxécutons la commande **TRN** (transposition), la matrice qui est sur la pile est modifiée (sauf cas exceptionnel d'une matrice carrée symétrique...).
Nous pouvons cependant constater que le contenu de la variable **'T'** n'a pas changé.

La conclusion qu'on peut en tirer est que la commande **TRN** a compris que le tableau était associé à une variable et elle a effectué un **NEWOB** (création d'une nouvelle copie de l'objet) avant de modifier cette copie du tableau initial.

Continuons notre expérience avec **TRN**. La matrice que nous venons de créer est donc un objet "*neuf*" qu'aucun autre objet ne *référence*.
Si nous dupliquons la matrice obtenue (ce qui revient en fait à dupliquer son adresse sur la pile), et si nous éxécutons à nouveau **TRN**, nous pouvons constater que la matrice au niveau 2 n'est pas affectée par les changements intervenant au niveau 1 (heureusement)...

La conclusion est ici que la HP48 a compris que l'adresse des tableaux aux niveaux 2 et 1 était la même. Elle en a déduit la nécessité de procéder à une nouvelle copie (**NEWOB**) de la matrice du niveau 1 avant que de la transformer par **TRN**.

Toutes ces observations peuvent être facilement vérifiées en utilisant la commande **MEM** avant et après chaque opération (mais en ayant pris soin de désactiver **LAST CMD**, **LAST STACK**, et **LAST ARG** qui sinon gèneraient l'interprétation de la mémoire disponible).

Les deux **NEWOB** que nous venons de voir sont de deux types différents. Pour signaler l'absence du premier, je dirai qu'il y a "***redirection***" (la modification du tableau venant du rappel d'une variable est "*redirigée*" sur cette variable).

Pour dire l'absence du second **NEWOB** (qui entraîne toujours l'absence du premier) je dirai que l'opération s'effectue "*sans aucun **NEWOB***".

Prenons à nouveau l'exemple de l'instruction **TRN**:

- Le SYSEVAL qui effectue **TRN** (*sans redirection*) est en **#3811Fh**.
- Le SYSEVAL qui effectue **TRN** (*avec redirection*) est en **#3814Ch**.
- Le SYSEVAL qui effectue **TRN** (*sans aucun **NEWOB***) est en **#38179h**.

◆ Expérience:

Plaçons la matrice **M** = [[1 2 3] [4 5 6]] dans la variable **'T'**.

- La séquence **T #3811Fh SYSEVAL** transpose la matrice au niveau 1 mais ne modifie pas la contenu de **'T'** (*pas de redirection*).
- La séquence **T #3814Ch SYSEVAL** transpose la matrice au niveau 1 et fait de même avec le contenu de **'T'** (*il y a redirection*).

  En fait c'est la matrice contenue dans la variable **'T'** qui est transposée, la modification de l'affichage au niveau 1 ne venant que confirmer la modification du contenu de **'T'**.
- La séquence **[[1 2 3][4 5 6]] ENTER ENTER** place la matrice **M** aux niveaux 1 et 2.

  La séquence **#38179h SYSEVAL** transpose alors simultanément les matrices au niveaux 2 et 1 (*aucun **NEWOB***)

◆ SYSEVALs SANS NEWOB:

Voici quelques adresses correspondant à des opérations simples sur une ou deux matrices, quand il y a ***redirection*** (mnémoniques débutant par le symbole @) ou quand il n'y a même aucun newob (mnémonique se terminant en plus par !).

Dans le cas où deux tableaux sont utilisés par l'opération, et qu'il y a *redirection*, cette *redirection* peut se faire sur l'adresse située au niveau 1 ou sur l'adresse située au niveau 2.

Exemple:

- Supposons que les variables **'T2'** et **'T1'** contiennent des tableaux de même format.
- La séquence **T2 T1 ENTER** place le tableau contenu dans **'T2'** au niveau 2 et le tableau contenu dans **'T1'** au niveau 1.
- La séquence **#361C9h SYSEVAL** (mnémonique **1@+(2ar)**) additionne les deux tableaux, et le résultat est *redirigé* sur l'adresse du niveau 1, c'est-à-dire dans la variable **'T1'** elle-même.

Adresse	Mnémonique	Type	Commentaires
#35CC2h	@con(ar,nb)	Rpl	redirigé
#35CCCh	@con(nb,ar)!	Rpl	aucun newob (NB position arguments)
#35D53h	@idn(ar)	Rpl	redirigé
#35D71h	@idn(ar)!	LM	aucun newob (ne teste pas mat. car.)
#35DFFh	@neg(ar)	Rpl	redirigé
#35E09h	@neg(ar)!	Rpl	aucun newob
#35F53h	@conj(ar)	Rpl	redirigé
#35F6Ch	@conj(ca)!	Rpl	redirigé (sur un tableau complexe)
#36B24h	@inv(ar)	Rpl	redirigé
#36B33h	@inv(s,ar)!	Rpl	aucun newob (s: ordre de la matrice)
#3814Ch	@trn(ar)	Rpl	redirigé
#38179h	@trn(ar)!	Rpl	aucun newob
#361C9h	1@+(2ar)	Rpl	redirigé sur adresse level1
#362AAh	1@-(2ar)	Rpl	redirigé sur adresse level1
#362C3h	2@-(2ar)	Rpl	redirigé sur adresse du level2
#36345h	1@*(nb,ar)	Rpl	redirigé sur adresse du level1
#363F9h	1@/(ar,nb)	Rpl	redirigé sur adresse du level1
#36458h	1@*(2ar)	Rpl	redirigé sur adresse du level1
#36476h	2@*(2ar)	Rpl	redirigé sur adresse du level2
#36B83h	1@/(2ar)	Rpl	redirigé sur adresse du level1
#36C7Ch	2@/(2ar)	Rpl	redirigé sur adresse du level2

◆ Opérations sur les lignes et les colonnes:

Il existe un certain nombre de SYSEVALs très intéressants qui s'appliquent aux lignes ou aux colonnes d'un même tableau.

Certains d'entre eux sont liés à la gestion de la matrice **ΣDAT** (mais peuvent être utilisés pour n'importe quel tableau).

Les adresses suivantes exigent que le tableau (ici désigné par "***ra***" c'est-à-dire "*real array*") soit une matrice réelle (et dans la pratique une matrice d'au moins deux lignes).

Adresse	Mnémonique	Type	Commentaires
#2CCDFh	MaxCol(s,ra)	LM	maximum de la colonne N°s
#2CCF8h	MinCol(s,ra)	LM	minimum de la colonne N°s
#2CD04h	ΣCol(s,ra)	LM	somme de la colonne N°s
#2CD13h	MeanCol(s,ra)	LM	moyenne de la colonne N°s
#2CD22h	VarCol(s,ra)	LM	variance de la colonne N°s
#2CD31h	SdevCol(s,ra)	LM	Ecart-type de la colonne N°s
#49569h	MinMaxCol(ra,s)	Rpl	2:ra 1:s => 2:MinCol 1:MaxCol

Les adresses suivantes, très intéressantes, méritent un peu plus de commentaires:

Adresse	Type	Commentaires
Mnémonique		
#2CCBAh nrows(ar)	LM	Donne le nombre de lignes de la matrice ou la taille du vecteur. Le résultat est un réel.
#2CF5Fh Chk2Col(2s,ar)	LM	Teste si s3 et s2 sont bien des N° de colonnes dans la matrice . Si oui, les objets restent sur la pile. Sinon Doerr "Invalid ΣPAR".
#3745Eh SwapRows(ar,2s)	LM	Echange lignes s2 et s1 du tableau. Le nouveau tableau reste au niveau 3. Les niveaux 1 et 2 sont inchangés. Adresse utilisable avec un vecteur (on échange ses coefficients).
#37500h SwapCols(ar,2s)	LM	Echange colonnes s2 et s1 du tableau. Le nouveau tableau reste niveau 3. Niveaux 1,2 inchangés

◆ Lire ou écrire dans un tableau:

On va voir maintenant que le monde des SYSEVALs offre bien des possibilités d'accéder à telle ou telle position, que ce soit pour y lire ou pour y écrire un élément.

Je signale que l'instruction **GET** (au niveau 2 un tableau et au niveau 1 une liste) autorise qu'il y ait une ou deux expressions dans la liste. Une telle possibilité n'est décrite nulle part dans les manuels de la HP48.

Par exemple: **[[1 2 3][4 5 6]] { '1+1' '3-2' } GET** donne le résultat 4 (comme l'aurait fait **{ 2 1 } GET**).

Voici différentes façons de lire dans un tableau. Ici **s** est un entier-système, **T** un tableau, et **T[s]** désigne le s-ème élément de T.

Rappel: "***ar***" = array, "***ra***" = real array

Adresse	Mnémonique	Type	Commentaires
#1D875h	Get(ar,lr)!	Rpl	Ici lr = réel ou liste de 1 ou 2 réels mais pas d'expressions.
#1D976h	Geti(ar,lr)!	Rpl	Voir ci-dessus pour lr.
#3558Eh	DupGet(ar,s)	LM	2:T, 1:s => 2:T, 1:T[s]
#355B8h	DupGet(ra,s)	LM	2:T, 1:s => 2:T, 1:T[s]
#355C8h	DupGet(ca,s)	LM	2:T, 1:s => 2:T, 1:T[s]
#355D8h	DupGetL(ar,s))	LM	2:T, 1:s => 2:T, 1:T[s] long
#35602h	DupGetL(ra,s)	LM	2:T, 1:s => 2:T, 1:T[s] réel long
#36A3Eh	get(ar,s)	LM	2:T, 1:s => T[s]

Quand on sait lire dans un tableau, il est normal de vouloir y écrire, alors voilà (mais avant tout un rappel de certaines notations):

lr = réel ou liste de 1 ou deux réels.
nb = nombre réel ou complexe
gn = nom global
ln = nom local
ra = real array
ca = complex array

Adresse	Mnémonique puis commentaires. (LM) si langage machine
#1D5BFh	PutSto(lm,ar,lr,nb) | place nb dans ar puis ar dans ln
#1D6DEh	@Puti(ar,lr,nb) | seul reste sur la pile 1:NextIndex | donc c'est redirigé.
#1D7A1h	PutiLsto(ln,ar,lr,nb) | Place "nb" en position "lr" dans tableau "ar" et le | tout va dans variable 'lm'. L'index est incrémenté. | Restent sur la pile: 2:lm 1:Newindex
#1DBB0h	put(nb,s,ar) |place "nb" en position "s" dans "ar"
#1DBC9h	@put(nb,s,ar) |idem, mais redirigé
#35628h	@put(ar,nb,s)! |(LM) place nb en position s. Aucun newob
#3566Fh	@put(ra,nb,s)! |(LM) idem avec tableau réel
#356F3h	@put(ca,nb,s)! |(LM) idem avec tableau complexe

L'adresse **#3566Fh** est le moyen le plus rapide de placer un élément dans un tableau (ici réel).

A titre de comparaison les deux programmes suivants effectuent 1000 fois la même opération: placer le réel 1999 en 3ème position dans la matrice identité d'ordre 5....

- **'EX1'**:
  « 3 5 IDN 1 1000 START OVER 1999 PUT NEXT SWAP DROP »

- **'EX2'**:
  ```
  « #400Dh SYSEVAL 5 IDN 1 1000 START
    1999 3 PICK #3566Fh SYSEVAL
    NEXT
  »
  ```

L'éxécution de '**EX1**' dure 31 secondes, celle de '**EX2**' 12 secondes. Encore '**EX2**' est-il ralenti par la présence de l'instruction SYSEVAL (on gagne encore du temps en remplaçant, dans le codage de '**EX2**' l'image de la séquence **#3566Fh SYSEVAL** par les 5 quartets **F6653**. Le temps de calcul est alors de 8 secondes)

◆ Création de tableaux:

Il est possible de former des tableaux, d'une façon plus globale, en utilisant des SYSEVALs bien choisis. Voici quelques exemples utiles.

Adresse	Mnémonique puis commentaires. (LM) si langage machine
#03442h	con(ls,nb) |(LM) ls=liste de un ou deux entiers-système
#19294h	fill(s,ar) |(LM) Remplit "ar" avec s objets pris aux niveaux ≥3. |L'objet du level 3, par ex, va en dernière position
#1D054h	->arry(li)! |interne à ->arry(li) en #1D040h. Il faut ici que la |liste contienne 1 ou 2 réels, et pas d'expression
#2C487h	lastv(ra,s) |2:ra 1:s => 2:ra 1:vct où vct est le vecteur des |"s" derniers élts de "ra" tableau réel
#2C4B4h	nulv(s) |=> vecteur nul de taille "s". s reste au niveau 2
#37E0Fh	rdm(ar,ls) |ls={shl, shc} ou {s}
#37E2Dh	@rdm(ar,ls) |ls={shl, shc} ou {s}. Redirigé.
#35C2Ch	map(proc,ar) |Evalue la procédure unaire "proc" sur |chaque élément du tableau."proc" doit transformer |un réel long en un autre réel long. Ce peut être |un programme, un nom...
#35C63h	map(proc,2ar) |Evalue procédure binaire "proc" sur les |éléments qui se correspondent dans les 2 tableaux. |Le résultat est un tableau placé au niveau 1. |la procédure doit s'appliquer à deux nombres au |format long et donner un résultat au format long
#44C31h	MatrixWriter! |Lance l'environnement MatrixWriter

Certaines des adresses précédentes méritent d'être illustrées d'exemples (dont'they?):

- Utilisation de "***con(ls,nb)***":

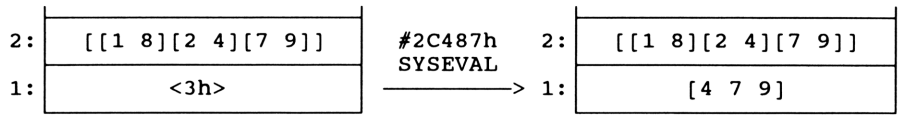
  2:{<2h> <3h>} 1:7 –#03442h SYSEVAL–> 1:[[7 7 7][7 7 7]]

  2:{<4h>} 1:(1,2) –#03442h SYSEVAL–> 1:[(1,2) (1,2) (1,2)]

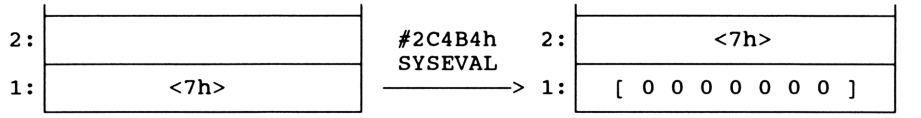
- Une illustration de "***fill(s,ar)***":

  7:13 6:11 5:9 4:1 3:7 2:<4h> 1:[[1 2 3][4 5 6]]
  ––––––––#19294h SYSEVAL––(fill(s,ar))––––>
  2:13 1:[[1 2 11][9 1 7]]

- Exemple avec "***lastv(ra,s)***":

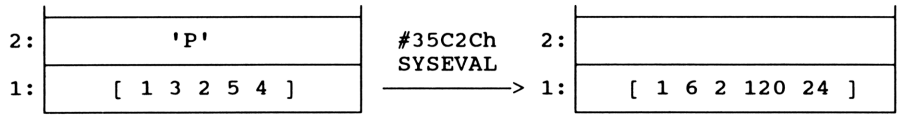


- Un exemple avec "***nulv(s)***":



- Une illustration de "***map(proc,ar)***":

On veut prendre la factorielle de chaque terme d'un tableau réel.
On utilise le programme:
« #2A5B0h SYSEVAL ! #2A5C1h SYSEVAL »
Le premier SYSEVAL transforme un *réel long* en réel "*normal*", le second revient au format réel long.
On place ce programme dans la variable '**P**'.



- Un exemple avec "***map(proc,2ar)***":

On veut, à partir de deux tableaux réels, construire le tableau formé des maximums terme à terme.
On utilise le programme **« #2A6DCh SYSEVAL »** (ce SYSEVAL prend deux réels longs et donne leur maximum, au format réel long).

◆ Tests divers sur les tableaux:

Voici des adresses qui vérifient la validité de certains arguments, à utiliser avant d'effectuer une opération sur un ou plusieurs tableaux.

Adresse	Mnémonique puis commentaires. (LM) si langage machine
#03685h	chk(ls,ar)? | (LM) "ls" = liste de 1 ou 2 entiers-sytème. Vérifie | si "ls" pointe sur un élément du tableau. Réponse: | 1:false si non, ou 2:s 1:true si oui | (s = entier-système = position dans le tableau)
#1DC7Dh	chk(ar,lr) | comme #03685h, mais "lr" = liste de 1 ou 2 réels. | erreur "Bad Argument Value" si liste non ok. | si ok, alors 2:ar 1:s, s est un entier-système | donnant la position dans le tableau.
#16973h	2D/3D(ar)? |répond false si "ar" n'est pas un vecteur réel de |dim 2 ou 3. Sinon répond true au niveau 1 et la |longueur (<2h> ou <3h>) au niveau 2
#357A8h	matrix(ar)? |(LM) teste si "ar" est une matrice ou un vecteur. |réponses: 3:sl 2:sc 1:true ou 2:s 1:false.
#37B7Bh	sqmat(ar)? |(LM) teste si le tableau est une matrice carrée. |réponses: 2:ar 1:size ou sinon "Invalid Dimension"
#37B9Eh	idsize(2ar)? |Teste si les 2 tableaux, qui restent sur la pile, |ont même format. Si non: erreur "Invalid Dimension"
#36868h	0ChkType(2ar) |2:ar2 1:ar1 => 3:ar2 2:ar1 1:x, avec x=0 si ar2,ar1 |sont réels et x=(0,0) si l'un au moins est complexe

LISTES ET "OBJETS-COMPOSÉS"

Le nombre de listes figurant dans la ROM de la HP48 est très important. Nous les avons vues dans un chapitre de la première partie.

Certaines de ces listes contiennent des objets "*amorphes*" (des réels, des entiers-systèmes, des chaînes de caractères).

D'autres ont un caractère plus "*éxécutable*" (on peut les évaluer comme on évaluerait un programme).

D'autres enfin contiennent les éléments constitutifs d'un menu ou d'une entrée de menu intégré à la HP48.

Toutes ces listes ont une adresse, et ces adresses sont autant de SYSEVALs susceptibles de vous être utiles (ou au moins d'exciter votre curiosité).

Nous verrons ici les adresses permettant d'effectuer toutes les opérations courantes (ou beaucoup moins courantes) sur les listes.

◆ Formations de listes par →LIST:

Commençons notre catalogue de SYSEVALs par quelques variations autour de l'instruction →**LIST**:

Adresse	Mnémonique	Type	Commentaires
#05459h	->list(s)	Rpl	Attention: L'instruction ->list(s) où s est un entier-système, ne vérifie pas qu'il y a au moins s objets sur la pile...
#23EEDh	1->list	Rpl	
#631A5h	->list(s-1)	Rpl	
#631B9h	2->list	Rpl	
#631CDh	3->list	Rpl	
#2FACEh	3->list;true;false		

◆ La notion d'objet composé:

Il est intéressant de constater que la représentation interne des *structures RPL*, des *expressions algébriques*, et des *objets-unité* est exactement la même, à l'adresse préfixe près (#8BA20h pour les expressions, #02D9Dh pour les Rpl, et #ADA20h pour les objets-unité), que la structure des *listes* (les *objets-unités* forment tout de même un cas à part, dans la mesure où les objets qui les composent sont dans un ordre qui ne doit rien au hasard).

La conséquence est qu'un certain nombre de SYSEVALs qui s'appliquent aux listes s'appliqueront aussi aux structures Rpl, aux expressions, et aux objets-unités.

Nous verrons que celà ouvre des horizons tout à fait surprenants....

LISTES ET "OBJETS-COMPOSÉS"

Notations:

Dans les mnémoniques qui suivent, j'ai utilisé l'abréviation "***cp***" pour désigner les ***objets-composés*** (*listes*, *structures Rpl*, *expressions*, et *objets-unité*).

Comme il a été dit ci-dessus, certains SYSEVALs s'appliquent d'une manière identique à tous les objets "*composés*". C'est le cas en particulier pour les opérations de lecture dans un *objet composé*.

En revanche, on doit être prudent avec les opérations qui modifient l'*objet composé* (écriture, insertion, concaténation): il peut y avoir des problèmes avec les **objets-unités** (allant jusqu'au "***Memory Clear***").

Quand l'objet "*composé*" est un **programme** saisi normalement par l'utilisateur, c'est-à-dire commençant par « et se terminant par », le premier objet de ce Rpl est « (et non pas la première instruction au sens où on l'entend habituellement), et que le dernier objet est ».

Quand l'objet "*composé*" est une **expression**, les SYSEVALs qui suivent peuvent conduire à une expression incorrecte, ce qui se traduit par l'objet ***'Invalid Expression'*** (que l'on peut encore évaluer); il n'y a donc en principe pas de risque de "*crash*" comme avec les objets-unités.

Je rappelle quelques notations utilisées dans le tableau ci-dessous:

s: entier-système ***li***: liste ***cp***: objet-composé
lr: réel r ou { r } ***2r***: 2 réels ***2s***: 2 entiers-système

Adresse	Mnémonique	Typ	Commentaires
#0521Fh	+(2cp)	Rpl	concatène les deux objets.
#052C6h	swap;+(_,cp)	Rpl	fait de obj1 le 1er élément de "cp"
#052FAh	+(cp,_)	Rpl	fait de obj1 le dernier élt de "cp"
#0E461h	insert(li,_,s)	Rpl	insère obj2 en position s de "li"
#1AC93h	+(_,cp)	Rpl	fait de obj2 le 1er élément de "cp"
#1D524h	put(li,lr,_)	Rpl	place obj1 en position "lr" dans "li"
#1D701h	puti(li,lr,_)	Rpl	place any1 dans li, à la manière PUTI
#1DC00h	put(_,s,li)	Rpl	place obj3 en position "s" dans "li"
#35491h	+IfNew(cp,_)	Rpl	ajoute obj1 s'il ne figure pas déjà
#1CAF0h	pos(cp,_)	Rpl	donne la position de obj1 dans "cp"
#644A3h	spos(cp,_)	Rpl	position exprimée en entier-système
#644BCh	spos(_,cp)	Rpl	idem, mais on cherche obj2 dans "cp"
#05153h	del1(cp)	LM	détruit premier objet de cp
#0E4DEh	del(li,s)	Rpl	enlève objet N° s de la liste
#05089h	get1(cp)	LM	donne le premier objet.
#056B6h	get(cp,s)?	LM	=> 2:obj, 1:true ou 1:false
#05821h	sub(cp,2s)	LM	sous-objet de position s2 à pos. s1
#1C8CFh	sub(cp,2r)	Rpl	idem mais position données par réels
#1D898h	get(cp,lr)	Rpl	lr = réel ou liste de un réel.
#1D8A2h	get(cp,lr)!	Rpl	idem, mais pas d'expression dans "lr"
#1D9BCh	geti(cp,lr)	Rpl	lecture dans le style de GETI
#510C1h	1get(li?)	Rpl	1get si obj1 est une liste sinon rien
#62B9Ch	get(cp,s)	Rpl	la séquence est: get(cp,s)?;drop
#62D1Dh	get(cp,s);dup	Rpl	idem, suivi de dup

Voici encore quelques adresses très utiles, certaines étant d'ailleurs assez "*hard*".

Adresse	Mnémonique	Type	Commentaires
#0567Bh	ssize(cp)	LM	longueur en entier-système
#1CA3Ah	size(cp)	Rpl	idem mais résultat réel
#63231h	dup;ssize(cp)	Rpl	permet de sauvegarder l'objet "cp"
#055B7h	={}?	LM	l'objet est-il une liste vide ?
#643EFh	InCp(_,cp)?		teste si obj2 est dans "cp" réponse: 1:true ou 2:obj2 1:false
#06F9Fh	eval(cp)	LM	évalue tous les objets-composés

Adresse	Mnémonique puis commentaires. (LM) si langage machine
#1D54Ch	PutSto(nm,li,lr,_) |place obj1 dans "li" puis li dans nm
#1D729h	PutiSto(nm,li,lr,_) |idem, mais incrémente index. |restent sur la pile: 2:nm 1:Newindex
#1945Ch	ChkType(li,s) |ici "li" est une liste d'objets dont le préfixe peut être reconnu par "prefix(_)" en #03C64h. Si tous les objets de "li" ne sont pas du type défini par "s", alors il y a erreur "Bad Argument Type"
#5E148h	prepend(_,s,cp) |Ajoute "s" objets, à partir du niveau 3, à l'objet "cp" qui subit un newob au départ. Le résultat est l'objet ainsi complété. Exemple: 10,20,30,40,<3h>,{A B} => 10,{20 30 40 A B}
#5E17Fh	prepend(cp,_) |Comme ci-dessus, mais pour un seul objet. Pas de Newob
#64426h	map(cp,_,test) |fait agir le programme "test", qui prend 2 arguments, et qui renvoie un résultat booléen, sur chaque élt de "li" et sur obj2, jusqu'à obtenir "true". Résulat: entier-système donnant position de l'élément de "li" ayant donné true, et <0h> sinon
#6448Ah	pos(ls,s) |donne la position de "s" dans "ls", qui doit être une liste d'entiers-système. Donne <0h> si "s" non trouvé
#644D0h	1st(cp,test)? |éxécute le programme "test", qui prend un argument et renvoie un booléen, sur chaque élément de "cp", jusqu'à ce que "true" soit obtenu. Le résultat est alors True au niveau 1, et l'objet de "cp" au niveau 2. Sinon le seul résultat est false au niveau 1.
#6480Bh	NextObj(cp,s)? |(LM) "s" est un entier-système indiquant une position dans "cp", en termes de quartets depuis le début de l'objet composé. Les deux réponses possibles sont: 4:li 3:s' 2:obj 1:true, où obj est l'objet trouvé à partir du quartet N°s, et où s' est la position qui suit l'objet obtenu. 2:li 1:false, si aucun objet n'a été trouvé à partir de la position s. Si l'objet trouvé est une liste, c'est cette liste qui apparaît sur la pile, le compteur s' pointant sur l'objet venant après cette sous-liste. |Le 1er objet est obtenu avec s=<0h> ou <5h>.

◆ "Casser" ou créer des objets composés:

Nous allons voir comment *rompre* un objet composé en ses différents éléments, ou on contraire comment former (ou reformer) un objet composé.

Voici les adresses avec un minimum d'explications, et on verra plus loin quelques exemples...

Adresse	Mnémonique	Type	Commentaires
#054AFh	breaks(cp)	LM	casse "cp". => taille entier syst.
#1C973h	breakr(cp)	Rpl	casse un "cp". Donne taille réelle
#62B88h	break;drop	Rpl	┐
#62C41h	break;dup	Rpl	├ Ici, break est un abrégé de
#631E1h	dup;break	Rpl	│ breaks(cp)
#631F5h	swap;break	Rpl	┘
#05331h	build(_,2s)	LM	construit un objet composé, de taille s2, de prologue s1
#05445h	->rpl(s)	Rpl	crée une structure Rpl de s objets
#632D1h	->rpl(s);eval	Rpl	idem, puis évalue
#63FCEh	->rpl1	Rpl	fait de obj1 un Rpl (si ce n'est pas déjà un Rpl)
#63FE7h	->rpl1!	Rpl	fait de obj1 un Rpl
#63FFBh	->rpl2	Rpl	combine obj2, obj1 en un Rpl
#0546Dh	->expr(s)	Rpl	crée une expression de s objets
#05481h	->unit(s)	Rpl	crée un objet-unité de s objets
#5E652h	->expr1	Rpl	fait de obj1 une expression (si ce n'en est pas déjà une) donne 'UNKNOWN' si impossible.
#5E661h	->expr1!	Rpl	fait de obj1 une expression.
#5A01Dh	->expL2	Rpl	idem, mais au level 2

Remarques:

Les instructions "***→rpl(s)***", "***→list(s)***", "***→expr(s)***" et "***→unit(s)***" appellent en fait l'instruction "***build(_,2s)***", en transmettant au niveau 1 le prologue: **s=<2D9Dh>** pour les *Rpl*, **s=<2A74h>** pour les *listes*, ***s=<2AB8h>*** pour les *expressions*, et **s=<2ADAh>** pour les *objets-unités*.

N'utilisez pas ***→expr1*** (ou les intructions ***→expr1!***, ***→expL2***) pour essayer de former une expression constituée d'un seul objet si cet objet est un *Rpl* (car on va droit vers un "*Try to recover memory*").

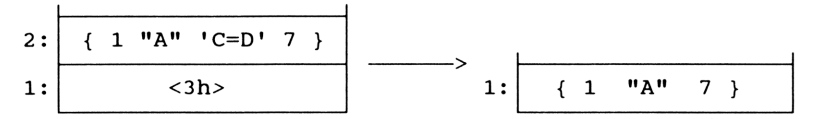
◆ EXEMPLES ET RÉCRÉATION:

Nous allons voir des exemples utilisant les SYSEVALs précédents.

Certains exemples sont sérieux, d'autres sont plus fantaisistes, et montrent qu'on peut se livrer, sur les "*objets-composés*" (notamment sur les *Rpl* et les *expressions*) à quelques manipulations génétiques non prévues par le comité d'éthique de **Hewlett-Packard**.

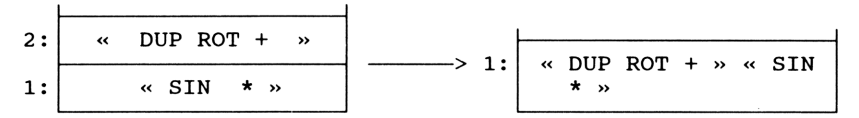
▸ #0E4DEh SYSEVAL (Mnémonique "del(li,s)")

Il s'agit ici d'enlever l'objet N°s de la liste.

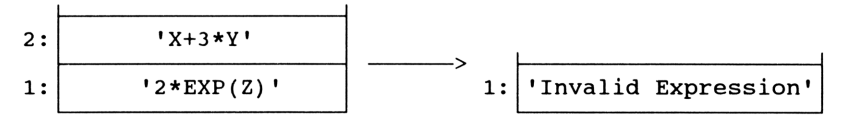


▸ #0521Fh SYSEVAL (Mnémonique: " + (2cp)")

On effectue ici la concaténation de deux objets composés. Pour les listes, c'est l'opération + habituelle, mais pour le reste:

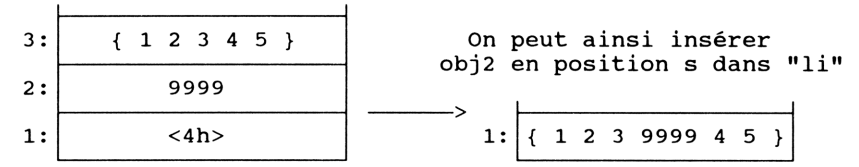


L'objet obtenu est un *Rpl* éxécutant en séquence les 2 programmes dont il est issu.



On pourrait penser que l'objet obtenu est inutilisable. Que nenni!! Vous pouvez évaluer cette expression: celà revient à évaluer successivement les expressions **'X+3*Y'** et **'2*EXP(Z)'**.

▸ #0E461h SYSEVAL (Mnémonique: "insert(li,_,s)")



On notera la différence avec la commande **REPL** de la HP48 qui place un objet ou une liste dans une autre liste, avec "*écrasement*" des objets précedents.

- **#1CAF0h SYSEVAL (Mnémonique "pos(cp,_)")**
  "***pos(cp,_)***" donne la position de ***obj1*** dans "***cp***", exprimée sous la forme d'un réel.

```
2: | 1.2345_m/s |
1: | "m"       |  ------>  1: | 2 |
```

Pour comprendre ce résultat (tout à fait correct), on se reportera au paragraphe décrivant les objets-unité.

```
2: | « DUP ROT 15 * » |
1: | 15               |  ------>  1: | 4 |
```

Ici on voit bien que l'objet « correspond en fait à la première instruction de cette structure *Rpl*.

- **#35491h SYSEVAL (Mnémonique " + IfNew(cp,_)")**
  Cette adresse permet d'ajouter ***obj1*** à "***cp***" s'il n'y figure pas déjà.

```
2: | { 1 2 3 4 5 } |   Ici "2" n'est pas ajouté à la liste
1: | 2             |      car il y figure déjà.
                     ------>  1: | { 1 2 3 4 5 } |
```

```
2: | { 1 2 3 4 5 } |   Ici "0" est ajouté à la liste car
1: | 0             |      il n'y figurait pas encore.
                     ------>  1: | { 1 2 3 4 5 0 } |
```

- **#05153 SYSEVAL (Mnémonique "del1(cp)")**
  Cette adresse permet de détruire le premier objet de "***cp***"

Appliquons successivement ce SYSEVAL au même ***Rpl***.

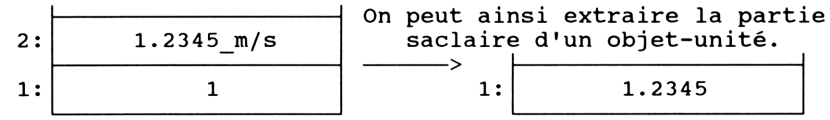
```
1: | « ROT DUP + » |  -->  1: | DUP ROT + » |  -->  1: | ROT + » |  -->  1: | + » |

1: | » |  -->  1: |  |
```

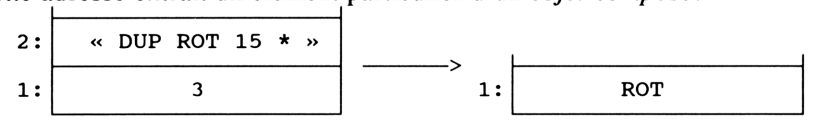
Vous venez de suivre la disparition d'un pauvre *Rpl* (en fait, il n'est pas tout à fait mort: il reste sur la pile, sous forme de *Rpl vide*...)

N.B: ce SYSEVAL est sans effet sur le Rpl vide, ou sur la liste vide. Evitez cependant l'expression vide (c'est un conseil d'ami, mais vous êtes libre), et n'appliquez pas cette adresse à un objet-unité.

▸ **#05089h SYSEVAL (Mnémonique "get1(cp)")**
Cette adresse extrait le 1er élément d'un *objet-composé*.

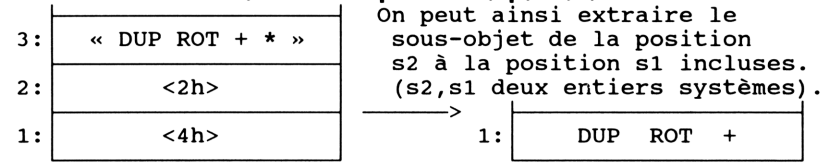


▸ **#1D898h SYSEVAL (Mnémonique "get(cp,lr)")**
Cette adresse extrait un élément particulier d'un *objet-composé*.



(rappelons que « est le premier objet du *Rpl*)

▸ **#05821h SYSEVAL (Mnémonique "sub(cp,2s)")**



Le résultat est ici un *Rpl* formé des 3 instructions **DUP**, **ROT**, +

▸ **#1945Ch SYSEVAL (Mnémonique "ChkType(li,s)")**
L'instruction **#03C64h SYSEVAL** (Mnémonique "***prefix(_)***") donne l'adresse-préfixe de l'objet placé au niveau 1, sous la forme d'un entier-système.
Par exemple **12345 #03C64h SYSEVAL** donne **<2933h>**, car l'adresse préfixe des réels est **#02933h**.
Les objets reconnus sont: les réels, les noms, les expressions, les listes, les unités, les programmes Rpl, les répertoires (<0h> pour autres objets).

Le programme suivant examine (avec la réserve ci-dessus), si les objets de la liste "***l2***" sont du type défini par "***obj1***".
'**CHKT**': (Checksum: #7813h; taille: 49.5 octets)
« #03C64h SYSEVAL #1945Ch SYSEVAL »

Exemple:
2:{(1,1) 3 (4,5)} 1:(0,0) ==> Erreur "*Bad Argument Type*"
2:{(1,1) (4,5) (3,2)} 1:(0,0) ==> 1:{(1,1) (4,5) (3,2)}

▸ **#644D0h SYSEVAL (Mnémonique "1st(cp,test)?")**

On souhaite savoir si l'un des objets d'un *objet-composé* satisfait à un test donné, et quel est alors le 1er élément à satisfaire à ce test.

Par exemple: étant donné une liste de réels, y-a-t-il un réel < 0, et si oui quel est le premier réel négatif de la liste?

On utilise l'instruction **#2A738h SYSEVAL** (mnémonique "***<0(r)?***") qui attend un réel au niveau 1 et qui donne le booléen ***True*** si ce réel est < 0, et ***False*** sinon.

Auparavant, on vérifie que la liste est bien formée de nombres réels en utilisant la séquence **0 CHKT** (voir le programme précédant)

Celà donne le programme:
'**NEG?**' (Cheksum: #C5AEh; taille: 72 octets)

```
« 0 CHKT
  « #2A738h SYSEVAL »
  #644D0h SYSEVAL
»
```

Exemple:

1:{ 1 5 "8" 6 } ---[NEG?]---> Erreur "*Bad Argument Type*"
1:{ 1 5 8 4 6 } ---[NEG?]---> 1:External (false)
1:{ 1 5 8 -4 6 } ---[NEG?]---> 2:-4 1:External (true)

▸ **#64426h SYSEVAL (Mnémonique "map(cp,_,test)")**

Il s'agit ici de faire agir un opérateur binaire "***test***", renvoyant un booléen, sur chaque objet de "***cp***", et sur "***obj2***", jusqu'à ce le test renvoie la valeur *true*, où jusqu'à la fin de l'*objet composé*.

Le résultat de tout celà est l'entier-système donnant la position correspondant au premier "*succès*", et **<0h>** sinon.

Exemple:

On part d'une liste de réels "***li***", et d'un réel donné α.
On veut savoir si l'un des réels de la liste est supérieur à α, et connaître alors sa position dans la liste.
On va utiliser la séquence **#2A88Ah SYSEVAL** qui attend 2 réels ***r2*** et ***r1***, et qui renvoie le résultat *True* si ***r2>r1*** , et le résultat *False* sinon.
On teste encore que la liste n'est formée que de nombres réels avec la séquence **0 CHKT** (voir page précédente).

Celà donne le programme:
'**SUP?**' (Cheksum: #B966h; taille: 77 octets)

```
« OVER 0 CHKT DROP « #2A88Ah SYSEVAL »
  #64426h SYSEVAL
»
```

On obtient par exemple:

```
2:{ 1 5 6 8 3 4 } 1:7 ---[SUP?]---> 1:<4h>
2:{ 1 5 6 8 3 4 } 1:4 ---[SUP?]---> 1:<2h>
2:{ 1 5 6 8 3 4 } 1:8 ---[SUP?]---> 1:<0h>
```

▸ **#05445h SYSEVAL (Mnémonique "->rpl(s)")**

Il s'agit ici de grouper les s premiers objets de la pile en une unique *structure Rpl* ("*s*" est ici un entier-système qui ne doit pas être compté parmi ces objets à grouper).
On peut à priori collecter ainsi des objets quelconques.

Exemple:

```
4:ROT 3:* 2:+ 1:<3h> --#05445h SYSEVAL-> 1:ROT * +
```

L'objet obtenu ici est équivalent au programe « **ROT * +** », mais sans les caractères « et » ...

▸ **#0546Dh SYSEVAL (Mnémonique "->expr(s)")**

On peut créer ainsi des expressions licites ou illicites (c'est ça qui est drôle). Quelques exemples:

```
7,8,9,<3h> --#0546Dh SYSEVAL-> 1:'Invalid Expression'
```

Si on évalue cette 'expression', on obtient 3:7, 2:8, 1:9

```
6:{ A B C } 5:OBJ-> 4:ROLLD 3:3 2:->LIST 1:<5h>
  --#0546Dh SYSEVAL-> 1:'Invalid Expression'
```

Si on évalue cette '*expression*', on obtient **1:{ C A B }**. On a donc ainsi "*caché*" un programme sous la forme d'une expression illicite.

▸ **#5E661h SYSEVAL (Mnémonique "->expr1!")**

C'est un peu un cas particulier du SYSEVAL précédent. On forme une expression dont l'unique élément est l'objet du niveau 1 (même si celui-ci est déjà une expression).

Exemple:

```
1:'1+2'-#5E661h SYSEVAL-> 1:'1+2'-#5E661h SYSEVAL-> 1:'1+2'
```

On pourrait penser n'avoir pas modifié l'expression initiale. Il suffit de l'évaluer par **EVAL** pour voir que ...

```
1:'1+2' --EVAL--> 1:'1+2' --EVAL--> 1:'1+2' --EVAL--> 1:3
```

L'expression '**1+2**' avait donc été "*enfermée*" deux fois...

▸ **#054AFh SYSEVAL (Mnémonique "breaks(cp)")**

Cette adresse permet de "*casser*" en ses différents éléments un *objet-composé*. On ne la confondra pas avec l'instruction **OBJ→** qui est plus *soft*.

```
1:1.2345_m/s
  --#054AFh SYSEVAL->  6:1.2345 5:"m" 4:"s" 3:{} 2:{} 1:<5h>

1:« DUP ROT + / »
  --#054AFh SYSEVAL->  7:« 6:DUP 5:ROT 4:+ 3:/ 2:» 1:<6h>

1:'X+3*Y'
  --#054AFh SYSEVAL->  6:'X' 5:3 4:'Y' 3:* 2:+ 1:<5h>

1:{ "A" "B" "C" }
  --#054AFh SYSEVAL->  4:"A" 3:"B" 2:"C" 1:<3h>
```

◆ ON SOLDE !

Il reste quelques adresses sur les listes ou sur les objets-composés. Leur seul intérêt est d'avoir assez peu d'intérêt (sauf la dernière peut-être). Je ne les commenterai donc qu'un minimum. Mais elles auront droit elles aussi à un joli tableau.

Adresse	Mnémonique puis commentaires.
#10720h	{};swap;1+(s)
#10824h	+(2s);1+s;{};swap
#10851h	{};swap;2+(s)
#2ED74h	in{1,2,3}(r)?
	|le réel r est-il dans la liste {1 2 3} ?
	|on effectue d'abord ABS et IP sur le réel.
#41328h	In{"0""1""2""&"}?
	|l'objet au niveau 1 est-t-il dans cette liste ?
#41E8Ch	49->list(_) |obj1 => liste de 49 obj1
#41ECDh	6->list(_) |obj1 => liste de 6 obj1
#4889Dh	drop(2);+(2cp)
#488B1h	drop;+(cp,_)
#4EA28h	Chkr=>s(s,r)
	|Transforme r en l'un entier-système s1 le plus
	|proche si 0≤s1<s. Sinon laisse r.
#1DB6Fh	break;<>1?:BAV!
	|erreur "Bad Argument Value" si taille <> 1
#51519h	break;<>2?:BAV! |voir ci-dessus
#636A0h	break;=1(s)? |résultat booléen
#64127h	AdrIn(cp,_)?
	| (LM) Répond true si l'adresse de obj1 figure dans
	| le contenu de "cp", ou si obj1 est égal à cp.
	| sinon répond false.

CHAINES DE CARACTERES

Le nombre de chaînes de caractères présentes en ROM de la HP48 est très important. Beaucoup d'entre elles sont des de labels de menu. Vous seront-elles utiles ?

Les adresses qui suivent sont plus "*opérationnelles*" car elles autorisent de nombreuses manipulations.

Il convient tout d'abord de noter la grande analogie, pour ce qui est de la représentation interne, entre les *chaînes de caractères* et les *entiers binaires*:

Ces deux objets sont en effet codés de la manière suivante:

- Une adresse-préfixe
- La longueur de l'objet (sans compter l'adresse-préfixe), sur cinq quartets. On compte ici les quartets (chaque caractère d'une chaîne est codé sur deux quartets).
- Les différents quartets constituant le corps de l'objet.

Les *entiers binaires* peuvent donc être interprétés comme des chaînes de chiffres hexadécimaux (rappel: le nombre de chiffres hexadécimaux d'un entier binaire n'est pas nécessairement égal à 16. Il peut varier de 0 à plusieurs milliers).

Cette similitude a pour effet que certaines adresses s'appliquent aussi bien aux *chaînes* qu'aux *entiers binaires*.

Les SYSEVALs qui ont été donnés dans le paragraphe consacré aux entiers binaires (et plus précisément les adresses **#53...h** et **#54...h** renvoient toujours des entiers binaires au format habituel, c'est-à-dire 16 chiffres hexadécimaux en mémoire).

Ici on trouvera des SYSEVALs pouvant utiliser et/ou renvoyer des entiers binaires de longueur quelconque.

J'indiquerai les SYSEVALs décrits ici qui s'appliquent aux *entiers binaires*. Je n'ai pas forcément détaillé toutes les combinaisons possibles, et je vous laisse le soin de faire les expériences que votre curiosité ne manquera pas d'imaginer.

Commençons par décrire les adresses correspondant à l'opération de concaténation de deux chaînes, ou d'une chaîne et d'un "*character*".

Adresse	Mnémonique	Type	Commentaires
#05193h	+(2st)!	LM	contrôle "Out of Memory"
#622EFh	swap;+(2st)!	Rpl	st2, st1 => st1+st2
#62376h	+(2st)	Rpl	remédie si possible à Out of M
#63F6Ah	+(2st)!;swap	Rpl	α, st2, st1 => st2+st1, α
#62F2Fh	+(2st);swap	Rpl	α, st2, st1 => st2+st1, α
#6910Ch	over++(2st)	Rpl	ex: "A","B" => "ABA"
#0525Bh	prep(st,ch)	LM	ajoute le "Character" en tête
#052EEh	+(st,ch)	LM	ajoute le "Character" à la fin
#1ACA7h	+(st,_)	Rpl	obj1 d'abord transformé en chaîne
#1ACBBh	+(_,st)	Rpl	obj2 d'abord transformé en chaîne

Puis viennent quelques variations (plutôt mineures) sur le même thème (ajouter quelque chose à la fin ou au début d'une chaîne):

Adresse	Mnémonique	Type	Commentaires
#35051h	+(sp,st)	Rpl	ajoute un espace en tête
#39F23h	+(3sp,st)	Rpl	ajoute 3 espaces en tête
#61C1Ch	add0(st,s)	LM	ajoute s chr(0) à la fin
#62BB0h	+(st,sp)	Rpl	ajoute un espace à la fin
#63191h	+(st,RC)	Rpl	ajoute un chr(10) (Newline)
#2E4DCh	+(st,▪RC)	Rpl	ajoute chr(13) et chr(10)
#322A1h	dup;+(2st)	Rpl	ex: "ABC" => "ABCABC"
#340CEh	+("+",st)	Rpl	ex: "123" => "+123"
#39F04h	+(1:sp,st)	Rpl	ex: "Hello" => "1: Hello"
#3A19Dh	+(...,st)	Rpl	ajoute en tête un chr(31)
#40693h	"()";+(2st)	Rpl	ex: "FONC" => "FONC()"

Voici les SYSEVALs permettant d'effectuer des opérations logiques sur les chaînes. Ils s'appliquent également aux entiers binaires.

Adresse	Mnémonique	Type	Commentaires
#18873h	and(2st)	Rpl	pour ces 3 adresses, il y a erreur
#18887h	or(2st)	Rpl	"Bad Argument Value" si les chaînes
#1889Bh	xor(2st)	Rpl	ont des longueurs différentes
#188E6h	and(2st)!	LM	appelé par adresses ci-dessus.
#188F5h	or(2st)!	LM	A n'utiliser que si les 2 chaînes
#18904h	xor(2st)!	LM	ont la même longueur!
#188D2h	not(st)	Rpl	avec newob
#18961h	not(st)!	LM	sans newob

Voici les SYSEVALs sur le thème "*longueur d'une chaîne*". Ils s'appliquent aux entiers binaires. Ici **s** représente un *entier-système*.

Suivent les quatres SYSEVAls permettant de comparer deux chaines (dans l'ordre lexicographique). Le résultat est alors le réel 0 ou 1.

Adresse	Mnémonique	Type	Commentaires
#05616h	quart(st)	LM	1:st => 1:s (s=nbre de quartets)
#05636h	ssize(st)	LM	1:st => 1:s (s=nbre de caractères)
#05622h	over;ssize(st)	LM	
#627BBh	dup;ssize(st)	LM	
#1782Eh	ssize+(st,st)	Rpl	st2,st1 => st2,st1,ssize(st2+st1)
#1CA26h	size(st)	Rpl	En fait: ssize(st);s=>r
#188AFh	ChkSize(2st)	Rpl	"Bad Argument Value" si tailles <> sinon st2,st1 => newob(st1),st2
#1420Ah	>(2st)	LM	Applicable aux entiers-binaires,
#142A6h	<(2st)	Rpl	mais sans intérêt (les modes de
#142BAh	≥(2st)	Rpl	comparaison numériques et lexi-
#142E2h	≤(2st)	Rpl	-cographiques sont différents)

Venons-en aux différentes manières de trouver la position d'une sous-chaîne, ou d'extraire une sous-chaîne d'une chaîne donnée.

Adresse	Mnémonique	Type	Commentaires
#1CAD7h	pos(2st)	Rpl	ex: "ABCDECDF", "CD" => 3
#15EF6h	spos(2st)	Rpl	ex: "ABCDECDF", "CD" => <3h>
#645B1h	spos(2st,s)	LM	recherche débute au car. n°s de st3 ex: "ABCDECDF", "CD" , <2h> => <3h> ex: "ABCDECDF", "CD" , <4h> => <6h> ex: "ABCDECDF", "CD" , <7h> => <0h>
#645BDh	rspos(2t,s)	LM	"rspos" = "reverse system pos" recherche en amont, depuis car. n°s ex: "ABCDECDF", "CD" , <2h> => <0h> ex: "ABCDECDF", "CD" , <4h> => <3h> ex: "ABCDECDF", "CD" , <7h> => <6h>
#05733h	sub(st,2s)	LM	ex: "ABCDEFGH",<2h>,<4h> => "BCD"
#1C8BBh	sub(st,2r)	Rpl	ex: "ABCDEFGH", 2, 4 => "BCD"
#62D6Dh	sub(st,2s);swap	Rpl	
#63245h	sub(st,s2,s1-1)	Rpl	ex: "ABCDEFGH",<2h>,<4h> => "BC"
#63259h	sub(st,1,s-1)	Rpl	ex: "ABCDEFGH",<5h> => "ABCD"
#6326Dh	sub(st,s,∞)	Rpl	ex: "ABCDEFGH",<5h> => "EFGH"
#63281h	sub(st,s+1,∞)	Rpl	ex: "ABCDEFGH",<5h> => "FGH"

Les adresses suivantes, plus "*atypiques*", n'en sont pas moins intéressantes.

Adresse	Mnémonique	Type	Commentaires
#0516Ch	del1(st)	Rpl	Ote car n° 1. Chaîne "" inchangée.
#12770h	trunc22(st)	Rpl	Si longueur(st) > 22 caractères, tronque à 21 car. et ajoute ...
#127A7h	CutRC(st)	Rpl	Sépare st au 1er chr(10) rencontré st=st1+RC+st2 => st2,st1 st2="" si pas de RC recontré
#140F1h	chr(r)	Rpl	Forme interne de l'instruction CHR
#1410Fh	num(st)	Rpl	Forme interne de l'instruction NUM
#45676h	space(s)		=> Chaîne de s espaces.
#30805h	snum(st,s)	LM	Code du caractère n°s de la chaîne ex: "123456", <4h> => <34h>
#451E4h	trunc+..(st,s)	Rpl	Garde caractères n°1 à n°s-1 de st et ajoute le caractère ...
#45522h	3LastChr(st)	Rpl	Chaîne des 3 derniers caractères
#4FAF7h	repl(st,r,st)	Rpl	Place st1 dans st3 à partir du car n°r (forme interne de REPL) "123456",5,"ABC" => "1234ABC"
#6D1BCh	SpStSp(st)	Rpl	Entoure une chaîne par 2 espaces
#0556Fh	""=?	LM	Réponse par true ou false
#04D3Eh	drop;""	LM	
#04D57h	drop2;""	LM	Quelques variations autour
#1613Fh	"";newob	Rpl	de la chaîne vide
#45469h	3dropn;""	Rpl	(qu'on trouve à l'adresse #55DFh)
#4A0D7h	"";false	Rpl	
#62D59h	"";swap	Rpl	
#63209h	dup;""=?	Rpl	Réponse par true ou false

◆ Transformations d'objets en chaîne:

Les multiples variantes internes de l'instruction **→STR** seront examinées dans le chapitre consacré aux conversions de type par SYSEVALs. Nous terminons par quelques adresses qui méritent des explications un peu plus détaillées qu'à l'habitude.

Adresse	Mnémonique puis commentaires. (LM) si langage machine
#15424h	DoSt:(nm,st) | Par ex, 2:'ABC' 1:"Hello", donne "ABC: coucou". | Attention, le nom est tronqué au 8ème caractère.
#15955h	DoSt:22(s,_) | Avec, par ex, 2:<Ah> 1:'ABC', crée la chaîne de 22 | caractères "10: 'ABC'"
#1596Eh	DoSt:(s,_,s) | Comme ci-dessus, mais longueur de la chaîne = s1+3
#15CBBh	"type(_) | Type de obj1 sous forme de chaîne (ex: "Program")
#2DF83h	Conc:RC(2st) | Ex: "DEFG", "ABC" => "ABC:▪DEFG" (▪=NewLine)
#47798h	+RJust9(st,r) | Ajoute le réel à la chaîne en le justifiant à droite | sur 9 car. Ajoute un espace. | Ex: "ABC", 1 => "ABC 1 " | Ex: "ABC", 123456789000 => "ABC 1.235E11 "
#47D76h	->StrRcl(nm) | Transforme en chaîne un nom et son contenu | (ou une indication sur ce contenu) | Ex: 'A' => " A: dir" si 'A' est un directory
#47E2Fh	->Str:(nm,ex) | Concatène un nom et une expression, séparés par ":" | Ex: 'ABC', 'SIN(X+Y)' => " ABC: 'SIN(X+Y)'"
#47E66h	->Str:(nm,li) | Comme ci-dessus, mais avec une liste
#47F56h	->Str(nm,ar) | Concatène le nom et le format du tableau. | Ex: 'ABC' [[1 2 3][4 5 6]] => " ABC:[2x3]".
#47FF0h	DoStr:(nm,st) | Concatène 1 nom et 1 chaîne, séparé par un : | Ex: 'ABC', "Hello" => " ABC:Hello"
#49709h	DoStr:(nm,_) | Comme adresses ci-dessus, mais obj1 est qcq, et | c'est son contenu complet qui est ajouté, dans la | limite de la taille du buffer de décompilation.
#42C24h	ChkNoChr0(st)! S'il y a un chr(0), doerror #102h
#42C3Dh	dup;NoChr0(st)? Répond True s'il n' y a pas de chr(0).
#42C5Bh	dup;BeforeRC(st)? S'il y a un premier chr(10) en position s, donne <1h>,s,true; sinon donne <1h>,<0h>,false
#42C74h	dup2;SposRC(st,s) Donne position du premier chr(10) trouvé à partir de la position s1 (<0h> si pas trouvé)

CONVERSIONS DE TYPE

Les adresses de ce chapitre permettent de passer d'un type d'objet à un autre (par exemple de transformer un *réel* en *réel long*).

Pour traduire ces transformations, le symbole **=>** sera utilisé dans les mnémoniques (par exemple **r=>s** pour la transformation d'un *réel* en un *entier-système*).

Rappelons une fois encore la signification des abréviations:

st:chaîne	***ch***:character	***s***:entier-système	***b***:entier binaire
lm: nom local	***gn***: nom global	***nm***: nom	***uo***: objet-unité
r:réel	***rr***:réel long	***c***:complexe	***cc***:complexe long

?:booléen (true ou false)

Adresse	Mnémonique	Type	Commentaires
#050EDh	st=>ch	LM	le premier caractère de la chaîne.
#6475Ch	ch=>st	Rpl	Character => chaîne de longueur 1
#059CCh	s=>b	LM	ex: <205C4h> => #205C4h
#05A03h	b=>s	LM	ex: #205C4h => <#205C4h>
#05A51h	ch=>s	LM	ou encore: s => s modulo 256
#05A75h	s=>ch	LM	entier système => character de code s
#05AEDh	nm=>ln	Rpl	force un nom à être local
#05B01h	nm=>gn	Rpl	force un nom à être global
#05B15h	st=>gn	LM	permet de créer des noms exotiques!!
#05BE9h	nm=>st!	LM	sans les '. Ex: 'ABC' => "ABC"
#57225h	nm=>st	Rpl	appelle le précédent
#0F218h	uo=>st	Rpl	1.25_m/s => "1.25_m/s"
#167E4h	s=>st	LM	résultat en décimal: <159h> => "345"
#454D2h	s=>st	Rpl	comme ci-dessus
#18CD7h	r=>s	LM	d'abord "round" et "abs" sur r ex: -1.4 => <1h>; -1.5 => <2h> ex: -345 => <159h>; 345 => <159h>
#18CEAh	r+=>s	LM	résultat=<0h> si réel≤0
#2EC11h	ip(r)=>s	Rpl	en fait: ip(r) puis r=>s ex: -1.4 => <1h>; -1.9 => <1h>
#18DBFh	s=>r	LM	ex: <159h> => 345
#28AC9h	nm=>uo?	Rpl	'ABC' => 1_ABC, true (évent. erreur "Invalid Unit")
#2A5B0h	rr=>r	LM	réel long => réel
#2A5C1h	r=>rr	LM	réel => réel long
#519CBh	c=>2rr	Rpl	1:(x,y) => 2:xx, 1:yy
#51A07h	2rr=>c	LM	réciproque du précédent
#519F8h	cc=>c	LM	complexe long => complexe
#05C8Ah	2rr=>cc	LM	2:xx, 1:yy => 1:(xx,yy)
#51A37h	r=>c	LM	ex: 5 => (5,0)
#5380Eh	?=>r	LM	false => 0, et true => 1
#5435Dh	b=>r	LM	#12345d=#3039h => 12345
#543F9h	r=>b	LM	12345 => #12345d=#3039h
#63B96h	s=>rr	Rpl	en fait s=>r puis r=>rr

CONVERSIONS DE TYPE

Remarque: l'adresse **#05B15h** (Mnémonique ***st=>gn***) transforme une chaîne de caractères (quelconques) en un nom global. On peut ainsi créer des noms de variables interdits !

Après ce catalogue de conversions, voici des SYSEVALs réalisant des transformations parfois plus élaborées.

Certaines adresses (les dernières) sont d'une utilité douteuse, mais je n'ai pas eu le courage de les abandonner. On ne devrait pas faire autant de sentiments avec les SYSEVALs.

Adresse	Mnémonique	Type	Commentaires
#10D14h	r-=>s	Rpl	-256≤r<0 donne s=256+r -1=><FFh> -2=><FEh> -128=><80h> ...
#1C2B0h	r=>s;r>0?	Rpl	conversion r=>s. ajoute true ou false suivant que r était > 0 ou non.
#18C34h	r?=>s	LM	"Two few Arguments" si depth < r
#193DAh	{r}=>{s}	Rpl	avec liste de 1 ou 2 éléments ex: {5}=>{<5h>}, {3 7}=>{<3h> <7h>}
#19529h	{s}=>{r}	Rpl	inverse du précédent
#19402h	r+=>{s}	Rpl	si r>0, r=>s, puis 1 ->LIST si r≤0, erreur "Bad Argument Value"
#19538h	s=>{r}	Rpl	ex: <7h> => { 7 }
#194F7h	2r=>2s	Rpl	ex: 2:15 1:8 => 2:<Fh> 1:<8h>
#1950Bh	2s=>2r	Rpl	ex: 2:<Fh> 1:<8h> => 2:15 1:8
#4F3D1h	2b=>2s	Rpl	ex: 2:#45h 1:#Fh => 2:<45h> 1:<Fh>
#51532h	{2b}=>2s	Rpl	Vérifie liste de deux binaires, sinon Doerr "Bad Argument Value". ex: 1:{#3Fh #4Ah} => 2:<3Fh> 1:<4Ah>
#2B45Ch	2r=>2rr	Rpl	2:x 1:y => 2:xx 1:yy
#2B470h	2rr=>2r	Rpl	2:xx 1:yy => 2:x 1:y
#4F0DEh	s=>±r	Rpl	conversion signée, le bit de poids max étant le bit de signe. Ex:<80000h>=>-524288 <7FFFFh>=>524887
#4F29Bh	r+=>s;true	Rpl	voir r+=>s en #18CEAh.
#519DFh	c=>2rr;swap	Rpl	1:(x,y) => 2:yy 1:xx
#62CE1h	r=>s;dup	Rpl	1:9 => 2:<9h> 1:<9h>
#62E7Bh	r=>s;swap	Rpl	2:α 1:5 => 2:<5h> 1:α
#62E8Fh	r=>rr;swap	Rpl	2:α 1:x => 2:xx 1:α
#63295h	dup;st=>gn		
#0EED0h	r=>b10	Rpl	donne un entier binaire de taille 10
#10CBFh	s256=>±r	Rpl	0≤s≤<FFh> => -128≤r≤127. Le bit fort de "s" servant de bit de signe. <FFh>=>-1, <7Fh>=>127, <80h>=>-128.
#10D90h	rr3=>r3	Rpl	conversion au niveau 3 de la pile
#2E650h	nm=>st!	Rpl	nm peut déjà être une chaîne; si ce n'est ni un nom ni une chaîne, alors erreur "Bad Argument Type"
#30794h	version48	LM	Donne chaine "HPHP48-A" (ou -B,..,-E)
#35CEAh	r2=>c	Rpl	2:x 1:α => 2:(x,0) 1:α
#4AF4Fh	2/(s);s=>rr	Rpl	divise s par 2, puis conversion
#4F287h	(r≥-.5)=>s?	Rpl	Si r≥-.5, alors r=>s, et ajoute true. Sinon false seul.
#51A71h	Σsq(2r)		2:x 1:y => 1:x²+y² (au format long)

◆ MODIFIER L'ADRESSE-PRÉFIXE:

Voici quelques SYSEVALs "*atypiques*" mais dont la place est sans doute dans ce chapitre, consacré aux conversions entre type différents. Vous savez que le type d'un objet est défini par son ***adresse-préfixe***.

Il est possible ici d'obtenir l'adresse-préfixe d'un objet (pas de tous) et et de modifier le type d'un objet en modifiant cette adresse-préfixe.

Adresse	Mnémonique	Commentaires (LM si langage machine)
#0358Fh	prefix(ar)	(LM) Donne <2933h> ou <2977h> suivant que le tableau est formé de réels ou de complexes.
#03C64h	prefix(_)	(LM). Donne l'adresse-préfixe, sous forme d'un entier-système. Reconnait les réels, les complexes les réels longs, les noms (locaux ou globaux), les les expressions, listes, directories, unités, et les programmes Rpl . Dans autres cas, donne <0h>.
#1CB90h	type!	Comme type, mais renvoie 27 si pile vide.
#05ACCh	ModType!(_,s)	obj2 prend le type défini par l'entier-système s, qui désigne un prologue. Pas de newob !!
#05AB3h	ModType(_,s)	comme ci-dessus, mais avec newob sur obj2

Soyez très prudent avec les instructions "*ModType*", qui ne font que changer l'adresse-préfixe de l'objet auquel elles s'appliquent.
Il faut compter que les différents types d'objets se différencient non seulement par leur adresse-préfixe, mais aussi avec la façon dont est codé leur contenu.

Pour ma part, je ne vois que quelques types d'objets pouvant se prêter à de telles expériences:

- Passer d'un nom local à un nom global ou l'inverse: On a déjà les adresses **#05AEDh** (***nm=>ln***) et **#05B01h** (***nm=>gn***) qui conviennent parfaitement. D'ailleurs ces deux routines utilisent l'instruction que j'ai nommée ***ModType!(_,s)***...

- "*Circuler*" entre les trois types suivants: Chaînes de caractères, entiers binaires, et objets-code (qui possèdent tous la propriété d'être codés par leur adresse-préfixe, suivie de la longueur de l'objet sur 5 quartets).

- "*Circuler*" entre les types suivants: Listes, expressions, structures Rpl, et objets-unités (ce sont les différents types d'objets-composés. Mais attention à ne pas transformer n'importe quoi en objet-unité!).

CONVERSIONS DE TYPE

Seuls les deux derniers points ci-dessus méritent qu'on s'y arrête. Nous allons maintenant voir quelques routines effectuant les transformations évoquées. On utilise ici la variante "***ModType(_,s)***" (**#5AB3h SYSEVAL**) qui provoque un "***newob***" sur ***obj2*** au moment de la transformation.

▸ **Exemple: Chaîne < = > Binaires < = > Codes**

Les trois programmes qui suivent attendent au niveau 1 un objet qui soit une chaîne, un binaire, ou un objet-code. Ils le transforment en un objet d'un des deux autres types. Toutes ces transformations sont réversibles.

'**→BI**' (*transformation en entier binaire*)
« #2A4Eh #5A03h SYSEVAL #5AB3h SYSEVAL »

'**→ST**' (*transformation en chaîne de caractères*)
« #2A2Ch #5A03h SYSEVAL #5AB3h SYSEVAL »

'**→CO**' (*transformation en objet-code*)
« #2DCCh #5A03h SYSEVAL #5AB3h SYSEVAL »

Ainsi:
"123" =[→BI]=> #333231h =[→CO]=> Code =[→ST]=> "123"

ATTENTION:
Les chaînes, binaires sont des objets de type "*donnée*".
Les évaluer n'a donc aucune autre conséquence que de les placer sur la pile. N'allez au contraire pas évaluer un ***Objet-Code*** obtenu de cette manière et au hasard, sinon un "*Memory Clear*" vous attend !!

▸ **Exemple: Listes < = > Rpl < = > Expressions:**

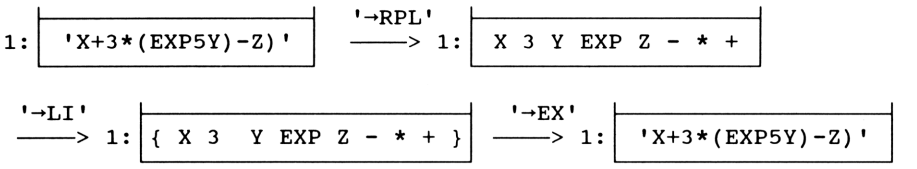
Les trois programmes qui suivent attendent au niveau 1 un objet qui soit une liste, un programme RPL, ou une expression. Ils le transforment en un objet d'un des deux autres types. Toutes ces transformations sont réversibles.

'**→LI**' (*transformation en liste*)
« #2A74h #5A03h SYSEVAL #5AB3h SYSEVAL »

'**→RPL**' (*transformation en programme*)
« #2D9Dh #5A03h SYSEVAL #5AB3h SYSEVAL »

'**→EX**' (*transformation en expression*)
« #2AB8h #5A03h SYSEVAL #5AB3h SYSEVAL »

▸ **Exemple:**



ATTENTION: N'essayez pas de transformer en un objet composé un objet qui n'en est pas déjà un !. La raison en est simple: tous les objets composés se terminent par une adresse-postfixe (**#0312Bh**).
Si vous partez d'un objet qui ne possède pas cette marque de fin, qu'obtiendrez vous ? (des ennuis peut-être).

◆ Transformations en chaînes de caractères:

L'utilisateur "*de base*" (ça n'a rien de péjoratif) sait comment transformer un objet quelconque en chaîne de caractères: il a à sa disposition l'instruction **→STR**.

La situation est beaucoup plus complexe dans l'univers imprévisible des SYSEVALs.
Il faut en effet déjà compter avec les formes internes de cette instruction (autrement dit les routines **→str** applicables à un type particulier d'objet).
Il faut également distinguer la transformation en chaîne d'un objet avec l'idée de poser cette chaîne sur la pile, ou pour placer l'objet en ligne de commande dans le but de le lire et/ou de le modifier. Les chaînes obtenues sont en effet parfois différentes.

Exemple:
Si l'objet est un **Grob** (*graphic object*), il existe deux chaînes susceptibles de le représenter:
▸ Une chaîne ayant l'allure **Graphic ...x...** (c'est elle qui est utilisée pour être placée sur la pile, ou pour représenter le "*Grob*" dans une liste, un programme)
▸ Une chaîne débutant par **GROB ....** (c'est elle qui est utilisé en mode édition, ou c'est la chaîne résultant de l'instruction **→STR**)

Il existe, pour chaque type d'objet, une ou deux adresses spécifiques transformant ce type d'objet en chaîne de caractères.

Quand un type d'objet relève de deux adresses distinctes, j'ai utilisé les mnémoniques ***→str1*** et ***→str2*** pour les distinguer. Il n'est cependant pas toujours très facile de comprendre ce qui les différencie.

CONVERSIONS DE TYPE

Adresse	Mnémonique	Commentaires (LM si langage machine)
#162ACh	->str1(r)	(LM) Ex: 12,568,799.279 => "12,568,799.279"
#162B8h	->str2(r)	(LM) Ex: 12,568,799.279 => "12568799.279"
#160D1h	->str1(nm)	Ex: 'ABC' => "'ABC'"
#160E5h	->str2(nm)	(LM) Difficile à différencier du précédent
#1622Ah	->str1(ar)	Donne une chaîne sans retours chariots. Convient pour tableaux, listes, programmes, taggués, répertoires, (c'est leur ->str1)
#161D0h	->str2(ar)	Transforme en chaîne comme en mode Edition. Convient pour tableaux, listes, programmes, unités, expressions ,taggués, répertoires
#15F3Dh	->str1(st)	Ex: "ABC" => ""ABC"" => """ABC"""
#15ED8h	->str2(st)	Ex: "ABC" => ""ABC"" => "C$ 5 "ABC""
#1746Eh	->str(c)	
#16103h	->str1(ex)	C'est aussi ->str1(uo)
#15D38h	->str(lb)	->STR appliqué à une librairie
#15D97h	->str(bk)	->STR appliqué à un objet backup
#15DF1h	->str1(go)	Crée par ex. la chaîne "Graphic 131 x 64" à partir d'un grob de dimensions 131x64.
#15E1Fh	->str2(go)	Donne par exemple "GROB 131 64 ...."
#1616Ch	->str(xn)	->STR appliqué à un XLIB name
#15D06h	->str(s)	<4Ah> => "<4Ah>"
#54061h	->str1(b)	Sur les 16 chiffres hexa de poids minimum
#15F83h	->str2(b)	->STR appliqué à un binaire. Utilise en particulier C# pour les binaires longs.

Venons-en à des SYSEVALs plus "généralistes".

Adresse	Mnémonique	Commentaires (LM si langage machine)
#14088h	->str	C'est la forme interne de ->STR. Ne vérifie pas la présence d'un argument
#15B31h	->str!	(LM) Interne à #14088h (->str). Utilise la longueur du buffer définie par #1795Ah (DoBuffStr(s))
#1795Ah	DoBuffStr(s)	(LM) Donne une taille de s carac. au buffer utilisé par #15B31h (->str!). Si s=<0h>, taille quelconque ?
#159FAh	BuffStr=<13h>	Longueur <13h>=19 pour le buffer de ->str!
#15B13h	->StackStr	Produit chaîne utilisée pour affichage sur la pile en mode multilignes.
#15A0Eh	->EditStr	Produit chaîne utilisée pour mode Edition
#15978h	->StackStr!	Produit chaîne utilisée pour affichage sur la pile en mode uniligne (22 caractères au plus). Utilise la longueur du buffer définie par DoBuffStr(s)
#15174h	->StackStr22!	Comme ci-dessus, mais précédé de BuffStr=<13h>
#496F5h	->StackStr22!+	Comme ci-dessus, mais suivi de +(2st)

Remarques:

Les différences entre les instructions "***→StackStr***" et "***→EditStr***" sont surtout sensibles pour les objets composés (listes, expressions, programmes) les tableaux, et les objets-répertoires.

Dans ces différents cas, l'instruction "***→StackStr***" transforme l'objet en une chaîne ne contenant pas de "***Newline***", alors qu'au contraire l'instruction "***→EditStr***" transforme l'objet en une chaîne formatée affichable par **DISP**.

Je n'ai pas encore vraiment compris l'utilité du "**buffer**" de décompilation utilisé dans certaines adresses ci-dessus. Quelque soit la longueur de ce "**buffer**", la chaîne obtenue est toujours la même.

◆ More SYSEVALs...:

Voici enfin quelques adresses pour clore le problème (bien plus compliqué que ce à quoi on pouvait s'attendre) des transformations d'objets en chaînes de caractères:

Adresse	Mnémonique	Commentaires (LM si langage machine)
#15E83h	->str!(go)	Transforme le contenu d'un grob en chaîne. appelé par #15E1Fh (->str2(go)) Là où ->str2(go) donne "GROB 131 64 FEDCB.." ->str!(go) donne "0400038000FEDCB...
#35287h	->StdStr(r)	Transforme un réel en chaîne, comme si on en mode STD, sans changer mode en cours.
#352C3h	->StdStr(c)	Idem avec nombres complexes
#540BBh	->str!(b)	(LM) Sans le h,o,d,ou b terminal appelé par ->str1(b) en #54061h
#5F9B9h	->StrExpr	Transforme obj1 en chaîne comme s'il faisait partie d'une expression. Les objets non reconnus deviennent "UNKNOWN"
#5FA04h	->StrOp(_)	Transforme un opérateur intégré en chaîne. NEG devient "-". Un objet non reconnu est transformé en "UNKNOWN".
#1605Fh	add2'(st)	(LM) Exemple "ABC" donne "'ABC'"
#1606Ch	add2"(st)	(LM) Ajoute 2 délimiteurs ": "ABC"=>""ABC""
#167D8h	->str:(s)	(LM) Exemple <Fh> => "15: "
#1685Ch	s:Str(st,s)	(LM) Exemple: "Hello",<Fh> deviennent "15: Hello" (22 car.)
#39A4Ch	swap;nm=>st	
#222EEh	swap;->EditStr;+(2st)	

EXPRESSIONS ALGÉBRIQUES

Les ***expressions*** sont des *objets-composés*, au même titre que les listes, les structures Rpl et, dans une moindre mesure, les objets-unité. Elles sont codées de la manière suivante:

- Une adresse préfixe (**#02AB8h**)
- Le contenu de l'expression, en notation polonaise inverse (chaque objet de l'expression étant représenté par son *contenu*, ou par son *adresse* s'il s'agit d'un objet référencé en ROM)
- Une adresse-postfixe (**#0312Bh**)

Un chapitre a déjà été consacré à l'étude des listes. On y a étudié les SYSEVALs qui s'appliquent, de manière très générale, aux objets-composés. Ces SYSEVALs peuvent donc être utilisés sur des expressions.

Il convient cependant de rappeler quelques adresses importantes. Pour éviter trop de répétitions, ne seront données ici que les adresses et les mnémoniques, mais pas les commentaires.

Adresse & mnémonique		Adresse & mnémonique		Adresse & mnémonique	
#1CAF0h	pos(cp,_)	#644A3h	spos(cp,_)	#644BCh	spos(_,cp)
#05089h	get1(cp)	#056B6h	get(cp,s)?	#1D898h	get(cp,lr)
#1D8A2h	get(cp,lr)!	#1D9BCh	geti(cp,lr)	#62B9Ch	get(cp,s)
#62D1Dh	get(cp,s);dup	#0567Bh	ssize(cp)	#1CA3Ah	size(cp)
#63231h	dup;ssize(cp)	#643EFh	InCp(_,cp)?	#06F9Fh	eval(cp)
#64426h	map(cp,_,test)	#644D0h	1st(cp,test)?	#054AFh	breaks(cp)
#6480Bh	NextObj(cp,s)?	#1C973h	breakr(cp)	#62B88h	break;drop
#62C41h	break;dup	#631E1h	dup;break	#631F5h	swap;break
#0546Dh	->expr(s)	#5E652h	->expr1	#5E661h	->expr1!

Un chapitre consacré spécialement aux expressions est nécessaire quand on connait l'importance du calcul formel pour la HP48.

Pourtant ce chapitre ne fermera pas la question du traitements des expressions sur la HP48. En effet, pour manipuler plus facilement les expressions algébriques, la calculatrice les "*casse*" sur la pile. L'adresse **#54AFh** (mnémonique "***sbreak(cp)***") est ici très importante.

Exemple: Considérons l'expression **'X+2*SIN(Y/Z)'**

Cette expression est "*cassée*" par **#54AF SYSEVAL**, pour donner:

9:'X' 8:2 7:'Y' 6:'Z' 5:/ 4:SIN 3:* 2:+ 1:<8h>

On reconnaît là la forme ***RPN*** équivalente:

'X' 2 'Y' 'Z' 5 / SIN * +

Au niveau 1, l'entier-système <8h> est là pour donner la taille du bloc ainsi libéré sur la pile

Nous reviendrons plus tard sur cet aspect du traitement des expressions. Ce chapitre ne traitera que des expressions représentées sous leur forme traditionnelle.

#18E18h	->num(ex)	#1CF2Eh	eq->(ex)	#1CFD0h	obj->(ex)
#1F38Bh	where(ex,li)	#20B00h	show(ex,li)	#20D7Eh	define(eq)
#1FABAh	match^(ex,li)	#20E0Ah	define(_,ex)	#54954h	der(ex,nm)
#1FACEh	matchv(ex,li)	#54EA0h	re(ex)	#54EB9h	im(ex)
#54ED2h	not(ex)	#54EEBh	neg(ex)	#54F04h	abs(ex)
#54F1Dh	conj(ex)	#54F36h	inv(ex)	#54F4Fh	arg(ex)
#54F68h	sign(ex)	#54F81h	√(ex)	#54F9Ah	sq(ex)
#54FB3h	sin(ex)	#54FCCh	cos(ex)	#54FE5h	tan(ex)
#54FFEh	sinh(ex)	#55017h	cosh(ex)	#55030h	tanh(ex)
#55049h	asin(ex)	#55062h	acos(ex)	#5507Bh	atan(ex)
#55094h	asinh(ex)	#550ADh	acosh(ex)	#550C6h	atanh(ex)
#550DFh	exp(ex)	#550F8h	ln(ex)	#55111h	log(ex)
#5512Ah	alog(ex)	#55143h	lnp1(ex)	#5515Ch	expm(ex)
#55175h	fact(ex)	#5518Eh	ip(ex)	#551A7h	fp(ex)
#551C0h	floor(ex)	#551D9h	ceil(ex)	#551F2h	xpon(ex)
#5520Bh	mant(ex)	#55224h	d->r(ex)	#5523Dh	r->d(ex)
#55256h	ubase(ex)	#5526Fh	uval(ex)	#5599Ah	and(ex,r)
#559B3h	and(r,ex)	#559CCh	and(2ex)	#559E5h	or(ex,r)
#559FEh	or(r,ex)	#55A17h	or(2ex)	#55A30h	xor(ex,r)
#55A49h	xor(r,ex)	#55A62h	xor(2ex)	#55A7Bh	==(ex,nb)
#55A7Bh	==(ex,uo)	#55A94h	==(nb,ex)	#55A94h	==(uo,ex)
#55AADh	==(2ex)	#55AC6h	<>(ex,nb)	#55AC6h	<>(ex,uo)
#55ADFh	<>(nb,ex)	#55ADFh	<>(uo,ex)	#55AF8h	<>(2ex)
#55B11h	<(ex,nb)	#55B11h	<(ex,uo)	#55B2Ah	<(nb,ex)
#55B2Ah	<(uo,ex)	#55B43h	<(2ex)	#55B5Ch	>(ex,nb)
#55B5Ch	>(ex,uo)	#55B75h	>(nb,ex)	#55B75h	>(uo,ex)
#55B8Eh	>(2ex)	#55BA7h	≤(ex,nb)	#55BA7h	≤(ex,uo)
#55BC0h	≤(nb,ex)	#55BC0h	≤(uo,ex)	#55BD9h	≤(2ex)
#55BF2h	≥(ex,nb)	#55BF2h	≥(ex,uo)	#55C0Bh	≥(nb,ex)
#55C0Bh	≥(uo,ex)	#55C24h	≥(2ex)	#55C3Dh	%(ex,nb)
#55C3Dh	%(ex,uo)	#55C56h	%(nb,ex)	#55C56h	%(uo,ex)
#55C6Fh	%(2ex)	#55C88h	%ch(ex,nb)	#55C88h	%ch(ex,uo)
#55CA1h	%ch(nb,ex)	#55CA1h	%ch(uo,ex)	#55CBAh	%ch(2ex)
#55CD3h	%t(ex,nb)	#55CD3h	%t(ex,uo)	#55CECh	%t(nb,ex)
#55CECh	%t(uo,ex)	#55D05h	%t(2ex)	#55D1Eh	comb(ex,r)
#55D37h	comb(r,ex)	#55D50h	comb(2ex)	#55D69h	perm(ex,r)
#55D82h	perm(r,ex)	#55D9Bh	perm(2ex)	#55DB4h	rnd(ex,r)
#55DCDh	rnd(ar,ex)	#55DCDh	rnd(nb,ex)	#55DCDh	rnd(uo,ex)
#55DE6h	rnd(2ex)	#55DFFh	trunc(ex,r)	#55E18h	trunc(ar,ex)
#55E18h	trunc(nb,ex)	#55E18h	trunc(uo,ex)	#55E31h	trunc(2ex)
#55E4Ah	max(ex,r)	#55E4Ah	max(ex,uo)	#55E63h	max(r,ex)
#55E63h	max(uo,ex)	#55E7Ch	max(2ex)	#55E95h	min(ex,r)
#55E95h	min(ex,uo)	#55EAEh	min(r,ex)	#55EAEh	min(uo,ex)
#55EC7h	min(2ex)	#55EE0h	^(ex,r)	#55EE0h	^(ex,uo)
#55EF9h	^(r,ex)	#55EF9h	^(uo,ex)	#55F12h	^(2ex)
#55F2Bh	+(ex,r)	#55F2Bh	+(ex,uo)	#55F44h	+(r,ex)
#55F44h	+(uo,ex)	#55F5Dh	+(2ex)	#55F76h	-(ex,r)
#55F76h	-(ex,uo)	#55F8Fh	-(r,ex)	#55F8Fh	-(uo,ex)
#55FA8h	-(2ex)	#55FC1h	*(ex,r)	#55FC1h	*(ex,uo)
#55FDAh	*(r,ex)	#55FDAh	*(uo,ex)	#55FF3h	*(2ex)
#5600Ch	/(ex,r)	#5600Ch	/(ex,uo)	#56025h	/(r,ex)
#56025h	/(uo,ex)	#5603Eh	/(2ex)	#56057h	mod(ex,r)
#56070h	mod(r,ex)	#56089h	mod(2ex)	#560A2h	xroot(ex,r)
#560A2h	xroot(ex,uo)	#560BBh	xroot(r,ex)	#560BBh	xroot(uo,ex)
#560D4h	xroot(2ex)	#572A2h	isol(ex,gn)	#57A0Ch	expan(ex)
#57D90h	colct(ex)	#58D75h	show(ex,gn)	#591ADh	quad(ex,gn)
#595DDh	taylr(ex,gn,r)	#59F91h	size(ex)		

EXPRESSIONS ALGÉBRIQUES

Vous avez compris que le tableau précédent contient les SYSEVALs correspondant aux opérations standards de la HP48, quand l'un au moins de(s) argument(s) est une expression algébrique. Toutes ces adresses désignent des routines RPL, et leur mnémonique leur tient lieu de commentaire.

Rappelons les abréviations utilisées:

ex: expression ***eq***: equation ***li***: liste ***uo***: objet-unité
r: nombre réel ***gn***: nom global ***nm***: nom local ou global
nb: nombre réel ou complexe.

Vous avez pu remarquer que certaines des adresses ci-dessus sont répétées. Celà correspond au fait que les routines qui prennent deux arguments (dont une expression) sont souvent les mêmes quand le deuxième argument est un réel ou un objet-unité.

Voyons maintenant le reste des SYSEVALs portant sur les expressions, et tout d'abord les différentes variantes de l'instruction **ROOT**:

Adresse	Mnémoniques et commentaires
#32F77h	root | C'est la forme interne de ROOT. Ne vérifie pas la | présence de trois arguments et leur validité.
#32FF9h	Solve(ex,nm,nb)? | Résoud "ex" par rapport à 'nm', en partant de "nb". | Le résultat est placé au niveau 3. | Au niveau 2 un code indique la nature du résultat: | (<A01h> <A02h> <A03h> <A04h> <A05h> <A06h>) | Au niveau 1, un booléen ("true" si pas de problème)
#49BD2h	Solve(ex,nm,nb) | Résoud "ex" par rapport à 'nm', en partant de "nb" | Le résultat est placé au niveau 1. Au niveau 2, on | trouve "Sign Reversal", ou "" si Zéro. | En cas de problème ou d'interruption par ON, un | message est affiché en ligne de menu. | 'nb' est remplaçable par une liste d'estimations.

Remarques: la signification des codes évoqués ci-dessus est:

<A01h> : "*Bad Guesse(s)*" <A02h> : "*Constant?*"
<A03h> : "*Interrupted*" <A04h> : "*Zero*"
<A05h> : "*Sign Reversal*" <A06h> : "*Extremum*"

Dans tous les cas ci-dessus, on peut "tracer" la recherche en appuyant sur une touche quelconque (sauf **ON**).

L'opérateur Σ possède quatre formes internes:

Adresse	Mnémoniques et commentaires
#56949h #56A06h #56A4Ch #56AC9h	Σ(3ex,_) Σ(2ex,r,_) Σ(ex,r,ex,_) Σ(ex,2r,_)
	4 adresses d'éxécution de l'opérateur Σ, suivant la nature des arguments.

Maintenant, variations sur le thème "*intégration & dérivation*". Vous savez que l'indicateur **-3** permet de définir le mode (***numérique*** ou ***symbolique***) de traitement des expressions algébriques. S'il est armé (c'est-à-dire à 1), on est en mode numérique. S'il est désarmé, on est en mode symbolique.

Dans certaines opérations sur les expressions, la HP48 force momentanément le mode symbolique, avant de retourner au mode initial (qui était peut-être d'ailleurs le mode symbolique...).

Vous trouverez ci-dessous certaines adresses où on "*court-circuite*" cette précaution de la HP48.

Adresse	Mnémoniques et commentaires
#1F201h	integr(3_,nm) | Il s'agit ici de l'intégration symbolique RPN
#54977h	1StepDer(ex,nm) | Un pas de dérivation de "ex" par rapport à "nm"
#5662Eh	NumInteg(ex,exn,2nb) | Évalue numériquement l'intégrale, de nb2 à nb1 | "exn" doit être une expression réduite à un nom. | (utiliser pour celà: ->expr1 en #5E652h)
#595B0h	SymDer(ex,nm) | Dérivation totale et symbolique.
#595F1h	taylr!(ex,gn,r) | Comme Taylr(ex,gn,r) en #595DDh, sans le passage | temporaire en mode symbolique.
#5AAC7h	SymbInteg(2nb,ex,exn) | voir #5662Eh. Intégration symbolique.
#57DA4h	colct!(ex) | Comme colct(ex) en #57D90h, sans le passage | temporaire en mode symbolique.
#591C1h	quad!(ex,gn) |Comme quad(ex,gn) en 591ADh, sans le passage | temporaire en mode symbolique.

Voici quelques SYSEVALs un peu inclassables (tous des Rpl).

Notez tout de même que les adresses "***->expr(s)***" et "***sbreak(s)***" me semblent supérieures à "***doexpr(s)***" et "***break(ex)***" (ne serait-ce que parce que les deux premières sont directement implantées en langage machine).

Adresse		Mnémoniques et commentaires
#1568Fh	Equation?	True si obj1 est une équation, False sinon
#1578Dh	->num	Forme interne de ->NUM, après vérification de la présence d'un argument, et DTAG
#22F86h	->num2	Effectue ->num sur niveau 2
#5A310h	doexpr(s)	Comme "->expr(s)" en #0546Dh
#5DEAAh	dup;ex?	1:obj => 2:obj, 1:true ou false
#5F2A3h	break(ex)	Comme "sbreak(cp)" en #054AFh

Dans les adresses ci-dessous (toutes des Rpl), on peut extraire le premier argument, ou le second, d'une expression dont la racine est un opérateur binaire (en principe).

N.B: Les *fonctions utilisateurs*, du type **f(X,Y,Z)** par exemple, sont codées de la manière suivante, en reprenant cet exemple:

Apply(['X','Y','Z','f'],<3h>).

Le premier argument est ici le "*bloc*" **[X Y Z 'f']**, où **f** est une expression réduite à un nom (créée par ***->expr1*** en **#5E652h**), et le deuxième argument est l'entier **<3h>** qui donne le nombre d'arguments de '**f**'.

Adresse	Mnémoniques et commentaires
#49B82h	2ndMember(_) | Donne le deuxième membre si obj1 est une équation | sinon laisse obj1 inchangé. | Ex: 'X+1=Y*Z' => 'Y*Z' => 'Y*Z' ...
#49D8Ah	dup;1stMember=gn? | Teste que obj1 est une équat. et que son 1er membre | est réduit à un nom global.
#49DD5h	dup;BreakEquation? | False si l'objet n'est pas une équation. | Sinon sort 1er membre, 2ème membre, et true. | Ex: 'X+1=Y*Z' => 'X+1=Y*Z', 'X+1', 'Y*Z', true
#51482h	1stArgRpn(ex) | Donne le 1er argument de "ex", sous forme rpn | Ex: 1: 'SIN(X+Y*Z)/(2*U-V)' => 1: X Y Z * + SIN
#51496h	1stArgb(ex) | Donne 1er argument de "ex", sous forme de bloc | Ex:'SIN(X+Y*Z)/(2*U-V)' => 'X','Y','Z',*,+,SIN,<6h>
#514AFh	2ndArgRpn(ex) | Donne le 2nd argument de "ex", sous forme rpn | Ex: 1:'SIN(X+Y*Z)/(2*U-V)' => 1: 2 U * V -
#514C3h	2ndArgb(ex) | Donne le 2nd argument de "ex", sous forme de bloc | Ex: 1:'SIN(X+Y*Z)/(2*U-V)' => 2,'U',*,'V',-,<5h>

Voici maintenant des SYSEVALs (Rpl) dont le but est au contraire de créer des expressions, au moyen d'instructions du type "***APPLY***".

Adresse	Mnémoniques et commentaires
#1F542h	quote(_) |Rappel 'f(1+1,QUOTE(2*3))' --EVAL--> 'f(2,2*3)'
#1F585h	apply(li,nm) |Exemple: 2:{X Y 'U+V'} 1:f => 1:'f(X,Y,U+V)'
#1F5F6h	apply(__,s) | Effectue l'opération APPLY en partant d'un bloc | Name,o(1),o(2),...o(n),<n+1>, où Name est le | nom à appliquer aux objets o(1),o(2),...,o(n). | Ex: 'f','X',1,'Y+Z',<4h> => 'f(X,1,Y+Z)'
#1F640h	apply(_,ex,s) | Part d'un bloc: o(1),o(2),...o(n),Expr,<n+1>, | où "Expr" est une expression dont le seul élément | est un nom (utiliser "->expr1" en #5E652h). | Crée une expression en appliquant ce nom aux objets | du bloc (du 1er à l'avant-dernier).

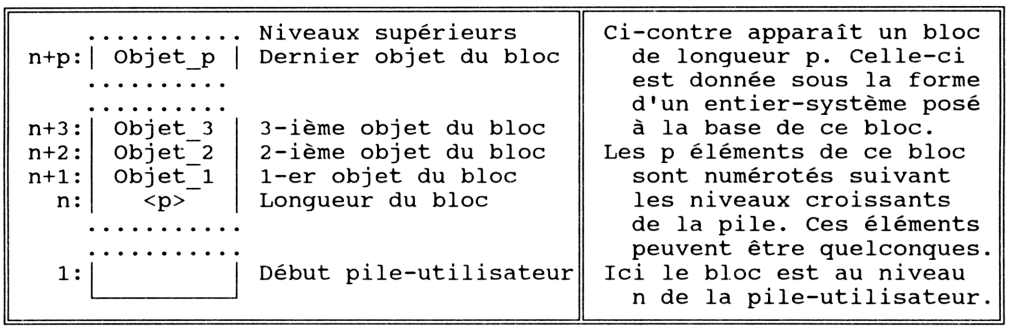
C'est la fin de ce chapitre....
Le temps est donc venu de brader les SYSEVALs qui restent.

Adresse	Mnémoniques et commentaires
#5910Bh	show!(ex,li) | ne vérifie pas que obj1 est une liste de noms.
#1F43Eh	ChkExpr(s) | Appelé au début de Where(__,s) en #1F439. Vérifie | la conformité du bloc Expr,n1,v1,n2,v2....,<m>.
#1F439h	Where(__,s) |Effectue l'opération de WHERE, en partant d'un bloc |du type Expr,n1,v1,n2,v2....,<m>, où Expr est une |expr. et où (n1,v1) (n2,v2) etc... sont des couples |"nom,valeur". Ici m = 1 + nombre de couples (ni,vi)
#547B5h	Where!(__,s) | Comme en #1F439, mais après vérification de la | conformité du bloc Expr,n1,v1,n2,v2....,<m>
#592B6h	quad(3ex)? | 3:a,2:b,1:c => 2:'(-b+s1*√(b^2-4*c*a))/(2*c)' | le booléen vaut True.
#1F8CFh	StoLT(_,ex) | Quand on veut stocker un objet à une place d'une | liste ou d'un tableau. N'enlève pas tags niveau 2. | Ex: avec 2:9 1:'T(3)', 9 va en 3ème pos. dans 'T'
#1F05Bh	ChkExpr1Name | (LM) Vérifie que obj1 est une expression réduite à | un nom. Un nom peut être transformé en une telle | expression par ->expr1 en #5E652h.
#54C63h	ChkAlg(s) | Vérifie que les s premiers niveaux de la pile sont | occupés par des nombres, des noms , des unités, | ou des expressions (Sinon "Bad Argument Type") | Ces objets sont laissés sur la pile, inchangés.
#60B17h	match'(ex,li) | Si l'objet suivant dans le Rpl est True, alors | effectue MatchUp(ex,li), sinon MatchDown(ex,li)
#4A194h	(A,B)=>'A+B*X' | Ici A,B peuvent être des express.,
#4A1D4h	(A,B)=>'A+B*LN(X)' | des noms, des nombres. Routines
#4A223h	(A,B)=>'A*EXP(B*X)' | utiles dans menu STAT (BESTFIT)

LES "MÉTA-OBJETS"

Ce chapitre est consacré à une structure très utilisée par la HP48, notamment dans la gestion des expressions algébriques, mais qui permet également des "*macro-manipulations*" de la pile-utilisateur.

On parlera de "***Méta-Objet***", ou de "***bloc***" (c'est cette deuxième appellation qui sera employée ici) pour désigner un groupement d'objets organisé de la manière suivante:



On considérera que l'entier-système <0h> est à lui seul un bloc (que l'on pourra appeler le "***bloc vide***")

On connait déjà au moins un SYSEVAL qui produit un ***bloc***:
Il s'agit de l'adresse **#54AFh** (mnémonique "***breaks(cp)***"), qui libère sur la pile les différents éléments d'un *objet-composé* (Rpl, liste, expression, objet-unité).

Rappelons les exemples donnés dans le chapitre consacré aux *objets-composés*, et qui illustrent le rôle de la séquence **#54AFh SYSEVAL**:

1:1.2_m/s ==> **6:1.2 5:"m" 4:"s" 3:{} 2:{} 1:<5h>**

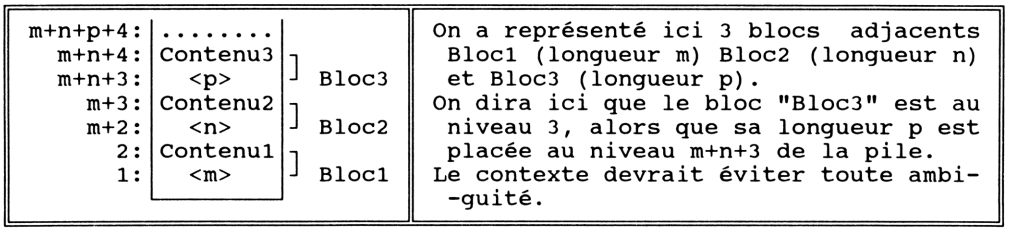
1:« DUP ROT + / » ==> **7:« 6:DUP 5:ROT 4:+ 3:/ 2:» 1:<6h>**

1:'X+3*Y' ==> **6:'X' 5:3 4:'Y' 3:* 2:+ 1:<5h>**

1:{ "A" "B" "C" } ==> **4:"A" 3:"B" 2:"C" 1:<3h>**

On obtient bien à chaque fois un ***bloc***, au niveau 1 de pile. Inversement, les adresses suivantes:
#05481h ("→unit(s)"), **#05445h ("→rpl(s)")**
#0546Dh ("→expr(s)") **#05459h ("→list(s)")**
permettent de *refermer* ce bloc en reconstituant l'objet initial.

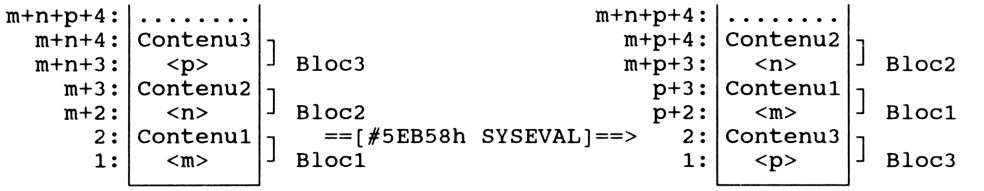
De nombreux SYSEVALs s'appliquent aux "blocs". Quand il s'agit d'un bloc placé au niveau 1 de la pile, il n'y a pas de confusion possible. Au contraire, il arrivera qu'on doive considérer plusieurs blocs adjacents sur la pile, le premier d'entre eux étant placé au niveau 1. Nous pouvons représenter cette situtation de la manière suivante:



Le fait de considérer plusieurs blocs successifs sur la pile (le premier "*démarrant*" au niveau 1) nous amène donc à imaginer une nouvelle notion: "*la pile des blocs*".

Sur cette pile, les blocs seront numérotés dans l'ordre croissant des niveaux de la pile-utilisateur. La pile des blocs démarre donc au niveau 1 de la pile utilisateur (à condition que s'y touve le début d'un bloc...) et se termine avec le dernier objet du dernier des blocs adjacents ainsi parcourus.

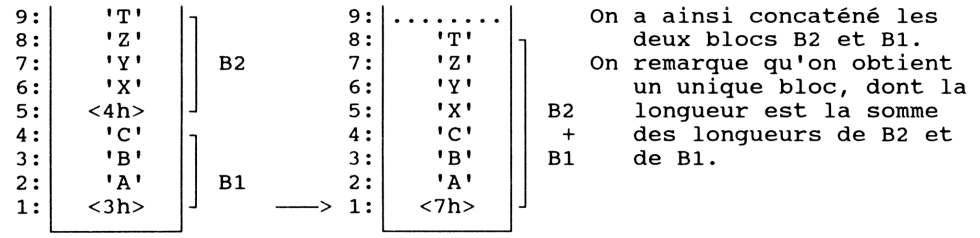
Un telle "*vue de l'esprit*" s'avère particulièrement utile pour interpréter des SYSEVALs comme celui de l'adresse **#5EB58h**, pour lequel j'ai choisi le mnémonique "***rotB***", car son rôle est effectivement d'effectuer un "**ROT**" sur les trois premiers blocs de la pile.



Il est aussi possible d'additionner, ou mieux de ***concaténer***, deux ou plusieurs blocs consécutifs (la séquence de blocs démarrant en principe au niveau 1 de la pile-utilisateur).

Pour prendre un exemple, considérons les deux blocs:

B2: 'T','Z','Y','X',<4h> et B1: 'C','B','A',<3h>



Dans cette opération, le ***bloc vide*** (qui se réduit à une longueur égale à <0h>) joue le rôle d'***élément neutre***.

Dans le tableau ci-dessous, j'ai regroupé les principaux SYSEVALs s'appliquant d'une façon globale à des blocs (autrement dit n'allant pas lire ou écrire d'élément à l'intérieur).

Il a fallu employer certaines notations pour les commentaires:

- **B5,B4,B3,B2,B1** désignent des blocs (au départ B1 est au niveau 1 de la pile; au dessus se trouve B2, etc...)
- ***α,ß,σ,μ*** sont des objets occupant chacun un niveau de la pile.
- **[*α*,<1h>]** est le bloc de longeur 1 dont le seul élément est α.
- **B*ϕ*** désigne le bloc vide, réduit à <0h>.

Adresse	Mnémonique	Type	Commentaires
#0326Eh	dropB	LM	effectue donc dropn(s)
#169A5h	dropB;false	Rpl	B1 => false
#25322h	<4h>;swapB	Rpl	B1,α,ß,σ,μ => [α,ß,σ,μ,<4h>],B1
#49DB2h	dropB;false	Rpl	B1 => false
#5551Ch	dropB;0;<1h>	Rpl	B1 => [0,<1h>]
#55535h	dropB;1;<1h>	Rpl	B1 => [1,<1h>]
#5554Eh	dropB;-1;<1h>	Rpl	B1 => [-1,<1h>]
#56183h	2dropB;0;<1h>	Rpl	B2, B1 => [0,<1h>]
#561D8h	2dropB;-1;<1h>	Rpl	B2, B1 => [-1,<1h>]
#5D886h	3rolldB;false	Rpl	B3, B2, B1 => B1, B3, B2, false
#5E35Ch	dupB	Rpl	B1 => B1, B1
#5E3E8h	<1h>;swapB	Rpl	B1,α => [α,<1h>],B1
#5E67Ah	<0h>;swapB	Rpl	B1 => Bϕ,B1
#5E857h	swap23B	Rpl	B3, B2, B1 => B2, B3, B1
#5E870h	swap34B	Rpl	B4, B3, B2, B1 => B3, B4, B2, B1
#5EB1Ch	swapB	LM	B2, B1 => B1, B2
#5EB58h	rotB	LM	B3, B2, B1 => B2, B1, B3
#5EBC6h	4rollB	LM	B4, B3, B2, B1 => B3, B2, B1, B4
#5EBDBh	3rolldB	LM	B3, B2, B1 => B1, B3, B2
#5EBEAh	4rolldB	LM	B4, B3, B2, B1 => B1, B4, B3, B2
#5EBFCh	rollB(s+1)	LM	Ex: si s=<4h>, effectue 5rollB
#5ED45h	5rollB	LM	B5,B4,B3,B2,B1 => B4,B3,B2,B1,B5
#5ED5Ah	5rolldB	LM	B5,B4,B3,B2,B1 => B1,B5,B4,B3,B2
#5ED6Ch	rolldB(s+1)	LM	Ex: si s=<4h>, effectue 5rolldB
#5FC38h	1;<1h>;swapB	Rpl	B1 => [1,<1h>], B1
#5FC24h	dup;pick(s+2)	Rpl	α,B1 => α,B1,α
#60D1Bh	dropB;false	Rpl	B1 => false
#61305h	dup;roll(s+2)	LM	α,B1 => B1,α
#63911h	dropB2	Rpl	B2, B1 => B1 (swapB puis dropB)
#63F1Ah	rotB;dupB	Rpl	B3, B2, B1 => B2, B1, B3, B3

Voici maintenant des SYSEVALs qui utilisent la notion de ***concaténation*** de blocs (voir ci-dessus). Notez bien que cette addition de blocs n'est pas ***commutative*** (on peut la comparer à l'addition des listes).

Adresse	Mnémonique	Type	Commentaires
#5E3ACh	+B31	Rpl	B3, B2, B1 => B3+B1, B2
#5E3C0h	+B32	Rpl	B3, B2, B1 => B3+B2, B1
#5E415h	+B21	LM	B2, B1 => B2+B1
#5E490h	+B321	Rpl	B3, B2, B1 => B3+B2+B1
#5E4D1h	swapB;+B21	Rpl	B2, B1 => B1+B2
#5E585h	break;+B21	Rpl	B2, Objet composé "cp" => B2+[cp]
#5E843h	+B23	Rpl	B3, B2, B1 => B2+B3, B1
#63F2Eh	rotB;+B21	Rpl	B3, B2, B1 => B2, B1+B3
#63F42h	3rolldB;+B21	Rpl	B3, B2, B1 => B1, B3+B2

Les adresses suivantes, qui s'appliquent encore globalement à un ou deux ***blocs***, sont particulièrement intéressantes:

Adresse	Mnémonique puis commentaires.
#584B2h	=2B? | Teste l'égalité des 2 blocs. Réponse True ou False. | Les deux blocs sont conservés sur la pile. | Pour être égaux, les deux blocs doivent avoir même | longueur et être composés des mêmes éléments, | dans le même ordre.
#5DE7Dh	revB | Inverse l'ordre des éléments du bloc B1. | Ex: 'A','B','C','D',<4h> ==> 'D','C','B','A',<4h>
#5E370h	conB(_,s) | Crée un bloc constant de s objets tous égaux à obj2 | Ex: 'Y','X',<4h> ==> 'Y','X','X','X','X',<4h> | 'Y','X',<0h> ==> 'Y',<0h>
#64345h	subB(2s) | Extrait du bloc B1 le sous-bloc constitué des objets | de positions s2 à s1 | Exemple: 9,8,7,6,5,4,3,2,1,<9h>,<3h>,<7h> | ===> 7,6,5,4,3,<5h> | Attention s2 doit être strictement inférieur à s1.

En particulier, l'adresse ***revB*** nous permet d'écrire un programme inversant rapidement l'ordre des éléments de la pile:

'***REVP***': (Checksum: #F081h; Taille 52 octets)

« #314Ch SYSEVAL #5DE7Dh SYSEVAL DROP »

Le programme équivalent:

« 2 DEPTH 1 - FOR k k ROLL NEXT »

met 8.46s à inverser 500 objets, alors que '**REVP**' met 4.53s.

LES "MÉTA-OBJETS"

Les SYSEVALs suivants permettent de lire le premier ou le dernier élément du ***bloc*** B1, en retirant ou non cet élément de B1.

Adresse	Mnémonique puis commentaires. (LM) si langage machine.
#5E4A9h	pullB[1] | Extrait et place au niveau 1 le premier élément du bloc | Ex: 'A','B','C','D',<4h> => 'A','B','C',<3h>,'D'
#5E4BDh	pullB[n] | Extrait et place au niveau 1 le dernier élément du bloc B1. La longueur de B1 diminue d'une unité. | Ex: 'A','B','C','D',<4h> => 'B','C','D',<3h>,'A'
#6119Eh	getB[n] | (LM) Lit et place au niveau 1 le dernier élément du bloc B1 (dup;pick(s+1) : B1 => B1,B1[n]) | Ex: 'A','B','C','D',<4h> => 'A','B','C','D',<4h>,'A'
#630DDh	getB[n-1] | Lit et place au niveau 1 l'avant-dernier élément du bloc B1 (dup;pick(s) : B1 => B1,B1[(n-1]) | Ex: 'A','B','C','D',<4h> => 'A','B','C','D',<4h>,'B'

Les adresses qui suivent permettent d'ajouter un objet situé au niveau 1 de la pile au *bloc* B1 situé au dessus de cet objet (ou même au *bloc* B2 situé au dessus de B1), en première ou en dernière position.

Adresse	Mnémonique puis commentaires. (LM) si langage machine.
#62904h	pushB[1] | (LM) Ajoute obj1 comme nouveau premier élément du bloc B1 dont la longeur augmente d'une unité. | Ex: 'A','B','C',<3h>,'X' => 'A','B','C','X',<4h>
#57441h	pushB[1];swapB | Comme ci-dessus, suivi d'un SWAP sur blocs B2 et B1
#5E401h	pushB[n] | Ajoute obj1 comme nouveau dernier élément du bloc B1 dont la longeur augmente d'une unité. | Ex: 'A','B','C',<3h>,'X' => 'X','A','B','C',<4h>
#5E706h	pushB2[n] | (LM) Comme pushB[n] mais l'élément du niveau 1 est ajouté comme dernier élément du bloc B2. | Ex: 'A','B','C','D',<4h>,'X','Y','Z',<3h>,1789 | ==> 1798,'A','B','C','D',<5h>,'X','Y','Z',<3h>
#5E7A5h	pushB2[1] | (LM) Comme pushB[1], mais l'élément du niveau 1 est ajouté comme premier élément du bloc B2. | Ex: 'A','B','C','D',<4h>,'X','Y','Z',<3h>,1789 | ==> 'A','B','C','D',1789,<5h>,'X','Y','Z',<3h>

Voici comment on peut *découper un bloc* en deux *sous-blocs*, ou comment on peut éliminer les éléments n°1,2,ou3 de ce *bloc*.

Adresse	Mnémonique puis commentaires
#5F996h	CutB(s) | Part du bloc B1, et le découpe en 2 blocs B'2, B'1: | B'2 est formé des des s-1 premiers éléments de B1. | B'1 est formé par le reste des éléments de B1. | Ex: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, <Ah>, <4h> | ==> 3, 2, 1, <3h>, 10, 9, 8, 7, 6, 5, 4, <7h>
#5EAF4h	delB[1] | Détruit le premier élément du bloc B1, dont la | longueur diminue d'une unité. | Ex: 'A','B','C','D',<4h> ==> 'A','B','C',<3h>
#5EB08h	delB[1-2] | Détruit les 2 premiers éléments du bloc B1, | dont la longueur diminue de deux unités. | Ex: 'A','B','C','D',<4h> ==> 'A','B',<2h>
#5CC12h	DelB[1-2] (comme ci-dessus)
#59FB9h	delB[1-3] | Détruit les 3 premiers éléments du bloc B1, | dont la longueur diminue de trois unités. | Ex: 'A','B','C','D','E',<5h> ==> 'A','B',<2h>

Les trois adresses suivantes permettent de transférer un élément du *bloc* B1 vers le *bloc* B2.

Adresse	Mnémonique puis commentaires
#5E4EAh	B2(n)<=B1(1) | Transfère le 1er élément du bloc B1 comme dernier | élément du bloc B2. Les dimensions sont corrigées. | Ex: 'A','B','C','D',<4h>,'X','Y','Z',<3h>, | ==> 'Z','A','B','C','D',<5h>,'X','Y',<2h>
#5E503h	B2(1)<=B1(1) | Transfère le 1er élément du bloc B1 comme premier | élément du bloc B2. Les dimensions sont corrigées. | Ex: 'A','B','C','D',<4h>,'X','Y','Z',<3h> | ==> 'A','B','C','D','Z',<5h>,'X','Y',<2h>
#5FC4Ch	B2(n)<='B1(1)' |Le 1-ier élément du bloc B1 est transféré en dernière | position dans le bloc B2, en même temps qu'il est | transformé en expression par ->expr1. |Ex: 5,<1h>,4,3,2,1,<4h> => '1',5,<2h>,4,3,2,<3h>

LES "MÉTA-OBJETS"

Voici les derniers SYSEVALs agissant sur des *blocs* (en attendant les adresses spécialisées dans le traitement des "*blocs-expressions*").

Evidemment ces adresses n'ont pas un intérêt aussi grand que les précédentes...

Adresse	Mnémonique puis commentaires	
#35222h	dup;roll(s);swap	[o(n),...,o(2),o(1),<n>] ==> [o(n),o(n-2),...,o(1),o(n-1),<n>]
#630F1h	dup;roll(s)	[o(n),...,o(2),o(1),<n>] ==> [o(n),o(n-2),...,o(1),<n>],o(n-1)
#0992Dh	drop;dropB;false	
#1F4CAh	->expr1;PushB[n]	Pour la signification des
#25246h	swapB;drop;2true	
#57414h	drop2;2dropB;false	mnémoniques PullB[1], swapB,
#57428h	drop;2dropB;false	
#5FA63h	3rolld;delB[1];swap	dropB, delB[1], ->expr1, et
#57B33h	+B21;PullB[1];<1h>	
#5843Fh	drop2;delB[1]	pushB[1], on se reportera aux
#5D6F4h	swapB;B2[n]=>B1[1]	
#5E6BBh	PullB[1];<1h>;swapB	pages précédentes.
#5FA45h	delB[1];PullB[1]	
#638FDh	swapB;drop	

LES "BLOCS-EXPRESSIONS"

Le calcul symbolique occupe une place très importante dans le langage de la HP48. Pour faciliter leur manipulation sur la pile-utilisateur, les expressions algébriques sont "*libérées*" en leurs différents éléments.

Le résultat d'une telle opération est un "*bloc*", au sens où nous l'avons vu dans le chapitre précédent.

Le SYSEVAL créant de tels *blocs* (et qui s'applique en fait à tous les *objets-composés*: listes, expressions, Rpl, objets-unités) se trouve à l'adresse **#54AFh** (mnémonique "***breaks(cp)***").

Exemple:

- Considérons l'expression **'SIN(A+B)*EXP(C)'**.
- Elle est formée d'un *opérateur racine* (ici *, un opérateur binaire), appliqué à 2 *arguments* (les sous-expressions **SIN(A+B)** et **EXP(C)**).
- Par **#54AFh SYSEVAL**, cette expression est transformée en:
  8:'A' 7:'B' 6:+ 5:SIN 4:'C' 3:EXP 2:* 1:<7h>
- On obtient donc un *bloc* de longueur 7. On y reconnaît la traduction de cette expression en notation polonaise inverse.

Développer ainsi une expression sous forme de *bloc* ne suffit pas. Encore faut-il des "*outils*" pour manipuler les *opérandes* (arguments) et *opérateurs* (fonctions) qui apparaissent dans ce *bloc*.

Si on reprend l'exemple précédent, le *bloc* obtenu peut s'écrire:

'A' 'B' + SIN	'C' EXP	*	<7h>

On y reconnait l'*opérateur racine* * (le premier élément du *bloc*), le premier *argument* (la séquence **'A' 'B' + SIN**, qui représente **SIN(A+B)**) et le second *argument* (**'C' EXP**, représentant **EXP(C)**).

Il est préférable de poser quelques définitions, pour éclairer l'action des SYSEVALs qui apparaîtront dans les pages suivantes.

On appellera "***bloc-expression***" tout *bloc* (se reporter au chapitre précédent) dont les éléments sont autorisés dans les expressions algébriques (les noms, les nombres, les objets-unités, les fonctions intégrées à la HP48...).

LES "BLOCS-EXPRESSIONS"

On appellera "*argument*" (dans un *bloc-expression*) toute séquence d'objets (éventuellement réduite à un élément) de ce *bloc* qui serait la traduction, en notation polonaise inverse, d'un argument valable d'une expression algébrique.

Un *argument* peut être associé à un niveau logique d'imbriquement dans le bloc-expression dont il fait partie (voir exemple ci-après).

Reprenons l'exemple ci-dessus: **'A' 'B' + SIN 'C' EXP * <7h>** est un *bloc-expression*, qui correspond d'ailleurs exactement à une expression algébrique (**'SIN(A+B)*EXP(C)'**).

Dans ce *bloc expression*, les arguments sont:

au niveau 2: **'A' 'B' + SIN** et **'C' EXP**
au niveau 3: **'A' 'B' +** et **'C'**
au niveau 4: **'A'** et **'B'**

Le *bloc* **'A' 'B' + SIN 'C' EXP <6h>** est encore un *bloc-expression*, même s'il n'est plus la traduction exacte d'une expression algébrique.

Dans ce dernier *bloc-expression*, il y a deux *arguments* au niveau 1: **'A' 'B' + SIN** et **'C' EXP**

If faut encore imaginer une notation pour décrire au mieux ces *blocs-expressions* et leurs *arguments* (notamment pour que les mnémoniques qui sont censés résumer les SYSEVALs soient suffisamment parlants).

- Les lettres **X,Y,Z** désignent des *arguments* au sens précédent.
- Les lettres **f,g** désignent des *opérateurs* (en général binaires ou unaires, c'est-à-dire prenant deux ou un argument(s)).
- Le symbole _ (*souligné*) désignera un argument quelconque, ou une succession (éventuellement vide) d'arguments quelconques (ou même une succession quelconque d'éléments) d'un *bloc*.
- Les deux symboles **[** et **]** montrent les limites d'un *bloc*.

On n'oubliera pas qu'à la base d'un *bloc* (au premier niveau de la pile qu'il occupe) se trouve un entier-système donnant le nombre d'objets dont est constitué le *bloc*.

- Avec ces notations, le bloc **'A' 'B' + SIN 'C' EXP * <7h>** peut être représenté **[X,Y,f]** ou mieux **[f(X,Y)]**, en notant:
  X: l'*argument* **'A' 'B' + SIN**
  Y: l'*argument* **'C' EXP**
  f: l'*opérateur* *****

- Le bloc **'A' 'B' + SIN 'C' EXP <6h>** peut être représenté **[X,Y]** , en donnant le même sens à **X** et à **Y**.

On pourrait très bien ne le noter que **[_,Y]** (**Y** gardant la même signification) si on veut mettre l'accent sur la présence de l'argument **'C' EXP** (en début de bloc).

Il faut garder présent à l'esprit que la HP48 analyse le contenu d'un *bloc-expression* en partant du début de ce *bloc* (et en remontant vers les niveaux supérieurs de la pile utilisateur).

De cette manière, et pour une fonction prenant plusieurs arguments, c'est toujours le dernier argument qui est analysé en premier).

Reprenons l'exemple du *bloc* **'A' 'B' + SIN 'C' EXP * <7h>**, et supposons que la HP48 veuille en comprendre la signification.

- Elle commence par observer la présence de l'opérateur *, dont elle sait qu'il s'agit d'un opérateur binaire.
- Elle réalise ensuite la présence de l'argument **'C' EXP** , et puis seulement celle de l'argument **'A' 'B' + SIN** , qui est pourtant le premier argument dans l'expression initiale **'SIN(A+B)*EXP(C)'**.

Je ne sais pas si ces définitions et explications vous conviennent. Elles ont en tout cas l'avantage de réduire au minimum les commentaires pour nombre des SYSEVALs qui suivent, au profit des exemples.

Commençons par quatre SYSEVALs qui n'utilisent pas ces notations! Ils permettent de créer un *bloc* ou plusieurs *blocs* à partir d'objets quelconques ou d'expressions, et de *concaténer* éventuellement les *blocs* obtenus

Adresse	Mnémoniques et commentaires
#5E067h	toBEX
	si obj1 est une expression, alors la "casse" en expression en plaçant la taille <n> au niveau 1. sinon place <1h> au niveau 1. On crée ainsi un bloc
	Ex: 'A+2*B' ==> 'A',2,'B',*,+,<5h>
	"Hello" ==> "Hello",<1h>
#5E30Ch	to2BEX
	comme toBEX mais avec objets placés aux niveaux 2 et 1. On crée ainsi 2 blocs.
	Ex: 'A+B','SIN(C)' ==> 'A','B',+,<3h>,'C',SIN,<2h>
#5E2F8h	to2BEX+
	comme to2BEX puis concatène les 2 blocs.
	Ex: 'A+B','SIN(C)' ==> 'A','B',+,'C',SIN,<5h>
#5E32Ah	tonBEX+(s)
	comme to2BEX+, mais avec s blocs.
	Ex:'A+B','SIN(C)',*,<3h> => 'A','B',+,'C',SIN,*,<6h>

◆ Cas des fonctions-utilisateurs:

Les fonctions utilisateurs, sont codées de la manière suivante, en prenant l'exemple de **'F(X,Y,Z)'**:

Apply(['A','B','C','f'],<3h>).

Le premier argument est ici le "*bloc*" **[X Y Z 'f']**, où f est une expression réduite à un nom (créée par ***→expr1*** en **#5E652h**) et le deuxième argument est l'entier-système **<3h>** qui donne le nombre d'arguments de **'f'**.

Quand on "*casse*" cette expression sur la pile, au moyen de l'adresse **#054AFh** ("***breaks(cp)***"), on obtient

7:'A' 6:'B' 5:'C' 4:'f' 3:<3h> 2:Apply 1:<6h>

En fait la HP48 n'affiche pas "*Apply*". Le niveau semble vide, mais il est tout de même occupé par un objet de type égal à 18, ce qui caractérise les fonctions intégrées (Attention à ne pas évaluer cette fonction au hasard, car un "*Memory Clear*" peut très bien en résulter).

Vous pouvez également vérifier que, si les objets **'A'**, **'B'**, **'C'** obtenus ci-dessus ont bien le type 6 (*noms globaux*), l'objet **'f'** a le type 9 (*expression*).

Ces quelques remarques vous permettront d'interpréter les résultats de certains des SYSEVALs qui suivent, si vous les appliquez à des *blocs-expressions* représentant des *fonctions-utilisateurs*.

Voici donc toute une collection de SYSEVALs agissant sur des *blocs-expressions*. Les notations ont été définies ci-dessus; les mnémoniques, et les exemples qui les illustrent, devraient éviter toute ambiguité.

Certains des SYSEVALs ci-dessous s'appliquent à un *bloc-expression* résultant de la "*libération*" sur la pile d'une expression algébrique (avec un opérateur racine que j'ai noté **f**).

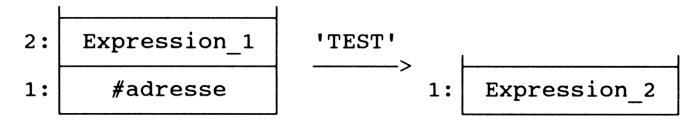
J'ai indiqué à chaque fois si la fonction **f** pouvait ou non être une *fonction-utilisateur*.

Dans le cas où un des SYSEVALs ci-dessous transforme une expression algébrique ("*cassée*" sous forme de bloc) en une autre expression algébrique (au même format), vous pouvez utiliser le programme '**TEST**' suivant pour vérifier le fonctionnement du SYSEVAL sur une expression particulière.

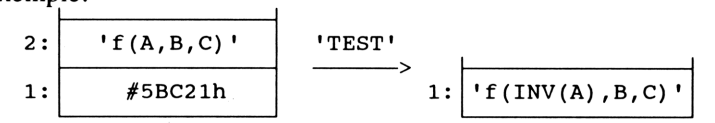
'TEST': (Checksum: #6939h; Taille 70.5 octets)

```
«  → ad
     « #54AFh SYSEVAL ad SYSEVAL #546Dh SYSEVAL »
»
```

Le schéma fonctionnel du programme **'TEST'** est le suivant:



Exemple:



Adresse	Mnémoniques et commentaires
#5BD11h	[X]=>[NEG(X)]! |Applique l'opérateur NEG, puis EXPAND |Ex: 'A','B',+,<3h> => 'A',NEG,'B',-,<4h> |=> 'A','B',+,<3h> |Avec le programme 'TEST' ci-dessus: 'A+B'=>'-A-B' |'A-B'=>'-A+B'; 'A*B'=>'-A*B' et 'A/B'=>'-A/B'
#5BD25h	[X]=>[INV(X)]! |Applique l'opérateur INV, puis EXPAND |Ex: 'A','B',*,<3h> => 'A',INV,'B',/,<4h> |=> 'A','B',*,<3h> |(cad 'A*B' => 'INV(A)/B' => 'A*B')
#5C137h	Expand[_,f] |Applique EXPAND à l'opérateur unaire involutif en |tête du bloc (cet opérateur est NEG, ou INV)

Voici quelques adresses s'appliquant à 2 *arguments* X et Y dans un *bloc-expression*.

Adresse	Mnémoniques et commentaires
#5BC67h	[X,Y]=>[X-Y] (Add-) |Ex: 'A','B',+,'C',EXP,<5h> |=> 'A','B',+,'C',EXP,-,<6h> |Ex: 'A',<1h> => 'A',-,<2h> => 'A',-,-,<3h>
#5BC94h	[X,Y]=>[X+Y] (Add+) |Ex: 'A','B',+,'C',EXP,<5h> |=> 'A','B',+,'C',EXP,+,<6h> |Ex: 'A',<1h> => 'A',+,<2h> => 'A',+,+,<3h>
#5E6F2h	[Y],[_,X]=>[X,Y],[_] |Ex: 'U',SIN,<2h>,'A','B',+,'C',EXP,<5h> |=> 'C',EXP,'U',SIN,<4h>,'A','B',+,<3h> |Cas particulier: [Y],[X]=>[X,Y],[ϕ] |Ex: 'U',SIN,<2h>,'A','B',+,<3h> |=> 'A','B',+,'U',SIN,<5h>,<0h>

LES "BLOCS-EXPRESSIONS"

Les SYSEVALs ci-dessous agissent sur le premier *argument* rencontré dans un *bloc-expression*.

Je rappelle que le symbole _ (*souligné*) désigne ici une séquence quelconque d'arguments (séquence éventuellement vide: j'ai précisé le cas particulier qui en découlait).

La signification attachée à un tel symbole _ est la même avant et après transformation du *bloc-expression* par le SYSEVAL (étudiez les exemples).

Le symbole **[ϕ]** désigne le *bloc vide*, réduit à **<0h>**.

Adresse	Mnémoniques et commentaires
#5BBE5h	[_,X]=>[_,-X] (ChsB) |Ex: 'A','B',+,'C',EXP,<5h> | => 'A','B',+,'C',EXP,NEG,<6h> |Cas particulier: [X]=>[-X] |Ex: -5 <1h> => 5 <1h> => 5 NEG <2h> => 5 <1h> |Ex: A B * <3h> => A B * NEG <4h> => A B * <3h>
#5BC3Fh	[_,X]=>[_,1/X] |Ex: 'A','B',+,'C',EXP,<5h> | => 'A','B',+,'C',EXP,INV,<6h> |Cas particulier: [X]=>[1/X] |Ex: 5 <1h> => 5 INV <2h> => 5 <1h> |Ex: A B * <3h> => A B * INV <4h> => A B * <3h>
#5EA9Fh	[_,X]=>[X],[_] |Ex: 'A','B',+,'C',EXP,<5h> | => 'C',EXP,<2h>,'A','B',+,<3h> |Cas particulier: [X]=>[X],[ϕ] |Ex: 'A','B',+,SIN,<4h> => 'A','B',+,SIN,<4h>,<0h>
#5EAC2h	[_,X]=>[_],[X] |Ex: 'A','B',+,'C',EXP,<5h> | => 'A','B',+,<3h>,'C',EXP,<2h> |Cas particulier: [X]=>[ϕ],[X] |Ex: 'A','B',+,SIN,<4h> => <0h>,'A','B',+,SIN,<4h>
#63F92h	|identique à #5EAC2h ci-dessus

Les SYSEVALs suivants ne font qu'ajouter un des 4 opérateurs +, *, **NEG** ou **INV** en tête du *bloc* présent au niveau 1.

Adresse	Mnémoniques et commentaires
#5CD02h	[_]=>[_,+] | Ajoute l'opérateur + en tête du bloc
#5CD16h	[_]=>[_,*] |Ajoute l'opérateur * en tête du bloc
#5CD2Ah	[_]=>[_,NEG] |Ajoute l'opérateur NEG en tête du bloc
#5CD3Eh	[_]=>[_,INV] |Ajoute l'opérateur INV en tête du bloc

Voici maintenant des SYSEVALs agissant sur un bloc représentant le "*développement*" d'une expression algébrique dont l'opérateur racine est noté ici **f**. Ces SYSEVALs s'appliquent également au premier ou au dernier *argument* (noté ici **X**) de **f**.

Je signale quand **f** est autorisée à être une *fonction-utilisateur*.

Là encore, le caractère "*souligné*" _ désigne une séquence, éventuellement vide, d'arguments de **f**.

Adresse	Mnémoniques et commentaires
#5BC03h	[f(X,_)]=>[f(-X,_)] f peut être une fonction utilisateur. Ex: 'A','B',+,'C',EXP,*,<6h> => 'A','B',+,NEG,'C',EXP,*,<7h> Cas particulier: [f(X)]=>[f(-X)] Ex: 'A',√,<2h> => 'A',NEG,√,<3h> => 'A',√,<2h> Ex: 5,<1h> => 5,NEG,<2h> <=> 5,NEG,NEG,<3h>
#5BC21h	[f(X,_)]=>[f(1/X,_)] f peut être une fonction utilisateur. Ex: 'A','B',+,'C',EXP,*,<6h> => 'A','B',+,INV,'C',EXP,*,<7h> Cas particulier: [f(X)]=>[f(1/X)] Ex: 'A',√,<2h> => 'A',INV,√,<3h> => 'A',√,<2h> Ex: 10,<1h> => 10,INV,<2h> => 10,INV,INV,<3h>
#5CBF9h	[f(_,X)]=>[X,_] Ex: 'A','B',+,'C',EXP,*,<6h> => 'C',EXP,'A','B',+,<5h> Cas particulier: [f(X)]=>[X] Ex: 'A','B',+,SIN,<4h> => 'A','B',+,<3h>
#5E68Eh	[f(X,_)]=>[_,f],[X] Ex: 'A','B',+,'C',EXP,*,<6h> => 'C',EXP,*,<3h>,'A','B',+,<3h> Cas particulier: [f(X)]=>[f],[X] Ex: 'A','B',+,SIN,<4h> => SIN,<1h>,'A','B',+,<3h> Ex: 'A',<1h> => <0h>,'A',<1h>
#63F56h	[f(_,X)]=>[X],[_] Ex: 'A','B',+,'C',EXP,*,<6h> => 'C',EXP,<2h>,'A','B',+,<3h> Cas particulier: [f(X)]=>[X],[ϕ] Ex: 'A','B',+,SIN,<4h> => 'A','B',+,<3h>,<0h>

A partir de maintenant, et pour garder à ce chapitre une importance raisonnable, les exemples vont se faire un peu plus rares.

Voici quatre SYSEVALs qui agissent sur un bloc-expression, de deux manières différentes selon que l'objet à la racine du bloc (donc au niveau 2 de la pile) est ou n'est pas l'opérateur **NEG** (ou l'opérateur **INV**):

Adresse	Mnémoniques et commentaires
#5BC5Dh	[_,NEG]=>[_,-] sinon [_]=>[_,+]
#5BC8Ah	[_,NEG]=>[_,+] sinon [_]=>[_,-]
#5BCB7h	[_,INV]=>[_,/] sinon [_]=>[_,*]
#5BCE4h	[_,INV]=>[_,*] sinon [_]=>[_,/]

LES "BLOCS-EXPRESSIONS"

Les SYSEVALs qui suivent s'appliquent à une expression (développée sous forme de *bloc*) possédant un opérateur racine **f** (obligatoirement une fonction intégrée à la HP48), et contenant deux *arguments* notés **X** et **Y**.

Le résultat est un bloc représentant une autre expression (obtenue, en gros, en remplaçant **f** par un autre opérateur).

Le programme '**TEST**' présenté plus haut de ce chapitre permet de bien se rendre compte du fonctionnement de ces SYSEVALs.

Adresse	Mnémoniques et commentaires
#5BCC1h	[f(X,Y)]=>[X/Y] |f ne peut pas être une fonction utilisateur. |Ex: 'A','B',+,'C',EXP,*,<6h> | => 'A','B',+,'C',EXP,/,<6h> |Ex: 'A',SIN,<1h> => 'A',/,<2h> => 'A',/,<2h>
#5BD89h	[f(X,Y)]=>[X/Y]! |cf ci-dessus + cas particulier: [f(X,1/Y)]=>[X*Y]
#5BCEEh	[f(X,Y)]=>[X*Y] |f ne peut pas être une fonction utilisateur. |Ex: 'A','B',+,'C',EXP,-,<6h> | => 'A','B',+,'C',EXP,*,<6h> |Ex: 'A',SIN,<1h> => 'A',*,<2h> => 'A',*,<2h>
#5BD70h	[f(X,Y)]=>[X*Y]! |cf ci-dessus + cas particulier: [f(X,1/Y)]=>[X/Y]
#5BD3Eh	[f(X,Y)]=>[X+Y]! |Cas particulier: [f(X,-Y)]=>[X-Y]
#5BD57h	[f(X,Y)]=>[X-Y]! |Cas particulier: [f(X,-Y)]=>[X+Y]
#5CC26h	[f(X,Y)]=>[-X-Y]! |Cas particulier: [f(X,-Y)]=>[-X+Y]
#5CC3Fh	[f(X,Y)]=>[-X+Y]! |Cas particulier: [f(X,-Y)]=>[-X-Y]
#5CC8Ah	[f(X,Y)]=>[INV(X)/Y]! |Cas particulier: [f(X,1/Y)]=>[INV(X)*Y]
#5CCA3h	[f(X,Y)]=>[INV(X)*Y]! |Cas particulier: [f(X,1/Y)]=>[INV(X)/Y]

Les SYSEVALs du tableau suivant s'appliquent à une expression (écrite sous forme de *bloc*), où apparaissent deux *opérateurs* **f** et **g** (ayant un ou plusieurs *arguments*, notés **X**, **Y**, **Z**).

Certains arguments sont optionnels, ce qui conduit à de nombreux cas particuliers. J'ai précisé les cas où les fonctions-utilisateurs sont autorisées.

On peut considérer **f** et **g** (quand elles prennent deux arguments) comme la *notation préfixée* d'opérations binaires **&** et **#**. J'ai réécrit les mnémoniques en utilisant ces *notations infixées*.

Adresse	Mnémoniques et commentaires
#5CA0Fh	[f(X,g(Y,Z))]=>[g(f(X,Y),Z)] (ou [X&(Y#Z)]=>[(X&Y)#Z]) g peut être une fonction-utilisateur. Les arguments X et Z pouvent être omis, ce qui conduit aux cas particuliers: [f(X,g(Y))]=>[g(f(X,Y))], [f(g(Y))]=>[g(f(Y))] et [f(g(Y,Z))]=>[g(f(Y),Z)]
#5CA32h	[f(g(X,Y),Z)]=>[g(X,f(Y,Z))] (ou [(X#Y)&Z]=>[X#(Y&Z)]) f peut être une fonction-utilisateur. Les arguments X et Z pouvent être omis, ce qui conduit aux cas particuliers: [f(g(Y),Z)]=>[g(f(Y,Z))], [f(g(Y))]=>[g(f(Y))] [f(g(X,Y))]=>[g(X,f(Y))]
#5CAE1h	[f(g(X,Y),Z)]=>[g(f(X,Z),Y)] Avec notations infixées: [(X#Y)&Z]=>[(X&Z)#Y] f et g peuvent être des fonctions utilisateurs. Y et Z peuvent être omis. L'opération réalisée est involutive. Y et Z peuvent être omis, ce qui conduit aux cas particuliers: [f(g(X),Z)]=>[g(f(X,Z))] [f(g(X,Y))]=>[g(f(X),Y)] et [f(g(X))]=>[g(f(X))]
#5CA50h	[f(X,g(Y,Z))]=>[g(f(X,Y),f(X,Z))] Avec notations infixées: [X&(Y#Z)]=>[(X&Y)#(X&Z)] g doit prendre deux arguments. L'argument X peut être omis, ce qui conduit au cas particulier: [f(g(X,Y))]=>[g(f(X),f(Y))]
#5CAA5h	[f(g(X,Y),Z)]=>[g(f(X,Z),f(Y,Z))] Avec notations infixées: [(X#Y)&Z]=>[(X&Z)#(Y&Z)] Ici f peut être une fonction utilisateur et g doit prendre deux arguments. L'argument Z peut être omis, ce qui conduit au cas particulier: [f(g(X,Y))]=>[g(f(X),f(Y))]
#5CAFFh	[f(g(X,Y),g(X,Z))]=>[g(X,f(Y,Z))]? Avec notations infixées: [(X#Y)&(X#Z)]=>[X#(Y&Z)]? Opération de factorisation. Sous le bloc résultat apparaît un booléen ("true" si une factorisation a eu lieu, "false" sinon).
#5CB7Ch	[f(g(X,Z),g(Y,Z))]=>[g(f(X,Y),Z)]? Avec notations infixées: [(X#Z)&(Y#Z)]=>[(X&Y)#Z]? Opération de factorisation. Voir ci-dessus.
#5CC58h	[f(g(X,Y))]=>[-X-Y]
#5CC71h	[f(g(X,Y))]=>[-X+Y]
#5CCBCh	[f(g(X,Y))]=>[INV(X)/Y]
#5CCD5h	[f(g(X,Y))]=>[INV(X)*Y]
	Pour ces quatre SYSEVALs, g doit être une fonction de deux variables.

◆ Les opérations du menu RULES:

Quand on édite une expression dans l'environnement **EQUATION-WRITER**, il est possible de mettre en surbrillance une partie (un opérateur, un opérande) de cette expression. Dans le même temps, apparaît un menu dont la première entrée conduit à un sous-menu appelé menu **RULES**. Ce menu est décrit dans le chapitre 22 du tome 1 du manuel de la HP48, aux pages 414 et suivantes.

Les SYSEVALs qui suivent correspondent aux différentes options du menu **RULES**. Elles s'appliquent à une expression "*développée*" sous forme de *bloc*. La plupart des opérations ainsi réalisables (factorisations, développements, etc...) ne sont effectives que si le *bloc* représente une expression adéquate (sinon le *bloc* reste inchangé).

Les mnémoniques utilisés ici ont tous la forme **Op[_]**, où "***Op***" est le label correspondant dans le menu **RULES**. Pour plus de détails sur ce menu et ses options, reportez-vous au manuel de la HP48.

Adresse	Mnémoniques et commentaires
#5BE81h	<-->[_] |Applique commutativité (<--> dans menu RULES) |Ex: 'A','B',+,<3h> => 'B','A',+,<3h> |Ex: 'A','B',-,<3h> => 'B',NEG,'A',+,<4h> |Ex: 'A','B',*,<3h> => 'B','A',*,<3h> |Ex: 'A','B',/,<3h> => 'B',INV,'A',*,<4h>
#5BECEh	<-A[_] |Association à gauche (<-A dans menu RULES) |'A','B','C',*,*,<5h> => 'A','B',*,'C',*,<5h> | ce qui correspond à [A*(B*C)]=>[A*B*C]
#5BF53h	A->[_] |Association à droite (A-> dans menu RULES) |Ex: [A*B/C]=>[A*(B/C)]
#5BFD8h	D->[_] |Distribution à droite (D-> dans menu RULES) |ex [LN(A*B)]=>[LN(A)+LN(B)], [LN(A/B)]=>[LN(A)-LN(B)] | [A*(B+C)]=>[A*B+A*C], etc....
#5C0B9h	<-D[_] |Distribution à gauche (<-D dans menu RULES) |Ex: [(A+B)*C]=>[A*C+B*C]
#5C204h	1/()[_] |Double inversion et distribution. |Ex: 'A',EXP,<2h> => 'A',NEG,EXP,INV,<4h> | ce qui équivaut à [EXP(A)]=>[INV(EXP(-A))]
#5C261h	-()[_] |Double signe - et distribution. |Ex: [A+B]=>[-A-B]; [LN(INV(A))]=>[-LN(A)]
#5C2CEh	E^[_] |Remplacement d'un produit de puissances par une | puissance de puissance. Ex: [EXP(A*B)]=>[EXP(A)^B]
#5C31Bh	E()[_] |Remplacement d'une puissance de puissance par un | produit de puissances. Ex: [EXP(A)^B]=>[EXP(A*B)]

Suite du tableau précédent (attention, quelques adresses peuvent conduire à des "*Memory Clear*". Elles sont signalées)

Adresse	Mnémoniques et commentaires
#5C348h	L*[_] |Remplacement du log d'une puissance. |Ex: 'A','B',^,LN,<4h> => A,LN,'B',*,<4h> | cad [LN(A^B)]=>[LN(A)*B]
#5C375h	L()[_] |Remplacement de LN(A)*B par LN(A^B) (idem avec LOG) |Analogue à L() dans le menu RULES. C'est l'inverse L*[_]. Fonctionne aussi avec LN(A)/B et LOG(A)/B. |Ex: A,LN,'B',/,<4h> => 'A','B',INV,^,LN,<5h> |cad : [LN(A)/B]=>[LN(A^INV(B))]
#5C3C2h	<-M[_] |Mise en facteur à gauche. |Ex: [(A*B)+(A*C)]=>[A*(B+C)]; [A+A*B]=>[A*(1+B)] | [LN(A)+LN(B)]=>[LN(A*B)]
#5C4CFh	M->[_] |Mise en facteur à droite. |Ex: [(B*A)+(C*A)]=>[(B+C)*A]
#5C53Ch	AF[_] |Addition de fraction. |Ex: [A+B/C]=>[(A*C+B)/C]; [A/B-C]=>[(A-B*C)/C]
#5C589h	(<-[_] Risque de Memory Clear.... |Développement vers la gauche |Analogue à (<- dans le menu RULES.
#5C5D6h	->)[_] Risque de Memory Clear.... |Développement vers la droite. |Analogue à ->) dans le menu RULES.
#5C623h	(())[_] Risque de Memory Clear.... |Mise entre parenthèses de termes voisins. |Analogue à (()) dans le menu RULES.
#5C670h	->TRG[_] |De EXP vers COS et SIN (voir ->TRG du menu RULES) | Ex: [EXP(A)]=>[COS(A/i)+SIN(A/i)*i] (en mode rad)
#5C68Dh	T->[_] Risque de Memory Clear.... |Déplacement de terme vers la droite. |Analogue à T-> dans le menu RULES
#5C6D9h	<-T[_] Risque de Memory Clear.... |Déplacement de terme vers la gauche. |Analogue à <-T dans le menu RULES
#5C73Dh	->()[_] |Distribution d'une fonction préfixe. |Ex: [IM(A*B)]=>[RE(A)*IM(B)+IM(A)*RE(B)]
#5C845h	->DEF[_] |Définition des fonctions trigonométriques. |Ex: [SIN(A)]=>[EXP(A*i)-EXP(-(A*i)))/(2*i)] |Ex: [ATANH(A)]=>[-LN((1-X)/√(1-X^2))]
#5C91Dh	TRG*[_] |Développement des fonctions trigonométriques. |Ex: [COS(A+B)]=>[COS(A)*COS(B)-SIN(A)*SIN(B)] |Ex: [TAN(X-Y)]=>[(TAN(A)-TAN(B))/(1+TAN(A)*TAN(B))]

◆ Répétition d'opérations du menu RULES:

Certaines opérations du menu **RULES** peuvent être éxécutées de manière répétitive (jusqu'à ce que l'expression à laquelle elles s'appliquent demeure inchangée).

Pour celà on doit préfixer la touche de menu (qui correspond à l'opération à répéter dans le menu **RULES**) par "*shift-right*".

Attention: les mises en garde associées à l'emploi des instructions (**<-[_], ->)[_], T->[_], <-T[_]** précédentes sont encore valables.

Adresse	Mnémoniques et commentaires
#5CDE3h	<-D[_]! |Distribution à gauche répétée.
#5CE15h	<-A[_]! |Association à gauche répétée. |Ex: 'A','B','C','D','E',*,*,*,*,<9h> |==> 'A','B',*,'C',*,'D',*,'E',*,<9h> |cad: [A*(B*(C*(D*E)))]=>[A*B*C*D*E]
#5CE4Ch	A->[_]! |Association à droite répétée.
#5CE83h	->)[_]! |Développement répété sur la droite.
#5CEBAh	(<-[_]! |Développement répété sur la gauche.
#5CEF1h	D->[_]! |Distribution à droite répétée.
#5CF23h	T->[_]! |Déplacement répété de terme sur la droite.
#5CF5Ah	<-T[_]! |Déplacement répété de terme sur la gauche.
#5CF91h	->Expand[_,f]! |Répétition de Expand[_,f] (voir adresse #5C137h)
#5CFC3h	->()[_]! |Distribution répétée d'une fonction préfixe.
#5CFF5h	<-M[_]! |Mise en facteur à gauche répétée.
#5D009h	M->[_]! |Mise en facteur à droite répétée.

Les adresses suivantes complètent le *bloc* présent au niveau 1 par des objets divers (l'action effectuée peut dépendre du mode en cours).

Adresse	Mnémoniques et commentaires	
#28F92h #2911Dh	[_]=>[(,_,)] [_]=>[',_,']	La longueur du bloc augmente de 2.
#281CEh #5DD83h #5DDB0h #5DD9Ch #5DDF6h	Addi*+[_] Addi*[_] Addπ180/*[_] Addπ180/*i*[_] Add180π/*[_]	L'action effectuée ici dépend du mode en cours (rad, deg, ou grad). Par exemple, la séquence π180/* est vide en mode rad, et doit être rem-placée par π200/* en mode grad.

Les SYSEVALs qui suivent utilisent une technique de contrôle de l'éxécution des structures Rpl. Ces problèmes seront amplement développés dans le tome 2.

Disons simplement qu'un mnémonique du type "***Test?:qd/sk***" représente l'action suivante (en notant ***OBJ*** l'objet qui suit l'adresse correspondant à ce mnémonique):

- Evaluation d'un test (précisé dans le mnémonique), qui renvoie une valeur booléenne ("*true*" ou "*false*").
- Si le test a renvoyé la valeur "*true*", alors on quitte le Rpl en cours, puis on évalue ***OBJ***, avant de poursuivre l'éxécution au point courant du Rpl "*Contenant*" le Rpl en cours. (l'abréviation "**dq**" signifie "***do & quit***").
- Si le test a renvoyé la valeur "*false*", alors l'éxécution du Rpl en cours se poursuit après l'objet qui suit ***OBJ*** (ce dernier est donc *sauté*): l'abréviation "**sk**" signifie "*skip*".

- Pour résumer, disons que "***Test?:qd/sk***" siginifie
  Si "*test vrai*" alors "*do & quit*" sinon "*skip*"

Toutes les adresses qui suivent correspondent à des routines écrites en langage machine:

Adresse	Mnémoniques et commentaires	
#5EE10h #5EF2Eh #5EF41h #5EF54h #5EF67h #5EF7Ah #5EF8Dh #5EFA0h #5EFB3h #5EFC6h	Head=Neg?:(chs;dq)/sk Head=+?:dq/sk Head=-?:dq/sk Head=*?:dq/sk Head=/?:dq/sk Head=NEG?:dq/sk Head=INV?:dq/sk Head=^?:dq/sk Head=SQ?:dq/sk Head=Apply?:dq/sk	Ces SYSEVALs lisent le premier objet du boc situé au niveau 1 de la pile, et testent si cet objet est l'un des opéra--teurs +,-(binaire),*,/,NEG, INV,^,SQ ou "Apply" (pour ce dernier, utilisé dans le code des fonctions-utlisateurs, voir le début du chapitre)

Voici pour terminer des SYSEVALs qui utilisent "*l'objet suivant*" (notons le ***OBJ***) ou les deux "*objets suivants*" ***OBJ1*** et ***OBJ2*** dans le "*Rpl en cours*". Ce ou ces objets ne sont pas évalués. Ils sont simplement, par exemple, ajoutés au bloc du niveau 1, puis l'éxécution se poursuit après ***OBJ*** (ou après ***OBJ1*** et ***OBJ2***).

Adresse	Mnémoniques et commentaires	
#5DD65h	AddNext2[_]	2:[_2], 1:[_1] => 2:[_2,OBJ], 1:[_1,OBJ]
#5E51Ch	AddNext[_]	1:[_] => 1:[_,OBJ]
#5E530h	2AddNext[_]	1:[_] => 1:[_,OBJ1,OBJ2]
#5E549h	AddNextToB2	2:[_2] 1:[_1] => 2:[_,OBJ] 1:[_1]
#5E562h	2AddNextToB2	2:[_2] 1:[_1] => 2:[_2,OBJ1,OBJ2] 1:[_1]
#63F01h	[A],[B]=>[A,B,OBJ]	

LES ALARMES

Dans le chapitre consacré au "***répertoire caché***", nous avons étudié la variable **'Alarms'**, qui dans ce répertoire contient la liste des alarmes.

Nous avons vu que dans cette liste, chaque alarme est représentée par une liste **{ bin obj }**, où "***obj***" est l'objet que doit éxécuter l'alarme (par défaut, il s'agit de la chaîne vide), et où "***bin***" est un entier binaire codé sur 24 chiffres hexadécimaux.
Pour désigner cette façon de coder une alarme, nous parlerons de ***format interne***.

Rappelons l'exemple donné pour illustrer ce format:

- Une alarme devant se déclencher le **08/11/1992** à **16h34mn48s**, avec une périodicité de 3 jours définit un entier binaire égal à:
  #00007E9000001D48E37B50000h.
- En mémoire: il est codé

E4420	D1000	00005B73E84D1	000009E7000

  On peut décomposer cet entier binaire en deux champs:

- Le champ **00005B73E84D1** définit l'entier binaire **#1D48E37B50000h** égal au contenu (en nombre de ticks) de l'horloge du système au moment précis où doit se déclencher l'alarme (à ce moment là, c'est l'entier binaire que donnerait l'instruction **TICKS**).

- Le champ **000009E7000** des 11 derniers quartets définit **#7E900000h**, égal à **2123366400** (ticks d'horloge entre deux déclenchements successifs de l'alarme). En effet, avec notre exemple:
  2123366400 = 3*707788800 = 3*24*60*60*8192, ce qui nous donne bien trois jours à raison de **8192** ticks par seconde.

- Inversement, la HP48 utilise un autre format, plus lisible, pour coder une alarme, que nous appelerons ici ***format externe***, et qui est une liste **{ date time obj rep }**. Dans cette liste:

  date est la date de déclenchement (c'est un réel)
  time est l'heure de déclenchement (c'est un réel)
  obj est l'objet à éxécuter par l'alarme.
  rep est l'intervalle de répétition (c'est un réel, exprimant un nombre de ticks d'horloge: **1s=8192 ticks**)
  Pour une alarme sans répétitions, **rep=0**.

Ce format externe est notamment utilisé dans la variable **ALRMDAT**, (créée temporairement par la HP48 pendant la saisie d'une alarme), ou par les instructions **STOALARM** et **RCLALARM**.

Commençons par les adresses contenant les formes internes des instructions **ACK**, **ACKALL**, **RCLALARM**, **DELALARM**, **FINDALARM**, et **STOALARM**.

Adresse	Mnémonique	Commentaires (N.B: ce sont des Rpl)
#0DDA8h	AckAll	Forme interne de ACKALL
#0DDC1h	ack?	=> 1:true si aucune alarme échue. sinon 1:false et accuse réception de la plus ancienne alarme échue
#0E3DFh	RclAlarm(r)	Forme interne de RCLALARM
#0E510h	StoAlarm(r)	Forme interne de STOALARM
#0E54Dh	StoAlarm(li)	Idem. "li" = alarme au format externe
#0E724h	DelAlarm(r)	Forme interne de DELALARM
#0EAD7h	FindAlarm(r)	Forme interne de FINDALARM
#0EB31h	FindAlarm(li)	idem. "li" au format { date time }

Les adresses suivantes concernent la gestion (création, recherche, élimination, etc...) des alarmes au niveau interne (dans la variable '**Alarms**' du *répertoire caché*), ou effectuent la traduction d'une alarme du type *externe* au type *interne*, ou vice-versa.

Adresse	Mnémoniques et commentaires
#0CB25h	DelAlarms |Supprime toutes les alarmes.
#0DEACh	InsAlarm(2li) |L1 est une alarme au format interne. |L2 est une liste de telles alarmes. |Insère L1 à sa position correcte dans L2. |Le résultat est 2:s 1:L1', où L1' est la nouvelle liste L1 et "s" est la position d'insertion.
#0DECFh	NewAlarms(li) |"li" est une liste d'alarmes au format interne. Elle devient la liste des alarmes.
#0DEF7h	StoAlarms(_) |Place l'objet au niveau 1 dans la variable 'Alarms' du répertoire caché.
#0DF1Eh	RclAlarms? |Rappelle le contenu de la variable 'Alarms' (réper- -toire caché). Donne 2:contenu 1:true ou 1:false
#0E141h	ExecAlarm(r) |Duplique le réel r, puis éxécute l'alarme N°r.
#0E1D8h	ToStdAlarm(li) |Transforme une alarme format interne (représentée par "li") en cette alarme au format externe.
#0E235h	RclAlarms |(LM) Rappelle la liste de toutes les alarmes. |Les alarmes sont au format interne.
#0E27Dh	IndexPastDue(li) |(LM) "li" est une liste d'alarmes au format interne. |Donne 2:li 1:s, où s est l'indice (entier-système) de la plus ancienne des alarmes échues. |Le résultat est <0h> s'il n'y en n'a pas.

La suite du tableau précédent....

Adresse	Mnémoniques et commentaires
#0E3A6h	pastdue(li)? |(LM) Teste si l'alarme "li" (format interne) est | échue. Réponse True ou False.
#0E402h	alarm(s)? |Vérifie s'il existe une alarme d'indice s |Réponse 1:false ou 2:list,1:true (dans ce cas | "liste" est au format interne)
#0E5EFh	StoAlarm(2r,_,r) |Stocke une nouvelle alarme à partir des éléments | suivants 4:r=date 3:r=time 2:any 1:r=rpt |Les vérifications sont faites.
#0E6EDh	StoAlarm(2r,_,r)! |Stocke une nouvelle alarme à partir des éléments | suivants 4:r=date 3:r=time 2:any 1:r=rpt |Aucune vérification n'est faite.
#0E76Fh	DelAlarm(s) |Supprime l'alarme d'index s (entier-système) |Ne teste pas la validité de l'entier s.
#0E78Dh	DelAlarm(0) |Supprime toutes les alarmes (<=> 0 DELALARM)
#0E925h	OldestPastDue(li)? |(LM) "li" est une liste d'alarmes au format interne. |Répond false si aucune alarme n'est échue. |Sinon répond 2:liste 1:true, où "liste" est la plus | ancienne des alarmes échues (au format interne).
#0E9B2h	NextAlarm(li)? |(LM) Prend une liste d'alarmes au format interne et | répond false si aucune n'est "à venir". Sinon | répond 2:liste 1:true, où "liste" est la plus | proche des alarmes "à venir" (format interne)
#0EB6Dh	FindAlarm! |Appelle "FindAlarm(b)" en #0EBD5h, avec b = TICKS |On obtient ainsi l'index de la 1ère alarme | due à partir de l'instant présent.
#0EBA8h	Alarm(s=>r) |Transforme un entier-système s en un réel r. |Mais si s>(nombre d'alarmes) alors r=0.
#0EBD5h	FindAlarm(b) |Donne l'index s (sous forme de short integer) de la | première alarme due après la date indiquée par | le binaire b (représentant une date) |S'il n'y en n'a pas, alors s=(nombre d'alarmes+1)
#0EBEEh	alarm<>{}? |Vérifie si le contenu de 'Alarms' est <> de {}.
#0EC07h	FindAlarm(li,b) |(LM) Comme ci-dessus, mais attend une liste des | alarmes (format interne) au niveau 2.
#0EF45h	ToIntAlarm(4_) |Crée une alarme au format interne (une liste) | à partir de 4:obj 3:r=date 2:r=time 1:r=ticks, |La liste est placée au niveau 1 (alarme non armée)

Voici maintenant des SYSEVALs traitant de la saisie d'une nouvelle alarme dans le menu **TIME**.

Rappelons que cette saisie s'accompagne de la création d'une variable temporaire '**ALARMDAT**' (ou de son utilisation si une sortie prématurée de l'environnement **TIME\ALRM** ne l'a pas purgée).

Les adresses du tableau ci-dessous correspondent toutes à des Rpl.

Adresse	Mnémoniques et commentaires
#19A2Ch	Alarmdat(a/pm) |Effectue a/pm sur l'heure située dans ALARMDAT
#19A68h	RclAlarmdat |Rappelle sur la pile le contenue d 'ALARMDAT', ou | si ce nom n'existe pas, la liste { date 0 "" 0 } | où "date" est la date du jour.
#19B06h	set |Place le contenu de ALARMDAT dans liste des alarmes | puis purge ALARMDAT, avant de nouveau écran TIME
#19BE8h	>time(r) |Place le réel r en position 2 dans la liste ALARMDAT
#19C1Fh	>date(r) |Place le réel r en position 1 dans la liste ALARMDAT
#19C56h	WeekRpt(r) |Intervalle de répétition de r semaines dans ALARMDAT
#19C88h	DayRpt(r) |Intervalle de répétition de r jours dans ALARMDAT
#19CBAh	HourRpt(r) |Intervalle de répétition de r heures dans ALARMDAT
#19CECh	MinRpt(r) |Intervalle de répétition de r minutes dans ALARMDAT
#19D1Eh	SecRpt(r) |Intervalle de répétition de r secondes dans ALARMDAT
#19C06h	>time(r)! | Sous-programmes appelés par les SYSEVALs
#19C3Dh	>date(r)! | ci-dessus. Ici, on ne vérifie plus la
#19C74h	WeekRpt(r)! | présence d'un argument, ni même que cet
#19CA6h	DayRpt(r)! | argument est un nombre réel.
#19CD8h	HourRpt(r)! | Pour ">time(r)!" et ">date(r)!", on teste
#19D0Ah	MinRpt(r)! | cependant la validité de ce réel comme
#19D3Ch	SecRpt(r)! | heure ou comme date.
#19D50h	*rpt(2r) |Effectue le produit des 2 réels pour définir le | nombre de ticks de l'intervalle de répétition
#19D73h	exec(_) |Définit l'objet à éxécuter par alarme ALARMDAT
#19D8Ch	┌─>exec |Rappelle l'objet à éxécuter par alarme ALARMDAT
#19DAAh	PutAlarmdat(_,s) |Place l'objet niveau 2 en position s dans ALARMDAT. | ("s" entier-système). Appelle ensuite "InputAlarm"

LES ALARMES

Tout au long de la procédure de création d'une alarme, un certain nombre de messages peuvent être affichés. Voici les SYSEVALs qui effectuent ces affichages (tous des Rpl).

Adresse	Mnémoniques et commentaires
#19DE2h	DispAlarm |Affiche le message concernant les alarmes obtenu | quand on rentre dans le menu TIME, à condition | par exemple qu'on ne soit pas en mode PRG, ou | qu'il n'y ait pas de touche dans le buffer...
#19DF6h	DispAlarm! |Comme le précédent, mais sans test préalable.
#48F72h	DispAlarm&Date |Comme "DispAlarm", plus la date et l'heure.
#19E0Fh	DispDueAlarm(li) |Affiche "Past Due Alarm" puis le contenu de l'alarme | définie par "li" (format interne). Pas de Freeze.
#19E50h	DispNextAlarm(li) |Affiche "Next Alarm" puis le contenu de l'alarme | définie par "li" (format interne). Pas de Freeze
#19E9Bh	DispAlarm(li,st) |Affiche le message st, puis le contenu de l'alarme | définie par "li" (liste au format externe). |Il n' y a pas de Freeze de l'écran.
#1A031h	InputAlarm |Affiche contenu ALARMDAT et invite à saisir une | nouvelle alarme. Effectue néanmoins quelques tests | préalables (mode prg, buffer clavier non vide ...)
#1A040h	InputAlarm! |Comme le précédent, sans les tests préalables.
#1A06Dh	DispSelRepInt |Affiche mess. "Select Repeat Interval", puis Freeze.
#1A0E0h	DispAck |Affiche le message "Acknowledged", pendant une sec.
#47AE7h	AlarmCat |Lance le catalogue des alarmes

LE MENU STAT

Le menu **STAT** est consacré aux calculs statistiques. La matrice **ΣDAT** joue ici un rôle central, de même que la liste placée dans la variable **ΣPAR**, et qui contient les paramètres utiles aux calculs d'*ajustements*.

Nous verrons ici les formes internes des instructions du menu **STAT**, ou les SYSEVALs qui s'appliquent aux contenus de **ΣDAT** et de **ΣPAR**.

Les adresses suivantes permettent de créer la matrice **ΣDAT**, ou de la modifier si elle existe déjà.

Adresse	Mnémoniques et commentaires
#2C1F3h	StoΣ |Comme STOΣ, mais ne vérifie pas la présence d'un argument et ne le "détaggue" pas.
#2C216h	Sto@Σ |Le stockage se fait directement dans ΣDAT si elle ne contient pas un nom, ou dans la variable 'nm' si ΣDAT contient le nom 'nm' déjà existant.
#2C2E8h	StoΣ(r) |Stocke le réel dans ΣDAT, sous forme [[r]].
#2C22Fh	clΣ |Forme interne de CLΣ. Purge le nom 'ΣDAT'.
#2C2D9h	Σ+(r) | Ajoute un ou plusieurs réels à ΣDAT
#2C32Eh	Σ+(ar) | Ajoute un tableau à ΣDAT
#2C423h	σ- | Forme interne de Σ-
#4A702h	DoNewΣdat(ar) |Procédure de saisie d'une nouvelle matrice ΣDAT, qui équivaut à un appui sur la touche NEW.

Voici quelques SYSEVALs réalisant des affichages obtenus dans les différentes pages du menu **STAT**.

Adresse	Mnémoniques et commentaires
#4A4E2h	DispΣDat |Affichage obtenu quand on entre dans menu STAT (les 2 dernières lignes de ΣDAT, ou le message "No current data point...", ou le mode d'ajustement en cours), puis gèle la ligne d'état.
#4A4F1h	DispΣDat! |Comme ci-dessus, sans tests (buffer clavier, etc...)
#4A663h	->StrΣdatStatus |Donne une chaîne contenant les informations sur Xcol Ycol, et le modèle d'ajustement en cours.
#47BA5h	StatCat |Lance le "catalogue statistique".

LE MENU STAT

Les adresses qui suivent vérifient l'existence d'une matrice statistique (de plusieurs manières), en testent ou en rappellent le contenu.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#2C270h	Exist@ΣDAT? |Répond False si 'ΣDAT' n'existe pas ou si elle | contient un nom n'existant pas. |Sinon répond 2:matrice 1:true
#2C293h	ExistΣDAT? |Répond 2:ΣDAT,1:true ou 1:false
#2C2ACh	rclΣ! |Rappelle le contenu de ΣDAT, quelqu'il soit. |Erreur éventuelle "Non Existent ΣDAT"
#2C2C5h	rclΣ |Comme ci-dessus, suivi d'une vérification | de la validité du contenu de ΣDAT.
#2D271h	ChkΣData(_) |(LM) Vérifie si l'objet au niveau 1 peut être un | contenu correct de ΣDAT. Si la réponse est non, | alors erreur "Invalid Σ data". |L'objet reste sur la pile après le test.
#4A555h	RclΣDAT? |False si Σdat n'existe pas. Sinon 3:name 2:mat 1:true | où "name" est le nom 'ΣDAT' ou le nom contenu dans | ΣDAT et qui renvoie à une matrice.

Voici maintenant les formes internes de plusieurs instructions figurant dans les différentes pages du menu **STAT**.

Adresse	Mnémoniques et commentaires	
#2C09Fh	utpn(3r)	Forme interne de UTPN
#2C149h	utpc(2r)	Forme interne de UTPC
#2C174h	utpf(3r)	Forme interne de UTPF
#2C19Ah	utpt(2r)	Forme interne de UTPF
#2C535h	nΣ	Forme interne de NΣ
#2C558h	maxΣ	Forme interne de MAXΣ
#2C571h	mean	Forme interne de MEAN
#2C58Ah	minΣ	Forme interne de MINΣ
#2C5BCh	tot	Forme interne de TOT
#2C5A3h	sdev	Forme interne de SDEV
#2C5D5h	var	Forme interne de VAR
#2C84Bh	corr	Forme interne de CORR
#2C8F5h	cov	Forme interne de COV
#2C94Fh	Σx	Forme interne de ΣX
#2C963h	Σy	Forme interne de ΣY
#2C977h	Σx^2	Forme interne de ΣX^2
#2C99Ah	Σy^2	Forme interne de ΣY^2
#2C9BDh	Σx*y	Forme interne de ΣX*Y
#2CA30h	lr	C'est LR mais sans les tags
#2CB02h	predy(r)	Forme interne de PREDY
#2CB75h	predx(r)	Forme interne de PREDX

Terminons par des SYSEVALs qui créent, modifient, rappellent ou utilisent le contenu de la variable **ΣPAR** (quand cette variable doit être utilisée et qu'elle n'existe pas, elle est automatiquement créée avec ses valeurs par défaut).

Adresse	Mnémoniques et commentaires
#20234h	StoModel |Place objet niveau 1 en 5ème position dans ΣPAR (la | position du modèle d'ajustement).
#2C684h	colΣ | Forme interne de COLΣ
#2C6A2h	StoΣPAR(5_) |Effectue 5 ->LIST puis 'ΣPAR' STO
#2C6C5h	xcol(r) | Forme interne de XCOL
#2C6DEh	ycol(r) | Forme interne de YCOL
#2C701h	┌─>xcol | Rappelle XCol au niveau 1
#2C715h	┌─>ycol | Rappelle YCol au niveau 1
#2C72Eh	ChkBreakΣPAR |Place et teste les objets de ΣPAR sur la pile.
#2C751h	ResetΣPAR |Place liste standard dans ΣPAR et au niveau 1.
#2C7F6h	RclXYCol&ΣDAT |Donne 3:<xcol> 2:<ycol> 1:ΣDAT, où <xcol>,<ycol> | sont des entiers-système. Effectue un test sur | le contenu de ΣPAR et de ΣDAT.
#2C81Eh	RclXYCols&ΣDAT! |Comme ci-dessus, mais erreur si NΣ=1
#2C9EAh	RclYCol&ΣDAT |Donne 2:<ycol> 1:ΣDAT (voir #2C7F6h).
#2C9FEh	RclXCol&ΣDAT |Donne 2:<xcol> 1:ΣDAT (voir #2C7F6h).
#2CC2Eh	PwrExpFIT? |Teste si le modèle d'ajustement en cours est | POWERFIT ou EXPFIT. Répond True ou False
#2CC74h	PwrLogFIT? |Comme ci-dessus avec POWERFIT et EXPFIT
#4A16Ch	Σline | Forme interne de ΣLINE
#4A122h	PlotΣLine | Trace le résultat de Σline.
#4A7CAh	bestfit | Forme interne de BESTFIT
#4C8F4h	bins(3r) | Forme interne de BINS
#50DCDh	StatPlotMode? |Teste si le mode de tracé en cours est SCATTER, | HISTOGRAM, ou bien BAR.

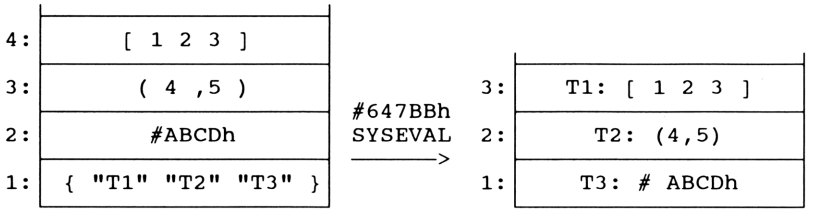
OBJETS-TAGGUÉS

Il y a peu de SYSEVALs qui s'appliquent spécifiquement aux ***objets taggués***. Sans plus de cérémonie, les voici:

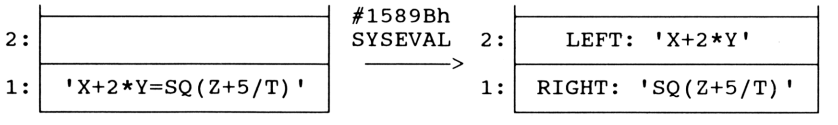
Adresse	Mnémonique	Commentaire (LM si langage machine)
#05E81h	->tag!(_,st)	Ne teste pas longueur de st
#05E9Fh	{nm,_}=>nm:_	(LM) Ex: 1: { ABC 1.234 } => 1: ABC: 1.234
#05F2Eh	->tag(_,nm)	Taggue par un nom
#225F5h	->tag(_,st)	Refuse de tagguer si size(st) > 255
#22618h	->tag(_,r)	Taggue par un réel
#647BBh	->mtag(_,li)	Multiple ->tag (voir plus loin)
#05EC7h	obj->(to)	forme interne de OBJ-> sur taggués
#05EEAh	{:tag:_}=>{tag,_}	(LM) Ex: 1: { :ABC: 1.234 } => { ABC 1.234 }
#64775h	dtag	forme interne de dtag
#647A2h	dtag2	dtag sur level 2
#1589Bh	2tag(eq)	OBJ-> sur équation. Puis taggue les deux membres respectivement par LEFT et RIGHT
#1EA44h	==1tag	== entre un taggué et un objet quelconque
#1EB65h	<>1tag	<> entre un taggué et un objet quelconque

Exemples:

- L'adresse **#647BBh** permet d'effectuer plusieurs opérations de "***tagguage***" simultanément. On doit placer les différents ***tags*** à utiliser dans une liste au niveau 1 (ce doit être des chaînes), les objets à ***tagguer*** devant se trouver aux niveaux supérieurs.



- L'adresse **#1589Bh** est évidemment utilisée dans l'environnement **SOLVER** pour décrire le contenu de l'équation en cours.



HEURE ET DATE

Ce chapitre est consacré aux SYSEVALs qui participent aux opérations sur la date et l'heure.

Le tableau ci-dessous contient des adresses qui correspondent aux formes internes d'instructions programmables du menu **TIME**. Elles ont en commun de ne pas modifier la date et l'heure courantes.

Adresse	Mnémonique	Type	Commentaires
#0CBFAh	time	LM	Forme interne de TIME
#0CC0Eh	date	LM	Forme interne de DATE
#0CC39h	ddays(2r)	LM	Forme interne de DDAYS
#0D304h	tstr(2r)	Rpl	Forme interne de TSTR
#0EB81h	ticks	Rpl	Forme interne de TICKS
#2A673h	->hms(r)	Rpl	Forme interne de ->HMS
#2A68Ch	hms->(r)	Rpl	Forme interne de HMS->
#2A6A0h	hms+(2r)	Rpl	Forme interne de HMS+
#2A6C8h	hms-(2r)	Rpl	Forme interne de HMS-

Voici maintenant des SYSEVALs qui modifient l'heure ou la date en cours. En principe, une telle modification s'accompagne d'une vérification de la liste des alarmes (pour savoir si certaines alarmes doivent maintenant être considérées comme "*past due*", par exemple). Tel est bien le cas pour les adresses de la première partie du tableau ci-dessous.

Au contraire, dans la seconde partie, on trouve des SYSEVALs (tous en langage machine) qui modifient la date ou l'heure sans examiner ensuite les alarmes éventuelles.

Adresse	Mnémonique	Type	Commentaires
#0CC5Bh	date+(2r)	Rpl	Forme interne de DATE+
#0CC79h	date+(r,n)	LM	Appelé par date+(2r). Ajoute n jours. n est entier
#0CD2Bh	->date(r)	Rpl	Forme interne de ->DATE
#0CD3Fh	clkadj(r)	Rpl	Forme interne de CLKADJ
#0CD53h	->time(r)	Rpl	Forme interne de ->TIME
#19A54h	a/pm	Rpl	Bascule entre am et pm (ajoute 12 heures sans changer le jour)
#19B43h	hr+	Rpl	Ajoute 1 heure à l'heure courante
#19B5Ch	hr-	Rpl	Enlève 1 heure à l'heure courante
#19B7Ah	min+	Rpl	Ajoute 1 min. à l'heure courante
#19B93h	min-	Rpl	Enlève 1 min. à l'heure courante
#19BB1h	sec+	Rpl	Ajoute 1 sec. à l'heure courante
#19BCAh	sec-	Rpl	Enlève 1 sec. à l'heure courante
#0CD67h	->date(r)!	LM	Comme les adresses ci-dessus, mais ne vérifient pas la liste des alarmes après modification de l'heure ou de la date
#0CDA6h	->time(r)!	LM	
#0CE0Fh	clkadj(r)!	LM	
#0CDD0h	a/pm!	LM	

L'instruction **TICKS** fournit un entier binaire (de 16 chiffres hexas) donnant le nombre de *ticks d'horloge* correspondant à la date et à l'heure en cours (en partant du 1er jour de notre ère, et ce à raison de 8192 ticks par seconde !)

De même, la HP48 utilise un entier binaire de 24 chiffres hexadécimaux pour coder les alarmes de façon interne (voir le chapitre, dans la première partie du livre, consacré au répertoire caché, où est décrit le contenu de la variable '**Alarms**').

Certaines des adresses ci-dessous utilisent un entier binaire représentatif d'une date. Le format de ce binaire est quelconque, sauf pour l'adresse **#0D169h** ("***RealRpt(b)***").

Les résultats obtenus dessous dépendent parfois du mode en cours de représentation de l'heure et/ou de la date.

Adresse	Commentaires (LM si langage machine)
#0CF5Bh	DayString(b) | (LM) idem ci-dessous, avec un binaire.
#0D2F0h	DayString(r) | Donne "SUN", ou "MON", ou "TUE", etc...
#0CFD9h	DateString(r) | (LM) Ex: 23.051992 => "23.05.92"
#0D06Ah	HourString(r) | (LM) Ex: 18.2345 => "06:23:45P"
#0D143h	RealHour(b) |(LM) Extrait la partie heure (au format réel), d'un binaire. C'est l'inverse de "TicksHour(r)" |Ex: #1D48147BD01E7h => 4.26160594482 |Ex: #7CD01E7h => 4.26160594482
#0D156h	RealDate(b) |(LM) Extrait la partie heure (au format réel), d'un binaire obtenu par TICKS |Ex: #1D48147BD01E7h => 22.081992
#0D169h	RealRpt(b) |b doit être ici la définition interne d'une alarme |Donne un réel égal au nombre de ticks d'horloge entre 2 répétitions de l'alarme (0 si pas de rép.)
#0D349h	tstr(b) |Transforme un binaire (tel qu'obtenu par TICKS) en la chaîne donnant le jour la date et l'heure, au format de l'instruction TSTR |Ex: #1D48147BD01E7h => "SAT 22.08.92 04:26:16A"
#0EE50h	TicksDate(r) |Convertit une date au format réel à sa forme binaire |On obtient un binaire codé sur 13 chiffres. |Ex: 22.081992 => #1D4813FF00000h
#0EE83h	TicksHour(r) |Convertit une heure (format réel) en sa forme binaire |On obtient un binaire codé sur 13 chiffres. |Ex: 4.26160595703 => #7CD01E7h
#0EE26h	AddHour(2b) |Combine deux binaires: b2 ,obtenu par "TicksDate(r)" et b1 obtenu par "TicksHour(r)". |Ex: #1D4813FF00000h, #7CD01E7h => #1D48147BD01E7h |Il n'y a pas de Newob. Le résultat est placé sur b2. |On obtient ainsi la forme binaire d'une date donnée, en tenant compte de l'heure à cette date.

Terminons par un *cocktail* de SYSEVALs, où il est beaucoup question de savoir *si* la date et l'heure sont affichées, *où* elles le sont, et *comment* elles le sont.

Adresse	Commentaires (LM si langage machine)
#0E630h	chkdate(r) |(LM) vérifie la validité du réel r comme date. |Erreur éventuelle "Invalid Date", sinon r est laissé
#0E66Ah	chktime(r) |(LM) vérifie la validité de r en tant qu'heure |Erreur éventuelle "Invalid Time", sinon r est laissé
#0D2D5h	clock? |Répond true si horloge affichée, false sinon
#199EBh	clk |Bascule entre affichage et non affichage de l'horloge.
#19A04h	m/d |bascule entre mode m/d et mode d.m
#19A18h	12/24 |bascule entre mode 0..12 et mode 0..24
#19ADEh	12<->24(r) |Conversion entre horaire0-24 et horaire0-12 |Utilisé dans l'instruction "a/pm" |Exemple: 3 => 15 => 3 => 15 etc...
#19EF5h	TicksStr(r) | Donne une chaîne du type "Rpt=....", en interprétant r comme un délai de répétition en ticks.
#0E006h	DispTstr(b) |Affiche la chaîne au format de TSTR (cf ci-dessus) sur l'écran en cours (Stack ou PICT), ligne 8 (sur 64). Il n'y a pas de "freeze". Essayer par exemple: «{#0 #0} PVIEW DO TICKS #E006h SYSEVAL UNTIL 0 END»
#398F4h	DispDate! | Comme "DispDate" en #39A83h, sauf que la zone qui est recouverte a la largeur de l'écran. En particulier, sur l'écran"stack", le chemin en cours est effacé.
#39A83h	DispDate |Affiche sur l'écran en cours (stack ou PICT) date&time correspondant au moment précis où cette instruction est éxécutée. Pas de FREEZE. Essayer par exemple: « {#0 #0} PVIEW #39A83h 7 FREEZE » |La zone recouverte par l'affichage est juste assez grande pour contenir la date et l'heure.
#53A9Eh	Clock? |Teste si la date et l'heure sont affichées.
#53AACh	DoClock |(LM). Affiche durablement la date et l'heure, sans pour autant modifier de flag.
#53ABAh	NoClock |(LM). Efface affichage date et heure s'il a été obtenu comme ci-dessus.

OBJETS-UNITÉS

Nous avons déjà vu des adresses où interviennent les ***objets-unités***:

- Les objets-unités sont des cas particuliers d'objets composés.
- De nombreuses opérations entre une expression et un autre argument ont la même adresse selon que ce deuxième argument est un réel ou un objet-unité.

Dans ce chapitre nous allons voir les SYSEVALS qui s'appliquent spécifiquement aux *objets-unités*.

Commençons par les adresses des opérations mathématiques courantes sur les *objets-unités*. Les mnémoniques tiennent ici lieu de commentaire.

Adresse	Mnémonique	Adresse	Mnémonique	Adresse	Mnémonique
#0F584h	==(2uo)	#0F598h	<>(2uo)	#0F5ACh	<(2uo)
#0F5C0h	>(2uo)	#0F5D4h	≤(2uo)	#0F5E8h	≥(2uo)
#0F5FCh	abs(uo)	#0F615h	neg(uo)	#0F62Eh	sin(uo)
#0F660h	cos(uo)	#0F674h	tan(uo)	#0F6A2h	+(2uo)
#0F774h	-(2uo)	#0F792h	*(2uo)	#0F823h	/(2uo)
#0F841h	inv(uo)	#0F878h	^(2uo)	#0F8FAh	xroot(2uo)
#0F913h	sq(uo)	#0F92Ch	√(uo)	#0F945h	ubase(uo)
#0FB6Fh	max(2uo)	#0FB8Dh	min(2uo)	#0FBABh	%(uo,r)
#0FC3Ch	%ch(2uo)	#0FCCDh	%t(2uo)	#0FCE6h	sign(uo)
#0FCFAh	ip(uo)	#0FD0Eh	fp(uo)	#0FD22h	floor(uo)
#0FD36h	ceil(uo)	#0FD68h	rnd(uo,r)	#0FD8Bh	trunc(uo,r)

Remarque: certains des SYSEVALs qui suivent nécessitent une bonne compréhension de la structure interne des *objets-unités*. On se reportera donc si nécessaire au chapitre où cette structure est décrite.

Voici donc quelques SYSEVALs s'appliquant aux *objets-unités*. Certains d'entre eux sont accompagnés de commentaires détaillés.

Adresse	Commentaires (LM si langage machine)
#0F371h	convert(2uo) |Forme interne de CONVERT
#0F33Ah	->unit(r,uo) |Forme interne de ->UNIT.
	|Appelle successivement les | adresses "units(uo)" puis "->unit(_,s)" ci-dessous |Ex: 2: 151 1: 3_m/s => 1: 151_m/s
#1008Dh	units(uo)=>(_,s) |Extrait et casse la partie unités. | s = entier système = taille de la partie unités |Ex: 151_m/s^2 => "m", "s", 2, { }, { }, <5h>
#100D3h	->unit(_,s) |Reconstitue uo; s=taille partie unités. A utiliser | après l'adresse ci-dessus, car les listes vides | doivent être basées en ROM !!

Adresse	Commentaires (LM si langage machine)
#0F34Eh	obj->(uo) |Une des formes internes de OBJ->
#0C2CBh	dup;stdunit(st)?
	|(LM) Vérifie si la chaîne est l'une des unités
	| standards de la 48. Résultat booléen.
#0F075h	st=>uo?
	|Tente de transformer une chaîne en un objet-unité
	| de partie scalaire égale à 1. La réponse est:
	| 2:Unit, 1:true ou 1:false.
	|Ex: "M" => 1_M,True "123" => False
#0F218h	uo=>st |Ex: 9.81_m/s^2 => "9.81_m/s^2"
#0F0EDh	makeuo(r,st)!
	|Crée un objet unité de partie scalaire égale à r,
	| et de partie unité réprésentée par st.
	| Le résultat est 2:Objet-unité 1:True
	|On peut ainsi créer des unités exotiques.
	|Ex: 1789,"***" => 1789_***,True
#0F561h	val(2uo)
	|Valeurs de uo2 et de uo1, converties à l'unité de uo1
	|Ex: 15_m/s, 28_kph => 54, 28 (car 15_m/s = 54_kph)
#0FDAEh	(ang/2π)(uo)
	|Donne un réel égal à la proportion de 2π définie
	| par "uo", qui doit être un angle.
	|Ex: 180_° => .5 500_grad => 1.25
#0FE44h	decomp(uo)
	|1: obj_unité => 3:rr 2:rr 1:b, avec:
	| rr3 = partie scalaire (c'est un réel long)
	| rr2 = facteur de conversion aux unités de base
	| b est un binaire s'écrivant #abcdefghijklmnop
	| avec ab:puissance en _? cd:puissance en _mol
	| ef:puissance en _cd gh:puissance en _K
	| ij:puissance en _s kl:puissance en _A
	| mn:puissance en _m op:puissance en _kg
	|Ces puissances sont signées, de -128 à 127
	|Les binaires obtenus pour deux objets-unités sont
	| égaux <=> leurs unités sont consistantes.
	|Ex: 25_knot => 3:%%25, 2:%%.514444444444, 1:binaire,
	| avec binaire = #FF000100h = #00000000FF000100h
	| (puissance en s: #FFh=-1; puissance en m:#01h=1)
	| et 1_knot = .514444444444_m/s.
#1003Dh	dup;uo=>r |idem ci-dessous, mais duplique avant
#10047h	uo=>r |valeur numérique de l'objet-unité
#10065h	(uo,nb)=>uo
	|Ex: 2:1_kph 1:1789 => 1:1789_kph
#197C8h	ufact(2uo) |Forme interne de UFACT
#1CA3Ah	size(uo) |Une des formes internes de SIZE
#412DDh	->StrUnit(uo)
	|Ex: 178_m/s^2 => "m/s^2"

OPÉRATIONS D'IMPRESSION

La HP48 peut transmettre un certain nombre de données, par la voie infra-rouge, à l'*imprimante thermique* **HP82240B** (ou la **HP82240A**, plus ancienne). On peut ainsi imprimer le contenu de la pile (ou uniquement du niveau 1), le contenu de l'écran **LCD**, celui d'une variable...

Les opérations d'impresssion ne se limitent pas à la voie infra-rouge puisque l'on peut transmettre des informations à une imprimante via le *port série*.

Ce chapitre est consacré aux SYSEVALs réalisant de telles opérations d'impression, toutes regroupées dans le menu **PRINT**.

Rappelons que la liste stockée dans la variable **PRTPAR** (dans le répertoire **HOME**) contient les paramètres d'impression. Son format est:

{ delay remap length termination }, où

- "***delay***" est le délai, exprimé en secondes, observé à l'impression par la HP48 entre 2 lignes successives (par défaut: 1.8 s)
- "***remap***" est une chaîne redéfinissant éventuellement les caractères étendus (de code > 128). Voir instruction **OLDPRT**.
- "***length***" est la longueur maximum d'une ligne, dans le cas d'une impression par la voie série. Par défaut, "*length*" vaut 80.
- "***termination***" est une chaîne représentant la séquence de fin de ligne, dans le cas d'une impression par la voie série.

  Par défaut cette chaîne est formée par deux caractères de codes respectifs 13 et 10.

Pour plus de détails sur **PRTPAR**, on se reportera aux pages 628 et suivantes du manuel de l'utilisateur de la HP48. Voici les formes internes des instructions les plus courantes:

Adresse	Mnémoniques et commentaires	
#31868h	cr	Forme interne de CR
#318A4h	prstc	Forme interne de PRSTC
#318FEh	pr1	Forme interne de PR1
#31A25h	prst	Forme interne de PRST
#31DABh	oldprt	Forme interne de OLDPRT
#31EE2h	prlcd	Forme interne de PRLCD
#31FFDh	delay(r)	Forme interne de DELAY
#1EEECh #1EF1Eh #31D56h	prvar(to) prvar(li) prvar(nm)	Les trois formes internes possibles de PRVAR. prvar(to) est l'impression du contenu d'un objet-sauvegarde
#32B74h	pr1(go) Forme interne de PR1, dans le cas où l'objet à imprimer est un objet-graphique.	

Adresse	Mnémoniques et commentaires (LM si langage machine)
#31854h	PrintRC |Envoie un saut de ligne à l'imprimante.
#318EAh	dup;print(st)!
#32161h	print(st)!
	|Analogue à "print(st)" en #32387h, sauf pour une impression sur la voie série (où l'impression se fait ligne par ligne, en tenant compte de la longueur des lignes de l'imprimante et des sauts de ligne éventuels dans "st").
#32251h	PrintRC(st)
	|Envoie la chaîne à l'imprimante. Envoie ensuite un saut de ligne si le flag -38 est désarmé.
	|Appelle #32260h ci-dessous
#32260h	PrintNoRC?(st,?)
	|Envoie la chaîne à l'imprimante. Envoie ensuite un saut de ligne si le booléen vaut FALSE.
	|Appelle #32387h ci-dessous
#32387h	print(st)
	|Imprime la chaîne présente au niveau 1. Teste le flag -24 (voie série ou voie IR)
	|L'impression se fait en un seul jet (sans tester si la chaîne contient des sauts de lignes)
#323B4h	IRprint(st)
	|(LM) Imprime la chaîne par infra-rouge.
	|Interne à "print(st)" en #32387h

Les adresses qui suivent créent ou utilisent la variable **PRTPAR** (contenant les paramètres d'impression).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#31F45h	Reset&BreakPRTPAR
	|Place liste standard dans PRTPAR (dans HOME). Donne sur la pile: 5:1.8 4:"" 3:80 2:"▪▪" 1:<4h>
#31F7Dh	StoInPRTPAR
	|Stocke l'objet du niveau 1 dans PRTPAR (dans HOME)
#31FAEh	ChkPRTPAR
	|Vérifie l'existence de PRTPAR dans HOME, et le fait que PRTPAR contient une liste. Si cette liste est vide, ou si PRTPAR n'existe pas, il y a un appel à Reset&BreakPRTPAR ci-dessus.
	|Sinon, il y a l'erreur "Invalid PRTPAR"
#3205Ch	ChkPRTPAR&Init
	|Teste l'existence et la validité de PRTPAR (le crée éventuellement avec ses valeurs par défaut), et initialise le port d'impression.
#323F9h	remap(2st)
	|(LM) Conversion des caractères étendus de st2. Tout caractère de code 127+k dans la chaîne st2 est remplacé par le kème caractère de st1.
	|(Utile en liaison avec le 2ème élt de PRTPAR)

LA LIAISON SÉRIE

Ce chapitre est consacré aux opérations utilisant *la voie série* de la HP48. Un chapitre a été consacré aux opérations d'impression, qui sont par nature des opérations à sens unique (vers l'extérieur de la calculatrice), et qui peuvent passer aussi bien par la voie *infra-rouge* (qui fonctionne uniquement en sortie) que par la voie *série*.

On trouvera dans ce chapitre essentiellement les formes internes des instructions du menu **I/O**, et les SYSEVALS correspondant aux différentes entrées du sous-menu **SETUP** du menu **I/O**.

J'ai préféré ne pas inclure ici un certain nombre d'adresses très "*internes*" pour deux raisons.

- Certaines sont d'une grande complexité, et quelques détails de leur fonctionnement peuvent encore m'échapper.
- Beaucoup de SYSEVALS (notamment ceux qui assurent le fonctionnement interne du protocole **KERMIT**) utilisent des variables locales (créés au départ des instructions du type **SERVER**, ou **SEND**, par exemple).

Il est dès lors assez illusoire de vouloir les utiliser de manière autonome.

Rappelons que les opérations d'entrée-sortie, par la voie série, utilisent une variable, nommée **IOPAR**, créée dans le répertoire **HOME**, et dont le contenu est une liste ayant le format suivant:

{ baud parity receive-pacing transmit-pacing checksum transio }

où:

- "***baud***" est la *vitesse de transmission*, exprimée en bauds. Elle peut prendre les valeurs 1200, 2400, 4800, ou 9600.
- "***parity***" est un entier indiquant si les transferts de données doivent utiliser un *contrôle de parité*, et lequel.
- "***receive-pacing***" contient un réel non nul ou 0 ,selon que la réception (hors **KERMIT**, qui gère très bien ça tout seul) de données doit (ou ne doit pas) s'accompagner d'un signal **XOFF** (de la part de la HP48 qui reçoit), quand le buffer d'entrée est plein, et d'un signal **XON** pour signaler que le transfert peut reprendre.
- "***transmit-pacing***". Même signification que "*receive-pacing*", mais pour dire à la HP48 qui émet de cesser d'émettre si elle reçoit un signal **XOFF**, et de reprendre au prochain signal **XON**.
- "***transio***" est un entier permettant d'ajuster la traduction des caractères étendus (de code supérieur à 127), en cas de transfert en ASCII.

  Si vous voulez éditer, ou simplement relire des programmes sur votre PC, vous avez intérêt à utiliser le code N°3.

Le contenu de **IOPAR**, par défaut, est: **{9600 0 0 0 3 1}**. Pour plus de détails, on se reportera aux pages 638 à 660 du manuel de l'utilisateur de la HP48.

Les adresses qui suivent sont les formes internes (avec quelques variantes) des instructions du menu **I/O**:

Adresse	Mnémoniques et commentaires (LM si langage machine)	
#2D816h	recn(nm)	Les 2 formes internes de RECN. Elles sont
#2D816h	recn(st)	implantées à la même adresse.
		La forme interne de RECV est la suivante:
		""; newob; recn(st)
#2D9F5h	server	Forme interne de SERVER.
#2E5ABh	send(nm)	Les 2 formes internes de SEND. En fait,
#2E6EBh	send(li)	send(nm) se réduit à 1 ->LIST send(li)
#2E7EFh	kget(nm)	Les trois formes internes de KGET.
#2E7EFh	kget(st)	On remarque que kget(nm) et kget(st)
#2E835h	kget(li)	sont implantés à la même adresse
#2E876h	finish	La forme interne de FINISH. Il s'agit en fait de: "F" "G" pkt(2st)
#2E8D1h	pkt(2st)	Forme interne de PKT
#2EC84h	baud(r)	Forme interne de BAUD
#2ECCAh	parity(r)	Forme interne de PARITY
#2ED10h	transio(r)	Forme interne de TRANSIO
#2ED4Ch	cksm(r)	Forme interne de CKSM
#2EDA6h	kerrm	Forme interne de KERRM
#2EDF5h	stime(r)	Forme interne de STIME
#2EE6Fh	xmit(st)	Forme interne de XMIT
#2EE97h	srcev(r)	Forme interne de SRCEV
#2EE18h	sbrk	Forme interne de SBRK
#2EDE1h	buflen	Forme interne de BUFLEN
#324C8h	OpenIO	
	|(LM). Forme interne de OPENIO. A faire précéder de	
	| de "InitIO". L'instruction OPENIO consiste en un	
	| appel de "InitIO", puis de "OpenIO".	
#2EB37h	InitIO	
	|(LM) Prépare Ouverture I/O et teste IOPAR.	
#315C6h	CloseIO	
	| Forme interne de CLOSEIO. En même temps, vide le	
	| buffer de la sortie série.	

Voici trois SYSEVALs un peu inclassables:

Adresse	Mnémoniques et commentaires
#303ACh	HeadIOString
	|Donne une chaîne du type " T(1)A(D)F(.)" utilisée
	| pour indiquer les paramètres d'un transfert ASCII.
	|Cette chaîne apparait en tête des fichiers transmis.
#303B6h	HeadIOString(s)
	|Comme ci-dessus, mais l'entier système "s" désigne
	| le code transIO.
#21B5Ah	archiveI/O
	|Archive par KERMIT sur fichier dont le nom est au
	| niveau 1 (chaîne, nom local ou global).

Voici des formes plus spécialisées des SYSEVALs donnés dans le premier tableau de la page précédente:

Adresse	Mnémoniques et commentaires (LM si langage machine)
#2D9B2h	ExitServer | (LM) sort du mode serveur
#2E803h	kget({2nm}) | Effectue KGET pour le 1er nom en renommant |avec le 2-ème nom de la liste. Appelle kget(nm,st)
#2E8A2h	kget(nm,st) | Effectue KGET sur le nom 'nm', en lui |donnant pour nouveau nom celui qui est indiqué par |"st" (et qui peut être un nom illégal). |Si "st" est la chaîne vide, le nom est conservé.
#31444h	xmit(st)! | Comme xmit(s) mais sans test sur validité de IOPAR, | ni traitement des erreurs éventuelles. Se termine | par le booléen True ou False (plutôt que 1 ou 0).
#314E5h	srecv(r)! | (LM) Comme srecv(r)! mais pas de test sur IOPAR ni | traitement des erreurs éventuelles. Se termine | par le booléen True ou False (plutôt que 1 ou 0).
#31579h	StimeSRecv(s) | (LM) Définit le délai d'attente de SRECV où "s" est | un entier-système exprimé en 1/10 de secondes.
#31589h	StimeXmit(s) | (LM) Définit le délai d'attente de XMIT, où "s" est | un entier-système exprimé en 1/10 de secondes.
#3133Bh	Sbuflen | (LM) Donne le même résultat que buflen, mais en 2 | entiers-système. Appelé par buflen.

Les adresses suivantes créent ou modifient la variable **IOPAR**, ou bien encore en testent le contenu (avec à la clef une erreur éventuelle du type "***Invalid IOPAR***").

Adresse	Mnémoniques et commentaires
#2E999h	ResetIOPAR&->|Place la liste par défaut {9600 0 0 0 3 1} dans la | variable IOPAR (dans HOME), puis place les objets | de cette liste sur la pile.
#2E9CBh	StoInIOPAR |Stocke l'objet du niveau 1 dans IOPAR (dans HOME)
#2EA4Fh	EvalIOPAR |Evalue le contenu de la liste IOPAR. Vérifie que | IOPAR contient bien une liste de 6 éléments (si | ce n'est pas le cas, erreur "Invalid IOPAR") |Si IOPAR n'existe pas dans HOME, il y est créé.
#2EB62h	okIOPAR? |Teste le contenu de 'IOPAR' (le crée s'il n'existe | pas, puis appel en #31608h)
#2ECACh	6->list;StoInIOPAR
#3187Ch	okIOPAR?;openIO

Voici enfin les adresses correspondant au sous-menu **SETUP** du menu **I/O**. Il s'agit surtout d'afficher le contenu de l'écran **SETUP**, après avoir éventuellement modifié un des éléments de la variable **IOPAR**.

Adresse	Mnémoniques et commentaires
#220F6h	IOSetup |Entre dans l'environnement SETUP du menu I/O, sous réserve de quelques tests préalables (est-on en mode PRG?, ligne de commande en cours?...)
#22105h	IOSetup! |Comme ci-dessus, sans les tests préalables.
#22307h	SetupA/B |Bascule ASCII/Binary. Suivi de SetUpScreen
#22325h	SetupIR/W |Bascule IR/Wire. Suivi de SetUpScreen
#22343h	SetupBaud |Next speed. Suivi de SetUpScreen
#223BFh	SetupCKSM |Next checksum. Suivi de SetUpScreen
#223F6h	SetupParit |Next parity. Suivi de SetUpScreen
#22437h	SetupTrans |Next translation code. Suivi de SetUpScreen
#2D730h	IO2disp(st) |if flag(-39)=0 then (2 disp) else drop
#2D74Eh	IO1disp(st) |if flag(-39)=0 then (1 disp) else drop
#2EC52h	s=OkBaud? |teste si l'entier-système "s" appartient à la liste {1200,2400,4800,9600}. Résultat booléen.
#30370h	ChkTransio |Teste la valeur de "Transio" et génère une erreur "Invalid IOPAR" si cette valeur est ≥ 4. Sinon cette valeur est retournée sous forme d'un entier système, au niveau 1 de la pile.

MENUS SOLVE ET PLOT

Ce chapitre est consacré aux instructions contenues dans les menus **SOLVE** et **PLOT** (et dans leurs sous-menus **PTYPE**, **SOLVR** et **PLOTR**), à l'exception de l'instruction **ROOT** (du menu **SOLVE**) qui est évoquée dans le chapitre consacré aux expressions.

Il s'agit ici essentiellement d'instructions se rapportant à la variable **'EQ'** (contenant l'expression en cours, ou une liste d'expressions, ou un nom, ou une liste de noms, chaque nom renvoyant à une expression), et à la variable **'PPAR'**.

Rappelons que la variable **'PPAR'** (qui peut exister individuellement dans chaque répertoire) contient une liste indiquant les différents paramètres de tracé de l'expression en cours.

Le format de cette liste est **{ pmin pmax indep res axes ptype depend }**.

Pour une description complète de **'PPAR'**, on se reportera aux pages 331 et suivantes du manuel de l'utilisateur de la HP48.

Voici, pour commencer, les formes internes des instructions programmables du menu **PLOTR** (la distinction entre l'instruction programmable **DRAW**, et la touche **DRAW** est importante: cette dernière trace les axes et provoque un passage dans l'environnemeent **GRAPH**).

Adresse	Mnémoniques et commentaires	
#4B553h	*h(r)	Forme interne de *H
#4B5ADh	*w(r)	Forme interne de *W
#491D5h	auto	Forme interne de AUTO
#4B03Ah	axes(c)	1ère forme interne de AXES
#4B04Eh	axes(li)	2ème forme interne de AXES
#4AC61h	centr(c)	1ère forme interne de CENTR
#1E101h	centr(r)	2ème forme interne de CENTR
#47A8Dh	depnd(2r)	1ère forme interne de DEPND
#4AFB3h	depnd(gn)	2ème forme interne de DEPND
#4AFC7h	depnd(li)	3ème forme interne de DEPND
#4B6ACh	draw	Forme interne de DRAW (trace pas les axes)
#4C607h	drax	Forme interne de DRAX
#4B60Ch	erase	Forme interne de ERASE
#47A6Ah	indep(2r)	1ère forme interne de INDEP
#4AF77h	indep(gn)	2ème forme interne de INDEP
#4AF8Bh	indep(li)	3ème forme interne de INDEP
#4B300h	pdim(2b)	1ère forme interne de PDIM
#4B206h	pdim(2c)	2ème forme interne de PDIM
#4B09Eh	pmin(c)	Forme interne de PMIN
#4B0C6h	pmax(c)	Forme interne de PMAX
#4B012h	res(b)	1ère forme interne de RES
#4AFEFh	res(r)	2ème forme interne de RES
#4AE3Ch	scale(2r)	Forme interne de SCALE
#47A1Ah	xrng(2r)	Forme interne de XRNG
#47A42h	yrng(2r)	Forme interne de YRNG

Voici les SYSEVALS responsables des nombreux affichages rencontrés dans les menus **SOLVE** et **PLOT** (notamment les écrans d'information obtenus quand on entre dans ces menus, ou quand, par exemple, on modifie un des éléments de **'PPAR'** dans le menu **PLOTR**).

Adresse	Mnémoniques et commentaires
#153C0h	EQreview |1 Disp sur le contenu de EQ, puis éxécute review
#47485h	PlotScreen |Affiche le mode de tracé et l'objet à tracer ('EQ' ou ΣDAT) et son contenu. C'est l'affichage obtenu quand on entre dans menu PLOT.
#474E9h	->StrPType |Donne une chaîne indiquant le type de tracé |Ex: "Plot type: CONIC", ou "Plot type: BAR"
#47548h	SolveScreen |Affiche "Current equation" puis, à la ligne qui suit, le nom et le contenu de cette équation. |C'est l'affichage obtenu quand on entre dans SOLVE
#475FCh	PlotrScreen |Affiche l'écran de présentation quand on rentre dans le sous-menu PLOTR (Affiche le type de tracé, l'équation en cours, la var indépendante, et les "Ranges"), sous réserve de certains tests préala- -bles (buffer clavier, mode PRG, etc....), puis gèle cette zone de l'écran.
#4760Bh	PlotrScreen! |Comme ci-dessus, mais sans les tests préalables.
#47642h	DispRanges |Affiche à sa place la partie "Ranges" de l'affichage produit par "PlotrScreen", puis gèle cette zone.
#4765Bh	->StrIndep |Produit une chaine donnant le nom de la variable indépendante. Ex: "Indep:'X'" ou "Indep:{ X 1 3 }"
#476B7h	DispDepend;<3h> |Produit l'affichage correpsondant à la variable dépendante. Pas de Freeze. Laisse <3h> sur la pile
#478ABh	New(ex) |Saisie d'un nom et stockage de l'expression sous ce nom (puis de ce nom dans EQ), ou stockage de l'expression dans 'EQ' si le nom est vide. |C'est la procédure attachée à la touche NEW des menus SOLVE et PLOT, avec affichage du message: "Name the Equation, press ENTER"
#47957h	InputPType |Affiche le type de tracé en cours et demande de choisir le nouveau "PType". C'est la séquence obtenue quand on rentre dans le menu PTYPE.
#47B5Ah	EQcat |lance catalogue d'équations

Le tableau ci-dessous contient les nombreuses adresses s'appliquant à la variable **'EQ'** et à son contenu (affichages, résolution d'équation, modification ou rappel du contenu de **'EQ'**, etc...).

Adresse	Mnémoniques et commentaires
#15273h	1FrDispEQ
	|Affiche EQ et son contenu en ligne d'état, puis gèle
	| l'affichage de cette zone (FrDisp = Freeze Disp)
	|(Si 'EQ' n'existe pas, n'affiche que le nom 'EQ')
#155E0h	SolveEQ(nm)
	|Exécute ROOT (avec message "Solving For") avec la
	| variable donnée au niveau 1, et sur le contenu
	| de 'EQ'. Donne résultat taggué par la variable.
	|C'est la séquence de résolution d'équation de SOLVR
#15717h	steq |Forme interne de STEQ.
#1572Bh	rceq |Forme interne de RCEQ.
#15744h	ExistEQ?
	|Vérifie l'existence de 'EQ'.
	|Répond 2:contenu,1:true ou 1:false
#15758h	'EQ |Place le nom 'EQ' sur la pile
#1587Dh	Trceq!
	|Rappelle le contenu de 'EQ'.Le résultat est au niveau
	| 1 (taggué par EXPR) ou aux niveaux 2 et 1 (taggués
	| par LEFT et RIGHT) dans le cas d'une équation.
	|Erreur "No Current Equation Eventuelle"
#475A2h	->StrEQ?
	|Si 'EQ' n'existe pas, répond
	| 2:"Enter eqn, press NEW" 1:false
	|Si 'EQ' existe, donne 2:Contenu de EQ 1:true,
	| le contenu de 'EQ' étant apprécié en fonction du
	| type de catalogue sélectionné ("Fast" ou non)
	|Appelé dans "PlotScreen" et dans "SolveScreen"
#4A08Ch	rceq!
	|Identique au précédent. Mais si 'EQ' contient une
	| liste, extrait le premier élément de cette liste.
#4A28Fh	Trceq
	|Rappelle le contenu de 'EQ', sous la forme d'objet
	| taggué par 'EQ' si 'EQ' existe.
	|Ne donne que le nom 'EQ' si 'EQ' n'existe pas.
#47AB0h	edeq |Forme interne de EDEQ
	|Suivi d'un écran "SolveScreen" ou "PlotScreen"
#49CB8h	NextInEQ
	|Comme ci-dessous, mais en plus affiche la nouvelle
	| équation courante, sur la ligne de menu (puis gèle
	| cette zone de l'écran)
#49CBDh	NextInEQ!
	|Si 'EQ' contient une liste { a b ... z }, place dans
	| 'EQ' la liste { b c ... z a }. Sinon 'EQ' inchangé.
#49E11h	ChkEQ(eq)
	|Vérifie que l'équation au niveau 1 est un contenu
	| valide pour EQ, c-a-d est du type X=expr, où X
	| est le nom de la variable indépendante, et où expr
	| ne contient pas de référence à ce nom.

Le tableau suivant regroupe les SYSEVALs modifiant ou rappelant (en totalité ou en partie) le contenu de la variable **'PPAR'**.

Adresse	Mnémoniques et commentaires
#49266h	ScaleY=ScaleX |Place dans l'échelle en y l'échelle en x.
#4973Ah	PlotrReset |Effectue l'action de la touche RESET du menu PLOTR | En particulier, se termine par "PlotrScreen"
#4AAEAh	Reset&RclPPAR |Stocke dans 'PPAR' ou dans le nom désigné par 'PPAR' | (si ce nom existe), la liste standard, et rappelle | cette liste sur la pile.
#4AC1Bh	FUNCTION! |Définit FUNCTION comme type de tracé dans 'PPAR'
#4AC34h	RclCentr |Rappelle le complexe donnant le centre de PICT
#4ADB0h	RclScale |Rappelle les échelles en X et en Y. |Donne deux réels: 2:ScaleX, 1:ScaleY
#4AF63h	RclIndep |Rappelle le contenu de la variable indépendante, | qui est peut-être une liste
#4AF9Fh	RclDepnd |Rappelle le contenu de la variable dépendante, | qui est peut-être une liste
#4AFDBh	RclRes |Rappelle la valeur de la résolution dans 'PPAR'
#4B026h	RclAxes |Rappelle le contenu de "Axes" dans 'PPAR'
#4B062h	RclPType |Rappelle PTYPE (commande intégrée)
#4B076h	StoPType |L'objet niveau 1 est le nouveau PTYPE
#4B08Ah	RclPmin |Rappelle le complexe définissant PMIN
#4B0B2h	RclPmax |Rappelle le complexe définissant PMAX
#4B0DAh	RclPmin&Pmax |Vérifie que PPAR est une liste de 7 objets et que | les 2 premiers sont des nombres complexes, qui | sont laissés sur la pile (2:Pmin 1:Pmax) | (sinon erreur "Invalid PPAR")
#4B10Ch	RclMinXRNG |Rappelle valeur minimum du range en X
#4B120h	RclMinYRNG |Rappelle valeur maximum du range en X
#4B139h	RclMaxXRNG |Rappelle valeur minimum du range en Y
#4B14Dh	RclMaxYRNG |Rappelle valeur maximum du range en Y
#4B166h	StoMinXRNG(r) |Stocke valeur minimum du range en X
#4B189h	StoMinYRNG(r) |Stocke valeur minimum du range en Y
#4B1ACh	StoMaxXRNG(r) |Stocke valeur maximum du range en X
#4B1CFh	StoMaxYRNG(r) |Stocke valeur maximum du range en Y
#4B364h	GetPPAR(s) |Extrait le s-ème élément de 'PPAR', où | s est un entier-système. Contôle 'PPAR' et le crée | s'il n'existe pas.

La suite du tableau précédent....

Adresse	Mnémoniques et commentaires
#4B37Dh	ChkPut('PPAR',li,s) |Vérifie que l'on peut bien placer la liste "li" en s-ème position dans 'PPAR'. 'PPAR' est contrôlé et cette instruction PUT est effectuée si aucune erreur n'est trouvée (s=<3h>, <5h>, ou <7h>).
#4C0FDh	RclYminYmax |Place Ymin au niveau2, Ymax au niveau 1.
#5103Ah	PutBackPPAR(_,s) |Place l'objet situé au niveau 2 en position "s" dans 'PPAR', en partant de la fin Il y a un contrôle de 'PPAR'. 'PPAR' peut contenir un nom. C'est dans la liste contenue dans ce nom qu'est stocké l'objet.
#51067h	StoInPPAR(7_) |Effectue 7 ->LIST, puis stocke la liste dans 'PPAR' ou dans le nom contenu dans 'PPAR'.
#51099h	RclNameDepnd |Donne le nom de la variable dépendante, au besoin en l'extrayant de la liste {var deb fin}, alors que "RclDepnd" rappelerait alors la liste.
#510ADh	RclNameIndep |Donne le nom de la variable indépendante, au besoin en l'extrayant de la liste {var deb fin}, alors que "RClIndep" rappelerait alors la liste.

Voici maintenant différents tests que peut effectuer la HP48 pour s'assurer que le contenu de la variable '**PPAR**' est correct.

Adresse	Mnémoniques et commentaires
#4A9AFh	Chk&RclPPAR |Teste 'PPAR'. (La recrée si elle n'existe pas) |Erreur "Invalid PPAR" éventuelle. |Si tout est OK, rappelle PPAR sur la pile.
#4AA59h	ChkIndep |Vérifie que l'objet au niveau 1 est un nom global ou une liste { gn, r, r } |Erreur "Invalid PPAR" éventuelle. Adresse utilisée pour vérifier la partie DEPND ou INDEP de 'PPAR'.
#4AAA9h	Chk&BreakPPAR |Teste l'existence de 'PPAR', sinon la crée. |Teste que 'PPAR' contient une liste de 7 éléments (sinon erreur "Invalid PPAR"). |Enfin effectue OBJ-> DROP sur cette liste.
#4ABC1h	ChkPType(_) |Teste l'objet au niveau 1, et si cet objet n'est aucune des instructions FUNCTION, CONIC, POLAR, etc... définissant les divers types de tracé, alors définit FUNCTION comme type de tracé. |Utilisé pour vérifier PTYPE dans 'PPAR'
#4B4EAh	PPAR(s);dup;=li? |Rappelle le s-ème élément de 'PPAR' et teste si c'est une liste (réponse booléenne).

Terminons par un cocktail de SYSEVALs aux goûts très divers:

Adresse	Mnémoniques et commentaires
#4AB7Bh	MakeRange(s) |Crée un "Range" à partir d'un entier-système. |Par ex: <83h> => -6.5,6.5 et <40h> => -3.1,3.2 |L'opération réalisée est s => -(s-1)/20,+(s-1)/20
#4B1F2h	RclPdim |Rappelle les dimensions de PICT (ex: 2:#83h 1:#40h)
#4B323h	pdim(2s) |Redimensionne PICT avec les dimensions s2 (largeur) | et s1 (hauteur). NB: si s2 est inférieur à <83h>, | il est remplacé par <83h>. Idem avec s1 et <40h>
#4B733h	DrawEQ |Trace les Axes puis le contenu de EQ. Après, on | reste dans GRAPH
#4C639h	drax(c) |Forme interne de drax, mais ne vérifie pas la | validité de PICT
#4C6CFh	DrawXaxix(2s,2r) |Trace un axe horizontal gradué dans PICT. s4 et s3 | sont les dimensions de PICT (s4=hauteur...) |r2,r1 = coordonnées du centre des axes. |Les 4 objets restent sur la pile. |Pas de test pour savoir si l'axe traverse bien PICT
#4C77Eh	DrawYaxix(2s,2r) |Comme ci-dessus mais axe vertical.
#4D8F5h	NewDrawEQ |Trace EQ (et les axes) après avoir effectué ERASE. |On reste après dans le menu GRAPH.
#4E875h	label |Effectue l'action de la touche LABEL du menu PLOTR
#4E889h	label! |Appelé par adresse ci-dessus. La différence est que | "label!" ne vérifie pas la validité de PICT
#50E04h	LookForEQ/ΣDAT |Selon le type de tracé, teste l'existence de ΣDAT | ou de EQ. Si non existence, affiche le message | pendant 1 s, puis freeze, mais pas d'erreur.
#5127Eh	';PPAR | Place le nom 'PPAR' sur la pile
#5129Ch	';FUNCTION | Place la commande FUNCTION sur la pile
#512B0h	';CONIC | Place la commande CONIC sur la pile
#512C4h	';POLAR | Place la commande POLAR sur la pile
#512D8h	';PARAMETRIC | Place la commande PARAMETRIC sur la pile
#512ECh	';TRUTH | Place la commande TRUTH sur la pile
#51300h	';SCATTER | Place la commande SCATTER sur la pile
#51314h	';HISTOGRAM | Place la commande HISTOGRAM sur la pile
#51328h	';BAR | Place la commande BAR sur la pile

LES MENUS

Le système des menus (intégrés ou non) est familier à l'utilisateur de la HP48. Derrière cette interface naturelle se cache une organisation très complexe. J'ai réussi à percer de nombreux mystères, mais il reste encore beaucoup à faire. Les adresses que je donne ici sont celles dont je suis sûr.

Sachez que derrière ces SYSEVALs se cachent des choses étonnantes. Alors si le coeur vous en dit...

Dans ce chapitre, nous aborderons la définition des menus (et notamment la description des actions assignées à chacune des 6 touches de menu, qu'elles soient *shiftées* ou non), puis l'aspect graphique de la création des labels de menu.

Voici tout d'abord les formes internes de **MENU** et de **TMENU** (les deux instructions qui permettent de créer des *menus-utilisateurs*), puis les formes internes de ces formes internes ("***DoMenu(li)***" et ses dérivés):

Adresse	Mnémoniques et commentaires
#21176h	tmenu(#r) |Forme interne de tmenu quand l'arguement n'est pas un nombre réel (donc un nom ou une liste) |Consiste en "ChkForMenu(_)", puis "DoMenu(li)!"
#41679h	tmenu(r) |Forme interne de TMENU quand l'argument est un réel. |C'est la même adresse que pour MENU
#40DC0h	DoMenu(li);3Freeze! |Comme ci-dessous, puis 3Freeze!
#40F86h	DoMenu(li) |Fait de "li" le menu courant, en l'affichant à partir de sa première entrée. |Le contenu de DoMenu(li) est "<1h> DomenuPos(li,s)"
#40F9Ah	DoMenuPos(li,s) |Fait de li le menu courant, en l'affichant à partir de sa s-ième entrée. |"li" peut en fait être un Rpl, si l'évaluation de ce Rpl place une liste au niveau 1. |L'évaluation de ce Rpl peut par exemple permettre d'initialiser l'action exacte des touches de menu shiftées ou non.
#3DB1Ah	ChkForMenu(_) |Teste si l'objet au niveau 1 est susceptible d'être un argument non réel de TMENU ou de MENU. |L'objet doit être une liste (elle reste sur la pile) ou sinon un nom (local ou global) connu dont le contenu est une liste (le nom est alors remplacé par cette liste sur la pile)
#41008h	DoMenuPos(li,s)! |Comme DoMenuPos(li,s), mais sans traitement d'une erreur éventuelle.

Remarque:

Les SYSEVALs précédents qui sont du type "***DoMenu***" sont très importants, car ils permettent de s'affranchir de la contrainte que connaissent les instructions **MENU** et **TMENU** (et les deux formes internes vues ci-dessus), à savoir que l'argument (dont on veut faire un menu) doit être un nombre réel ou une liste (ou un nom dont le contenu renvoie à un réel ou à une liste).

En effet de nombreux menus intégrés de la HP48 sont créés en utilisant les intructions du type "***DoMenu***", l'argument étant un programme.

Ce programme est évalué et cette évaluation doit conduire à placer une liste sur la pile. Mais dans le même temps, l'évaluation de ce programme permet de régler un certain nombre de caractéristiques de ce menu (est-ce un menu de saisie comme **SOLVR**, ou un menu de sous-menus comme **MTH** ?; quelle est la signification exactes des touches de menus, shiftées ou non, quel est l'action de la touche **REVIEW**?, etc...)

Avant d'en venir à ces subtilités, voyons les formes internes qui découlent de l'instruction **RCLMENU**, ou encore les SYSEVALs qui sont associés à l'action des touches **NXT** et **PREV**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#415C9h	rclmenu |Forme interne de RCLMENU
#416F1h	MenuAdr(s) |Rappelle contenu du Menu N° s. En fait c'est bien sûr l'adresse de ce menu qui est obtenue.
#418A4h	CurMenuAdr |(LM) Rappelle le menu en cours
#4185Bh	RclMenuPos |(LM) Donne un entier-système égal à la position, dans le menu courant, de l'entrée correspondant à la touche de gauche.
#41848h	DoMenuPos(s) |(LM) Fait de l'entrée N°s la première entrée à gauche dans le menu VAR (ne modifie pas l'ordre l'ordre des entrées les unes / aux autres). |Faire suivre de #410C6h SYSEVAL (SetThisRow) pour que celà s'applique aux autres menus.
#4161Ah	NumCurMenu |Donne le réel égal au N° de menu actuel (mais sans donner la page comme RCLMENU) |On obtient 2 pour le menu VAR, 1 pour le menu CST
#3B243h	ToFirstPage |Affiche la première page du menu en cours.
#3A1E8h	DispMenu |Affiche dernier menu spécifié. |Essayer « 1 20 FOR k k MENU #3A1E8h SYSEVAL NEXT » |(Sans "DispMenu", les menus successifs ne seraient pas affichés pendant l'éxécution du programme)
#3A1CAh	NoKey?:DispMenu |Exécute "DispMenu" si le buffer du clavier est vide.
#3A9E7h	NextLabel(s) |Décale le menu en cours de |s| positions vers la droite si s>0, vers la gauche si s<0.

On sait que la HP48 garde toujours en mémoire le dernier menu affiché avant le menu actuel (et qui ne soit pas un menu de sous-menus comme **MTH** par exemple, ou encore **UNITS**). La touche **LASTMENU** permet de revenir à ce menu mémorisé.

Bien sûr l'action de la touche **LASTMENU** dépend d'un certain nombre de SYSEVALs .Voici les plus intéressants d'entre eux.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#413B9h	DoLastMenu |Exécute l'action de la touche LastMenu
#419F4h	LastMenuAdr |(LM) Rappelle le contenu de LastMenu
#41881h	RclLastMenuPos |(LM) Rappelle, sous forme d'entier-système, la | position dans LastMenu (<1h>, <7h>, <Dh>....)
#419E4h	StoLastMenuAdr(li) |(LM) Stocke "li" comme contenu de LastMenu. "li" | peut être un RPL s'évaluant sur une liste.
#4186Eh	StoLastMenuPos(s) |(LM) Stocke "s" comme 1-ière position dans LastMenu
#4139Bh	LastMenu=CurMenu |Place le menu actuel dans LASTMENU (en stockant | aussi la position courante dans ce menu).

On sait que les *menus-utilisateurs* sont créés au moyen de listes. Les *menus intégrés* de la HP48 échappent un peu à cette règle. Dans tous les cas cependant ils sont essentiellement définis par une liste telle que celles utilisées pour les menus-utilisateurs.

Les adresses des listes correspondant aux menus intégrés de la HP48 ont été données dans le chapitre consacré aux objets de type "liste".

Cependant, comme celà a été dit plus haut, certains menus de la HP48 sont définis par la donnée d'un programme (qui s'évalue sur une liste, mais qui procède à certaines initialisations).

Voici les adresses de ces programmes:

Adresse	Menu	Adresse	Menu	Adresse	Menu
#3B239h	MenuCST	#3B284h	MenuMTH	#3B542h	MenuPRG
#3BA03h	MenuSETUP	#3BE22h	MenuSOLVE	#3BEB8h	MenuPLOT
#3BD82h	MenuPort0	#3BDAAh	MenuPort1	#3BDD2h	MenuPort2
#3C039h	MenuPTYPE	#3C4C9h	MenuTIME	#3C79Ch	MenuALARM
#3CAA7h	MenuSTAT	#3CD96h	MenuMODL	#3CE65h	MenuUNITS
#3D08Ch	MenuLENG	#3D1F3h	MenuAREA	#3D2D6h	MenuVOL
#3D451h	MenuTIME	#3D4BAh	MenuSPEED	#3D553h	MenuMASS
#3D642h	MenuFORCE	#3D6B5h	MenuENRG	#3D764h	MenuPOWR
#3D797h	MenuPRESS	#3D838h	MenuTEMP	#3D887h	MenuELEC
#3D93Ah	MenuANGL	#3D9B3h	MenuLIGHT	#3DA42h	MenuRAD
#3DABFh	MenuVISC	#3DE93h	MenuGRAPH	#3DF2Fh	MenuFCN
#3F376h	MenuLIB	#3F6D8h	MenuVAR	#43D06h	MenuSTACK
#46003h	MenuMATRIX	#484FEh	MenuEQcat	#48544h	MenuSTATcat
#48585h	MenuALRMcat	#46003h	MenuEQUAT		

La touche **REVIEW** n'a pas toujours le même rôle suivant le menu dans lequel on se trouve. Dans le menu **VAR**, ou dans la plupart des menus intégrés, elle permet de consulter la liste des 6 entrées de la page de menu en cours, et le contenu de ces entrées (ou le type de ce contenu).

Dans le sous-menu **SETUP** de **I/O**, la touche **REVIEW** provoque plutôt l'affichage d'un écran rappelant les paramètres actuels de transfert.

Pour que la touche **REVIEW** effectue son rôle le plus courant, il faut bien qu'existent quelque part des adresses extrayant les noms et les objets correspondant aux différentes entrées du menu.

L'action d'une touche de menu peut se faire sans *shift*, ou avec.

Le résultat obtenu dépend là encore du menu en cours (voyez par exemple les différences entre le menu **VAR** et le menu de saisie **SOLVR**).

Les SYSEVALs du tableau ci-dessous traitent de ces questions.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#413D2h	review |Exécute REVIEW pour le menu en cours. Donne le | nom et le contenu des 6 entrées.
#41994h	LocalReview |Exécute REVIEW selon le sens qu'à cette touche | dans le menu en cours (par exemple, dans le | menu PLOTR, la touche REVIEW rappelle l'écran | des données actuelles du tracé)
#04A0Bh	GetProc(s) |(LM) avec s=<1h>..<6h> donne le s-ème nom du menu | VAR ou la s-ème procédure du menu HP en cours.
#40828h	';GetProc(s) |Place sur la pile l'entier-système qui suit dans | le Rpl, puis éxécute "GetProc(s)".
#04A41h	GetName(s) |(LM). Comme "getproc(s)", sauf qu'on obtient le | nom de l'entrée sélectionnée du menu en cours, | sous forme de chaîne si c'est un menu intégré.

#3FFDBh	NoShift1	
#3FFF4h	LeftShift1	Les SYSEVALs "NoShift" évaluent
#4000Dh	RightShift1	l'objet associé à une entrée k
#40026h	LeftShift2	particulière du menu en cours
#4003Fh	NoShift2	de k=1 à k=6, dans le mode
#40058h	RightShift2	"non shifté".
#40071h	NoShift3	
#4008Ah	LeftShift3	
#400A3h	RightShift3	
#400BCh	NoShift4	Les SYSEVALs "RightShift" font de
#400D5h	LeftShift4	même avec le mode "shifté à droite"
#400EEh	RightShift4	
#40107h	NoShift5	
#40120h	LeftShift5	Les SYSEVALs "LeftShift" font de
#40139h	RightShift5	même avec le mode "shifté à gauche"
#40152h	NoShift6	
#4016Bh	LeftShift6	
#40184h	RightShift6	

LES MENUS

Quand on définit un menu au moyen d'un programme, celui-ci doit s'évaluer sur une liste. Pendant cette évaluation, il peut néanmoins procéder à des *initialisations* concernant l'action de la touche **REVIEW**, ou l'action des touches de menu (*shiftées* ou non).

Il est également possible de prévoir une procédure à éxécuter au moment où on en sort. Certains menus de la HP48 le montrent: **SOLV**, **GRAPH**, **UNITS**, **ΣDAT**, etc...), ou de définir l'aspect graphique des labels de menu (entrées de répertoires, menus de saisie, etc...).

Tout celà est possible à l'aide d'un certain nombre de SYSEVALS qui sont éxécutés lors de l'évaluation du programme définissant un menu. L'exemple listé à la page suivante vous permettra de mieux comprendre comment fonctionnent les adresses ci-dessous.

Adresse	Mnémoniques et commentaires
#4019Dh	StdNoShift | Ces trois objets définissent l'action
#401D4h	StdLeftShift | standard d'une touche de menu, shiftée
#4021Fh	StdRightShift | ou non.
#418D4h	DoWhenNewPage | L'objet du niveau 1 est évalué à chaque changement de page dans le menu
#418F4h	InitLabels |L'objet du niveau 1 est évalué pour définir l'aspect graphique des labels de menus (voir plus loin)
#41914h	DefNoShift |L'objet du niveau 1 devient l'objet à évaluer dans ce menu lors d'un appui non shifté sur une touche de menu (Cet appui place le nom associé à la touche sur la pile). L'objet en question peut donc s'appliquer à ce nom. |Par exemple l'objet DROP inhibe l'action non shiftée des touches de menu.
#41944h	DefLeftShift |Comme ci-dessus, avec touches shiftées à gauche
#41964h	DefRightShift |Comme ci-dessus, avec touches shiftées à droite
#3F00Eh	OnlyDefNoShift! |Comme DefNoShift, de plus inhibe les appuis shiftés (qui se traduisent alors par un bip)
#41984h	DoWithReview |L'objet du niveau 1 devient l'objet à évaluer dans ce menu lors d'un appui sur la touche REVIEW (Par exemple l'objet "review") |Cette assignation cesse avec un changement de menu, même si on revient au menu initial.
#419C4h	DoWhenExit |L'objet du niveau 1 devient l'objet à évaluer dans ce menu quand on sort de ce menu. |Cette assignation doit être renouvelée à chaque fois qu'on revient au menu initial.
#3EC85h	NothingWhenExit |La sortie du répertoire actuel ne s'accompagnera d'aucune procédure particulière.

◆ UN MENU UN PEU PARTICULIER:

Nous allons créer un menu possédant les propriétés suivantes:

- Quand on change de page de menu, il y a un message "***New Page***", affiché pendant une seconde.
- Quand on en sort, le message "***See You Later***" est affiché.
- La touche **REVIEW** effectue un bip de 1 seconde.
- Les touches de menu ont les définitions suivantes:
  - *Non shifté*: usage normal (évaluation)
  - *Shifté à gauche*: le nom est taggué par "**L**"
  - *Shifté à droite*: le nom est taggué par "**R**"

Ce menu sera créé par l'instruction **#40F86h SYSEVAL** (***DoMenu(li)***), et il consistera en une liste des entrées, suivi d'appels aux procédures effectuant les réglages ci-dessus.

La liste définissant ce menu sera **{ A B C D E F G H I J }**, par exemple, mais vous pouvez bien sûr imaginer plus compliqué.

Voici le programme créant cet étrange menu:

'**MAKE**': (CheckSum: #177Ch; Taille: 368.5 octets)

```
« « { A B C D E F G H I J }                | Les entrées du menu.
    « CLLCD "New Page" 3 DISP 1 WAIT »     | Définit un objet
    #418D4h SYSEVAL                        |  pour NXT et PRV.
    « CLLCD "See You Later" 3 DISP »       | Définit un objet de
    #419C4h SYSEVAL                        |  sortie.
    « 440 1 BEEP »                         | Définit rôle de la
    #41984h SYSEVAL                        |  touche REVIEW.
    « #4019Dh SYSEVAL »                    | Action non shiftée
    #41914h SYSEVAL                        |  des touches de menu.
    « "L" →TAG »                           | Action shiftée gauche
    #41944h SYSEVAL                        |  des touches de menu.
    « "R" →TAG »                           | Action shiftée droite
    #41964h SYSEVAL                        |  des touches de menu.
  »
  #40F86h SYSEVAL                          | "DoMenu(li)".
»
```

Remarques:

- Le menu obtenu n'est pas placé dans la variable **CST** (seule l'instruction **MENU** effectue cette opération, et je rappelle que **MENU** ne permet pas de créer des menus tels que celui qui précède).

- Tel qu'il est défini ci-dessus, ce menu n'est pas sauvegardé dans **LASTMENU** quand on en sort.
  Pour effectuer une telle sauvegarde, il faut placer l'instruction **#4139Bh SYSEVAL** (*"LastMenu=CurMenu"*) dans l'objet correspondant à la procédure de sortie du menu.

  En prenant l'exemple précédent, il suffit de modifier les lignes:
  « CLLCD "See You Later" 3 DISP » #419C4h SYSEVAL
  et d'écrire:
  « CLLCD "See You Later" 3 DISP #4139Bh SYSEVAL » #419C4h SYSEVAL
  Il est à remarquer qu'alors, au retour dans ce menu, par **LASTMENU**, on retrouve toutes les particularités définies ci-dessus.

Voici pour terminer cette partie quelques SYSEVALs, dont certains sont de la meilleure eau (les trois premiers).

Adresse	Mnémoniques et commentaires
#151A6h	->StoMenu |transforme menu courant en menu de saisie (sur | toute sa longueur.
#152D2h	+NullEntries(li,s) |Ajoute à la liste li (représentant un menu), s-1 | entrées de menu vides (voir ci-dessus)
#3EC71h	NullEntry |Entrée de menu vide (c'est une liste dont le premier | élément est un chaîne vide, et dont le second est | un programme éxécutant un bip)
#3B234h	MainMenuList |Adresse où se trouve une liste de 59 objets, ces | objets étant les 59 menus intégrés de la 48, dans | l'ordre utilisé par l'intruction MENU.
#43716h	RclStackMenu | Rappelle le contenu du menu Pile Interactive.
#41373h	DoMenuMTH |Passe dans le menu MTH, en vidant le contenu de | LastMenu (en y plaçant le menu MTH...)

◆ CRÉÉR UN MENU DE SAISIE:

Le tableau ci-dessus contient l'adresse **#151A6h** (Mnémonique "**→*StoMenu***") qui transforme le menu courant en menu de saisie. Il est intéressant de créer un menu dont certaines entrées sont des labels de saisie, les autres étant des labels "*normaux*" (exemple du menu **SOLVR**).

Exemple:
On veut construire un menu de saisie de deux variables notées **A** et **B**. Les touches **A,B** étant en position 1 et 3. Une touche "**EXIT**", en position 6 dans le menu, permet de revenir au menu précédent (cette touche ne doit donc pas être une touche de saisie de valeur). Les touches 2,4 et 5 doivent être vides.
Le programme suivant apporte la solution:

```
«   { A { } B { } { } { "EXIT" « 0 MENU » } }
    TMENU #151A6 SYSEVAL
»
```

Les adresses suivantes donnent un peu plus de précision sur les menus de saisie, pour ceux ou celles qui voudraient aller plus loin.

Adresse	Mnémoniques et commentaires
#1548Ch	PutStoLabel(s,_) Analogue à PutLabel(s,_), d'adresse #3A260h, à ceci près que si l'objet au niveau 1 est une chaîne vide ou un nom global, c'est un label de saisie qui est placé à la colonne N°s du menu.
#4019Dh	StdStoMenuNS | Ces 3 objets définissent l'action stan-
#401D4h	StdStoMenuLS | -dard d'une touche d'un menu de saisie,
#4021Fh	StdStoMenuRS | non shiftée, ou left/right shiftée.

◆ L'ASPECT GRAPHIQUE DES MENUS:

Dans cette deuxième partie, nous allons examiner le rapport entre les menus et les objets-graphiques. Il y a plusieurs problèmes à considérer:

- Les labels de menu (c'est-à-dire les images correspondant à chaque entrée de ce menu): Comment sont-ils créés, ou placés sur la ligne de menu?

- Où intervient la différence entre des labels classiques, ou des labels de saisie (menu **SOLVR**), ou des labels de sous-menu (menu **MTH**) ? (sans compter la variante qui consiste à placer un petit carré dans un menu comme c'est le cas pour certaines entrées du menu **MODES**)

- La ligne des labels de menu occupe une zone rectangulaire à l'écran (c'est un **GROB** de dimensions 131x8). Peut-on modifier l'emplacement de cette zone? Peut-on la masquer ?

Les adresses qui suivent répondent à ces questions. Voilà tout d'abord les SYSEVALs permettant de fabriquer un label de menu (et il y a plusieurs solutions), et à partir de plusieurs types d'objets (*grob*, *chaîne*, *nom*, *programme* ...).

Tous les labels obtenus sont des **GROBS** de dimension 21x8, c'est-à-dire la dimension habituelle dans un menu habituel (mais comme on l'a vu plus haut, on peut maintenant créer des menus tout à fait fantaisistes. Nous en verrons encore un exemple plus loin).

Adresse	Mnémoniques et commentaires
#12645h	RclMenuGrob |(LM) Rappelle le Grob 131x8 du menu (sans newob)
#3A328h	MakeLabel(st) |Donne un grob de dimensions 21x8, représentant le label de menu associé à "st".
#3A38Ah	MakeLabel▪(st) |Comme ci-dessus, pour un label avec petit carré comme celui de certaines entrées du menu MODE.
#3A3ECh	MakeDirLabel(st) |Comme MakeLabel(st), mais le label obtenu est un label d'entrée de sous-répertoire.
#3A44Eh	MakeStoLabel(st) |Comme MakeLabel(st), mais le label obtenu est un label de menu de saisie (comme dans SOLVR)
#3EC99h	MakeLabel▪(st,?) |Si le booléen au niveau 1 est égal à True, alors éxécute "MakeLabel▪(st)", sinon "MakeLabel(st)"
#3ED0Ch	MakeLabel(st,?) |Si le booléen au niveau 1 est égal à True, alors éxécute "MakeLabel(st)", sinon "MakeLabel▪(st)"
#3A2FBh	MakeLabel(st,di?) |Si l'objet situé au niveau 1 est un Directory, alors éxécute "MakeDirLabel(st)", sinon "MakeLabel(st)"

Voici les adresses des Rpl correspondant à la création de nombreux labels de menus intégrés: tous ces SYSEVALs produisent un objet graphique de dimensions 21x8.

Adresse	Label	Adresse	Label	Adresse	Label
#3BB64h	SYMLabel	#3BBABh	BEEPLabel	#3BC03h	CNCTLabel
#3BE31h	SOLVRLabel	#3BEC7h	PLOTRLabel	#3BF26h	PTYPELabel
#3C4D8h	SETLabel	#3C51Fh	ADJSTLabel	#3C56Ah	ALARMLabel
#3C847h	RPTLabel	#3DB60h	HEXLabel	#3DBA2h	DECLabel
#3DBE4h	OCTLabel	#3DC26h	BINLabel	#3DEC0h	FCNLabel
#3E096h	+/-Label	#3E173h	ZOOMLabel	#3E58Bh	INSLabel
#3E71Fh	SETUPLabel	#3E779h	STDLabel	#3E7BBh	FIXLabel
#3E7EEh	SCILabel	#3E821h	ENGLabel	#3E854h	MLLabel
#3E894h	DEGLabel	#3E8D6h	RADLabel	#3E90Dh	GRADLabel
#3E953h	XYZLabel	#3E98Fh	R<ZLabel	#3E9F3h	R<<Label
#3EA52h	CMDLabel	#3EAA3h	STKLabel	#3EB21h	ARGLabel
#3EB6Dh	FM,Label	#3EBB4h	CLKLabel	#3EBFBh	MODLLabel
#461B8h	GO->Label	#461E6h	GOvLabel	#48C5Fh	FASTLabel

Une fois que les labels de menu sont fabriqués (instructions "***MakeLabel***"), il faut les placer quelque part sur la ligne des menus. Les SYSEVALs qui suivent répondent à cette question:

Adresse	Mnémoniques et commentaires
#3A260h	PutLabel(s,_) |"s" est ici un N° de colonne, en pixels. |L'objet du niveau 1 peut ici être un grob, une chaîne, un programme, ou un nom global. |Superpose graphiquement l'objet du niveau 1 à la ligne de menu, à partir de la colonne N°s |A faire suivre de 4Freeze pour geler cette image. |Les "s" logiques: <0h>,<16h>,<2Ch>,<42h>,<58h>,<6Eh>. |Celà ne modifie pas la signification de la touche
#3A297h	PutLabel(s,go) |Forme interne de "PutLabel(s,_)", pour un grob.
#3A2B5h	PutLabel(s,st) |Forme interne de "PutLabel(s,_)", pour une chaîne. Appelle en fait MakeLabel(st), puis PutLabel(s,go)
#3A2C9h	PutLabel(s,prog) |Forme interne de "PutLabel(s,_)", pour un programme.
#3A2DDh	PutLabel(s,gn) |Forme interne de "PutLabel(s,_)", pour un nom. |Le nom est transformé en chaîne. Si c'est un nom connu contenant un Directory, on obtient un label d'entrée de sous-menu, sinon un label normal.
#41904h	PutCurLabel(s,_) |Comme "PutLabel(s,_)" en #3A260h, sauf que le label est créé avec le format usité dans ce menu (par ex: un menu de sous-menus comme MTH)

Le menu occupe confortablement les 8 dernières lignes de l'écran. C'est beaucoup trop monotone. Quelques SYSEVALs vont nous permettre de rompre cette désespérante routine.

Les adresses ci-dessous (sauf la première d'entre elles, un peu inclassable) vont permettre d'effacer temporairement le menu, de le faire réapparaître, ou de le déplacer.

Il y a deux façons de faire disparaître le menu:

- Ou bien on le recouvre par un objet-graphique vide (c'est le cas avec "***ClearMenuGrob***", à l'adresse **#51125h**)
- Ou bien on élargit la taille du **GROB** contenant l'image de la pile, repoussant du même coup le **GROB** du menu vers le bas, assez pour qu'il disparaisse. C'est le cas avec "***MenuOff***" (**#4E2CFh**)

Adresse	Mnémoniques et commentaires (LM si langage machine)
#17F15h	Y/N-Menu |(LM) Affiche un menu "YES NO" en superpo- -sition du menu courant. Faire suivre d'un 4Freeze pour geler l'image après arrêt du programme. Celà ne change pas le sens de la ligne de menu en cours
#4E2CFh	MenuOff |Efface la ligne de menu. Faire suivre de 4 Freeze pour geler l'affichage.
#4E347h	MenuOn |Réaffiche la ligne de menu, si elle avait été effacée par "MenuOff", ou déplacée par "DoRowMenu(s)" |Consiste en fait en "<37h> DoRowMenu(s)"
#4E360h	MenuOff? |Teste si la ligne à laquelle est affiché le menu est différente de <37h>, c-à-d si le menu n'est pas affiché à la ligne habituelle (par ex: <3Fh> s'il a été caché par "MenuOff".
#4E37Eh	DoRowMenu(s) |(LM) Définit la ligne où afficher le menu en cours. |Ex: Avec s=<37h> le menu est affiché normalement, mais avec s=<3Fh>, il est caché.
#50701h	RclRowMenu |(LM) Rappelle la ligne où est affiché le menu. |Le résultat est un entier-système, comme, par ex: <37h> si le menu est en ligne normale <3Fh> si il a été caché par MenuOff
#51125h	ClearMenuGrob |Efface le grob des menus. Faire suivre de 4Freeze pour geler l'affichage (Ici, il ne s'agit pas comme dans "MenuOff" de descendre la ligne des menus en augmentant la taille du grob de la pile)
#6A478h	HideGraphMenu |Efface menu. Seules sont autorisées les touches de curseur qui ne font rien, et ON qui sort. |Il s'agit en fait de la redéfinition du clavier que l'on trouve dans l'environnement GRAPH.

◆ RÉCRÉATION:

Le programme suivant est une application amusante des SYSEVALs de ce chapitre. Placez vous par exemple dans un répertoire intégré à la HP48 (par exemple le menu **MTH\PARTS**); tapez **FUN** et vous verrez.

'**FUN**': (ChekSum: #D4F5h; Taille: 170 octets)

```
« « 1 6 FOR k k #18CD7h SYSEVAL #4A41h SYSEVAL
     NEXT
     1 10 START 6 RAND * CEIL ROLLD NEXT
     1 6 START « FUN » 2 →LIST 6 ROLLD NEXT
     6 →LIST
   » #40F86h SYSEVAL
»
```

LA GESTION DU CLAVIER

Dans ce chapitre, nous allons aborder les questions relatives à l'utilisation du clavier, de deux points de vue:

- Lectures du clavier "*au vol*", ou en attendant qu'une touche soit effectivement actionnée. Gestion du *buffer* du clavier.
- *Assignations* de touches pour le mode **USER**.

◆ Le buffer du clavier:

Quand une touche est actionnée, elle est placée dans une *zone-tampon* que nous appelerons *buffer* du clavier, et qui permet de mémoriser 15 touches successives.

Quand aucun programme n'est en cours d'éxécution, une touche actionnée est immédiatement traitée (et en même temps elle est évidemment retirée du *buffer*).

Au contraire, pendant l'éxécution d'un programme, les touches qu'actionne l'utilisateur sont stockées dans le *buffer* du clavier (dans l'ordre chronologique).

Si l'on actionne plus de 15 touches avant le retour au mode direct, la HP48 émet un bip vous avertissant que le *buffer* du clavier est plein et que les touches éxcédentaires ne sont pas mémorisées.

Après l'éxécution d'un programme, et au retour dans le mode direct, les touches présentes dans le *buffer* sont traitées (et retirées de celui-ci) dans l'ordre naturel (de la plus ancienne à la plus récente).

Dans le *buffer* du clavier, chaque touche est codée sur un octet (deux quartets consécutifs); cet octet contient le numéro de touche sur le clavier (lu de gauche à droite, et de haut en bas: ces numéros vont donc de 1 pour la première touche de menu à 49 pour la touche +)

En particulier, les touches α, ┌─> (*shift droit*), <─┐ (*shift gauche*) comptent comme n'importe quelle autre touche.

Certaines combinaisons utilisent un *double shift* (par exemple, le caractère **$** est obtenu par α <─┐ 4). Ces touches occupent trois positions distinctes (et consécutives) dans le buffer du clavier. Il ne sera donc possible de mémoriser que 5 de ces combinaisons.

◆ L'appui sur la touche ON:

Il est faux de penser qu'un appui sur la touche **ON** interromp automatiquement l'éxécution du programme en cours. Celà est certes vrai pour les programmes constitués d'objets standards de la HP48. Celà n'est plus forcément vrai dès lors que l'on écrit directement en langage machine, ou même si on écrit en "*Externals*" (appels directs à des adresses de Rpl non standards).

Certaines routines de lecture du clavier présentées ci-dessous traitent même la touche **ON** comme n'importe quelle autre touche.

◆ L'instruction KEY:

L'instruction intégrée **KEY** effectue une lecture "*au vol*" du clavier, et renvoie l'un des deux résultats suivants:

- 0 au niveau 1 de la pile, si le *buffer* du clavier est vide.
- 1 au niveau 1 de la pile, et un réel **x** au niveau 2, si le *buffer* du clavier est non vide.

Le réel x identifie ici la plus ancienne des touches mémorisées dans le *buffer* (et du coup cette touche en est retirée), suivant le format suivant: **x = 10*L + C**, où **L** et **C** sont respectivement la ligne (de 1 à 9) et la colonne (de 1 à 6) de cette touche.

◆ Les instructions 0 WAIT et -1 WAIT:

Les séquences **0 WAIT** et **-1 WAIT** suspendent au besoin le déroulement du programme en cours jusqu'à ce qu'une touche complète (avec des *shifts* éventuels) apparaisse dans le *buffer* du clavier.

La touche complète ainsi reconnue est alors retirée du *buffer* et retournée au niveau 1, sous la forme d'un réel **x** s'écrivant **x = 10*L + C + P/10**, où **L** est le numéro de ligne, **C** le numéro de colonne, et où **P** est un entier (compris entre 1 et 6) désignant le "*plan*" du clavier auquel appartient la touche.

Rappelons les différents plans du clavier:

1: non shifté	2: shifté par <-┐	3: shifté par ┌->
4: shifté par α	5: shifté par α <-┐	6: shifté par α ┌->

Ainsi un appui sur α <-┐ 4 (caractère **$**) correspond-il à la position 72.5 au sens de l'instruction **0 WAIT**.

◆ La scrutation du clavier:

Les problèmes traitant de la *scrutation* du clavier sont particulièrement importants dans toute programmation avancée.

Il est essentiel de pouvoir contrôler les accès au clavier, pour en tirer les conséquences quant au déroulement du programme en cours.

On peut envisager deux façons de lire le clavier:

- "*Au vol*", c'est-à-dire sans attendre qu'une touche soit effectivement actionnée.
  On pourra alors uniquement contrôler qu'une touche a été pressée (sans s'inquiéter de savoir laquelle), ou au contraire lire la touche actionnée. La touche ainsi lue au vol pourra être retirée du *buffer* du clavier, ou y rester.

- En attendant qu'une touche soit effectivement actionnée. Dans ce cas, le fait de savoir quelle touche a été pressée est en général important.

Le tableau suivant contient les SYSEVALs qui traitent de ces questions.

Adresse	Commentaires (LM si langage machine)
#00D71h	ClearKeyPressed |Vide le buffer du clavier, sans traiter les touches | qui pourraient s'y trouver.
#04708h	CheckKey? |Teste si le buffer du clavier est non vide. | La réponse est 1:false ou 2:<n> 1:true, ou n est | est le numéro de la touche (de <1h> à <31h>=49) | la plus ancienne. |Cette touche N'EST PAS retirée du buffer.
#04714h	PopKey? |Comme ci-dessus, mais touche lue retirée du buffer
#3FE44h	Plane=>Key |Transforme <80h> en <1Eh>, <40h> en <23h> et <C0h> | en <28h>. Redonne donc à un code représentant un | plan du clavier le N° de touche qui lui correspond |(Voir "GetKey?" à l'adresse #420A0h)
#41CA2h	rc.p=><n><p> |Traduction "rc.p vers <n>,<p>", où rc.p est un réel | décrivant l'emplacement d'une touche complète, au | sens de l'instruction "0 WAIT", et où: | p et n sont 2 entiers-systèmes ("p" pour le plan | de la touche, de <1h> à <6h>, et n pour la position | sur le clavier, de <1h> à <31h>=49)
#41D92h	<n><p>=>rc.p |Traduction inverse de rc.p=><n><p>
#41F65h	GetTouch |Attend au besoin que le buffer du clavier contienne | une touche complète, et donne le résultat suivant: | 2:n 1:p, où n est le numéro de la touche, et où p | est le plan du clavier auquel elle appartient. | (n et p sont tous deux des entiers-système) |Par exemple: α TAN donne 2:<15h>,1:<4h>. |La touche complète est retirée du buffer.
#420A0h	GetKey? |Attend au besoin que le buffer du clavier contienne | au moins un code de touche, et donne le résultat | 2:n, 1:true si "tout s'est bien passé", en parti- | -culier si aucune alarme ne s'est déclenchée, in- | -terrompant l'attente. |Sinon la réponse est 1:False |Ici n est un entier système indiquant le numéro de | la touche sur le clavier. |N.B: α,<¬,⌐> sont respectivement représentées ici | par les codes <80h>,<40h>, et <C0h>
#42159h	GetNoAttnKey? |Comme ci-dessus, sauf que un appui sur la touche ON | renvoie le résultat 1:False et s'accompagne d'une | sortie prématurée du Rpl en cours d'éxécution.
#42402h	KeyPressed? |Teste si le buffer du clavier est non vide. La seule | réponse est 1:false ou 1:true. |Aucune touche n'est retirée du buffer

LA GESTION DU CLAVIER

Comme il a été dit plus haut, la touche **ON** est, dans certains cas, considérée comme une touche comme les autres, et son action ne consiste pas nécessairement à interrompre prématurément le programme en cours.

Il apparaît que la HP48 gère un *flag*, situé à l'adresse **#70679h**, et qui permet de signaler l'appui sur la touche **ON** ou l'irruption d'autres événements susceptibles d'interrompre le programme en cours, comme une alarme, ou une interruption consécutive à un défaut des piles, ou encore l'addition d'une carte dans un port, etc....)

Adresse	Commentaires (LM si langage machine)
#05040h	RclAttnFlag |(LM) Donne sous forme d'un entier-système, le contenu | de l'adresse #70679h (AttnFlag).
#05068h	ClearAttnFlag |(LM) Met à zéro l'adresse en #70679h (AttnFlag), sans | retirer la touche Attn (ON) si elle est présente | dans le buffer du clavier.
#42262h	AttnPressed? |Teste si la touche Attn (ON) a été actionnée. |La réponse est un booléen (True ou False)
#4243Eh	AbortIfAttn |Interruption du programme en cours si la touche Attn | (ON) a été actionnée. Vide le buffer du clavier, | et annule le contenu de l'adresse #70679h.

Voici maintenant quelques SYSEVALS qui permettent de rappeler sur la pile, sans l'évaluer, l'objet correspondant à telle ou telle touche du clavier (*shiftée* ou non).

Adresse	Commentaires (LM si langage machine)
#3FB1Ah	RclKey(2s) |Place sur la pile, sans l'évaluer, l'objet qui | correspond à la touche N°s2, sur le plan N°s1.
#3F831h	RclKeyP1(s) |Rappelle sans l'évaluer l'objet qui correspond à | la touche N°s du plan 1 du clavier.
#3F859h	RclKeyP2(s) |Rappelle sans l'évaluer l'objet qui correspond à | la touche N°s du plan 2 du clavier.
#3F87Ch	RclKeyP3(s) |Rappelle sans l'évaluer l'objet qui correspond à | la touche N°s du plan 3 du clavier.
#3F89Fh	RclKeyP4(s) |Rappelle sans l'évaluer l'objet qui correspond à | la touche N°s du plan 4 du clavier.
#3F8C2h	RclKeyP5(s) |Rappelle sans l'évaluer l'objet qui correspond à | la touche N°s du plan 5 du clavier.
#3F8E0h	RclKeyP6(s) |Rappelle sans l'évaluer l'objet qui correspond à | la touche N°s du plan 6 du clavier.

◆ Routines des touches du clavier:

Quand on actionne une touche quelconque du clavier (éventuellement *shiftée*), que l'on soit en mode *direct* (pas de ligne de commande, pas de programme en cours) ou en édition d'une ligne de commande, la HP48 éxécute un programme spécifique à cette touche.

L'action de ce programme peut dépendre du "*contexte*". Prenons par exemple la touche +/−:

- Si cette touche est pressée alors qu'aucune ligne de commande n'est en cours, la HP48 éxécute l'instruction **NEG**.
- Si une ligne de commande est en cours, la HP48 placera le signe − ou l'instruction **NEG** dans la ligne de commande (celà dépend du mode en cours: **ALG**, **PRG** ou **ALG PRG**, et du mot que l'on est en train de saisir).

Il est particulièrement instructif d'étudier ces routines. On y apprend par exemple beaucoup de choses sur la façon dont la HP48 gère sa ligne de commande (voir chapitre suivant).

Les adresses qui suivent regroupent tous les SYSEVALs "*intéressants*" (beaucoup de routines sont en fait des adresses standards de la HP48. Quand on actionne la touche **HOME**, par exemple, on éxécute l'instruction **HOME**...).

Adresse	Mnémonique	Adresse	Mnémonique	Adresse	Mnémonique
#3A5CDh	\ON	#3A5F0h	\<=	#3A645h	\Down
#3A69Ah	\'	#3A6B3h	_	#3A6CCh	\()
#3A6E5h	\I/O	#3A6FEh	\LASTMENU	#3A71Ch	\NXT
#3A735h	\PREV	#3A753h	\.	#3A77Bh	\,
#3A7A3h	\TIME	#3A7C6h	\EDIT	#3A7F3h	\ENTER
#3A80Ch	\┌>Right	#3A834h	\Left	#3A893h	\<┐
#3A8ACh	\┌>┌>	#3A8ACh	\<┐<┐	#3A8C5h	\┌>
#3A8DEh	\Right	#3A93Dh	\Up	#3A992h	\STO
#3A9CEh	\┌>ON	#3AA0Ah	\α	#3AA37h	\α<┐
#3AA50h	\α<┐<┐	#3AA50h	\α┌>┌>	#3AA69h	\α┌>
#3AA82h	\+/-	#3AAEBh	\┌>DEL	#3AB09h	\DEL
#3AB59h	\ALGEBRA	#3AB72h	\CST	#3AB8Bh	\{}
#3ABA4h	\[	#3ABBDh	\α«»	#3ABD6h	\«»
#3ABEFh	\::	#3AC08h	\#	#3AC21h	\""
#3AC3Ah	\EEX	#3ACBCh	\LASTCMD	#3ACF8h	\LASTSTACK
#3AD57h	\MTH	#3AD70h	\MEMORY	#3AD89h	\┌>MEMORY
#3ADA2h	\MODES	#3ADBBh	\┌>CST	#3ADD4h	\Newline
#3ADEDh	\PLOT	#3AE1Ah	\PRG	#3AE33h	\PRINT
#3AE4Ch	\<┐Down	#3AE6Fh	\<┐Up	#3AE88h	\SOLVE
#3AEB5h	\STAT	#3AEE2h	\┌>STAT	#3AF05h	\UNITS
#3AF1Eh	\┌>UNITS	#3AF37h	\VAR	#3AF50h	\<┐ENTER
#3AF69h	\┌>Down	#3AFE6h	\┌>Left	#3B00Eh	\┌>Up
#3B036h	\αα	#3B068h	\┌>ENTER	#3B081h	\ENTRY
#3B0DBh	\USR	#3B12Bh	\VISIT	#3B15Dh	\2D
#3B18Fh	\3D	#3B1B7h	\POLAR	#3B1CBh	\RAD
#3B1DFh	\PURGE	#3B211h	\┌>NXT	#3BE54h	\┌>SOLVE
#3F167h	\α<┐7	#3F17Bh	\α┌>7	#3F18Fh	\α<┐8
#3F1A3h	\α┌>8	#3F1B7h	\α<┐9	#3F1CBh	\α┌>9
#47AE7h	\┌>TIME	#47B5Ah	\┌>ALGEBRA	#48FD6h	\┌>PLOT

Quand on entre dans un menu intégré de la HP48, les 6 touches de menu prennent une signification particulière. Voici les adresses des routines éxécutées par la HP48 quand on actionne l'une de ces touches de menu, *shiftée* ou non.

Toutes les adresses qui sont présentées ici ne sont pas bonnes à éxécuter par SYSEVAL. En effet, dans certains environnements comme **GRAPH**, **MATRIX-WRITER**, **EQUATION-WRITER**, ou encore les catalogues, la HP48 créé un certain nombre de *variables locales* avant d'afficher le menu de l'application.

Il est dès lors illusoire (et même risqué !) de vouloir éxécuter les adresses internes à ces environnements en faisant l'*impasse* sur ces variables locales.

Les adresses que vous trouvez ici ont, cependant, un intérêt indéniable car leur étude donne de nombreux renseignements permettant de mieux comprendre le fonctionnement de la HP48.

Adresse	Mnémonique	Adresse	Mnémonique	Adresse	Mnémonique
IO et IO/SETUP		#3BA5Bh	\BAUD	#3BA88h	\PARITY
#3BAB5h	\CKSM	#3BAE2h	\TRANSIO	#3E742h	\SETUP
Menu MODES		#3BB88h	\SYM	#3BBD1h	\BEEP
#3BC29h	\CNCT	#3DB84h	\HEX	#3DBC6h	\DEC
#3DC08h	\OCT	#3DC4Ah	\BIN	#3E79Dh	\STD
#3E876h	\ML	#3E8B8h	\DEG	#3E8EFh	\RAD
#3E935h	\GRAD	#3E96Ch	\XYZ	#3E9C6h	\R<Z
#3EA25h	\R<<	#3EA76h	\CMD	#3EAC7h	\STK
#3EB45h	\ARG	#3EB91h	\FM,	#3EBD8h	\CLK
Menu EDIT		#3E2F6h	\<-SKIP	#3E378h	\SKIP->
#3E3FDh	\<-DEL	#3E47Fh	\┌-><-DEL	#3E4E6h	\DEL->
#3E54Ah	\┌->DEL->	#3E5AFh	\INS	#3E5EEh	\α<-┐α
TIME et sous-menus		#3C4F7h	\SET	#3C542h	\ADJST
#3C58Bh	\ALARM	#3C5C2h	\ACK	#3C5EFh	\ACKALL
#3C68Bh	\HR+	#3C6B9h	\HR-	#3C6E9h	\MIN+
#3C719h	\MIN-	#3C749h	\SEC+	#3C779h	\SEC-
#3C866h	\RPT	#3C990h	\NONE	#3CA1Ah	\A/PM
#3CA51h	\12/24	#3CA84h	\M/D		
Menu SOLVR		#15343h	\NXEQ		
Menu PLOT		#3BEEAh	\PLOTR	#3BF49h	\PTYPE
#3BF86h	\NEW	#3BFCFh	\STEQ		
Menu PLOTR		#3C0C3h	\ERASE	#3C0FFh	\DRAW
#3C11Dh	\<-┐DRAW	#3C14Fh	\AUTO	#3C19Ah	\XRNG
#3C1B8h	\┌->XRNG	#3C1E5h	\YRNG	#3C203h	\┌->YRNG
#3C230h	\INDEP	#3C27Bh	\DEPND	#3C2BCh	\RES
#3C2F8h	\CENTR	#3C334h	\SCALE	#3C393h	\AXES
#3C3D4h	*H	#3C3FCh	*W	#4973Ah	\RESET
#4AC34h	\┌->CENTR	#4ADB0h	\┌->SCALE	#4AF63h	\┌->INDEP
#4AF9Fh	\┌->DEPND	#4AFDBh	\┌->RES	#4B026h	\┌->AXES
#4B1F2h	\┌->PDIM				

Voici la suite du tableau précédent....

Adresse	Mnémonique	Adresse	Mnémonique	Adresse	Mnémonique
Menu ΣDAT		#3CAC0h	\Σ+	#3CAE8h	\<┐Σ+
#3CB24h	\CLΣ	#3CB5Ch	\NEW	#3CBA7h	\STOΣ
#3CC15h	\XCOL	#3CC56h	\YCOL	#3CC92h	\BARPLOT
#3CCC9h	\HISTPLOT	#3CCF6h	\SCATRPLOT	#3EC1Ch	\MODL
#4A4A6h	\EDITΣ				
IO et IO/SETUP		#3BA5Bh	\BAUD	#3BA88h	\PARITY
#3BAB5h	\CKSM	#3BAE2h	\TRANSIO	#3E742h	\SETUP
Menu PRG\BRCH		#3DC77h	\<┐IF	#3DC95h	\┌─>IF
#3DCD6h	\<┐CASE	#3DCF9h	\┌─>CASE	#3DD30h	\<┐START
#3DD49h	\┌─>START	#3DD80h	\<┐FOR	#3DD99h	\┌─>FOR
#3DDD0h	\<┐DO	#3DE0Ch	\<┐WHILE	#3DE48h	\<┐IFERR
#3DE66h	\┌─>IFERR				
Pile interactive		#43E40h	\ECHO	#43EE0h	\VIEW
#43F03h	\┌─>VIEW	#43F6Ch	\PICK	#43F85h	\ROLL
#43F9Eh	\ROLLD	#43FB7h	\LEVEL	#43FD5h	\DUPN
#43FEEh	\DRPN	#4400Ch	\KEEP	#4405Ch	\->LIST
Matrix Writer		#46181h	\->STK	#46242h	\GO->
#4626Fh	\GO|	#46305h	\VEC	#46378h	\-ROW
#46481h	\-COL	#4659Eh	\+ROW	#466EDh	\+COL
#46873h	\<-WID	#46945h	\WID->	#469F4h	\EDIT
Catalogue Stat.		#485D5h	\1-VAR	#48669h	\PLOT
#486CDh	\2-VAR	#4872Fh	\EDIT		
Menus RULES		#4877Ah	\SOLVR	#487DEh	\PLOTR
Catalogue Equat.		#48825h	\EQ+	#488DEh	\<┐EQ+
#48931h	\EDIT	#489A9h	\PURGE	#48A88h	\->STK
#48B56h	\VIEW	#48C85h	\FAST	#48CD0h	\ORDER
Catalogue Alarms		#48D37h	\EDIT	#48DA6h	\PURGE
#48E0Ah	\EXECS				
GRAPH et GRAPH\FCN		#3DFF1h	\EXIT	#3E06Eh	\NXEQ
#49641h	\<┐Z-BOX	#497FDh	\SLOPE	#4978Fh	\FCN
#4984Dh	\ROOT	#4988Ch	\AREA	#49948h	\ISECT
#49A1Ch	\EXTR	#49A6Ch	\F(X)	#49AB5h	\F'
#4DAB2h	\+/-	#4DEC2h	\DOT+	#4DF03h	\DOT-
#4E2ACh	\KEYS	#4E3CAh	\MARK	#4E55Ah	\BOX
#4E5E6h	\CIRCLE	#4E631h	\LINE	#4E663h	\TLINE
#4E68Bh	\COORD	#4E753h	\CENTR	#4E776h	\Z-BOX
#4E875h	\LABEL	#4ECE4h	\REPL	#4ED70h	\SUB
#4EDB6h	\DEL	#4EE1Ah	\ZOOM		
Menu LIBRARY		#3F420h	Port0	#3F475h	Port1
#3F4CFh	Port2				
#4365Dh	\\|STK (dans menu EDIT, ou Matrix-Writer)				

◆ Assignations de touches:

On sait qu'il est possible de *réassigner* tout ou partie des touches du clavier, ces réassignations étant valables à l'intérieur de l'environnement **USER**. Les instructions standards de La HP48 qui traitent de ces questions sont **ASN**, **DELKEYS**, **RCLKEYS**, **STOKEYS**.

Un point important est de savoir si les touches non explicitement *réassignées* gardent leur définition standard en mode **USER**, ou si au contraire elles doivent y être complètement *désactivées*.

Les intructions ci-dessus admettent des formes particulières permettant de traiter ce dernier point. Plus généralement, on se reportera au manuel de l'utilisateur de la HP48 pour tous détails sur ces questions (pages 225 et suivantes).

Voici les (nombreux) SYSEVALs ayant un rapport direct avec la *réassignation* des touches du clavier. Je vous renvoie au chapitre traitant du *répertoire caché*, et qui donne les explications concernant les variables **'UserKeys'** et **'UserKeys.CRC'**.

Adresse	Commentaires (LM si langage machine)
#11086h	ChkUserKeys |Teste les contenus de 'UserKeys' et de 'UserKeys.CRC' | et supprime toutes les assignations de touches en | cas de non conformité. |Si tout est OK, appelle tout de même "StoAdrUserKeys" | en #41F2Ch, avec pour objet du niveau 1 le contenu | de la variable 'UserKeys'.
#110DBh	NoUserKeys |Supprime toutes les assignations de touches, mais ne | redonne pas aux touches qui n'étaient pas assignées | leur déf. standards (si elles en ont été privées | par 'S' DELKEYS), comme le fait "DelUserKeys" en | #41F13h. |Se contente d'éxécuter "StoUserKeys({})"
#3FF75h	StdDefActive? |(LM) Teste si les touches non réassignées gardent | leurs définitions standards. |La réponse est le booléen True ou False
#3FF86h	StoKeys'S' |(LM) Les touches non réassignées reprennent leur | signification standard (utile si elles en avaient | été privées par 'S' DELKEYS) |Le nom 'S',ici implicite, n'a pas à être sur la pile
#3FF97h	DelKey'S' |(LM) Les touches non réassignées deviennent inactives | dans le mode USER (elles perdent leur signification | standard). |Le nom 'S',ici implicite, n'a pas à être sur la pile
#41A39h	'UserKeys |Rappelle le nom 'UserKeys' sur la pile
#41A5Fh	'UserKeys.CRC |Rappelle le nom 'UserKeys.CRC' sur la pile
#41AA1h	StoKeys(li) |Forme interne de STOKEYS, pour une liste.

Adresse	Commentaires (LM si langage machine)
#41B28h	asn(_,r) |Forme interne de ASN, pour un réel. Exécute en fait "rc.p=><n><p>", puis "asn(_,2s)"
#41B3Ch	DelKeys(li) |Forme interne de DELKEYS, pour une liste.
#41B69h	DelKeys(r) |Forme interne de DELKEYS, pour un réel. |Si r=0, éxécute "DelUserKeys". |Si r<>0, éxécute "rc.p=><n><p>", puis "DelKeys(_,2s)"
#41B8Ch	DelKeys(_,2s) |Forme interne de DELKEYS. Ici s2 est le N° de touche, et s1 est le plan du clavier. |Exécute en fait: "asn({},2s)"
#41BA5h	StoKeys(nm) |Forme interne de STOKEYS, pour un nom. Seulement accepté si ce nom est 'S' (appelle "ChkName'S'")
#41BB9h	DelKeys(nm) |Forme interne de DELKEYS, pour un nom. Seulement accepté si ce nom est 'S' (appelle "ChkName'S'")
#41BCDh	ChkName'S' |Vérifie que l'objet au niveau 1 est bien le nom 'S' (global ou local). Utilisé par StoKeys et RclKeys. |En cas de différence, Erreur "Bad Argument Value"
#41C02h	RclKeys! |Donne la liste des réassignations de touches, non précédée du 'S' éventuel (indépendamment donc du fait que les touches non réassignées gardent leur signification standard ou pas). |La liste est obtenue au format de l'instruction RCLKEYS, et pas au format interne de 'UserKeys'
#41E78h	asn(_,2s) |Forme interne de ASN. Ici, s2 est le N° de touche, et s1 est le plan du clavier.
#41E32h	StoUserKeys(li) |Stocke la liste dans 'UserKeys' (elle doit être au format de UserKeys, c-à-d une liste de 49 listes de 6 sous-listes). La variable 'UserKeys.CRC' du répertoire caché est modifiée en conséquence.
#41F13h	DelUserKeys |Annule toutes les réassignations de touches, les touches qui n'étaient pas réassignées reprenant leur signification standard. (éxécute "StoUserKeys({})", puis "StoKey'S'") |"DelUserKeys" est la forme interne de: 0 DELKEYS
#41F2Ch	StoAdrUserKeys |(LM) l'adresse de l'objet du niveau 1 est placée à l'adresse de la liste UserKeys, en #705A6h
#41F3Fh	RclUserKeys |Rappelle sur la pile le contenu de 'UserKeys'. |En fait, place au niveau 1 l'adresse contenue à l'adresse #705A6, ce qui revient au même.
#41F52h	ClearAdrUserKeys |(LM) met à zéro l'adresse #705A6h, où se trouve l'adresse de la liste UserKeys.

LA LIGNE DE COMMANDE

Ce chapitre peut sembler relativement long, eu égard à l'importance du sujet. Pourtant, le nombre de SYSEVALs qui touchent de près à la création, à la modification, et à l'évaluation d'une ligne de commande est considérable, et vous n'en trouverez ci-après qu'une partie, les autres adresses étant ou bien encore nimbées de mystère, ou bien trop difficiles à décrire en quelques mots.

Il faut se pencher longtemps sur le sujet, comme je l'ai fait, pour avoir une idée de la complexité des opérations liées à la gestion d'une ligne de commande (déplacements, insertion ou modification de caractères, effacements, etc...).

Bien sûr, les SYSEVALs qui suivent n'ont pas tous un intérêt pratique évident. Ils constituent cependant un aperçu de ce que la HP48 a de plus intime et de plus complexe. A ce titre, leur importance est indéniable, et l'étude de leur contenu est une grande leçon de programmation...

Commençons par les SYSEVALs qui traitent de la fonction **LAST CMD** (mémorisation des 4 dernières lignes de commande validées).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#18242h	NoMemComLine? |(LM) True si aucune ligne de commande n'est mémorisée
#40BDDh	StrLastCmd? |(LM) Donne la dernière ligne de commande évaluée, au niveau 2, sous forme de chaîne,et True au niveau 1 |En même temps, effectue une permutation circulaire sur les lignes de commande mémorisées. |Ne donne que False si aucune ligne de commande n'est actuellement mémorisée.
#40C76h	StoInLastCmd(st) |Place la chaîne comme dernière commande mémorisée, en en retirant les délimiteurs ". |Si le mode LASTCMD est inactif, la chaîne st est simplement détruite .
#40C94h	StoInLastCmd(st)! |(LM) Comme ci-dessus sans tester si LASTCMD est actif
#53860h	LastCmd? |(LM). Teste si LAST CMD est active.
#5386Eh	UnableLastCmd |(LM). Active LAST CMD
#5387Ch	DisLastCmd |(LM) Désactive LAST CMD, sans effacer les commandes déjà mémorisées, et qu'on pourra récupérer après un appel à "UnableLastCmd" en #5386Eh.
#5388Ah	DisLastCmd! |(LM) Désactive LAST CMD en effaçant celles qui sont déjà mémorisées.
#5389Eh	DelMemComLine |(LM). Détruit les lignes de commandes mémorisées, sans changer le mode LAST CMD.

Le tableau suivant regroupe des adresses qui permettent de sélectionner un mode particulier de saisie, pour la ligne de commande qui va débuter, ou pour celle qu'on est en train de saisir.

Ces différents modes sont désignés par un indicateur qui s'affiche éventuellement en ligne d'état. Il s'agit des modes:

- "**Alpha**". La saisie se fait (se fera) dans le clavier alphabétique, et il y a 2 cas: le mode "*minuscules*" et le mode "*majuscules*".
- **PRG** (mode *programme*).
- **ALG** (mode *algébrique*).
- Les deux modes précédents sont cumulables; quand on n'est ni dans l'un, ni dans l'autre, on est en mode "*immédiat*" (les touches contenant implicitement **ENTER** évaluent la ligne de commande).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#11501h	αMode? |(LM) Teste si on est en mode α.
#11511h	PrgMode? |(LM) Teste si on est en mode PRG.
#11533h	PrgMode |(LM) Actionne le mode PRG.
#11543h	αMode |(LM) Passe en mode α (locked) pour la prochaine ligne de commande, sans allumer l'indicateur α
#1155Ch	UnModePrg |(LM) Annule le mode PRG
#1156Ch	UnαMode |(LM) Annule le mode α institué par "Modeα"
#408AAh	NorAlg&Prg? |Répond True si on est ni en mode ALG, ni en mode PRG
#40D25h	Lockα |Verrouille le clavier α. Consiste en un appel à "Modeα" (ci-dessous) suivi d'un affichage du shift α seul.
#40D39h	UnLockα |Déverrouille le clavier α. Consiste en un appel à "UnModeα" (ci-dessous) puis l'extinction des indicateurs "shift" éventuels (α en particulier)
#42EC7h	Obj1Mode |Passe en mode PRG ou ALG suivant la nature de l'objet situé au niveau 1.
#42F30h	OnlyPrgMode |Actionne le mode PRG seul (pas de ALG)
#42FC6h	=1(s)?:NoALG/ALG |Désactive ou active le mode "ALG" selon que l'entier système du niveau 1 est égal ou différent de <1h>
#42FF8h	=1(s)?:α/Noα |Sélectionne le mode α ou l'annule selon que l'entier système du niveau 1 est égal ou différent de <1h>
#53968h	AlgMode? |(LM). Teste si on est en mode ALG.
#53976h	AlgMode |(LM). Actionne le mode ALG.
#53984h	UnAlgMode |(LM). Désélectionne le mode ALG
#53992h	MinusMode? |(LM) Teste si, pour le clavier α, c'est le mode minuscules qui est actif.
#539A0h	MinusMode |(LM). Fait passer le clavier α en mode "minuscules", sans pour autant sélectionner le mode α.
#539AEh	MajusMode |idem avec le mode "majuscules"

LA LIGNE DE COMMANDE

Les SYSEVALs qui suivent interviennent sur la façon dont le curseur apparaît dans l'édition de la ligne de commande (soit en mode "*Insertion*", soit en mode "*Recouvrement*").

Adresse	Mnémoniques et commentaires (LM si langage machine)
#42F8Fh	=1(s)?:(Ins/Ov)Cur |Sélectionne le curseur "insertion" ou "recouvrement" | selon que l'entier-système du niveau 1 est égal | ou différent de <1h>.
#53A12h	InvCursor |(LM) Bascule le curseur entre le mode "insertion" | et le mode "recouvrement".
#53A20h #53A2Eh	OverCursor |(LM) Passe le curseur en mode "recouvrement" InsCursor |(LM) Passe le curseur en mode "insertion"
#53A3Ch	InsCursor? |(LM) Teste si le curseur est en mode "insertion". | Donne une réponse booléenne.

Dans le tableau suivant, on trouve des adresses testant s'il y a ou non, à un moment précis, édition d'une ligne de commande, et qui tirent éventuellement de cette observation des conséquences sur le déroulement du programme en cours.

Je rappelle la convention utilisée ici:

- "***q***" signifie quitter le Rpl en cours, et retour prématuré au Rpl qui le contient.
- "***sk/dq***" signifie: si le test précédent a donné "***True***", alors "*sauter*" l'élément suivant du Rpl, sinon évaluer cet élément en même temps qu'on sort de ce Rpl.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#3E607h	NoComLine?:(bip;q) |Bippe et quitte le Rpl en cours si aucune ligne de | commande n'est en cours.
#3E62Ah	ComLine?:(bip;q) |Bippe et quitte le Rpl en cours si une ligne de | commande est en cours.
#40D93h	ComLine?:sk/dq |Précédant l'alternative do;quit (dans le cas | donc où il n'y a pas de ligne de commande), | il y a un appel à l'adresse #44394.
#4488Ah	NoCurComLine? |Teste s'il n'y a pas de ligne de commande en cours. | Réponse booléenne.
#53A4Ah	CurComLine? |(LM) True si une ligne de commande est en cours.
#63DCBh	CurComLine?:dq/sk |S'il y a une ligne de commande en cours, éxécute | l'objet "obj" qui suit dans le Rpl en cours, | puis quitte ce Rpl; sinon poursuit après "obj" | l'éxécution de ce Rpl.

Nous entrons maintenant dans le "*noyau*" de l'environnement développé par la HP48 pour sa ligne de commande, avec toutes les opérations de déplacements.

La ligne de commande est une ligne "*logique*", qui peut être composée de plusieurs lignes "*physiques*". Dans les mnémoniques ci-dessous, j'ai employé le mot anglais "*Row*" pour désigner les différentes lignes dont peut être composée la ligne de commande, de manière à éviter toute ambiguité sur l'emploi du mot "*ligne*".

Adresse	Mnémoniques et commentaires (LM si langage machine)
#13E85h	CurrentCol |(LM) Donne le N° de colonne du curseur dans la ligne de commande en cours, sous la forme d'un entier-système (<0h>=début)
#13EF1h	PosComLine |(LM) Donne la position dans la ligne de commande en cours, sous forme d'un entier-système (<0h>=début)
#13F47h	CurrentRow |(LM) Donne le N° de ligne dans la ligne de commande en cours, sous forme d'entier-système (<1h>=début)
#424DAh	ToNextRow |Passe à la ligne suivante, sur la même colonne
#42525h	ToTheLastRow |Jusqu'à la dernière ligne, sans changer de colonne
#4256Bh	<-OnTheRow |On se déplace d'un caractère vers la gauche, sur la ligne en cours (si en est en début de ligne, on ne passe pas à la ligne précédente).
#425B6h	ToTheStartOfRow |Passe au début de la ligne de cours.
#425D4h	->OnTheRow |On se déplace d'un caractère vers la droite, sur la ligne en cours (si en est en fin de ligne, on ne passe pas à la ligne suivante).
#425EDh	ToTheEndOfRow |On se déplace jusqu'à la fin de la ligne en cours
#42660h	ToPrevRow |Passe à la ligne précédente, sans changer le numéro de colonne.
#4269Ch	ToTheFirstRow |On se déplace sur la première ligne, en conservant le numéro de colonne.
#426F1h	OnTheLastRow? |Est-on sur la dernière ligne?
#4270Ah	OnTheFirstCol? |Est-On sur la première colonne?
#4272Dh	OnTheFirstRow? |Est-on sur la première ligne?
#42764h	BegCmdLine?:(bip;q) |Bippe et quitte le Rpl en cours si on est au début de la ligne de commande en cours.
#42787h	EndCmdLine?:(biq;q) |Bippe et quitte le Rpl en cours si on est à la fin de la ligne de commande en cours.

La suite du tableau précédent....

Adresse	Mnémoniques et commentaires
#428A9h	PrevLineIfStart |Passe à la fin de la ligne L-1 si on est au début de la ligne L (à réserver à cette situation)
#428C2h	ToNextRowIfEnd |Passe à la ligne L+1 si on est à la fin de la ligne N°L (à réserver à cette situtation).
#4325Ah	SetCursPos |Place le curseur en position définie par l'objet situé au niveau 1, et qui peut être une liste de 2 entiers-systèmes, ou un seul entier-système (Voir ci-dessous)
#43273h	SetCursPos(li) |Comme ci-dessus, où "li" est une liste { L C } de 2 entiers-système: L=n°de ligne, C=n° de colonne. L=<0h>=>dernière ligne, C=<0h>=>dernière colonne.
#43309h	SetCursPos(s) |Comme ci-dessus, la position étant définie par un entier système s (s'il est nul, on va à la fin de la ligne de commande en cours)
#44487h	EndOfComLine? |Est-on est à la fin de ligne de commande ?
#444A5h	EndOfRow? |Teste si on est à la fin d'une ligne dans la ligne de commande, où si on est à la fin de celui-ci.
#44711h	SizeOfComLine |Taille (entier-système), en caractères, de la ligne de commande en cours.
#44730h	RowsOfComLine |Nombre de lignes (entier-système) de la ligne de commande en cours.

Voici quatre adresses permettant de prendre connaissance d'un caractère de la ligne de commande (notamment celui qui est situé à la position du curseur)

Adresse	Mnémoniques et commentaires
#443DFh	NextChrInComLine |Donne le "Character" qui suit la position actuelle dans la ligne de commande.
#443F3h	PrevChrInComLine |Donne le "Character" qui précède la position actuelle dans la ligne de commande.
#44407h	ChrInComLine(s) |Donne le "Character" qui se trouve à la position s-1 par rapport à la position actuelle dans la ligne de commande en cours.
#4446Eh	NextChrInCL=Space? |Répond True si le caractère qui suit la position actuelle dans la ligne de commande est un espace.

Les Sysevals qui suivent éxécutent une des instructions **->SKIP**, **<-SKIP**, **DEL->** ou **<-DEL**, (ou quelque chose qui y ressemble) à partir de la position courante dans la ligne de commande.

Adresse	Mnémoniques et commentaires
#3E425h	<-Del! |Force l'instruction <-DEL
#3E49Dh	DelLeftChr |C'est l'action de la touche BasckSpace
#3E64Dh	Chk<-Skip
	|au niveau 2: True si on n'est pas au tout début.
	|au niveau 1: True si le caractère précédant est un
	espace ou le caractère de saut de ligne chr(10).
#3E684h	<-Skip! |Force l'instruction <-SKIP
#3E6A2h	ChkSkip->
	|au niveau 2: True si on n'est pas tout à la fin.
	|au niveau 1: True si le caractère suivant est un
	espace ou le caractère de saut de ligne chr(10).
#3E6CFh	Skip->! |Force l'instruction SKIP->
#429CBh	DelCurChr |C'est l'action de la touche DEL

Les SYSEVALs qui suivent sont autant de façons de rajouter quelque chose (une chaîne, un caractère, ou même un objet quelconqe) à la ligne de commande en cours (en la créant si elle n'existe pas).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#3EDC5h	EvalNextChkPRG
	|L'objet "obj", qui suit dans le RPl, est évalué si on
	n'est pas en mode PRG.
	|Sinon il est placé en ligne de commande.
#3EDF2h	Eval2NextChkPRG
	|Soient obj1,obj2 les 2 objets suivants dans le Rpl.
	|Si on est en mode PRG, obj1 est placé en ligne de
	commande, sinon, obj2 est évalué.
#3EE47h	2NextInCmdLine
	|Envoie les deux objets (qui suivent dans le Rpl)
	en ligne de commande, sur 2 lignes distinctes.
	|Le deuxième objet est d'abord envoyé, puis le
	premier. Curseur en fin de première ligne créée
#3EE65h	3NextInCmdLine |Comme "2NextInCmdLine", avec 3 objets.
#3EE92h	4NextInCmdLine |Comme "2NextInCmdLine", avec 4 objets.
#404A9h	PutInComLine(st)!
	|Place la chaîne st en ligne de commande, sans les ""
	En cas d'interruption par ON, l'objet est perdu.
#40580h	PutInComLine(_)!
	|Place l'objet en ligne de commande. Les fonctions
	|intégrées se voient ajouter une paire de parenthèses
#40625h	PutInComLIne(_)!!
	|Comme ci-dessus, sauf pour les fonctions intégrées.
#443CBh	PutInStartComLine(st)
	|Place la chaîne en tête de la ligne de commande en
	cours (créée au besoin). Curseur placé au début
#444C3h	AddInComLine(st) |Ajoute "st" en ligne de commande
#444EEh	AddInComLine(ch) |Comme ci-dessus avec un "Character"

Un dernier tableau pour traiter des opérations d'édition (touche **EDIT**), de saisie sur demande (instruction **INPUT**), et d'évaluation de la ligne de commande.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#14137h	obj->(st) |Evalue une chaîne. Appelle d'abord "SyntaxOK(st)?" | puis évalue la chaîne.
#238A4h	SyntaxOK(st)? |Vérifie la syntaxe d'une chaîne de caractères, | interprétée comme une ligne de commande. |Réponse: 2:st 1:true (si pas d'erreur de syntaxe) |En cas d'erreur détectée, la réponse est: | 4:st 3:s 2:st' 1:False où "st'" est la sous-chaîne | de st où la première erreur a été détectée, et où | "s" indique la position suivant st' dans st
#3B095h	EditPrg("") |Débute l'édition d'une chaîne vide, en mode PRG
#40BB5h	EnterComLine |Evalue ENTER sur la ligne de commande (en tenant | compte du mode "Enter redirigé" si nécessaire)
#42D32h	Edit(_) |Forme interne de EDIT. Le fond de l'écran est effacé. | L'objet est restitué en cas d'appui sur ON. | Le menu EDIT est effacé.
#42D46h	BestEdit(_) |Edition dans le meilleur environnement
#42DFFh	EditLevel(r) |Edition de l'objet situé au niveau r
#42E13h	BestEditLevel(r) |Edition de l'objet du niveau r, dans le meilleur | environnement (Matrix-Writer ou Equation-Writer)
#42E5Eh	Visit(nm) |Visite le contenu de 'nm'
#42E72h	BestVisit(nm) |Visite le contenu de 'nm', dans le meilleur | environnement possible (Matrix ou Equation-Writer)
#42F44h	input! |C'est la forme la plus interne de INPUT |Voir exemple page suivante
#43395h	input(2st) |Forme interne de INPUT
#433CCh	input(st,1i) |Forme interne de INPUT
#44277h	AbortCurComLine |Abandonne ligne de commande en cours
#44683h	->StrComLine |Transforme en chaîne la ligne de commande en cours, | sans la supprimer pour autant.
#4A770h	InputName(2st)? |Saisie d'un nom par l'instruction INPUT. | st2 est la chaîne affichée en zone pile. | st1 est la chaîne affichée en ligne de commande. |Le curseur est placé après st2, en mode α. |L'objet saisi doit être un nom global. |Le résultat est false si la saisie interrompue, | et 2:nm 1:true sinon (Il y a une erreur | "Bad Argument Type" si ça n'est pas un nom).
#53A58h	BeginComLine |(LM) Débute une ligne de commande.

◆ Utilisation de "input!" (#42F44h):

Toutes les demandes de saisie au clavier (instruction **INPUT**, requêtes dans les environnements **ΣDAT**, **SOLVE**, etc ...) utilisent la forme suivante, très interne, de l'instruction **INPUT** (située à l'adresse **#42F44h**, avec le mnémonique ***input!***: voir le tableau précédent).

La pile doit contenir les éléments suivants:

- 10: La ***chaîne à afficher*** (à titre de message).
- 9: La chaîne représentant la ***ligne de commande initiale***.
- 8: La ***position initiale du curseur***, sous la forme d'un entier-système ou d'une liste de 2 entiers-système.
- 7: Un entier-système s indiquant le ***mode du curseur***:
  s=<0h> => Mode *courant*.
  s=<1h> => Mode *insertion*.
  s=<2h> => Mode *recouvrement*.
- 6: Un entier-système s indiquant le ***mode de saisie***:
  s=<0h> => Mode *courant* + Mode **PRG**.
  s=<1h> => Mode **PRG**.
  s=<2h> => Mode **ALG PRG**.
- 5: Un entier-système s indiquant le ***mode alphabétique***:
  s=<0h> => Mode *courant*.
  s=<1h> => Mode α *verrouillé*.
  s=<2h> => Mode α *désactivé*.
- 4: La liste donnant le ***menu à afficher*** (en principe le menu **EDIT**).
- 3: Le ***numéro de page*** de ce menu (en principe **<1h>**).
- 2: Un booléen indiquant si l'appui sur **ON** se traduit par une sortie de l'édition (True) ou simplement un effacement de l'objet en cours d'édition (False).
- 1: Un entier-système indiquant comment doit être retourné le résultat:
  s=<0h> => Sous forme de ***chaîne***.
  s=<1h> => Sous forme de ***chaîne*** et d'***objet évalué***.
  s=<2h> => Sous forme d'***objet évalué***.

On peut ainsi personnaliser les demandes de saisie au clavier. Le plus grand intérêt que je vois à cette adresse est qu'elle débouche sur de nombreuses autres (certains SYSEVALs de ce chapitre ont d'ailleurs été obtenus en décryptant cette adresse et ses conséquences).

LES FLAGS DE LA HP48

Ce chapitre regroupe des SYSEVALs se rapportant aux nombreux *indicateurs* qui déterminent le comportement de la calculatrice.

Certains de ces indicateurs sont matérialisés par un affichage à l'écran (en ligne d'état, par exemple, l'indicateur **RAD**).

D'autres modifient certains aspects du fonctionnement de la HP48:

- Mémorisation ou non de la dernière pile, des derniers arguments.
- Mode de calcul des angles, des entiers binaires, etc...
- Gestion du mode **USER**.
- Gestion des flags-système ou des flags-utilisateur (lecture et/ou modification).

etc...

Il n'y a pas vraiment d'unité entre ces différents problèmes, et ce chapitre reflète la diversité des situations rencontrées.

Mais puisqu'il faut bien commencer par un bout, voici les SYSEVALs qui traitent de la mémorisation éventuelle d'une pile-utilisateur (fonction **LAST-STACK** du clavier).

On remarquera, que contrairement à **LASTARG**, la fonction **LAST-STACK** n'est pas programmable. Aucun indicateur ne permet de la "*débrayer*", et la seule possibilité resterait d'utiliser l'entrée **STK** du menu **MODES** pour annuler ou rétablir cette fonction, s'il n'y avait les SYSEVALs que voici:

Adresse	Commentaires (LM si langage machine)
#17D46h	DelLastStack |Efface la pile mémorisée, et toutes les variables | locales existantes (pas uniquement les dernières | à avoir été créées).
#1825Fh	NoLastStack? |(LM). Teste s'il n'y a ni pile mémorisée ni variable | locale en cours. La réponse est un booléen.
#18355h	DelStack |(LM). Efface la pile en cours (comme CLEAR), | et provoque un "Garbage Collection".
#538C0h	LastStack? |(LM). Teste si LAST STACK est active. |La réponse est le booléen True ou False.
#538CEh	UnableLastStack |(LM). Active LAST STACK
#538DCh	DisableLastStack |(LM). Désactive LAST STACK
#61D41h	SaveStack |(LM). Sauve la pile en cours.
#61F8Fh	RclSavedStack |(LM). Rappelle la pile sauvée par "SaveStack", en | en lieu et place de la pile actuelle.

Suite logique du tableau précédent, voici les adresses ayant un rapport avec la fonction **LASTARG** (conservation des derniers arguments).

Adresse	Commentaires (LM si langage machine)
#112ECh	DelLastArg |(LM). Efface les derniers arguments, même si le mode | LastArg est actif (c-à-d si flag N°-55 = 0)
#18282h	NoLastArg? |(LM). Répond True si aucun argument n'est mémorisé.
#1A631h	LastArg |(LM). Forme interne de LASTARG. Ne teste le flag N°-55
#53842h	ChkDelLastArg |Efface derniers arguments, uniquement si le mode | LASTARG est inactif (flag N°-55 = 1) |Est appelé après toute modification des flags | pouvant affecter le flag -55.

Voici les SYSEVALs qui se rapportent aux entiers binaires, et qui sont les formes internes des instructions **HEX**, **BIN**, **OCT**, **DEC** (choix du mode de numération) ou **STWS**, **RCWS** (choix du nombre de bits utiles et affichés, et rappel de ce nombre).

Adresse	Commentaires (LM si langage machine)
#53C37h	hex |(LM) Formes
#53C43h	bin |(LM) internes
#53C4Fh	oct |(LM) des instructions
#53C5Bh	dec |(LM) HEX, BIN, OCT, DEC.
#1C5DEh	stws(b) |Forme interne de STWS, avec argument binaire
#53C96h	stws(r) |Forme interne de STWS, avec argument réel
#53CAAh	stws(s) |(LM) Forme interne de STWS, s=entier-système
#53CF0h	rcws |Forme interne de RCWS
#54039h	Srcws |(LM). Donne la taille des entiers binaires, exprimée | sous la forme d'un entier-système.
#54050h	RclBinBase |Donne un entier-système égal à la base de numération | en cours pour les binaires (<2h>,<8h>,<Ah>,ou<10h>)
#5407Ah	RclCharBinBase |Donne le "Character" h,d,o, ou b, suivant le mode de | numération en cours.

Dans le tableau suivant, on voit comment allumer ou éteindre les différents indicateurs "*shift*", à savoir:
α: "*Alpha-Shit*", ┌-> "*Right-Shift*", et <-┐ "*Left-Shift*".

Ce tableau est divisé en deux parties. Dans la première, on allume ou on éteint un des trois indicateurs, sans que celà influe sur la prochaine touche actionnée.

Dans la seconde partie, au contraire, la prochaine touche qui sera actionnée verra son action conditionnée par les indicateurs "*shift*" affichés.

Adresse	Commentaires (LM si langage machine)
#11320h	ClearBusy |(LM). Efface l'indicateur de "programme en cours". | Celui-ci n'en continue pas moins de tourner!
#1132Dh	LightαShift |(LM) Allume l'indicateur α
#1133Ah	ClearαShift |(LM) Eteint l'indicateur α
#11347h	LightRShift |(LM) Allume l'indicateur ┌─>
#11354h	ClearRShift |(LM) Eteint l'indicateur ┌─>
#11361h	LightLShift |(LM) Allume l'indicateur <─┐
#1136Eh	ClearLShift |(LM) Eteint l'indicateur <─┐
#11387h	ClearIOindic |(LM) Efface l'indicateur I/O
#3FCAFh	NoShift |Éteint les 3 shift (α, ┌─>, <─┐)
#3FCDCh	LShiftOnly |Allume <─┐, éteint α et ┌─>
#3FD09h	RShiftOnly |Allume ┌─>, éteint α et <─┐
#3FD36h	αShiftOnly |Allume α, éteint <─┐ et ┌─>
#3FD63h	αLShifOnly |Allume α et <─┐, éteint ┌─>
#3FD90h	αRShifOnly |Allume α et ┌─>, éteint <─┐

Voici les adresses qui traitent du mode d'affichage des réels, ou du mode de calcul des angles.

Adresse	Commentaires (LM si langage machine)
#15A54h	RclOldMode |(LM) Rétablit le mode d'affichage des réels tel qu'il | a pu être modifié par "TempStd" (ci-dessous)
#15A97h	TempStd |(LM) Passe en mode STD, en mémorisant le mode actuel, | qui pourra être rétabli par "RclOldMode"
#166E3h	fix(s) |(LM) Formes internes de FIX, SCI, ou ENG.
#166EFh	sci(s) |(LM) Ici s est un entier-système.
#166FBh	eng(s) |(LM)
#16707h	std |(LM) Forme interne de STD.
#1C403h	fix(r) |Formes internes
#1C437h	sci(r) | de FIX, SCI, ou ENG.
#1C46Bh	eng(r) | Ici r est un réel.
#53B61h	RclRealMode |(LM) Donne le mode d'affichage des nombres réels, | <0h>=STD, <1h>=FIX, <2h>=SCI, <3h>=ENG
#5A983h	RclAccuracy |(LM). Donne sous forme d'entier-système, le nombre |de décimales affichées (de <0h> à <9h> pour 0 à 9, |puis <10h> pour 10, et <11h> pour le mode STD)
#2A5D2h	deg |Forme interne de DEG
#2A5F0h	rad |Forme interne de RAD
#2A604h	grad |Forme interne de GRAD
#53BC9h	deg? |True si on est en mode degrés, false sinon.
#53BDDh	rad? |True si on est en mode radians, false sinon.
#53BF1h	rad?;fs?(-18)? |2:False 1:True caractériserait le mode "grades"

Les SYSEVALs suivants regroupent les nombreuses formes internes des instructions **RCLF**, **STOF**, **SF**, **CF**, **FS?**, **FC?**, **FS?C**, **FC?C** (lecture et/ou modifications des *flags-système* ou des *flags-utilisateur*).

Rappelons que les *flags-système* sont habituellement désignés par un entier négatif, alors que les *flags-utilisateur* sont désignés par un entier positif.

Certaines des adresses ci-dessous agissent spécifiquement sur les *flags-système* ou sur les *flags-utilisateur*. Dans ces conditions, un flag pourra être désigné par un entier-système (donc positif) sans ambiguité.

Adresse	Commentaires (LM si langage machine)
#1C28Dh	sf(r) | Arme le flag n°r
#1C2EEh	cf(r) | Désarme le flag n°r
#1C32Ch	fs?(r) | Teste si flag n°r armé. Résultat 1 ou 0
#1C331h	fs?(r)? | Teste si flag n°r armé. Résultat booléen
#1C379h	fc?(r) | Teste si flag n°r désarmé. Résultat 1 ou 0
#1C4BAh	fs?c(r) | Comme fc?(r) puis désarme le flag.
#1C4BFh	fs?c(r)? | Comme fc?(r)? puis désarme le flag.
#1C4C9h	UserFs?c(s)?
	| Teste si le flag-utilisateur n°s est armé (réponse booléenne True ou False), puis le désarme.
#1C4E7h	SystFs?c(s)?
	| Teste si le flag-système n°-s est armé (réponse booléenne True ou False), puis le désarme.
#1C539h	fc?c(r)
	| Teste si flag n°r désarmé (=> 1 ou 0) puis le désarme
#1C637h	RclSystF
	| (LM) Rappelle l'entier binaire des flags-système
#1C64Eh	RclUserF
	| (LM) Rappelle l'entier binaire des flags-utilisateur
#1C6A2h	StoF(li)
	| Forme interne de STOF. L'argument est une liste.
#1C6CFh	StoF(2b)
	| b2 va dans flags-système, b1 dans flags-utilisateur
#1C6E3h	SystStoF(b)
	| L'entier binaire "b" va dans flags-système
#1C6F7h	UserStoF(b)
	| (LM) L'entier binaire "b" va dans flags-utilisateur
#1C731h	SystStoF(b)!
	| (LM) Comme "SystStoF(b)", mais n'effectue pas un test sur les nouveaux flags-système pour savoir s'il faut effacer les arguments sauvegardés (au cas où LASTARG vienne tout juste d'être désactivé).
#3EDA2h	InvSysFlag(s) | Inverse l'état du flag-système n°-s
#53725h	UserSf(s) | (LM) Arme le flag-utilisateur n°s
#53731h	SysSf(s) | (LM) Arme le flag-système n°-s
#53755h	UserCf(s) | (LM) désarme le flag-utilisateur n°s
#53761h	SysCf(s) | (LM) désarme le flag-système n°-s
#53778h	UserFs?(s)?
	| (LM) Teste si le flag-utilisateur n°s est Armé. La réponse est le booléen True ou False.
#53784h	SysFs?(s)?
	| (LM) Teste si le flag-système n°-s est Armé. La réponse est le booléen True ou False.

LES FLAGS DE LA HP48

Le tableau suivant contient quelques adresses, autour d'une même idée: afficher les *indicateurs* de la ligne d'état.

Adresse	Commentaires
#395E2h	DispAllStatus! |Affiche en ligne d'état tous les statuts dans leur état actuel. Appelle pour celà "StatusDisp(st,3s)" après appel de "AngleDispStatus", "PolarDispStatus" "HaltDispStatus", etc... (voir ci-dessous)
#39632h	StatusDisp(st,3s) |Affiche (sans freeze) la chaîne "st" en ligne d'état et en petits caractères. |La signification des entiers-système est la suivante: |s3: n° de colonne où débute l'affichage de "st". |s2: n° de la colonne débutant la zone couverte. |s1: n° de la colonne terminant la zone couverte. |L'affichage se fait sur le GROB courant (celui de la pile, ou bien PICT). |Si la chaîne est trop longue pour tenir dans la zone ainsi définie, cette zone est élargie pour accueillir tout l'affichage de la chaîne
#39673h	AngleDispStatus |Donne de quoi afficher le mode de calcul des angles au sens de #39632h (StatusDisp(st,3s)), selon le mode en cours. |On obtient par ex: 4:"RAD" 3:<4h> 2:<0h> 1:<14h> si on est en mode radians.
#396C8h	PolarDispStatus |Donne de quoi afficher le mode polaire en cours, au sens de #39632h (StatusDisp(st,3s)). |On obtient par ex: 4:"R<Z" 3:<16h> 2:<14h> 1:<25h> si on est en mode "coordonnées polaires".
#3970Eh	HaltDispStatus |Donne de quoi afficher le mode "Halté" ou non, au sens de #39632h (StatusDisp(st,3s)). |Par ex, si un programme est suspendu, on obtient: 4:"HALT" 3:<27h> 2:<25h> 1:<38h>
#39748h	1-5DispStatus |Donne de quoi afficher l'état des indicateurs 1 à 5 au sens de #39632h (StatusDisp(st,3s)).
#397BBh	UsrDispStatus |Donne de quoi afficher le mode 1USR, USER ou non, au sens de #39632h (StatusDisp(st,3s)). |Si on est en mode USER, par exemple, on obtient: 4:"USER" 3:<51h> 2:<4Fh> 1:<62h>
#3981Bh	AlgDispStatus |Donne de quoi afficher le mode ALG ou non, au sens de #39632h (StatusDisp(st,3s)). |Si on est mode ALG, ou ALG PRG, on obtient: 4:"ALG" 3:<64h> 2:<62h> 1:<72h>
#39853h	PrgDispStatus |Donne de quoi afficher le mode PRG ou non, au sens de #39632h (StatusDisp(st,3s)). |Si on est en mode PRG, on obtient: 4:"PRG" 3:<74h> 2:<72h> 1:<83h>

Voici enfin un cocktail de SYSEVALs se rapportant à tel ou tel *indicateur*. Tous n'ont évidemment pas le même intérêt mais ceux qui traitent du mode **USER** peuvent se révèler très utiles.

Adresse	Commentaires (LM si langage machine)
#167BFh	DecSep=.? |Teste si le séparateur décimal est "." (c-à-d si le flag -51 est désarmé). Réponse booléenne.
#27FA3h	NoDecSep |Donne "," ou "." (le contraire du séparateur décimal)
#3988Bh	DispPath?Time |Affiche le chemin, et éventuellement l'heure (si l'heure doit être affichée)
#39971h	2GrobPath(?) |Donne 2 grob et un booléen. |Le grob au niveau 3 est l'image du PATH, sauf le caractère initial { ou ... |Le grob au niveau 2 est l'image du caractère { ou ... qui doit débuter l'image du PATH. |Le booléen initial vaut false si l'heure n'est pas affichée, et true si elle l'est (il peut être obtenu par "TimeToDisp?" ci-dessous) |Le booléen final vaut true si le caractère de début du PATH est ... (chemin trop long).
#39AF1h	TimeToDisp? |Teste s'il faut afficher le jour et l'heure. |La réponse est un booléen (True ou False)
#39F56h	MLmode? |Teste si le mode multi-lignes est actif, c-à-d si le flag n°-52 est désarmé.
#3FFA8h	1USR? |(LM) A utiliser si on est on mode USER. |Teste si on est en mode 1USER. Résultat booléen.
#40A4Ch	UserMode&VectoredENTER? |Teste si on est en mode USER et si le mode "Enter redirigé" est actif, c-à-d si les deux flags n°-63 et -62 sont armés. Réponse booléenne.
#40D4Dh	InvUserModeFlag |Inverse l'état du flag n°-62, qui détermine si le mode USER est actif (flag armé) ou inactif.
#40D61h	SetUserMode |Arme le flag -62 (passe en mode USER)
#40D7Ah	QuitUserMode |Désarme le flag -62 (on n'est plus en mode USER)
#41A8Dh	UserMode? |Teste si le flag -62 est armé, c'est-à-dire si on est en mode USER

La suite du tableau précédent...

Remarquez le très joli SYSEVAL à l'adresse **#4730Ah** (mnémonique "***SetColWidth(s)***") qui permet de visualiser jusqu'à dix colonnes simultanément dans l'environnement **Matrix-Writer** !

Adresse	Commentaires (LM si langage machine)
#472FAh	RclColWidth |(LM) Donne la largeur de colonne dans "Matrix-writer" | (une des valeurs <4h>,<5h><6h>,<Ah>,ou <14h)
#4730Ah	SetColWidth(s) |(LM) Définit la largeur de colonne dans | "Matrix-writer", avec les notations ci-dessus |N.B!! Il est possible de définir des tailles qui ne | sont pas permises à priori. Par exemple, une taille | de 2 caractères permet d'afficher 10 colonnes !!
#4731Ah	RclGoMode |(LM) Teste l'indicateur GO dans "Matrix-writer" |Renvoie <0h> si GO-> , <1h> si GO| ,<2h> sinon.
#4732Ah	SetGoMode(s) |(LM) Définit l'indicateur GO dans "Matrix-Writer", en | utilisant les notations ci-dessus.
#48D0Ch	FastCat? |Teste si la flag -59 est armé, c'est-à-dire si on | est en mode "Fast Catalogue Display".
#49035h	StatPlotMode? |Teste si le mode tracé est un mode "STAT" (c-à-d | s'il est égal à BAR, HISTOGRAM ou SCATTER) |Le résultat est un booléen.
#4DB11h	fs?(-32)? |Teste si le flag n°-32 est armé, c-à-d si le curseur | graphique est affiché en mode XOR
#53B88h	cf(-3) |Désarme le flag des résultats numériques
#53B9Ch	sf(-3) |Arme le flag des résultats numériques
#53BB0h	fc(-3)? |Teste si le flag des résultats numériques est | désarmé. Le résultat est un booléen.
#53C0Ah	fc(-2)? |Teste si le flag n°-2 des constantes symboliques | est désarmé. Le résultat est un booléen.
#53C23h	fs(-1)? |Teste si le flag n°-1 "de la solution principale" | est armé. Le résultat est un booléen.

LE GRAPHISME

Voilà un sujet qui intéresse tous les programmeurs et autres amateurs d'effets spéciaux. Ils ne seront pas déçus. La moisson de SYSEVALs est en effet particulièrement abondante.

Vu son importance, ce chapitre a été divisé en 4 parties:

- 1) SYSEVALs et objets-graphiques (sans référence à l'affichage, que ce soit l'affichage de la pile ou celui de **PICT**)
- 2) L'objet-graphique *courant*.
- 3) L'objet-graphique *de la pile*.
- 4) L'objet-graphique **PICT**.

Quelques explications préliminaires s'imposent, notamment en ce qui concerne les points 2) et 3) ci-dessus.

La HP48 gère en parallèle trois objets graphiques:

- L'objet-graphique **PICT** (c'est celui que nous connaissons bien car il est le centre de l'environnement **GRAPH**, et c'est à lui que s'appliquent nombre des instructions du menu **PRG/DSPL** (les deux premières pages de ce menu, pour être précis).
- L'objet-graphique ***de la pile*** (quel autre nom lui donner?) sur lequel la HP48 affiche le contenu des quatre premiers (au plus) étages de la pile, la date et l'heure, le chemin (**PATH**) en cours, etc...
- L'objet-graphique ***des menus***.

Les dimensions de ce dernier sont fixes (131x8). Quant aux deux premiers (y compris l'objet-graphique de la pile, ce qui en surprendra beaucoup), ils sont de dimensions variables et peuvent être "*scrollés*".

Par défaut, leurs dimensions sont les suivantes:
- Objet-graphique *de la pile*: 131x56
- **PICT**: 131x64.

Pour vous convaincre que l'objet-graphique de la pile peut jouer un rôle tout à fait étonnant, sachez que des applications **HP** comme la carte **HP-SOLVE**, ou bien tout simplement l'environnement **Equation-Writer**, fonctionnent avec cet objet-graphique (de manière à ne pas interférer avec le contenu de **PICT**, qui vous est personnellement réservé).

A tout moment, un seul de ces deux objets-graphiques est affiché. Nous l'appelerons l'*objet-graphique courant* (ou le "*Grob courant*").

La deuxième partie de ce chapitre couvrira donc les opérations qui portent sur le *Grob courant*, qu'il s'agisse de l'objet-graphique de la pile, ou bien de **PICT**.

Les parties 3) et 4) correspondent bien sûr à des opérations portant spécifiquement sur l'un des deux objets-graphiques dont nous venons de parler.

N.B: L'objet-graphique des menus a déjà été traité dans un précédent chapitre. Nous n'y reviendrons donc pas ici.

◆ LES OBJETS-GRAPHIQUES:

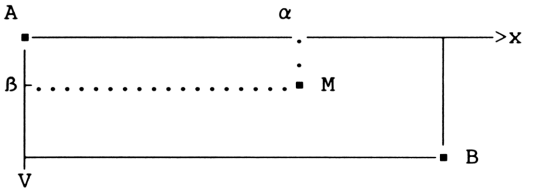
Nous verrons dans cette partie les SYSEVALs qui s'appliquent aux objets-graphiques en général (donc à priori différents de **PICT** et du *Grob de la pile*, encore que rien ne s'y oppose).

Commençons par quelques mises au point, sur certaines notations, qui seront d'ailleurs utiles dans les trois autres parties de ce chapitre.

Les ***dimensions*** d'un objet-graphique sont le plus souvent indiquées par deux entiers-systèmes successifs **<W>** et **<L>**, le premier indiquant la largeur **W** et le second indiquant la hauteur **H** (en pixels).

Un point est repéré dans un objet-graphique par un couple de deux entiers-système **x** et **y**, représentant les coordonnées relatives de ce point par rapport au point supérieur gauche du *Grob* (qui a quant à lui les coordonnées <0h>,<0h>).

Dans cette représentation, **x** est la coordonnée relative à l'axe horizontal, et **y** est celle relative à l'axe vertical.



Si le rectangle ci-contre représente un Grob de dimensions <W> (largeur) et <H> (hauteur), les points A,B,M ont pour coordonnées relatives: A:<0h>,<0h> B:<W-1>,<H-1> M:<α>,<ß>

Il pourra arriver que l'ordre dans lequel apparaissent les coordonnées (d'abord la coordonnée horizontale **x**, puis la coordonnée verticale **y**) soit inversé pour certains SYSEVALs. Celà sera toujours explicitement signalé.

Commençons par quelques adresses permettant de créer des objets-graphiques "*blancs*".

Adresse	Mnémoniques et commentaires (LM si langage machine)
#1158Fh	blankYX(2s) |(LM) Crée un objet-graphique blanc de hauteur y=s2 | et de largeur x=s1. Noter l'interversion de x et y.
#4F6A1h	blank(2b) |Forme interne de BLANK, pour deux binaires.
#11A6Dh	blank(go,4s) |(LM) "blanchit" la zone de Grob5 délimitée par les | points de coordonnées relatives (s4,s3) et | (s2-1,s1-1) (noter ce dernier point!) |N.B: pas de contrôle d'erreur. La zône vidée ne doit | pas déborder de Grob5 !!
#6389Eh	blank(go,4s);drop |Ne se conçoit utilement que quand Grob5 est le Grob | de la pile, ou bien PICT.

L'instruction →**GROB** admet de nombreuses formes internes. Les voici:

Adresse	Mnémoniques et commentaires (LM si langage machine)
#11CCEh	3->grob(s) |(LM) Effectue 3->GROB sur le caractère de code s. |On obtient un Graphic 6x10.
#11CF3h	3->grob(st) |(LM) Effectue 3->GROB sur une chaîne.
#11D00h	2->grob(st) |(LM) Effectue 2->GROB sur une chaîne.
#11F80h	1->grob(st) |(LM) Effectue 1->GROB sur une chaîne.
#4EC58h	1->grob(_) |L'objet est d'abord transformé en chaîne. |Les chaînes conservent leurs délimiteurs ".
#5048Dh	->grob(_,r) |Forme interne de ->GROB, où r est un réel devant | valoir 0,1,2 ou 3 (sinon "Bad Argument Value") |Si l'objet à transformer est une chaîne, elle | perd ses délimiteurs ".
#5050Ah	0->grob(_) |Forme interne de ->GROB. Equivaut à 3 ->GROB, sauf | pour les expressions et les objets-unités, qui | sont transformés comme dans Equation-Writer.
#69920h	0->grob(ex) |Comme dans Equation-Writer
#69920h	0->grob(uo) |Comme dans Equation-Writer (même adresse)

Voici maintenant différentes façons de modifier la taille d'un objet graphique, ou de prendre connaissance de cette taille.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#12BB7h	IncrW(go,s) |(LM). Augmente la largeur de Grob2 de s colonnes | blanches à droite. La transformation se fait | directement en mémoire (attention!). |On peut le faire sur le Grob courant, en le rappelant | avec "RClCurGrob", en #12665h. |A éviter si Grob2 n'est ni égal au Grob de la pile, | ni égal à PICT.
#12DD1h	IncrH(go,s) |(LM) Augmente la hauteur de Grob2 de s lignes | blanches en bas. Mêmes remarques que ci-dessus.
#50578h	YXsize(go) |(LM) Donne la hauteur y puis la largeur x du Grob, | sous la forme de deux entiers-système. |Attention à l'interversion de x et de y, par rapport | à l'instruction SIZE (qui donne deux binaires)
#1CA62h	size(go) |Forme interne de SIZE. Le résultat est 2:bw 1:bh, où | bw (largeur) et bh (hauteur) sont deux binaires.
#5179Eh	dup;YXsize(go) |1:Grob => 3:Grob, 2:<Y>=hauteur, 1:<X>=largeur
#63C04h	Width(go) |Donne la largeur du Grob (un entier-système)

Voici maintenant les formes internes de l'instruction **SUB**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#1192Fh	sub(go,4s) |(LM). Extrait de Grob4 l'objet graphique délimité | par les points de coordonnées relatives (s4,s3) et | (s2-1,s1-1) dans Grob4 (notez ce dernier point!)
#4E712h	sub(go,0,0,83h,8h) |Extrait de Grob1 un rectangle de 8 lignes et de 131 | colonnes, à partir du coin supérieur gauche.
#4FB74h	sub(go,2li) |Forme interne de SUB
#4FB74h	sub(go,c,li) |Forme interne de SUB (même adresse)
#4FB92h	sub(go,4s) |Extrait de Grob5 la fenêtre définie par les 2 points | opposés de coordonnées (s4,s3) et (s2,s1)
#4FBC4h	sub(go,2c) |Forme interne de SUB

Voici les formes internes des instructions **REPL** et **INV**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#11679h	repl(2go,2s) |(LM) Applique Grob4 sur Grob3, à partir du point dont les coordonnées relatives dans Grob3 sont x=s2, et y=s1. Le résultat va dans Grob3, sans NewOb. |Les 4 arguments disparaissent de la pile. |N.B: Grob4 ne doit pas déborder de Grob3 !! |Pour placer Grob4 sur le Grob courant, il faut placer celui-ci au niveau 3 ("RclCurGrob" en #12635h). |En partant de go3,go4,go3,s2,s1, (même adresse aux niveaux 5 et 3), le nouveau "go3" reste sur pile.
#12815h	repl(go,2s,st) |Place la chaîne (après un 1->GROB) sur le Grob en position (s3,s2). |Le nouveau Grob reste, seul, au niveau 1.
#12829h	repl(go,2s,go) |Place Grob1 sur Grob4, en position (s3,s2). |Le nouveau Grob reste, seul, sur la pile
#4F7E6h	ChkBeforeRepl(2go,2s) |Ici Tout se passe comme si Grob3 allait être placé sur Grob4 au point de coordonnées relatives (s2,s1) (mesurées sur Grob4). |La routine tronque éventuellement les dimensions de de Grob3, pour qu'il ne dépasse pas de Grob4. |On retrouve Grob4,Grob3,s2,s1 sur la pile (Grob3 ayant donc éventuellement été réduit).
#4F999h	repl(go,li,go) |Forme interne de l'instruction REPL. Il y a un NewOb sur Grob3 si il est référencé.
#4F9B2h	repl(go,li,go)! |Comme ci-dessus, mais sans Newob
#4F9CBh	ChkRepl(2go,2s) |Replace Grob3 sur Grob4, en position (s2,s1). (Teste la validité de (s2,s1) et des dimensions) Grob3 est éventuellement réduit au départ pour ne pas dépasser de Grob4.
#4F9F3h	repl(go,c,go) |Forme interne de l'instruction REPL. Il y a un NewOb sur Grob3 si il est référencé.
#4FA0Ch	repl(go,c,go)! |Comme ci-dessus, mais sans Newob.
#122FFh	inv(go) |(LM) Inversion vidéo du Grob, sans NewOb.
#4FC28h	inv(go) |Inversion vidéo du Grob, précédée d'un NewOb si le Grob est référencé.

Formes internes de **GOR**, **GXOR**, et **INV**:

Adresse	Mnémoniques et commentaires (LM si langage machine)
#1E46Ah	gor(go,li,go) | Forme interne de GOR
#1E488h	gor(go,c,go) | Forme interne de GOR
#1E4F8h	gxor(go,li,go) | Forme interne de GXOR
#1E516h	gxor(go,c,go) | Forme interne de GXOR
#4F6BAh	g?or(go,li,go,?) | Si le booléen vaut True, alors il s'agit d'une opération GOR; sinon c'est GXOR
#4F6F6h	g?or(go,c,go,?) | Comme ci-dessus, avec un nombre complexe "c"
#4F78Ch	g?or(?,2go,2s) | Effectue GOR (si ?5=true) ou GXOR (si ?5=false), de Grob3 sur Grob4, à partir du point de coordonnées (s2,s1) mesurées dans Grob4. | Il y a une erreur "Bad Argument Value" si le point (s2,s1) n'appartient pas à Grob4.
#4F8D1h	+(2go) | Forme interne de l'instruction + pour deux Grobs qui doivent avoir les mêmes dimensions (sinon il y a une erreur "Bad Argument Value"). | On obtient leur réunion logique (GOR)

Ce tableau contient quatre adresses utiles quand on veut tester la validité du choix d'un point (ou de deux points quand on veut désigner une "*boîte*"), dans un objet-graphique.

Adresse	Mnémoniques et commentaires
#51564h	InGrob(2s,go)?#:BAV! | Si la réponse à "InGo(2s,go)?" (ci-dessous) est False alors erreur "Bad Argument Value".
#51578h	InGrob(2s,go)? | Teste si le point de coordonnées relatives (s3,s2) dans Grob1 appartient bien à Grob1. | Réponse booléenne. Seuls s3,s2 restent sur la pile.
#51893h	ChkCorners(4s) | Soient A et B les points de coordonnées (s4,s3) et (s2,s1), délimitant une zone rectangulaire Z | Cette routine les transforme en les points A',B' représentant respectivement le coin sup. gauche et le coin inférieur droit de Z. | Ex: <5h>,<1h>,<2h>,<4h> => <2h>,<1h>,<5h>,<4h>
#518CAh	ChkCorners(4r) | Comme ci-dessus, avec des coordonnées réelles.

◆ LE "GROB COURANT".

Cette partie est donc consacrée aux opérations qui portent sur l'objet-graphique actuellement affiché, qu'il s'agisse de **PICT** ou du ***Grob de la pile***.

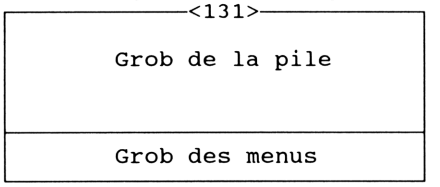
Un grand nombre d'opérations sont en effet communes à ces deux environnements. C'est le cas notamment pour les opérations de *scrolling*.

Si les dimensions de **PICT** et celles du *Grob de la pile* sont variables (comme on va d'ailleurs s'en rendre compte avec les SYSEVALs qui suivent), celles de l'écran sont fixes (!), égales à 131x64.

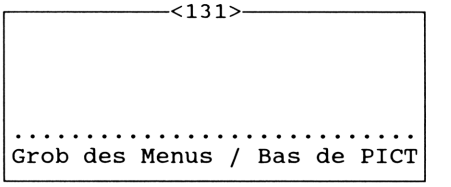
La situation par défaut est la suivante, sachant que les dimensions du *Grob des menus* sont 131x8.

- Les dimensions du *Grob de l'écran* sont 131x56.
- Les dimensions de **PICT** sont 131x64 (comme l'écran).

A l'affichage, les choses se passent alors de la manière suivante:



(Affichage de la pile)

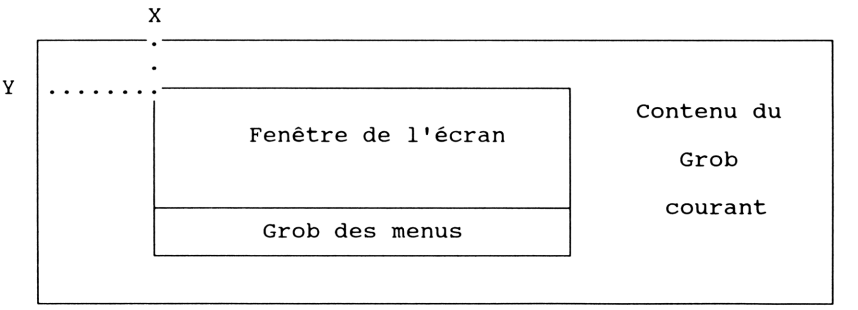


(Affichage de PICT)

Comme on le voit, il n'y a alors pas de superposition quand le *Grob de la pile* est affiché. Quand **PICT** est actif, le contenu du Grob des menus est affiché après celui de **PICT** (mais 8 lignes de **PICT** restent accessibles derrière, comme on le constate dans l'environnement **GRAPH**).

Tout est très différent dès lors que l'on agrandit les dimensions du *Grob courant*. L'écran n'occupe plus alors qu'une *fenêtre* sur cet objet graphique, fenêtre caractérisée par les coordonnées **X,Y** (calculées relativement au *Grob courant*) de son point supérieur gauche.

Voilà comment la situation peut alors se présenter:



Là encore, la position du *Grob des menus* ne doit pas laisser penser qu'il est partie intégrante du *Grob courant*: il est simplement affiché en superposition, 56 lignes en dessous du point supérieur gauche de la fenêtre de l'écran (mais on a vu dans le chapitre consacré aux menus que même cette situation pouvait être modifiée...).

Quand les dimensions du *Grob courant* sont supérieures aux dimensions 131x64 de l'écran, il est possible de ***scroller*** la fenêtre de l'écran sur le *Grob courant* (des SYSEVALs existent pour celà).

On trouvera dans ce chapitre un certain nombre de SYSEVALs effectuant des affichages sur le *Grob courant*.

Certains d'entre eux ne tiennent pas compte d'un *scrolling* éventuel (et affichent par exemple en haut à gauche du *Grob courant*, l'affichage ainsi réalisé étant peut-être invisible à l'écran).

D'autres au contraire (moins nombreux) tiennent compte d'un *scrolling* éventuel, et effectuent l'affichage en haut à gauche de la fenêtre qu'occupe l'écran sur le *Grob courant*. Tout celà apparaîtra bien entendu dans les commentaires accompagnant les SYSEVALs.

Remarque:

Si vous placez dans le *Grob de la pile* un objet graphique de dimensions supérieures à 131x64, et que vous le *scrollez*, vous risquez d'avoir quelques problèmes de visibilité (les niveaux de la pile peuvent avoir disparu en partie, etc...)

Sachez qu'un appui sur **ON-C** replace dans le *Grob de la pile* et dans **PICT** deux objets-graphiques de tailles "*normales*".

Il existe d'autre part un SYSEVAL (**#130CAh**: Mnémonique "***StdCurGrob***") pour ramener le *Grob courant* à ses dimensions standards, sans toucher au contenu de **PICT** (à condition bien sûr que vous éxécutiez ce SYSEVAL en dehors de l'affichage de **PICT**...)

Pour commencer, voici quelques SYSEVALs "*effaceurs*". Les trois derniers tiennent compte d'un *scrolling* éventuel. Les deux premiers supposent implicitement que le *Grob courant* est de largeur 131 et que la fenêtre de l'écran est placée à la première ligne de ce *Grob* (d'une manière précise, "***ClrScr***" efface les 131x55 premiers pixels du ***Grob courant***...).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#01F6Dh	ClrScr
	|(LM) Efface les lignes 0 à 55 du Grob courant,
	| à supposer qu'il soit de largeur standard 131.
#01FA7h	ClrScr&Menu
	|(LM) Comme ci-dessus, et de plus efface le Grob des
	| menus.
#0E06Fh	ClrScr0-15 |Efface les lignes 0 à 15 de l'écran.
#0E083h	ClrScr0-7 |Efface les lignes 0 à 7 de l'écran.
#0E097h	ClrScr8-15 |Efface les lignes 8 à 15 de l'écran.

On continue à vouloir effacer quelque chose dans le *Grob courant*. Les SYSEVALs de ce tableau effacent une ou plusieurs lignes de 131 pixels de large, cadrées à gauche dans le *Grob courant* (donc sans tenir compte d'un *scrolling* éventuel).

Adresse	Mnémoniques et commentaires
#126DFh	ClrRows(2s) |Efface ligne s1 lignes, à partir de la ligne N°s2, | dans le Grob courant, sur une largeur de <83h>=131 | pixels, à partir du coin sup. gauche de ce Grob.
#39958h	ClrRows6-13 |Efface les lignes 6 à 13 du Grob courant, en partant | du bord gauche, et sur une largeur de 131 pixels. |Exécute en fait <6h> <8h> ClrRows(2s)
#3A546h	ClrRows0-15 |Efface les lignes 0 à 15 du Grob courant, en partant | du bord gauche, et sur une largeur de 131 pixels. |Exécute en fait <0h> <10h> ClrRows(2s)
#3A55Fh	ClrRows16-55 |Efface les lignes 16 à 55 du Grob courant, en partant | du bord gauche, et sur une largeur de 131 pixels. |Exécute en fait <10h> <28h> ClrRows(2s)
#3A578h	ClrRows0-55 |Efface les lignes 0 à 55 du Grob courant, en partant | du bord gauche, et sur une largeur de 131 pixels. |Exécute en fait <0h> <38h> ClrRows(2s)

Voici quelques adresses qui rappellent, outre le *Grob courant*, les points extrêmes d'une zone rectangulaire de la fenêtre de l'écran (ou les coordonnées du point supérieur gauche de l'écran sur le *Grob courant*).

Adresse	Mnémoniques et commentaires
#0E0ABh	RclCurGrob0-15 |Donne le Grob courant puis des coordonnées relatives | des points limitant la zone rectangulaire formée | par les lignes 0 à 15 de la fenêtre courante. |=> 5:CurGrob 4:x 3:y 2:x+131 1:y+16 |x et y sont des entiers-système, donnant les coordon- | -nées relatives du point sup. gauche de l'écran. |Il suffit alors d'appeler "sub(go,4s)" en #1192Fh | pour obtenir le Grob 131x16 des lignes 0 à 15. |Ex: 5:Graphic 358x128 4:<71h> 3:<20h> 2:<F4h> 1:<30h>
#0E0D3h	RclCurGrob0-7 |Comme ci-dessus, pour les lignes 0 à 7 de l'écran. |=> 5:CurGrob 4:x 3:y 2:x+131 1:y+8
#0E0FBh	RclCurGrob8-15 |Comme ci-dessus, pour les lignes 8 à 15 de l'écran. |=> 5:CurGrob 4:x 3:y+8 2:x+131 1:y+16.
#0E128h	RclCurGrobXY |Donne le Grob courant, puis les coordonnées relatives | du point supérieur gauche de l'écran. |Ex: 3:Graphic 358x128 2:<71h> 1:<20h>

Le tableau suivant contient des SYSEVALs essentiels pour initialiser, modifier, ou examiner le *Grob courant*:

Adresse	Mnémoniques et commentaires (LM si langage machine)
#0E05Bh	ToCorner(go) |Applique Grob1 sur l'objet-graphique courant à partir de la position qu'y occupe la fenêtre de l'écran. |En pratique, celà revient donc à afficher Grob1 dans le coin supérieur gauche de l'écran.
#12635h	RclCurGrob |(LM) Rappelle le Grob courant (sans newob)
#128B0h	GrobDisp(2s,go) |Applique Grob1 sur le Grob courant, à partir du point de coordonnées relatives (s3,s2). |Si le Grob courant n'est pas suffisamment grand pour contenir Grob1, il est agrandi en conséquence.
#12964h	IncrHCurGrob(s) |Ajoute s lignes blanches en bas du Grob courant.
#1297Dh	IncrWCurGrob(s) |idem avec s colonnes blanches à droite
#12B58h	RclWCurGrob |Donne la largeur (un entier-système) du Grob courant
#12B6Ch	RclHCurGrob |Donne la hauteur (un entier-système) du Grob courant
#12E89h	StoCurGrob(go) |(LM) Fait de Grob1 l'objet-graphique courant.
#12F0Ah	StoNoCur(go) |(LM) Stocke Grob1 dans l'objet-graphique (PICT ou Stack) qui n'est pas l'objet-graphique courant.
#130CAh	StdCurGrob |(LM) Ramène le Grob courant à ses dimensions standards, c'est-à-dire 131x56 si c'est le Grob de la pile et 131x64 si c'est PICT.
#13167h	CurGrob=Pict? |(LM) Teste si le Grob courant est le contenu de PICT (réponse True) ou si c'est le Grob de la pile.
#134AEh	ClrCurGrob |(LM) Efface le Grob courant en gardant ses dimensions
#137B6h	RclCornerYX |(LM) Donne 2:y 1:x, où x,y sont deux entiers-système donnant les coordonnées du point supérieur gauche de l'écran, relativement au Grob en cours. |Noter l'interversion de l'ordre entre x et y.
#4578Eh	CurGrobCheksum |Donne le Checksum du Grob courant.
#515A0h	RclRowY |Donne un entier-système égal au N° de la ligne où apparaît l'image de l'écran sur le Grob en cours
#515B4h	Rcl(RowY+RowMenu) |Additionne le résultat précédent au numéro "RowMenu" de la ligne où apparaît le Grob des menus.
#515FAh	RclColX |Donne un entier-système égal au N° de la colonne où apparaît l'image de l'écran sur le Grob en cours
#5160Eh	RclLastColX |Donne un entier-système égal au N° de la dernière colonne de l'image de l'écran sur le Grob courant. |Exécute en fait "RclColX" puis ajoute <82h>=130.

Les adresses contenues dans ce tableau sont celles qui permettent de *scroller* la fenêtre de l'écran sur le *Grob courant* (à condition bien sûr que celui-ci soit de dimensions supérieures à 131x64), ou de placer directement l'écran en une position précise du *Grob courant*.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#131C8h	^CurGrob! |(LM). Scrolling rapide de 1 pixel vers le haut. |Appelé par la routine "^CurGrob", mais n'utilise | pas de répétition
#13220h	vCurGrob! |(LM). Scrolling rapide de 1 pixel vers le bas. |Appelé par la routine "vCurGrob", mais n'utilise | pas de répétition
#134E4h	<CurGrob! |(LM). Scrolling rapide de 1 pixel vers la gauche. |Appelé par la routine "<CurGrob", mais n'utilise | pas de répétition
#1357Fh	>CurGrob! |(LM). Scrolling rapide de 1 pixel vers la droite. |Appelé par la routine ">CurGrob", mais n'utilise | pas de répétition
#13679h	CurGrobViewYX(2s) |(LM) Affiche le Grob courant en plaçant le point | s2=ligne,s1=colonne comme point supérieur gauche | à l'écran. Noter l'interversion des numéros de | colonne et de ligne. |Il n'y a pas de test sur les valeurs de s2 et s1.
#4D132h	^CurGrob |Scrolling de 1 pixel vers le haut, avec répétition | si la touche "Flèche-haut" reste enfoncée, | sur le Grob courant
#4D150h	<CurGrob |Scrolling de 1 pixel vers la gauche, avec répétition | si la touche "Flèche-gauche" reste enfoncée, | sur le Grob courant
#4D16Eh	vCurGrob |Scrolling de 1 pixel vers le bas, avec répétition | si la touche "Flèche-bas" reste enfoncée, | sur le Grob courant
#4D18Ch	>CurGrob |Scrolling de 1 pixel vers la droite, avec répétition | si la touche "Flèche-droite" reste enfoncée, | sur le Grob courant
#51690h	^^CurGrob |Scrolling maximum vers le haut, sur le Grob courant
#516AEh	vvCurGrob |Scrolling maximum vers le bas, sur le Grob courant
#516E5h	<<CurGrob |Scrolling maximum vers la gauche, sur le Grob courant
#51703h	>>CurGrob |Scrolling maximum vers la droite, sur le Grob courant

GRAPHISME: Le "Grob courant"

Voici quelques SYSEVALs *afficheurs*. Comme on le voit, les effets sont très variés. Je vous laisse aux explications accompagnant les SYSEVALs.

Adresse	Mnémoniques et commentaires
#0E029h	PutOnCurGrob00(st) |Affiche la chaîne en haut à gauche du Grob courant, en caractères de taille médiane (2 ->GROB) La zone affichée se limite à la taille de la chaîne. Si celle-ci est trop grande, elle est tronquée à ses 22 premiers caractères, et terminée par un ...
#1270Ch	1&2disp(st)! |Sépare la chaîne st en deux, au niveau du premier retour chariot (st => st,"" si aucun RC), puis éxécute "1disp(st)!" (ci-dessous) sur la première partie, et "2disp(st)!" sur la deuxième partie.
#12725h	1disp(st)! |Affiche la chaîne st, en caractères de taille médium (2 ->GROB), sur le Grob courant, sur la première ligne de l'écran. Cet affichage tient donc compte du scrolling éventuel
#12748h	2disp(st)! |Idem, sur deuxième ligne
#12B85h	Chk1&2disp(st)1w |Comme "1&2disp(st)!" en #1270Ch, mais avec un temps d'attente d'environ 1 seconde, puis restaure le contenu initial de la zone servant à l'affichage
#2382Eh	Chk1&2disp(st)!Fr |Comme ci-dessous, mais erreurs "Two Few Arguments" s'il n'y a pas d'argument, et "Bad Argument Type" si l'argument n'est pas une chaîne.
#23865h	1&2disp(st)!Fr |Apelle 1&2disp(st)!, puis gèle la zone ayant servi à l'affichage (Fr = freeze) à condition que cet affichage ait eu lieu sur le Grob de la pile.
#38926h	bip;Chk1&2disp(st);3w |Affichage de la chaîne st, sur le Grob courant, au moyen de "1&2disp(st)!" (en #1270Ch), précédé d'un bip, et suivi d'une temporisation de ≈ 3 secondes. Restaure ensuite la zone de l'affichage.
#39B0Ah	14Disp2BWlines |Affiche une ligne noire en ligne 14, et une ligne blanche en ligne 15, sur une largeur de 131 pixels à la ligne 14 du Grob en cours.
#659DEh	DispEQW(ex) |Affiche, sur le Grob en cours, l'expression ex comme elle apparaît dans Equation-Writer. Il se peut que l'image de cette expression soit trop grande pour tenir dans le Grob courant. Celui-ci est alors agrandi en conséquence. L'affichage se fait en haut à gauche du Grob courant et celui-ci est d'abord complètement "nettoyé"
#6738Fh	Alert(st)! |Exécute un bip, puis l'adresse ci-dessus (#12B85h) en gelant l'écran (qui sera restauré ensuite). C'est cette méthode qu'emploie la HP48 pour envoyer des messages d'erreur sans éxécuter l'erreur.

Les dernières adresses ne sont pas forcément les meilleures. A vous de juger si celles-ci peuvent vous être utiles (la quatrième donne un effet amusant. Faites la blague autour de vous).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#0E047h	RclRows0-15 |Donne le Grob 131x16 des lignes 0 à 15 de la fenêtre | de l'écran sur le Grob courant.
#46BCFh	CurGrobDown |Applique la zone du Grob en cours délimitée par les | points (0,6) et (131,37) (32 lignes) à partir du | point de coordonnées (0,14) (8 lignes plus bas)
#46DF0h	CurGrobUp |Applique la zone du Grob en cours délimitée par les | points (0,14) et (131,45) (32 lignes) à partir du | point de coordonnées (0,6) (8 lignes plus haut)
#12690h	HideScr |(LM) Rend l'écran invisible, sauf le menu. |Il n'y a plus d'écho à l'écran (même après la fin du | programme en cours). Il faut provoquer une erreur | où "rebooter" la HP48 par ON-C pour que l'affichage | revienne. |Le contenu du Grob en cours n'est pas modifié.

◆ RÉCRÉATION!

Le programme suivant va vous donner une idée de la rapidité des opérations de *scrolling*. Pour celà, on crée un objet-graphique de dimensions 354x200 (formé à l'aide de lettres de l'alphabet) que l'on place dans **PICT**. On passe à l'affichage centré de **PICT**, tout en éteignant la ligne des menus (pour mieux voir).

Le programme effectue alors des *scrollings aléatoires* (de 20 pixels dans une direction donnée, pixel par pixel). Il suffit d'appuyer sur une touche pour laisser la HP48 respirer.

'**SCROL**': (Checksum #C33Ah; Taille 372 octets)

```
«  "" 65 90 FOR K K CHR + NEXT DUP DUP + +
   1 20 FOR K
      DUP K DUP 58 + SUB 3 →GROB SWAP
   NEXT DROP #162h #C8h BLANK
   0 19 FOR K
      #0h K 10 * R→B 2 →LIST ROT GOR
   NEXT
   PICT STO #4CF05h SYSEVAL #4E2CFh SYSEVAL
   { #131C8h #13220h #134E4h #1357Fh }
   DO DUP RAND 4 * CEIL GET
      1 20 START DUP SYSEVAL NEXT DROP
   UNTIL KEY END DROP2
»
```

◆ LE GROB DE LA PILE.

Cette troisième partie est consacrée aux opérations portant uniquement sur le *Grob de la pile*. Un certain nombre des remarques qui ont été faites dans les deux premières parties continuent à s'appliquer, notamment pour se qui concerne les conséquences que peut avoir un scrolling de l'écran par rapport au *Grob de la pile* (si les dimensions de ce dernier sont supérieures à 131x64).

Commençons par des formes internes de l'instruction **DISP**:

Adresse	Mnémoniques et commentaires (LM si langage machine)
#12429h	Disp(st,s) |(LM) Affiche st au niveau s (au sens de DISP). | Les caractères sont dans la taille médium. |Si la chaîne fait plus de 22 caractères, elle est | tronquée et terminée par ... |Cet affichage se fait sur le Grob de la pile, dans | la boîte de coordonnées (x=0,y=8*(s-1)) et | (x=130,y=8*(s-1)+7). Il ne tient donc pas compte | du fait que le Grob de la pile a pu être "scrollé"
#1245Bh	1disp(st) |(LM) Equivalent de 1 DISP sur une chaîne. |Analogue à: <1h> Disp(st,s)
#1246Bh	2disp(st) |(LM) Equivalent de 2 DISP sur une chaîne. |Analogue à: <2h> Disp(st,s)
#1247Bh	3disp(st) |(LM) Equivalent de 3 DISP sur une chaîne. |Analogue à: <3h> Disp(st,s)
#1248Bh	4disp(st) |(LM) Equivalent de 4 DISP sur une chaîne. |Analogue à: <4h> Disp(st,s)
#1249Bh	5disp(st) |(LM) Equivalent de 5 DISP sur une chaîne. |Analogue à: <5h> Disp(st,s)
#124ABh	6disp(st) |(LM) Equivalent de 6 DISP sur une chaîne. |Analogue à: <6h> Disp(st,s)
#124BBh	7disp(st) |(LM) Equivalent de 7 DISP sur une chaîne. |Analogue à: <7h> Disp(st,s)
#140ABh	disp(_,r) |Forme interne de l'instruction DISP. |Appelle Ndisp(st,2s) (ci-dessous)
#3A4CEh	NDisp(st,2s) |Affiche la chaîne sur le Grob de la pile, en taille | médium (2 ->GROB), à partir de la ligne s2, et sur | s1 lignes au maximum. |Ce sont les retours-chariots de la chaînes qui déter- | -minent les passages à la ligne. |Si la longueur d'une ligne est supérieure à 22 , elle | est tronquée et terminée par un caractère ... |L'affichage se fait cadré à gauche dans le Grob de la | pile, sans tenir compte d'un "scroll" éventuel.

Les SYSEVALs de ce tableau permettent de rappeler, réinitialiser, ou stocker le *Grob de la pile*.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#12655h	RclStkGrob |(LM) Rappelle le Grob de la pile (sans newob)
#130ACh	ResetStkGrob |Sélectionne le Grob de la pile comme Grob courant, l'efface, et redonne au besoin à ce Grob ses dimensions standards (131x56).
#1314Dh	SelectStkGrob |(LM) Le Grob de la pile devient le Grob courant
#503C5h	text |Forme interne de TEXT (se contente d'appeler "SelectStkGrob", ci-dessus, à l'adresse #1314Dh)
#503D4h	lcd-> |Forme interne de LCD->. |On obtient un Grob de dimensions 131x64, obtenu en superposant le rectangle 131x56 situé en haut à gauche du Grob de la pile, et le Grob des menus.
#50438h	->lcd(go) |Place le Grob dans l'objet-graphique de la pile, à partir du point supérieur gauche de celui-ci. |L'objet-graphique de la pile est d'abord complètement nettoyé, puis le Grob à placer est éventuellement tronqué pour tenir dans l'objet-graph. de la pile
#5046Ah	cllcd |Forme interne de CLLCD. Vide le Grob de la pile

Les SYSEVALs qui suivent montrent qu'on peut vraiment utiliser le *Grob de la pile* pour faire du graphisme (sans toucher à **PICT**).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#1383Bh	StkPixoff(2s) |(LM) Eteint le point de coordonnées relatives (x=s2,y=s1) dans le Grob de la pile.
#1384Ah	StkPixon(2s) |(LM) Allume le point de coordonnées relatives (x=s2,y=s1) dans le Grob de la pile.
#13992h	StkPix?(2s)? |(LM) Teste si le point (x=s2,y=s1) dans le Grob de la pile est allumé. Réponse booléenne.
#50AF9h	StkTLine(4s) |(LM) Trace une ligne en mode xor sur le Grob de la pile, entre les points de coordonnées (s4,s3) et (s2,s1). s2 doit être supérieur à s1. |Il n'y a pas de test vérifiant que les extrémités appartiennent ou non au Grob de l'écran.
#50B08h	StkLineoff(4s) |(LM). Efface une ligne du Grob de la pile.
#50B17h	StkLine(4s) |(LM). Trace ligne en mode "or" sur le Grob de la pile

GRAPHISME: Le "Grob de la pile"

Voici un certain nombre de SYSEVALs *afficheurs*. La taille des caractères obtenus est celle utilisée pour l'affichage des objets de la pile (c'est la taille *maximum*, obtenue par **3 →GROB**, alors que l'instruction **DISP** et ses formes internes, vues au début de cette partie, utilisent la taille *médium*, obtenue par **2 →GROB**).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#123C8h	BigDisp(st,s) |(LM) Affiche la chaîne st au niveau 5-s de la pile, en caractères de grande taille. Si s=<0h> ou si s est supérieur à 4, alors affichage au niveau 1. Si la chaîne fait plus de 22 caractères, elle est tronquée et terminée par ... Cet affichage se fait sur le Grob de la pile, dans la boîte de coordonnées (x=0,y=10*(s-1)+17) et (x=130,y=10*(s-1)+26). Il ne tient donc pas compte du fait que le Grob de la pile a pu être "scrollé"
#123E5h	BigDisp1(st) |(LM) affiche st au level 1 de la pile, en caractères de grande taille. Analogue à: <4h> BigDisp(st,s)
#123F5h	BigDisp2(st) |(LM) Comme ci-dessus, au niveau 2 de la pile
#12405h	BigDisp3(st) |(LM) Comme ci-dessus, au niveau 3 de la pile
#12415h	BigDisp4(st) |(LM) Comme ci-dessus, au niveau 4 de la pile

Voici de quoi afficher de beaux messages... A part la première adresse, les affichages se font sur la zone des menus.

Adresse	Mnémoniques et commentaires
#38908h	bip;1&2StkDisp(st)Fr |Revient au Grob de la pile pour affichage. Celui-ci se fait au moyen de "1&2disp(st)!" (en #1270Ch) c'est-à-dire sur deux lignes (avec séparation au premier retour-chariot rencontré), et en tenant compte d'un scrolling éventuel. Il y a d'abord un bip, puis la zone de l'affichage est gelée par 1 Freeze
#4A000h	DispOnMenu(st)1W |Exécute "DispOnMenu(st)" puis une pause d'une seconde
#4A055h	DispOnMenu(st)! |Affiche la chaîne st sur la ligne de menu, avec un appel à "DispOnMenu(st)" (ci-dessous), puis attend qu'une touche soit actionnée. Il y a alors retour au menu initial (la touche est ignorée).
#4E6E5h	DispOnMenu(_) |Transforme obj1 en chaîne puis appelle DispOnMenu(st)
#4E6EFh	DispOnMenu(st) |Affiche la chaîne st sur la ligne de menu, en petits caractères. Si trop longue, elle est tronquée

Enfin voici les formes internes de l'instruction **FREEZE**. Dans la deuxième partie du tableau, on trouve des adresses diverses ayant comme point commun d'effectuer un *Freeze* sur tout l'affichage. A vous de voir si elles sont utiles...

Adresse	Mnémoniques et commentaires	
#142FBh	freeze(r)	Forme interne de FREEZE.
#3902Ch	1freeze	Forme interne de 1 FREEZE
#39072h	4freeze	Forme interne de 4 FREEZE
#39207h	2freeze	Forme interne de 2 FREEZE
#3921Bh	3freeze	Exécute 1freeze puis 2freeze
#3922Fh	7freeze	Exécute 1freeze, 2 freeze, puis 4 freeze
#3A9CEh	off;freeze	
#3FDBDh	drop2;bip;freeze	
#3FDC7h	drop;bip;freeze	
#3FDD1h	bip;freeze	

◆ L'objet-graphique PICT.

Cette quatrième et dernière partie est bien entendu exclusivement consacrée aux opérations portant sur **PICT**, dont on sait qu'il est l'objet graphique sur lequel on travaille dans l'environnement interactif **GRAPH**, ou dans lequel sont *tracées* les expressions (menu **PLOTR**).

Rappelons que l'environnement **Equation-Writer** n'utilise pas **PICT**, mais le *Grob de la pile* (l'image contenue dans **PICT** n'a en effet pas à souffir d'un passage momentané dans l'environnement **Equation-Writer**).

C'est le même souci de préservation du contenu de **PICT**, qui vous est personnel, qui fait que les cartes d'application (Carte **HP-Solve** par exemple) utilisent le *Grob de la pile* pour leurs dessins.

Ce qui a été dit précédemment, notamment dans la partie consacrée au "*Grob courant*", continue à s'appliquer ici. C'est le cas en particulier pour les problèmes liés au *scrolling* de l'écran sur le contenu de **PICT** (si les dimensions de celui-ci sont supérieures à ses dimensions par défaut, à savoir 131x64).

Commençons par les SYSEVALs qui permettent de rappeler, stocker, purger, sélectionner, ..., le contenu de **PICT**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#12665h	RclPictGrob |(LM) Rappelle le Grob de PICT (sans newob)
#12F94h	StoPict(go) |Le grob du niveau 1 est stocké dans PICT
#13061h	PurgePict |(LM) Purge le contenu de PICT, qui devient Grob 0x0
#13135h	SelectPict |(LM) PICT devient le Grob courant
#5187Fh	YXsizePict |Donne 2:<Y> 1:<X>, où Y et X sont respectivement la | hauteur et la largeur de PICT.

Voici successivement les formes internes des instructions **PDIM**, **PVIEW**, **PIXON**, **PIXOFF**, et **PIX?**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#4B206h	pdim(2c) |Forme interne de PDIM, avec deux complexes
#4B300h	pdim(2b) |Forme interne de PDIM, avec deux binaires
#4B323h	pdim(2s) |Place dans PICT un rectangle blanc de largeur s2 et de hauteur s1. NB: si s2 est inférieur à <83h>, il est remplacé par <83h>. Idem avec s1 et <40h>
#4F052h	pview(2s) |Affiche PICT en plaçant le point (x=s2,y=s1) au coin supérieur gauche de l'écran. Vérifie d'abord le format de PICT. Finit en effaçant le Grob du menu.
#4F011h	pview(c) |Forme interne de PVIEW, pour un complexe.
#4F02Fh	pview(li) |Forme interne de PVIEW, pour une liste.
#4CE6Fh	pview({}) |Forme interne de {} PVIEW. Affiche PICT centré et active le clavier de l'environnement GRAPH.
#4CF05h	CenterPview |Affiche PICT centré, mais sans activer le clavier de de l'environnement GRAPH. Appelé par pview({})
#13825h	pixon(2s) |(LM) Allume un point dans PICT. Ne fait aucun test sur PICT ou les coordonnées s2,s1. |Appelé par "pixon(c)" et "pixon(li)"
#4F3EFh	pixon(c) |Forme interne de PIXON, pour un nombre complexe
#4F458h	pixon(li) |Forme interne de PIXON, pour une liste
#1380Fh	pixoff(2s) |(LM) Allume un point dans PICT. Ne fait aucun test sur PICT ou les coordonnées s2,s1. |Appelé par "pixoff(c)" et "pixoff(li)"
#4F471h	pixoff(c) |Forme interne de PIXOFF, pour un nombre complexe
#4F48Ah	pixoff(li) |Forme interne de PIXOFF, pour une liste
#13986h	pix?(2s)? |(LM) Vérifie si le point de coordonnées (s2,s1) est allumé dans PICT. Réponse = True ou False. |Aucun test sur PICT ni sur les coordonnées s2,s1. |Appelé par "pix?(2s)".
#4F4CBh	pix?(2s) |Vérifie si le point de coordonnées (s2,s1) est allumé dans PICT (Réponse 1 ou 0). Teste la validité des coordonnées par rapport aux dimensions de PICT |Appelé par "pix?(c)" et "pix?(li)"
#4F4A3h	pix?(c) |Forme interne de PIX?, pour un complexe
#4F4BCh	pix?(li) |Forme interne de PIX?, pour une liste

Voici maintenant les formes internes des instructions **LINE**, **TLINE**, **BOX** et **ARC** (toutes les quatre dans le menu **PRG\DSPL**):

Adresse	Mnémoniques et commentaires (LM si langage machine)
#4F525h	line(2li) |Forme interne de LINE, pour deux listes
#4F54Dh	line(2li,?) |Exécute "line(2li)" si le booléen vaut True et | "tline(2li)" s'il vaut false.
#4F584h	line(2c) |Forme interne de LINE, pour deux complexes
#4F5ACh	line(2c,?) |Exécute "line(2c)" si le booléen vaut True et | "tline(2c)" s'il vaut false.
#50758h	line(4s) |Trace une ligne dans PICT, en mode "or", entre les | points (s4,s3) et (s2,s1). Il faut que s2 > s1. |Vérifie que les extrémités sont bien dans PICT, et au | besoin tronque la droite à sa partie utile.
#50AEAh	line(4s)! |(LM) Comme line(4s), mais sans vérifier si les | extrémités appartiennent bien à PICT.
#4F539h	tline(2li) |Forme interne de TLINE, pour 2 listes
#4F598h	tline(2c) |Forme interne de TLINE, pour 2 complexes
#5072Bh	tline(4s) |Trace une ligne dans PICT, en mode "xor", entre les | points (s4,s3) et (s2,s1). Il faut que s2 > s1. |Vérifie que les extrémités sont bien dans PICT, et au | besoin tronque la droite à sa partie utile.
#50ADBh	tline(4s)! |(LM) Comme tline(4s), mais sans vérifier si les | extrémités appartiennent bien à PICT.
#4E582h	Box(4s) |Trace une boîte dans PICT, délimitée par les points | de coordonnées relatives (s4,s3) et (s2,s1). |Ne teste pas les dimensions de PICT. |Appelé par "box(2c)" et par "box(2li)"
#4F688h	box(2c) |Forme interne de BOX, pour deux complexes
#4F665h	box(2li) |Forme interne de BOX, pour deux listes
#4FC5Fh	arc(c,3r) |Forme interne de ARC, pour trois réels
#4FD2Ch	arc(li,b,2r) |Forme interne de ARC, pour une liste, un binaire, | et deux réels
#50ACCh	LineOff(4s)! |(LM). Efface ligne dans PICT, d'extrémités les points | de coordonnées (s4,s3) , et (s2,s1) |N.B: s2 doit être > à s4. Il n'y a pas de test pour | vérifier que les extrémités sont dans PICT

Voici les formes internes des instructions **PX→C** et **C→PX**, qui permettent des traductions réciproques entre les coordonnées de type "*pixel*" et les *coordonnées-utilisateur* (qui sont des nombres complexes).

Ces traductions nécessitent, outre la connaissance des dimensions de **PICT**, la lecture des valeurs des points **PMIN** et **PMAX** que l'on trouve aux deux premières positions dans la variable réservée **PPAR** (voir le chapitre réservé aux SYSEVALs des menus **SOLVE** et **PLOT**).

Adresse	Mnémoniques et commentaires
#4F0ACh	px->c(li) |Forme interne de PX->C, pour une liste.
#4F110h	px->c(2rr,2s) |Convertit les coordonnées-pixels d'un point M en ses coordonnées-utilisateur; |Les 1-ères sont données par les réels longs rr4,rr3. |s2 et s1 désignent ici les dimensions de PICT.
#4F179h	c->px(c) |Forme interne de C->PX, pour un nombre complexe.
#4F408h	1c->pxs |Convertit un complexe représentant des coordonnées utilisateur dans PICT en les 2 entiers-système représentant ses coordonnées-pixels.
#4F5D9h	2c->pxs |Convertit deux complexes représentant les coordonnées utilisateurs de 2 points A,B en 4 entiers-système s4,s3,s2,s1 représentant leurs coordonnées-pixels. | ((s4,s3) pour A et (s2,s1) pou B)

Les SYSEVALs suivants effectuent des tests sur le contenu de **PICT** (utiles avant de procéder au tracé d'une droite ou d'une "*boîte*").

Adresse	Mnémoniques et commentaires (LM si langage machine)
#50785h	OutOfPict(4s)? |(LM) Teste si l'un des deux points A,B de coordonnées (s4,s3) et (s2,s1) tombe à l'extérieur de PICT. |La réponse est donnée sous forme d'un booléen. |Les couples (s4,s3) et (s2,s1) sont éventuellement permutés pour qu'à la sortie le point (s5,s4) soit le point le plus à gauche et le point (s3,s2) soit le point le plus à droite.
#508E2h	ChkPictLine(4s) |(s4,s3) sont les coordonnées pixels du point gauche, (s2,s1) celles du point droit. |Modifie éventuellement ces points pour que la ligne qui les joint n'ait pas d'extrémité à l'extérieur de PICT (mais, au plus, sur le bord).
#51166h	ChkPict |Vérifie que PICT est bien au moins rectangulaire de <83h> sur <40h>. Au besoin complète par du blanc.

Le tableau suivant regroupe les formes internes d'instructions de la HP48 où l'un des arguments doit être le nom **PICT**.

Rappelons que l'instruction **PICT**, située à l'adresse **#1E436h**, place le nom **PICT** sur la pile.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#1CA85h	size(PICT) |Une forme interne de SIZE
#1E4A6h	gor(PICT,li,go) |Une forme interne de GOR
#1E4C4h	gor(PICT,c,go) |Une forme interne de GOR
#1E534h	gxor(PICT,li,go) |Une forme interne de GXOR
#1E552h	gxor(PICT,c,go) |Une forme interne de GXOR
#20CADh	rcl(PICT) |Une forme interne de RCL. Il y a Newob
#4F37Ch	sto(go,PICT) |Une forme interne de STO
#4F741h	g?or(PICT,li,go,?) |si le booléen vaut True c'est GOR, sinon c'est GXOR
#4F741h	g?or(PICT,c,go,?) |si le booléen vaut True c'est GOR, sinon c'est GXOR
#4FA2Fh	repl(PICT,c,go) |Forme interne de REPL
#4FA2Fh	repl(PICT,li,go) |Forme interne de REPL
#4FBF6h	sub(PICT,2c) |Forme interne de SUB
#4FBF6h	sub(PICT,li,c) |Forme interne de SUB
#4FC3Ch	neg(PICT) |Une forme interne de NEG
#51148h	PICT?#:BAT! |Erreur "Bad Argument Type" Objet1 <> PICT

Terminons par un bouquet varié de SYSEVALs (le dernier a échappé de peu à la censure).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#47975h	1FrDisp(st) |Affiche la chaîne "st" en première ligne du Grob de la pile, après avoir effacé les lignes 0 à 15, puis affiche les deux lignes noire et blanche en lignes 14 et 15. Gèle la zone d'affichage. |Ne tient pas compte d'un scrolling éventuel.
#479B6h	1&2FrDisp(2st) |Affiche st2 en ligne 1, puis st1 en ligne 2, sur le Grob de la pile, puis gèle l'affichage. |Ne tient pas compte d'un scrolling éventuel.
#4EADCh	PutOnPict(2s,go) |Place Grob1 en position (s3,s2) sur PICT (Teste si le point (s3,s2) est bien dans PICT, et Grob1 est éventuellement tronqué pour ne pas dépasser de PICT)
#5120Bh	RclMaxStdPict |Donne 2:H 1:W, où H et W sont deux entiers-systèmes représentant respectivement: H: le maximum de <40h> et de la hauteur de PICT W: le maximum de <83h> et de la largeur de PICT Celà ne modifie pas les dimensions de PICT.

VARIABLES GLOBALES

La question des variables, des répertoires, et plus largement de l'occupation de la mémoire-utilisateur, recouvre bien des aspects différents. C'est pourquoi plusieurs chapitres seront nécessaires pour avoir une vue d'ensemble.

Les chapitres suivants sont consacrés aux *variables locales*, puis aux *répertoires*, puis aux objets de type *backup*, *librairies*, *XLIB name*, avant un dernier chapitre traitant de questions plus générales (c'est vague).

Si vous cherchez un sujet particulier, j'espère que le découpage que j'ai adopté est suffisamment naturel pour vous mener rapidement droit au but (il est cependant difficile d'établir une classification rigoureuse pour des notions aussi imbriquées).

Ce chapitre regroupe les SYSEVALs qui agissent sur des noms de variables, (et sur les contenus de ces variables), quand celles-ci sont des ***variables globales*** (c'est-à-dire apparaissant quelque part dans un *menu-utilisateur*).

Dans certains cas, il sera également question de variables locales, mais d'une façon très marginale (notamment par exemple quand une instruction se rapporte aussi bien aux variables locales qu'aux variables globales).

Je rappelle les notations employées dans tout le livre pour désigner des noms (dans les mnémoniques):

- Un *nom global* est désigné par: **gn**
- Un *nom local* est désigné par : **ln**
- Un nom quelconque (quand celà n'a aucune importance qu'il soit local ou global) est désigné par: **nm**.

Remarque:

Dans les SYSEVALs qui suivent, une *variable globale* est en général désignée par son nom. Dans certains cas, cependant, elle sera représentée par son contenu (dans la mesure où ce contenu vient d'un rappel par une instruction du type **RCL**, et en supposant qu'il n'a pas subi de *NewOb*).

Celà mérite une petite explication. On sait que la pile ne contient pas des objets, mais les *adresses* de ces objets. Quand on rappelle le contenu d'une variable sur la pile, on place sur celle-ci l'adresse où débute ce contenu.

Dans une telle situation, la variable dont il est question est parfaitement connue. La connaissance de l'adresse du contenu permet d'ailleurs de remonter au nom de la variable (voir le chapitre consacré à l'étude des objets de type répertoire).

D'une certaine manière, la connaissance du contenu d'une variable (plutôt que de son nom) permet des opérations plus rapides (notamment parce qu'on évite ainsi la recherche séquentielle nécessaire pour localiser une variable dont on ne connait que le nom).

Bien sûr, si on rappelle le contenu d'une variable sur la pile, et si l'on éxécute un **NEWOB** sur ce contenu, la pile contient l'adresse de la nouvelle copie du contenu initial, et on a ainsi "*perdu le contact*" entre le contenu de la variable et sa localisation dans la mémoire occupée par le menu **VAR** auquel elle appartient.

Comme on peut le constater, il y a bien des façons de rappeler le contenu d'une variable, ou de vérifier qu'elle existe....

Adresse	Mnémoniques et commentaires (LM si langage machine)
#02FD6h	ChkRcl(nm) |Tente de rappeler un nom local ou global. |Si le nom est inconnu (tant comme variable globale que comme variable locale), alors il y a une erreur "Undefined Local Name"
#07BFDh	dup;exist(gn)? |(LM) Réponse 2:gn 1:booléen, où le booléen vaut 1 si le nom global est celui d'une variable visible dans le répertoire en cours ou dans le chemin menant de ce répertoire à HOME, et 0 sinon.
#20B81h	rcl(nm) |Forme interne de RCL, pour un nom global ou local
#20B9Ah	rcl(li) |Forme interne de RCL, pour une liste. Cette liste doit avoir le format { Chemin Nom }, ou "Chemin" indique dans quel répertoire on cherche ce nom.
#2EA6Ah	RclInHome(nm)? |Tente de rappeler le contenu de la variable 'nm', où 'nm' est un nom local, ou global cherché dans HOME |La réponse est 2:contenu, 1:true ou 1:false (si le nom est inconnu dans HOME).
#5E5B7h	NoRcl(nm)? |Teste si un nom (global ou local) est inconnu. |La réponse est true (nom inconnu) ou false (connu). |On trouve au niveau 2 le contenu (si connu) ou le nom initial (si inconnu). |'nm' peut aussi être un nom XLIB.
#62A2Fh	dup;known(nm)? |Comme ci-dessous avec duplication préalable du nom
#62A34h	known(nm)? |Tente de rappeler le contenu du nom 'nm' |La réponse est 2:contenu, 1:true ou 1:false |Le nom peut être local ou global. S'il est global, il est cherché dans le menu VAR en cours et dans le chemin ascendant vers HOME. |'nm' peut également être un nom XLIB.
#0797Bh	rcl(nm)? |(LM) Tente de rappeler un nom (local ou global) |Le résultat est 2:contenu 1:true, ou 1:false
#4B941h	rcl(nm)?;drop |Comme ci-dessus, puis détruit le booléen
#62C05h	dup;rcl(nm)? |Comme ci-dessus, avec une duplication du nom.
#473A0h	dup;VectoredRcl(nm)? |Rappelle le contenu de 'nm'. Si celui-ci est un nom 'nm2', alors rappelle le contenu de 'nm2'. |Ajoute True si le rappel a pu avoir lieu et False si ce rappel n'a pu se faire. |Cette méthode est employée par la HP48 pour des noms comme 'EQ', 'ΣDAT', 'PPAR', etc...

VARIABLES GLOBALES

On cherche toujours à rappeler le contenu d'un nom, ou à vérifier qu'il existe, mais en se limitant ici au *répertoire-utilisateur* en cours.

Adresse	Mnémoniques et commentaires
#18536h	dup;RclInVars(nm)? |Réponse 3:nm 2:Contenu 1:True, si le nom (local ou global) est connu dans le répertoire VAR en cours. |Sinon la réponse est 2:nm 1:False
#1853Bh	RclInVars(nm)? |Comme ci-dessus, sans la duplication du nom.
#1936Ch	dup;RclInVars(nm) |Comme à l'adresse #18536h, mais il y a une erreur "Undefined Name" si le nom n'est pas connu dans le répertoire VAR en cours.
#19385h	over;RclInVars(nm) |Comme ci-dessus, à ceci près que l'instruction OVER a remplacé l'instruction DUP initiale.
#193A3h	3pick;RclInVars(nm) |Comme en #1936Ch, à ceci près que l'instruction 3 PICK a remplacé l'instruction DUP initiale.
#20962h	RclRealInVars(nm) |Rappelle le contenu de la variable 'nm', qui doit appartenir au menu VAR en cours (sinon "Undefined Name") et avoir un contenu réel (sinon "Bad Argument TYpe")
#48848h	RclInVars(nm) |En fait il s'agit de "RclInVars(nm)?" (#1853Bh), suivi de drop. Si le nom est trouvé, on obtient donc son contenu au niveau 1. Sinon rien...
#48875h	RclInVars(nm)?:q/dup |Exécute "RclInVars(nm)?" puis: |Si la réponse est True: quitte le Rpl en cours. |Si la réponse est False: duplique le nom.

Voici tout d'abord trois façons d'évaluer un nom. Les deux derniers SYSEVALs accompagnent cette évaluation d'une instruction **→num** (indépendamment de l'état du flag -3, qui gouverne le mode symbolique).

Adresse	Mnémoniques et commentaires
#02F4Ch	eval(nm) |Evaluation d'un nom local ou global. Laisse les autres objets inchangés.
#1583Ch	->num(nm) |Evalue un nom puis éxécute ->num si depth≥1. |Ne génère pas d'erreur si la pile est vide après la première évaluation.
#15941h	Chk->num(nm) |Comme ci-dessus mais Erreur "Two Few Arguments" si la pile est vide après la première évaluation.

Voici maintenant les multiples possibilités que nous laissent les SYSEVALs pour modifier ou créer une variable globale.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#07D27h	HardSto(_,nm) |(LM) Utilisé comme forme interne de l'instruction | STO pour stocker un objet dans un nom local. |Permet aussi de stocker un objet dans un nom global, |- Les tags éventuels de l'objet ne sont pas otés. |- Les répertoires ne sont pas protégés...
#08696h	NewSto(_,nm) |(LM) Si le nom existe dans le menu VAR en cours, il | est recréé (peut être utile avec des noms locaux) |Quand plusieurs noms identiques sont présents dans | le même menu VAR, seule la signification du 1er | est utilisée. |Le fait de purger le premier des noms identiques | "démasque" le second, etc...
#18513h	StoT(_,gn) |Forme interne de STO pour les noms globaux. |Les tags de l'objet à stocker ne sont pas enlevés | (Ce SYSEVAL est interne à "sto(_,gn)" en #20CE6h) |Les répertoires sont protégés.
#1955Bh	sto(nm,_) |Stocke l'objet du niveau 1 dans le nom du niveau 2 |Utilise HardSto(_,nm) en #07D27h.
#20CE6h	sto(_,gn) |Forme interne de STO pour les noms globaux. |Les tags de l'objet à stocker sont enlevés
#20D18h	sto(_,ex) |Cette adresse permet de stocker un objet à une | position particulière d'une liste ou d'un tableau. |Les tags éventuels de l'objet sont enlevés.
#1F8CFh	StoTt(_,ex) |Cette adresse permet de stocker un objet à une | position particulière d'une liste ou d'un tableau. |Les tags éventuels de l'objet ne sont pas enlevés. |Cette adresse est appelée par la précédente.
#20DBFh	define(_,nm) |Forme interne de l'instruction DEFINE. |L'objet du niveau 2 est donc stocké dans 'nm'. |Si on est mode d'évaluation numérique (c-à-d si le | flag n°-3 est désarmé), alors l'objet subit une | instruction ->num avant d'être stocké dans le nom.
#2E9E6h	HomeSto(_,gn) |Stockage dans le répertoire HOME |C'est cette instruction qui est utilisée pour créer | les variables réservées comme IOPAR et PRTPAR. |Le stockage se fait avec "HardSto(_,nm)" en #07D27h
#47467h	VectoredSto(_,nm) |Stocke l'objet du niveau 2 dans le nom 'nm', sauf | si le contenu de 'nm' est un nom 'nm2' existant | auquel cas le stockage se fait dans 'nm2'. |Cette méthode est utilisée avec ΣDAT, PPAR, ...

VARIABLES GLOBALES

Un jour où l'autre, il faut se résoudre à purger les variables globales qui ne plaisent plus, ou tout au moins à en changer radicalement le contenu.

Comme on le voit, trois des adresses ci-dessous utilisent le *contenu* d'une variable (*pas son nom*) pour y stocker un autre contenu (mnémonique "***replace(2_)***"), ou pour purger cette variable (mnémoniques "***purge(_)***" et "***PurgeNameOf(_)***").

Adresse	Mnémoniques et commentaires (LM si langage machine)
#085D3h	replace(2_) |(LM) Remplace le contenu d'une variable (au niveau 1) |par l'objet placé au niveau 2 (qui reste sur la pile) |L'objet du niveau 1 (qui doit être le contenu rappelé | d'une variable) ne doit pas avoir subi de Newob.
#08C27h	purge!(nm) |Purge une variable, même si c'est un répertoire.
#08C4Ah	purge(_) |Purge la variable dont le contenu est au niveau 1. |Ce peut être un répertoire non vide.
#1854Fh	purge(gn) |Forme interne de PURGE, pour un nom global
#20F35h	purge(li) |Forme interne de PURGE, pour une liste.
#21047h	PurgeSto(nm,_) |Commence par purger 'nm', puis stocke l'objet du | niveau 1 dans 'nm'. Si cet objet est le contenu | initial de 'nm', celà permet donc de placer 'nm' | en tête du menu courant. |Cette méthode est utilisée par l'instruction ORDER.
#21133h	PurgeNameOf(_) |Purge le nom de variable dont le contenu est placé | au niveau 1 de la pile. |Ce contenu ne doit pas avoir subi de NEWOB.

Les SYSEVALs suivants permettent de rappeler sur la pile la liste de toutes les variables du menu **VAR** en cours, ou bien la liste de certaines d'entre elles. La forme interne de **VTYPE** permet de connaître le type du contenu d'une de ces variables (résultat -1 si la variable est inconnue).

Adresse	Mnémoniques et commentaires
#186E8h	tvars(r) |Forme interne de TVARS, pour un réel
#18706h	tvars(li) |Forme interne de TVARS, pour une liste
#18779h	vars |Forme interne de l'instruction VARS
#1CE55h	vtype(nm) |Forme interne de l'instruction VTYPE
#47B6Eh	tvars9&15 |Donne la liste des noms de variable du menu en cours | contenant une expression ou un directory.
#47BB9h	tvars3&15 |Donne la liste des noms de variable du menu en cours | cours contenant un tableau réel ou un directory

Voici des adresses qui ont pour point commun (sauf la 1ère) de tester des noms, des listes de noms, ou de rappeler de telles listes, sans se préoccuper si ces noms sont effectivement des noms de variables globales.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#082E3h	NameOf(_) |(LM). Donne le nom de la variable dont le contenu est placé au niveau 1 (évidemment ce contenu ne doit pas avoir subi de NEWOB, sinon on risque d'obtenir le nom vide, où un nom imprévisible)
#20BE0h	dup;ListOfNames?#:BAT! |Si l'objet au niveau 1 n'est pas une liste de noms, alors erreur "Bad Argument Type". cet objet est dupliqué avant d'être testé.
#20ECAh	dup;nm?#:"IDEF" |Si l'objet au niveau 1 n'est pas un nom, il y a une erreur "Invalid Definition". Cet objet est dupliqué avant d'être testé.
#212D1h	ChkName2 |Vérifie que le niveau 2 est occupé par un nom. Si ce n'est pas le cas, erreur "Bad Argument Type".
#2E7A4h	dup;#nm? |Répond true <=> l'objet n'est pas un nom
#353ABh	ListOfNames(ex) |Donne la liste des noms figurant dans l'expression

N.B: Le dernier des SYSEVALs ci-dessus est particulièrement intéressant. Il est utilisé de façon interne dans l'environnement **SOLVR** (pour former la liste des variables contenues dans l'équation en cours). Il faut noter que ce SYSEVAL donne des listes ne comportant jamais deux fois le même nom (et ce même si un nom apparait plusieurs fois dans l'expression initiale).

Les SYSEVALs du tableau ci-dessous ne méritent pas de commentaires particuliers (leurs mnémoniques en tiennent lieu). Ce sont des formes internes de **PUT**, **PUTI**, **GET**, **GETI**, et des instructions du menu **STORE** (accessible par ┌─>**MEMORY**).

Je rappelle que l'abréviation "***lr***" désigne aussi bien un réel qu'une liste de un ou deux réels (il s'agit de désigner une position dans une liste ou dans un tableau).

Comme d'habitude, l'abréviation _ ("*souligné*") désigne un objet quelconque.

Adresse	Mnémoniques	Adresse	Mnémoniques	Adresse	Mnémoniques
#1D484h	put(gn,lr,_)	#20412h	sconj(ln)	#20729h	sto/(gn,ar)
#1D565h	put(ln,lr,_)	#20482h	sto+(_,nm)	#207C6h	sto*(_,nm)
#1D65Ch	puti(gn,lr,_)	#204C3h	sto+(nm,_)	#207E4h	sto*(nm,_)
#1D747h	puti(ln,lr,_)	#20583h	sto-(_,nm)	#20802h	sto*(nb,gn)
#1D825h	get(nm,lr)	#205A1h	sto-(nm,_)	#2082Ah	sto*(gn,nb)
#1D926h	geti(nm,lr)	#205BFh	sto-(ar,gn)	#2086Bh	sto*(ar,gn)
#202F1h	sinv(gn)	#205E2h	sto-(gn,ar)	#208ACh	sto*(gn,ar)
#20314h	sinv(ln)	#2066Bh	sto/(_,nm)	#20917h	incr(gn)
#20370h	sneg(gn)	#20689h	sto/(nm,_)	#20980h	incr(ln)
#20393h	sneg(ln)	#206A7h	sto/(gn,nb)	#209CDh	decr(gn)
#203EFh	sconj(gn)	#206E8h	sto/(ar,gn)	#209EBh	decr(ln)

VARIABLES GLOBALES

Enfin j'ai gardé pour la bonne bouche ces SYSEVALs qui permettent de créer, rappeler, ou modifier des variables dans le ***répertoire caché***.

Je rappelle que ce répertoire est attaché au répertoire **HOME**, qu'il porte comme nom le nom global vide, et qu'il est en principe constitué de trois variables:

- '**Alarms**': pour coder la liste des alarmes en cours.
- '**UserKeys**': pour stocker les différentes *réassignations* de touches dans le mode **USER**.
- '**UserKeys.CRC**': contient le *CheckSum* du contenu de '**UserKeys**'.

Le répertoire caché a fait l'objet d'une étude assez complète dans la première partie de ce livre.

Adresse	Mnémoniques et commentaires
#64023h	HidRcl(nm)? |Tente de rappeler le contenu d'une variable du répertoire caché. |La réponse est 2:contenu 1:true si variable cachée existe bien. |Sinon, on obtient false au niveau 1.
#64037h	NextHidEval |Evalue l'objet suivant du rpl, en se plaçant dans le menu caché (pendant le temps de l'évaluation).
#64078h	HidSto(_,nm) |Stocke l'objet situé au niveau 2 dans le répertoire caché, sous le nom figurant au niveau 1.
#6408Ch	HidPurge(nm) |Purge la variable cachée dont le nom est au niveau 1
#640BEh	HidMenu |Fait du répertoire caché le répertoire courant. Seules ses entrées sont utilisables. |Les instructions standards ne sont pas reconnues quand on les entre par la ligne de commande. |Au clavier, utiliser UP ou HOME pour en sortir.

VARIABLES LOCALES

Dans le chapitre précédent, nous avons vu les SYSEVALs qui traitent des *variables globales*, ou qui s'appliquent indifféremment aux noms globaux et aux noms locaux. Intéressons-nous maintenant plus spécifiquement aux *variables locales*.

Pour l'utilisateur, ces variables permettent de stocker temporairement des objets (sans qu'il soit nécessaire de créer une entrée dans un répertoire **VAR**). L'existence de ces objets (leur *accessibilité*) est limitée à la fois dans le temps et dans l'espace à la seule structure locale qui suit leur définition.

L'intérêt des variables locales est double:

- Elles permettent de simplifier la gestion d'une pile parfois trop encombrée, et donne plus de *lisibilité* aux procédures qui les utilisent.
- La durée de vie d'une variable locale étant limitée au strict nécessaire, celà améliore la *portabilité* des programmes (leur indépendance à l'égard du contenu des menus-utilisateur).

L'étude du contenu de la ROM de la HP48 fait apparaître une utilisation fréquente de variables locales. C'est le cas tout particulièrement dans les routines qui assurent le fonctionnement d'environnements complexes (*catalogues*, opérations d'*entrées-sorties*, **GRAPH**, *pile interactive*, **Equation-Writer** et **Matrix-Writer**, etc...).

Ce qui frappe surtout, quand on entreprend le "*décryptage*" des routines internes, c'est l'originalité des outils dont se sont munis les auteurs de la ROM pour *créer*, *utiliser*, et *purger* les variables locales dont ils pouvaient avoir besoin.

La principale nouvauté est la possibilité de créer d'un coup tout un lot de variables locales pour ensuite les utiliser (les *rappeler*, les *modifier*) uniquement en précisant leur ***numéro d'ordre*** (au moment de la création de ce groupe de variables). Il apparaît alors superflu de donner un nom à ces variables locales (ce qui se traduit par un accès plus rapide, la donnée d'un nom nécessitant une recherche séquentielle).

Dans la mesure où on décide d'accéder aux variables créées par leur numéro d'ordre (et que leur nom n'a pas d'importance), il est particulièrement simple d'utiliser des ***noms locaux vides***, comme on le verra ci-après.

Même si la descente au niveau des SYSEVALs fait apparaître, comme on le voit, bien des nouveautés, il reste quelques principes constants qu'il est bon de rappeler.

On peut créer, en même temps, une ou plusieurs variables locales. Si on en crée plusieurs à la fois, elles ont toutes *la même durée de vie*. Il est par exemple impossible de purger l'une d'elles prématurément.

Chaque création de variables locales crée un "*environnement*" dans lequel ces variables sont utilisables, et qui se refermera par une instruction spécifique.

VARIABLES LOCALES

Si plusieurs variables locales portant le même nom sont simultanément "*en vie*", l'utilisation de ce nom ne permettra que d'accéder à la dernière créée de ces variables (pour accéder aux autres, on pourra utiliser un ***numéro d'ordre***).

Comme on va le voir, certains SYSEVALs nous apportent une plus grande liberté de mouvement. Il est cependant prudent de continuer à traiter le problème "*dans les règles de l'art*".

Notamment, si une procédure nécessite l'utilisation d'une ou de plusieurs variables locales, essayez de cerner au mieux la séquence où elles sont utiles, pour les créer au plus tard, et les détruire au plus tôt. Créez les ensembles (et détruisez les ensembles).

◆ Créer des variables locales:

Il y a surtout 3 SYSEVALs permettant de créer des variables locales (le premier et le troisième du tableau ci-dessous sont les plus utiles).

On notera ce point très important: Pour numéroter les n variables locales créées dans un même mouvement, on utilise une numérotation allant de 1 à n, et qui correspond à l'ordre des niveaux de la pile où l'on trouve les objets à stocker.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#074D0h	Ldef(_,li) |Ici "li" est une liste de k noms locaux. |Il y a création de k variables locales. Celle qui | portera le numéro i est la (k+1-i)ème de la liste. |Elle reçoit comme valeur initiale l'objet situé au | niveau i+1. |Ainsi, si li = { n_k .... n_2, n_1 } |L'objet au niveau 2 va dans la var. n°1, nommée n_1 |L'objet au niveau 3 va dans la var. n°2, nommée n_2 | .... |L'objet au niveau k+1 va dans la var. n°k, nommée n_k |"Ldef(_li)" se contente d'éxécuter "Break(cp)", pour | casser la liste "li" (et placer la taille de cette | liste au niveau 1, sous la forme d'entier-système), | puis d'appeler "Ldef(_,s)" ci-dessous.
#074E4h	Ldef(_,s) |(LM) "s" est ici un entier-système non nul. |Il y a création de s variables locales. |Si on représente la pile par: | o_s,...., o_2, o_1, n_s, ..., n_2, n_1, s, | où o-1,..,o_s sont des objets quelconques et où | n_1, n_2, ..., n_s sont des noms locaux, alors: |La i-ème variable a le nom n_i, et le contenu o_i.
#61CE9h	LBlockDef(_,s,ln) |(LM). Crée s+1 variables locales ayant toutes le | même nom (celui qui est placé au niveau 1). |La variable n°1 ne doit pas être utilisée. |La variable n°2 reçoit l'objet situé au niveau 3. |.... |La variable n°(s+1) reçoit l'objet du niveau (s+2)

Remarque (à ne lire que si vous êtes bien réveillé):

Comme on le voit dans les trois SYSEVALs précédents, il faut, semble-t-il, placer sur la pile (éventuellement dans une liste) un certain nombre de noms locaux, avant même que ces noms aient servi à désigner des variables locales (ce sont donc des noms locaux *non encore assignés*).

Celà pose un problème si vous voulez utiliser ces adresses dans un programme traditionnel. En effet, quand vous entrez un nom (ou un objet contenant un nom) en ligne de commande, et que ce nom est inconnu, il est automatiquement considéré par la HP48 comme un *nom global.*

Le seul moyen de placer un nom local sur la pile semble être que ce nom soit déjà le nom d'une variable locale (créée avec l'instruction →).

Il y a là comme un cercle vicieux...

En fait les SYSEVALS précédents acceptent des noms quelconques! Les variables ainsi créées ont des noms qui seront, s'ils sont entrés en ligne de commande (le programme qui les a créés étant, par exemple, suspendu) considérés par la HP48 comme des noms *locaux*.

Mais à l'intérieur du programme lui-même, ces noms seront toujours considérés comme *globaux* (car entrés comme tels au moment où vous avez stocké ce programme, bien avant de l'éxécuter...).

Vous êtes toujours là ? Alors vous méritez bien un exemple:

Considérons le programme suivant:

« 1 2 3 { A B C } #74D0h SYSEVAL A HALT »

Il crée trois variables locales, de noms **'A'**, **'B'**, **'C'**, et de contenus respectifs 1, 2, et 3. Puis il tente d'évaluer le nom **A**, avant que de suspendre le programme par **HALT**.

Si vous lancez ce programme, vous obtenez le nom (*global*) **'A'** sur la pile, et si vous essayez de le rappeler par **RCL**, vous obtenez (à supposer que vous n'ayiez pas déjà créé une variable globale ayant ce nom) le message d'erreur "***Undefined Name***".

Par contre, pendant que le programme est suspendu, si vous entrez le nom **A** en ligne de commande (sans les délimiteurs ') vous obtenez 1. Si vous entrez **'A'** en ligne de commande, le nom qui apparait sur la pile est de type **7** (*nom local*), et si vous le rappelez par **RCL**, vous obtenez le résultat correct **1**.

Que faire pour contourner ces difficultés (tout en restant dans la légalité des programmes traditionnels, entrés en ligne de commande, et utilisant l'instruction **SYSEVAL**)? Il y a plusieurs solutions.

- *Forcer* un nom à être local, au moyen de l'instruction: **#5AEDh SYSEVAL** (mnémonique ***nm=>ln***).

  C'est plutôt fastidieux (il faudrait le faire à chaque fois que le nom apparaît).

- Ne pas utiliser de nom, mais se limiter aux numéros de variables.
  C'est tout à fait possible (d'autant que les SYSEVALs qui viennent le permettent d'une façon tout à fait simple).
  Dans ce cas, l'adresse la plus indiquée est **#61CE9h** (Mnémonique "***LBlockDef(_,s,ln)***"), mais vous vous souviendrez qu'elle crée une variable supplémentaire à ne pas utiliser, et que la première variable "*utile*" porte le numéro 2.

- Utiliser des listes de noms locaux déjà présentes en ROM, et évaluer ces noms locaux par un SYSEVAL à leur adresse.
  C'est jouable, bien qu'un peu lourd. Dans ce cas, on utilisera le SYSEVAL de l'adresse **#074D0h** (Mnémonique "***Ldef(_,li)***").

Encore faut-il, pour employer cette méthode, connaître l'adresse en ROM de noms locaux et de listes de noms locaux. Vous en trouverez beaucoup plus que nécessaire dans le chapitre qui est consacré aux objets de type "*nom*" et dans celui qui décrit les objets de type "*liste*".

Avant de pouvoir développer des exemples crédibles, il faut encore engranger quelques SYSEVALs.

◆ Rappeler des variables locales:

On peut rappeler le contenu d'une variable locale en précisant son ***nom***, ou son ***numéro d'ordre***.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#075A5h	Lget(s) |(LM). Rappelle le contenu de la variable locale | de numéro s (où s est un entier-système).
#07943h	Lrcl(nm)? |(LM) Tente de rappeler un nom en l'interprétant comme | un nom local, même si ce nom est global. |Le résultat est 2:contenu 1:true, ou 1:false
#5E602h	UnKnown(ln)? |Teste si le nom local placé au niveau 1 est inconnu. |S'il est inconnu, la réponse est: 2:Contenu 1:False |S'il est connu, la réponse est: 2:ln 1:True
#61EA7h	LBlockRcl(ln) |C'est l'opération inverse de celle réalisée par | "LBlockDef(_,s,ln)", à l'adresse #61CE9h. |Ici le nom local du niveau 1 est celui sous lequel | on a créé un bloc de variables locales. |La réponse est o_s,...., o_2, o_1, s, où: | o_1,o_2,...,o_s sont les contenus des s variables | utiles (dont les numéros sont 2,3,..,(s+1)), et | où s (entier-système) est le nombre de variables | utiles du bloc (sans compter donc celle qui porte | le numéro 1, et qui est à écarter)

Les adresses suivantes rappellent la variable locale de N°k ($1 \leq k \leq 22$).

On peut ainsi envisager d'avoir simultanément 22 variables locales existantes (ce qui est beaucoup: c'est cependant le nombre de variables utilisées par les environnements **Equation-Writer** et **Matrix-Writer**).

Adresse	Mnémonique	Adresse	Mnémonique	Adresse	Mnémonique
#613B6h	1Lget	#6148Ch	8Lget	#614FCh	15Lget
#613E7h	2Lget	#6149Ch	9Lget	#6150Ch	16Lget
#6140Eh	3Lget	#614ACh	10Lget	#6151Ch	17Lget
#61438h	4Lget	#614BCh	11Lget	#6152Ch	18Lget
#6145Ch	5Lget	#614CCh	12Lget	#6153Ch	19Lget
#6146Ch	6Lget	#614DCh	13Lget	#6154Ch	20Lget
#6147Ch	7Lget	#614ECh	14Lget	#6155Ch	21Lget
				#6156Ch	22Lget

Remarque importante:

La HP48 place dans la même zone de sa mémoire les tables décrivant les créations successives de variables locales et le *bloc* contenant la pile mémorisée par **UNDO**.

Si vous suspendez un programme par **HALT**, après avoir créé des variables locales, et que vous modifiez la pile, la copie de la pile sauvegardée par la fonction **UNDO** va venir se placer "*devant*" le bloc des dernières variables locales créées, modifiant en permanence les numéros d'accès à toutes les variables locales.

Conclusion: n'utilisez pas les adresses désignant une variable locale par un numéro quand vous êtes dans cette situation !!

◆ Modifier des variables locales:

Une fois que des variables locales ont été créées (et on vient de voir comment rappeler leur contenu), on peut songer à les *modifier*...

Voici donc les adresses qui permettent d'intervenir sur le contenu de telle ou telle des variables locales déjà créées (Les SYSEVALs qui suivent ne servent pas à créer ces variables, mais à les modifier !)

Adresse	Mnémoniques et commentaires (LM si langage machine)
#075E9h	Lput(_,s) |(LM). Place l'objet situé au niveau 2 dans la | variable locale portant le numéro s.
#07D1Bh	Lput(_,ln) |(LM). Stocke l'objet du niveau 2 dans la variable | locale (déjà existante !) dont le nom est placé au | niveau 1. Si ce nom n'est pas celui d'une variable | locale, il y a une erreur "Undefined Local Name"
#07D27h	HardSto(_,nm) |Voir dans le chapitre précédent. Si le nom du niveau | 1 n'est pas celui d'une variable locale, crée une | variable globale (et peut écraser un répertoire!).
#20CFFh	sto(_,ln) |Forme interne de STO pour les noms locaux. |Les tags éventuels de l'objet à stocker sont enlevés. |Utilise ensuite "HardSto(_,nm)" en #07D27h.

Comme précédemment, on trouve 22 SYSEVALs pour *stocker* un nouveau contenu dans une variable locale dont on connait le numéro (de 1 à 22). Ces adresses attendent au niveau 1 le nouvel objet à stocker.

Adresse	Mnémonique	Adresse	Mnémonique	Adresse	Mnémonique
#615E0h	1Lput	#61655h	8Lput	#616C5h	15Lput
#615F0h	2Lput	#61665h	9Lput	#616D5h	16Lput
#61600h	3Lput	#61675h	10Lput	#616E5h	17Lput
#61615h	4Lput	#61685h	11Lput	#616F5h	18Lput
#61625h	5Lput	#61695h	12Lput	#61705h	19Lput
#61635h	6Lput	#616A5h	13Lput	#61715h	20Lput
#61645h	7Lput	#616B5h	14Lput	#61725h	21Lput
				#61735h	22Lput

◆ Purger des variables locales:

Il n'y a guère qu'une instruction permettant de détruire (d'un coup) toutes les variables locales dernièrement créées par les instructions du type "***Ldef(_,li)***" (adresse **#074D0h**), "***Ldef(_,s)***" (adresse **#074E4h**), ou "***LBlockDef(_,s,ln)***" (adresse **#61CE9h**).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#07497h	Lpop |(LM). Purge le dernier groupe de variables locales

Plusieurs appels successifs à "***Lpop***" détruiraient les différents groupes de variables locales précédemmment créées, des plus récentes vers les plus anciennes (penser que les différents groupes de variables locales sont placés sur une pile, l'intruction "***Lpop***" détruisant le dernier groupe placé sur cette pile).

◆ Destruction automatique des variables locales:

Supposons qu'un programme (nommé '**P1**' par exemple) contienne une création de variables locales, par l'une des méthodes décrites ci-dessus.

Après l'éxécution complète du programme '**P1**', et retour à la pile-utilisateur, les variables locales créées dans '**P1**' sont automatiquement purgées (et celà même si on n'a pas inclus d'instruction "***Lpop***" dans le corps du programme).

Il semble donc que la HP48 purge toutes les variables locales existantes au moment d'un retour au *mode direct* (clavier, pile, ligne de commande).

Cependant, pour des raisons presque "*déontologiques*", il est fortement recommandé d'inclure une instruction "***Lpop***" avant la fin du programme, pour procéder soi-même à la destruction des variables locales devenues inutiles.

Quand on crée des variables locales de la manière habituelle (par l'instruction →) la HP48 prévoit elle-même de purger ces variables au moment où la structure qui les a créées est terminée.

Considérons les deux exemples suivants:
'**P1**': **« 333 222 111 → a b c « a b + c * » »**
'**P2**': **« 333 222 111 → a b c '(a+b)*c' »**
Ces deux programmes créent les trois variables locales **a**,**b**,**c** (en leur donnant les valeurs respectives 333,222,111) pour ensuite calculer la valeur de **(a+b)*c**.

Quand on examine la manière dont sont codés ces programmes, on constate que:

- Dans '**P1**': l'avant dernier » correspond à l'adresse **#235FEh**, alors que le dernier correspond à l'adresse **#23639h**.
  En fait la routine à l'adresse **#235FEh** contient un appel à la routine "***Lpop***" (adresse **#07497h**).
- De même, dans '**P2**': C'est dans la routine située à l'adresse **#22FEBh** (correspondant à → devant une expression) qu'est inscrite l'instruction "***Lpop***".

Les choses sont différentes si vous créez des variables locales à l'aide d'un des trois SYSEVALs qui ont été décrits ci-dessus ("***Ldef(_,li)***" à l'adresse **#074D0h**, "***Ldef(_,s)***" en **#074E4h**, et "***LBlockDef(_,s,ln)***" à l'adresse **#61CE9h**).

Si, à l'intérieur d'un programme '**A**', vous appelez un programme '**B**' qui crée de telles variables locales, sans les purger, ces variables continuent à exister à l'intérieur de '**A**', après l'éxécution de '**B**'...

◆ Transmission de variables locales:

Considérons les deux programmes suivants:
'**P1**': **« 333 222 111 → a b c « P2 » »**
'**P2**': **« a b c »**

Le premier crée trois variables locales **a**,**b**,**c** (auxquelles il donne les valeurs respectives 333,222,111) puis il appelle le programme '**P2**' qui tente de rappeler la valeur de **a**, **b** et **c**.
Le résultat de l'éxécution de '**P1**' est: 3:'**a**' 2:'**b**' 1:'**c**'

L'explication est que quand vous créez '**P2**', les noms **a**,**b**,**c** ne sont *pas connus* (en tant que *noms locaux*) et sont donc codés comme des *noms globaux*. L'évaluation du nom '**a**' dans '**P2**' ne peut donc qu'évaluer une variable globale ayant ce nom.

Il y a plusieurs solutions pour demander à '**P2**' de rappeler effectivement les noms locaux **a**,**b**,**c** créés dans '**P1**'.

Première solution:

Evaluez le programme suivant: **« 0 0 0 → a b c « HALT » »**

Pendant qu'il est suspendu, les noms **a**,**b**,**c** sont connus comme noms locaux (peu importe leur valeur).

Vous pouvez alors créer le programme '**P2**' ci-dessus, et éxécuter **CONT** pour sortir du mode "*Halté*".

Les noms que contient '**P2**' sont bien les noms locaux **a**,**b**,**c**, et l'éxécution de '**P1**' donne le résultat 3:333 2:222 1:111.

Si vous essayez d'éxécuter '**P2**' isolément, vous aurez bien sûr droit à l'erreur "*Undefined Local Name*".

Deuxième solution:

Dans le programme '**P2**', utilisez **#5AEDh SYSEVAL**, qui force un nom à être un nom local.

'**P2**' pourrait alors s'écrire (mais c'est très inélégant!)

« 'a' 'b' 'c' 1 3 START #5AEDh SYSEVAL EVAL ROT NEXT »

Troisième solution:

Il faut se résoudre à rappeler le contenu des variables locales par leur numéro d'ordre (dans '**P1**', la variable '**a**' porte le numéro 3, '**b**' porte le numéro 2, et '**c**' porte le numéro 1).

Pour celà on peut appeler l'utilitaire suivant:

'**LRCL**': **« #18CD7h SYSEVAL #75A5h SYSEVAL »**

'**LRCL**' attend un réel **r** au niveau 1 de la pile, le transforme en un entier-système **s**, puis rappelle le contenu de la variable locale de numéro **s**.

On peut alors écrire le programme '**P2**' de la manière suivante:

'**P2**': **« 3 LRCL 2 LRCL 1 LRCL »**

et le résultat donné par '**P1**' est bien: 3:333 2:222 1:111.

◆ Déclarations successives de variables locales:

On peut fort bien être amené à créer des variables locales, en plusieurs vagues successives.

Chaque création simultanée de une ou plusieurs variables locales s'accompagne du dépot, sur une pile prévue à cet effet, d'un *bloc d'adresses*.

Si **n** variables ont été créées, la longueur de ce bloc est de **2*n+2** adresses (une adresse = 5 quartets).

Les deux premières adresses contiennent respectivement la longueur totale du bloc, puis un code (sur 5 quartets) utilisé par les procédures de gestion des erreurs (si une erreur survient dans le déroulement du programme, la HP48 doit savoir quels blocs de variables locales elle doit purger).

La présence de ces deux adresses doit être prise en compte dans la numérotation des variables locales (si on veut accéder à un bloc plus ancien que le dernier bloc créé). Voir plus loin.

Chaque variable occupe ensuite deux adresses (une première adresse pointe sur le *nom* de la variable, et la seconde, immédiatement après, pointe sur son *contenu*).

Une nouvelle création de variables locales place un nouveau bloc de *pointeurs* sur cette pile.
En cas d'homonymie, la variable *la plus récente* a priorité.
L'instruction "***Lpop***" détruit uniquement le dernier bloc créé.

Si plusieurs blocs de variables locales existent simultanément, et si ces variables portent des noms différents, il n'y a aucun problème pour accéder à l'une quelconque d'entre elles (utiliser par exemple "***Lrcl(nm)?***" à l'adresse **#07943h**).
Par contre, si on veut accéder aux variables des différents blocs par un numéro d'ordre, il faut tenir compte du ***décalage d'une unité*** que produit dans cette numérotation les deux adresses de début de bloc.

Exemple:

- On crée d'abord trois variables locales **a**,**b**,**c**, par une instruction du type: **{ a b c } Ldef(_,li)**. Ici '**a**' porte le numéro 3, '**b**' le numéro 2, '**c**' le numéro 1.

- On crée ensuite (sans détruire le bloc précédent) un nouveau bloc de deux variables **d** et **e**, par: **{ d e } Ldef(_,li)**.

- A ce moment précis:
  La variable locale numéro 1 est '**e**' et '**d**' porte le numéro 2.
  Le "*numéro*" 3 (au sens des SYSEVALs "***3Lget***" et "***3Lput***", désignerait une variable décrite sur les 2 premières adresses du dernier bloc créé, et on sait que ces 2 adresses n'ont pas cette signification.
  Le numéro permettant d'accéder à la variable '**c**' est donc 3.
  De même, '**b**' porte le numéro 4 et '**a**' le numéro 5.

- Si on éxécute le SYSEVAL "***Lpop***", on détruit le bloc des deux variables '**d**' et '**e**', et les variables '**a**','**b**','**c**' retrouvent leur numérotation précédente.

◆ Evaluation d'un nom local:

Quand un nom local apparaît dans un programme (sans être précédé du *délimiteur* '), il est évalué (tout comme le serait d'ailleurs un nom global, placé dans les mêmes conditions).
Il y a cependant des différences.

- Quand un *nom global* inconnu est évalué, ce nom est simplement placé sur la pile.
- Quand un *nom local* inconnu est évalué, il se produit une erreur "*Undefined Local Name*".
- Quand un *nom global* connu est évalué, son contenu est évalué (si c'est un programme, il est éxécuté).
- Quand un *nom local* connu est évalué, son contenu est rappelé sur la pile.

Pour que le Rpl en cours place un nom local sur la pile, **sans l'évaluer**, il faut que ce nom local soit précédé de l'instruction ' (son rôle est d'éviter l'évaluation de l'objet qui la suit dans le rpl en cours), dont l'adresse est **#06E97h**.

Un autre solution, si on veut utiliser des noms locaux présents dans la ROM (et qui sont souvent éléments de listes exclusivement formées de noms locaux) est d'effectuer un SYSEVAL sur la liste qui contient ce nom local, avant d'effectuer un **GET** pour récupérer le nom.

Exemple:

On veut placer le nom local vide sur la pile, sans l'évaluer.

- A l'adresse **#34D2Bh** se trouve une liste dont le seul élément est le nom local vide.
- Il suffit d'éxécuter **#34D2Bh SYSEVAL 1 GET**, et on obtient bien le nom local vide au niveau 1 de la pile.
- L'adresse de ce nom local vide est **#34D30h**.
  L'instruction **#34D30h SYSEVAL** a donc pour effet d'évaluer ce nom. Je vous déconseille cependant d'*évaluer le nom local vide*, vu l'emploi intensif qu'en fait la HP48 sans que vous le sachiez.

◆ Quelques listes utiles de noms locaux:

Dans l'optique de l'emploi du SYSEVAL "***Ldef(_,li)***" (adresse **#074D0h**) il est utile de rappeler quelques adresses de noms locaux en ROM de la HP48, et plus particulièrement des adresses de *noms locaux vides* (quitte à accéder aux variables locales ainsi créées par leur numéro d'ordre).

Adresse	Mnémonique	Contenu de la liste
#0E475h	{M,N}	Deux noms locaux, M et N.
#24A28h	{i,j}	Deux noms locaux, i et j.
#2D4DBh	{['']}	Une adresse du nom local vide
#2E9F0h	{['']}	Une adresse du nom local vide
#3306Ch	{7['']}	7 adresses du nom local vide
#34D2Bh	{''}	Un nom local vide
#36D5Ah	{2['']}	2 adresses du nom local vide
#36D82h	{3['']}	3 adresses du nom local vide
#52D26h	{4['']}	4 adresses du nom local vide

N.B: Que les listes ci-dessus contiennent des noms locaux ou des adresses de noms locaux ne change absolument rien pour la procédure de création de variable locale définie par "***Ldef(_,li)***" (adresse **#074D0h**).

Il reste maintenant un certain nombre d'adresses concernant les variables locales. La plupart d'entre elles sont des variations autour des adresses qui ont déjà été décrites. Seul le premier de ces SYSEVALs est un peu inclassable.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#61745h	Ldup |(LM). Duplique la dernière création de variables locales. Permet de contourner la destruction auto- -matique des var. locales par la HP48. Exemple: 'P1': « 222 111 -> a b « #61745h SYSEVAL P2 » » 'P2': « #613B6h SYSEVAL #613E7h SYSEVAL » c'est-à-dire "1Lget" puis "2Lget". L'éxécution de 'P1' donne 2:111 1:222
#634CAh	dup;Ldef'' |Comme ci-dessous, mais en dupliquant d'abord l'objet situé au niveau 1
#634CFh	Ldef'' |Crée une variable locale de nom vide, en lui donnant comme valeur l'objet situé au niveau 1.
#285EEh	<1h>;1Lput |Place <1h> dans la variable N°1
#28602h	<0h>;1Lput |Place <1h> dans la variable N°1
#29248h	3dropn;true;1Lput
#2927Ah	drop2;true;1Lput
#292CFh	true;1Lput |Place True dans la variable N°1
#292E3h	drop;true;1Lput |Idem, mais effectue "drop" avant
#43192h	true;dup;1Lput
#43BDAh	drop;3Lput
#482BFh	true;4Lput |Place True dans la variable N°4
#4DE86h	<0h>;3Lput |Place <0h> dans la variable N°3
#5AF51h	false;1Lput |Place False dans la variable N°1
#5AFF1h	dup;1Lput |Duplique puis stocke dans variable N°1
#61610h	dup;4Lput |Duplique puis stocke dans variable N°4
#62F07h	1Lget;swap |Rappelle variable N°2, puis swappe.
#632E5h	2Lget;eval |Evalue le contenu de la variable N°2
#62DB3h	1Lget;Lpop;swap |Lpop = détruit dernières var. locales
#634B6h	1Lget;Lpop |Rappelle contenu var N°1, puis Lpop
#2D6FEh	Lpop;true;false
#2D71Ch	Lpop;2false |2false => 2:False 1:False
#2EF50h	Lpop;true |Lpop puis 1:True
#2EF64h	false;Lpop |1:False puis Lpop
#59628h	drop;Lpop |Drop puis Lpop

RÉPERTOIRES

La structure des répertoires (y compris le répertoire **HOME**) a été développée dans la première partie de ce livre. On a vu en particulier que l'ordre dans lequel apparaissent les objets à l'intérieur d'un *objet-répertoire*, est l'ordre inverse de celui des entrées de menu.

La première variable à apparaître dans le menu est ainsi la dernière à être codée en mémoire dans l'*objet-répertoire* correspondant.

De même, quand on rappelle sur la pile le contenu d'un répertoire (ce contenu apparait encadré par les mots réservés **DIR** et **END**), on y lit les différentes entrées dans l'ordre qui est celui du menu.

Plusieurs des adresses de ce chapitre font référence à l'ordre dans lequel sont placés les différents objets d'un répertoire.

Puisqu'il a bien fallu choisir, j'ai préféré considérer l'ordre dans lequel les objets sont codés en mémoire, dans l'*objet-répertoire* (c'est-à-dire, répétons le, dans l'ordre inverse des entrées de menus).

D'autre part, quelques unes des adresses ci-dessous utilisent un *objet-répertoire*, ou le contenu d'une variable de ce répertoire, placés sur la pile.

Il va de soi que ces SYSEVALs s'intéressent en fait aux adresses de ces objets (ce sont ces adresses qui sont sur la pile, pas les objets que désignent ces adresses, mais je crois que je me répète...).

Tout celà pour en venir au point suivant: Les contenus d'objets requis sur la pile ne doivent pas subir de **NEWOB**, sous peine de *perdre le contact* avec la position initiale en mémoire (et je rappelle à tout hasard que le simple fait d'éditer un objet lui fait subir un **NEWOB**, même si l'édition est immédiatement interrompue par un appui sur **ON** !!).

Voici tout d'abord quelques SYSEVALs simples, qui nécessitent peu de commentaires.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#08D08h	SetCurDir(di) |(LM) Fait de l'objet-répertoire le répertoire courant
#08D5Ah	RclCurDir |(LM) Rappelle sur la pile le contenu du répertoire en | cours (y compris s'il s'agit de HOME).
#08D82h	RclHomeDir |Rappelle sur la pile le contenu du répertoire HOME | (sans oublier le contenu du repertoire caché...)
#08D92h	home |(LM) Forme interne de l'instruction HOME.
#08DD4h	home(di)? |(LM) Teste si l'objet-répertoire est le contenu de | HOME. Réponse booléenne.
#1A16Fh	updir |Forme interne de l'instruction UPDIR.
#1848Ch	path |Forme interne de l'instruction PATH

Les SYSEVALs ci-dessous sont un plus "*hard*". Les quatre derniers font référence à l'ordre dans lequel sont placés les objets dans un *objet-répertoire* (c'est l'ordre contraire des entrées de menu !!)

Adresse	Mnémoniques et commentaires (LM si langage machine)
#08309h	WhatDir(_)? |(LM) Teste si l'objet placé au niveau 1 est le contenu d'une variable d'un répertoire. |Dans la négative, répond 1:false. |Dans l'affirmative, répond 2:dir 1:true, où "dir" est l'objet-répertoire trouvé. |Il est possible que ce répertoire soit HOME lui-même. |Evidemment, l'objet initial ne doit pas avoir subi de NEWOB pour ne pas être coupé de son répertoire. |L'objet initial peut être un sous-répertoire, et dans ce cas on obtient le répertoire-parent.
#08326h	LastObj(di)? |(LM). Teste si l'objet-répertoire est non vide. |S'il est vide, la réponse est 1:False |Sinon, la réponse est 2:LastObj 1:True, où "LastObj" est le contenu du dernier objet de ce répertoire.
#08376h	PrevInDir(_)? |(LM). L'objet du niveau 1 doit être le contenu d'une variable du répertoire en cours. Ce SYSEVAL teste si il y a encore une variable avant celle-ci. |Dans la négative, on obtient 1:false. |Sinon donne 2:PrevObj 1:True, où "PrevObj" est le contenu de la variable précédente.
#18621h	LastObj#''(di)? |Fonctionne comme LastObj(di)? en #08326h, mais sans compter la présence éventuelle d'un nom vide.
#1863Ah	PrevInDir#''(_) |Fonctionne comme PrevInDir(_)? en #08376h mais sans compter la présence éventuelle d'un nom vide.

Voici trois SYSEVALs bien curieux. Les deux premiers sont utilisés par l'instruction **ORDER**, qui pour *réordonner* les variables du répertoire en cours, doit purger et re-stocker ces variables, sans être gênée par des problèmes d'homonymie avec des répertoires "*parents*". Il faut donc ***isoler temporairement*** le répertoire en cours...

Adresse	Mnémoniques et commentaires (LM si langage machine)
#08D4Ah	Isol(di) |(LM) Isole "di" qui doit être le contenu du répertoire courant, de toute l'arborescence. |Seuls, parmi tous les noms de variables, demeurent connus ceux du repertoire isolé.
#08DC4h	StopIsol |Annule l'isolement du répertoire en cours.
#185C7h	EvalNextInIsolDir |Evalue l'objet qui suit dans le Rpl en isolant le répertoire en cours

Puisque l'on parlait à l'instant de l'instruction **ORDER**, en voici trois formes internes (dans un ordre croissant de "*profondeur*").

Suivent deux SYSEVALs un peu inclassables. Le dernier est assez excitant puisqu'il permet de créer des catalogues du type "*Catalogue des équations*" ou "*Catalogue des alarmes*". Tout n'est pas encore très clair dans ce SYSEVAL. Voyez par vous même si le sujet vous intéresse.

Adresse	Mnémoniques et commentaires
#20FF2h	order(li) |Forme interne de l'instruction ORDER
#21006h	order(li)! |Appelé par "order(li)" ci-dessus, mais avec moins de tests. En particulier le répertoire en cours n'est pas isolé de l'arborescence. Si un nom est dans la liste, n'appartient pas au répertoire courant mais à un répertoire "parent", cet objet est déplacé vers le répertoire en cours.
#648BDh	MakeFirstInMenu(_) |L'objet au niveau 1 doit être le contenu d'une variable du répertoire courant. Cette variable devient alors la dernière dans l'objet-répertoire, et donc la première entrée de menu. |Cette modification est très rapide.
#2DC39h	StringVarsBytes(st) |Ajoute à la chaîne st une chaîne décrivant, ligne par ligne (séparées par des retours-chariots) les différentes entrées du répertoire en cours. |Chaque ligne est de la forme suivante. |"Nom Taille Type Checksum", les tailles et les CheckSums étant écrits en décimal.
#47BF0h	DoCat(di,li) |Crée un catalogue avec les entrées du répertoire "di" qui figurent dans la liste "li" (les éléments de "li" sont des noms, tandis que "di" est un objet de type répertoire). |"di" doit être le contenu du répertoire en cours.

Voici maintenant trois SYSEVALS "testeurs":

Adresse	Mnémoniques et commentaires
#184FAh	dir?:DNA! |Si l'objet placé au niveau 1 est un objet-répertoire, alors il y a l'erreur "Directory Not Allowed"
#1856Dh	<>0(di)?:NED! |Si l'objet-répertoire placé au niveau 1 est non vide, il y a l'erreur "Non Empty Directory".
#18608h	ChkEmptyDir(di) |Teste si le répertoire est vide. Dans le cas contraire il y a l'erreur "Non empty Directory" et on trouve le contenu du dernier objet au niveau 1.

Voici pour finir les formes internes, plus ou moins "*sévères*", des instructions **CRDIR** et **PGDIR**, puis trois SYSEVALs permettant de rappeler le contenu d'une variable en indiquant un "*chemin*" pour y accéder.

Adresse	Mnémoniques et commentaires
#08E0Fh	crdir(nm)! |Forme très interne de CRDIR. Le répertoire est recréé | s'il existe déjà.
#184E1h	crdir(nm) |Forme interne de CRDIR. Si le nom placé au niveau 1 | désigne déjà un répertoire existant non vie, on | obtient une erreur "Non-Empty Directory".
#09752h	pgdir(di)! |Forme interne de l'instruction PGDIR. Le répertoire | désigné par son contenu au niveau 1 est purgé de la | mémoire-utilisateur, et ce même s'il est non vide!
#097EDh	clear(di) |Comme ci-dessus, sauf que le nom du répertoire reste | (mais le contenu est vidé).
#18595h	pgdir(gn) |Forme interne de PGDIR, pour un nom global. Teste | si le nom n'est pas celui d'un répertoire non vide
#0985Bh	rcl(di,li)? |Tente de rappeler le contenu d'une variable X | désignée par un chemin { dir1 dir2 ... dirn X }, | où le contenu de dir1 est placé au niveau 2. |La réponse est 2:contenu 1:true, ou 1:false
#20C4Eh	rcl(li,di) |Rappelle le contenu d'une variable X désignée par le | chemin li = { dir1 ... dirn X }, où le contenu de | 'dir1' est l'objet-répertoire "di". |Si le rappel est impossible, il se produit l'erreur | "Undefined Name"
#20C71h	RclFromHome(li) |Rappelle , sans changer de répertoire courant, le | contenu d'une variable définie par un chemin du | type { HOME di1 di2 ... din var } |le chemin doit commencer par HOME, et se terminer | par le nom de la variable.

LES NOMS XLIB

On sait que les ***noms XLIB*** sont utilisés, au sein d'une librairie, pour désigner les objets qui la constituent. Il sont, dans une librairie, ce que sont les noms (en toutes lettres) dans un répertoire-utilisateur.

Un ***nom XLIB*** est déterminé de manière unique par:
- Le *numéro de la librairie* à laquelle il est associé.
- Le *numéro de l'objet* qu'il désigne dans cette librairie.

L'utilisation des ***noms XLIB*** permet d'accéder aux différents éléments d'une librairie d'une manière *indexée*, donc plus rapidement que ne le permettrait une recherche *séquentielle* (qui est le mode d'accès aux objets des répertoires-utilisateurs).

Quand il est placé sur la pile, un ***nom XLIB*** est affiché sous la forme **XLIB L N**, (**L** = n° de librairie, **N** = n° d'objet), sauf s'il correspond à une *entrée nommée* d'un menu de librairie (que cette librairie soit intégrée ou non, qu'elle soit attachée ou non), auquel cas il est affiché "*en clair*" (c'est-à-dire avec le nom qui correspond à cette entrée dans le menu de la librairie, ce nom n'étant pas entouré de délimiteurs ').

Voici trois SYSEVALs qui évaluent un ***nom XLIB***, ou qui testent son existence (en rappelant éventuellement son contenu).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#02FEFh	eval(xn) |Evalue un nom XLIB. Il se produit éventuellement une | erreur "Undefined XLIB Name" s
#07E99h	rcl(xn)? |(LM). Si l'objet au niveau 1 n'est pas un nom XLIB | connu , la réponse est 1:false. |Sinon: on trouve 2:obj 1:true, où "obj" est le | contenu de l'objet désigné par ce nom XLIB. |Dans le cas d'une librairie intégrée (N°2 ou #700h) | "obj" et "xn" sont tous deux affichés par le nom | de la commande ou de la fonction intégrée. |Ex: 2:<2h>,1:<Fh> ==[#07E50h SYSEVAL]==> 1:UFACT | (et le type de UFACT est ici 14: c'est un XLIB) | puis 1:UFACT ==[#07E99h SYSEVAL]==> 2:UFACT,1:True | et ce nouveau UFACT a le type 19 (c'est donc une | commande intégrée)
#5E616h	UnKnown(xn)?: |Teste si le nom XLIB est inconnu. |S'il est inconnu, la réponse est 2:xn 1:true |S'il est connu, la réponse est 2:obj 1:false, | où "obj" est le contenu de l'objet correspondant, | obtenu par "rcl(xn)?".
#62C19h	dup;rcl(xn)? |Voir "rcl(xn)?" en #07E99h

Les SYSEVALs du tableau ci-dessous effectuent diverses conversions:

- Transformation de deux entiers-système en le ***nom XLIB*** qui leur correspond (n° de librairie, n° d'objet).
- Transformation inverse (***nom XLIB*** = > deux entiers-système).
- Transformation du contenu d'un ***nom XLIB*** en ce nom lui-même.
- Transformations réciproques, limitées aux objets des librairies intégrées, entre un ***nom XLIB*** et son contenu (qui est une des fonctions ou commandes intégrées à la HP48).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#07E50h	2s=>xn |(LM). Place sur la pile le nom "XLIB s2 s1", c-à-d désignant l'objet n°s1 dans la librairie n°s2 (sans qu'il soit nécessaire que cette librairie existe, ou que cet objet existe dans cette librairie). |Ex: 2: <12Ah> 1: <3Fh> => 1: XLIB 298 63 | 2: <2h> 1: <34h> => 1: HOME | 2: <700h> 1: <Eh> => 1: HALT | 2: <2h> 1: <180h> => 1: DTAG | 2: <2h> 1: <181h> => 1: XLIB 2 385, | car DTAG est le dernier nom de la librairie N°2
#07E76h	_=>xn |(LM) Crée un nom XLIB à partir de l'objet au level 1. |Cet objet est en principe le contenu d'une entrée d'une librairie (et ne doit pas avoir subi de NEWOB pour que son adresse n'ait pas changé). |Ce SYSEVAL permet alors d'obtenir le nom XLIB de cet objet. C'est donc un peu l'inverse de "rcl(xn)?" |L'objet SIN (Type 18: fonction intégrée), obtenu sur la pile par { SIN } 1 GET, est transformé en le nom XLIB (Type 14: nom XLIB)
#08CCCh	xn=>2s |(LM) Transforme un XLIB name en deux entiers-système: | s2 est la numéro de la librairie. | s1 est le numéro de l'objet dans cette librairie.
#62A61h	dup;RomRpl?:=>xn |Duplique l'objet du niveau 1; Teste ensuite si cet objet est un Rpl de la ROM intégrée. Si oui, alors transforme cet objet en nom XLIB. |Dans la pratique, les procédures et fonctions intégrées à la HP48 sont transformées en les noms XLIB qui leur correspondent (voir aussi #07E76h)
#62A84h	dup;RomXn?:=>rpl |Commence par dupliquer l'objet du niveau 1. |Transforme un nom XLIB correspondant à une librairie intégrée en l'objet éxécutable qui lui correspond (fonction ou commande intégrée. Voir aussi #07E99h) |Les autres objets ne sont pas modifiés.
#27264h	dup;xn?:sk/dq |(LM) Duplique l'objet du niveau 1 puis teste s'il s'agit d'un nom XLIB. Si c'en est un, saute le mot qui suit dans le Rpl en cours, puis continue ... |Sinon, éxcute ce "mot" et sort du rpl en cours.

Terminons enfin par quelques adresses dont le point commun est de vouloir transformer un ***nom XLIB*** en quelque chose de plus *lisible*: un nom global ou une chaîne de caractères (en tout cas le nom "*en clair*" qui apparaît quand on édite un programme contenant des ***noms XLIB***, ou quand le ***nom XLIB*** est affiché sur la pile).

Adresse	Mnémoniques et commentaires
#0821Fh	Name(xn)? |Essaie de transformer le nom XLIB en un nom global "en clair". Ajoute True si ça a marché. Sinon la réponse est 1:False. Ex: 2:<2h>,1:<Fh> ==[#07E50h SYSEVAL]==> 1:UFACT ==[#0821Fh SYSEVAL]==> 2:'UFACT' 1:True Ex: 2:<Fh>,1:<1h> ==[#07E50h SYSEVAL]==> 1:XLIB 15 1 ==[#0821Fh SYSEVAL]==> 1:False
#1616Ch	->str(xn) |Transforme un nom XLIB en chaîne. Ex: 1: XLIB 17 238 => 1: "XLIB 17 238" Si nom XLIB correspond à une entrée nommée du menu d'une librairie, c'est la chaîne formée par ce nom qui est obtenue. Ex: 2:<2h>,1:<Fh> ==[#07E50h SYSEVAL]==> ==[#1616Ch SYSEVAL]==> 1:"UFACT"
#62B1Fh	NameProc? |Essaie de trouver un nom correspondant à l'objet figurant au niveau 1. Si cet objet est un nom XLIB, appelle "->str(xn)" et ajoute true. Si cet objet est basé dans la ROM intégrée, et correspond au contenu d'une fonction ou commande intégrée, alors donne le chaîne égale à ce nom et ajoute True. Dans les autres cas, on obtient l'objet au niveau 2 et False au niveau 1. Cette adresse est appelée pour décompiler un objet qui doit être affiché ou édité.
#62B5Bh	->str(xn);true |Ajoute True après le résultat ci-dessus.

LIBRAIRIES

La structure, fort complexe, des librairies de la HP48 a été complètement explicitée dans un chapitre de la première partie de ce livre. On ne reviendra pas ici sur la signification des différents objets qui participent à cette structure (*Hash-Table*, *Link-Table*, *objet de configuration*, *table des messages*, ...)

Commençons par les formes internes de l'instruction **ATTACH**, qui permet d'attacher une librairie au répertoire en cours (rappelons que l'on peut attacher un nombre quelconque de librairies au répertoire **HOME**, mais qu'un sous-répertoire de **HOME** ne peut s'en voir attacher qu'une à la fois).

Attacher une librairie à un sous-répertoire de **HOME**, c'est en même temps détacher la librairie qui pouvait déjà y être attachée.

Attacher une librairie (préalablement stockée dans un des ports 0, 1 ou 2) à un répertoire, c'est associer à ce répertoire (et à toute sa descendance) tous les objets de la librairie, comme si ces objets venaient étendre le jeu des commandes intégrées.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#214A9h	ChkN°Lib(r) | Transforme r en un entier-système puis vérifie que | que le résultat est un N° de librairie "attachable" | ou "détachable" c-à-d que #100h ≤ r et r <> #700h | Si r<#100h, où r=#700h, alors "Bad Argument Value" | Appelé au début de "attach(r)" et de "detach(r)", | avant "attach(s)" et "detach(s)"
#21461h	attach(r) | Forme interne de ATTACH, avec un argument réel | Appelle d'abord "ChkN°Lib(r)", puis "detach(s)"
#21C6Fh	attach(s) | Forme interne de ATTACH, pour un entier-système. | L'entier s peut être quelconque.
#07709h	HomeAttach(s)! | (LM). Attache la librairie N°s au répertoire Home, | sans vérifier si elle ne lui est pas déjà attachée
#21C88h	HomeAttach(s) | Attache la Librairie N°s au répertoire Home. | Teste auparavant si cette librairie n'est pas déjà | attachée à HOME.
#21CBAh	Attach(di,s) | (LM) Attache la librairie N°s à l'objet-répertoire, | distinct de HOME.
#18A01h	HomeAttach(<2>)! | Attache librairie N°2 au répertoire HOME. | sans vérifier si elle ne lui est pas déjà attachée
#22EA3h	HomeAttach(<700h>)! | Attache librairie N°700h au répertoire HOME. | sans vérifier si elle ne lui est pas déjà attachée

Quelques rappels:

Pour reprendre le tableau précédent, rappelons que les librairies de numéro 2 et #700h=1792 sont les deux principales librairies intégrées à la HP48 (les autres librairies intégrées sont apparemment utilisées à l'intérieur de la ligne de commande, et ne sont pas attachées à **HOME**).

Toute interruption *à chaud* de la HP48 (par **ON-C**) éxécute une relecture de toutes les librairies (intégrées ou stockées dans un port), et les différents *objets de configuration* (s'il en existe) sont alors éxécutés.

Toute modification du contenu des ports 1 ou 2 (ajout ou retrait d'une carte, modification du mode lecture-écriture d'une carte **RAM**) provoque un tel "*Reset*" lors du rallumage de la machine (on évitera ces opérations en laissant la HP48 allumée...).

Enfin, le fait de stocker une librairie dans un port ne suffit pas à ce qu'elle soit prise en compte. Encore faut-il éteindre puis rallumer la machine (l'*objet de configuration* éventuel de la librairie est alors éxécuté).

Venons-en maintenant aux formes internes de l'instruction **DETACH** qui permet de détacher une librairie du répertoire courant.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#21495h	detach(r) |Forme interne de DETACH, avec un argument réel |Appelle d'abord "ChkN°Lib(r)", puis "detach(s)"
#21CE5h	detach(s) |Forme interne de DETACH, pour un entier-système. |L'entier s peut être quelconque.
#076AEh	HomeDetach(s) |(LM) Détache la librairie N°s du répertoire HOME.
#21D2Bh	DetachLib(di) |(LM) Détache la librairie qui est attachée à | l'objet-répertoire "di", distinct de HOME.

Voici deux SYSEVALs qui éxécutent les *objets de configuration* d'une, de plusieurs, ou de toutes les librairies connues, qu'elles soient attachées ou non (à supposer que ces objets existent).

En général l'*objet de configuration* d'une librairie sert à l'attacher automatiquement, mais ça n'a rien d'obligatoire.

Adresse	Mnémoniques et commentaires
#021E4h	RunAllConfig |Exécute les objets de configuration des librairies | connues, qu'elles soient attachées ou non. |Si une librairie ne possède pas d'objet de | configuration, alors rien ne se passe.
#02216h	RunConfig(_,s) |Exécute les objets de configuration (s'ils existent) | des s librairies dont les numéros vont des niveaux | 2 à s+1 (sous forme d'entiers-système)

Les SYSEVALs qui suivent permettent de savoir si une librairie (de numéro bien précis) est attachée à un répertoire, ou de connaître la ou les librairies attachées à un répertoire donné ("*les*" si **HOME**).

On notera le caractère un peu étonnant de l'adresse **#07819h**, qui permet de créér un répertoire vide en précisant dès le départ quelle librairie lui est attachée.

Adresse	Mnémoniques et commentaires
#077C2h	HomeAttached(s)? Teste si la librairie N°s est attachée au répertoire HOME. La réponse est un booléen.
#07819h	MakeDirWithLib(s) Crée un objet-répertoire non vide en spécifiant que la librairie N°s lui est attachée (même si cette librairie n'existe pas). Le fait de prendre s=<7FFh> signifie qu'aucune librairie n'est attachée à ce répertoire (c'est ce que fait l'instruction CRDIR)
#07F98h	LibAttached(di)? Précise si une librairie est attachée au répertoire dont le contenu est "di" La réponse est 1:false ou 2:s 1:true , où s est le N° de librairie Si le répertoire est HOME, et s'il y a au moins une librairie attachée (au moins les librairies <2h> et <700h>), la réponse est 2:s 1:li, où "li" est la liste des numéros de librairies attachées (sous la forme d'entiers-système)
#21D54h	libs Forme interne de l'instruction LIBS. Rappelons que cette instruction donne la liste de toutes les librairies attachées au répertoire en cours. Dans cette liste, une librairie apparaît par son nom suivi de son numéro, suivi du numéro de port où elle est stockée.
#21D5Eh	GetLibs(di) Donne un bloc contenant les noms, les numéros, et les adresses des librairies attachées au directory. La forme de ce bloc est chaine,N°,adr,...,chaine,N°,adr,s où s est la longueur du bloc (multiple de trois) où "chaine" est le nom d'une librairie, N° son numéro (un réel), et "adr" l'adresse où elle débute (entier-système). Du haut du bloc vers le bas, les adresses vont en croissant. Ces adresses pointent en fait sur le début de la zone du nom de la librairie.

Il faut distinguer les librairies *attachées* des librairies "*connues*" c'est-à-dire qui sont stockées quelque part dans un des ports 0,1, ou 2.

Les adresses qui vont suivre concernent précisément les librairies *connues*, sans qu'on demande à ces librairies d'être *attachées* ou non à tel ou tel répertoire.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#0807Fh	WhereLib(s)? |(LM) Teste si la libraire N°s existe (qu'elle soit attachée ou non). La réponse est 1: false, ou 2:adr 1:true où "adr" est un entier système indiquant l'emplacement de la librairie ("adr" est en fait l'adresse du premier quartet de la zone du nom) juste après le champ indiquant la longueur. Il faut enlever 10 à cette adresse pour obtenir le début de l'objet.
#0809Eh	SizeLib(s)? |(LM) Teste si la libraire N°s existe (qu'elle soit attachée ou non). La réponse est 1: false, ou 2:size 1:true où "size" est un entier-système indiquant la taille de l'objet librairie, à partir de la zone du nom
#080C9h	ExistLib≥(s)? |(LM) Teste si la librairie N°s existe (ou une librairie de numéro > s, qu'elles soient attachées ou non). La réponse est 1: false ou 2:num 1:true avec "num" (entier-système) égal au + petit N° de librairie existante (num ≥ s).
#08101h	HashLib(s)? |(LM) Teste si la libraire N°s existe (qu'elle soit attachée ou non). La réponse est 1: false, ou 2:b 1:true où b est un binaire long égal au contenu de la Hash-Table.
#0811Ch	MessLib(s)? |(LM) Teste si la libraire N°s existe (qu'elle soit attachée ou non), et si elle est munie d'une table de messages. |La réponse est 1: false ou 2:adr 1:true où "adr" est un entier-système donnant l'adresse de la table des messages.
#08128h	LinkLib(s)? |(LM) Teste si la libraire N°s existe (qu'elle soit attachée ou non). La réponse est 1: false, ou 2:b 1:true où b est un binaire long égal au contenu de la Link-Table.
#08143h	ConfigLib(s)? |(LM) Teste si la librairie N°s existe (qu'elle soit attachée ou non). La réponse est 1: false, ou 2:obj 1:true où obj est l'objet de configuration éventuel de la librairie.
#081B9h	NameLib(s)? |(LM) Teste si la librairie N°s a un nom. |La réponse est 2:name 1:true, ou 1:false. |Si la librairie n'existe pas, la réponse est false |Si c'est une librairie intégrée (s=<2h> ou <700h>) la réponse est 2:'' 1:true, où '' est le nom global vide.

Les adresses qui vont suivre sont, pour beaucoup, communes avec des adresses du prochain chapitre, qui est consacré aux *objets-backup*.

Il faut savoir que les *objets-librairie* et les *objets-backup* ont en commun de devoir être stockés dans l'un des ports 0,1 ou 2. Partant de là, de nombreux SYSEVALs sont communs.

J'ai néanmoins préféré présenter deux fois ces SYSEVALs, en leur donnant un mnémonique différent (et en les accompagnant d'explications différentes) selon qu'on les fait s'appliquer aux *objets-librairie* ou aux *objets-backup*. On gagne ainsi en lisibilité.

Voici donc des adresses essentielles dans la gestion des librairies par la HP48, à commencer par les SYSEVALs qui permettent de stocker un *objet-librairie* dans tel ou tel port (rappelons que pour placer un *objet-librairie*, ou un *objet-sauvegarde*, dans le port 1 ou dans le port 2, il doit s'y trouver une carte d'extension **RAM** en mode lecture-écriture, et que cette mémoire additionnelle doit être "*libérée*" de la mémoire centrale (instruction **FREE**).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#215BFh	sto(lb,r) |Stocke un objet-librairie dans le port n°r |Le réel r est testé (on vérifie qu'il est bien égal | à 0,1,ou 2).
#215A1h	sto(lb,2s) |Stocke l'objet-librairie "lb", de numéro s2, | dans le port n°s1. |Le nombre s2 ne sert qu'à vérifier qu'une librairie | de même numéro n'existe pas déjà dans ce port | (l'objet "lb" contient déjà le n° de librairie). |Appelé par "sto(lb,r)", en #215BFh.
#09269h	sto0(lb,s) |Stocke, dans le port 0, l'objet-librairie "lb", de | numéro s1. |Le nombre s1 ne sert qu'à vérifier qu'une librairie | de même numéro n'existe pas déjà dans ce port | (l'objet "lb" contient déjà le n° de librairie). |Appelé par "sto(lb,2s)", en #215A1h.
#0AC2Ah	sto0(lb)! |Appelé par "sto0(lb,s)" en #09269h. |Ici, la librairie est stockée, avec son numéro, dans | le port 0, et ce même s'il existe déjà une | librairie portant le même numéro dans ce port !
#0AC70h	sto0(lb)!NoChk |(LM) Comme ci-dessus, mais pas de Garbage Collection | et ne vérifie pas que la mémoire disponible est | suffisante (N.B: je n'ai essayé qu'une fois, et le | résultat a été un Memory Clear. Mais vous faites | comme vous voulez...)
#08F86h	sto12(lb,2s) |Stocke l'objet-librairie "lb", de numéro s2, | dans le port n°s1. |Appelé par "sto(lb,2s)", en #215A1h. |Le nombre s2 ne sert qu'à vérifier qu'une librairie | de même numéro n'existe pas déjà dans ce port | (l'objet "lb" contient déjà le n° de librairie).

Les adresses qui suivent permettent de tenter de rappeler le contenu d'une librairie sur la pile (dans la mesure où le port qui la contient est disponible). Ces adresses ne vous sont données qu'à titre de documentation personnelle, et leur utilisation éventuelle n'engage que votre seule responsabilité.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#09287h	RclLib12(2s)? |Tente de rappeler le contenu de la librairie n° s2 dans le port s1 (avec s1=<1h> ou <2h>). |Le résultat est 2:lb 1:true ou 1:false |Il peut y avoir une erreur "Port Not Avalaible"
#092E1h	RclLib0(s)? |Tente de rappeler le contenu de la librairie n° s dans le port 0. |Le résultat est 2:lb 1:true ou 1:false
#093D1h	RclLib(s,adr)? |Tente de rappeler le contenu de la librairie n°s2, la recherche se faisant à partir de l'adresse "adr" (un entier-système). |La réponse est 2:lb 1:true, ou 1:false.
#21AFDh	RclLib(2s)? |Comme "RclLib12(2s)?", mais ici s1 peut être égal à <0h>,<1h>,ou <2h>.
#09318h	swap;over;RclLib(2s)? |Comme ci-dessus, mais s1 est conservé.

Voici quels sont les SYSEVALs qui permettent de purger une librairie du port où elle est stockée (en supposant bien sûr que le port en question contient de la mémoire en lecture-écriture).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#09408h	PurgeLib12(2s) |Purge la librairie de numéro s2 dans le port s1. |Vérifie que la librairie n'est pas "in use". |Aucun problème si la librairie n'existe pas.
#09453h	PurgeLib0(s) |Comme ci-dessus mais dans le port 0. |"s" est le numéro de la librairie à purger.
#0AD15h	Purge(lb,s) |Purge l'objet-librairie placé au niveau 2, du port spécifié par l'entier-système s.
#0AD47h	Purge!(lb,s) |(LM) Comme ci-dessus mais + interne (douteux)

Voici maintenant un cocktail d'adresses assez intéressantes.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#081EEh	LibNum(lb)? |Teste si l'objet au niveau 1 est une librairie. |La réponse est 2:<Num> 1:true (où Num est le numéro de librairie, sous forme d'entier-système) |Sinon la réponse est 1:false.
#081D9h	LibName(lb)? |(LM) Teste si l'objet au niveau 1 est une librairie. |La réponse est 2:'Name' 1:true (où 'Name' est le nom de librairie, sous forme de nom global) |Sinon la réponse est 1:false.
#08E32h	RclKnownLibs |Rappelle la liste des numéros (sous-forme d'entiers-système) des librairies connues (y compris les librairies intégrées à la HP48). |Pour les librairies non intégrées, on compte celles qui apparaissent dans l'un des ports, qu'elles soient attachées ou non.
#08E73h	dup2;idem(s,lb)? |(LM) "s" est un entier-système et "lb" un objet-libr. |Les duplique et teste si le numéro de la librairie est bien égal à s. Réponse booléenne.
#15D38h	->str(lb) |Donne une chaîne représentant l'objet-librairie.
#215E8h	SLibNum(lb) |Donne le numéro (un entier-système) qui correspond à l'objet-librairie.
#219FEh	LibNum(lb) |Donne le numéro (un réel) de l'objet-librairie.
#3F5BFh	NumEntryLib(s) |Donne un entier-sytème indiquant le nombre d'entrées dans le menu de la librairie N°s, même si elle n'est pas attachée. |On obtient <0h> si cette librairie n'existe pas.
#3F5DDh	NumEntryLib(b) |(LM) Donne le nombre d'entrées d'une librairie, en partant de l'entier binaire long représentant la Hash-Table. |Appelé par "NumEntryLib(s)", en #3F5BFh

Un point méconnu: (utiliser une librairie sans l'attacher)

Si, par exemple, la librairie de numéro 1234 est stockée dans un des trois ports, alors la séquence **1234 MENU** (ou **1234 TMENU**) fait de la première page du menu de cette librairie le menu courant.

On pourrait même accéder à la quatrième page, par exemple, du menu de cette librairie en éxécutant **1234.4 MENU**.

On peut alors utiliser les objets de la librairie directement avec les touches de menu, mais ils restent inconnus en ligne de commande.

Essayez donc **1792 MENU**...

◆ Les librairies intégrées de la HP48:

Voici, réunies dans un même tableau, quelques caractéristiques des librairies intégrées de la HP48. Les librairies **n°2** et **n°1792** (#700h) contiennent les commandes et fonctions intégrées de la calculatrice.

La librairie **n°240** (#F0h) a été évoquée dans la première partie de ce livre. Quant aux autres librairies, j'avoue humblement qu'il reste des zones d'ombre....

Le temps ayant manqué pour les dissiper, voici quelques éléments pour éventuellement engager des recherches sérieuses. J'ai groupé les résultats que donnent pour chacune de ces librairies certains des SYSEVALs commentés dans ce chapitre, à savoir, et dans cet ordre:

- Adresse de la *zone du nom*: ..... ***WhereLib(s)?*** en **#0807Fh**
- *Taille* en quartets: ..... ***SizeLib(s)?*** en **#0809Eh**
- Adresse de la *Hash-table*:..... ***HashLib(s)?*** en **#08101h**
- Adresse de la *table des messages*:..... ***MessLib(s)?*** en **#0811Ch**
- Adresse de la *Link-table*:..... ***LinkLib(s)?*** en **#08128h**
- Adresse de l'*objet-configuration*:..... ***ConfigLib(s)?*** en **#08143h**
- *Nom* de la librairie: ..... ***NameLib(s)?*** en **#081B9h**
- *Nombre d'entrées* non cachées:..... ***NumEntryLib(s)*** en **#3F5BFh**

Numéro	0	1	2	3
Adresse du nom	<028E3h>	<10F14h>	<189E8h>	<29DB3h>
Taille	<952Ch>	<7ACFh>	<A3F8h>	<22CEh>
Hash-Table	sans	sans	<7448Ah>	sans
Table Messages	<72000h>	<72704h>	<72DCFh>	<72F1Eh>
Link-Table	sans	sans	<22651h>	sans
Objet-Config	<0BB40h>	<11086h>	<18A01h>	sans
Entrées	aucune	aucune	<A2h>	aucune

Numéro	5	6	10	11
Adresse du nom	<35504h>	<2C086h>	<32F5Eh>	<0F05Ch>
Taille	<2E2Dh>	<12B5h>	<25A1h>	<1EB3h>
Hash-Table	sans	sans	sans	sans
Table Messages	<736F9h>	<72FE6h>	<72281h>	<726A5h>
Link-Table	sans	sans	sans	sans
Objet-Config	sans	sans	sans	sans
Entrées	aucune	aucune	aucune	aucune

Numéro	12	13	25	240	1792
Adresse du nom	<2D340h>	<0CABEh>	<0BE14h>	<38336h>	<22DE5h>
Taille	<5C19h>	<2599h>	<CA5h>	<253h>	<6FC9h>
Hash-Table	sans	sans	sans	sans	<7427Ch>
Table Messages	<7232Ch>	<7260Ah>	sans	sans	sans
Link-Table	sans	sans	sans	<3834Fh>	<22E08h>
Objet-Config	<2D359h>	<CAD7h>	sans	sans	<22EA3h>
Entrées	aucune	aucune	aucune	aucune	<82Ah>

LES PORTS 0, 1 et 2

Les ***objets-sauvegarde*** (ou "*backup objects*") ont en commun avec les librairies d'"*habiter*" l'un des des ports de la HP48 (le port 0 dans la mémoire vive conventionnelle, et les ports 1 et 2 sur une carte d'extension RAM).

Un ***objet-sauvegarde*** est caractérisé par un nom, un contenu (le contenu de l'objet que l'on a voulu sauvegarder) et un numéro de port.

La séquence pour placer un objet "***obj***", dans le port "***n***", sous le nom "***name***", est: ***obj n:name*** **STO**.

Il arrive souvent que l'objet "***obj***" ainsi sauvegardé soit le contenu d'une variable d'un répertoire-utilisateur, ou même le contenu d'un répertoire entier. Cette variable ou ce répertoire possède un nom, au départ. Il n'est pas nécessaire de donner ce nom à l'*objet backup* créé pour sauvegarder l'objet "***obj***" (bien que pour des raisons de lisibilité, ce soit plutôt recommandé).

On peut créer un ***objet sauvegarde*** dans l'un des ports 0, 1 ou 2. Pour ce qui est des deux derniers, ils doivent bien sûr contenir une carte RAM en lecture-écriture. Par contre, cette mémoire additionnelle ne doit pas être *fusionnée* (la libérer éventuellement avec l'instruction **FREE**).

N.B: le fait de *fusionner* une carte RAM qui était jusqu'ici gérée comme mémoire *indépendante* déplace vers le port 0 les *objets-sauvegarde* (et les librairies) qui pouvaient y être stockés (le port 0 bénéficie du renfort de la mémoire *fusionnée*).

La zone des *objets-sauvegarde* (et des librairies), dans le port 0, augmente de façon dynamique avec le nombre d'objets que vous y stockez. Elle est située à la fin de la mémoire conventionnelle (c'est-à-dire les 32K de la mémoire de base, augmentés de la mémoire additionnelle *fusionnée*).

Comme on l'a dit, les ports 1 et 2 doivent contenir une carte RAM en lecture-écriture, non *fusionnée*, pour pouvoir accueillir des *objets-backup* ou des librairies. Toute la capacité d'une telle carte est alors réservée à ces objets, qui y sont stockés à partir de l'adresse 0, les uns derrière les autres.

Une dernière remarque, qui aidera peut-être à mieux comprendre certains des SYSEVALS qui vont suivre. Quand on crée une variable dans un répertoire-utilisateur, on crée un couple "nom+contenu". Le contenu peut être rappelé ou évalué par l'intermédiaire du nom. En tout cas le nom et le contenu de la variable, même s'ils sont logiquement associés, sont deux entités physiquement séparées.

Au contraire, quand on stocke un *objet-sauvegarde* (ou une librairie) dans un port, on crée un objet nommé (ou numéroté) car il faut bien qu'une entrée dans le menu **PORT0**, **PORT1** ou **PORT2**, permette d'accéder à cet objet.

Le nom de l'objet ainsi créé (ou son numéro) fait alors **physiquement** partie de l'objet lui-même, et ne peut en être dissocié. Pour plus de détails, reportez vous aux chapitres où les structures des *objets-backups* et des *objets-librairies* ont été commentées.

Voici tout d'abord, en deux tableaux successifs, les différentes adresses permettant de créer un *objet-backup* et de le stocker dans un port donné.

Dans ce premier tableau, les SYSEVALs associent un objet quelconque et un nom pour en faire un *objet-backup*, qui est ensuite stocké dans le port spécifié.

Adresse	Mnémoniques et commentaires
#214F4h	sto(_,to) |Stocke l'objet du niveau 2 dans les conditions indi- -quées par l'objet taggué du niveau 1. |C'est une forme interne de l'instruction STO
#21553h	sto2(_,nm) |Stockage dans le port 2, sous le nom "nm" |Appelé par "sto(_,to)" en #214F4h |N.B: l'objet du niveau 2 peut être une librairie, et "nm" être un réel quelconque (non pris en compte)
#2156Dh	sto1(_,nm) |Stockage dans le port 1, sous le nom "nm" |Appelé par "sto(_,to)" en #214F4h |Même N.B que ci-dessus.
#21587h	sto0(_,nm) |Stockage dans le port 0, sous le nom 'nm' |Appelé par "sto(_,to)" en #214F4h |Même N.B que ci-dessus.

Les SYSEVALs du tableau de la page suivante partent d'un *objet-backup* déjà constitué, et permettent de stocker cet objet dans un port disponible.

On remarquera que certaines de ces adresses prennent comme argument (en plus du numéro de port et de l'*objet-backup*) un nom qui doit être celui de l'*objet-backup*.

On pourrait penser qu'il s'agit là d'une information redondante (dans la mesure où le nom figure déjà à l'intérieur de l'*objet-backup*). En fait celà ne sert qu'à vérifier qu'un *objet-backup* portant le même nom n'existe pas déjà dans le port spécifié (sous peine d'erreur "***Object In Use***").

On touche là à une des différences (qui ne sont que des conséquences des différences de structures évoquées ci-dessus) entre les *objets-backup* (ou les *objets-librairie*), et les variables des répertoires **VAR**.

On ne peut en effet changer le contenu d'un *objet-backup*, (tout en gardant le même nom) qu'en purgeant préalablement le premier *objet-backup* portant ce nom (tout simplement, mais je me répète, parce que le contenu de l'*objet-backup* contient le nom sous lequel il est connu).

Vous pouvez très bien indiquer (comme argument "***nm***") un nom n'ayant rien à voir avec l'*objet-backup* "***bk***". Vous contournez ainsi l'interdiction d'homonymie entre deux *objets-backup* du même port, et vous pouvez donc stocker plusieurs *objets-backup* différents sous le même nom.

Vous aurez cependant des difficultés à les purger individuellement (l'instruction **PURGE** testant d'abord que le nom de l'*objet-backup* que vous voulez effacer n'est pas utilisé ailleurs, ce qui est bien sûr le cas...).

Adresse	Mnémoniques et commentaires
#215BFh	sto(bk,r) |Stocke un objet-sauvegarde dans le port n°r. |C'est une forme interne de l'instruction STO. |Le réel r est testé (on vérifie qu'il est bien égal | à 0,1,ou 2).
#215A1h	sto(bk,nm,s) |Stocke l'objet backup dans le port s, sous le nom "nm" |Appelé par "sto(bk,r)", en #215BFh. |N.B: La présence du nom "nm" ne sert qu'à vérifier | qu'un tel nom n'existe pas déjà dans le port n°s | (en effet l'objet "bk" contient déjà son nom) | Si c'est le cas, erreur "Object In Use"
#08F86h	sto12(bk,nm,s) |Stocke l'objet-backup du niveau 3 dans le port s | (s=<1h> ou <2h>) sous le nom "nm". |Appelé par "sto(bk,nm,s)", en #215A1h |Même N.B. que ci-dessus.
#09269h	sto0(bk,nm) |Stocke l'objet-backup du niveau 2 dans le port 0 | sous le nom "nm". |Appelé par "sto(bk,nm,s)", en #215A1h. |Même N.B. que ci-dessus.
#0AC2Ah	sto0(bk)! |Appelé par "sto0(bk,nm)" en #09269h. |Ici, l'objet-backup est stocké, avec son nom, dans le | port 0, et ce même s'il existe déjà un objet-backup | portant le même numéro dans ce port !

Quelques adresses sans grand intérêt, juste pour finir la page....

Adresse	Mnémoniques et commentaires
#21893h	4dropn;"ROM"
#218BCh	3dropn;"SYSRAM"
#219C2h	<port>=>"port" |<0h>=>"0", <1h>=>"1", <2h>=>"2"
#40413h	dup;PortTag?(to) |Teste si le tag de l'objet-taggué est l'un des | objets "0" "1" "2" ou "&". Réponse booléenne.

LES PORTS 0, 1, ET 2

Voici quelques adresses permettant de connaître l'état d'un port:

Adresse	Mnémoniques et commentaires
#21DB0h	ChkPort(s) |Donne le résultat 3:début 2:fin 1:booléen | où début et fin sont les adresses qui délimitent | le port n°s, sous la forme d'entiers-système. |Ici s doit être égal à <1h> ou <2h>. |Le booléen vaut True si le port est occupé et n'est | pas fusionné.
#0AAB2h	PortStatus(s) |Donne l'état du port n°s, avec s=<1h> ou <2h>. |Le résultat est de la forme: | 5:in? 4:ram? 3:merged? 2:taille 1:début, |La taille est exprimée en quartets (<40000h>) |L'adresse de début est <80000h> pour le Port 1, et | <C0000h> pour le Port 2. |in?, ram? et merged? sont des booléens: | in? = True si le port occupé, et False sinon. | ram? = True s'il est occupé par une carte RAM en | lecture-écriture, et False sinon. | merged? = True si la mémoire correspondant à cette | carte est fusionnée.

Voici les SYSEVALs utilisés par la HP48 pour ***libérer***, ou au contraire ***fusionner*** la mémoire d'une carte RAM en lecture-écriture.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#0B037h	merge(s) |(LM) Appelé par merger(r). Fusionne la mémoire du | port n°s, avec s=<1h> ou <2h>
#21398h	merge(r) |C'est la forme interne de MERGE
#21408h	free(2r) |Forme interne de FREE. Libère la mémoire du port r1, | en y transférant la librairie n°r2
#21408h	free(nm,r) |Forme interne de FREE. Libère la mémoire du port r1, | en y transférant l'objet-backup de nom 'nm'. |C'est la même adresse que "free(2r)". Tous les deux | appelent "free(li,r)"
#21B74h	free(li,r) |Forme interne de FREE. Libère la mémoire du port r1, | en y transférant les objets-backup dont les noms | (et les librairies dont les numéros) apparaissent | dans la liste "li".

Suivent cinq SYSEVALs qui ont en commun de faire référence à des adresses précises dans l'un des ports (soit pour rappeler une telle adresse, soit pour l'utiliser).

Pour les adresses qui sont ici des arguments, je les ai désignées par l'abréviation "***adr***". Elles doivent être des entiers-système et, en principe, pointer sur le début d'un objet dans un port (sinon il risque de se produire une erreur "***Invalid Card Data***").

Adresse	Mnémoniques et commentaires (LM si langage machine)
#092F5h	AdrPort0 (LM) Donne l'adresse du début du port 0, sous la forme d'un entier-système. C'est l'adresse où débute le premier objet du port (et qui est le dernier à y avoir été placé).
#093D1h	RclBkp(nm,adr)? Tente de rappeler le contenu de "nm" dans la succession d'objets-backup commençant à l'adresse "adr", donnée sous forme d'entier-système. Dans la pratique, "adr" doit être l'adresse du début d'un objet du port (par exemple l'adresse de début de ce port) La réponse est 2:bk 1:true, ou 1:false.
#0AB51h	FreeAdrInPort(s) (LM) Donne, sous la forme d'un entier-système, la première adresse après la zone occupée par les librairies et les objets-sauvegarde dans le port de numéro s=<0h>,<1h>,<2h>. On obtient donc l'adresse où le prochain objet serait stocké par la HP48.
#0AB82h	NextObjInPort(adr)? (LM) Donne 3:obj 2:NextAdr 1:true si l'adresse "adr" pointe sur un backup-object ou un objet-librairie (cet objet se retrouvant alors au niveau 3) "NextAdr" est alors l'adresse qui suit immédiatement cet objet. La réponse est 1:false s'il n'y a pas d'objet-backup ou d'objet-librairie à l'adresse "adr"
#21972h	RclPortFrom(adr) Comme #21922 (cf plus loin) mais on part de l'adresse "adr", écrite sous la forme d'un entier-système, et on obtient les objets-backup et objets-librairies (ainsi que leur nombre au niveau 1) apparaissant à partir de l'adresse "adr". "adr" doit être l'adresse de début d'un objet-backup ou d'un objet-librairie (sous peine d'une erreur "Invalid Card Data")

Le tableau suivant est en plusieurs parties. On voit d'abord comment "*fabriquer*" un *objet-backup*, à partir d'un nom et d'un contenu (sans stocker cet objet dans un quelconque port, le résultat restant sur la pile).

Ensuite, on voit comment extraire le nom d'un *objet-backup*, ou bien comment tester si un *objet-backup* porte un nom donné. On voit aussi comment extraire d'un *objet-backup* l'objet qui y avait été sauvegardé.

Enfin, on voit les différentes façons de rappeler un *objet-backup* sur la pile, quand on en connait le nom, et le numéro de port. Il s'agit de rappeler ici l'*objet-backup* lui même, et non le contenu de l'objet qui y est sauvegardé.

Autrement dit, si vous avez créé un *objet-backup* de contenu "12345", et de nom '**ABC**' (avec par exemple **"12345" 0:ABC STO**), alors **'ABC' #92E1h SYSEVAL** (voir ci-dessous) place l'*objet-backup* entier sur la pile (et pas la chaîne "12345").

Adresse	Mnémoniques et commentaires (LM si langage machine)
#21624h #21674h	DoBkp(nm,_);true DoBkp(nm,_) |Crée un objet backup avec le nom "nm" et ayant | pour contenu l'objet au niveau 1.
#081D9h #08E73h #0948Eh #215D9h #21761h #217C7h	BkpName(bk)? |(LM) Donne le nom correspondant à l'objet-backup | placé au niveau 1, et ajoute True pour confirmer. |Si le nom n'est pas obtenu, la réponse est 1:False dup2;idem(nm,bk)? |(LM) Duplique préalablement le nom et l'objet-backup. |Teste ensuite 'nm' et le nom de l'objet-backup. |On obtient une réponse booléenne. Rcl(bk) |(LM) Rappelle le contenu de l'objet sauvegardé dans | le backup-object. dup;BkpName(bk) |Donne le nom qui correspond à l'objet-backup. |Cet objet passe au niveau 2. rcl(to) |Forme interne de RCL, pour un objet-taggué eval(to) |Forme interne de EVAL, pour un objet-taggué
#09287h #092E1h #09318h #21AFDh	RclBkp12(nm,s)? |Tente de rappeler l'objet-backup de nom "nm" dans | le port s=<1h> ou <2h>. |Le résultat est 2:bk 1:true ou 1:false RclBkp0(nm)? |Le résultat est 2:bk 1:true ou 1:false over;swap;RclBkp(nm,s)? |Tente de rappeler le contenu de "nm" dans le port | spécifié sous forme d'entier-système "s". |Le résultat est 3:nm 2:bk 1:true ou 2:nm 1:false RclBkp(nm,s)? |Tente de rappeler l'objet-backup de nom "nm" dans | le port s = <0h>,<1h>,ou <2h>.

Voici maintenant les différentes formes internes de l'instruction **PVARS**, qui permet de connaître le statut d'un port ("**ROM**", "**SYSRAM**") et, éventuellement, s'il s'agit de mémoire vive non *fusionnée*, la taille de mémoire encore disponible, ainsi que la liste des *objets-backup* et des *objets-librairie* qui figurent dans ce port.

Dans cette liste, les différents objets apparaissent par leur *nom* (ou leur *numéro* pour les librairies) taggués par le numéro du port.

Adresse	Mnémoniques et commentaires
#2120Bh	pvars(r) | Forme interne de PVARS, pour un argument réel, qui | représente le numéro de port étudié.
#21839h	pvars(s) | Forme interne de PVARS, appelée par pvars(r). | Ici s est un entier-système (s=<0h>,<1h> ou <2h>) | Appelé par "pvars(r)". | Vérifie que l'entier-système s est bien inférieur ou | égal à <2h> (sinon "Bad Argument Type").
#2190Eh	pvars->(s) | Avec s=<0h>,<1h> ou <2h>, donne les différents noms | ou numéros (taggués par le n° de port) des objets- | backup ou objets-librairie du port s, suivi de leur | nombre au niveau 1 (entier-système). | Il suffit alors d'appeler ->list(s) pour obtenir le | résultat de l'instruction PVARS.
#21922h	RclPort(s) | Comme ci-dessus, mais donne les objets-backup et les | objets-librairie du port (plutôt que leurs noms ou | numéros taggués par s) | L'entier s est dupliqué avant ces résultats. | La pile a donc l'allure suivante: | <s>,obj_1,...,obj_n,<n>, où n est un entier-système | indiquant le nombre d'objets trouvés dans le port, | ces objets étant obj_1,...,obj_n (dans l'ordre des | adresses croissantes).
#21931h	RclPort0 | Comme ci-dessus, pour le port 0 seulement (mais | l'entier s n'est pas dupliqué au préalable). | Le résultat sur la pile a donc l'allure suivante: | obj_1,...,obj_n,<n>, où n est un entier-système | donnant le nombre d'objet du port 0 (cf ci-dessus)
#219A9h	pvars(s,B) | On part de la pile: <s>,obj_n,....,obj_1,<n>, où | obj_1,...,obj_n sont des objets-backup ou des | objets-librairie du port numéro s. | Une telle situation est obtenue après utilisation | du SYSEVAL "RclPort(s)". | On obtient la liste de leurs noms (ou numéros), | taggués par le numéro de port: c'est la forme | du résultat obtenu par PVARS.

Voici maintenant les différentes formes internes des instructions **ARCHIVE** (qui permet de stocker le contenu de **HOME**, y compris les alarmes et les *assignations* de touches, dans un *objet-backup*) et **RESTORE** (qui rétablit le contenu de **HOME** précédemment sauvegardé par **ARCHIVE**).

Adresse	Mnémoniques et commentaires
#094A4h	HomeInBck(nm) |Place le contenu du répertoire HOME dans un objet | backup de nom "nm". |Cet objet-backup est laissé au niveau 1.
#21273h	archive(to) |C'est la forme interne de l'instruction ARCHIVE, | l'argument étant un objet taggué. |Si le tag de cet objet est "IO", alors il s'agit | d'un archivage par Kermit sur la voie série. |Sinon il y a un appel à "archive(nm,st)"
#212C7h	archive(nm,st) |Exécute l'instruction ARCHIVE pour le nom "nm", dans | le port donné par st ="0", "1" , ou "2". |Teste que l'objet au niveau 2 est bien un nom, et que | la chaîne est bien égale à "0", "1", ou "2". |Appelle ensuite "archive(nm,s)".
#21A49h	archive(nm,s) |Stocke le contenu du répertoire HOME dans le port | n°s, sous le nom "nm". |Vérifie que ce nom n'est pas "In Use". |Appelle ensuite "archive0(nm)" pour le port 0, ou | "sto(bk,nm,s)", en #215A1h, pour les ports 1 et 2.
#21AB7h	archive0(nm) |Stocke le contenu du répertoire HOME dans le port 0, | sous le nom "nm".
#2134Bh	restore(to) |Forme interne de RESTORE. |Appelle "restore(nm,s)" ou "restore(bk)"
#21AF8h	restore(nm,s) |Restaure le répertoire HOME avec le contenu de | l'objet-backup dont le nom est "nm", et qui se | trouve dans le port n°s (s=<0h>,<1h>,ou <2h>)
#21B2Fh	restore(bk) |Restaure le répertoire HOME avec le contenu de | l'objet-backup (créé précédemment pas ARCHIVE)

Voici, pour terminer, les différentes façons de purger des *objets-backup* du port où ils ont été stockés.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#09408h	PurgeBkp12(nm,s) |Purge l'objet-backup de nom "nm" dans le port s. |Vérifie que l'objet n'est pas "in use". |Aucun problème si l'objet n'existe pas.
#09453h	PurgeBkp0(nm) |Comme ci-dessus mais sur le port 0.
#0AD15h	Purge(bk,s) |Purge l'objet-backup placé au niveau 2, dans le | port spécifié par l'entier-système s.
#0AD47h	Purgebk!(bk,s) |(LM) Comme ci-dessus mais + interne (douteux)
#217F1h	purge(to) |Forme interne de PURGE, pour un objet-taggué

LA MÉMOIRE

La plupart des chapitres qui précèdent contiennent des SYSEVALs dont on peut dire qu'ils agissent *sur* la mémoire, ou *en fonction* de la mémoire. C'est le cas notamment pour les chapitres consacrés aux *menus*, à la *ligne de commande*, aux *variables* globales et locales, aux *répertoires*).

Il y pourtant des SYSEVALs qui transcendent tous ces cas particuliers et traitent de problèmes beaucoup plus généraux liés à la mémoire de la HP48 (toute la mémoire, ou uniquement la mémoire vive, ou bien encore certaines zones de celle-ci comme la zone des *objets temporaires*...). Ce chapitre leur est consacré.

Commençons par quelques variations autour de la notion d'évaluation. S'il y a une instruction de la HP48 qui est implicitement présente tout au long de ce livre, c'est bien l'instruction **SYSEVAL** dont le rôle est d'évaluer un objet dont l'adresse est connue (et donnée sous la forme d'un entier binaire).

Le problème, avec l'instruction **SYSEVAL**, est qu'elle ne permet pas d'évaluer un objet situé dans la ***ROM cachée*** (entre les adresses **#70000h** et **#7FFFFh**). On sait que ces adresses sont partagées avec les 32K de la mémoire libre de base de la HP48.

Et quand on précise une adresse dans cette zone, **SYSEVAL** l'interprète comme une adresse de la mémoire vive....

D'autre part, il peut s'avérer utile de rappeler sur la pile l'objet dont l'adresse est spécifiée, sans l'évaluer.

Pour des objets qui sont de type "*donnée*" (les listes, les réels, ...), il n'y a aucune différence. C'est par contre utile pour les objets éxécutables (les *programmes* RPL; les *expressions*; et les *noms locaux* qui ne doivent pas être évalués s'ils ne sont pas connus, sous peine d'erreur "*Undefined Name*", etc...).

Tous ces problèmes sont résolus avec les formes internes suivantes de l'instruction **SYSEVAL**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#0C612h	sysrcl(s) |(LM) Place sur la pile, sans l'évaluer, l'objet | débutant à l'adresse s (un entier-système). |Si "s" est compris entre <70000h> et <7FFFFh>, | on accède à la ROM "cachée".
#1A547h	syseval(b) |C'est la forme interne de l'instruction SYSEVAL.
#1A556h	syseval(s) |(LM) Appelé par "syseval(b)". L'adresse est ici | donnée sous la forme d'un entier-système. |Si "s" est compris entre <70000h> et <7FFFFh>, | on accède à la mémoire vive...
#5DE69h	syseval!(s) |Comme ci-dessus, sauf dans le cas où "s" est compris | entre <70000h> et <7FFFFh>, car on accède alors à | la ROM "cachée".

Après les différentes formes internes de **SYSEVAL**, revoyons plus simplement comment la HP48 est amenée à évaluer tel ou tel objet.

Dans la plupart des cas, évaluer un objet, c'est évaluer l'*adresse-préfixe* qui débute cet objet.

Comme on le voit ci-dessous, il y a des cas particuliers, comme l'évaluation des *objets taggués*. On notera la possibilté de placer sur la pile un entier-système sans éxécuter le moindre SYSEVAL!

On voit d'autre part que la HP48 a également prévu la possibilité d'évaluer un objet en contrôlant les erreurs éventuelles qui pourraient se produire pendant cette évaluation.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#06F8Eh	eval |(LM) Forme interne de EVAL, pour évaluer les objets qui ne sont ni des expressions, ni des listes ni des objets-taggués. Ces trois types d'objets restent ici inchangés. |En fait cette routine passe directement le contrôle à l'adresse préfixe débutant l'objet à évaluer.
#217C7h	eval(to) |Forme interne de EVAL, pour les objets taggués. |Si cet objet taggué désigne un objet-backup, alors le contenu de ce dernier est évalué. |Si cet objet taggué désigne une librairie, elle est rappelée sur la pile (si le port est disponible) |Sinon le tag est enlevé et le contenu de l'objet taggué est évalué. |Remarque: Si l'entier N n'est pas le numéro d'une librairie du port 0, la séquence 0:N EVAL place l'entier-système <N> sur la pile !
#18EBAh	eval(ex) |(LM) Forme interne de EVAL, pour les expressions.
#18EBAh	eval(li) |(LM) Forme interne de EVAL, pour les listes. C'est la même adresse que pour les expressions.
#06F9Fh	eval(cp) |(LM) Evalue les objets composés: listes, expressions, programmes rpl, objets-unité (ces derniers sont sont décomposés en leurs éléments)
#28670h	ChkEval? |Evalue l'objet au niveau 1, en effectuant un contrôle des erreurs éventuelles, et en passant en mode symbolique pour le temps de l'évaluation. |Si cette évaluation devait aboutir à une erreur, le résultat False est renvoyé au niveau 1, et aux niveaux supérieurs (pas toujours) le ou les arguments qui auraient provoqué l'erreur. |Sinon on obtient au niveau 2 le résultat de cette évaluation, et True au niveau 1.
#28657h	ChkEvalNext? |Comme "ChkEval?" en #28670h, mais l'objet évalué est celui qui suit dans le Rpl.

Voici maintenant les formes internes de l'instruction **BYTES** qui fournit, pour un objet donné, une somme de contrôle (***CheckSum***) sous forme d'un entier binaire, suivie du nombre d'octets occupés par cet objet.

Il y a deux cas particuliers pour l'instruction **BYTES**.

- Si on lui donne un nom de variable, c'est le **contenu** de cette variable qui est analysé.
- Enfin pour un objet basé dans la ROM de la HP48 (comme toutes les commandes et fonctions intégrées, et les entiers de -9 à 9) le résultat est #0h, puis 2.5 (un tel objet n'occupe effectivement que 2.5 octets dans un programme, puisqu'il est en permanence remplacé par son adresse).

Les adresses ci-dessous montrent qu'on peut facilement contourner ces particularités de l'instruction **BYTES**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#1A1FCh	bytes(_) | Forme interne de BYTES, pour un objet quelconque. | Si cet objet est un nom, c'est ce nom qui est pris | en considération, et pas le contenu de la variable | qu'il pourrait désigner. | Les objets basés en ROM (de l'adresse #0 à l'adresse | #7FFFFh) donnent tous le résultat 2:#0h, 1:2.5.
#1A23Dh	bytes(_)! | Comme ci-dessus, sauf pour les objets basés en ROM, | dont le contenu réel est pris en compte. | Ex: 1:SIN ==> 2:#3525h 1:27.5
#1A265h	bytes(nm) | Forme interne de BYTES, pour un nom. Si ce nom est | celui d'une variable, c'est le contenu de cette | variable qui est pris en considération.
#05902h	nibbles(_) | (LM) Donne la taille, en quartets, de l'objet situé | au niveau 1, même s'il est basé en ROM. | Ex: 1:SIN ==> 1:<37h>
#05944h	NibChkSum(_) | (LM) Donne, pour un objet quelconque (même basé en | ROM) la taille en quartets (sous forme d'un entier | système), puis le CheckSum de cet objet (sous la | forme d'un entier binaire) | Appelé par "bytes(_)!" en #1A23Dh et par "bytes(nm)" | Ex: 1:SIN ==> 2:<37h> 1:#3525h

◆ LA ZONE DES OBJETS-TEMPORAIRES:

Vous savez que la pile-utilisateur ne contient pas des objets, mais les ***adresses*** de ces objets. Ce n'est qu'à l'affichage que ces objets sont *décompilés* pour permettre leur identification.

Cette solution a l'avantage de la rapidité. Les opérations sur la pile sont des opérations sur des adresses (5 quartets) et sont donc immédiates (quand on duplique un grand tableau, on ne duplique que son adresse).

Tout celà est "*transparent*" pour l'utilisateur qui ne se pose pas trop de questions. Il n'en va pas de même pour vous, qui souhaitez en savoir davantage...

Les choses se passent différemment selon le type de l'objet qui figure à un niveau donné de la pile:

Les objets intégrés à la HP48:

- Ce sont les *fonctions* et les *commandes* intégrées, ainsi que les entiers de **−9** à **+9**. Ils sont désignés en permanence par leur adresse en mémoire morte (de l'adresse **#0h** à l'adresse **#6FFFFh**).
- Quand on entre un tel objet en ligne de commande (par son nom), la HP48 isole ce nom, s'aperçoit qu'il figure dans une de ses librairies intégrées et trouve l'adresse de ce nom, qu'elle place sur la pile.
- Inversement, une telle adresse, placée sur la pile, est immédiatement convertie (uniquement pour l'affichage) en un nom *lisible*. Toutes les adresses d'objets standards de la HP48 sont en effet précédées d'un numéro permettant d'identifier la librairie concernée et la position dans cette librairie. Il est alors facile de retrouver le nom de l'objet dans la *Hash-Table* de la librairie.

Les contenus de variables:

- Quand on rappelle le contenu d'une *variable globale*, on place en fait sur la pile l'adresse de ce contenu (située à l'intérieur de la zone où est codé le répertoire contenant cette variable).
- Il en va de même avec les *variables locales*. Ce sont les adresses des contenus de ces variables qui sont réellement sur la pile (ces adresses figurent dans une table. Pour en savoir plus, voir le chapitre consacré aux variables locales).

Les objets-temporaires:

- Quand vous créez un objet en ligne de commande (qui ne soit pas de l'un des types précédents), il est placé dans la mémoire vive de la calculatrice à une adresse donnée, et c'est cette adresse qui se trouve réellement sur la pile.
- Nous appelerons ***zone des objets-temporaires*** la partie de la mémoire disponible où sont stockés de tels objets. On peut parler d'*objets temporaires* dans la mesure où ils n'apparaissent que sur la pile, et tant qu'on ne décide pas de les stocker dans une variable.

- Les choses sont beaucoup moins simples qu'il n'y parait à première vue. Quand un *objet-temporaire* est créé, il est évidemment placé à la suite des *objets-temporaires* en cours d'utilisation. La mémoire disponible diminue donc à chacune de ces opérations.
- On pourrait penser que le simple fait d'éliminer un objet par l'instruction **DROP** détruit le corps de cet objet de la mémoire. Il n'en est rien.
- En effet l'instruction **DROP** élimine une adresse de la pile, et c'est tout: l'objet qui était désigné par cette adresse reste en mémoire (ne serait-ce que parce qu'il doit pouvoir être retrouvé par **Last-Stack** ou par **Last-Arg**).
- Il est donc faux de croire que le fait de vider la pile des *objets temporaires* qui y ont été placés libère complètement la mémoire de ces objets. On assiste ainsi à une accumulation d'objets dont un certain nombre ne servent plus.

- Il arrive un moment où la mémoire de la HP48 est saturée. C'est le moment que choisit la calculatrice pour faire "*le ménage*" c'est-à-dire éliminer les objets-temporaires qui ne servent plus. Cette opération porte le nom de "***Garbage Collection***". La HP48 éxécute un "*Garbage Collection*" dans les cas suivants:
  - appel de l'instruction **MEM**,
  - stockage d'un objet-backup ou d'une librairie dans le port 0.
  - libération, par l'instruction **FREE**, d'une carte **RAM** fusionnée.

- L'opération de "*Garbage Collection*" peut prendre un certain temps (une seconde par exemple) et elle peut survenir à tout moment.
  C'est là qu'il faut chercher l'origine de certaines pauses observées ça et là dans l'éxécution d'un programme (par exemple, c'est le plus visible, dans un tracé par **DRAW**).

- Le temps mis à effectuer un "*Garbage Collection*" peut d'ailleurs être mesuré en éxécutant l'instruction **MEM**.
  Si vous éxécutez **MEM** après avoir travaillé un certain temps avec votre calculatrice, vous observerez souvent un certain délai avant l'obtention du résultat.
  Si vous éxécutez **MEM** à nouveau, immédiatement après, vous observerez que le résultat est cette fois-ci quasiment instantané.
- L'explication est simple. Le premier "*Garbage Collection*" a *fait le ménage* dans la mémoire des objets temporaires.
  Le second n'a alors pratiquement plus rien à faire...

Quand un objet temporaire devient-il inutile ?

- On vient de voir qu'un "*Garbage Collection*" permettait d'éliminer tous les objets temporaires devenus inutiles. Mais à partir de quand un tel objet est-il considéré comme inutile ? Contrairemnt à ce qu'on pourrait penser, un *objet temporaire* peut encore être utile longtemps après avoir été éliminé de la pile.

- Un exemple simple permettra de comprendre ce point délicat.

- Si vous placez la liste **{ "ABC" 123456 1_m/s^2 }** sur la pile, vous y placez en fait l'adresse de cet objet, cet adresse renvoyant quelque part dans la mémoire disponible.
  Si vous éxécutez **1 GET** , la liste disparait et vous voyez s'afficher la chaîne **"ABC"** au niveau 1.
  L'adresse de cette chaîne est en fait l'adresse où elle se trouve *à l'intérieur* de la liste qui la contenait au départ. Cette liste ne peut pas être détruite, car cette destruction signifierait celle de la chaîne **"ABC"**, et l'adresse du niveau 1 pointerait alors sur quelque chose d'imprévisible.
- Un "*Garbage Collection*" évitera donc de détruire cette liste, qui contient en son sein des objets qui sont encore sur la pile.
- Il peut s'avérer utile de libérer un objet de la structure à laquelle il est accroché (comme ci-dessus la chaîne **"ABC"** reste accrochée à une liste qui ne sert plus). L'instruction **NEWOB** est faite pour celà.

L'instruction NEWOB:

- Elle recrée une nouvelle copie (dans la zone des objets temporaires) de l'objet situé au niveau 1, et choisit comme adresse du niveau 1 l'adresse de cette nouvelle copie.
- Il y a tout de même une exception: les objets intégrés de la HP48 ne subissent pas cette duplication (et c'est normal dans la mesure où ils résident dans la mémoire morte, à une adresse fixe).
- On verra tout de même que la forme interne de **NEWOB** effectue cette opération sur tous les objets sans exception...

- Si on reprend l'exemple précédent, le fait d'éxécuter **NEWOB** sur la chaîne **"ABC"** la libère de la liste initiale, qui sera donc détruite (car devenue réellement inutile) lors du prochain "*Garbage Collection*".

- Il y a des cas où la HP48 éxécute elle-même l'instruction **NEWOB**:
  Si le contenu d'une variable figure sur la pile, et si on modifie cette variable (ou si on la purge), alors l'objet qui était le contenu initial subit un **NEWOB** (normal sinon on verrait sur la pile apparaître le nouveau contenu de la variable...)
  Quand on édite un objet de la pile, il subit un **NEWOB**. C'est normal au moins pour deux raisons. S'il n'en était pas ainsi et si on avait préalablement dupliqué cet objet, alors tous les niveaux de la pile où se trouve cet objet seraient modifiés en même temps que l'objet édité. Si l'objet édité est le contenu d'une variable (précédemment rappelé sur la pile), le résultat de cette édition ne doit pas rejaillir sur le contenu de la variable (qui lui ne change pas).

- Il y a deux situations un peu analogues aux précédentes où la HP48 crée une nouvelle copie d'un objet (sans se contenter donc d'en conserver l'adresse de l'exemplaire initial).
  Quand on stocke un objet dans une variable, c'est le *contenu* de cet objet qui est stocké, et pas son *adresse* (sauf en ce qui concerne les objets intégrés).
  Quand on inclut un objet dans une liste ou dans un programme, c'est le *contenu* qui est inclus, pas l'*adresse* (même restriction).

Un cas particulier:

- Il y a une exception notable au premier cas ci-dessus.
- Quand on rappelle sur la pile le contenu d'un répertoire et qu'on modifie le répertoire lui-même, son contenu initial sur la pile ne subit pas de **NEWOB** (sans doute pour gagner du temps, les objets-répertoires étant souvent volumineux).
- La preuve en est que ces modifications sont répercutées sur l'*objet-directory* placé au niveau 1.

- Prenons un exemple: imaginons, dans le répertoire **HOME**, un sous-répertoire nommé **'XYZ'**, et contenant les deux variables: **'A'** de contenu 12345 , **'B'** de contenu 67890.
  Plaçons nous dans **HOME** et éxécutons **'XYZ' RCL**:
  Nous obtenons l'*objet-directory* suivant au niveau 1:

```
DIR
  A 12345
  B 67890
END
```

  Entrons alors dans le répertoire **'XYZ'**, et modifions le contenu de la variable **'A'**. Nous voyons se répercuter ces modifications dans l'*objet-directory* placé au niveau 1.
  Il en va de même si on purge une des deux variables **'A'** ou **'B'**, ou si on crée une variable supplémentaire.
  Essayez ensuite de modifiez le répertoire **'XYZ'** après avoir éxécuté l'instruction **NEWOB** sur l'*objet-directory* du niveau 1.
  Vous verrez alors que les modifications apportées au répertoire **'XYZ'** n'apparaissent plus au niveau 1 (le *cordon ombilical* entre le répertoire et l'objet du niveau 1 ayant été rompu...).

J'espère que les explications précédentes vous ont convaincus (ou ne vous ont pas trop rebutés). Il est vrai que le problème des objets temporaires n'est pas un sujet très facile.

Voici de nouveaux SYSEVALs, qui traitent de manière assez générale (et variée) des problèmes de mémoire. Parmi ces SYSEVALs, les formes internes de **MEM** et de **NEWOB**, et l'adresse de la routine éxécutant un "*Garbage Collection*".

On y trouve à la fin le moyen de lire ou d'écrire 16 quartets à une adresse quelconque de la mémoire.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#05F42h	GarbColl |(LM) Effectue un "Garbage Collection"
#05F61h	FreeNibbles |(LM) Donne la mémoire disponible, en quartets, sous forme d'entier-système. Pas de "Garbage Collection" |L'instruction MEM appelle d'abord "GarbColl" puis éxécute "FreeNibbles" avant de transformer le résultat en réel et de le diviser par 2.
#06657h	newob |(LM) C'est la forme interne de NEWOB. |Contrairement à l'instruction NEWOB, l'opération s'applique aussi aux objets basés en ROM.
#0CBA2h	TRM? |(LM) Affiche le message "Try to Recover Memory?" et le menu correspondant (choix YES/NO). |Attention, ç'est "pour de vrai" ....
#62C69h	newob;swap
#1A2DAh	dup;InRom? |(LM) Teste si l'objet au niveau 1 est basé en ROM, entre l'adresse #0h et l'adresse #6FFFFh, quelque soit le type de cet objet (même un External)
#61FA9h	dup;StdObj? |(LM) Teste si l'objet au niveau 1 est basé en ROM et est l'un des objets standards de la HP48 (fonctions et commandes intégrées, entiers de -9 à 9)
#08207h	NameOfStdObj? |(LM) Comme ci-dessus, mais ne duplique pas l'objet initial, et, si le résultat est True, place au niveau 2 le nom (global) identifiant la commande ou la fonction intégrée. |Ex: 1:SIN => 2:'SIN' 1:True
#6594Eh	Put16Nib(2b) |Place le contenu du binaire b2 (16 quartets), à l'adresse indiquée par le binaire b1 |C'est donc l'opération inverse de "Rcl16Nib(2b)"
#6595Ah	Rcl16Nib(2b) |(LM) Rappelle, dans b2, le contenu des 16 premiers quartets (nibbles) à partir de l'adresse b1. |Par exemple, avec b1=#0h, place # 8001FDAD801B9632h dans le binaire b2 (le début de la ROM contenant les quartets 2369B108DAF1008)

Terminons par un tableau d'adresses assez "*hard*", où l'on voit comment la HP48 peut régler les problèmes liés au fait que certaines objets de la pile peuvent être basés à la même adresse (si on décide par exemple de modifier l'un de ces objets).

Je vous laisse aux explications accompagnant ces SYSEVALs, que j'espère suffisamment claires.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#064BDh	dup;FreeRefs |Commence par dupliquer l'objet du niveau 1, puis | libère les objets des niveaux 2 et plus (donc au | moins l'objet du niveau 2) qui sont basés à la | même adresse que l'objet du niveau 1. |La libération de l'objet du niveau 2 se fait par | un "newob" (adresse #6657h) et tous les autres | objets concernés (niveaux 3 et plus) sont basés | à l'adresse qui est celle du niveau 2
#064E2h	(Adr3+=Adr1)=>Adr2 |(LM) Les objets des niveaux 3 et plus, qui auraient | la même adresse que l'objet du niveau 1, sont | remplacés par l'objet du niveau 2 (son adresse)
#065E5h	dup;Ref? |Teste si l'objet correspondant au niveau 1 est placé | à d'autres niveaux de la pile (le test ne détruit | pas l'objet du niveau 1). La réponse est un booléen
#06B4Eh	Dup;FreeObj? |(LM) Teste si l'objet placé au niveau 1 est "libre" | c-à-d appartient à la zone des objets temporaires, | n'est pas référencé, n'est élément d'aucun objet | composé (liste, programme, expression, unité) et | n'est pas le contenu d'une variable (glob. ou loc.) |L'objet est dupliqué. La réponse est booléenne. |En mode d'éxéuction directe, les objets qui font | partie de la pile mémorisée par LastStack ne sont | pas "libres".
#06BC2h	dup;NotRef? |"dup;Ref?", suivi d'une négation logique
#2C4A0h	0;newob |Crée une nouvelle copie de 0
#366BFh	rot;NewobIfNotFree |Voir ci-dessous
#37B44h	NewobIfNotFree |Exécute "newob" (adresse #06657h) sur l'objet placé | au niveau 1 si cet objet n'est pas libre (voir | la définition avec "Dup;FreeObj?" (adr. #06B4Eh)
#37B5Dh	FreeRefs |Exécute "dup;FreeRefs" en #064BDh puis détruit | l'objet du niveau 2 (tout ça à la condition que | "dup;ref?", en #065E5h) ait renvoyé True. |Autrement dit: si certains objets des niveaux ≥ 2 | sont basés à la même adresse que l'objet du niveau | 1, ils sont libérés (leur nouvelle adresse est | commune). L'objet du niveau 1 ne bouge pas
#63F7Eh	swap;NewobIfNotFree |Voir ci-dessus

SYSEVALs DE CONTROLE

Ce chapitre est consacré à des opérations qui permettent d'effectuer certains contrôles sur l'éxécution d'un programme. De telles opérations apparaissent dans le menu **PRG\CTRL**.

Les formes internes de certaines des instructions de ce menu ont déjà été étudiées dans un précédent chapitre. C'est le cas pour les instructions **INPUT**, **PROMPT**, **DISP**, **MENU**, et **KEY**.

Les instructions **DOERR**, **ERRN**, **ERRM**, **ERR0** (toujours dans ce menu) seront évoquées dans le chapitre suivant qui traite de la gestion des erreurs par la HP48.

Il nous reste donc les instructions **WAIT**, **BEEP** et **OFF**, ainsi que les instructions permettant d'éxécuter des programmes "*pas à pas*".

Commençons par les différentes formes internes de **WAIT**. Les pauses générées par ces instructions sont toutes interruptibles par **ON** (ou par le déclenchement d'une alarme). La ou les touches qui seraient actionnées pendant le délai d'attente sont placées dans le *buffer* du clavier (et peuvent donc être traitées par la suite).

On constate que la HP48 permet d'utiliser trois SYSEVALs générant des délais d'attente fixés (ce qui dispense l'utilisateur d'avoir à préciser lui même ce délai). Ces pauses ***ne sont pas interruptibles*** par **ON**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#1A738h	wait(r) |C'est la forme interne de WAIT, y compris quand le réel r est égal à 0 ou à -1.
#1A7B5h	wait(r>0) |Le réel r est ici exprimé en secondes.
#1A7C9h	wait(r>0)! |Appelé par "wait(r>0)" en #1A7B5h
#1A7EDh	wait(b) |(LM) b est ici un entier binaire indiquant la durée de l'attente, exprimée en top d'horloge (ticks). Un top d'horolge est égal à 1/8192 seconde. Ce SYSEVAL, comme les deux précédents, peut être interrompu par un appui sur ON. Si une autre touche est actionnée pendant l'attente, elle est placée dans le buffer du clavier.
#40EC4h	eval;wait! |Evalue l'objet placé au niveau 1, puis éxécute wait!
#40EE7h	wait! |(LM). Attente de environ 3.5/100 secondes
#40F02h	wait!! |(LM). Attente de environ 4/10 secondes.
#40F12h	wait!!! |(LM). Attente de environ 3.4 secondes.

SYSEVALs DE CONTROLE

Voici maintenant trois façons d'éteindre la calculatrice...

Adresse	Mnémoniques et commentaires (LM si langage machine)
#041A7h	off |Forme interne de l'instruction OFF. N'éteint pas | la calculatrice si une alarme est due.
#041D4h	off! |Eteint la machine. Rallumage par ON. |Ne teste pas si une alarme est due. |Appelle "coma?" et détruit le booléen qui en résulte
#041EDh	coma? |(LM) Plonge la HP48 dans un état de veille. |Appelé par "off", mais ne teste pas si une alarme est | due. L'horloge est maintenue en fonctionnement. |Le redémarrage se fait par ON, et un booléen sur la | pile vaut True si une erreur "Invalid Card Data" | est diagnostiquée (une telle erreur se produit | réellement avec "off" en #041A7h)

Si vous n'avez pas désactivé le *buzzer* de votre HP48, vous pouvez la faire "*bipper*" de plusieurs manières:

Adresse	Mnémoniques et commentaires (LM si langage machine)
#141E5h	bip |(LM) C'est le bip signalant une erreur.
#1415Ah	beep(2r) |Forme interne de BEEP, avec deux réels.
#141B2h	beep(2s) |(LM) Fait sonner le buzzer, pendant une durée égale à | s2 (exprimé en millisecondes) et avec une fréquence | égale à s1 (exprimée en Hertz)
#3FDD1h	biiip |C'est le bip signalant l'appui sur une une touche | incorrecte (par exemple une touche de menu vide) |Exécute <46h> <151h> beep(2s), puis gèle l'écran.

Il existe des instructions qui ne font rien. La preuve....

Quand à **WSLOG**, c'est une instruction standard de la HP48, mais qui n'est pas documentée dans les manuels de l'utilisateur.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#06E8Eh	nop |(LM) Instruction vide.
#0D18Ah	lastint(s) |(LM) Fournit une chaîne décrivant les circonstances | de la s-ième dernière interruption de la HP48. |Ici s est compris entre <1h> et <5h>
#0D2A3h	wslog |Forme interne de l'instruction WSLOG. On obtient | les chaînes décrivant les 4 dernières interruptions
#14E1Dh	«nop» |Rpl se réduisant à une instruction "nop"

Venons en maintenant aux SYSEVALs qui représentent les formes internes des instructions **HALT**, **KILL**, **SST**, et **CONT**.

Il s'agit donc d'introduire une marque d'arrêt dans un programme, pour être en mesure d'éxécuter ensuite ce programme "*pas à pas*".

Comme on le voit avec plusieurs des SYSEVALs ci-dessous, il n'est pas toujours permis d'inclure de tels points d'arrêt.

La HP48 propose même un moyen radical d'interdire l'instruction **HALT**, à savoir l'instruction "***NoHalt!***" à l'adresse **#38D9Bh**. Une telle interdiction ne peut être levée que par un appel à l'instruction "***OkHalt!***" à l'adresse **#38D8Ah**.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#10FD6h	kill |(LM) Forme interne de l'instruction KILL |On abandonne ainsi l'éxécution de tous les programmes | qui pourraient être suspendus.
#11076h	cont |(LM) Forme interne de l'instruction CONT |Termine l'éxécution du dernier programme suspendu.
#14378h	halt |Forme interne de l'instruction HALT. Teste d'abord | si cette instruction est autorisée (cf ci-dessous) |Crée une variable locale 'halt (dans laquelle est | placé un booléen indiquant si la fonction | LastStack est active)
#1446Fh	HaltAllowed? |Répond True si l'instruction "halt" est permise. |Elle n'est pas permise dans les cas suivants: | * Une variable locale porte le nom 'nohalt | * Un appel en #38D79h donne le résultat False | (voir ci-dessous)
#1506Bh	Halted? |Vérifie si la variable locale 'halt existe.
#38D9Bh	NoHalt! |(LM) Interdit l'utilisation de l'instruction HALT |L'instruction ci-dessous peut seul lever cette | interdiction.
#38D8Ah	OkHalt! |(LM) Autorise l'utilisation de l'instruction HALT
#38D79h	OkHalt? |(LM) Teste si l'instruction HALT est autorisée (voir | les deux adresses ci-dessous)
#144ACh	next |Décrit l'opération suivante, en mode "halté" |C'est l'équivalent de la touche NEXT du menu PRG/CTRL
#144DEh	sstDown |Exécute l'instruction suivante, à la manière de | la touche SST| du menu PRG CTRL |On peut donc ainsi "tracer" l'instruction suivante | si elle est le nom global d'un programme.
#14506h	sst |Exécute l'instruction suivante, à la manière de | la touche SST du menu PRG CTRL

SYSEVALs DE CONTROLE

Voici enfin les différentes formes internes de **DEBUG** (qui n'est pourtant pas une instruction programmable de la HP48, mais seulement une entrée du menu **PRG\CTRL**).

On constate qu'il est apparemment possible de "*tracer*" des programmes de la ROM de la HP48.

Ne vous faites cependant pas trop d'illusions: de nombreux programmes de la ROM utilisent des *structures de contrôle* (structures conditionnelles ou répétitives), qui sont d'une toute autre nature que les instructions usuelles, et qui ne sont pas reconnues par le SYSEVAL "***debug(rpl)!***" (ce qui, au mieux, se traduit par un message d'erreur).

Adresse	Mnémoniques et commentaires (LM si langage machine)
#1508Eh	debug(_) |Lance l'éxécution, en mode "halté" du programme | placé au niveau 1 , ou contenu dans le nom placé | au niveau 1.
#150C0h	debug(rpl) |Lance l'éxécution du programme placé au niveau 1, en | mode halté. Il y a une erreur "Bad Argument Type" | si ce programme est basé en ROM |Appelé par "debug(_)" à l'adresse #1508Eh
#150E3h	debug(rpl)! |Comme ci-dessus, mais sans interdire l'éxécution | "pas à pas" des programmes basés en ROM. |Appelé par "debug(rpl)" en #150C0h.
#1512Eh	debug(nm) |Lance l'éxécution, en mode "halté" du programme | ou contenu dans le nom (global ou local) placé au | niveau 1. Il y a une erreur "Undefined Name" si | ce nom est inconnu.

LA GESTION DES ERREURS

La HP48 met à la disposition de l'utilisateur (dans le menu **PRG\CTRL**), un certain nombre d'instructions lui permettant de provoquer des *messages d'erreurs* (**DOERR**), ou de rappeler les caractéristiques de la dernière erreur à s'être produite (**ERRN** pour le *numéro* de cette erreur, et **ERRM** pour le *message* qui l'accompagne).

L'instruction **ERR0** réinitialise les données retournées par **ERRN** (qui renvoie alors #0h) et **ERRM** (qui renvoie alors une chaîne vide).

On va voir que, de façon interne, les choses sont légèrement différentes.

Tous d'abord, il existe un grand nombre d'adresses dont l'effet est d'éxécuter une erreur donnée. Ces routines, très courtes, sont écrites en langage machine. Elles permettent ainsi d'obtenir un code plus compact et plus rapide (on n'a pas à préciser le numéro de l'erreur avant de l'éxécuter).

D'autre part, on va voir qu'il est possible de *présélectionner* une erreur standard (par son numéro) ou une erreur *personnalisée* (par un message, le numéro de l'erreur personnalisée étant par convention **<70000h>**), cette *présélection* ne débouchant pas automatiquement sur l'*éxécution* de l'erreur.

Ce n'est qu'après cette présélection que peut intervenir l'ordre d'éxécuter effectivement l'erreur.

Voici donc les différentes adresses permettant de sélectionner une erreur, de l'éxécuter, ou de rappeler les caractéristiques de la dernière erreur à s'être produite.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#04CE6h	rclerr |Rappelle le numéro de l'erreur sélectionnée, sous | la forme d'un entier-système.
#04D0Eh	errsto(s) |(LM) Sélectionne l'erreur numéro s. Si s=<70000h> | il s'agit de l'erreur personnalisée (prévoir le | message en utilisant "uerrm(st)".
#04D33h	clerr |(LM) Annule la sélection d'une erreur. L'erreur | sélectionnée a pour code 0 (erreur sans message | se limitant à l'interruption du programme)
#04D64h	message(s) |Rappelle au niveau 1 le message de l'erreur qui | est présélectionnée. Si le code de cette erreur | est <70000h>, il s'agit du message personnalisé | créé au moyen de "uerrm(st)" |Si ce code est égal à <0h>, ou ne correspond à rien | de connu (en particulier dans les librairies), | alors la chaîne obtenue est vide.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#04D87h	message(s)!
	|Comme ci-dessus, mais ne teste pas si s=<70000h>
	|Ne gère donc pas l'erreur personnalisée.
#04E07h	drop;uerrm
	|(LM) Détruit l'objet du niveau 1, puis renvoie la
	| chaîne égale au message d'erreur personnalisé.
	|Renvoie "" si aucun message d'erreur personnalisé
	| n'a été sélectionné par "uerrm(st)"
#04E37h	uerrm(st)
	|(LM) Stocke la chaîne comme le message d'erreur
	| personnalisé. Cette erreur pourra être générée
	| en sélectionnant d'abord <70000h> comme code de
	| l'erreur (par "errsto(s)"), avant d'éxécuter
	| cette erreur par "error!"
#04EA4h	abort |Exécute "clerr", puis "error!".
#04ED1h	error!
	|(LM). Exécute l'erreur présélectionnée.
	|Si le code de cette erreur est <70000h>, alors le
	| message d'erreur est celui qui a été sélectionné
	| par l'instruction "uerrm(st)"
#1400Eh	err0
	|(LM) Forme interne de ERR0 (met à zéro le code de
	| la dernière erreur éxécutée).
#14039h	errn=>s
	|(LM) Forme interne de ERRN. Appelé par "errn=>b".
	|Le code de la dernière erreur éxécutée est donc
	| renvoyé sous forme d'un entier-système.
#1404Ch	errn=>b
	|Forme interne de l'instruction ERRN. Renvoie le code
	| de la dernière erreur éxécutée, sous la forme d'un
	| entier-binaire.
#14065h	errm
	|Forme interne de l'instruction ERRM.
	|Appelle "errn=>s", puis renvoie la chaîne ""
	| si le code de la dernière erreur est nul,
	| et la chaîne donnée par "message(s)" sinon.
#15007h	doerr(r) |Forme interne de DOERR, pour un réel
#1501Bh	doerr(b) |Forme interne de DOERR, pour un binaire
#1502Fh	doerr(s)
	|Forme interne de DOERR, pour un entier-système.
	|Appelé par "doerr(r)" et "doerr(b)"
	|Appelle "errsto(s)", puis "error!" (cf ci-dessus)
#15048h	doerr(st)
	|Forme interne de DOERR, pour une chaîne.
	|Fait de la chaîne le message d'erreur sélectionnée,
	| puis éxécute "error" après fait de l'entier
	| <70000h> le code de l'erreur sélectionnée.
#63155h	';error
	|Place l'adresse de la routine "error!" sur la pile,
	| sans l'évaluer.
#63169h	?:error
	|Attend un booléen au niveau 1. Si ce booléen est
	| égal à True, alors éxécute "error!"
#6383Ah	doerr(s) |Analogue à "doerr(s)" en #1502Fh.

Voici les différents SYSEVALs éxécutant une erreur donnée. Je les ai désignés par des mnémoniques utilisant leur numéro (écrit en hexadécimal), mais on peut bien sûr envisager des mnémoniques plus "*parlants*" pour les erreurs les plus classiques (par exemple **IS!** pour l'erreur "*Invalid Syntax*", ou **BAT!** pour "*Bad Argument Type*").

Adresse	Mnémonique	Message d'erreur
#04FAAh	doerr006	"Power Lost"
#04FB6h	doerr001	"Insufficient Memory"
#04FC2h	doerr002	"Directory Recursion"
#04FCEh	doerr003	"Undefined Local Name"
#04FDAh	doerr008	"Invalid Card Data"
#04FE6h	doerr009	"Object In Use"
#04FF2h	doerr00A	"Port Not Available"
#04FFEh	doerr00B	"No Room in Port"
#0500Ah	doerr00C	"Object Not in Port"
#05016h	doerr004	"Undefined XLIB Name"
#0CBAEh	doerrD04	"Nonexistent Alarm"
#10EEAh	doerrB01	"Invalid Unit"
#10EFAh	doerrB02	"Inconsistent Units"
#10F54h	doerr102	"Can't Edit Null Char."
#10F64h	doerr103	"Invalid User Function"
#10F74h	doerr104	"No Current Equation"
#10F86h	doerr106	"Invalid Syntax"
#10F96h	doerr12E	"Invalid PPAR"
#10FA6h	doerr12F	"Non-Real Result"
#10FB6h	doerr130	"Unable to Isolate"
#10FC6h	doerr126	"Halt Not Allowed"
#10FE6h	doerr124	"LAST STACK Disabled"
#10FF6h	doerr125	"LAST CMD Disabled"
#11006h	doerr128	"Wrong Argument Count"
#11016h	doerr129	"Circular Reference"
#11026h	doerr12A	"Directory Not Allowed"
#11036h	doerr12B	"Non-Empty Directory"
#11046h	doerr12C	"Invalid Definition"
#11056h	doerr12D	"Missing Library"
#11066h	doerr13C	"Name Conflict"
#18C92h	doerr204	"Undefined Name"
#18CA2h	doerr203	"Bad Argument Value"
#18CB2h	doerr202	"Bad Argument Type"
#18CC2h	doerr201	"Two Few Arguments".
#28AE2h	doerrB01	"Invalid Unit"
#29DCCh	doerr301	"Positive Underflow"
#29DDCh	doerr302	"Negative Underflow"
#29DECh	doerr303	"Overflow"
#29DFCh	doerr304	"Undefined Result"
#29E0Ch	doerr305	"Infinite Result"
#2D2E6h	doerr602	"Nonexistent ΣDAT"
#2D2F6h	doerr603	"Insufficient Σ Data"
#2D306h	doerr604	"Invalid ΣPAR"
#2D316h	doerr605	"Invalid Σ Data LN(Neg)"
#2D326h	doerr606	"Invalid Σ Data LN(0)"
#2EC34h	doerrC12	"Invalid IOPAR"
#37DE2h	doerr501	"Invalid Dimension"
#4C86Eh	doerr607	"Invalid EQ"

LA GESTION DES ERREURS

Plutôt que d'éxécuter une erreur (ce qui interromp le programme en cours) il peut être préférable d'adresser à l'utilisateur un message lui signalant le problème (voir certains messages dans l'environnement **GRAPH**, ou dans **EQUATION-WRITER**).

C'est ce que font les adresses suivantes (Les instructions "***Warning***" affichent un message, alors que les instructions "***Message***" ne font que le placer sur la pile).

Les modalités des instructions "*Warning*" sont parfois différentes (présence ou non d'un "*bip*", écran *gelé* ou non, *durée* de l'affichage du message, *ligne* où il est affiché, etc...).

Je n'ai pas fait figurer ces différences. A vous de voir les effets ainsi obtenus.

Adresse	Mnémoniques	Contenu du message
#0426Ah	Warning008	"Warning : Invalid Card Data"
#38638h	Warning005	"Memory clear"
#43671h	WarningC15	"Empty Stack"
#478BFh	Warning60A	"Enter eqn, press NEW"
#47D26h	Warning60D	"Empty catalog"
#497D0h	Warning620	"Invalid PTYPE"
#4A627h	Warning612&613	"No current data....,press Σ+"
#4A64Ah	Warning614	"Select a model"
#4A716h	Warning62E	"Enter matrix, then NEW"
#4ECF8h	Warning201	"Too Few Arguments"
#4ED20h	Warning202	"Bad Argument Type"
#15410h	Message60E	"undefined"
#31827h	MessageC15	"Empty Stack"
#474BCh	Message60F	"No stat data to plot"
#474E9h	Message61D	"Plot type: ....."
#47557h	Message608	"Current equation:"
#475D4h	Message609;false	"No current equation.", false

Dans le même ordre d'idée, voici trois adresses permettant de créer ou de "*manipuler*" des messages d'erreur. Le numéro de message est le numéro de l'erreur qui lui correspond.

Adresse	Mnémoniques et commentaires
#15D79h	AddMess(st,s) |Ajoute, devant la chaine st, le message n°s
#3889Fh	MakeMessage(proc,s) |Crée les deux chaînes utilisées pour afficher une erreur, et les place aux niveaux 1 et 2 |L'objet du niveau 2 doit être la procédure supposée produire l'erreur, et l'entier-système "s" indique le code de l'erreur. |Ex: 2:SIN 1:<3h> => 2:"Undefined Local Name" | 1:"SIN Error:"
#673A3h	Warning(s) |Emet un bip, puis affiche le message n°s pendant une seconde, avant de geler l'écran.

Quand on allume sa HP48, il peut arriver qu'un message soit affiché, dans le genre "***Warning: Low Bat(s)***", ou "***Warning: Alarm***", indiquant une insuffisance de l'état des piles (de la calculatrice ou d'une carte RAM), ou le fait qu'une alarme s'est déclenchée...

Voici les trois SYSEVALs qui se rapportent à ces messages.

Adresse	Mnémoniques et commentaires (LM si langage machine)
#04546h	alarmbat |(LM) Donne un résultat indiquant si une alarme est due, ou si certaines piles sont jugées faibles (piles internes, ou du port1, ou du port2) |Donne un entier-système égal à a+b+c+d, avec | a=1 si "alarm past due", 0 sinon | b=2 si lowbats(S), 0 sinon | c=4 si lowbats(P1), 0 sinon | d=8 si lowbats(P2), 0 sinon
#04577h	alarmbat(s) |(LM) Renvoie au niveau 1 le message qui correspond à la situation indiquée par l'entier-système s de "alarmbat", en #04546h
#386D8h	WarningAlarmBats |Affiche le message généré par "alarmbat(s)", si ce message indique réellement un problème.

Voici quelques adresses mineures, se terminant par une erreur, mais éxécutant préalablement telle ou telle opération.

Adresse	Mnémoniques et commentaires
#0F407h	4dropn;IU! |Exécute 4 Dropn, puis l'erreur "Inconsistent Units"
#1563Ah	Lpop;error(s) |Purge les dernières variables locales créées, puis éxécute l'erreur de numéro s.
#15814h	cf(-3);error |Désarme l'indicateur -3, puis éxécute l'erreur déjà sélectionnée.
#15828h	sf(-3);error |Arme l'indicateur -3, puis éxécute l'erreur déjà sélectionnée.
#457CAh	drop;doerr502 |Effectue l'instruction DROP, puis éxécute l'erreur "Invalid Array Element"
#524DEh	drop;doerr304 |Effectue l'instruction DROP, puis éxécute l'erreur "Undefined Result"
#54CC7h	drop(2);BAT! |Efface l'objet du niveau 2, puis éxécute l'erreur "Bad Argument Type"

Pour terminer, voici des SYSEVALS éxécutant (ou n'éxécutant pas) une erreur, selon la réponse apportée à un test préalable.

Adresse	Mnémoniques et commentaires
#0908Fh	dup;=7(s)?:BAT! |Erreur "Bad Argument Type" si l'objet du niveau 1 est | égal à l'entier-système <7h>.
#63B2Dh	dup;r?#:BAT! |Erreur "Bad Argument Type" si l'objet du niveau 1 | n'est pas réel
#36417h	dup;=0?:UR! |Erreur "Undefined Result" si l'objet (qui peut être | un réel ou un complexe de type normal ou long) | est nul.
#193C1h	dup;ar?#:BAT! |Erreur "Bad Argument Type" si l'objet du niveau 1 | n'est pas un tableau
#194BBh	dup;ra?#:BAT! |Erreur "Bad Argument Type" si l'objet du niveau 1 | n'est pas un tableau réel.
#194D9h	dup;ca?#:BAT! |Erreur "Bad Argument Type" si l'objet du niveau 1 | n'est pas un tableau complexe.
#19443h	dup;?#li=>BAT! |Erreur "Bad Argument Type" si l'objet du niveau 1 | n'est pas une liste
#1592Dh	depth=0?:TFA! |Si la pile vide, erreur "Two Few Arguments"
#37DF6h	?#:doerr501 |Si le booléen du niveau 1 est égal à False, alors | éxécute l'erreur "Invalid Dimension"
#51085h	?#:doerr12E |Si le booléen du niveau 1 est égal à False, alors | éxécute l'erreur "Invalid PPAR"
#63B05h	?:BAV! |Si le booléen du niveau 1 est égal à True, alors | éxécute l'erreur "Bad Argument Value"
#63B19h	?#:BAV! |Si le booléen du niveau 1 est égal à False, alors | éxécute l'erreur "Bad Argument Value"
#63B46h	?#:BAT! |Si le booléen du niveau 1 est égal à False, alors | éxécute l'erreur "Bad Argument Type"

INSTRUCTIONS STANDARDS

Ce livre a été en grande partie consacré à la description d'adresses (de "SYSEVALs") plutôt secrètes.

Il s'est agi, le plus souvent, de formes internes plus ou moins sophistiquées des instructions standards de la HP48, ou d'adresses offrant des possibilités bien supérieures à celles qu'autorisent ces instructions.

Il ne faut pas oublier, pour autant, que les éléments de base du langage de la HP48 sont implantés quelque part en mémoire. Ce sont ces adresses qui feront l'essentiel de ce chapitre, accompagnées éventuellement de commentaires (notamment pour décrire la manière dont sont codées, en mémoire, les *instructions de boucles* ou les *instructions conditionnelles*).

◆ LA COMPILATION:

Quand on édite une ligne de commande, on peut entrer le nom des instructions en toutes lettres, ou bien utiliser les touches des menus intégrés. Dans ce dernier cas, tout dépend du mode de saisie en cours (mode d'***éxécution directe***, ou **PRG**, ou **ALG**, ou **ALG PRG**).

En mode **PRG**, par exemple, le nom de l'instruction sélectionnée vient se placer à la position du curseur. Seule la touche **ENTER** et la touche **ON** peuvent interrompre la saisie (la première pour évaluer la ligne de commande, la seconde pour l'annuler).

En mode d'*éxécution directe*, la plupart des touches correspondant à des instructions standards évaluent la ligne de commande puis évaluent la routine associée à cette touche.

Dans tous les cas, l'évaluation de la ligne de commande est une opération essentielle assurée par le module de *compilation* de la HP48. Il s'agit ici d'isoler et d'analyser les mots dont est formée la ligne de commande.

Quand un mot est isolé (et qu'il commence par un caractère autre qu'un chiffre ou un délimiteur), la HP48 examine s'il correspond à une *variable* connue (*locale*, puis *globale*), puis, sinon, s'il est une *commande* ou une *fonction* d'une *librairie* (d'abord les librairies utilisateurs, puis les librairies intégrées).

Pour examiner si un mot de la ligne de commande correspond à une des instructions des librairies n°2 et n°1792 (les deux librairies intégrées), la HP48 parcourt la "*Hash-Table*" de ces librairies (voir le chapitre consacré aux *objets-librairies* dans la première partie de ce livre).

Si le nom est trouvé, il est suivi par le ***numéro XLIB*** qui lui correspond. Muni de ce renseignement, la calculatrice trouve, dans la "*Link-Table*" de la librairie, l'adresse de cette instruction en mémoire.

Quand vous créez un programme en ligne de commande, la HP48 traduit ainsi les *mots* représentant des instructions standards en les *adresses* désignant ces instructions.

◆ LA DÉCOMPILATION:

Inversement, quand le programme que vous venez de créer en ligne de commande a été ainsi compilé, il est placé sur la pile, de façon "*lisible*" (le fait d'afficher les *adresses* des instructions plutôt que leurs *noms* aurait un effet plutôt désagréable).

Il y a donc immédiatement un travail de *décompilation* qui s'effectue.
La HP48 sait, à partir de l'adresse d'une de ses instructions standards, retrouver le nom de cette instruction (c'est indispensable pour assurer un affichage décent sur la pile, et pour permettre une édition efficace, en ligne de commande, des programmes existants).

Pour celà, elle utilise l'"*astuce*" suivante: Les 6 quartets qui précèdent cette adresse donnent le numéro de librairie et le numéro d'objet dans cette librairie, qui correspondent à l'instruction considérée.
Avec ces renseignements, elle va dans la "*Hash-Table*" de la librairie trouver le nom "*lisible*" de l'instruction...

◆ UN MOT = UNE ADRESSE ?

Considérons le programme « **SIN SWAP COS * »**.

En mémoire, il est codé de la manière suivante:

D9D20	#02D9Dh est l'adresse de début des Rpl.
E1632	#2361Eh est l'adresse de «
CA4B1	#1B4ACh est l'adresse de SIN
DBBF1	#1FBBDh est l'adresse de SWAP
505B1	#1B505h est l'adresse de COS
EEDA1	#1ADEEh est l'adresse de *
93632	#23639h est l'adresse de »
B2130	#0312Bh est l'adresse de fin des Rpl

L'affaire semble donc entendue: une instruction standard (écrite sous forme littérale) correspond de manière biunivoque à une adresse de la HP48. Les choses ne sont cependant pas si simples....
Si la donnée d'une adresse conduit, sans ambiguité possible, à un mot du vocabulaire de la HP48, l'opération inverse s'effectue parfois en fonction du "*contexte général de la phrase*" (c'est le travail du compilateur).

Voici deux exemples (il y en a quelques autres) d'interprétations différentes (en termes d'*adresses*) d'un même *mot* de la calculatrice:

- Certaines instructions de la HP48, autorisées dans les expressions algébriques, ont un comportement différent selon qu'elles sont utilisées algébriquement ou en notation polonaise inverse.

- C'est le cas par exemple pour l'instruction **XROOT**:
  En notation algébrique, '**XROOT(x,y)**' a le sens qu'a la séquence **y x XROOT** en notation RPL (ce qui constitue une dérogation à l'ordre des arguments d'une fonction).
  Cette particularité implique que le mot **XROOT** sera traduit en deux adresses différentes suivant le contexte.

- Certaines instructions, comme **THEN** ou **END**, ne se conçoivent qu'associées à une autre instruction (comme **IF**, **CASE**, etc...).
  Le **END** d'une séquence **IF..END**, ne sera pas logé à la même adresse que le **END** d'une séquence **WHILE..DO..END**.
  Là encore, le *compilateur* tranche en fonction du contexte.

Voici maintenant la liste des points d'entrée standards de la HP48, triés par ordre croissant d'adresses.

Les instructions dont les noms sont en caractères gras font l'objet de commentaires particuliers à la fin de l'énumération.

#1957Bh	ASR	#1A3BEh	EVAL	#1B8A2h	ATANH
#1959Bh	RL	#1A3FEh	IFTE	#1B905h	EXP
#195BBh	RLB	#1A4CDh	IFT	#1B94Fh	LN
#195DBh	RR	#1A52Eh	SYSEVAL	#1B9C6h	LOG
#195FBh	RRB	#1A584h	DISP	#1BA3Dh	ALOG
#1961Bh	SL	#1A5A4h	FREEZE	#1BA8Ch	LNP1
#1963Bh	SLB	#1A5C4h	BEEP	#1BAC2h	EXPM
#1965Bh	SR	#1A5E4h	→NUM	#1BB02h	!
#1967Bh	SRB	#1A604h	LAST	#1BB41h	FACT()
#1969Bh	R→B	#1A71Fh	WAIT	#1BB6Dh	IP
#196BBh	B→R	#1A858h	CLLCD	#1BBA3h	FP
#196DBh	CONVERT	#1A873h	KEY	#1BBD9h	FLOOR
#1971Bh	UVAL	#1A8BBh	CONT	#1BC0Fh	CEIL
#1974Fh	→UNIT	#1A8D8h	=	#1BC45h	XPON
#19771h	UBASE	#1A995h	NEG	#1BC71h	MAX
#197A5h	UFACT	#1AA1Fh	ABS	#1BCE3h	MIN
#197F7h	TIME	#1AA6Eh	CONJ	#1BD55h	RND
#19812h	DATE	#1AABDh	π	#1BDD1h	TRNC
#1982Dh	TICKS	#1AADFh	MAXR	#1BE4Dh	MOD
#19848h	WSLOG	#1AB01h	MINR	#1BE9Ch	MANT
#19863h	ACKALL	#1AB23h	e	#1BEC8h	D→R
#1987Eh	ACK	#1AB45h	i	#1BEF4h	R→D
#1989Eh	→DATE	#1AB67h	+	#1BF1Eh	→HMS
#198BEh	→TIME	#1ACDDh	+	#1BF3Eh	HMS→
#198DEh	CLKADJ	#1AD09h	-	#1BF5Eh	HMS+
#198FEh	STOALARM	#1ADEEh	*	#1BF7Eh	HMS-
#19928h	RCLALARM	#1AF05h	/	#1BF9Eh	RNRM
#19948h	FINDALARM	#1B02Dh	^	#1BFBEh	CNRM
#19972h	DELALARM	#1B185h	XROOT	#1BFDEh	DET
#19992h	TSTR	#1B1CAh	XROOT()	#1BFFEh	DOT
#199B2h	DDAYS	#1B278h	INV	#1C01Eh	CROSS
#199D2h	DATE+	#1B2DBh	ARG	#1C03Eh	RSD
#1A105h	CRDIR	#1B32Ah	SIGN	#1C060h	%
#1A125h	PATH	#1B374h	√	#1C0D7h	%T
#1A140h	HOME	#1B426h	SQ	#1C149h	%CH
#1A15Bh	UPDIR	#1B4ACh	SIN	#1C1B9h	RAND
#1A194h	VARS	#1B505h	COS	#1C1D4h	RDZ
#1A1AFh	TVARS	#1B55Eh	TAN	#1C1F6h	COMB
#1A1D9h	BYTES	#1B5B7h	SINH	#1C236h	PERM
#1A2BCh	NEWOB	#1B606h	COSH	#1C274h	SF
#1A303h	KILL	#1B655h	TANH	#1C2D5h	CF
#1A31Eh	OFF	#1B6A4h	ASIN	#1C313h	FS?
#1A339h	DOERR	#1B72Fh	ACOS	#1C360h	FC?
#1A36Dh	ERR0	#1B79Ch	ATAN	#1C399h	DEG
#1A388h	ERRN	#1B7EBh	ASINH	#1C3B4h	RAD
#1A3A3h	ERRM	#1B830h	ACOSH	#1C3CFh	GRAD

#1C3EAh	FIX	#1E07Eh	PMIN	#1E809h	OR
#1C41Eh	SCI	#1E09Eh	PMAX	#1E88Fh	NOT
#1C452h	ENG	#1E0BEh	AXES	#1E8F6h	XOR
#1C486h	STD	#1E0E8h	CENTR	#1E972h	==
#1C4A1h	FS?C	#1E126h	RES	#1EA9Dh	<>
#1C520h	FC?C	#1E150h	*H	#1EBBEh	<
#1C559h	BIN	#1E170h	*W	#1EC5Dh	>
#1C574h	DEC	#1E190h	DRAW	#1ECFCh	≤
#1C58Fh	HEX	#1E1ABh	AUTO	#1ED9Bh	≥
#1C5AAh	OCT	#1E1C6h	DRAX	#1EE38h	OLDPRT
#1C5C5h	STWS	#1E1E1h	SCALE	#1EE53h	PR1
#1C5FEh	RCWS	#1E201h	PDIM	#1EE6Eh	PRSTC
#1C619h	RCLF	#1E22Bh	DEPND	#1EE89h	PRST
#1C67Fh	STOF	#1E25Fh	ERASE	#1EEA4h	CR
#1C783h	→LIST	#1E27Ah	PX→C	#1EEBFh	PRVAR
#1C79Eh	R→C	#1E29Ah	C→PX	#1EF43h	DELAY
#1C7CAh	RE	#1E2BAh	GRAPH	#1EF63h	PRLCD
#1C819h	IM	#1E2D5h	LABEL	#1EF7Eh	∂
#1C85Ch	SUB	#1E2F0h	PVIEW	#1F354h	**WHERE**
#1C8EAh	REPL	#1E320h	PIXON	#1F55Dh	**APPLY**
#1C95Ah	LIST→	#1E344h	PIXOFF	#1EFD2h	∂()
#1C98Eh	C→R	#1E36Eh	PIX?	#1F133h	RCEQ
#1C9B8h	SIZE	#1E398h	LINE	#1F14Eh	STEQ
#1CAB4h	POS	#1E3C2h	TLINE	#1F16Eh	ROOT
#1CB0Bh	→STR	#1E3ECh	BOX	#1F1D4h	∫
#1CB26h	STR→	#1E416h	BLANK	#1F223h	∫ ()
#1CB46h	NUM	#1E436h	PICT	#1F2C9h	Σ
#1CB66h	CHR	#1E456h	GOR	#1F3F3h	**WHERE**()
#1CB86h	TYPE	#1E4E4h	GXOR	#1F500h	QUOTE
#1CE28h	VTYPE	#1E572h	LCD→	#1F5C5h	**APPLY**()
#1CEE3h	EQ→	#1E58Dh	→LCD	#1F996h	**XLIB 2 261**
#1CF7Bh	OBJ→	#1E5ADh	→GROB	#1F9AEh	**XLIB 2 262**
#1D009h	→ARRY	#1E5D2h	ARC	#1F9C4h	→Q
#1D092h	ARRY→	#1E606h	TEXT	#1F9E9h	→Qπ
#1D0DFh	RDM	#1E621h	XRNG	#1FA59h	MATCH↑
#1D186h	CON	#1E641h	YRNG	#1FA8Dh	MATCH↓
#1D2DCh	IDN	#1E661h	FUNCTION	#1FAEBh	_
#1D392h	TRN	#1E681h	CONIC	#1FB5Dh	RATIO
#1D407h	PUT	#1E6A1h	POLAR	#1FB87h	DUP
#1D5DFh	PUTI	#1E6C1h	PARAMETRIC	#1FBA2h	DUP2
#1D7C6h	GET	#1E6E1h	TRUTH	#1FBBDh	SWAP
#1D8C7h	GETI	#1E701h	SCATTER	#1FBD8h	DROP
#1DD06h	V→	#1E721h	HISTOGRAM	#1FBF3h	DROP2
#1DE66h	→V2	#1E741h	BAR	#1FC0Eh	ROT
#1DEC2h	→V3	#1E761h	SAME	#1FC29h	OVER
#1E04Ah	INDEP	#1E783h	AND	#1FC44h	DEPTH

INSTRUCTIONS STANDARDS

#1FC64h	DROPN	#2025Eh	BESTFIT	#21FECh	CKSM
#1FC7Fh	DUPN	#202CEh	SINV	#2200Ch	BAUD
#1FC9Ah	PICK	#2034Dh	SNEG	#2202Ch	PARITY
#1FCB5h	ROLL	#203CCh	SCONJ	#2204Ch	TRANSIO
#1FCD0h	ROLLD	#2044Bh	STO+	#2206Ch	KERRM
#1FCEBh	CLEAR	#20538h	STO-	#22087h	BUFLEN
#1FD0Bh	STOΣ	#2060Ch	STO/	#220A2h	STIME
#1FD2Bh	CLΣ	#20753h	STO*	#220C2h	SBRK
#1FD46h	RCLΣ	#208F4h	INCR	#220DDh	PKT
#1FD61h	Σ+	#209AAh	DECR	#224CAh	INPUT
#1FD8Bh	Σ-	#20A15h	COLCT	#224F4h	ASN
#1FDA6h	NΣ	#20A49h	EXPAN	#22514h	STOKEYS
#1FDC1h	CORR	#20A7Dh	**RULES**	#22548h	DELKEYS
#1FDDCh	COV	#20A93h	ISOL	#22586h	RCLKEYS
#1FDF7h	ΣX	#20AB3h	QUAD	#225BEh	→TAG
#1FE12h	ΣY	#20AD3h	SHOW	#22633h	DTAG
#1FE2Dh	ΣX^2	#20B20h	TAYLR	#22EC3h	**IF**
#1FE48h	ΣY^2	#20B40h	RCL	#22EFAh	**THEN**
#1FE63h	ΣX*Y	#20CCDh	STO	#22FB5h	**ELSE**
#1FE7Eh	MAXΣ	#20D65h	DEFINE	#22FD5h	**ENDofIF**
#1FE99h	MEAN	#20EFEh	PURGE	#22FEBh	→'
#1FEB4h	MINΣ	#20FAAh	MEM	#23033h	**WHILE**
#1FECFh	SDEV	#20FD9h	ORDER	#2305Dh	**REPEAT**
#1FEEAh	TOT	#210FCh	CLUSR	#230C3h	**DO**
#1FF05h	VAR	#2115Dh	TMENU	#230EDh	**UNTIL**
#1FF20h	LR	#21196h	MENU	#23103h	**START**
#1FF7Ah	PREDV	#211E1h	RCLMENU	#231A0h	**FOR**
#1FF9Ah	PREDY	#211FCh	PVARS	#2324Ch	**NEXT**
#1FFBAh	PREDX	#2123Ah	PGDIR	#23380h	**STEP**
#1FFDAh	XCOL	#2125Ah	ARCHIVE	#233DFh	**IFERR**
#1FFFAh	YCOL	#2133Ch	RESTORE	#23472h	HALT
#20020h	UTPC	#2137Fh	MERGE	#2349Ch	**Hidden'**
#2003Ah	UTPN	#213D1h	FREE	#234C1h	→«
#2005Ah	UTPF	#2142Dh	LIBS	#235FEh	»**Lpop**
#2007Ah	UTPT	#21448h	ATTACH	#2361Eh	«
#2009Ah	ΣCOL	#2147Ch	DETACH	#23639h	»
#200C4h	SCLΣ	#21E75h	XMIT	#23654h	**Begin'**
#200F3h	ΣLINE	#21E95h	SRECV	#23679h	**End'**
#2010Eh	BINS	#21EB5h	OPENIO	#23694h	**ENDofWHILE**
#20133h	BARPLOT	#21ED5h	CLOSEIO	#236B9h	**ENDofDO**
#20167h	HISTPLOT	#21EF0h	SEND	#2371Fh	**THENofERR**
#2018Ch	SCATRPLOT	#21F24h	KGET	#237A8h	**THENofCASE**
#201B1h	LINFIT	#21F62h	RECN	#2378Dh	**CASE**
#201D6h	LOGFIT	#21F96h	RECV	#23813h	**DIR**
#201FBh	EXPFIT	#21FB6h	FINISH	#23824h	PROMPT
#20220h	PWRFIT	#21FD1h	SERVER		

La longue liste précédente réserve quelques surprises, notamment parce qu'il apparaît quelques instructions non documentées dans les manuels qui accompagnent la HP48 (**WSLOG**, **DIR**, **RULES** ...).

Nous reviendrons plus loin sur ces instructions mystérieuses.

Dans un premier temps, nous allons examiner comment la HP48 code, de manière interne, les instructions composées que sont:

- Les séquences **IF..THEN..ELSE..END**
- Les structures **CASE..THEN..END..END**
- Les boucles **FOR** et **START**
- Les structures **DO..UNTIL..END**
- Les structures **WHILE..REPEAT..END**

◆ LA SÉQUENCE IF...THEN...ELSE...END:

Cette instruction conditionnelle est codée, symboliquement:

... IF ... THEN Objet_True (ELSE Objet_False) END

La position du **IF** initial n'a pas grande importance.

"***Objet_True***" et "***Objet_False***" sont les objets (au sens du langage RPL) qui doivent être éxécutés selon que la valeur numérique au niveau 1 de l'objet la pile, au moment où l'instruction **THEN** est éxécutée, est un réel (ou un expression dont la valeur est un réel) non nul (***True***) ou nul (***False***).

Ces 2 objets ne doivent pas être absents. Ils sont donc remplacés éventuellement par une structure Rpl vide (voir les 2 premiers exemples).

Dans le cas où "***Objet_True***" et/ou "***Objet_False***" sont composés (plusieurs objets à évaluer), ils sont codés sous la forme de structures Rpl. Voir le dernier exemple ci-desssous.

▸ Programme « IF THEN END »

D9D20 E1632	Début de Rpl et «
3CE22 AFE22	IF THEN
D9D20 B2130	Rpl vide
5DF22	END (of IF)
93632 B2310	Fin de Rpl et »

▸ Programme « IF THEN ELSE END »

D9D20 E1632	Début de Rpl et «
3CE22 AFE22	IF THEN
D9D20 B2130	Rpl vide
5BF22	ELSE
D9D20 B2130	Rpl vide
5DF22	END (of IF)
93632 B2310	Fin de Rpl et »

▸ **Programme**
« DUP2 IF SAME THEN "EGAUX" ELSE "DIFFERENTS" END »

```
D9D20 E1632                          |Début de Rpl et «
   2ABF1 3CE22 167E1                 |DUP2 IF SAME
   AFE22                             |THEN
   C2A20 F0000 5474145585            |"EGHAUX"
   5BF22                             |ELSE
   C2A20 91000 44946464542554E44535  |"DIFFERENTS"
   5DF22                             |END (of IF)
   93632                             |»
B2310                                |Fin de Rpl
```

▸ **Programme « IF SAME THEN 1 + ELSE 1 - END »**

```
D9D20 E1632                          |Début de Rpl et «
   3CE22 167E1 AFE22                 |IF SAME THEN
   D9D20 9C2A2 76BA1 B2130           |Rpl 1 +
   5BF22                             |ELSE
   D9D20 9C2A2 90DA1 B2130           |Rpl 1 -
   5DF22                             |END (of IF)
   93632                             |»
B2310                                |Fin de Rpl
```

◆ LA SÉQUENCE CASE ... THEN ... END ... END:

En mémoire, elle est codée **CASE Contenu END**, où "***Contenu***" est une structure Rpl.

"***Contenu***" est formé de la manière suivante (les adresses **D9D20**, au début, et **B2130**, à la fin, sont sous-entendues)

Event_1 THEN Réponse_1 END
Event_2 THEN Réponse_2 END
....
Event_n THEN Réponse_n END
(Sinon)

Où "***Event_1***", ... , "***Event_n***" peuvent être des séquences d'objets conduisant à une valeur réelle non nulle (***True***: la réponse appropriée est éxécutée, puis on saute après le dernier **END**) ou nulle (***False***: on passe au test de l'éventualité suivante).

"***Réponse_1***", ..., "***Réponse_n***" doivent être formées d'un seul objet (éventuellement une structure Rpl, pour englober plusieurs instructions).

L'éventualité "***Sinon***", qui est facultative, est éxécutée si aucune des "***Eventualités***" n'a été reconnue. Ce peut être une séquence d'objets.

Voici le codage de deux exemples simples:

▸ **Programme « CASE THEN END END »**

```
D9D20 E1632          |Début de Rpl et «
   D8732             |CASE
   D9D20             |Début Rpl
      8A732          |THEN (of CASE)
         D9D20 B2130 |Rpl vide
      5DF22          |End (of IF)
   B2130             |Fin Rpl
   5DF22             |End (of IF)
   93632             |»
B2130                |Fin de Rpl
```

▸ **Programme**

```
« CASE
   DUP 1 SAME THEN "1" END
   DUP 2 SAME THEN "2" END
   0 "?"
   END
»
```

```
D9D20 E1632          |Début de Rpl et «
   D873              |CASE
   D9D20             |Début du contenu du CASE
      78BF1 9C2A2 167E1 |DUP 1 SAME
      8A732          |THEN (of CASE)
      C2A20 70000 13 |"1"
      5DF22          |End (of IF)
      78BF1 ED2A2 167E1 |DUP 2 SAME
      8A732          |THEN (of CASE)
      C2A20 70000 23 |"2"
      5DF22          |End (of IF)
      4B2A2 C2A20 70000 F3 |0 "?"
   B2130             |Fin du contenu du CASE
   5DF22             |End (of IF)
   93632             |»
B2130                |Fin de Rpl
```

◆ LES BOUCLES START ET FOR:

Comme on va le voir sur des exemples, le codage des boucles **FOR** et **START** en mémoire ne récèle aucune surprise (il correspond point par point à l'image qu'en donne la HP48 à l'affichage sur la pile, ou en édition en ligne de commande).

Dans le cas d'une boucle **FOR**, la variable qui contient le compteur de boucle est créée comme variable locale, et c'est l'instruction **FOR** elle-même qui prévoit de purger cette variable locale au sortir de la boucle.

▸ **Programme « START NEXT »**

```
D9D20 E1632          |Début de Rpl et «
   30132 C4232       |START NEXT
93632 B2130          |Fin de Rpl et »
```

▸ **Programme « 1 9 START DUP 5 * NEXT »**

```
D9D20 E1632          |Début de Rpl et «
   9C2A2 173A2 30132      |1 9 START
   78BF1 D13A2 EEDA1      |DUP 5 *
   C4232                  |NEXT
93632 B2130          |Fin de Rpl et »
```

▸ **Programme « 1 9 FOR k DUP k * NEXT »**

```
D9D20 E1632          |Début de Rpl et «
   9C2A2 173A2 0A132      |1 9 FOR
   D6E20 10 B6            |k (nom local)
   78BF1 D6E20 10 B6 EEDA1     |DUP k *
   C4232                  |NEXT
93632 B2130          |Fin de Rpl et »
```

▸ **Programme « 1 9 FOR k DUP k * 2 STEP »**

C'est le même codage que précédemment, mais en remplaçant la ligne

```
      C4232          |NEXT
```

par la ligne

```
      ED2A2 08332    |2 STEP
```

▸ **Programme**

```
« 1 5 START DUP
    1 25 FOR k k * 3 STEP
  NEXT
»
```

```
D9D20 E1632          |Début de Rpl et «
   9C2A2 D13A2 30132      |1 5 START
      78BF1 9C2A2              |DUP 1
      33920 100 0000000000520|25
      0A132 D6E20 10 B6        |FOR k
         D6E20 10 B6 EEDA1     |k *
      3F2A2 08332              |3 STEP
   C4232                  |NEXT
93632 B2130          |Fin de Rpl et »
```

◆ LA STRUCTURE DO...UNTIL...END:

Comme pour les boucles **FOR** et **START**, il n'y a aucune particularité notable dans le codage interne de la structure **DO..UNTIL..END**.

Les deux exemples ci-dessous en témoignent.

▸ Programme « DO UNTIL END »

```
D9D20 E1632          |Début de Rpl et «
   3C032 DE032 9B632     |DO UNTIL END(of DO)
93632 B2130          |Fin de Rpl et »
```

▸ Programme « DO 1 + UNTIL 9 ! > END »

```
D9D20 E1632          |Début de Rpl et «
   3C032 9C2A2 76BA1     |DO 1 +
   DE032 173A2 20BB1 D5CE1    |UNTIL 9 ! >
   9B632                 |END(of DO)
93632 B2130          |Fin de Rpl et »
```

◆ LA STRUCTURE WHILE...REPEAT...END:

La seule particularité, ici, est que la *clause* à répéter, située entre **REPEAT** et **END**, doit être vue par la HP48 comme un seul objet.

Par conséquent, si cette clause contient plusieurs instructions, elle doit être formée comme une structure Rpl.

▸ Programme « WHILE REPEAT END »

```
D9D20 E1632          |Début de Rpl et «
   33032 D5032       |WHILE REPEAT
      D9D20 B2130    |Structure Rpl vide
   49632             |End(of WHILE)
93632 B2130          |Fin de Rpl et »
```

▸ Programme « WHILE 0 > REPEAT LN END »

```
D9D20 E1632          |Début de Rpl et «
   33032 4B2A2 D5CE1     |WHILE 0 >
   D5032 F49B1           |REPEAT LN
   49632                 |End(of WHILE)
93632 B2130          |Fin de Rpl et »
```

▸ Programme « WHILE 0 > REPEAT LN 1 + END »

```
D9D20 E1632          |Début de Rpl et «
   33032 4B2A2 D5CE1 D5032      |WHILE 0 > REPEAT
   D9D20             |Début structure Rpl
      F49B1 9C2A2 76BA1   |LN 1 +
   B2130             |Fin structure Rpl
   49632             |End(of WHILE)
93632 B2130          |Fin de Rpl et »
```

◆ LA STRUCTURE IFERR...THEN...(ELSE)...END:

Cette structure permet de garder le contrôle de l'éxécution d'un programme (en passant la main à une clause "***traitement***") dans le cas où une erreur surviendrait dans une clause "***risque***".

De manière interne, le codage est, symboliquement:

```
IFERR Clause_Risque
    THEN Clause_Traitement
    ELSE Clause_Sinon
END
```

Les différentes ***clauses*** doivent être vues par la HP48 comme un seul objet. Si ces clauses sont constituées de plusieurs instructions, elles doivent être formées comme des structures Rpl.

▸ Programme « IFERR THEN END »

```
D9D20 E1632          |Début de Rpl et «
   FD332             |IFERR
      D9D20 B2130    |Structure Rpl vide
      F1732          |THEN(of ERR)
      D9D20 B2130    |Structure Rpl vide
   5DF22             |END(of IF)
93632 B2130          |Fin de Rpl et »
```

▸ Programme « IFERR THEN ELSE END »

```
D9D20 E1632          |Début de Rpl et «
   FD332             |IFERR
      D9D20 B2130          |Structure Rpl vide
   F1732             |THEN(of ERR)
      D9D20 B2130          |Structure Rpl vide
   5BF22             |ELSE
      D9D20 B2130          |Structure Rpl vide
   5DF22             |END(of IF)
93632 B2130          |Fin de Rpl et »
```

▸ **Programme « IFERR LN THEN 1 + ELSE 1 - END »**

```
D9D20 E1632          |Début de Rpl et «
   FD332 F49B1       |IFERR LN
   F1732             |THEN(of ERR)
      D9D20 9C2A2 76BA1 B2130    |1 +
   5BF22             |ELSE
      D9D20 9C2A2 90DA1 B2130    |1 -
   5DF22             |END(of IF)
93632 B2130          |Fin de Rpl et »
```

Nous allons maintenant nous appliquer à éclaircir quelques mystères qui apparaissent dans la liste des instructions "*standards*" de la HP48.

Plutôt que d'instructions "*standards*" il faudrait parler d'adresses susceptibles d'apparaître dans le codage en mémoire d'un programme entré normalement en ligne de commande ("*normalement*" et donc sans utilisation de techniques "*Sysevalesques*").

◆ QUELQUES INSTRUCTIONS MYSTÉRIEUSES:

Voici quelques instructions tout à fait "*légales*", mais qui ne sont pas documentées dans les manuels de l'utilisateur de la HP48:

J'ai donné à chaque fois le numéro d'objet dans la librairie à laquelle cet objet appartient (une des deux librairies intégrées, de numéro 2 ou #700h=1792).

Pour placer sur la pile un tel objet, à partir des deux entiers **L** et **n** (**L**=numéro de librairie au niveau 2, **n**=numéro d'objet au niveau 1) utilisez le programme suivant:

« 1 2 START #18CD7h SYSEVAL SWAP NEXT #7E50h SYSEVAL »

▸ La commande **WSLOG**: (adresse **#19848h**)
C'est l'objet n°19 de la librairie n°2.
Elle donne, sous forme de chaînes, les circonstances des 4 derniers *redémarrages* de la HP48.
Cette instruction a été évoquée dans un des chapitres précédents.

▸ La commande **DIR**: (adresse **#23813h**)
C'est l'objet n°27 de la librairie n°1792 (1792=#700h)
Elle place un *objet-répertoire vide* au niveau 1 de la pile.
Cette instruction ne peut qu'être éxécutée en ligne de commande.
Il est impossible de la placer dans un programme.
Assez bizarrement, l'adresse **#23813h** ne contient qu'une structure Rpl vide...

- La commande **RULES**: (adresse **#20A7Dh**):
  C'est l'objet n°335 de la librairie n°2.
  Elle émet un *bip* (semblable à celui qu'on obtient en actionnant une touche de menu vide) puis *gèle* l'écran.

- La commande +: (adresse **#1ACDDh**)
  C'est l'objet n°69 dans librairie n°2.
  Elle ne fait rien...Elle est représentée sur la pile par le signe + mais il ne s'agit pas de l'addition...
  Peut être doit-on considérer qu'il s'agit du + "**unaire**" (comme il existe un − unaire (synonyme de **NEG**).

- Les objets **XLIB 2 261**, et XLIB **2 262**:
  Leurs adresses respectives sont: **#1F996h** et **#1F9AEh**.
  Aucun nom "*lisible*" ne leur correspond.
  Les séquences **#2845Ah SYSEVAL** et **#2846Eh SYSEVAL** permettent de placer sur la pile les routines qui leur correspondent. Elles sont "*invisibles*" (mais elles sont bien sur la pile).
  Leur rôle est celui d'un *marqueur*, permettant de savoir si, dans une expression, un *argument complexe* doit être interprété sous forme *cartésienne* (**XLIB 2 261**) ou sous forme *polaire*.

- L'instruction " _ ":
  C'est le mot "*lisible*" qui correspond au nom **XLIB 2 267**.
  L'adresse de la routine qui lui correspond est **#1FAEBh**.
  Elle apparaît de manière implicite dans les expressions contenant des objets-unités. Mais son rôle demeure assez mystérieux.

- Les instructions → (création de variables locales):
  Il y a deux instructions différentes, dont le rôle est de créer des variables locales, suivant que cette création précède une *expression algébrique*, ou un *programme* entre « et ».
  Je les ai notées →' (adresse **#22FEBh**) et →« (adresse #234C1h), bien qu'elles soient toutes les deux représentées, par la HP48, comme un unique caractère "*flèche à droite*".
  Leurs ***noms XLIB*** sont: **XLIB 1792 4** pour →', et **XLIB 1792 16** pour →«.

  Exemple: considérons les deux programmes suivants, qui font la même chose, pour comparer la façon dont ils sont codés en mémoire.
  'P1': « → a b c « a b + c * » »
  'P2': « → a b c '(a+b)*c' »

Le codage de '**P1**' est:

```
D9D20  E1632          |Début Rpl et «
   1C432              |→ (devant «)
   D6E20 10 16  D6E20 10 26  D6E20 10 36      |a b c (locaux)
   E1632                                      |«
      D6E20 10 16  D6E20 10 26  76BA1 |a b +
      D6E20 10 36  EEDA1                      |c *
   EF532              |»Lpop
93632  B2130          |» et Fin Rpl
```

(L'instruction »**Lpop** est décrite plus loin. Son rôle est en particulier de *purger* les variables locales **a**,**b**,**c**).

Le codage de '**P2**' est:

```
D9D20  E1632          |Début Rpl et «
   BEF22              |→ (devant ')
   D6E20 10 16  D6E20 10 26  D6E20 10 36      |a b c (locaux)
   8BA20                                      |Début Expression
      D6E20 10 16  D6E20 10 26  76BA1 |a b +
      D6E20 10 36  EEDA1                      |c *
   B2130                                      |Fin d'expression
93632  B2130          |» et Fin Rpl
```

(L'instruction → devant l'expression contient un mécanisme permettant de purger les variables locales **a**,**b**,**c** quand l'expression '**(a+b)*c**' a été évaluée).

- Les instructions « et »:

Il y a une seule instruction « (adresse **#2361Eh**; **XLIB 1792 18**). Il y par contre deux instructions »**:**

La première: adresse **#23639h**; **XLIB 1792 19**.

La seconde: adresse **#235FEh**; **XLIB 1792 17**.

Comme on vient de le voir, cette dernière est utilisée pour clore les structures locales « » (afin de purger les variables locales qui y ont été créées).

D'une manière générale, ces instructions se limitent à tester si la touche **ON** a été actionnée, afin d'interrompre le programme en cours.

Leur intérêt est par contre essentiel en ligne de commande, car elles permettent de grouper plusieurs objets dans une même structure, sans les évaluer individuellement.

- Les délimiteurs ':

Quand vous éditez un programme en ligne de commande et que vous y placez une expression algébrique, vous incluez cette expression entre deux délimiteurs '. Il en va de même quand vous placez un nom dans ce programme (nom que vous ne voulez pas voir évalué).

Dans la phase de *compilation*, la HP48 examine l'objet que vous avez placé entre deux délimiteurs '.
Si cet objet est une expression, elle en en génère une image Rpl, et crée un *objet-expression* (adresse préfixe **#8BA20h**).
Inversement, lors de l'édition de ce programme (et donc lors de la phase de *décompilation*), la HP48, voyant arriver un *objet-expression*, reconstitue la forme usuelle de cette expression, en l'incluant entre deux délimiteurs '.
Si cet objet est un nom, elle crée un objet de type nom (global ou local, tout dépend si un nom local portant ce nom est connu à ce moment là). Pour autant, il lui faut bien inclure quelque part un marqueur indiquant que ce nom, lors de la phase de *décompilation*, devra à nouveau être entouré de deux '.

Autrement-dit:
Dans le programme « **'A' 'SIN(X)'** », les délimiteurs ' qui entourent l'expression **SIN(X)** sont "*factices*" (ils ne sont là que pour indiquer visuellement un objet de type expression). Au contraire, les délimiteurs ' qui entourent le nom **A** sont bien présents dans le code du programme.
Voici d'ailleurs quel est le codage de ce programme en mémoire:

D9D20 E1632	|Début Rpl et «
45632	|Begin'
84E20 10 14	|A
97632	|End'
8BA20	|Début expression
84E20 10 85 CA4B1	|X SIN
B2130	|Fin expression
93632 B2130	|» et Fin Rpl

On constate donc que les délimiteurs ' entourant un nom correspondent à deux instructions distinctes; je leur ai donné les noms suivants:
Begin' (adresse **#23654h**; **XLIB 1792 20**) pour le ' initial.
End' (adresse **#23679h**; **XLIB 1792 21**) pour le ' final.

Le rôle de "**Begin'**" est de placer sur la pile, sans l'évaluer, l'objet qui suit dans le Rpl (et également de tester si la touche **ON** n'est pas actionnée). Le rôle de "**End'**" est surtout visuel, car elle se contente d'effectuer le même test du clavier.

On peut par exemple envisager le programme suivant:

D9D20 E1632	|Début Rpl et «
45632	|Begin'
78BF1	|DUP
93632 B2130	|» et Fin Rpl

Ce programme est affiché: « **' DUP** ».
Quand il est évalué, il place l'instruction **DUP** sur la pile, sans l'éxécuter.

▸ Une autre instruction ' (mais *invisible*):
Il existe une autre instruction (à l'adresse **#2349Ch**, et identifiée par **XLIB 1792 15**), qui joue le même rôle que l'instruction "**Begin'**" qui vient d'être étudiée:
Elle place sur la pile, *sans l'évaluer*, l'objet suivant dans le Rpl. Elle scrute d'abord le clavier afin d'arrêter l'éxécution du programme si **ON** est actionnée.
La différence avec "**Begin'**" est qu'aucun nom *lisible* ne lui correspond à l'affichage. J'ai noté **Hidden'** cette instruction.

Par exemple:

D9D20 E1632	Début Rpl et «
45632	Hidden'
78BF1	DUP
93632 B2130	» et Fin Rpl

Ce programme est affiché: « **DUP** ».
Quand il est évalué, il place l'instruction **DUP** sur la pile, sans l'éxécuter....(en fait l'instruction **Hidden'** est placée devant DUP, mais elle reste *invisible*!).

◆ QUELQUES FORMES INTERNES DE THEN, START, etc...

Les instructions **THEN** et **REPEAT**, par exemple, attendent (pour décider de la suite du programme) un réel ou une expression au niveau 1. Ces deux possibilités conduisent à des formes internes distinctes. Voici les formes internes de **THEN**, **REPEAT**, etc..., avec des commentaires succints.

Adresse	Mnémonique	Commentaires (LM si langage machine)
#1A4A3h	ifte(r,2_)	Evalue obj2 si r<>0, sinon évalue Obj1
#1A4F0h	ift(r,_)	Evalue obj1 si r<>0, sinon le détruit
#1A513h	ift(ex,_)	Idem, avec une expression au niveau 2
#22F22h	then(r)	Forme interne de THEN, pour un réel
#22F4Fh	then(ex)	Idem, pour une expression
#23085h	repeat(r)	Forme interne de REPEAT, pour un réel
#230A3h	repeat(ex)	Idem, pour une expression
#23144h	start(2r)	Forme interne de START, pour deux réels
#23167h	start(2ex)	Idem, pour deux expressions
#23167h	start(r,ex)	Idem, pour un réel, puis une expression
#23180h	start(ex,r)	Idem, pour une expression, puis un réel
#231E1h	for(2r)	Forme interne de FOR, pour deux réels
#23213h	for(2ex)	Idem, pour deux expressions
#23213h	for(r,ex)	Idem, pour un réel, puis une expression
#2322Ch	for(ex,r)	Idem, pour une expression, puis un réel
#2326Ah	next	(LM) Forme interne de NEXT
#233A8h	step(ex)	Forme interne de STEP, pour une expr.
#233C1h	step(r)	Forme interne de STEP, pour un réel
#236E1h	enddo(r)	Forme interne du END de DO, pour un réel
#236FFh	enddo(ex)	Idem, pour une expression.
#237D5h	thencase(r)	Forme interne du THEN de CASE (réel)
#237F3h	thencase(ex)	Idem, pour une expression.
#54565h	ifte(ex,2_)	Forme de IFTE, une expr. au niveau 3

◆ L'exemple des deux formes de XROOT:

De nombreuses instructions de la HP48 sont autorisées aussi bien en *notation polonaise inverse* qu'en *notation algébrique* (on les appelle les *fonctions*, par opposition aux *commandes*).

Un petit nombre d'entre elles sont implantées en deux adresses différentes. Elles constituent donc deux objets distincts de la HP48, bien que représentées "*en clair*" par un même mot.

C'est le cas pour les instructions **XROOT**, **!** (factorielle), **WHERE**, **∂** (dérivation), **APPLY**, et **∫** (intégration).

Prenons l'exemple de **XROOT**.

En notation polonaise inverse, la séquence **x y ROOT** calcule la valeur de **x^(1/y)** (c'est la racine y-ième de x).

Par exemple, le résultat de **8 3 XROOT** est 2 .

Pourtant, la forme équivalente à la séquence **x y XROOT** est, en notation algébrique: **'XROOT(y,x)'**, ce qui constitue une exception avec le cas général.

L'origine de cette situation est à chercher dans l'environnement **EQUATION-WRITER**, où on constate que, dans une expression comme **'XROOT(3,8)'**, l'exposant 3 précède l'argument 8 (à la saisie comme à l'affichage).

Les deux programmes **« 8 3 XROOT »** et **« 'XROOT(3,8)' EVAL »**, pourtant équivalents, sont codés des deux manières suivantes:

▸ **Programme « 8 3 XROOT »:**

```
D9D20 E1632          |Début de Rpl et «
   C53A2 3F2A2 581B1      |8 3 XROOT
93632 B2130          |Fin de Rpl et »
```

▸ **Programme « 'XROOT(3,8)' EVAL »:**

```
D9D20 E1632          |Début de Rpl et «
   8BA20             |Début d'expression
   3F2A2 C53A2 AC1B1      |8 3 XROOT
   B2130             |Fin d'expression
   EB3A1             |EVAL
93632 B2130          |Fin de Rpl et »
```

INDEX

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

NOTES

Achevé d'imprimer par

PARA GRAPHIC

31240 L'UNION (Toulouse)
Tél. (16) 61.74.27.67
Dépôt légal : Août 1992

LES *SECRETS* DE LA HP 48

TOME 1

Ce nouveau livre de **Jean-Michel FERRARD** va combler tous les amoureux de la HP 48. Le tome 1 contient une quantité énorme d'informations inédites sur cette calculatrice.

Dans une première partie, l'auteur se livre à une présentation très détaillée de tous les objets de la HP 48 et de la façon dont ils sont codés en mémoire. Les objets les plus mystérieux sont minutieusement décrits (entiers-système, librairies, linked-arrays, noms XLIB, réels et complexes longs, etc.).

Les adresses où apparaissent tous ces objets dans la ROM de la HP 48 sont également données.

La deuxième partie du livre est la plus importante et la plus novatrice. Elle fournit et décrit plusieurs milliers de points d'entrées (SYSEVALs) inédits de la HP 48. Vous accédez ainsi au fonctionnement le plus intime de votre calculatrice.

Pour faciliter la lecture et les recherches, ces milliers de SYSEVALs sont organisés en plus de trente chapitres traitant chacun d'un sujet précis.

- Ce sont des centaines d'instructions supplémentaires qui apparaissent, des dizaines d'effets spéciaux.
- C'est la possibilité de faire ce que vous voulez de votre calculatrice, d'épater les copains.

Les secrets de la HP 48 : pour en savoir toujours plus que tous les autres...

ISBN 2-908791-08-0

PRIX 200 F TTC