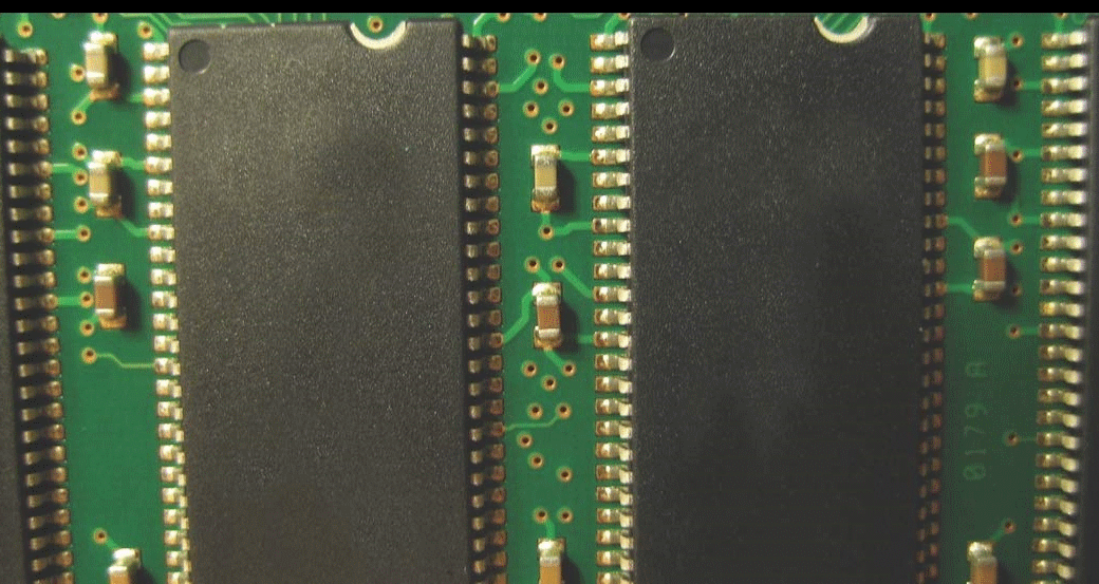
Voyage au centre de la HP48 s/sx

Paul Courbis & Sébastien Lalande



VOYAGE AU CENTRE DE LA HP48

VOYAGE AU CENTRE DE LA HP48

Troisième édition

Revue et complétée

Paul COURBIS
&
Sébastien LALANDE

Hewlett-Packard, HP71, HP28, HP48, HP48s, HP48sx,
Macintosh, Atari, Unix, Amiga et IBM
sont des marques déposées.

Première édition: Août 1991
Deuxième édition: Mars 1992
Troisième édition : Décembre 2000
Version électronique distribuée sur http://www.courbis.com
avec l'aimable autorisation des éditions Angkor

© 1991-2000, Angkor, Paris.
ISBN 2-87892-003-1

Tous les efforts ont été faits pour que les informations, programmes et schémas présentés dans ce livres 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.

Paul COURBIS
24 ans, ingénieur Civil des Mines de Saint-Etienne, ancien élève de mathématiques spéciales au lycée Pasteur (Neuilly-sur-Seine).

Sébastien LALANDE
22 ans, élève-ingénieur à l'Ecole Nationale Supérieure des Télécommunications de Paris, ancien élève de mathématiques spéciales au lycée Pasteur (Neuilly-sur-Seine).

Des mêmes auteurs:

Voyage au centre de la HP28 c/*s*

Nous tenons à remercier:

Nos familles respectives pour l'aide et le soutien qu'elles nous ont apportés;

Marc BERNARD DE COURVILLE pour ses nombreuses critiques;

Christophe DUPONT DE DINECHIN pour son programme µSOLVER;

Dominique MOISESCU pour son programme SSAG;

Christophe NGUYEN pour ses programmes CIRCLE et BANNER;

Yann ROUSSE de Maubert Electronic;

Jean TOURRILHES pour ses précieuses corrections;

Tous les membres du groupe comp.sys.hp48;

Ainsi que toutes les personnes qui ont contribué par leurs remarques et leurs conseils à la réalisation de cet ouvrage...

Note au lecteur

Cet ouvrage s'adresse à la fois aux néophytes et aux programmeurs expérimentés: il comprend des chapitres d'initiation à l'utilisation "classique" de la HP48 ainsi que des explications permettant d'accéder à des ressources non révélées par le constructeur... Il a un double but: expliquer comment accéder à toutes ces ressources et servir de boîte à outils.

Il est divisé en quatre parties:

- Une première partie qui familiarise le lecteur avec les principes de base de la HP48: notation polonaise inversée, utilisation de la pile, langage de programmation... Des exercices corrigés sont proposés;
- Une deuxième partie qui présente de manière progressive les ressources cachées de la HP48 sous une forme claire, accessible à tout utilisateur et émaillée de nombreux exercices (corrigés en annexe); ce cours d'initiation au langage machine pourra ensuite servir de manuel de référence pour le programmeur;
- Une bibliothèque de programmes variés prêts à l'emploi. Jeux, programmes mathématiques, utilitaires divers, programmes musicaux... sont au rendez-vous !
- Une série d'annexes contenant des documents de référence pour le programmeur (liste exhaustive des messages d'erreur, liste complète des instructions...).

Il est important de noter que l'existence de différentes versions de HP48 (S ou SX) est prise en compte dans cet ouvrage: tous les programmes, schémas et autres informations, à l'exception de ce qui concerne les modules-mémoire, sont indépendants du type de machine possédée.

Maintenant c'est à vous !

Nous vous souhaitons une agréable lecture !

Table des matières

Note au lecteur ..... Page 1

Table des matières ..... Page 3

Première partie
La HP48

Les principes de base de l'utilisation de la HP48 comme le constructeur les décrit...

Introduction ..... Page 13

Chapitre 1 Première approche de la HP48 ..... Page 15

En guise d'entrée: comment s'y retrouver parmi toutes les inscriptions présentes sur cette machine...

Chapitre 2 La notation polonaise inversée ..... Page 19

Les principes de base de cette notation, accompagnés d'exemples et d'exercices...

Chapitre 3 Bien organiser ses données.....Page 27

Comment stocker ses données de manière à les retrouver facilement: notions d'arborescence, de répertoires...

Chapitre 4 La programmation de la HP48.....Page 33

Qu'est ce qu'un programme ? Comment concevoir un programme ? Comment fonctionne le langage de la HP48 ? Des conseils de programmation et des exemples pas à pas...

Chapitre 5 Bien présenter ses données.....Page 45

Comment présenter ses données de manière à les rendre facilement utilisables: utilisation du menu CST, des redéfinitions de touches...

Chapitre 6 Sauver et échanger des données.....Page 51

Ou comment utiliser les capacités d'échange avec l'extérieur de la HP48: cartes mémoire, prise RS232c, émetteur/récepteur infra-rouge...

Chapitre 7 Les autres points forts de la HP48.....Page 55

La HP48 possède de nombreux atouts: calcul symbolique, graphiques, gestion d'unités...

Conclusion.....Page 59

Deuxième partie
Le langage machine

Les ressources cachées de la HP48:
Comment faire plus que ce que Hewlett-Packard a prévu...

Introduction.....Page 63

Chapitre 1 Qu'est ce que le langage machine ?.....Page 67

Une initiation au langage-machine et aux notions de base utiles pour lire le reste de cette partie...

Chapitre 2 Le microprocesseur Saturn.....Page 71

Une vision générale du microprocesseur de la HP48: vue détaillée de tous ses registres et de ses différentes particularités...

Chapitre 3 Les instructions du Saturn.....Page 81

Toutes les instructions disponibles, classées par type de fonction réalisée et par registres mis en cause...

Chapitre 4 Les objets de la HP48.....Page 121

Les principes de stockage en mémoire de tous les objets accessibles à l'utilisateur (réel, entier, objet graphique...) et des autres...

Chapitre 5 Organisation générale de la mémoire.....Page 159

Une vision globale de la mémoire de la HP48 éclairant les explications détaillées qui suivent...

Chapitre 6 La ram des entrées-sorties ..... Page 163

Comment accéder directement à certains des périphériques de la HP48 (l'horloge, les entrées sorties infra-rouge...).

Chapitre 7 La mémoire vive..... Page 173

L'organisation détaillée de la mémoire vive de la HP48...

Chapitre 8 Programmer en langage-machine ..... Page 203

Ou comment accéder à toutes les ressources de la HP48...

Troisième partie
Bibliothèque de programmes

Toute une collection de programmes utiles et prêts à l'emploi...

Avertissement ..... Page 211

Comment entrer un programme en langage machine...

Programmes concernant le langage machine

GASS	Installation de programmes assembleurs	Page	213
ALLBYTES	Calcul des checksums d'un répertoire	Page	214
PAR5	Présentation de chaînes de codes	Page	215
CLEAN	Nettoyage de chaînes de codes	Page	216
PEEK	Lire dans la mémoire de la HP48	Page	218
POKE	Ecrire dans la mémoire de la HP48	Page	220
HRPEEK	Lire la rom cachée de la HP48	Page	222
?ADR	Connaître l'adresse d'un objet dans la pile	Page	226
SAGG	Fonction inverse de GASS	Page	227
RASS	Un GASS instantané	Page	228
CHK	Vérification d'arguments	Page	230
REVERSE	Retourner des chaînes de caractères	Page	235
CRNAME	Créer des noms non-standards	Page	237
CLVAR	Inhiber la fonction CLVAR	Page	238
SYSEVAL	Inhiber la fonction SYSEVAL	Page	239
CONTRAST	Régler le contraste par programme	Page	240
DISPOFF	Eteindre l'écran par programme	Page	241
DISPON	Rallumer l'écran	Page	241
FAST	Augmenter la vitesse de calcul de la HP48	Page	242
DESASS	Un désassembleur SATURN	Page	243
B->SB	Transformer un entier en entier système	Page	260
SB->B	Entier système en entier	Page	260
R->SB	Réel en entier système	Page	260
SB->R	Entier système en réel	Page	260
C->SB	Caractère en entier système	Page	260

SB->C	Entier système en caractère	Page	260
ROMRCL	Rappeler des objets en rom cachée	Page	261
A->STR	Transformer une adresse en chaîne	Page	262
STR->A	L'opération inverse	Page	262
ROMSEARCH	Chercher un objet en rom	Page	263
RAMSEARCH	Chercher un objet en ram	Page	263
MODUSEARCH	Chercher un objet dans un module	Page	263
CRC	Calcul de checksum	Page	266
CRCLM	Une version assembleur de CRC	Page	266

Programmes mathématiques

CALC	Un calculateur entier à précision infinie	Page	267
PI	Calculer π avec précision	Page	287
VAL	Valeur d'un polynôme stocké en vecteur	Page	289
A->V	Polynôme algébrique en vecteur	Page	290
V->A	Fonction inverse de la précédente	Page	290
DER	Dérivée d'un polynôme en vecteur	Page	291
DIVP	Division de deux polynômes en vecteurs	Page	292
PCAR	Calcul de polynômes caractéristiques	Page	293
LAGU	Toutes les racines de tous les polynômes	Page	294
PMAT	Image d'une matrice par un polynôme	Page	297
µSOLVER	Résolution de systèmes d'équations	Page	298

Jeux

LABY	Sortez du labyrinthe maudit !	Page	302
MASTER	Jouez au Master-Mind	Page	309
ANAG	Tous les anagrammes d'un mot	Page	313
CARRE	Jouez au carré magique	Page	315

Programmes divers

PR40	Imprimer en 40 colonnes	Page	318
DSP	Un écran texte de 33 colonnes	Page	319
MUSICLM	Un peu de musique	Page	321
MODUL	Des effets sonores	Page	323
RABIP	De la musique aléatoire	Page	325
JINGLE	Un petite musique sympa	Page	326
RENAME	Renommer une variable	Page	327
AUTOST	Un programme en démarrage automatique	Page	328
CAL	Affiche un calendrier sur un mois	Page	329
CIRCLE	Tracé rapide de cercles	Page	331
BANNER	Ecrire en caractères géants	Page	334

Annexes

Réponses aux exercices, documents de référence pour le programmeur glossaire et index...

Réponses aux exercices.....Page 343

Toutes les réponses aux exercices...

Informations diverses .....Page 349

Quelques informations utiles: comment déterminer la version de sa machine, que faire en cas de graves problèmes...

Binaire, hexadécimal et autres barbaries.....Page 351

Une explication détaillée de notions chères aux informaticiens: hexadécimal, binaire, bits, quartets, octets...

Routines utiles .....Page 355

Quelques programmes en langage machine tous faits...

Liste exhaustive des messages d'erreurs .....Page 357

Tous les messages d'erreurs que la HP48 est capable d'émettre...

Instructions du langage machine.....Page 365

Sur deux pages en vis-à-vis, toutes les instructions-assembleur de la HP48 avec leur code. L'idéal pour le programmeur en langage-machine...

Liste de toutes les instructions de la HP48

Ou comment allier la rapidité du langage-machine à la puissance des instructions développées par Hewlett-Packard...

Par ordre alphabétique.....Page 369
Par numéro de commande .....Page 375

Liste d'objets utiles présents en mémoire morte ..... Page 381

Une liste d'objets déjà codés par Hewlett-Packard. Pourquoi se fatiguer alors que le travail est déjà fait ?..

Glossaire ..... Page 405

Un petit dictionnaire des termes utilisés dans cet ouvrage...

Index..... Page 409

Ou comment retrouver la bonne page en ne se souvenant que d'un mot...

Première partie

La HP48

Introduction

Vous avez entre les mains une des meilleures machines à calculer du marché, si ce n'est la meilleure...

Elle diffère des autres machines du commerce car elle est beaucoup plus complexe du point de vue matériel, bien plus simple du point de vue de l'utilisateur, et peut vous permettre de résoudre des problèmes d'une haute complexité.

Etant donné le nombre incroyable de fonctions internes et leur puissance, il a fallu élaborer un système d'utilisation très puissant, que tout le monde puisse utiliser, du mathématicien émérite, à l'informaticien le plus compétent, en passant par les physiciens, les statisticiens (...) mais aussi par ceux qui n'entendent rien à tous ces domaines.

L'utilisation de cette machine étant bien différente de celle des calculatrices habituelles, elle apparaît souvent au premier abord comme compliquée alors qu'en réalité c'est certainement la plus simple qui soit. Ce n'est qu'une question d'habitude et en quelques jours, avec un peu de pratique, vous deviendrez un virtuose de la HP48...

Les chapitres de cette première partie sont consacrés à une vision générale de l'utilisation standard de la machine: quelques trucs à savoir, comment faire des programmes simples, comment s'organiser...

Mais attention: ces quelques informations ne peuvent en aucun cas se substituer aux manuels fournis par Hewlett-Packard ! Le but de cette partie n'est que de vous présenter les capacités de votre machine, de manière à vous faciliter la lecture de ces manuels...

En fait, la HP48 permet de faire beaucoup plus de choses que ce que Hewlett-Packard présente dans ses manuels: grâce au langage-machine il est possible d'accéder à de nouvelles ressources, de réaliser des programmes infiniment plus rapides...

C'est pourquoi une deuxième partie vous apprendra, de manière très didactique, abordable par les programmeurs de tous

niveaux, ce qu'est la programmation en langage machine et vous décrira la structure interne de la HP48...

Si vous ne connaissez rien au langage machine ou à l'assembleur voici une bonne occasion de vous en faire une idée...

Mais avant de passer à cela, il convient de bien connaître l'utilisation normale de la machine !

Pour vous aider dans cet apprentissage, des exemples de programmes, depuis des programmes élémentaires jusqu'à des programmes complexes, sont donnés en troisième partie (bibliothèque de programmes).

En les utilisant et en les modifiant au gré de votre imagination, vous serez très rapidement capable de faire vous-même des programmes sophistiqués...

Première approche de la HP48

Votre machine est sous vos yeux, elle est tapissée de boutons et d'inscriptions bleues, oranges et blanches qui ne signifient pas grand chose à première vue...

NON ! Ne partez pas en courant ! C'est comme un sapin de Noël: à première vue ça fait fouillis, mais si on s'y arrête quelques instants, on s'aperçoit que les décorations ont été placées judicieusement, que chacune est à sa place, et que son créateur n'a pas travaillé à la légère, bien au contraire.

Dites vous bien que bientôt vous arriverez à maîtriser l'ensemble et trouverez cela génial...

Avant toute chose, comme chaque appareil électrique, la HP48 a besoin de courant. Vérifiez donc que les trois piles électriques se trouvent bien dans le compartiment (au dos de la machine en bas) et dans le bon sens (la pile du haut et celle du bas ayant le "plus" vers la gauche, celle du milieu l'ayant vers la droite).

I) Le clavier

La deuxième chose à faire est de la mettre en marche. Jusque là tout est simple, il suffit d'appuyer sur le bouton `[ON]` qui est la première touche en bas à gauche (c'est écrit en blanc).

Au-dessus (c'est-à-dire dans la direction de l'écran à cristaux liquides) se trouvent deux touches [↱] (bleue) et [↰] (orange).

Les inscriptions blanches inscrites sur la touche correspondent en général à l'action d'un simple appui sur la touche.

Les inscriptions bleues au dessus d'une touche correspondent à l'appui de [↱] (bleue) suivi de l'appui de cette touche. De même les inscriptions oranges correspondent à un appui de [↰] suivi d'un appui sur cette touche.

Ainsi [↱] [STO] exécute la commande RCL (qui comme nous le verrons plus tard effectue un ReCaLl, c'est à dire rappelle le contenu d'une variable).

Au dessus de [↰] se trouve la touche [α].

Si vous appuyez une seule fois sur [α] la prochaine touche appuyée correspondra à une lettre (celle inscrite en blanc à droite de certaines touches).

Par exemple, [α] puis [SIN] donnera la lettre "S", alors qu'un simple appui sur [SIN] exécutera la fonction sinus.

Pour rester en mode alphabétique, il convient d'appuyer deux fois de suite sur [α].

Pour sortir de ce mode, il suffit d'appuyer une autre fois sur [α].

Ainsi, pour taper 'AB' il faut appuyer consécutivement sur les touches: ['] [α] [α] [A] [B] [ENTER].

II) L'écran

Il est divisé en 3 parties:

- Au dessus de la barre horizontale se trouve l'état de la machine. Vous y trouverez entre accolades ({ }) le répertoire courant (voir chapitre 3 pour vous familiariser avec l'arborescence).

  Peuvent s'y trouver aussi de petits chiffres (1, 2, 3, 4, et 5) indiquant l'état de certains indicateurs de la machine, l'indication du mode de mesure des angles (RAD, pour le mode "radians", ou GRAD, pour le mode "gradians", rien n'apparaissant en mode "degrés") ainsi que la date et l'heure.

- En dessous, séparées de la première zone par une barre horizontale, sont affichées 4 lignes:

  Il s'agit de l'affichage de la pile (voir le chapitre 2).

- Enfin, la troisième zone représente le "menu" courant qui est constitué de 6 cases noires dans lesquelles se trouvent des noms évoquant la fonction des 6 touches blanches situées juste en dessous (première rangée du clavier).

  Ainsi la touche blanche [ ]A engendre-t-elle la fonction dont le nom figure sur la première case du menu (en bas à gauche de l'écran) et ainsi de suite pour les suivantes.

  Une case surmontée d'une petite barre horizontale correspond à l'accès à un sous-menu.

  Ces notions de menus et de sous-menus seront revues dans le chapitre 3...

  Ainsi, si vous appuyez sur "MEMORY" ([↰] puis [VAR]), conduira à afficher le menu de gestion de la mémoire: "MEM", "BYTES", "VARS", ORDER", "PATH", "CRDIR". Appuyer alors sur la touche [ ]C revient à exécuter la fonction VARS...

Cette première vision de la HP48, orientée vers l'aspect physique de la machine, est maintenant terminée.

Nous allons à présent entrer dans le monde merveilleux de l'utilisation de cette fantastique machine...

Exercices:

A-1-1: Quelle séquence de touches doit-on utiliser pour accéder à "=" ?

A-2-2: Même question pour "RCL".

La notation polonaise inversée

La HP48 utilise un mode de calcul appelé "la notation polonaise inversée" (en anglais Reverse Polish Notation, RPN) qui repose sur le principe de pile.

Définissons tout d'abord le principe de pile...

I) La pile

Imaginez une pile d'assiettes... La seule assiette accessible à un instant donné est celle du dessus (la première).

La HP48 stocke les données temporaires de la même manière: elle les empile... et vous les montre à l'écran (du moins pour les 4 dernières entrées) précédées de leur numéro d'ordre (1:, 2:, 3: et 4:). Evidemment cela ne ressemble plus trop à notre pile d'assiettes puisque la première est celle du bas... Mais le principe reste le même !

Suivant le principe de pile, seule la donnée au niveau 1 (la plus en bas de l'écran) est disponible... Heureusement il existe des commandes permettant d'influer sur l'ordre des éléments ! Mais avant de les étudier, apprenons à placer des données dans cette fameuse pile...

La HP48 gère plusieurs types de données (réels, entiers, chaînes de caractères, noms, programmes, équations, objets graphiques, etc...). Chacun de ces types d'éléments est susceptible d'être placé dans la pile.

Pour cela, il suffit de taper l'intitulé de l'objet et de presser [ENTER] qui valide cette entrée.

Par exemple, pour placer le réel 123 dans la pile, il suffit de taper la séquence de touches: [1] [2] [3] [ENTER].

L'écran change alors d'aspect, et sa partie centrale présente un affichage du type suivant:

Cela signifie que la pile contient un élément, 123, placé au niveau 1...

Remarque: la HP48 n'affiche que les quatre premiers niveaux de la pile, mais celle-ci peut être beaucoup plus importante (limitée seulement par la mémoire disponible).

II) Calculer en RPN

Les différentes fonctions de la HP48 (addition, soustraction...) vont donc devoir prendre leurs données dans la pile... Et leur résultat ? Très logiquement elles le remettent dans cette pile !

Ce style de notation est souvent déroutant pour l'utilisateur débutant habitué à la notation standard. Mais avec l'usage il se rendra vite compte qu'elle est plus performante. En particulier, elle évite l'usage de parenthèses, car la pile sert au stockage des données intermédiaires: par exemple pour calculer (2+3)*(4+5), on effectuera les commandes suivantes:

- On part de la pile vide (si elle ne l'est pas, utilisez la commande CLR ([↱] [←]) qui la nettoie, et que nous reverrons bientôt). L'affichage central est alors:

- `[2] [ENTER]` La pile est alors:

- [3] [ENTER] La pile est alors:

Remarquez que le "3" a poussé le "2" au deuxième niveau, ce qui est normal puisque la nouvelle "assiette du dessus" est "3"...

- [+] qui réalise l'addition:

- [4] [ENTER] La pile est alors:

- [5] [ENTER] La pile devient:

- [+] qui donne:

- Et enfin [*], d'où le résultat:

Nous n'avons eu aucune parenthèse à taper, et en plus nous avons pû contrôler les résultats intermédiaires (5 et 9)...

La seule chose à se rappeler est donc: une commande prend ses arguments (les données dont elle a besoin) dans la pile, et y replace ses résultats...

III) Gérer la pile

Nous avons vu que les commandes n'influaient que sur les premiers éléments de la pile, alors est-il impossible d'accéder aux autres ? Non, car tout est prévu: nous avons à notre disposition des commandes de gestion de la pile... En particulier nous disposons au clavier des commandes suivantes:

- SWAP ([↰] [▶]) qui échange les deux premiers éléments de la pile (niveaux 1 et 2). Par exemple:

  donnera après SWAP:

- DROP ([↰] [←]) qui enlève l'élément au niveau 1. Par exemple:

  donnera:

- CLR ([↱] [←]) qui vide la pile. Appliquée à une pile quelconque, il conduira à:

Mais il en existe d'autres: elles sont accessibles par le menu STK, sous menu de PRG (appuyer sur [PRG] puis [STK], première touche de menu, et n'oubliez pas, les menus s'affichent par pages de six fonctions et deux commandes permettent de passer de page en page: NXT et PREV). Ces commandes de gestion sont:

- OVER place dans la pile une copie de l'élément situé au niveau 2:

conduira à:

- ROT effectue une rotation des 3 premiers éléments de la pile:

donnera:

- ROLL est une fonction similaire, mais qui prend un argument (au niveau 1 de la pile) correspondant au

nombre d'éléments à traiter. Ainsi 2 ROLL correspond à SWAP, 3 ROLL à ROT...

- ROLLD fonctionne comme ROLL mais effectue une rotation dans l'autre sens. Par exemple si la pile est:

```
4:                4
3:                5
2:                6
1:                3
```

Alors ROLLD donnera:

```
4:
3:                6
2:                4
1:                5
```

(ne pas oublier que ROLLD prend un argument, ici 3).

- PICK prend aussi un argument dans la pile. Elle considère alors qu'il s'agit d'un numéro de niveau et copie l'élément qui s'y trouve. 2 PICK correspond donc à OVER. Exemple:

```
4:            123456789
3:                    1
2:                    1
1:                    3
```

alors PICK donnera:

```
4:            123456789
3:                    1
2:                    1
1:            123456789
```

(ne pas oublier que PICK prend un élément dans la pile, ici 3).

- DEPTH renvoie le nombre d'éléments dans la pile, c'est à dire le nombre d'étages occupés. Si la pile est vide on obtiendra donc 0. Exemple:

```
4:
3:
2:                33333
1:                44444
```

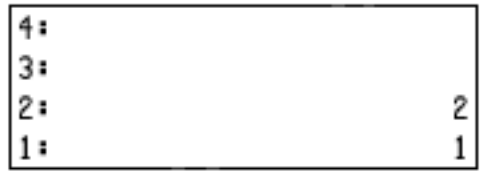
donnera:

(il y avait 2 éléments dans la pile).

- DUP duplique l'élément au niveau 1:

donne:

- DUP2 duplique les 2 premiers éléments de la pile. Ainsi:



donnera:

- DUPN est une généralisation de DUP et DUP2: elle prend un argument (n) et duplique les n premiers éléments de la pile. Ainsi 1 DUPN correspond à DUP et 2 DUPN à DUP2.
- DROP2 ôte les deux premiers éléments de la pile:

donnera:

- DROPN est une généralisation de DROP et DROP2. Elle prend un arguments dans la pile (n) et ôte les n premiers éléments de la pile. 1 DROPN correspond à DROP, 2 DROPN à DROP2...

Cette revue des commandes de gestion de la pile est terminée. Comme vous pouvez le constater, le jeu de commandes mis à votre disposition est très complet...

Exercices:

A-2-1: Calculer 5/((3+1)*(9-5))

A-2-2: Si la pile contient:

comment arriver à:

A-2-3: Que calcule la séquence de touches suivante ?

[5] [ENTER] [3] [*] [1] [1] [-] [4] [/] [1] [-] [COS]

Quel en est le résultat ?

Bien organiser ses données

La HP48 est un véritable petit ordinateur, et à ce titre elle doit être capable de stocker des données. Celles-ci peuvent être de différents types: réels, entiers, programmes, listes...

Elles peuvent se regrouper en deux familles: données internes (fonctions préprogrammées...) et données utilisateur (celles que vous entrerez dans votre machine).

Toutes ces données apparaissent soit sous la forme d'objets dans la pile (partie centrale de l'écran), soit sous formes d'entrées dans les menus...

I) Menu

Il existe deux types de menus: les menus de fonctions internes et les menus utilisateur. Dans ces derniers, vous verrez apparaître vos propres données. Un menu est une série d'objets accessibles par leurs noms, mis en correspondance avec les six touches du haut du clavier.

Si il y a plus de 6 objets, les autres apparaîtront si l'on fait défiler la liste grâce à [NXT] (NEXT, page suivante, c'est-à-dire les six cases suivantes) et [↰] [NXT] qui est PREV (PREVIOUS, page précédente).

Ainsi: [↰] [VAR] (MEMORY) vous amène dans le menu MEMORY qui rassemble l'ensemble des fonctions internes assurant la gestion de la mémoire. Si vous appuyez sur [ ]A (au dessous de MEM, en bas à gauche de l'écran), la machine renvoie une valeur dans la pile (partie centrale de l'écran). L'affichage ressemblera à:

Lorsque vous avez appuyé sur la touche [ ]A, la HP48 a reconnu que vous désiriez exécuter l'objet MEM et a répondu à votre demande. Cette fonction renvoie la mémoire libre (c'est à dire la place qui reste à votre disposition). Cette valeur est exprimée en octets (voir l'annexe "Binaire, hexadécimal et autres barbaries...").

Si vous appuyez sur [NXT], vous pourrez accéder aux autres fonctions du menu MEMORY...

Remarque: cette promenade dans le menu est cyclique: si, arrivé à la dernière page de menu, vous appuyez encore sur [NXT], vous vous retrouverez à la première page...

Autre exemple: [↰] [MODES] vous met dans le menu MODES qui se décompose en 4 pages ressemblant à:

page 1: [STD] [FIX] [SCI] [ENG] [SYM] [BEEP]

page 2: [STK] [ARG] [CMD] [CNC] [ML] [CLK]

page 3: [DEG] [RAD] [GRAD] [XYZ] [R<Z] [R<<]

page 4: [HEX] [DEC] [OCT] [BIN] [FM,] [ ]

Les pages 1, 2, 3 et 4 s'enchaînent dans cet ordre avec NXT et dans l'ordre inverse avec PREV.

Si vous appuyez sur [CLK] en fin de page 2, l'heure et la date apparaissent et disparaissent en haut de l'écran à chaque appui et [CLK] devient [CLK•] et réciproquement. Lorsqu'un "•" apparait dans une case du menu cela signifie que l'option en question est active.

C'est grâce à ces menus que l'on peut paramétrer le fonctionnement de la HP48 (mode de mesure des angles...).

Dans certains menus se trouvent de petites barres au dessus de certaines cases. Tel est le cas pour le menu PROGRAMS (accessible en appuyant sur [PRG]).

Cette barre signifie que si vous appuyez sur la touche correspondante, vous aurez accès à un menu, sous-menu du premier.

Cette structure peut se décrire par une arborescence:

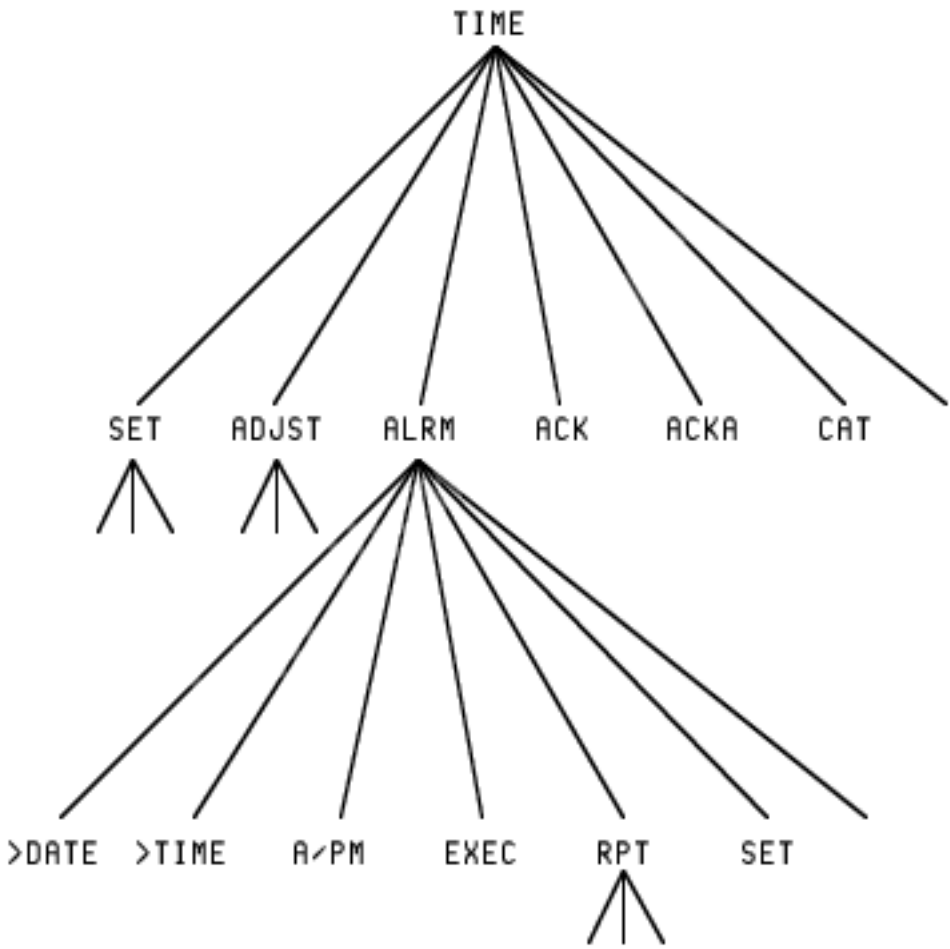
II) L'arborescence des menus

Pour expliquer ceci nous utilisons la notion d'arbre.

Le menu principal (le premier) s'appelle la racine. Dans cette racine on voit des cases "normales" et éventuellement d'autres qui possèdent une petite barre sur le dessus.

Ces cases "spéciales", correspondant à des sous-menus, sont des branches qui partent vers d'autres menus.

Au bout de chaque branche se trouve un nouveau menu qui est sous-menu de la racine. Par exemple, pour TIME ([↰] [4]), on a l'arborescence suivante (représentée partiellement):



Ces sous-répertoires peuvent eux-mêmes avoir des sous-répertoires en plus des données qu'ils contiennent (par exemple

REPT constitue un sous-menu du sous-menu ALRM) et ainsi de suite.

Pour nommer les menus les uns par rapports aux autres, on parle de menu-père et de menu-fils. Ce sont deux menus reliés par une branche, le père est celui le plus proche de la racine, le fils est celui le plus éloigné...

III) Le menu "VAR"

Le menu "VAR" est votre menu. C'est là que vous pouvez stocker vos données, créer vos propres sous-menus...

La racine du menu VAR a un nom particulier: HOME. Pour descendre dans un sous-menu, il suffit d'appuyer sur la touche correspondante à une case de menu (avec la petite barre au dessus) ou de taper son nom.

Pour revenir au menu- père il suffit de faire faire [↰] ['] (UP).

Et pour revenir directement à la racine: [↱] ['] (HOME).

On appelle menu VAR courant, le répertoire dans lequel nous nous trouvons à un instant donné.

Pour stocker une donnée, il suffit de la mettre dans la pile, de rentrer un nom (suite de caractères entre ') et de le STOcker...

Exemple: 512 [ENTER] (place le réel 512 dans la pile), puis:

['] [α] [α] [A] [B] [C] [ENTER]

L'écran ressemblera alors à:

Tapez [VAR] (pour vous placer dans votre menu) puis [STO] (pour stocker le réel). [ABC] apparaît à gauche (au début) du menu courant...

Pour rappeler la donnée, il suffit de faire ['] [α] [α] [A] [B] [C] [ENTER] [↱] [STO].

Vous pouvez aussi faire [↱][ABC] où [ABC] est la touche du menu correspondant à ABC. De même si [ABC] existe déjà dans le menu pour stocker quelque chose sous le même nom (ce qui efface l'ancienne donnée), il suffit de mettre dans la pile le nouveau contenu et de faire [↰][ABC].

Si le contenu d'une case de menu n'est pas un programme, vous pouvez rappeler son contenu en appuyant simplement sur la touche correspondante. Ainsi pour rappeler le réel 512 précédemment stocké, suffit-t-il d'appuyer sur la touche [ABC] ...

Vous pouvez créer un sous-répertoire avec la fonction [CRDIR] dans le menu MEMORY en tapant un nom (par exemple 'DIREC'), puis en appuyant sur la touche [CRDIR].

Grâce à cette possibilité de créer des sous-menus, vous pourrez regrouper vos différentes données par affinités.

Par exemple, si vous entrez dans votre machine des programmes de mathématiques, des programmes en langage machine et des programmes de jeux, vous aurez tout intérêt à créer 3 sous-menus dans le menu HOME: 'MATHS', 'LM', et 'JEUX'.

Vous placerez dans chacun d'eux les données correspondantes, ce qui vous permettra de les retrouver facilement...

Trois autres commandes sont importantes à connaître dans le cas du menu VAR: il s'agit de UPDIR ([↰] [']) qui permet de remonter d'un étage dans l'arborescence de VAR (pour passer du menu-fils au menu père), HOME ([↱][']) qui permet de remonter directement à la racine de VAR. Enfin, la commande PATH (dans le menu memory: [↰][VAR][PATH]) permet de savoir où l'on se trouve dans l'arborescence de VAR. Cette commande renvoie une liste contenant la suite des noms des menus (le premier élément est donc toujours HOME).

Exercices:

A-3-1: Créer un sous-répertoire EXO du menu HOME regroupant trois variables A B et C, contenant respectivement les réels 1 2 et 3.

A-3-2: Combien de sous-menus contient le menu MTH ?

La programmation de la HP48

Jusqu'à présent, nous avons utilisé diverses commandes de la HP48.

Il est possible de créer ses propres commandes en utilisant ces dernières.

La HP48 ne possède pas seulement des commandes de base, mais un véritable langage de programmation...

Ce langage s'appelle le RPL (Reverse Polish Lisp ou Lisp Polonais Inversé)... Pourquoi ce nom étrange ? Parce que ce langage dérive d'un autre, le LISP ("LISt Processor" ou "Lot of Insane and Stupid Parenthesis" selon les auteurs).

Ce langage, très puissant et utilisé en intelligence artificielle, est malheureusement assez difficile d'emploi du fait de sa syntaxe: chaque commande de base s'écrit entre parenthèses... D'où une profusion de "(" et de ")" dans les programmes, ce qui les rend peu lisibles...

Mais la notation polonaise inversée, comme nous l'avons vu, permet de se passer de parenthèses...

Le RPL était né !

Ce langage gère des objets... Ce terme peut paraître assez vague, et c'est bien son but ! En effet la HP48 fait le moins de distinctions possibles sur le type des entités qu'elle manipule: la fonction utilisée est générique et s'adapte au cas particulier qui lui est proposé...

Ainsi, c'est le même "+" qui servira à additionner deux réels, deux entiers, deux matrices mais aussi un réel à un entier, une chaîne de caractères à une liste...

Grâce à cette aptitude de la fonction à s'adapter, il est possible de réaliser facilement des programmes complexes, qui, dans la plupart des cas, seront eux aussi génériques...

Par exemple: si la pile contient les réels 2 et 3:

appuyer sur + conduira à:

Ce qui est bien le résultat de 2+3...

Si vous y mettez "ABC" et "DEF":

Alors "+" réalisera l(addition (ou plus exactement la concaténation) de chaînes et la pile sera alors:

La commande s'est donc bien adaptée au types des arguments qui lui sont fournis...

Les concepts de base du Reverse Polish Lisp étant fixés, nous allons maintenant étudier les méthodes de programmation de la HP48...

I) Méthodes

Comme nous l'avons vu, un programme est un groupe de commandes; dans le cas du RPL, ce groupe de commandes est signalé par deux symboles: « et ».

Un programme en RPL est donc une suite de commandes mises entre « et ».

Prenons un exemple:

Pour calculer la puissance cubique d'un réel, nous taperions le réel puis la séquence suivante:

[3] [y^x]

Mais si nous avions beaucoup de cubes à calculer, il serait intéressant d'automatiser cette procédure. Nous allons pour cela créer le programmes CUBE1...

Comme nous l'avons vu, un programme est une suite de commandes délimitée par deux caractères spéciaux: « et ».

Tapons donc le programme...

Remarque: en cas de faute de frappe, la touche "←" permet d'effacer le caractère à gauche du curseur. En cas de faute grave, appuyer sur [ON] effacera tout ce que vous avez tapé (sans détruire le contenu de la pile).

- Pour commencer un programme, il faut un caractère spécial. Celui-ci s'obtient grâce à la séquence [↰] [-]. Comme vous l'avez remarqué, le caractère réciproque (») s'affiche lui aussi. L'écran ressemble alors à:

  et un curseur clignote à droite du "«". C'est là que vos prochains caractères vont apparaître...

- La première commande à effectuer est de placer 3 dans la pile, tapons donc [3] puis un espace ([SPC]) qui servira de séparateur...

- La seconde est y^x, appuyons donc sur cette touche... Surprise, ce n'est pas "y^x" qui s'affiche mais le symbole "^". Ce symbole signifie lui aussi "mise à la puissance".

Notre écran est alors:

Avec le curseur à droite de "^"...

- Notre programme est terminé, il ne reste plus qu'à le valider en tapant [ENTER]. L'écran sera alors:

Le programme est maintenant dans la pile, c'est le premier objet puisqu'il se trouve au niveau un...

Nous pourrions exécuter le programme en tapant [EVAL], mais cela nous conduirait à une erreur (puisque la pile ne contient pas assez de données) et nous perdrions le programme (une fois exécuté, il disparaîtrait de la pile).

Nous allons donc le stocker dans une variable:

['] [α] [α] [C] [U] [B] [E] [1] [ENTER] [STO]

Si maintenant vous appuyez sur la touche [VAR], vous devez voir "CUBE1" dans la case de gauche du menu... C'est votre programme !

Maintenant mettez un nombre dans la pile, appuyez sur [VAR] si vous ne l'avez pas déjà fait, appuyez sur la touche correspondant à CUBE1, le nombre dans la pile sera alors mis au cube en appuyant sur une seule touche au lieu de trois !

Il existe d'autres manières d'effectuer le calcul. En voici quelques-unes présentées comme tous les programmes de la bibliothèque (partie 3 de ce livre):

```
CUBE2 (# D649h)
«
  DUP DUP * *
»
```

```
CUBE3 (# E4F0h)
«
  → A
  «
    A A * A *
  »
»

CUBE4 (# 4526h)
«
  → A
  'A*A*A'
»
```

Ce listing s'interprète de la manière suivante:

- En caractères gras, se trouve le nom de l'objet;
- A côté du nom, entre parenthèses, se trouve la valeur du checksum de l'objet, qui permet de vérifier que ce dernier a été correctement entré (pour calculer ce checksum, placer le nom de l'objet dans la pile, par exemple 'CUBE2' et exécuter la fonction BYTES. Celle-ci renvoie deux valeurs: la valeur du checksum et la taille de l'objet. Le checksum est ici donné en hexadécimal, il faut donc se placer dans ce mode, par HEX, pour effectuer la comparaison);
- Dessous, jusqu'au nom suivant se trouve le listing de l'objet, c'est à dire l'aspect qu'il aura une fois tapé.

Pour entrer ces objets il faut donc:

- Entrer l'objet lui-même (comme nous l'avons fait pour CUBE1) et le mettre dans la pile (en le validant par [ENTER]);
- Mettre son nom dans la pile;
- Taper [STO].

Quelques remarques sur les quatre programmes:

- CUBE1 utilise la fonction interne déjà programmée: la puissance notée "^" qui prend deux arguments dans la pile: un réel et la puissance à laquelle vous voulez l'élever. CUBE1 s'occupe de mettre la puissance dans la pile (3), c'est à vous de spécifier le réel...
- CUBE2 utilise la pile. La fonction DUP duplique le niveau 1 de la pile (ce qui est très rapide comme toutes les fonctions de manipulation de pile). L'utilisation de

trois "DUP" permet d'obtenir 3 exemplaires de l'objet dans la pile, que l'on multiplie ensuite entre eux. Par exemple, si l'utilisateur lance 'CUBE2' sur la pile:

```
4:
3:
2:
1:                                5
```

Après le premier DUP on aura:

```
4:
3:
2:                                5
1:                                5
```

Après le second:

```
4:
3:                                5
2:                                5
1:                                5
```

Après la première multiplication:

```
4:
3:
2:                                5
1:                               25
```

Après la seconde:

```
4:
3:
2:
1:                              125
```

Ce qui est bien le cube de 5...

- CUBE3 utilise le concept de "variable locale". Nous avons déjà vu ce qu'était une variable lorsque nous avons stocké des objets. Une variable locale n'est visible que par le programme pour lequel elle est déclarée. Pour créer une telle variable on utilise le symbole "→" suivi de un ou plusieurs noms de variables puis d'un "«" qui signifie que la liste de noms s'arrête là. Cela va créer, pour la partie de programme entre "«" qui

suit et le "»" correspondant, les variables correspondates, en utilisant les valeurs qui étaient dans la pile; dans cette partie du programme toute utilisation du nom d'une de ces variables rappellera la valeur qu'elle a prise par "→".

Quelques remarques:

- "→" conserve l'ordre d'empilage: si la pile contient 5 au niveau 2 et 42 au niveau 1, la séquence "→ A B" placera 5 dans la variable A et 42 dans la variable B...
- Si une variable locale a le même nom qu'une autre variable, c'est le contenu de la variable locale la plus proche qui est utilisée. Par exemple dans le cas du programme:

  ```
  « 1 → A « 2 → A « A » » »
  ```

  On place 1 dans une première variable locale A, puis 2 dans une variable locale de même nom, alors, lors de l'utilisation de A, c'est la valeur 2 qui sera utilisée...
- Les variables locales auront nécessairement toutes disparues lorsque le programme se terminera (normalement, par erreur ou par interruption);
- Par opposition aux variables locales, qui ne sont visibles que localement, on parlera de variables globales à propos des variables du menu VAR, celles-ci étant visibles de partout...

• CUBE4 est semblable à CUBE3 mais au lieu de faire suivre "→ A" par un objet programme on le fait suivre par une expression algébrique qui joue le même rôle.

CUBE1 est le plus court mais si l'utilisateur oublie de mettre une donnée dans la pile, il obtiendra non seulement un message d'erreur "^ Error: Too Few Arguments", mais aussi l'apparition d'un 3 dans la pile ce qui n'est pas très "propre"... Au contraire les autres programmes commencent par une fonction qui teste la présence d'un objet dans la pile avant toute autre chose... En fait c'est le programme suivant:

```
CUBE (# C875h)
«
  → A
  'A^3'
»
```

qui est le plus court et le plus performant, mais aussi le plus correctement programmé. Il est possible d'écrire un tel programme, grâce à l'existence d'une fonction interne "^". En règle générale, pour faire vos programmes, vous aurez à choisir parmi les méthodes de CUBE2 ou CUBE3, tout en sachant que:

- CUBE2 est le plus rapide.
- CUBE3 est bien programmé car il utilise des variables locales pour stocker les entrées et la pile pour les calculs, mais plus lent que CUBE2 car le rappel d'une variable locale est plus lent que l'exécution d'un DUP.

Il faut surtout éviter au maximum le genre de programme suivant:

```
« 'A' STO A A * A * 'A' PURGE »
```

Très lent car il doit créer et détruire une variable globale, qui peut écraser des données préexistantes (si une variable globale A existe déjà) et peut laisser des traces s'il est interrompu (la variable globale A peut subsister). Cependant avoir recours à un tel style de programmation est parfois nécessaire...

II) Variables et arborescence

Nous avons vu qu'une variable locale est une variable qui n'est visible que d'une partie bien définie d'un programme, qui apparaît au début de l'exécution de cette partie et qui disparaît à la fin.

Nous avons vu qu'une variable globale est une donnée stockée dans le menu VAR ou dans un de ses menus-fils... Il est possible pour des variables de porter des noms identiques. En effet, on peut avoir des variables de même nom dans des menus utilisateur différents, ainsi que des variables locales...

Alors comment savoir quel contenu va être utilisé lorsqu'on appelle une variable ? Il suffit de savoir comment la HP48 recherche ce contenu:

- Première étape: elle recherche si une variable locale porte ce nom, en commençant par les variables les plus récemment crées;
- Ensuite, si elle n'a pas trouvé, elle regarde dans le menu VAR courant si la variable existe. Si tel est le cas, elle prend le contenu. Sinon, et si ce menu n'est pas HOME, elle passe au menu-père. Sinon la variable n'existe pas

et au lieu de prendre le contenu de la variable, la HP48 va utiliser le nom de celle-ci (variable mise entre "' ").

Cette capacité de la HP48 à gérer des variables locales permet une technique de programmation classique: la récursivité...

III) Récursivité

Il existe des problèmes mathématiques dits récursifs, c'est-à-dire qui se référencient eux-mêmes. Par exemple, le calcul de la valeur d'une fonction f en un point n peut être tel que:

- f(n)=g(f(n-1)) où g est une fonction connue et calculable;
- il existe un nombre n_0 tel que $f(n_0)$ est connu et vaut f_0.

Nous sommes parfaitement capables de calculer f(n), pour n quelconque supérieur à n_0 puisqu'il suffit d'appliquer plusieurs fois la première formule à $f(n_0)=f_0$ connu, puis à $f(f(n_0))$, puis à $f(f(f(n_0)))$...

Inversement on peut dire: pour calculer f(n), je suppose f(n-1) connu et je fais mon calcul, pour calculer f(n-1), je suppose f(n-2) connu et je fais mon calcul... Calculons par exemple la fonction factorielle. Nous savons que:

- factorielle(n) vaut n*factorielle(n-1);
- factorielle de 0 vaut 1.

Pour calculer factorielle n, nous écririons donc:

- si n vaut 0 je sais faire, c'est 1 !
- si n est plus grand que 0, je dois d'abord calculer factorielle(n-1) et le multiplier par n...

Ce qui se programme directement:

```
FACTORIELLE (# 83DBh)
« → N
  « IF
      N 0 ==
    THEN
      1
    ELSE
      N 1 - FACTORIELLE N *
    END
  »
»
```

Explication du programme:

- On commence par prendre une valeur dans la pile et on la place dans la variable locale N;
- On teste ensuite si N contient 0:
  - si tel est le cas, "on sait faire" et on renvoie la valeur 1 (qui correspond à factorielle(1)) dans la pile...
  - sinon on commence par demander le calcul de factorielle(N-1) que l'on multiplie ensuite par le contenu de N.

Pour bien comprendre le fonctionnement d'un programme récursif, il faut garder à l'esprit que lorsqu'un programme "s'appelle lui-même", c'est une copie de ce programme qui est exécutée, copie qui n'a rien à voir avec le premier...

Regardons par exemple le calcul de factorielle(2). Pour calculer ce nombre nous aurons besoin de la valeur de factorielle(1) donc de factorielle(0) que l'on sait enfin calculer...

Trois copies de 'FACTORIELLE' vont donc s'enchaîner... Observons-les:

Copie 1	Copie 2	Copie 3
C'est celle que l'on appelle avec la valeur 2 dans la pile: pour elle N contient le réel 2...		
N est différent de 0, elle doit donc utiliser factorielle(1), elle place 1 (2-1) dans la pile et appelle factorielle		
Elle attend la réponse... et N vaut toujours 2 pour elle.	Factorielle est lancée avec 1 dans la pile... factorielle(1) ? on ne sait pas faire on doit appeler factorielle pour 1-1=0 !	

Copie 1	Copie 2	Copie 3
Elle attend toujours... N continue à valoir 2.	Elle aussi attend	Factorielle est lancée, elle trouve 0 dans la pile... Elle sait faire et place 1 !
Devinez quoi, elle attend encore. Pourquoi voudriez-vous que N change ?	On trouve le résultat dans la pile (1) on le multiplie par N (1) d'où le résultat: 1	
Ca y est: le résultat de factorielle(1) est arrivé, on le multiplie par N (2), d'où le résultat (2)...		

Le principe est le même quelle que soit la valeur du premier N. Le voici par exemple résumé pour 5:

N=5, f(4)=?					
N=5,...	N=4, f(3)=?				
N=5,...	N=4,...	N=3, f(2)=?			
N=5,...	N=4,...	N=3,...	N=2, f(1)=?		
N=5,...	N=4,...	N=3,...	N=2,...	N=1, f(0)=?	
N=5,...	N=4,...	N=3,...	N=2,...	N=1,...	N=0, f(0)=1
N=5,...	N=4,...	N=3,...	N=2,...	N=1, f(0)=1 ⇒ f(1)=1	
N=5,...	N=4,...	N=3,...	N=2, f(1)=1 ⇒ f(2)=2		
N=5,...	N=4,...	N=3, f(2)=2 ⇒ f(3)=6			
N=5,...	N=4, f(3)=6 ⇒ f(4)=24				
N=5, f(4)=24 ⇒ f(5)=120					

D'où factorielle(5)=120...

Tout au long de ce chapitre, vous avez acquis quelques notions de base de programmation... A présent, il faut vous perfectionner: en essayant de bien comprendre le fonctionnement de petits programmes (ceux des manuels de la HP48 ou ceux de la bibliothèque de programmes de ce livre), ainsi qu'en en écrivant vous même... Voici donc quelques exercices:

Exercices:

A-4-1: Ecrire un programme additionnant deux réels pris dans la pile. Peut-il additionner deux chaînes de caractères ?

A-4-2: Que fait le programme suivant ?

```
« → A B « A B + A B * / » »
```

A-4-3: Ecrire un programme récursif calculant le $n^{ième}$ terme de la suite de Fibonacci U_n définie par:

- Si n est supérieur ou égal à 2, $U_n=U_{n-1}+U_{n-2}$;
- $U_0=U_1=1$.

Bien présenter ses données

Jusqu'à présent, nous avons découvert les capacités de calcul, de stockage et de programmation de la HP48. Mais savoir calculer, stocker des données ou écrire des programmes n'est pas suffisant...

En effet, la mémoire de la HP48 est très importante (32 Ko de base, jusqu'à 280 Ko pour la HP48sx munie de deux cartes 128 Ko, soit l'équivalent de plus de 200 pages de texte...). Il est donc important de bien organiser et de bien présenter ses programmes et données, de manière à pouvoir "s'y retrouver" plus tard.

Pour ce faire, il existe quelques techniques que nous allons étudier ici.

I) Faciliter l'accès aux données

Dans le chapitre 3, nous avons étudié l'arborescence des menus: c'est un des éléments essentiels d'une bonne présentation des données et programmes car cette arborescence nous permet de classer les différentes variables et programmes par affinité (tous les programmes mathématiques ensemble dans un menu 'MATHS', tous les programmes matriciels dans un sous-menu...).

De plus, au sein d'un même menu, nous pouvons classer les variables grâce à la fonction `ORDER`. Cette commande prend en argument une liste contenant les noms des variables dans l'ordre souhaité par l'utilisateur.

Ainsi, nous pourrons mettre en premier les programmes importants, suivis par leurs sous-programmes, moins utiles.

Il est aussi essentiel de bien choisir les noms, de manière à ce que la simple vision de l'intitulé d'une variable en évoque son contenu.

Cependant, il serait quelquefois utile d'associer un nom de fonction préexistante ou un dessin à un programme que nous venons de créer. Ceci est possible grâce au menu CST (touche à gauche de VAR).

Ce nouveau menu permet la création de liens entre des objets de la HP48 et un intitulé de touche de menu, sans consommation excessive de mémoire.

Le mécanisme de fonctionnement de ce menu est simple: lorsqu'on appuie sur la touche [CST], la HP48 recherche une variable 'CST' dont nous allons voir la structure. Si elle ne la trouve pas dans le menu courant, elle va explorer le menu-père et ainsi de suite. Si aucune variable 'CST' n'est trouvée, nous obtiendrions un menu vide sans utilité.

Il est donc possible d'avoir un menu CST par sous-menu de VAR et donc des menus CST adaptés au menu courant (d'où, encore une fois, l'intérêt de bien organiser ses données).

La variable CST doit contenir une liste. Pour chaque élément nous avons plusieurs possibilités:

- Il s'agit d'un nom: dans ce cas chaque touche du menu CST sera associée à la variable de ce nom;
- Il s'agit d'une chaîne de caractères: celle-ci sera mise dans la ligne de commande si l'on appuie sur la touche;
- Il s'agit d'une liste constituée de deux objets: dans ce cas le premier sera l'intitulé de la touche, et le second l'objet associé. Si le premier élément est un objet graphique de 21 colonnes et 8 lignes, alors la fonction sera représentée par le graphique correspondant...
- Tout autre objet sera exécuté. Son intitulé servira de label pour la touche correspondante.

Voici un exemple de menu CST. La règle de présentation des objets est toujours la même: le nom en gras, suivit du listing de l'objet. Pour le rentrer, il conviendra de taper la séquence ‹ ... › suivie de [ENTER] 'CST' STO.

```
CST (# 9D17h)
{ { "A" "Un " } { GROB 21 8
0000000400C10A00E08FFFF0EFFFF1F700C10CFF70000000 "avion " )
{ "in" "dans "} { "the" "le "} {"sky" "ciel "} "!" }
```

Après avoir stocké cet objet, passer en menu CST (en appuyant sur la touche [CST], à gauche de [VAR]). Amusant, non ? A présent, appuyez successivement sur les six touches de menu, de gauche à droite... Bravo ! Votre HP48 vient de réaliser une traduction anglais-français !

Ce menu nous permet donc d'associer des icônes à des fonctions, mais aussi de mélanger fonctions internes de la HP48 et fonctions-utilisateur...

Mais nous pouvons encore faire mieux: ce type d'assignation de fonctions à des touches peut se faire pour la totalité du clavier. Cette redéfinition des touches est alors globale.

Nous allons voir cette capacité de la HP48 à travers un exemple. Voici un petit programme qui joue un air de musique aléatoire:

```
«
  -56 CF 1 10
  START
    4400 RAND * .1 RAND * BEEP
  NEXT
»
```

Tapez-le. L'affichage de la pile est alors:

```
2:
1:    « -56 CF 1 10 START
      4400 RAND * .1 RAND
      * BEEP NEXT »
```

Tapez à présent: [5] [1] [ENTER] [A] [S] [N] [ENTER]

Passez à présent en mode 1USR (par [↰] [α]), puis appuyez sur [ENTER] vous entendez une petite musique !

L'explication est simple: nous avons assigné le programme à la touche [ENTER]. Cette assignation n'est valable que dans un mode particulier: le mode USER. Nous sommes temporairement passé dans ce mode par [↰] [α] (cette séquence fait passer en mode "1USR", mode "USER" actif pour une seule touche). Pour y passer durablement, il convient de taper [↰] [α] [↰] [α]. "USER" s'affiche alors en haut de l'écran. Pour revenir au mode normal: [↰] [α]. Remarque: les touches non redéfinies gardent leurs significations premières en mode USER.

Vous pouvez ainsi redéfinir tout le clavier, y compris la touche ON, pour peu que vous soyez en mode USER ou 1USR. La syntaxe de ASN est la suivante:

```
arg1 arg2 ASN
```

arg1 représente la fonction que l'on désire faire effectuer à la machine lors de l'appui de la touche. Ce peut être un nom de programme, un programme lui-même ou tout autre objet. Un nom particulier est prédéfini: 'SKEY' qui rend à la touche sa fonction standard.

arg2 est un réel qui se décompose ainsi:

- Chiffre des dizaines: numéro de la ligne de la touche (entre 1 et 9, 1 correspondant à la ligne du haut);
- Chiffre des unités: colonne de la touche (entre 1 et 6, 1 correspondant à la première colonne);
- Chiffre des dixièmes: le mode de la touche:
  - 0 ou 1 mode normal
  - 2 mode [↰] (shift-orange);
  - 3 mode [↱] (shift-bleu);
  - 4 mode [α] (alpha);
  - 5 mode [α] [↰] (alpha, shift-orange);
  - 6 mode [α] [↱] (alpha, shift-bleu);

Par exemple, pour remplacer DROP au clavier, il conviendra d'assigner une nouvelle fonction à la touche 56.2.

A noter que la séquence 0 DELKEYS remet toutes les touches dans leur état standard.

Les techniques que nous venons de voir permettent de faciliter l'accès aux données. Nous allons à présent voir comment en faciliter la compréhension...

II) Faciliter la compréhension

Plusieurs méthodes existent pour améliorer cette compréhension des programmes ou de leurs résultats.

Nous en avons retenu trois, importantes et faciles à mettre en œuvre:

- La HP48 permet de placer des commentaires qui commencent par le caractère "@" ([α] [↱] [ENTER]). Malheureusement ces commentaires disparaissent dès l'appui sur [ENTER]... Ils sont donc peu utiles, si ce n'est lorsque l'on stocke les programmes sur un autre ordinateur. Pour laisser des commentaires dans un programme de manière constante, on peut placer une séquence du type "commentaire" DROP, où "commentaire" est le texte désiré. Ce type de remarque restera dans le programme. Vous pouvez en particulier noter le but du programme, sa syntaxe (nombre et type d'arguments en entrée) et quels résultats il renvoie...
- Les messages: il est intéressant de signaler à l'utilisateur ce qui se passe: il est donc souhaitable d'inclure des messages d'erreur et des indications sur le déroulement du programme...
- Expliciter les résultats: quoi de plus difficile à utiliser que le résultat d'un programme lorsqu'on ne sait pas à quoi correspondent les résultats ? Pour simplifier leur lecture, il est utile de les "tagguer", c'est à dire de leur rajouter un prefixe (nom, commentaire...) non pris en compte par les fonctions de la HP48. Cette opération s'effectue à l'aide de la fonction →TAG qui prend en arguments l'objet à tagguer et son tag. Le programme μSOLVER de la bibliothèque de programme utilise cette technique.

La morale de ce chapitre est simple: il vous faut concevoir vos programmes comme si quelqu'un d'autre devait les utiliser. De cette manière, si quelque temps plus tard vous décidez de les reprendre, vous ne rencontrerez pas trop de difficultés...

Sauver et échanger des données

La mémoire dont vous disposez n'est pas infinie: en standard, elle est de 32Ko (32 Kilo octets soit environ 32000 caractères). De ce fait, il est nécessaire de pouvoir l'augmenter à l'aide de cartes mémoire. Deux ports sont prévus à cette effet, au dos de la machine, sous le cache du haut.

De plus, pourquoi retaper des données ou programmes déjà présents sur une autre HP48 ? Ceci est non seulement loin d'être amusant, mais conduit souvent à faire des erreurs. Il serait donc utile de pouvoir les échanger directement entre machines ou de les conserver sur un ordinateur.

Nous allons à présent étudier tout cela...

I) les cartes mémoire (HP48sx)

Elles sont de deux types: ROM ou RAM.

a) Les ROMs.

Les ROMs sont des mémoires que l'on ne peut que lire (Read Only Memory): les informations qu'elles contiennent ne peuvent être modifiées.

En fait, il existe quatre types de ROMs:

- Les ROMs pures, (comme celles contenues dans la HP48);
- Les PROMs ou ROMs programmables;
- Les EPROMs qui sont des PROM effaçables par rayons ultra-violets;
- Les EEPROMs qui sont des PROM effaçables électriquement.

Ce type de carte le plus souvent vendu préprogrammée. La carte HP-SOLVE est une carte de ce type. Moyennant l'achat d'un programmeur adéquat, on peut en réaliser soi-même (de type EPROM ou EEPROM). Cependant cette technique est coûteuse...

2) Les RAMs

Les RAMs sont des mémoires que l'on peut modifier. Les RAMS existantes pour la HP48 sont de 32 ou 128Ko. Sur ces RAMs, existe un petit sélecteur permettant de les protéger contre l'écriture (donc de les transformer en ROMs). Elles peuvent être utilisées de deux manières différentes:

- Comme une extension de votre mémoire interne grâce à la commande `MERGE`;
- Comme une disquette en mode `BACKUP`.

Il faut faire deux remarques importantes:

- Une carte en mode `MERGE` ne doit pas être protégée contre l'écriture;
- Une carte en mode `BACKUP` protégée contre l'écriture par son sélecteur n'est pas affectée par un 'memory lost';
- Si une carte se trouve dans l'un des ports, qu'elle n'est pas "mergée" et qu'aucune donnée n'y est encore stockée, vous obtiendrez le message `"Invalid Card Data"` à l'allumage, car la carte n'est pas encore configurée.

Pour mettre une carte en mode `MERGE`, éteignez la machine, insérez la carte dans l'un des deux ports à votre guise, allumez (`ON`), puis tapez `1 MERGE` ou `2 MERGE` selon si vous avez placé la carte dans le premier port (celui le plus au fond) ou dans le port 2. A présent faites `MEM`, vous verrez que votre mémoire disponible a considérablement augmenté.

Pour mettre une carte en mode `BACKUP`, insérez la carte dans un port, et stockez-y vos données directement. Les noms des objets d'un port ne sont pas de la forme `'nom'` mais des objets "taggués" de la forme `:x:nom` où x est le numéro du port (0, 1 ou 2). Lors de ce stockage, l'écriture doit être permise. En fait, nous vous conseillons de toujours laisser vos cartes "backup" en mode écriture interdite, sauf lorsque vous devez faire des sauvegardes.

Par exemple, si la carte est dans le port 2, "hello" :2:BONJOUR STO stockera la chaîne "hello" de nom BONJOUR dans le port 2.

II) HP48 <-> ordinateur, la RS232c

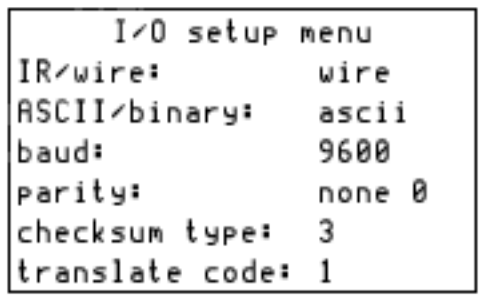
Hewlett-Packard commercialise un câble pour relier votre HP48 à un ordinateur Macintosh ou compatible PC. Un logiciel de transmission est fourni avec le câble, ce qui vous permet de récupérer sur votre disque dur ou sur une disquette, les données de votre HP48. Ce logiciel s'appelle KERMIT.

Pour l'utiliser, il faut le lancer sur votre ordinateur. Vous pouvez alors:

- Transmettre des données de la HP48 vers l'ordinateur. Pour cela, il faut taper:
  - sur la HP48:
    `'nom_de_la_variable_a_envoyer' SEND`
  - sur l'ordinateur: RECEIVE.
- Transmettre des données de l'ordinateur vers la HP48:
  - sur la HP48 lancer RECEIVE (menu I/O);
  - sur l'ordinateur:
    SEND nom_du_fichier_à_transmettre

Toutefois, pour tout échange, il faut s'assurer que les paramètres de transmission sont les bons. Il faut vérifier que les paramètres ont des valeurs correspondantes. Nous vous proposons une configuration:

- Sur la HP48, placez-vous dans le menu I/O et lancez SETUP. Grâce aux différentes touches de menu, amenez l'affichage à:



- Sur l'ordinateur, il convient de s'assurer que les réglages correspondent bien aux précédents... En particulier, sur IBM PC et compatibles, il faut taper les commandes suivantes (après chaque lancement de Kermit et avant la première transmission):

```
SET BAUD 9600
SET PORT 1
```

III) Communication par Infra Rouge

Deux HP48 peuvent s'échanger des données sans fil à une distance inférieure à 5 cm. Pour se faire, les deux machines doivent posséder le même SETUP. Par exemple:

```
     I/O setup menu
IR/wire:         IR
ASCII/binary:    ascii
baud:            9600
parity:          none 0
checsum type:    3
translate code:  1
```

Notez en particulier que le mode de transmission doit être IR (infra-rouge) et non plus wire (par câble) comme pour la liaison avec un ordinateur.

Placez les deux machines face à face, les petites flèches (au dessus du second 't' de "Hewlett-Packard") en vis-à-vis, faites `'nom_de_la_variable_à_transmettre' SEND` sur la machine émettrice, et `RECEIVE` sur l'autre. La donnée transmise sera stockée dans le répertoire courant.

Si le nom existe déjà, elle sera stockée sous un nouveau nom, de la forme `nom_précédent.1` (puis `nom_précédent.2` et ainsi de suite à chaque transfert du même nom), à moins que le flag -36 ne soit mis à un (par `36 SF` pour le mettre à un, `-36 CF` pour le remettre à 0), dans ce dernier cas, l'ancienne variable sera alors écrasée par la nouvelle.

ATTENTION: Si les piles sont trop faibles, les transmissions ne fonctionnent pas.

Les autres points forts de la HP48

La HP48 est avant tout une calculatrice scientifique dont nous allons voir un aperçu des capacités. Le but de ce chapitre n'étant pas d'expliciter les fonctions, mais de vous en faire connaître l'existence (de manière à ce que vous sachiez aller les chercher dans les manuels fournis avec la machine), elles ne seront pas détaillées...

I) Calcul symbolique

La HP48 permet d'effectuer du calcul "symbolique". Ce terme étrange, signifie qu'elle n'est pas limitée à la manipulation de valeurs approchées, mais qu'elle peut effectuer directement des opérations mathématiques complexes sur des expressions littérales... Parmi les manipulations possibles, nous pouvons citer:

- La dérivation: pour obtenir la dérivée d'une expression par rapport à une variable, il suffit de faire:

  `'expression' 'variable' ∂`

  Par exemple: `'SIN(X)/X' 'X' ∂` donne `'COS(X)/X-SIN(X)/X^2'`

  Attention: si une valeur est stockée dans la variable `'X'` du menu courant ou de un de ses pères, l'expression sera évaluée et nous obtiendrions une valeur approchée de la dérivée. Dans ce cas, il faut donc la supprimer (`'X' PURGE`) ou utiliser une autre variable dans l'expression algébrique.

- Développement limité: `'expression' 'var' n TAYLR`

  Où `'expression'` est la formule algébrique dont on veut le développement limité, `'var'` est la variable par rapport à laquelle on le veut et n, l'ordre du développement.

  Exemple: `'SIN(X)' 'X' 5 TAYLR` donne:

  `'X-1/3!*X^3+1/5!*X^5'`

Remarque: TAYLR se trouve dans le menu ALGEBRA (accessible par [↰] [9]).

- Résolution d'équation, recherche d'extrema, calcul de la valeur d'une fonction en un point grâce aux fonctions du menu SOLVE ([↰] [7]).

II) Calcul numérique

La HP48 possède de nombreuses fonctions de calcul numérique (dont la liste serait trop longue à évoquer).

La plupart de ces fonctions se trouvent dans le menu MTH et sont classées en six catégories: travail sur des parties de nombres (partie entière, fractionnelle...), probabilités, calcul hyperbolique, calcul matriciel, calcul vectoriel, calcul entier (en bases particulières).

De plus, des fonctions statistiques sont accessibles (menu STAT, [↰] [5]).

La HP48 utilise jusqu'à 12 chiffres significatifs pour vous rendre un résultat le plus précis possible (et jusqu'à 15 pour les calculs internes).

Si ce résultat obtenu doit être exprimé sous forme de fraction, la fonction →Q ([↰] [EVAL]) vous permet d'obtenir la fraction la plus proche du réel donné.

III) Graphisme

Le menu PLOT ([↰] [8]) vous présente toutes les fonctions nécessaires au tracé de courbes de toutes sortes (classique, conique, polaire, paramétrique...).

A noter que le graphique courant peut être visualisé et édité par simple appui sur [GRAPH] ([↰] [GRAPH]). On peut alors déplacer le curseur grâce aux quatre flèches, placer les coordonnées dans la pile en appuyant sur [ENTER] et revenir en mode normal par simple pression sur la touche [ON]. De nombreuses fonctions (zoom, déplacement de blocs, allumage ou extinction de points, tracé de lignes, de cercles, marquage de points...) sont aussi disponibles à travers les six touches du menu.

IV) Unités

La HP48 vous permet de faire vos calculs avec des unités. Pour créer un objet unité, il suffit de mettre à la suite d'un réel, le caractère underscore ("_" obtenu par [↱] [*]) suivi de l'unité désirée. Par exemple:

[1] [_] [m]

Vous pouvez aussi placer la valeur dans la pile, aller dans le menu UNITS ([↰] [6]) et choisir votre unité dans une des 16 catégories présentes (longueur, aire, volume, date et heure, vitesse, masse, force, énergie, puissance, pression, température, électricité, angles, lumière, radiation et viscosité).

[↱] [6] vous donne un autre menu d'UNITs dans lequel se trouve en particulier la fonction CONVERT qui permet la conversion entre unités.

V) Gestion du temps

Le menu TIME ([↰] [4]) vous donne accès à toute une série de fonctions de gestion du temps.

En particulier, vous pourrez définir des alarmes et effectuer des calculs sur les dates et les heures...

Notons que [↱] [4] vous donne un accès direct au catalogue des alarmes.

Conclusion

Ce qui vous a été présenté ici n'est qu'un aperçu des fantastiques possibilités de la HP48 ainsi que quelques astuces, de manière à vous permettre d'acquérir une vue d'ensemble de la machine.

Nous vous conseillons d'approfondir cette première approche en parcourant vos manuels et en utilisant votre machine le plus souvent possible: plus vous pratiquerez, plus cela vous semblera facile et vous apprendrez à résoudre très rapidement des problèmes longs et fastidieux.

Lorsque vous vous serez familiarisé avec l'utilisation de la machine telle qu'elle est décrite par Hewlett-Packard, vous vous émerveillerez devant ses incroyables atouts...

Mais ce n'est pas fini ! Au cours de la partie suivante, vous allez découvrir que vous pouvez encore en faire bien plus, grâce à la programmation en langage machine...

Deuxième partie

Le langage machine

Introduction

Dans cette partie intitulée "le langage-machine" nous apprendrons non seulement comment concevoir des programmes dans ce langage mais aussi comment est organisée la mémoire de la HP48.

En effet tout programmeur désirant aller aux limites de sa machine doit avoir une connaissance la plus étendue possible de sa structure, de manière à pouvoir accéder aux informations dont il a besoin et auxquelles le constructeur ne lui donne pas accès...

Cette visite guidée de la HP48 se fera en plusieurs étapes: du niveau le plus bas (le langage-machine à proprement parler, c'est-à-dire le seul langage que connaisse le processeur de la HP48), au niveau le plus haut (l'organisation de la mémoire) en passant par les objets qu'elle sait gérer.

En résumé nous verrons donc:

- Le langage-machine:
  - Qu'est ce que le langage-machine ?
  - Le langage-machine du microprocesseur de la HP48 (le Saturn);
  - Les instructions du langage-machine (classées par type de fonction).
- Les objets de la HP48, c'est-à-dire les entités que la machine est capable de gérer (objets 'classiques' auxquels l'utilisateur a accès, mais aussi objets internes non documentés par Hewlett-Packard);
- L'organisation de la mémoire de la HP48:
  - Mémoire en général;
  - Ram des entrées sorties, ou comment accéder directement au contraste, à l'horloge, à l'écran...
  - Mémoire vive réservée qui contient les informations internes de la HP48;

- Mémoire utilisateur qui contient les objets (programmes, variables...) créés par ce dernier.

- Comment programmer en langage-machine ?

Dans certains de ces chapitres sont présentés des tableaux décrivant la mémoire de la calculatrice. Des conventions ont été prises de manière à en assurer l'homogénéité.

Voici un tableau type:

adresse 1	Contenu 1	Longueur 1
adresse 2	Contenu 2	Longueur 2
adresse 3	Contenu 3	Longueur 3
adresse fin		

On a toujours...

- Une adresse est un nombre hexadécimal (base 16) qui correspond à la position dans la mémoire du contenu décrit dans la case contigüe (case encadrée du tableau); On a toujours `adresse 1` < `adresse 2` < `adresse 3`: le tableau se lit de haut en bas (comme un texte normal); la description des objets n'impliquant pas des adresses fixes, on utilisera le symbole @(quelquefois indicé, sous la forme @¡, si plusieurs adresses sont utilisées) pour indiquer l'adresse de début de l'objet; la dernière adresse (`adresse fin`) indique l'adresse du premier quartet suivant le dernier contenu étudié;

- La colonne centrale indique de manière succincte à quoi correspond le contenu de la zone mémoire spécifiée; ce champ est repris de manière détaillée dans le texte accompagnant le tableau, de manière à en spécifier la signification;

- Le champ 'longueur' (colonne de droite) indique, en décimal, le nombre de quartets (élément mémoire élémentaire de la HP48 ayant une valeur comprise entre 0 et F, donc stockée sur 4 bits, d'où son nom) du contenu étudié. Ainsi `longueur 1` est égal à `adresse 2 - adresse 1`. Ce champ pourra correspondre à une valeur spécifiée dans un des champs de l'objet: par exemple `longueur 3` pourrait être `contenu 2`.

Pour finir, voici quelques conseils de lecture.

Etant donné que le but de cette partie est double, à la fois didactique et de référence, le lecteur ne doit pas être impressionné par les informations qu'il y trouvera.

La lecture se fera en fait en deux temps: tout d'abord un survol rapide qui permettra au lecteur d'acquérir les différentes notions exposées.

Connaissant ces notions, le lecteur pourra ensuite utiliser cette partie comme un ouvrage de référence dans lequel il trouvera les informations précises dont il aura besoin pour réaliser des programmes performants en langage-machine.

Dernier point: le premier chapitre de cette partie est une approche générale du langage-machine et n'est pas intéressant pour le lecteur connaissant déjà ce type de langage.

Le langage machine

Si vous savez déjà ce que sont assembleur et langage-machine vous pouvez passer directement au chapitre suivant... Sinon il ne vous reste qu'à lire ce qui suit !

Pour expliquer ce qu'est le langage-machine, on peut le comparer au langage évolué auquel l'utilisateur a naturellement accès. Cette comparaison peut se faire au travers d'une petite histoire: celle de monsieur Durand et de monsieur Dupont.

Ces deux personnages désirent installer des prises de courant chez eux.

Monsieur Dupont n'est pas bricoleur et prend donc la solution la plus simple: celle de faire appel à un homme du métier... Il prend donc son téléphone et appelle l'électricien de son quartier. Quelque temps plus tard celui-ci vient (enfin) chez monsieur Dupont et réalise le travail demandé en contrepartie d'une somme conséquente (matériel + main d'œuvre + charges sociales + frais de déplacement + pourboire...). Monsieur Dupont paye de mauvais cœur car le travail réalisé ne correspond pas exactement à ses souhaits...

Monsieur Durand au contraire est habile de ses mains. Il décide donc de faire le travail lui-même. Pour cela il se rend chez son quincaillier où il achète une prise et du fil. Il rentre ensuite chez lui et pose ces derniers comme il le voulait, là où il le voulait... le tout pour une somme plus que modique.

On peut dire que, dans le premier cas, Monsieur Dupont a utilisé un langage évolué en donnant un ordre qui recouvre de nombreuses opérations élémentaires (chercher du fil, chercher une prise...) alors que Monsieur Durand a directement effectué ces opérations élémentaires.

Monsieur Durand a donc utilisé un langage de bas niveau, directement exécutable qui ressemble fortement au langage machine alors que monsieur Dupont lui a préféré un langage de haut niveau...

Cette petite histoire correspond bien à la comparaison de ces deux types de langages pour d'autres raisons:

- Appeler l'électricien est plus facile que réaliser le travail soi-même puisqu'il suffit de savoir donner des ordres !
- Cependant un langage de haut niveau est plus coûteux en temps (tout comme l'électricien est plus coûteux en argent);
- Le langage de haut niveau ne permet pas de faire exactement ce que l'on désire;
- Et même il n'est pas possible de tout demander dans un tel langage (avez-vous déjà demandé à un électricien de venir vous changer une lampe ?..).

Le langage-machine quant à lui permet l'accès direct à toutes les ressources de la machine et de manière très rapide mais est compliqué à utiliser puisqu'il se compose d'instructions très élémentaires (il faut donc un grand nombre de ces instructions, même dans le cas de la réalisation d'une fonction simple)...

En fait les deux langages sont complémentaires: les deux sont très utiles, les deux sont agréables à utiliser, mais ils ne permettent pas les mêmes choses...

Pour préciser un peu, ajoutons que du point de vue de la machine, le langage-machine est le seul langage reconnu (ainsi tout langage de haut niveau est en fait constitué d'appels à des programmes écrits dans ce langage).

Cependant, si il est très clair pour la machine, il est absolument illisible pour un être humain car il se compose d'une suite de chiffres...

C'est pourquoi on introduit un troisième langage: l'assembleur. Il s'agit d'une représentation symbolique de ces chiffres par des mnémoniques, c'est-à-dire de noms abrégés qui rappellent la fonction effectuée par l'instruction machine (on notera par exemple P=0 au lieu de 20).

Cependant la machine n'est pas à même de comprendre ces symboles. Il faut donc les transformer en une suite de nombres qui lui seront compréhensibles.

Ce passage de l'assembleur au langage-machine s'appelle l'assemblage, l'opération inverse se nomme, logiquement, le désassemblage. On commence donc par écrire le programme en

assembleur puis on effectue l'assemblage pour le rendre exécutable par la machine...

Il faut savoir que dans le cas de la HP48, on pourra soit faire l'assemblage à la main, soit de manière automatique en utilisant une machine annexe suffisamment puissante (Il existe au moins deux assembleurs Saturn: Areuh pour IBM PC et machines Unix, écrit par MM. Pierre DAVID et Janick TAILLANDIER et Satas pour Atari St, Amiga, IBM PC et machines Unix, écrit par M. Christophe DUPONT DE DINECHIN).

Un programme de désassemblage, qui fonctionne sur toute HP48, est proposé en annexe (dans la bibliothèque de programmes).

Dernier terme à expliciter: celui de "microprocesseur". Il s'agit en fait du cœur de la machine: l'entité électronique qui exécute les instructions en langage machine.

L'unité élémentaire d'information reconnue par le microprocesseur est le bit (c'est à dire la valeur 0 ou 1). Du fait de cette utilisation d'une base binaire par la machine, il est intéressant d'utiliser une base multiple de 2, en l'occurrence la base 16 ou hexadécimale.

Dans cette base on compte 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10... Ainsi la valeur 23h (le 'h' est mis pour rappeler qu'il s'agit d'une valeur hexadécimale) vaut 35 en décimal (16 * 2 + 3).

Cependant il peut être nécessaire de stocker des nombres décimaux. On utilise alors la notation "décimal codé binaire", pour laquelle on considère qu'un nombre hexadécimal est en fait décimal. Par exemple on considèrera que le nombre 15h est le nombre décimal 15.

Ce type de stockage conduit à prévoir deux modes de calculs pour le microprocesseur: le mode hexadécimal où les registres sont considérés comme contenant des nombres hexadécimaux, et le mode décimal où les registres sont supposés contenir des nombres en notation "décimal codé binaire".

Le mode en cours influe sur la manière dont les opérations mathématiques sont effectuées par le microprocesseur: en mode hexadécimal l'addition de 9 et 3 donnera Ch, alors qu'en mode décimal on obtiendra le nombre 12h, correspondant à la valeur décimale 12 en notation "décimal codé binaire"...

Le chapitre suivant présente les caractéristiques du microprocesseur de la HP48 ainsi que les caractéristiques de son langage-machine (la liste des instructions se trouve en chapitre 3, et une fiche de référence est proposée en annexe).

Exercices:

B-1-1: Coder les valeurs décimales 1, 10, 25, 65535 et 48830 en hexadécimal.

B-1-2: Exprimer en base décimale les valeurs hexadécimales 123h, 10h, 100h, B52h, 3h.

Le microprocesseur Saturn

Le microprocesseur de la HP48 est un Saturn 4 bits. Il s'agit sensiblement du même microprocesseur que celui du HP71 ou de la HP28.

I) Les registres

Le Saturn possède un jeu de 19 "registres" ("registre" est le nom d'une mémoire du microprocesseur. Un registre ne contient que des entiers positifs). Ces 19 registres peuvent être classés en 6 catégories.

- Les registres d'entrées/sorties;
- Les registres-drapeaux;
- Les pointeurs;
- Les registres de sauvegarde;
- Les registres de calcul;
- Le pointeur de champs.

a) Les registres d'entrées/sorties.

Ils sont deux:

- **OUTPUT** (12 bits). Il permet d'envoyer du courant sur un ou plusieurs des 12 fils du clavier et du buzzer. Ce registre ne peut être qu'écrit.
- **INPUT** (16 bits). Il permet de lire l'état des 16 entrées (provenant en particulier du clavier).

Ces deux "registres" sont utilisés pour l'émission d'un BEEP (écriture dans **OUTPUT**) ainsi que pour l'échantillonnage du clavier (on envoie du courant sur une ligne de touche et on regarde si ce courant ressort sur une colonne de touches ce qui permet de savoir si le bouton, situé à l'intersection ligne/colonne, est enfoncé ou non [s'il est enfoncé, le courant passe...]).

Le tableau ci-contre représente la correspondance entre les touches du clavier et les couples OUT/IN à utiliser pour les tester (toutes les valeurs sont hexadécimales): pour tester une touche, il suffit de faire le OUT correspondant, de récupérer IN et de faire un ET entre cette valeur et la valeur donnée dans le tableau. Si le résultat est non nul, cela signifie que la touche en question est enfoncée.

Il est possible de tester simultanément plusieurs touches en utilisant un masque de sortie constitué par l'addition logique (OR) de plusieurs masques. Attention: pour tester la touche `ON`, il convient d'interdire les interruptions (voir plus loin).

Pour émettre un "click" sur le buzzer, faire un OUT #800h suivi d'un OUT #0h..

Voici quelques exemples:

- Pour tester l'appui de la touche "A", il convient de faire un "OUT" de la valeur #002, de récupérer la valeur entrante par "IN" et de réaliser un OU logique avec le masque #0010. Ceci donne un morceau de programme du type:

```
LCHEX    #002     Masque de sortie
OUT=C
GOSBVL   #01160   réalise C=IN
LAHEX    #00010   Masque d'entrée
A=A&C    A
?A=0     A
GOYES    Touche_non_appuyée..
* la touche A est enfoncée
```

  Remarque: on utilise la routine en #01160h au lieu de l'instruction C=IN car cette dernière ne fonctionne pas correctement lorsqu'elle est exécutée en mémoire vive (elle corrompt la zone mémoire où elle se trouve). Une autre adresse utile est #01EECh qui réalise successivement les deux opérations OUT=C et C=IN.

- Tester l'appui d'une touche quelconque: le programme est le même que précédemment, le masque de sortie devenant #1FFh (#001h OR #002h OR #004h OR #008h OR #010h OR #020h OR #040h OR #080h OR #100h) et le masque d'entrée #001Fh (#0001h OR #0002h OR #0004h OR #0008h OR #0010h);
- Emettre un son: utiliser périodiquement et alternativement les masques de sortie #800h et #000h (activer le haut parleur et le désactiver).

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

Couples OUT / IN des touches du clavier

b) Les registres-drapeaux:

Ils sont au nombre de trois:

- **CARRY** (1 bit). C'est le bit de retenue; lorsqu'une opération donne lieu à une retenue, ce drapeau est armé: il peut ensuite être utilisé lors d'un test.
- **HST** (hardware status) (4 bits). C'est un registre de 4 drapeaux (**MP** module pulled, **SR** service request, **SB** sticky bit, **XM** external module missing).
- **STATUS** (16 bits). Il s'agit de drapeaux comme ceux accessibles par SF et CF (Attention: ce ne sont pas les mêmes !). Les drapeaux 12 à 15 sont utilisés de manière constante par la HP48: les drapeaux 0 à 11 sont donc à disposition de l'utilisateur pour la réalisation de programmes. On représente ce registre par **ST**.

c) Les pointeurs:

Ce sont des registres utilisés pour indiquer une zone mémoire particulière. Ils sont au nombre de trois:

- **D0** et **D1** (20 bits chacun). Ils sont utilisés pour la lecture et l'écriture en mémoire;
- **PC** (program counter - 20 bits). Il contient l'adresse de l'instruction en cours d'utilisation.

Ces registres ont une longueur de 20 bits. La HP48 est donc capable de gérer 2^{20} quartets soit 512 Kilo-octets...

d) Les registres de sauvegarde:

Ils sont de deux types:

- **RSTK** (return stack) (8 étages de 20 bits chacun): il s'agit d'une pile de 8 étages utilisée pour la sauvegarde d'adresses. Cette pile se comporte exactement comme la pile RPL de la HP48 avec toutefois une différence: même vide, elle contient des 00000. Elle sert à la sauvegarde d'informations et en particulier à la sauvegarde de l'adresse de retour lors de l'appel à une sous-routine.
- **R0 R1 R2 R3** et **R4** (64 bits chacun): ils sont utilisés pour la sauvegarde des registres de calcul.

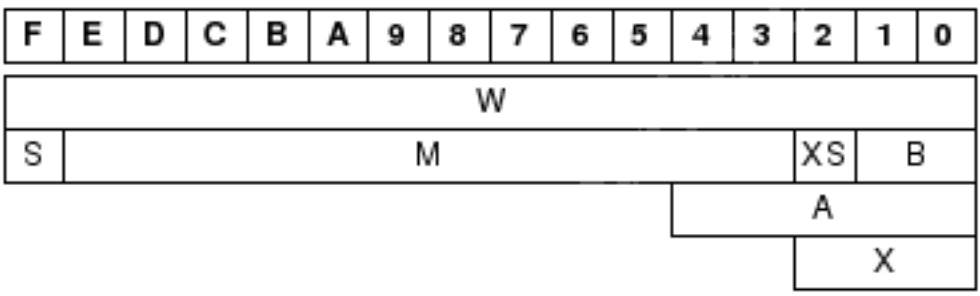
e) Les registres de calcul:

Ces quatres registres **A B C** et **D** (64 bits chacun) sont utilisés pour effectuer des calculs. **A** et **C** sont spécialement dédiés aux opérations de lecture et d'écriture en mémoire (ils sont alors utilisés avec **D0** et **D1**).

f) Le pointeur de champs:

Les registres **A B C** et **D** sont très longs (64 bits) et peu nombreux. Cependant ils sont divisés en zones plus petites appelées "champs". Ces champs, s'ils ne se recouvrent pas, peuvent être utilisés indépendamment les uns des autres. Ceci permet de conduire simultanément des calculs différents avec peu de registres occupés. Voici un tableau des champs:

numéro des quartets du registre



Ainsi le champ **M** représente les quartets E à 3, A les quartets 4 à 0, W la totalité du registre...

Pour se rappeler la signification de ces symboles, il suffit de savoir qu'ils représentent les initiales de mots anglais correspondant à l'origine à leur utilisation par le HP71:

- **A** comme **A**ddress car le champ A fait 5 quartets de long (qui est la longueur d'une adresse) et qui est prévu pour contenir des adresses;
- **B** comme **B**yte (octet) car deux quartets valent un octet;
- **M** comme **M**antisse (sur le 71 un réel était stocké dans un registre et se décomposait en son signe, sa mantisse, le signe de l'exposant et l'exposant);
- **S** comme **S**ign (pour la même raison que précédemment);

- **X** comme e**X**ponent (exposant signé);
- **XS** comme e**X**ponent-**S**ign (sur le HP71 l'exposant allait de 0 à 99: il fallait aussi stocker son signe...);
- **W** comme **W**ide.

Ces champs sont de longueur et de position fixes. Cependant il en existe deux autres, le champ **P** et le champ **WP** (**W**ide-**P**) dont la taille dépend du registre **P**.

Ce registre de 4 bits contient un nombre entre 0 et F. Le champ **P** est alors défini comme étant quartet de numéro **P**, **WP,** quant à lui, contient les quartets 0 à P (voir tableau ci dessus).

Il faut faire attention au fait que ce registre conditionne le chargement de constantes dans **A** et **C** (instructions LAHEX et LCHEX) comme décrit dans le chapitre 3.

Dans un programme en assembleur, on écrit le nom du champ impliqué dans l'opération après cette dernière. Par exemple `?C=0 A` signifie: "le champ **A** du registre **C** vaut-il zéro?".

Pour l'assemblage, on se trouve en présence de deux cas:

- soit l'opération existe directement avec le champ concerné (c'est toujours le cas pour le champ **A**, quelquefois pour le champ **B**), alors le code est donné directement;
- Sinon le code est donné avec une lettre minuscule (**a**, **f**, ou **b**) à remplacer par le quartet signifiant le champ donné, selon le tableau de correspondance ci-dessous.

Champ	a	f	b
P	0	0	8
WP	1	1	9
XS	2	2	A
X	3	3	B
S	4	4	C
M	5	5	D
B	6	6	E
W	7	7	F
A		F	

Ainsi, lorsque l'on rencontre, dans la liste des instructions une ligne du type:

```
Ab0      A=0 b
```

Pour A=0 W, on utilisera les codes AF0 (F pour **W** puisque le symbole est b).

Un dernier type de codage de champ est de définir le nombre de quartets sur lequel doit porter l'opération: dans la liste, ceci est symbolisé par **x**. Par exemple:

```
158x     DAT0=A x+1
```

signifie que l'opération se fera sur x+1 quartets pour le code 158x (ainsi 1583 signifie "effectuer l'opération DAT0=A pour les quartets 0...x de A"). Ce type d'opération est équivalent à l'utilisation d'un champ de type **WP** sans avoir à changer la valeur du registre **p**.

II) Notes diverses sur le Saturn

Le Saturn possède une particularité qu'il faut connaître: il "retourne" ce qu'il lit. Par exemple, sachant qu'en #00000 on trouve "2", qu'en #00001h on trouve "3", lire 2 quartets en #00000h conduira à trouver la valeur 32 dans le registre impliqué. De ce fait toutes les valeurs écrites en mémoires sont "à l'envers".

Cette particularité est valable pour tous les chargements de la mémoire vers les registres (lecture en mémoire ou chargement de constantes).

Les différentes instructions qui le composent peuvent être groupées selon certains critères.

Dans les Hardware internal Design Specifications du HP71 (HP00071-90071), elles sont classées par ordre alphabétique. Nous avons donc choisi de les présenter ici de deux manières complémentaires à celle de cet ouvrage:

- Par type de fonction réalisée (chapitre suivant) ce qui est commode lorsqu'on cherche à effectuer une opération particulière sans connaître sa syntaxe ni les registres auxquels elle s'applique;
- Par code (en annexe) et sous la forme d'une carte de référence (deux pages en vis-à-vis) qui servira d'aide-mémoire aux programmeurs connaissant déjà les

fonctions ou aux personnes désirant désassembler des programmes existants...

Dernier point à noter les registres utilisés par la HP48:

- **D0** qui pointe sur le prochain objet à exécuter (c'est pourquoi on termine toujours un programme en langage-machine par une lecture à cette adresse);
- **D1**, le pointeur de pile. Il pointe sur le premier étage de la pile. Lire 5 quartets à cette adresse renvoie l'adresse de l'objet au niveau 1;
- **B**, le pointeur de pile des retours. Lorsqu'on exécute des objets, il peut être nécessaire de stocker des adresses de retour. **B** pointe sur la prochaine case libre de la pile des retours (Attention: cette pile n'est pas le registre **RSTK**);

Ces registres sont utilisés par le système. Ils peuvent être utilisés dans des programmes en langage-machine, à condition que leur valeur soit restaurée en fin d'exécution.

Par contre, les drapeaux 12 à 15 sont utilisés en permanence par la HP48 (lors des interruptions). Il ne faut donc jamais les modifier. Notons en particulier le drapeau 15 qui permet de supprimer le traitement des interruptions liées au clavier.

Le drapeau 10, quant à lui, est utilisé par la HP48 lors des allocations-mémoire. Si on prend la précaution de mettre ce drapeau à zéro avant la tentative de réservation, celui-ci se retrouvera à 1 si un nettoyage de la mémoire (garbage collector) a dû être effectué.

Ces registres sont en effet utilisés par la HP48 pendant les interruptions (programme exécuté dans le cas d'événements extérieurs comme l'appui d'une touche ou la mise en place d'une carte mémoire).

Exercices

B-2-1: Comment se code le champ **W** pour les instructions:

```
Ba3      D=D-C    a
AbB      C=D      b
```

B-2-2: Mêmes questions pour les champs **P** et **WP.**

B-2-3: Sachant que:

```
Aa3      D=D+C    a
Ab3      D=0      b
```

désassembler les instructions machine `A13`, `A73`, `A83` et `A93`.

B-2-4: Sachant que #00321h contient "1", #00322h contient "1", #00323h contient "4", #00324h contient "C" et que #00325h contient "8", que contiendra un registre après lecture de 3 quartets en #00321h ?

B-2-5: Même question après lecture de deux quartets en #00322h.

B-2-6: Même question après lecture de 4 quartets en #00321h.

B-2-7: Si le registre A champ X contient 210h (2 dans le quartet de numéro 0, 1 dans le quartet de numéro 1 et 0 dans celui de numéro 0) et que l'on écrit cette valeur en #70080h, que contiendront #70080h, #70081h et #70082h ?

B-2-8: Si on relit ensuite 3 quartets à partir de #70080h dans C champ X, quelle sera la valeur contenue dans le champ X de ce registre ? Dans son champ B ? Dans son champ XS ?

B-2-9: Si **P** vaut 2, combien de quartets seront impliqués par l'instruction "A=DAT0 P" ?

Les instructions du Saturn

Voici le jeu complet des instructions du microprocesseur Saturn. Elles sont présentées par type de fonction réalisée.

Grâce à cette liste, le lecteur désireux de réaliser un programme en langage machine pourra facilement trouver les instructions dont il a besoin... Elles sont présentées dans l'ordre suivant:

- Affectations:
  - Chargements de constantes;
  - Chargements de valeurs;
  - Sauvegardes et récupérations (Rn et RSTK);
  - Transferts avec la mémoire;
  - Transferts avec l'extérieur;
- Echanges de contenus de registres;
- Opérations mathématiques:
  - Incrémentations;
  - Additions;
  - Décrémentations;
  - Soustractions;
  - "Et" logiques;
  - "Ou" logiques;
  - "Non" logiques;
  - Complémentations à 2;
  - Multiplications par 2;
  - Divisions par 2;
  - Multiplications par 16;
  - Divisions par 16;
  - Rotations des quartets vers la gauche;
  - Rotations des quartets vers la droite;

- Sauts:
    - Sauts relatifs inconditionnels;
    - Sauts relatifs conditionnels;
    - Sauts absolus fixes;
    - Sauts absolus par valeur;
    - Manipulations du compteur de programme.
- Appels de sous-programmes:
    - Appels relatifs;
    - Appels absolus;
    - Retours de sous-programmes.
- Tests:
    - Comparaisons avec des constantes;
    - Comparaisons entre registres.
- Commandes du Bus;
- Instructions de contrôle;
- Instructions sans effet;
- Pseudo-opérations.

I) Affectations

a) Chargements de constantes

On peut charger des constantes dans certains des registres du Saturn. Il existe notamment des mises à zéro. Voici, par registre, les chargements possibles:

- Pour **A**:
    - Mise à zéro de **A** champ **A**:
      A=0 A D0
    - Mise à zéro d'un autre champ:
      A=0 b Ab0
    - Mise à zéro du bit x de **A**. Le numéro de ce bit est compris entre 0 et F. Cette instruction n'a donc d'effet que sur les 4 premiers quartets du registre A:
      ABIT=0 x 8084x

- Mise à un du bit d de **A**. Il s'agit de l'opération inverse de la précédente:
  ABIT=1 x 8085x
- Chargement d'une valeur dans **A**. On spécifie le nombre x de quartets à charger. Les quartets en question sont h_0 à h_x. La valeur de **P** influe sur le chargement: le quartet h_0 va dans le quartet n° **P** de **A**, h_1 dans le n° **P**+1... Etant donné que le Saturn "retourne" les quartets lus lors d'un accès mémoire; le code est:
  LAHEX(x) $h_x...h_0$ 8082x$h_0...h_x$

- Pour **B**:
  - Mise à zéro de **B** champ **A**:
    B=0 A D1
  - Mise à zéro d'un autre champ:
    B=0 b Ab1

- Pour **C**:
  - Mise à zéro de **C** champ **A**:
    C=0 A D2
  - Mise à zéro d'un autre champ:
    C=0 b Ab2
  - Mise à zéro du bit x ($0h \leq x \leq Fh$) de **C**:
    CBIT=0 x 8088x
  - Mise à un du bit x ($0h \leq x \leq Fh$) de **C**:
    CBIT=1 x 8089x
  - Chargement d'une valeur dans **C**:
    LCHEX #$h_x...h_0$ 3x$h_0...h_x$

- Pour **D**:
  - Mise à zéro de **D** champ **A**:
    D=0 A D3
  - Mise à zéro d'un autre champ:
    D=0 b Ab3

- Pour **P**:
  - Affecter la valeur n ($0 \leq n \leq Fh$) à **P**:
    P= n 2n

- Pour **D0**:
  - Affectation des 2 quartets de poids faible:
    D0=(2) qp 19pq
  - Affectation des 4 quartets de poids faible:
    D0=(4) srqp 1Apqrs
  - Affectation des 5 quartets:
    D0=(5) tsrqp 1Bpqrst
- Pour **D1**:
  - Affectation des 2 quartets de poids faible:
    D1=(2) qp 1Dpq
  - Affectation des 4 quartets de poids faible:
    D1=(4) srqp 1Epqrs
  - Affectation des 5 quartets:
    D1=(5) tsrqp 1Fpqrst
- Pour **HST**:
  - Mise à zéro de **XM**:
    XM= 0 821
  - Mise à zéro de **SB**:
    SB= 0 822
  - Mise à zéro de **SR**:
    SR= 0 824
  - Mise à zéro de **MP**:
    MP= 0 828
  - Mise à zéro des quatre:
    CLRHST 82F
- Pour **ST**:
  - Mise à zéro du drapeau d (0h ≤ d ≤ Fh) :
    ST=0 d 84d
  - Mise à zéro de tous les drapeaux:
    CLRST 08
  - Mise à un du drapeau d:
    ST=1 d 85d

b) Chargements de valeurs

- Pour **A**:
  - Chargement du champ A de B dans le champ A de A:

    A=B A D4
  - Idem pour un champ quelconque:

    A=B b Ab4
  - Idem pour C:

    A=C A DA

    A=C b AbA
- Pour **B**:
  - Chargement du champ A de A dans le champ A de B:

    B=A A D8
  - Idem pour un champ quelconque:

    B=A b Ab8
  - Idem pour B et C:

    B=C A D5

    B=C b Ab5
- Pour **C**:
  - Chargement du champ A de A dans le champ A de C:

    C=A A D6
  - Idem pour un champ quelconque:

    C=A b Ab6
  - Idem pour C et B:

    C=B A D9

    C=B b Ab9
  - Idem pour C et D:

    C=D A DB

    C=D b AbB
  - Chargement de P dans le quartet n de C:

    C=P n 80Cn
  - Chargement des drapeaux 0 à 11 de ST dans le champ X de C:

    C=ST 09

- Pour **D**:
  - Chargement du champ A de C dans le champ A de D:
    D=C A D7
  - Idem pour un champ quelconque:
    D=C b Ab7
- Pour **P**:
  - Chargement du quartet n de C dans P:
    P=C n 80Dn
- Pour **D0**:
  - Chargement de A champ A dans D0:
    D0=A 130
  - Chargement des quartets 0 à 3 de A dans D0:
    D0=AS 138
  - Idem avec C:
    D0=C 134
    D0=CS 13C
- Pour **D1**:
  - Chargement de A champ A dans D1:
    D1=A 131
  - Chargement des quartets 0 à 3 de A dans D1:
    D1=AS 139
  - Idem avec C:
    D1=C 135
    D1=CS 13D
- Pour **ST**:
  - Chargement du champ X de C dans les drapeaux 0 à 11 de ST:
    ST=C 0A

c) Sauvegardes et récupérations (Rn et RSTK)

- Pour **A**:
  - Sauvegarde de la totalité des quartets:
    R0=A 100
    R1=A 101
    R2=A 102
    R3=A 103
    R4=A 104

- Idem pour le champ A seulement:

R0=A	A	81AF00
R1=A	A	81AF01
R2=A	A	81AF02
R3=A	A	81AF03
R4=A	A	81AF04

- Idem pour un champ quelconque:

R0=A	a	81Aa00
R1=A	a	81Aa01
R2=A	a	81Aa02
R3=A	a	81Aa03
R4=A	a	81Aa04

- Récupération de la totalité des quartets:

A=R0	110
A=R1	111
A=R2	112
A=R3	113
A=R4	114

- Idem champ A seulement:

A=R0	A	81AF10
A=R1	A	81AF11
A=R2	A	81AF12
A=R3	A	81AF13
A=R4	A	81AF14

- Idem pour un champ quelconque:

A=R0	a	81Aa10
A=R1	a	81Aa11
A=R2	a	81Aa12
A=R3	a	81Aa13
A=R4	a	81Aa14

- Pour **C**:
  - Sauvegarde de la totalité des quartets:

R0=C	108
R1=C	109
R2=C	10A
R3=C	10B
R4=C	10C

- Idem pour le champ A seulement:

R0=C	A	81AF08
R1=C	A	81AF09
R2=C	A	81AF0A
R3=C	A	81AF0B
R4=C	A	81AF0C

- Idem pour un champ quelconque:

R0=C	a	81Aa08
R1=C	a	81Aa09
R2=C	a	81Aa0A
R3=C	a	81Aa0B
R4=C	a	81Aa0C

- Récupération de la totalité des quartets:

C=R0	118
C=R1	119
C=R2	11A
C=R3	11B
C=R4	11C

- Idem champ A seulement:

C=R0	A	81AF18
C=R1	A	81AF19
C=R2	A	81AF1A
C=R3	A	81AF1B
C=R4	A	81AF1C

- Idem pour un champ quelconque:

C=R0	a	81Aa18
C=R1	a	81Aa19
C=R2	a	81Aa1A
C=R3	a	81Aa1B
C=R4	a	81Aa1C

- Récupération du champ A de C dans RSTK:

C=RSTK	07

- Sauvegarde du champ A de C dans RSTK:

RSTK=C	06

c) Transferts avec la mémoire

- Pour **A**:
  - Lecture de 5 quartets pointés par D0 dans le champ A de A:

    A=DAT0 A 142
  - idem pour le champ B:

    A=DAT0 B 14A
  - idem pour un champ quelconque:

    A=DAT0 a 152a
  - idem pour x+1 quartets:

    A=DAT0 x+1 15Ax
  - idem pour une zone pointée par D1:

    A=DAT1 A 143

    A=DAT1 B 14B

    A=DAT1 a 153a

    A=DAT1 x+1 15Bx
  - Ecriture de 5 quartets pointés par D0 dans le champ A de A:

    DAT0=A A 140
  - idem pour le champ B:

    DAT0=A B 148
  - idem pour un champ quelconque:

    DAT0=A a 150a
  - idem pour x+1 quartets:

    DAT0=A x+1 158x
  - idem pour une zone pointée par D1:

    DAT1=A A 141

    DAT1=A B 149

    DAT1=A a 151a

    DAT1=A x+1 159x
- Pour **C**:
  - Lecture de 5 quartets pointés par D0 dans le champ A de C:

    C=DAT0 A 146
  - idem pour le champ B:

    C=DAT0 B 14E

- idem pour un champ quelconque:
  C=DAT0 a 156a
- idem pour x+1 quartets:
  C=DAT0 x+1 15Ex
- idem pour une zone pointée par D1:
  C=DAT1 A 147
  C=DAT1 B 14F
  C=DAT1 a 157a
  C=DAT1 x+1 15Fx
- Ecriture de 5 quartets pointés par D0 dans le champ A de C:
  DAT0=C A 144
- idem pour le champ B:
  DAT0=C B 14C
- idem pour un champ quelconque:
  DAT0=C a 154a
- idem pour x+1 quartets:
  DAT0=C x+1 15Cx
- idem pour une zone pointée par D1:
  DAT1=C A 145
  DAT1=C B 14D
  DAT1=C a 155a
  DAT1=C x+1 15Dx

d) Transferts avec l'extérieur

Ces instructions permettent de tester le clavier, ainsi que d'émettre des sons sur le buzzer de la HP48 (voir le chapitre précédent). Attention: les instructions A=IN et C=IN fonctionnent incorrectement (voir le chapitre précédent).

- Pour **A**:
  - Lecture des entrées (dans les 4 quartets 0,1,2 et 3 de A):
    A=IN 802
- Pour **C**:
  - Lecture des entrées (dans les 4 quartets 0,1,2 et 3 de C):
    C=IN 803
  - Ecriture de C champ X dans les sorties:
    OUT=C 801

- Ecriture du quartet 0 de C dans le quartet 0 du registre de sortie:
  OUT=CS 800

II) Echanges de contenus de registres

- Pour **A**:
  - Echange de A champ A et B champ A:
    ABEX A DC
  - Idem pour un champ quelconque:
    ABEX b AbC
  - Idem pour A et C:
    ACEX A DE
    ACEX b AbE
  - Idem pour tous les quartets de A et R0:
    AR0EX 120
  - Idem le champ A seulement:
    AR0EX A 81AF20
  - Idem un champ quelconque:
    AR0EX a 81Aa20
  - Idem pour A et R1:
    AR1EX 121
    AR1EX A 81AF21
    AR1EX a 81Aa21
  - Idem pour A et R2:
    AR2EX 122
    AR2EX A 81AF22
    AR2EX a 81Aa22
  - Idem pour A et R3:
    AR3EX 123
    AR3EX A 81AF23
    AR3EX a 81Aa23
  - Idem pour A et R4:
    AR4EX 124
    AR4EX A 81AF24
    AR4EX a 81Aa24

- Echange de A champ A et D0:
  AD0EX 132
- Idem pour les quartets 0 à 3 de A et D0:
  AD0XS 13A
- Idem pour A et D1:
  AD1EX 133
  AD1XS 13B

• Pour **B**:
- Echange de B champ A et B champ A:
  BAEX A DC
- Idem pour un champ quelconque:
  BAEX b AbC
- Idem pour B et C:
  BCEX A DD
  BCEX b AbD

• Pour **C**:
- Echange de C champ A et A champ A:
  CAEX A DE
- Idem pour un champ quelconque:
  CAEX b AbE
- Idem pour C et B:
  CBEX A DD
  CBEX b AbD
- Idem pour C et D:
  CDEX A DF
  CDEX b AbF
- Idem pour C et R0:
  CR0EX 128
- Idem le champ A seulement:
  CR0EX A 81AF28
- Idem un champ quelconque:
  CR0EX a 81Aa28
- Idem pour C et R1:
  CR1EX 129
  CR1EX A 81AF29
  CR1EX a 81Aa29

- Idem pour C et R2:

CR2EX		12A
CR2EX	A	81AF2A
CR2EX	a	81Aa2A

- Idem pour C et R3:

CR3EX		12B
CR3EX	A	81AF2B
CR3EX	a	81Aa2B

- Idem pour C et R4:

CR4EX		12C
CR4EX	A	81AF2C
CR4EX	a	81Aa2C

- Echange de C champ A et D0:

CD0EX	136

- Idem pour les quartets 0 à 3 de C et D0:

CD0XS	13E

- Idem pour C et D1:

CD1EX	137
CD1XS	13F

- Echange du quartet n de C avec P:

CPEX	n	80Fn

- Echange de ST (drapeaux 0 à 11) avec C champ X:

CSTEX	0B

• Pour **D**:

- Echange de D champ A et C champ A:

DCEX	A	DF

- Idem pour un champ quelconque:

DCEX	b	AbF

III) Opérations mathématiques

a) Incrémentations

Ces instructions modifient la valeur de la retenue (carry).

- Pour **A**:
  - Incrémentation de A champ A:
    A=A+1 A E4
  - Idem pour un champ quelconque:
    A=A+1 b Ba4
  - Incrémentation du champ A de A de x+1 (0h ≤ x ≤ Fh):
    A=A+x+1 A 818F0x
  - Idem pour un champ quelconque:
    A=A+x+1 a 818a0x
- Pour **B**:
  - Incrémentation de B champ A:
    B=B+1 A E5
  - Idem pour un champ quelconque:
    B=B+1 b Ba5
  - Incrémentation du champ A de B de x+1 (0h ≤ x ≤ Fh):
    B=B+x+1 A 818F1x
  - Idem pour un champ quelconque:
    B=B+x+1 a 818a1x
- Pour **C**:
  - Incrémentation de C champ A:
    C=C+1 A E6
  - Idem pour un champ quelconque:
    C=C+1 b Ba6
  - Incrémentation du champ A de C de x+1 (0h ≤ x ≤ Fh):
    C=C+x+1 A 818F2x
  - Idem pour un champ quelconque:
    C=C+x+1 a 818a2x

- Pour **D**:
  - Incrémentation de D champ A:
    D=D+1 A E7
  - Idem pour un champ quelconque:
    D=D+1 b Ba7
  - Incrémentation du champ A de D de x+1 (0h ≤ x ≤ Fh):
    D=D+x+1 A 818F3x
  - Idem pour un champ quelconque:
    D=D+x+1 a 818a3x
- Pour **P**:
  - Incrémentation de P:
    P=P+1 0C
- Pour **D0**:
  - Ajout de x+1 à D0:
    D0=D0+ x+1 16x
- Pour **D1**:
  - Ajout de x+1 à D1:
    D1=D1+ x+1 17x

b) Additions

Ces instructions modifient la valeur de la retenue (carry).

- Pour **A**:
  - Addition des champs A de A et B:
    A=A+B A C0
  - Idem pour un champ quelconque:
    A=A+B a Aa0
  - Idem pour A et C:
    A=A+C A CA
    A=A+C a AaA
- Pour **B**:
  - Addition des champs A de B et A:
    B=B+A A C8
  - Idem pour un champ quelconque:
    B=B+A a Aa8

- Idem pour A et C:
  B=B+C A C1
  B=B+C a Aa1

- Pour **C**:
  - Addition des champs A de C et A:
    C=C+A A C2
  - Idem pour un champ quelconque:
    C=C+A a Aa2
  - Idem pour C et B:
    C=C+B A C9
    C=C+B a Aa9
  - Idem pour C et D:
    C=C+D A CB
    C=C+D a AaB
  - Ajout de P+1 à C champ A:
    C+P+1 809

- Pour **D**:
  - Addition des champs A de D et C:
    D=D+C A C3
  - Idem pour un champ quelconque:
    D=D+C a Aa3

c) Décrémentations

Ces instructions modifient la valeur de la retenue (carry).

- Pour **A**:
  - Décrémentation de A champ A:
    A=A-1 A CC
  - Idem pour un champ quelconque:
    A=A-1 a BaC
  - Décrémentation du champ A de A de x+1 (0h ≤ x ≤ Fh):
    A=A-(x+1) A 818F8x
  - Idem pour un champ quelconque:
    A=A-(x+1) a 818a8x

- Pour **B**:
  - Décrémentation de B champ A:
    B=B-1 A CD
  - Idem pour un champ quelconque:
    B=B-1 a AaD
  - Décrémentation du champ A de B de x+1 (0h ≤ x ≤ Fh):
    B=B-(x+1) A 818F9x
  - Idem pour un champ quelconque:
    B=B-(x+1) a 818a9x
- Pour **C**:
  - Décrémentation de C champ A:
    C=C-1 A CE
  - Idem pour un champ quelconque:
    C=C-1 a AaE
  - Décrémentation du champ A de C de x+1 (0h ≤ x ≤ Fh):
    C=C-(x+1) A 818FAx
  - Idem pour un champ quelconque:
    C=C-(x+1) a 818aAx
- Pour **D**:
  - Décrémentation de D champ A:
    D=D-1 A CF
  - Idem pour un champ quelconque:
    D=D-1 a AaF
  - Décrémentation du champ A de D de x+1 (0h ≤ x ≤ Fh):
    D=D-(x+1) A 818FBx
  - Idem pour un champ quelconque:
    D=D-(x+1) a 818aBx
- Pour **P**:
  - Décrémenter P:
    P=P-1 OD
- Pour **D0**:
  - Décrémenter D0 de x+1:
    D0=D0- x+1 18x

- Pour **D1**:
  - Décrémenter D1 de x+1:
    D1=D1- x+1 1Cx

d) Soustractions

Ces instructions modifient la valeur de la retenue (carry).

- Pour **A**:
  - Soustraction du champ A de C au champ A de A:
    A=A-C A EA
  - Idem pour un champ quelconque:
    A=A-C a BaA
  - Soustraction du champ A de A au champ A de B et stockage du résultat dans A champ A:
    A=B-A A EC
  - Idem pour un champ quelconque:
    A=B-A a BaC
- Pour **B**:
  - Soustraction du champ A de A au champ A de B:
    B=B-A A E8
  - Idem pour un champ quelconque:
    B=B-A a Ba8
  - Idem pour B et C:
    B=B-C A E1
    B=B-C a Ba1
  - Soustraction du champ A de B au champ A de C et stockage du résultat dans B champ A:
    B=C-B A ED
  - Idem pour un champ quelconque:
    B=C-B a BaD
- Pour **C**:
  - Soustraction du champ A de A au champ A de C:
    C=C-A A E2
  - Idem pour C et D:
    C=C-D A EB
    C=C-D a BaB

- Idem pour un champ quelconque:
  C=C-A a Ba2
- Soustraction du champ A de C au champ A de A et stockage du résultat dans C champ A:
  C=A-C A EE
- Idem pour un champ quelconque:
  C=A-C a BaE

• Pour **D**:

- Soustraction du champ A de C au champ A de D:
  D=D-C A E3
- Idem pour un champ quelconque:
  D=D-C a Ba3
- Soustraction du champ A de D au champ A de C et stockage du résultat dans D champ A:
  D=C-D A EF
- Idem pour un champ quelconque:
  D=C-D a BaF

e) <u>"Et" logiques</u>

• Pour **A**:

- Entre les champs A de A et B:
  A=A&B A 0EF0
- Idem pour un champ quelconque:
  A=A&B a 0Ea0
- Idem entre A et C:
  A=A&C A 0EF6
  A=A&C a 0Ea6

• Pour **B**:

- Entre les champs A de B et A:
  B=B&A A 0EF4
- Idem pour un champ quelconque:
  B=B&A a 0Ea4
- Idem entre B et C:
  B=B&C A 0EF1
  B=B&C a 0Ea1

- Pour **C**:
  - Entre les champs A de C et A:
    C=C&A A 0EF2
  - Idem pour un champ quelconque:
    C=C&A a 0Ea2
  - Idem entre C et B:
    C=C&B A 0EF5
    C=C&B a 0Ea5
  - Idem entre C et D:
    C=C&D A 0EF7
    C=C&D a 0Ea7
- Pour **D**:
  - Entre les champs A de D et C:
    D=D&C A 0EF3
  - Idem pour un champ quelconque:
    D=D&C a 0Ea3

f) "Ou" logiques

- Pour **A**:
  - Entre les champs A de A et B:
    A=A!B A 0EF8
  - Idem pour un champ quelconque:
    A=A!B a 0Ea8
  - Idem entre A et C:
    A=A!C A 0EFE
    A=A!C a 0EaE
- Pour **B**:
  - Entre les champs A de B et A:
    B=B!A A 0EFC
  - Idem pour un champ quelconque:
    B=B!A a 0EaC
  - Idem entre B et C:
    B=B!C A 0EF9
    B=B!C a 0Ea9

- Pour **C**:
  - Entre les champs A de C et A:
    C=C!A A 0EFA
  - Idem pour un champ quelconque:
    C=C!A a 0EaA
  - Idem entre C et B:
    C=C!B A 0EFD
    C=C!B a 0EaD
  - Idem entre C et D:
    C=C!D A 0EFF
    C=C!D a 0EaF
- Pour **D**:
  - Entre les champs A de D et C:
    D=D!C A 0EFB
  - Idem pour un champ quelconque:
    D=D!C a 0EaB

g) "Non" logiques

Ces instructions modifient la valeur de la retenue (carry).

- Pour **A**:
  - Non logique pour le champ A de A:
    A=-A-1 A FC
  - Idem pour un champ quelconque:
    A=-A-1 b BbC
- Pour **B**:
  - Non logique pour le champ A de B:
    B=-B-1 A FD
  - Idem pour un champ quelconque:
    B=-B-1 b BbD
- Pour **C**:
  - Non logique pour le champ A de C:
    C=-C-1 A FE
  - Idem pour un champ quelconque:
    C=-C-1 b BbE

- Pour **D**:
  - Non logique pour le champ A de D:
    D=-D-1 A FF
  - Idem pour un champ quelconque:
    D=-D-1 b BbF

h) Complémentations à 2

Ces instructions modifient la valeur de la retenue (carry).

- Pour **A**:
  - Complément à 2 pour le champ A:
    A=-A A F8
  - Idem pour un champ quelconque:
    A=-A b Bb8
- Pour **B**:
  - Complément à 2 pour le champ A:
    B=-B A F9
  - Idem pour un champ quelconque:
    B=-B b Bb9
- Pour **C**:
  - Complément à 2 pour le champ A:
    C=-C A FA
  - Idem pour un champ quelconque:
    C=-C b BbA
- Pour **D**:
  - Complément à 2 pour le champ A:
    D=-D A FB
  - Idem pour un champ quelconque:
    D=-D b BbB

i) Multiplications par 2

- Pour **A**:
  - Multiplication par 2 de A champ A:
    A=A+A A C4
  - Idem pour un champ quelconque:
    A=A+A a Aa4

- Pour **B**:
  - Multiplication par 2 de B champ A:
    B=B+B A C5
  - Idem pour un champ quelconque:
    B=B+B a Aa5
- Pour **C**:
  - Multiplication par 2 de C champ A:
    C=C+C A C6
  - Idem pour un champ quelconque:
    C=C+C a Aa6
- Pour **D**:
  - Multiplication par 2 de D champ A:
    D=D+D A C7
  - Idem pour un champ quelconque:
    D=D+D a Aa7

j) Divisions par 2

Il s'agit d'un décalage vers la droite d'un bit sur l'ensemble du registre concerné, d'où la dénomination de l'instruction sous la forme:

<registre>SRB

les lettres SRB signifiant "Shift Right Bit". Le bit sortant est perdu, mais SB est mis à 1 si il était non nul. Le bit entrant est zéro.

- Pour **A**:
  - Sur la totalité des quartets:
    ASRB 81C
  - Pour le champ A:
    ASRB A 819F0
  - Pour un champ quelconque:
    ASRB a 819a0
- Pour **B**:
  - Sur la totalité des quartets:
    BSRB 81D
  - Pour le champ A:
    BSRB A 819F1
  - Pour un champ quelconque:
    BSRB a 819a1

- Pour **C**:
  - Sur la totalité des quartets:
    CSRB 81E
  - Pour le champ A:
    CSRB A 819F2
  - Pour un champ quelconque:
    CSRB a 819a2
- Pour **D**:
  - Sur la totalité des quartets:
    DSRB 81F
  - Pour le champ A:
    DSRB A 819F3
  - Pour un champ quelconque:
    DSRB a 819a3

k) Multiplications par 16

Il s'agit d'un décalage vers la gauche d'un quartet, d'où la dénomination de l'instruction sous la forme:

<registre>SL <champ>

les lettres SL signifiant "Shift Left". Le quartet sortant est perdu, mais SB est mis à 1 si il était non nul.

- Pour **A**:
  - Multiplication par 16 de A champ A:
    ASL A F0
  - Idem pour un champ quelconque:
    ASL b Bb0
- Pour **B**:
  - Multiplication par 16 de B champ A:
    BSL A F1
  - Idem pour un champ quelconque:
    BSL b Bb1
- Pour **C**:
  - Multiplication par 16 de C champ A:
    CSL A F2
  - Idem pour un champ quelconque:
    CSL b Bb2

- Pour **D**:
  - Multiplication par 16 de D champ A:

    DSL A F3
  - Idem pour un champ quelconque:

    DSL b Bb3

l) Divisions par 16

Il s'agit d'un décalage vers la droite d'un quartet, d'où la dénomination de l'instruction sous la forme:

<registre>SR <champ>

les lettres SR signifiant "Shift Right". Le quartet sortant est perdu, mais SB est mis à 1 si il était non nul.

- Pour **A**:
  - Division par 16 de A champ A:

    ASR A F4
  - Idem pour un champ quelconque:

    ASR b Bb4
- Pour **B**:
  - Division par 16 de B champ A:

    BSR A F5
  - Idem pour un champ quelconque:

    BSR b Bb5
- Pour **C**:
  - Division par 16 de C champ A:

    CSR A F6
  - Idem pour un champ quelconque:

    CSR b Bb6
- Pour **D**:
  - Division par 16 de D champ A:

    DSR A F7
  - Idem pour un champ quelconque:

    DSR b Bb7

m) Rotations des quartets vers la gauche

Il s'agit d'une rotation circulaire vers la gauche des quartets (le quartet 0h devient le quartet 1h, le 1h devient le 2h... le Fh devient le 0), d'où le nom de ces instructions:

<registre>SLC

SLC signifiant "Shift Left Circular".

- Pour **A**:
  ASLC 810
- Pour **B**:
  BSLC 811
- Pour **C**:
  CSLC 812
- Pour **D**:
  DSLC 813

n) Rotations des quartets vers la droite

Il s'agit d'une rotation circulaire vers la droite des quartets (le quartet 0h devient le quartet Fh, le 1h devient le 0h... le Fh devient le Eh), d'où le nom de ces instructions:

<registre>SRC

SLC signifiant "Shift Right Circular".

- Pour **A**:
  ASRC 814
- Pour **B**:
  BSRC 815
- Pour **C**:
  CSRC 816
- Pour **D**:
  DSRC 817

IV) Sauts

La longueur des sauts relatifs se calcule toujours de la même manière: on compte le nombre de quartets entre la fin de l'instruction de saut hors-longueur et le début de l'instruction où l'on veut sauter. Si il s'agit d'un saut en arrière, on code la valeur négative par complément à 2.

Par exemple si on a affaire au saut relatif GOTO dont le code est 6aaa (aaa étant le longueur du saut) qui commence à l'adresse $@_1$ et que l'on veut sauter à l'adresse $@_2$, on codera le saut:

- par $(@_2-(@_1+1))$ si c'est un saut en avant (1 est le nombre de quartets du code hors-longueur). Par exemple si $@_1$=#00123h et $@_2$=#00456h le saut devra se faire sur 332h quartets et on le codera par la séquence: 6**233** (ne pas oublier que le Saturn 'retourne' les données).
- par le complément à deux de $((@_1+1)-@_2)$ s'il s'agit d'un saut en arrière. Par exemple si $@_1$=#00456h et $@_2$=#00123h, le saut se fera sur 334h quartets, ce qui donnera en complément à deux: CCCh et le saut se codera par 6**CCC**.

Comme il y a complémentation à 2, on aura des limitations sur la longueur des sauts:

- Pour un codage sur 2 quartets à -80h +7Fh quartets;
- Pour un codage sur 3 quartets à -800h +7FFh quartets;
- Pour un codage sur 4 quartets à -8000h +7FFFh quartets.

Remarque: Dans les listings des programmes-assembleur on pourra utiliser des labels (ou étiquettes) pour indiquer les points d'arrivée des sauts.

a) Sauts relatifs inconditionnels

Ils sont classés par longueur de saut maximal croissante:

GOTO	abc	6cba
GOLONG	abcd	8Cdcba

b) Sauts relatifs conditionnels

- Tous les sauts suivant un test (voir le paragraphe VI);
- Sauts dépendants de la retenue (CARRY):
  - Saut si la CARRY est à zéro:
    GONC ab 5ba
  - Saut si la CARRY est à un:
    GOC ab 4ba

c) Sauts absolus fixes

GOVLNG abcde 8Dedcba

d) Sauts absolus par valeur

- Contrôlés par A:
  - Saut à l'adresse contenue dans A champ A:
    PC=A 81B2
  - Idem avec sauvegarde dans A champ A. L'adresse sauvegardée est celle de l'instruction suivante:
    APCex 81B6
  - Saut à l'adresse écrite dans les 5 quartets pointés par A (on lit 5 quartets à l'adresse contenue dans A champ A et on continue l'exécution à cette adresse):
    PC=(A) 808C
- Contrôlés par C:
  - Saut à l'adresse contenue dans C champ A:
    PC=C 81B3
  - Idem avec sauvegarde dans C champ A:
    CPCex 81B7
  - Saut à l'adresse écrite dans les 5 quartets pointés par C:
    PC=(C) 808E

e) Manipulations du compteur de programme

En plus des instructions précédemment citées qui permettent d'affecter une valeur au compteur de programme PC (saut relatif, saut par valeur...), il est possible de connaître la valeur courante de PC.

La valeur obtenue correspond à l'adresse de la prochaine instruction devant être exécutée.

- Chargement de PC dans A champ A:
  A=PC 81B4
- Chargement de PC dans C champ A:
  C=PC 81B5

V) Appels de sous-programmes

Les sous-programmes sont appelés grâce à une instruction d'appel. Le retour de sous-programme se fait grâce à des instructions de retour de sous-programme dont la liste est donnée au paragraphe c.

Les longueurs des appels relatifs se calculent différemment de celles des sauts relatifs. On compte en effet le nombre de quartets à partir de l'adresse du premier quartet après l'instruction de saut.

Par exemple pour un programme du type:

```
                Gosub @1
@2              * suite du programme...


@1              * petit sous programme utile...
```

la longueur du saut sera @1-@2.

Le principe est le même que dans le cas les sauts relatifs pour distinguer sauts en avant et sauts en arrière...

Remarque: Dans les listings assembleurs, on pourra utiliser des labels (ou étiquettes) pour indiquer les points d'arrivée des appels de sous-programmes.

a) Appels relatifs

Ils sont classés par longueur de saut maximal croissante:

GOSUB	abc	7bca
GOSUBL	abcd	8Edcba

b) Appels absolus

GOSBVL	abcde	8Fedcba

c) Retours de sous-programmes

- Retours inconditionnels:
  - Retour simple:
    RTN 01
  - Retour avec mise à zéro de la retenue:
    RTNCC 03
  - Retour avec mise à un de la retenue:
    RTNSC 02
  - Retour avec mise à un de XM:
    RTNSXM 00
  - Retour de routine de gestion d'interruptions:
    RTI 0F
- Retours conditionnels:
  - L'ensemble des instructions de tests permet d'effectuer des retours conditionnels de sous-programme (voir le paragraphe VI).
  - Retour si la carry est à un:
    RTNC 400
  - Retour si la carry est à zéro:
    RTNNC 500

VI) Tests

Les tests sont tous sous la forme:

```
? <registre> <comparateur> <registre ou constante> <champ>
```

Ils sont toujours suivis d'une instruction de saut (`GOYES`) ou de retour de sous-programme (`RTNYES`). Cette instruction qui vient à la suite du test se code sur deux quartets suivant la règle:

- 00 pour coder `RTNYES`;
- La valeur du saut en relatif sur deux quartets pour coder `GOYES`. La longueur du saut se compte à partir de l'adresse de stockage de l'instruction `GOYES` (voir le calcul des sauts relatifs dans le paragraphe IV).

Remarques:

- Ces instructions modifient la valeur de la retenue (carry). Elle est en effet mise à 1 si le test est vérifié;
- Les valeurs des registres étant des nombres positifs, ces comparaisons sont non-signées.

a) Comparaisons avec des constantes

- Pour **A**:
  - Le champ A de A est-il nul ?
    ?A=0 A 8A8
  - Idem pour un champ quelconque:
    ?A=0 a 9a8
  - Le champ A de A est-il non-nul ?
    ?A#0 A 8AC
  - Idem pour un champ quelconque:
    ?A#0 a 9aC
  - Le bit x (0h ≤ x ≤ Fh) de A est il à zéro ?
    ?ABIT=0 x 8086x
  - Le bit x (0h ≤ x ≤ Fh) de A est il à un ?
    ?ABIT=1 x 8087x
- Pour **B**:
  - Le champ A de B est-il nul ?
    ?B=0 A 8A9

- Idem pour un champ quelconque:
  ?B=0 a 9a9
- Le champ A de B est-il non-nul ?
  ?B#0 A 8AD
- Idem pour un champ quelconque:
  ?B#0 a 9aD

• Pour **C**:
- Le champ A de C est-il nul ?
  ?C=0 A 8AA
- Idem pour un champ quelconque:
  ?C=0 a 9aA
- Le champ A de C est-il non-nul ?
  ?C#0 A 8AE
- Idem pour un champ quelconque:
  ?C#0 a 9aE
- Le bit x (0h ≤ x ≤ Fh) de C est-il à zéro ?
  ?CBIT=0 x 808Ax
- Le bit x (0h ≤ x ≤ Fh) de C est-l à un ?
  ?CBIT=1 x 808Bx

• Pour **D**:
- Le champ A de D est-il nul ?
  ?D=0 A 8AB
- Idem pour un champ quelconque:
  ?D=0 a 9aB
- Le champ A de D est-il non-nul ?
  ?D#0 A 8AF
- Idem pour un champ quelconque:
  ?D#0 a 9aF

• Pour **HST**:
- XM est-il nul ?
  ?XM=0 831
- SB est-il nul ?
  ?SB=0 832
- SR est-il nul ?
  ?SR=0 834
- MP est-il nul ?
  ?MP=0 838

- Pour **P**:
  - P est-il égal à n ?
    ?P= n 89n
  - P est-il différent de n ?
    ?P# n 88n
- Pour **ST**:
  - Le drapeau n de ST est-il égal à 0 ?
    ?ST=0 n 86n
  - Le drapeau n de ST est-il égal à 1 ?
    ?ST=1 n 87n
  - Le drapeau n de ST est-il différent de 0 ?
    ?ST#0 n 87n
  - Le drapeau n de ST est-il différent de 1 ?
    ?ST#1 n 86n

b) Comparaisons entre registres

- Pour **A**:
  - Le champ A de A est-il égal à celui de B ?
    ?A=B A 8A0
  - Idem pour un champ quelconque:
    ?A=B a 9a0
  - Idem pour A et C:
    ?A=C A 8A2
    ?A=C a 9a2
  - Le champ A de A est-il différent de celui de B ?
    ?A#B A 8A4
  - Idem pour un champ quelconque:
    ?A#B a 9a4
  - Idem pour A et C:
    ?A#C A 8A6
    ?A#C a 9a6
  - Le champ A de A est-il inférieur ou égal à celui de B ?
    ?A<=B A 8BC
  - Idem pour un champ quelconque:
    ?A<=B a 9bC

- Le champ A de A est-il strictement inférieur à celui de B ?
  ?A<B A 8B4
- Idem pour un champ quelconque:
  ?A<B a 9b4
- Le champ A de A est-il supérieur ou égal à celui de B ?
  ?A>=B A 8B8
- Idem pour un champ quelconque:
  ?A>=B a 9b8
- Le champ A de A est-il strictement supérieur à celui de B ?
  ?A>B A 8B0
- Idem pour un champ quelconque:
  ?A>B a 9b0

• Pour **B**:

- Le champ A de B est-il égal à celui de A ?
  ?B=A A 8A0
- Idem pour un champ quelconque:
  ?B=A a 9a0
- Idem entre B et C:
  ?B=C A 8A1
  ?B=C a 9a1
- Le champ B de A est-il différent de celui de A ?
  ?B#A A 8A4
- Idem pour un champ quelconque:
  ?B#A a 9a4
- Idem entre B et C:
  ?B#C A 8A5
  ?B#C a 9a5
- Le champ A de B est-il inférieur ou égal à celui de C ?
  ?B<=C A 8BD
- Idem pour un champ quelconque:
  ?B<=C a 9bD
- Le champ A de B est-il strictement inférieur à celui de C ?
  ?B<C A 8B5

- Idem pour un champ quelconque:
  ?B<C a 9b5
- Le champ A de B est-il supérieur ou égal à celui de C ?
  ?B>=C A 8B9
- Idem pour un champ quelconque:
  ?B>=C a 9b9
- Le champ A de B est-il strictement supérieur à celui de C ?
  ?B>C A 8B1
- Idem pour un champ quelconque:
  ?B>C a 9b1

• Pour **C**:
- Le champ A de C est-il égal à celui de A ?
  ?C=A A 8A2
- Idem pour un champ quelconque:
  ?C=A a 9a2
- Idem entre C et B:
  ?C=B A 8A1
  ?C=B a 9a1
- Idem entre C et D:
  ?C=D A 8A3
  ?C=D a 9a3
- Le champ A de C est-il différent de celui de A ?
  ?C#A A 8A6
- Idem pour un champ quelconque:
  ?C#A a 9a6
- Idem pour C et B:
  ?C#B A 8A5
  ?C#B a 9a5
- Idem pour C et D:
  ?C#D A 8A7
  ?C#D a 9a7
- Le champ A de C est-il inférieur ou égal à celui de A ?
  ?C<=A A 8BE

- Idem pour un champ quelconque:
  ?C<=A a 9bE
- Le champ A de C est-il strictement inférieur à celui de A ?
  ?C<A A 8B6
- Idem pour un champ quelconque:
  ?C<A a 9b6
- Le champ A de C est-il supérieur ou égal à celui de A ?
  ?C>=A A 8BA
- Idem pour un champ quelconque:
  ?C>=A a 9bA
- Le champ A de C est-il strictement supérieur à celui de A ?
  ?C>A A 8B2
- Idem pour un champ quelconque:
  ?C>A a 9b2

• Pour **D**:

- Le champ A de D est-il égal à celui de C ?
  ?D=C A 8A3
- Idem pour un champ quelconque:
  ?D=C a 9a3
- Le champ A de D est-il différent de celui de C ?
  ?D#C A 8A7
- Idem pour un champ quelconque:
  ?D#C a 9a7
- Le champ A de D est-il inférieur ou égal à celui de C ?
  ?D<=C A 8BF
- Idem pour un champ quelconque:
  ?D<=C a 9bF
- Le champ A de D est-il strictement inférieur à celui de C ?
  ?D<C A 8B7
- Idem pour un champ quelconque:
  ?D<C a 9b7

- Le champ A de D est-il supérieur ou égal à celui de C ?
  ?D>=C A 8BB
- Idem pour un champ quelconque:
  ?D>=C a 9bB
- Le champ A de D est-il strictement supérieur à celui de C ?
  ?D>C A 8B3
- Idem pour un champ quelconque:
  ?D>C a 9b3

VII) Commandes du Bus

Ces commandes sont très mal connues car peu documentées dans les HDS du HP71 publiées par Hewlett-Packard.

- Commandes:
  - Commande "B":
    BUSCB 8083
  - Commande "C":
    BUSCC 80B
  - Commande "D":
    BUSCD 808D
  - Commande de Reset à tous les chips:
    RESET 80A
  - Commande d'arrêt des chips:
    SHUTDN 807
  - Commande de déconfiguration: déconfigure le module situé à l'adresse contenue dans C champ A:
    UNCNFG 804
  - Commande de configuration: copie le champ A de C dans le registre de configuration du module courant (premier module non configuré sur le bus). Cette commande est en général effectuée juste après un UNCNFG. C'est à l'aide de ces deux commandes que s'effectue le déplacement de la Ram utilisateur pour laisser libre accès à la Rom cachée (voir les chapitres sur la mémoire). Il

faut noter que les mémoires de 32 Ko ou plus nécessitent une double configuration. La première correspond à la taille du module en complément (#100000 - taille en quartet), ce qui permet de n'utiliser qu'une partie du module, le second correspondant à l'adresse de départ. Ainsi le déplacement de la Ram interne de #70000h en #F0000h s'effectue par un UNCNFG sur #70000h, puis par un double CONFIG en #F0000h, la remise en mode normal par une déconfiguration en #F0000h, suivie d'une configuration en #F0000h puis en #70000h.

CONFIG 805

- Demande d'identification du module courant. Cet identificateur est stocké dans le champ A de C:

  C=ID 806

- Recherche du service demandé par un module sur le bus. Le résultat est stocké dans le quartet 0 de C, 1 bit par type de demande.

  SREQ? 80E

VIII) Instructions de contrôle

- Instructions de contrôle des interruptions:
  - Autorisation des interruptions masquables:
    INTON 8080
  - Interdiction des interruptions masquables:
    INTOFF 808F
  - Remise à zéro des interruptions:
    RSI 80810
- Contrôle du mode de calcul. Ces deux instructions influent sur le résultat des opérations mathématiques comme décrit dans le chapitre précédent:
  - Passage en mode de calcul décimal codé binaire:
    SETDEC 05
  - Passage en mode de hexadécimal:
    SETHEX 04

IX) Instructions sans effet

Afin de réserver de la place pour des ajouts futurs, il peut être intéressant d'utiliser des opérations sans effet (NOP=no operation).

On utilise conventionnellement les trois opérations suivantes, construites à partir de sauts relatifs:

NOP3	420
NOP4	6300
NOP5	64000

X) Pseudo-opérations

De manière à pouvoir insérer des valeurs fixes dans un programme (par exemple des prologues d'objets), on utilise une pseudo-instruction CON (constant) qui insère les quartets précisés en argument:

CON(n)	$q_1 \ldots q_n$	$q_n \ldots q_1$

Exercices

B-3-1: Assembler le programme suivant (ce programme ne réalise aucune fonction particulière, il a comme seul but de familiariser le lecteur avec la technique d'assemblage):

```
        CON(5)   #02DCC
debut   CON(5)   (fin)-(debut)

        GOTO     L1

sub1    A=A-1    A
        LCHEX    #12345
L2      C=C-1    A
        GONC     L2
        RTNCC

L1      LCHEX    #00005
        A=C      A
L3      GOSUB    L2
        ?A#0     A
        GOYES    L3

        LCHEX    #00001
        A=C      A
        GOXUB    L4
        ?A=0     A
        GOYES    L5
        A=A-1    A
L5      A=DAT0   A
        D0=D0+   5
        PC=(A)

L4      ?C=0     A
        RTNYES
        C=0      A
        A=A+1    A
        RTN
fin
```

B-3-2: A l'aide du tableau présenté en annexe, désassembler le code suivant:

```
14313 31791 57787 61557 13114 21648 08C
```

Les objets

La HP48 gère des entités appelées "objets". Il en existe 28 dont 17 ne sont indirectement accessibles à l'utilisateur (ceux indiqués par une étoile), voire totalement innacessible en standard (indiqués par deux étoiles).

Ils commencent tous par un prologue de 5 quartets qui indique leur nature. Voici les objets avec leur prologue et leur type (renvoyé par la fonction TYPE):

Prologue	Objet		Type
02911	System Binary	(**)	20
02933	Real		0
02955	Long Real	(**)	21
02977	Complex		1
0299D	Long Complex	(**)	22
029BF	Character	(**)	24
029E8	Array		3/4
02A0A	Linked Array	(**)	23
02A2C	String		2
02A4E	Binary Integer		10
02A74	List		5
02A96	Directory		15
02AB8	Algebraic		9
02ADA	Unit		13
02AFC	Tagged		12
02B1E	Graphic		11
02B40	Library	(**)	16
02B62	Backup	(*)	17
02B88	Library Data	(**)	26
02BAA	Reserved 1	(**)	27
02BCC	Reserved 2	(**)	27
02BEE	Reserved 3	(**)	27
02C10	Reserved 4	(**)	27
02D9D	Program		8
02DCC	Code	(**)	25
02E48	Global Name		6
02E6D	Local Name	(*)	7
02E92	XLIB name	(**)	14

Chacun de ces 28 objets possède une structure bien précise que nous allons étudier en détail. Chacun des objets sera présenté sous forme condensée par un tableau dont les différents éléments seront explicités dans le texte.

Lors de la lecture, il faut garder à l'esprit le fait que le microprocesseur Saturn "retourne" les valeurs lues et que de ce fait les différentes valeurs sont écrites à l'envers en mémoire et en particulier les prologues (ainsi le prologue 02911 sera en fait écrit 11920 dans la mémoire de la HP48).

Remarque: la représentation mémoire est indépendante du mode d'affichage (binaire/octal/décimal/hexadécimal...).

I) System Binary

@	Prologue (02911)	5 quartets
@+5h	Contenu	5 quartets
@+Ah		

Le "system binary" est un entier court (5 quartets) qui est utilisé par la HP48 de manière interne.

Il apparaît à l'écran sous la forme <XXXXXb> où XXXXX est le contenu et b la base courante. Il peut en particulier servir au passage de paramètres entre deux programmes indépendants.

Exemples:

- 1192000000 est le system binary <00000h>;
- 1192054321 est le system binary <12345h>;

Exercices:

B-4-1: Que représente 1192012345 ?

B-4-2: Coder le system binary <ABCDEh>;

B-4-3: Même question pour le system binary <123d>.

II) Real

@	Prologue (02933)	5 quartets
@+5h	Exposant	3 quartets
@+8h	Mantisse	12 quartets
@+14h	Signe	1 quartet
@+15h		

Il s'agit du réel usuel auquel l'utilisateur a accès. Pour le coder, on le sépare en trois partie: son signe, sa mantisse, nombre compris entre 1 (inclus) et 10 (exclus), et son exposant c'est-à-dire la puissance de 10 par laquelle il faut multiplier la mantisse pour ré-obtenir le réel: Réel = Signe * Mantisse * $10^{Exposant}$

En représentation interne ces trois éléments sont codés de la manière suivante:

- Si l'exposant est négatif, on le remplace par "1000 - exposant" afin d'obtenir un nombre positif (de ce fait les exposants de la HP48 sont compris entre -499 et +499, puisque la machine doit distinguer les exposants positifs négatifs). Ce nombre (compris entre 0 et 999) est stocké en Décimal Codé Binaire sur 3 quartets;
- La mantisse, rendue entière par multiplication par 10^{11} est stockée en Décimal Codé Binaire sur 12 quartets;
- Le signe est codé sur 1 quartet par 0 = '+' et 9 = '-'.

Exemples

12345.6789 est codé 339204000009876543210

-3.14159265359E-2 est codé 339208999535629514139

Exercices

B-4-4: Coder le réel 12.

B-4-5: Que représente 339204000000000543779 ?

III) Long Real

@	Prologue (02955)	5 quartets
@+5h	Exposant	5 quartets
@+Ah	Mantisse	15 quartets
@+19h	Signe	1 quartet
@+1Ah		

Cet objet est utilisé de manière interne par la HP48 pour effectuer des calculs en grande précision. Le principe de codage est le même que pour le réel simple, l'exposant ayant une valeur comprise entre -49999 et +49999, et la mantisse ayant 15 chiffres significatifs.

Exemples

559200000097985356295141 30 représente le réel long:
3,1415926535897 (π).

Le réel -123E45678 est représenté (de manière étendue) par:
559208765400000000000003219

Exercices

B-4-6: Comment la HP48 coderait elle le réel long 1234567890123456 ?

B-4-7: Que représente 559208999900000000000000 19 ?

IV) Complex

@	Prologue (02977)		5 quartets
@+5h	Exposant 1	Partie	3 quartets
@+8h	Mantisse 1	Réelle	12 quartets
@+14h	Signe 1		1 quartet
@+15h	Exposant 2	Partie	3 quartets
@+18h	Mantisse 2	Imaginaire	12 quartets
@+24h	Signe 2		1 quartet
@+25h			

C'est l'objet classique auquel l'utilisateur a accès. Sa structure est simple: après les 5 quartets du prologue, on trouve deux réels privés de leur prologue, le premier constituant la partie réelle du complexe, le second sa partie imaginaire.

Exemple

Le complexe (123456789012,210987654321) est codé:

77920 110 210987654321 0 110 123456789012 0

Exercices

B-4-8: Coder le complexe (1,2).

B-4-9: Quel est le complexe suivant ?

77920100000000000033910000000000000330

V) Long Complex

@	Prologue (0299D)		5 quartets
@+5h	Exposant 1	Partie	5 quartets
@+Ah	Mantisse 1	Réelle	15 quartets
@+19h	Signe 1		1 quartet
@+1Ah	Exposant 2	Partie	5 quartets
@+1Fh	Mantisse 2	Imaginaire	15 quartets
@+2Eh	Signe 2		1 quartet
@+2Fh			

Le complexe long possède une structure similaire au complexe, les deux réels de ce dernier ayant fait place à deux réels longs.

Exemple

Le complexe long (123456789012345,543210987654321) est codé:

D9920 11000 543210987654321 0 11000 123456789012345 0

Exercices

B-4-10: Coder le complexe étendu (0,0).

B-4-11: Que représentent les codes ci-dessous ?

D992000000543210987654321911000123456789012345 9

VI) Character

@	Prologue (029BF)	5 quartets
@+5h	Caractère	2 quartets
@+7h		

Il s'agit d'un nombre compris entre 0 et 255 (00h et FFh), ce qui correspond au domaine de valeur des codes ASCII étendus (codage des caractères).

Exemple

FB92014 est le caractère 'A' ('A' est de code ASCII 41h).

Exercices

B-4-12: Coder le caractère 'C' (de code ASCII 43h).

B-4-13: Que représente FB92044 ?

VII) Array

@	Prologue (029E8)	5 quartets
@+5h	Longueur totale hors-prologue l_t	5 quartets
@+Ah	Type des objets (de longueur l_o)	5 quartets
@+Fh	Nombre d de dimensions	5 quartets
@+14h	Dimension 1 (d_1)	5 quartets
@+d*5+14h	Dimension d (d_d)	5 quartets
@+d*5+19h	Contenu objet 1	l_o quartets
	Contenu objet d_2+1	l_o quartets
@+l_t-l_o+5h	Contenu objet d_1*...*d_d	l_o quartets

@+l_t+5h

L'objet array est utilisé pour stocker vecteurs et matrices. En fait, son usage peut être généralisé.

En effet, juste après la longueur totale hors-prologue de l'objet en question, on trouve une indication sur le type d'objets contenus.

Ce type (codé sur 5 quartets) correspond au prologue des objets contenus (la matrice est donc forcément homogène, c'est à dire qu'elle ne contient que des objets de même type).

Seconde généralisation: le nombre de dimensions. Habituellement 1 ou 2 (1 pour les vecteurs, 2 pour les matrices), on peut l'étendre à volonté...

Viennent ensuite des indications sur les dimensions (pour une matrice, ce seront les nombres de lignes et de colonnes).

Après cette partie déclarative, on trouve les valeurs stockées dans l'objet.

Ces valeurs sont des objets privés de leur prologue (qui serait inutile puisque le prologue est précisé dans la partie déclarative de la matrice). Ils sont rangés dans l'ordre des dimensions (par exemple, pour une matrice à deux dimensions, on remplit la ligne 1 puis la ligne 2 puisque la première dimension d'une matrice est son nombre de lignes).

Il faut cependant noter que si tous les objets matriciels envisageables (matrice de dimension 25 contenant des vecteurs par exemple) sont possibles à créer, ils ne seront pas forcément utiles dans la mesure où la HP48 ne saura pas les gérer correctement.

Exemple

La matrice [[1 2] [3 4]] est codée en:

8E920 95000 33920 20000 20000 20000 000000000000001 0 000000000000002 0 000000000000003 0 000000000000004 0

Exercices

B-4-14: Donner les 35 premiers quartets d'une matrice de system binary de dimensions 3x5x8.

B-4-15: Quels est le type des éléments de la matrice dont le code débute par la séquence ci-dessous ?

8E92010F00C2A20100009100052000...

VIII) Linked Array

@	Prologue (02A0A)	5 quartets
@+5h	Longueur totale hors-prologue l_t	5 quartets
@+Ah	Type des objets (de longueur l_o)	5 quartets
@+Fh	Nombre d de dimensions	5 quartets
@+14h	Dimension 1 (d_1)	5 quartets
@+d*5+14h	Dimension d (d_d)	5 quartets
@+d*5+19h	Pointeur sur objet 1	5 quartets
	Pointeur sur objet d_2+1	5 quartets
	Pointeur sur objet d_1*...*d_d	5 quartets
	Elément 1	l_o quartets
@+l_t-l_o+5h	Elément n	l_o quartets
@+l_t+5h		

Les "linked array" sont des matrices (voir pages précédentes) où les éléments ont été remplacés par des offsets (5 quartets) sur des objets (privés de prologue) situés en fin de matrice.

La nullité d'un offset indique l'absence d'élément dans la case correspondante.

Cette structure permet un stockage économique en terme de place mémoire dans le cas de matrices comportant un grand nombre d'éléments identiques. Ainsi dans l'exemple ci-contre la matrice identité d'ordre 2 sera-t-elle stockée en 82 quartets au lieu de 94...

Exemple

Voici la matrice identité d'ordre 2:

A0A20 D4000 33920 20000 20000 20000 41000 F1000 A1000 50000 0000000000000010 0000000000000000

IX) String

@	Prologue (02A2C)	5 quartets
@+5h	Longueur totale hors-prologue l_t	5 quartets
@+Ah	Premier caractère	2 quartets
@+l_t-2h	Dernier caractère	2 quartets
@+l_t+5h		

Il s'agit des chaînes de caractères (objets du type de "Hello World") bien connues des utilisateurs.

Son codage est simple: la longueur totale hors-prologue suivie par la liste des codes ASCII des caractères dans leur ordre d'apparition.

Exemple

"STRING" se code: C2A20 11000 35 45 25 94 E4 74

Exercices

B-4-16: Coder la chaîne de caractères "Hello World".

B-4-17: Décoder l'objet:

C2A203100024271667F60212

X) Binary Integer

@	Prologue (02A4E)	5 quartets
@+5h	Longueur totale hors-prologue l_t	5 quartets
@+Ah	Valeur de l'entier	l_t-5 quartets

@+l_t+5h

Il s'agit de l'entier classique. La longueur usuelle est au maximum de 15h (ce qui correspond à un entier hexadécimal de 16 chiffres, c'est-à-dire à une taille de mots de 64 bits), mais peut en fait être étendue à volonté (d'ailleurs la HP48 utilise de tels grands entiers de manière interne).

Exemple

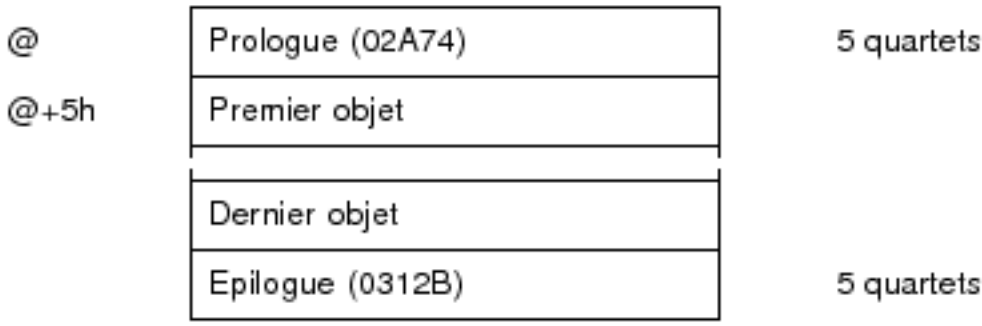
#12345678h est codé en E4A20 51000 8765432100000000

Exercices

B-4-18: Coder #87654321d.

B-4-19: Décoder E4A20A000012345.

XI) List



La liste est un objet composite. Elle contient une collection d'objets quelconques. De ce fait sa structure est très élémentaire: un prologue, les objets mis bout à bout et un épilogue (on peut considérer que le prologue joue le rôle de "{", et l'épilogue le rôle de "}").

Exemple

{ "A" B } se code 47A20 C2A20700004 84E201024 B2130

Exercices

B-4-20: Coder une liste vide.

B-4-21: Décoder 47A2084E2020F4B4B2130.

XII) Directory

@	Prologue (02A96)	5 quartets
@+5h	Nombre de librairies attachées n|	3 quartets
@+8h	N° Librairie 1	3 quartets
@+Bh	Adresse Hash Table 1	5 quartets
@+10h	Adresse Liste Messages 1	5 quartets
	N° Librairie n|	3 quartets
	Adresse Hash Table n|	5 quartets
	Adresse Liste Messages n|	5 quartets
$@_1$	Offset sur dernier objet ($@_d$-$@_1$)	5 quartets
$@_1$+5h	00000	5 quartets
$@_2$	Nombre n_1 de caractères du nom_1	2 quartets
$@_2$+2h	Caractère 1 nom_1 (nom	2 quartets
	Caractère n_1 objet 1)	2 quartets
	Nombre n_1 de caractères du nom_1	2 quartets
	Objet 1	
$@_3$	Taille zone précédente ($@_3$-$@_2$)	5 quartets
$@_3$+2h	Nombre n_2 de caractères du nom_2	2 quartets

@d	Nombre n_d de caractères du nom_d		2 quartets
@d+2h	Caractère 1	nom_d (nom	2 quartets
	Caractère n_d	objet d)	2 quartets
	Nombre n_d de caractères du nom_d		2 quartets
	Objet d		

Le directory est l'objet qui contient les variables utilisateur.

Il comporte aussi des indications sur les librairies qui lui sont attachées.

Il existe deux types de directories différents: le HOME-directory qui constitue le répertoire principal du menu VAR auquel peuvent être attachées un nombre quelconque de librairies et l'objet directory constituant un sous-menu...

Voyons tout d'abord le premier d'entre eux, représenté par le tableau ci-contre.

Le premier champ après le prologue indique le nombre de librairies attachées.

Vient ensuite un champ descriptif pour chacune des librairies attachées.

Ce champ est divisé en trois parties qui sont respectivement:

- Le numéro de la librairie. Ces numéros sont attribués selon les critères définis par Hewlett-Packard:
  - 000h à 100h lib. HP en Rom;
  - 101h à 200h lib. HP en Ram;
  - 201h à 300h lib. non HP (distribués par HP);
  - 301h à 6FFh utilisation libre;
  - 700h à 7FFh usage interne par la HP48.
- L'adresse de la "hash-table" de la library (voir l'objet "library");

- L'adresse de la liste des messages de la library (voir l'objet "library"). Ce pointeur est nul lorsqu'il n'y a pas de table des messages.

Il y a deux remarques à faire:

- Le home-directory est toujours attaché à 2 librairies au minimum: les librairies 002h et 700h;
- Lorsque les deux pointeurs font référence à des tables situées en Rom cachée (voir le chapitre 5), l'adressage est indirect: l'adresse correspond à un system binary situé en Rom normale qui contient l'adresse de l'objet dans la Rom cachée.

Ce début est différent dans le cas d'un sous directory... Pour ce dernier, la structure est la suivante:

@	Prologue (02A96)	5 quartets
@+5h	Numéro de la librairie attachée	3 quartets
@+8h	Offset sur dernier objet ($@_d$-$@_1$)	5 quartets
@+Dh	00000	5 quartets
@+12h	Nombre n_1 de caractères du nom_1	2 quartets
	⋮	

Le numéro de la librairie vaut #7FFh si aucune n'est attachée, et vaut le numéro de la librairie attachée sinon.

La suite est la même pour les deux types de directories:

On trouve ensuite un offset qui permet de déterminer la position du dernier objet du directory.

Vient ensuite une zone de 5 quartets nuls puisque l'objet suivant est le premier (ceci sert lorsque l'on parcourt le directory du dernier objet vers le premier).

Chaque variable du directory est ensuite décrite par les champs suivants:

- Nombre de caractères du nom (sur 2 quartets);
- Caractères du nom (en code ASCII);
- Nombre de caractères du nom (sur 2 quartets);
- L'objet;
- La longueur totale des 4 zones précédentes (sauf pour le dernier objet). Ce champ permet le parcours du directory depuis le dernier objet vers le premier.

Exemple

Voici le listing d'un directory vide: 69A20 FF7 00000

Directory ne contenant que "3" dans "D":

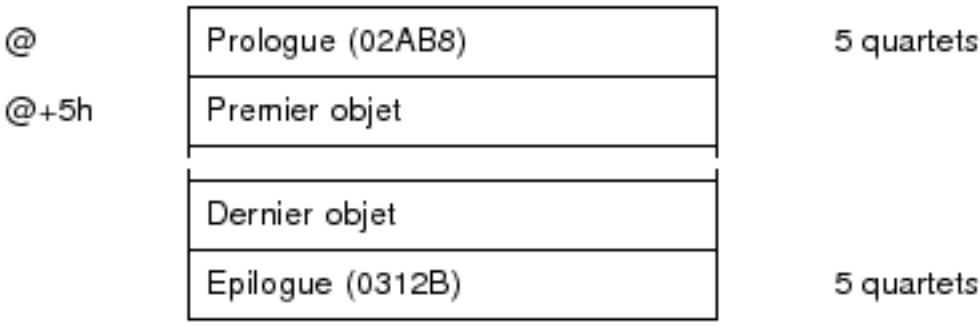
69A20 FF7 A0000 00000 10 44 10 C2A207000033

Exercice

B-4-22: Ajouter la variable 'A' contenant "4" au directory précédent.

B-4-23: Attacher la librairie 123h dont la table de hash-codes est en 7FE30h et sans table des messages au directory précédent.

XIII) Algebraic



L'expression algébrique représentée par cet objet est stockée sous forme de calcul en notation polonaise inversée.

De cette manière il n'y a pas stockage de parenthèses.

Les opérations sont codées par l'adresse en Rom (sur 5 quartets) de la fonction les réalisant.

Exemple

'C+D' est codé sous la forme "C D +" par

8BA20 84E201034 84E201044 76BA1 B2130

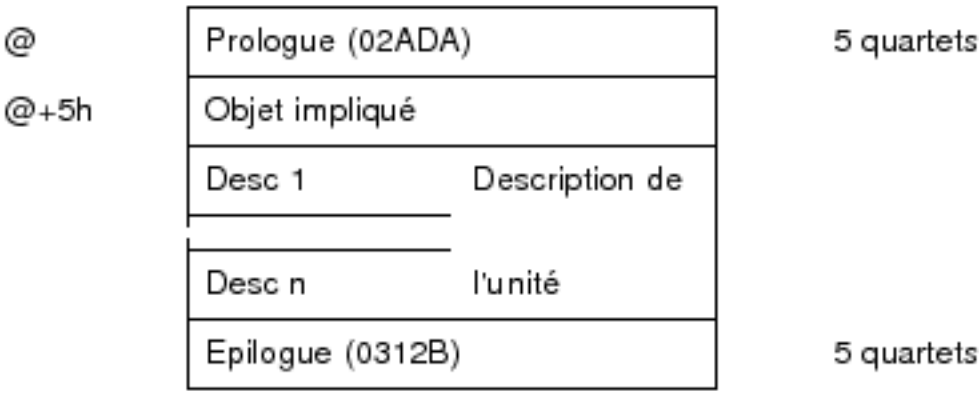
Exercice

B-4-24: Coder l'expression 'A+B'.

B-4-25: Sachant que la routine de soustraction se trouve en #1AD09h et celle de multiplication en #1ADEEh, décoder l'objet:

8BA2084E20101484E20102484E20103490DA1EEDA1B2130

XIV) Unit



Après le classique prologue, on trouve l'objet impliqué par l'unité.

Il fait en fait partie d'un calcul en notation polonaise inversée qui décrit ses rapports avec l'unité.

Les unités élémentaires la constituant sont stockées sous forme d'objets-chaînes de caractères (voir la description de cet objet).

Seules 3 opérations de type multiplicatif sont possibles entre les unités (on ne crée pas d'unité en additionnant des joules à des secondes ou en soustrayant des grammes à des kilomètres !):

- Multiplication;
- Division;
- Elévation à une puissance.

Ces trois opérations sont représentées par une référence à un objet situé dans la mémoire morte (à accès direct) de la machine.

Pour coder ou décoder les objets unités, on a donc besoin d'un tableau de correspondance entre les opérations et le pointeur les représentant.

Ce tableau est présenté page suivante...

_	#10B86h
*	#10B86h
/	#10B68h
^	#10B72h

Tableau de conversion
des opérations

Exemple

9,81 m/s^2 est représenté par:

ADA20 339200000000000001890 C2A2070000D6...
...C2A207000037 339200000000000000020 27B01...
...86B01 68B01 B2130

(En fait, pour une telle unité, la HP48 remplace l'objet réel 2 par un pointeur sur un objet réel situé en mémoire morte).

Exercices

B-4-26: Coder 1,2 m.

B-4-27: Décoder
ADA203392000000000000150C2A2070000D63...
...F2A227B0168B01B2130

XV) Tagged

@	Prologue (02AFC)		5 quartets
@+5h	Longueur l_t du tag		2 quartets
@+7h	Carac. 1	Caractères	2 quartets
@+l_t *2+5h	Carac l_t	du tag	2 quartets
@+l_t *2+7h	Objet "taggué"		

Cet objet est utilisé pour représenter les objets "taggués". A la suite du prologue on trouve le nombre de caractères du tag, suivi par les codes ASCII du dit tag.

Vient ensuite l'objet impliqué.

Exemple

REEL:1,23456789012 est codé:

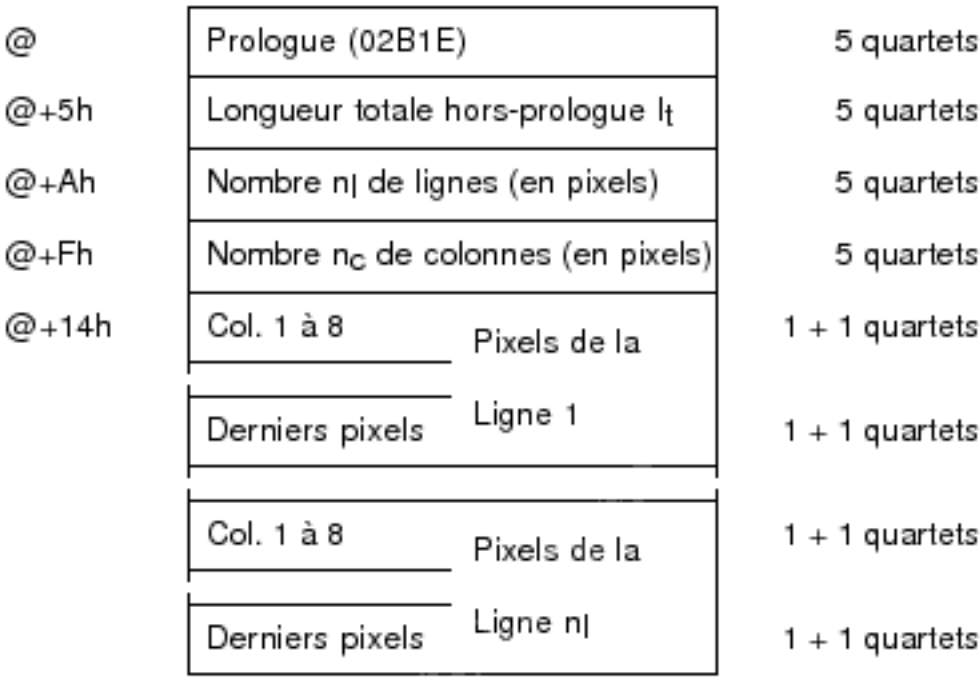
CFA20 40 255454C4 339200002109876543210

Exercices

B-4-28: Coder UN:TAG.

B-4-29: Décoder CFA2020F4B484E206034F4252514C4.

XVI) Graphic



L'objet graphique comporte l'indication de ses dimensions en pixels car il est stocké sur un nombre de colonnes multiple de 8. Pour les colonnes inexistantes, on stocke des bits 0.

Le premier quartet représente les 4 premières colonnes, le quartet qui suit les quatre suivantes et ainsi de suite... Pour un groupe donné de quatre colonnes, le bit de poids faible du quartet représente la colonne la plus à gauche, le bit de poids fort celle de droite.

Exemple

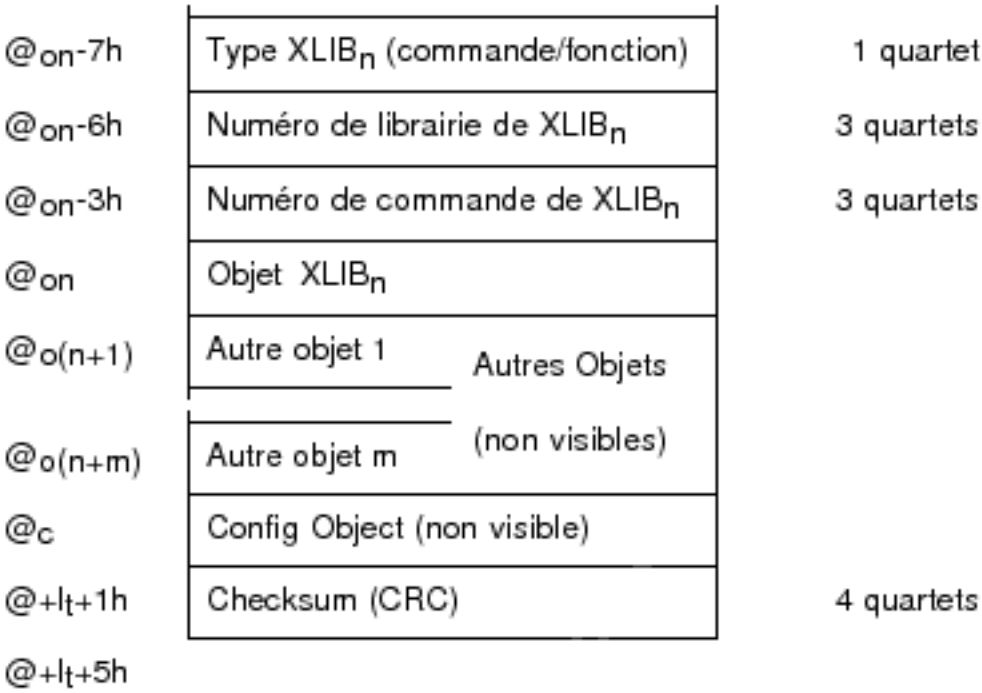
GROB 8 1 FF se code E1B20 11000 10000 80000 FF.

Exercice

B-4-30: Décoder E1B20110001000040000F0.

XVII) Library

@	Prologue (02B40)	5 quartets
@+5h	Longueur totale hors-prologue l_t	5 quartets
@+Ah	Nombre n_c de caractères du nom	2 quartets
@+Ch	Carac 1 (Caractères du nom)	(2 quartets)
@+n_c*2+Ah	Carac n_c	(2 quartets)
@+n_c*2+Ch	Nombre n_c de caractères du nom	(2 quartets)
@+n_c*2+Eh	Library number	3 quartets
$@_1$	Offset to Hash Table ($@_h$-$@_1$)	5 quartets
$@_2$	Offset to Message Array ($@_m$-$@_2$)	5 quartets
$@_3$	Offset to Link Table ($@_l$-$@_3$)	5 quartets
$@_4$	Offset to Config Object ($@_c$-$@_4$)	5 quartets
$@_h$	Hash Table	
$@_m$	Message Array	
$@_l$	Link Table	
$@_{o1}$-7h	Type $XLIB_1$ (commande/fonction)	1 quartet
$@_{o1}$-6h	Numéro de librairie de $XLIB_1$	3 quartets
$@_{o1}$-3h	Numéro de commande de $XLIB_1$	3 quartets
$@_{o1}$	Objet $XLIB_1$	



Cet objet est le plus complexe des objets de la HP48. Sa création se fait le plus souvent grâce à des logiciels sur machines plus puissantes (type PC ou station de travail).

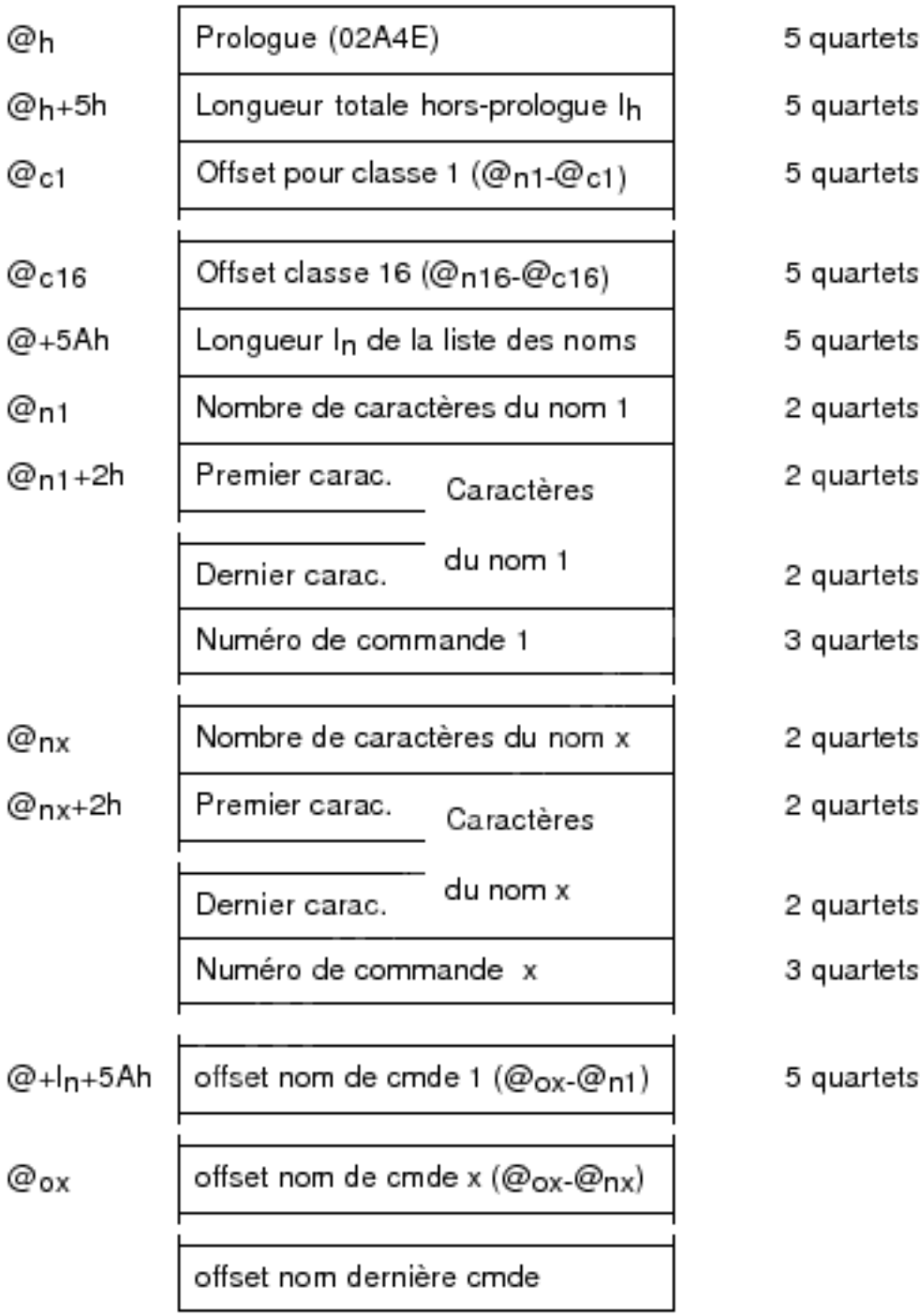
Il commence par son nom. Ce champ n'est pas indispensable. Dans le cas d'une librairie non nommée, les champs contenant les caractères et le second champ contenant la longueur sont absents. Vient le numéro d'identification de la librairie, qui doit être unique (voir la description de l'objet directory).

On trouve ensuite 4 offsets permettant de retrouver respectivement la hash-table, la message-table, la link-table (ces trois tables sont décrites plus loin) et l'objet de configuration (exécuté après chaque arrêt système). Un champ nul indique que la table ou l'objet n'existe pas. Viennent ensuite les 3 tables (si elles existent et dans un ordre quelconque).

On trouve ensuite les objets visibles de la librairie (précédés de leur numéro de librairie et de commande) puis les objets à usage interne (sous-programmes...), dont l'objet de configuration (qui est généralement le dernier).

Le checksum se calcule sur la zone @+5h à @+lt+1h selon la formule décrite avec l'objet Backup.

La structure de la hash-table est la suivante:



De manière à minimiser le temps d'accès aux fonctions d'une library, la HP48 utilise une technique de hash-coding.

Le principe est simple: on utilise une fonction qui renvoie un nombre (ici compris entre 1h et 10h) à partir du nom. Dans le cas de la HP48, cette fonction correspond au nombre de caractères du nom (1 pour 1 caractère ... 16 pour 16 et plus).

Pour chaque classe de nom, on réalise une table où sont stockés noms et informations s'y rattachant (ici le numéro de commande dans la librairie).

Pour retrouver un nom, il suffit de calculer le hash-code et de chercher dans la table indiquée. Statistiquement, si N est le nombre de noms, on aura à faire N/(2*16) comparaisons au lieu de N/2, d'où un gain de temps appréciable...

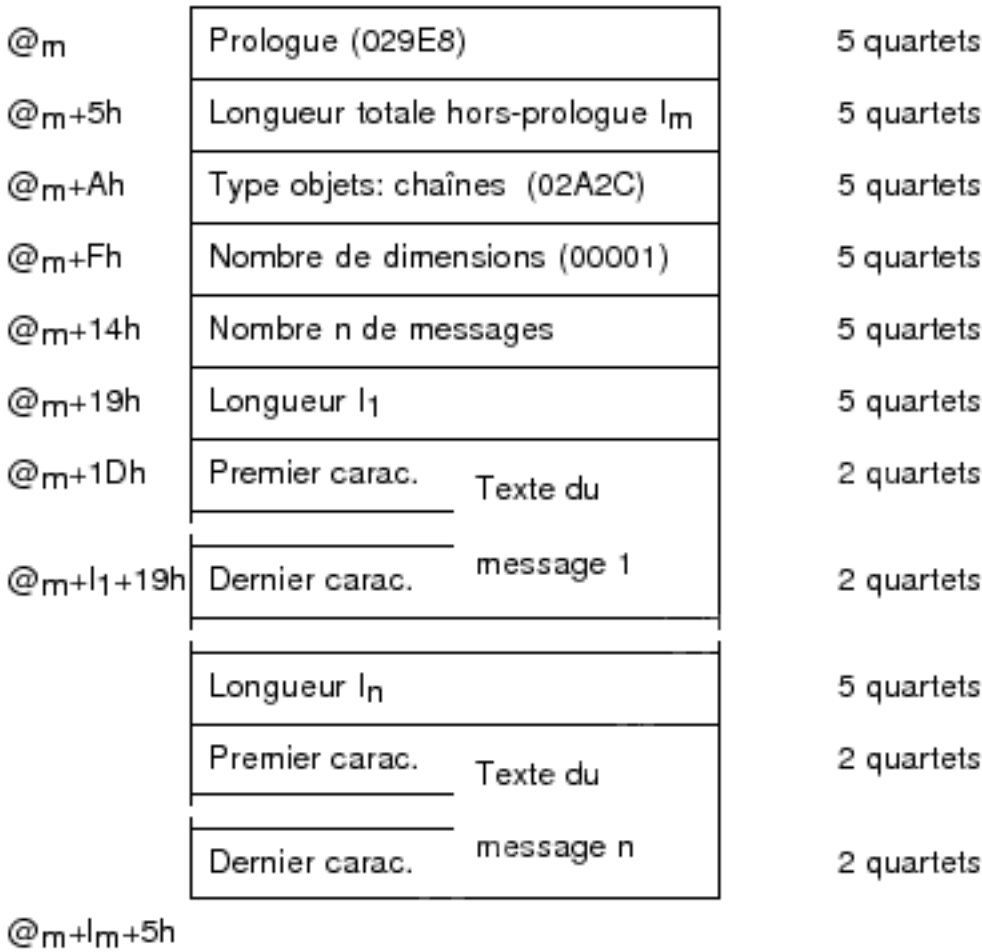
Ici la table est un grand entier. Les 16 premiers champs sont des offsets permettant de retrouver le début de la table des noms d'une classe donnée.

On trouve ensuite la longueur de cette table des noms.

Cette table est constituée d'une entrée par nom. Pour chacun d'entre eux on trouve, dans cet ordre, le nombre de caractères du nom, les codes ASCII de ces caractères, puis le numéro de commande.

La dernière zone de la hash-table comporte dans l'ordre des numéros de commandes, les offsets utilisés pour retrouver les noms de commandes dans la hash-table (pour l'affichage).

La structure de la table des messages est la suivante:

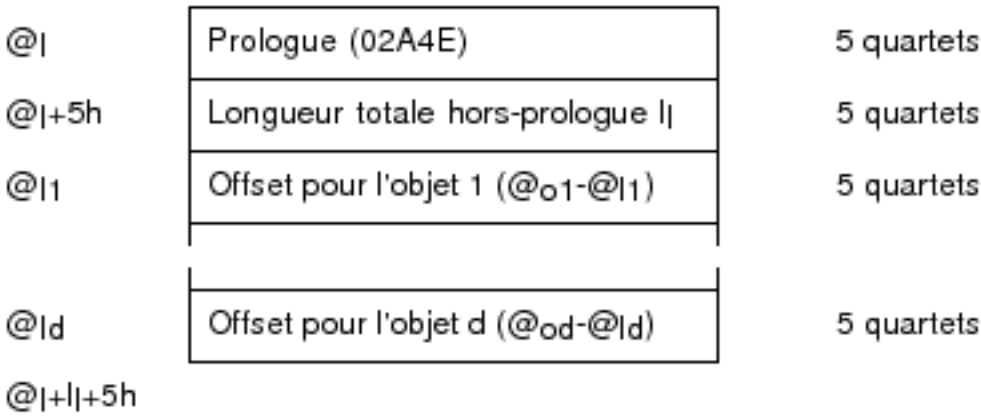


Il s'agit d'un vecteur de chaînes de caractères (voir la structure de l'objet "array").

Ce vecteur contient les messages utilisés par la librairie. Le numéro du message correspond à sa place dans le vecteur.

La librairie interne de la HP48 (002h) utilise une telle table pour stocker les messages d'erreurs...

La structure de la link-table est la suivante:



Cette troisième et dernière table permet de retrouver l'adresse de début d'un objet de la librairie.

Il s'agit encore d'un grand entier qui contenant une série d'offsets sur 5 quartets.

Ces offsets sont dans le même ordre que les objets dans la librairie.

Exemple

Une librairie vide se code:

04B20 32000 40 65944454 40 FF6 00000 00000 00000 00000

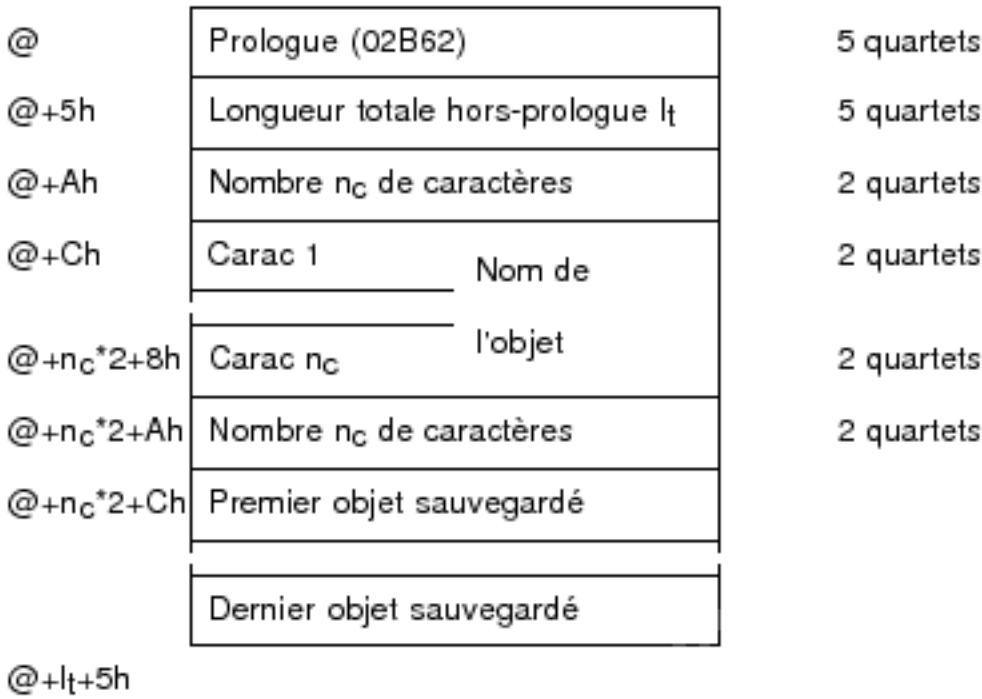
Exercice

B-4-31: Quel est le numéro de librairie de l'exemple ci dessus ?

B-4-32: Quel est son nom ?

B-4-33: A-t-elle une table de messages ?

XVIII) Backup



Il s'agit de l'objet utilisé pour stocker les sauvegardes dans un port quelconque.

Après le prologue et la longueur totale hors-prologue de l'objet, on trouve le nom de l'objet sauvegardé.

Vient ensuite le ou les objets sauvegardés.

Dans l'utilisation standard faite de cet objet par la HP48, il contient toujours deux objets.

Le premier est l'objet sauvegardé, le second étant un System Binary contenant le CRC (code de redondance cyclique) de l'objet en question.

La structure ainsi réalisée est explicitée page suivante.

@	Prologue (02B62)		5 quartets
@+5h	Longueur totale hors-prologue l_t		5 quartets
@+Ah	Nombre n_c de caractères		2 quartets
@+Ch	Carac 1	Nom de	2 quartets
		l'objet	
@+n_c*2+8h	Carac n_c		2 quartets
@+n_c*2+Ah	Nombre n_c de caractères		2 quartets
@+n_c*2+Ch	Objet		
@+l_t-5h	Prologue 02911	System Integer	5 quartets
@+l_t	0	du CRC	1 quartet
@+l_t+1h	Valeur du CRC		4 quartets
@+l_t+5h			

L'objet 'backup' ne contient qu'un seul objet sauvé, suivi d'un 'system binary' qui contient la somme de contrôle de l'objet, calculée par la même formule que dans le cas de la librairie.

Il s'agit du même code de contrôle que pour les échanges fait avec Kermit, à savoir le calcul du reste d'une division par le polynôme:

$$x^{16}+x^{12}+x^5+1$$

Ce calcul n'est pas réalisé de manière logicielle dans le cas de la HP48, mais à l'aide d'un circuit électronique spécialisé (voir le chapitre sur la Ram des entrées sorties). Le programme `CRC` présenté dans la biliothèque de programmes réalise un tel calcul (de manière purement logicielle).

Cette somme de contrôle est calculée, dans le cas du backup, sur la zone allant de @+5h à @+l_t inclus.

Exemple

26B20 92000 40 2434B405 40 C2A2090000F4B4 11920 0 6D26 est le backup de la chaîne "OK".

Exercices

B-4-34: Quel est le nom du backup ?

B-4-35: Quelle est la valeur de sa somme de contrôle ?

XIX) Library Data

@	Prologue (02B88)	5 quartets
@+5h	Longueur totale hors-prologue l_t	5 quartets
@+Ah	Contenu	l_t-5 quartets

@+l_t+5h

Cet objet n'a pas d'existence dans la HP48 de base. Il est laissé libre d'usage aux programmeurs réalisant des librairies et ayant besoin d'un nouvel objet pour stocker leurs données (on peut, par exemple, envisager une librairie correspondant à un mini-tableur, et ayant besoin de stocker des tableaux sous une forme autre que matricielle).

Il n'y a donc pas de structure particulière pour cet objet, si ce n'est qu'il débute par son prologue (comme pour tout objet) suivi de la longueur totale hors-prologue...

XX) Reserved 1, 2, 3 et 4

@	Prologue	5 quartets
@+5h	Longueur totale hors-prologue l_t	5 quartets
@+Ah	Contenu	l_t-5 quartets

@+l_t+5h

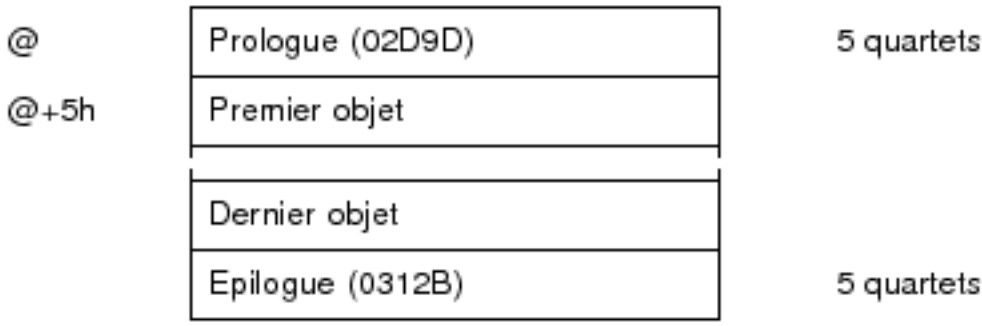
Ces quatre objets ont la même structure (hormis la différence de prologue). Ils ne sont pas utilisés, mais sont probablement réservés pour un usage futur (pour que Hewlett-Packard puisse créer un nouvel objet sans avoir à bouleverser toute la structure de la Rom existante).

Les prologues sont:

- #02BAAh pour Reserved 1;
- #02BCCh pour Reserved 2;
- #02BEEh pour Reserved 3;
- #02C10h pour Reserved 4.

Comme de tels objets n'existent pas, aucun exemple ni exercice ne sera donné à leur sujet.

XXI) Program



Cet objet sert à stocker tous les programmes-utilisateur. Il s'agit d'un objet similaire à la liste: un prologue, une collection d'objets quelconques, un épilogue.

Cependant, l'épilogue et le prologue ne correspondent pas à "«" et "»" qui sont des objets faisant partie de la collection.

Exemple

Le programme « A B + » se code en:

D9D20 E1632 84E201014 84E201024 76BA1 93632 B2130

Exercices

A l'aide de l'exemple précédent, déterminer:

B-4-36: Par quoi sont codés "«" et "»" ?

B-4-37: Comment est codée l'addition ?

XXII) Code

@	Prologue (02DCC)	5 quartets
@+5h	Longueur totale hors-prologue l_t	5 quartets
@+Ah	Codes machine	l_t-5 quartets

@+l_t+5h

Cet objet sert à stocker des programmes en langage-machine. Le champ "code-machine" contient la suite des codes des instructions-machine du programme assembleur qu'il représente.

Exemple

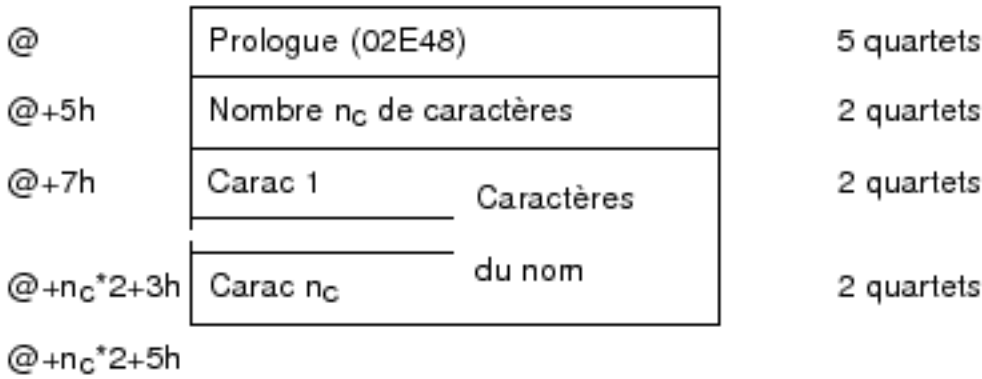
Pour des exemples, voir les programmes en langage-machine présentés en annexe.

Exercices

B-4-38: Comment est stocké un code vide ?

B-4-39: Ecrire un code ne faisant rien (à l'aide des autres chapitres).

XXIII) Global Name



C'est l'objet utilisé pour stocker les noms globaux. Le champ après le prologue indique le nombre de caractères constituant le nom. On trouve ensuite les caractères, dans leur ordre d'apparition, sous forme de codes ASCII.

Exemple

Le nom global 'Voyage' est codé:

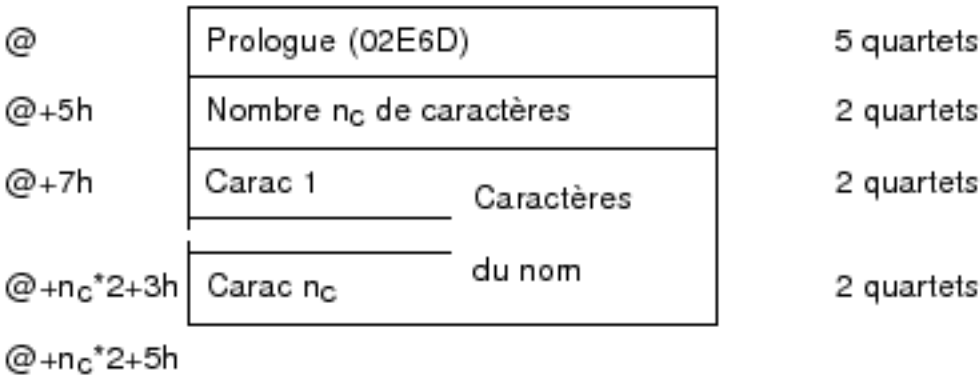
84E20 60 65 F6 97 16 67 65

Exercices

B-4-40: Coder 'Hello'.

B-4-41: Que représente 84E2000 ?

XXIV) Local Name



C'est l'objet utilisé pour stocker les noms locaux. Sa structure est la même que celui du nom global (voir plus haut) hormis la différence de prologue.

Exemple

'Local' se code D6E20 50 C4 F6 36 16 C6.

Exercices

B-4-42: Combien de caractères dans le nom local codé ci-dessous ?

D6E2040E416D656

B-4-43: Quel est ce nom ?

XXV) XLIB name

@	Prologue (02E92)	5 quartets
@+5h	N° de librairie	3 quartets
@+8h	N° de commande	3 quartets

@+Bh

Lorsqu'on utilise des librairies, on fait référence aux commandes qu'elles contiennent.

Pour optimiser l'accès à ces commandes leur nom est remplacé par un "XLIB name" qui contient le numéro de librairie et le numéro d'ordre de la commande en question.

Cette notation peut être utilisée pour les commandes des deux librairies standard de la HP48 (les librairies 002h et 700h).

Exemple

La commande FREE , librairie #002, commande #163 peut être représentée par 29E20 200 361.

Exercices

B-4-44: Coder la commande #123h de la librairie #456h

B-4-45: Quels sont les numéros de librairie et de commande de la XLIB "29E20100200" ?

XXVI) Autres objets

A tous les objets précédents, on peut ajouter :

- Les adresses d'objets structurés qui sont utilisées dans les programmes, listes et autres objets composites. Par exemple, l'instruction SWAP est représentée par l'adresse en mémoire morte #1FBBDh;
- Des codes en langage-machine stockés sous la forme <adresse courante + 5h> <codes LM>, ou, plus généralement <adresse d'un programme LM>. Ces objets ne sont concevables qu'en mémoire morte où leur adresse est fixe... Ils sont représentés à l'affichage par l'appellation générique d'"externals".

Organisation générale de la mémoire

Nous avons vu que le microprocesseur Saturn utilise des registres de 20 bits pour adresser la mémoire et peut donc de ce fait accéder à 2^{20} éléments élémentaires de mémoire, les quartets, soit 512 Ko (kilo-octets).

Cet espace-mémoire est divisé en 5 parties:

- La mémoire morte (Rom) qui contient tous les programmes utilisés par la machine (calcul de racines carrées, tracés de courbes, beep...). Cette mémoire qui, comme son nom l'indique, ne peut être modifiée (elle est "morte"), a une taille de 256 Ko;
- La mémoire vive (Ram) des entrées-sorties qui sert à accéder aux périphériques de la HP48 (émetteur/récepteur infra-rouge, horloge, écran...). Cette zone est très courte puisqu'elle ne comporte que 64 quartets. Elle recouvre une partie de la mémoire morte;
- La mémoire vive interne où sont stockées les informations de l'utilisateur (programmes, variables, alarmes...). Cette zone fait 32 Ko;
- Les deux cartes enfichables (128 Ko maximum chacune).

Si on totalise ainsi la mémoire utilisée en configuration maximale (2 cartes de 128 Ko), on obtient un total de 32 Ko supérieur au maximum adressable par le microprocesseur Saturn...

Pour remédier à ce problème, la HP48 utilise une technique appelée le 'bank-switching'.

Cette méthode est très classique: elle consiste à attribuer la même adresse à deux zones-mémoire distinctes, l'une des deux étant prioritaire sur l'autre. Cette mémoire plus prioritaire est la seule visible et "cache" l'autre.

Si on veut accéder à la mémoire cachée, on effectue une reconfiguration de la mémoire visible, afin de lui attribuer une autre adresse. La zone cachée est alors accessible.

De façon à minimiser les temps d'accès, on ne stocke dans la mémoire cachée que des informations auxquelles on accède peu souvent. Dans le cas le la HP48, il s'agit des routines d'auto-test, des messages d'erreurs, des polices de caractères...

La mémoire du HP48 peut donc se trouver dans deux états:

- L'état standard où la Ram interne va de #70000h à #7FFFFh (voir la figure 1 sur la page suivante);
- L'état d'accès aux informations, où la Ram se trouve déplacée en #F0000h. Cet état de la mémoire est celui auquel on a accès au travers du moniteur (figure 2 page suivante) .

Le mini-éditeur de mémoire dont nous venons de parler permet une visualisation simple du second état, et donc un accès à toutes les informations contenues dans la calculatrice. Pour accéder à ce moniteur, il suffit de lancer l'auto-test manuel (par ON-D) puis d'appuyer sur la touche "BACKSPACE" (la touche de DROP/CLR). Ce moniteur affiche sur une ligne une adresse-mémoire et le contenu des 16 quartets situés à partir de cette adresse. On dispose alors des commandes suivantes:

- [0],[1],...[9],[A],...[F] qui permettent de stocker une valeur à l'adresse courante (à utiliser avec précautions !);
- Des commandes de déplacement dans la mémoire:
  - Par #1000h avec [↓] et [↑]
  - Par #100h avec [*] et [/]
  - Par #1h avec [+] et [-]
- Des commandes de sortie par la prise série:
  - Par #10h avec [.]
  - Par #10000h avec [SPC]
- Des commandes d'accès à des zones-mémoire prédéfinies:
  - #00100h (Ram I/O) par [ENTER]
  - #80000h (Port 1) par [EEX]
  - #C0000h (Port 2) par [DEL]
  - #F000Ah (Données de WSLOG) par [+/-]
  - #F0A8Ch (zone écran) par [1/x]

- Une commande de mise à jour de l'affichage: [←];
- Une commande d'exécution du programme en langage-machine situé à l'adresse courante: [EVAL] (à utiliser avec précautions).

Pour la HP48sx et dans ce mode les cartes, apparaissent respectivement en #80000h et #C0000h alors qu'elles sont reconfigurées de manière à former une zone continue lors de l'utilisation normale de la machine.

#00000h	Début de Rom		256 quartets
#00100h	Ram des entrées-sorties		64 quartets
#00140h	Suite de la Rom		458432 quartets
#70000h	Ram interne		65536 quartets
#80000h	Port 1	Cartes	262144 quartets
#C0000h	Port 2	enfichables	262144 quartets
#100000h			

Figure 1

#00000h	Début de Rom		256 quartets
#00100h	Ram des entrées-sorties		64 quartets
#00140h	Suite de la Rom		523968 quartets
#80000h	Port 1	Cartes	262144 quartets
#C0000h	Port 2 (partiel)	enfichables	196608 quartets
#F0000h	Ram interne déplacée		65536 quartets
#100000h			

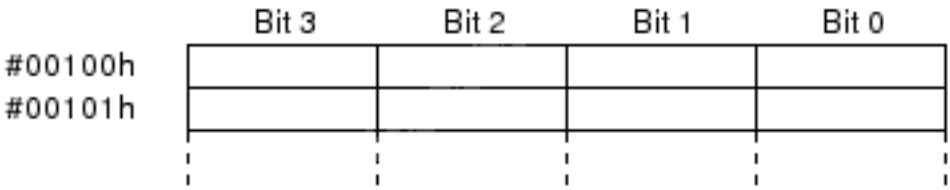
Figure 2

La ram des entrées-sorties

Pour communiquer avec ses périphériques, la HP48 utilise, entre autres méthodes, une zone particulière de la mémoire: la ram des entrées-sorties.

Cette zone de 64 quartets est en effet un moyen d'échanges avec le monde extérieur. En écrivant ou en lisant des quartets, il est possible d'envoyer des ordres ou de consulter des données provenant des périphériques...

Dans les pages suivantes, la ram des entrées-sorties est décrite bit par bit, dans un tableau du type suivant:



où bit 3 est le bit de poids fort du quartet concerné, 0 son bit de poids faible.

L'usage du ou des bits est précisé dans le tableau et expliqué plus longuement dans le texte lui faisant face.

#00100h	Display	Marge gauche		
#00101h	Contraste écran			
#00102h				
#00103h				
#00104h #00105h #00106h #00107h	Calculateur de codes de redondance cyclique			
#00108h				
#00109h				
#0010Ah				
#0010Bh	Alert	Alpha	Right Shift	Left Shift
#0010Ch	Indicateurs		Transmitting	Busy
#0010Dh		Vitesse RS232c		
#0010Eh				
#0010Fh	Informations sur les ports (HP48sx)			
#00110h	Interruptions RS232c			
#00111h				
#00112h			Entrée ok	Sortie ok
#00113h				
#00114h #00115h	Entrée RS232c			
#00116h #00117h	Sortie RS232c			
#00118h				
#00119h				
#0011Ah	IR in			IR in mem.
#0011Bh				
#0011Ch	IR out			
#0011Dh				
#0011Eh				
#0011Fh	Adresse de base de la ram interne			

Marge gauche

La marge gauche est codée sur 3 bits et a donc une valeur de 0 à 7. Elle sert au scrolling d'écran de la partie haute (hors-zone des menus).

Par exemple, mettre la valeur 1 conduit à décaler l'affichage d'une colonne à gauche.

Son usage est lié à la marge droite et à l'adresse de bitmap écran décrites plus loin.

Display

Display est à mettre à 1 pour allumer l'écran, à 0 pour l'éteindre.

Il est intéressant de noter qu'éteindre l'écran rend le clavier inactif et accélère la machine d'environ 13%. Cette accélération s'explique par le fait que les bitmaps-écran se trouvent en mémoire. Si l'écran est éteint, il n'y a plus accès à la mémoire à chaque rafraîchissement de l'affichage.

De ce fait le bus se trouve déchargé et les échanges entre la mémoire et le microprocesseur se font plus rapidement. En particulier, l'exécution de programme est accélérée...

Contraste

Le contraste de l'écran est codé sur 5 bits (bit de poids fort en #00102h).

Le contraste est donc réglable sur 32 niveaux. Seules les valeurs 3h à 13h sont accessibles par `[ON][+]` et `[ON][-]`.

Le programme `CONTRAST` (voir la bibliothèque de programmes) utilise cette adresse pour régler l'afficheur de manière logicielle.

Calculateur de CRC

La HP48 utilise des checksums (voir le chapitre sur les objets). pour vérifier l'intégrité des données en mémoire.

Afin d'en calculer rapidement la valeur, elle utilise un circuit électronique branché sur le bus.

Ce circuit lit les informations circulant entre la mémoire et le microprocesseur et calcule le CRC (code de redondance cyclique) correspondant.

Pour calculer un CRC (comme le réalise la fonction BYTES) il suffit donc de forcer ces 4 quartets à zéro, de lire les quartets de l'objet dont on veut calculer le CRC, puis de relire les 4 quartets en #00104h.

Il faut noter que cette lecture ne doit pas être perturbée. Il faut donc interdire les interruptions lorsqu'on l'effectue (par l'instruction assembleur INTOFF. Attention: il ne faut pas oublier de ré-autoriser ces interruptions en fin de programme grâce à l'instruction machine INTON).

De plus, ces quatre quartets ayant des valeurs très variables, on peut les utiliser comme générateur de nombres aléatoires en langage-machine.

La valeur du CRC étant fonction des quartets lus en mémoire, on pourra en particulier commencer par lire des valeurs pseudo-aléatoires (horloge, adresse de fin de pile, mémoire libre...) avant de lire la valeur pseudo-aléatoire en #00104h...

Indicateurs

Les indicateurs (alpha, busy...) sont allumés grâce à 1 bit chacun (1=allumé, 0=éteint).

Le bit 3 de #0010Ch conditionne l'allumage ou l'extinction de l'ensemble des indicateurs.

Vitesse RS232c

L'émission et la réception de données sur la prise RS232c se font suivant une vitesse exprimée en "bauds" (dans le cas de la norme RS232c, cette unité correspond à un nombre de bits par seconde).

La HP48 est capable d'émettre des données à quatre vitesses différentes: 1200 bauds, 2400 bauds, 4800 bauds et 9600 bauds.

Les 3 bits de poids faibles de #0010Dh permettent de régler la vitesse d'émission / réception de la RS232c.

Le codage des vitesses se fait comme le décrit le tableau ci-dessous:

Bit 2	Bit 1	Bit 0	Vitesse
0	0	0	1200 bauds
0	1	0	2400 bauds
1	0	0	4800 bauds
1	1	0	9600 bauds

Informations sur les ports (HP48sx)

Le quartet #0010Fh donne des indications sur l'état des deux ports de la HP48. Les bits ont la signification suivante:

Bit n°	Rôle
0	à 1: Carte présente en port 1.
1	à 1: Carte présente en port 2.
2	à 1: Ecriture possible en port 1.
3	à 1: Ecriture possible en port 1.

Interruptions RS232c

Lorsqu'un caractère arrive sur l'entrée RS232c, il peut provoquer une interruption. Celle-ci déroute le microprocesseur de la HP48 vers un programme spécial qui traite cette interruption.

Par exemple, dans le cas de la réception d'un caractère par la RS232c, le traitement correspond à une lecture du caractère en attente et en son stockage dans le buffer RS232c (voir le chapitre suivant).

Le quartet en #00110h permet d'autoriser ou d'interdire ces interruptions, ainsi que de déterminer si elles se sont produites.

Chacun des bits de ce quartet a un rôle précis décrit dans le tableau ci-dessous:

Bit n°	Rôle
0	à 1: une interruption liée à une réception a eu lieu.
1	à 1: les interruptions liées aux réceptions sont autorisées.
2	à 1: une interruption liée à une émission a eu lieu.
3	à 1: les interruptions liées aux émissions sont autorisées.

Si on désire adresser directement la prise RS232c, il convient d'interdire ces interruptions...

Entrée ok

Si ce bit est à 1, cela signifie qu'un caractère vient d'être reçu via la prise RS232c de la HP48. Il convient donc d'aller le lire en #00114h.

Sortie ok

Si ce bit est à 1, il est possible d'émettre un caractère via la prise RS232c: on peut aller écrire en #00116h.

Entrée RS232c

Les entrées-sorties RS232c (la prise au sommet de la HP48) s'effectuent via un circuit spécialisé. Pour recevoir un octet sur cette prise, il suffit de lire les deux quartets en #00114h.

Sortie RS232c

Pour émettre un octet sur cette prise, il suffit d'écrire les deux quartets en #00116h.

Entrée infra-rouge

En #0011Ah, se trouve l'entrée infra-rouge. Le bit 3 passe à 1 ou 0 selon qu'il y a réception ou non. Le bit 0 passe à 1 à la première réception et constitue donc une sorte de mémoire de l'entrée (qu'il faut remettre à 0 manuellement).

Sortie infra-rouge

IR out est le bit correspondant à la sortie infra-rouge. Mettre ce bit à 1 déclenche l'émission, à 0 l'arrête.

Adresse de base de la ram interne

#0011Fh contient l'adresse de base de la ram interne (#7h ou #Fh). #7h est la valeur normale (ram interne en #**7**0000h), #Fh étant la valeur correspondant à un déplacement de la ram (ram interne en #**F**0000h).

Cette valeur est mise à jour par le système lors des déplacements de la mémoire interne (pour accéder à la rom cachée). il ne s'agit que d'une petite mémoire vive: changer sa valeur n'a aucune influence sur l'adresse de base de la ram interne.

Elle est utilisée par les routines devant fonctionner en mode normal et en mode ram déplacée (comme le gestionnaire d'écran) de manière à ce que la position de la ram interne n'ait aucune influence sur le fonctionnement de la machine

#00120h #00121h #00122h #00123h #00124h	Adresse du début de la bitmap écran			
#00125h #00126h #00127h	Marge à droite (en quartets)			
#00128h	Hauteur menu			
#00129h				
#0012Ah				
#0012Bh				
#0012Ch				
#0012Dh				
#0012Eh				
#0012Fh				
#00130h #00131h #00132h #00133h #00134h	Adresse de début de la bitmap des menus			
#00135h				
#00136h				
#00137h				
#00138h #00139h #0013Ah #0013Bh #0013Ch #0013Dh #0013Eh #0013Fh	Horloge			

Adresse bitmap écran

L'écran de la HP48 est divisé en deux demis-écrans horizontaux: l'écran lui-même (affichage de la pile) et le menu.

Les pixels de ces écrans sont stockés en mémoire à une adresse quelconque que le driver d'écran doit connaître.

En particulier l'adresse de début du bitmap correspondant à la partie haute de l'écran est stockée en #00120h. Elle correspond à l'adresse du contenu de l'objet graphique (GROB) correspondant.

Il y a deux remarques à faire:

- Cette adresse doit impérativement être paire (ceci étant sans doute dû à l'utilisation d'un circuit électronique de gestion d'écran 8 bits);
- Cette adresse ne peut être qu'écrite: sa relecture est impossible (la valeur lue ne correspond à rien de valide). Elle doit donc être sauvegardée si l'on veut la réutiliser ultérieurement. Un emplacement est prévu à cet effet dans la ram réservée (voir le chapitre suivant).

Marge à droite

De manière à permettre des scrolls rapides de l'écran, on stocke aussi une marge à droite (en quartets, le bit 0 est ignoré) en #00125h et à gauche (en pixels) en #00100h.

Pour effectuer des scrolls rapides, il suffit donc de fixer l'adresse de début de bitmaps, ainsi que les deux marges aux valeurs correspondant à la partie de l'objet graphique que l'on désire visualiser...

Les deux remarques faites pour l'adresse de début de la bitmap écran sont toujours valables (la valeur doit être paire, ce qui explique que les objets graphiques soient constitués de blocs de 8 pixels, et ne peut être relue, ce qui nécessite une sauvegarde de la valeur).

Hauteur menu

La hauteur de la séparation entre les deux écrans est réglable grâce à l'offset en #00128h. Mettre cet offset à 3Fh conduit à faire disparaître la barre des menus.

Là encore, la valeur ne peut être qu'écrite et doit donc être sauvegardée.

Les valeurs standards (pas de librairie attachée) sont:

- #7097Ch pour l'adresse de la bitmap écran (grob de la pile);
- #70858 pour l'adresse de la bitmap menu;
- #0h pour la marge à droite;
- #000h pour la marge à gauche;
- #37h pour la hauteur de séparation.

Horloge

Dernière zone de la ram des entrées-sorties: l'horloge. Sa valeur, en 8192èmes de seconde, est stockée sur 8 quartets et va décroissant de #FFFFFFFFh à #00000000h.

La HP48 n'utilise pas la totalité de cette plage de valeurs:

- Lorsque l'affichage de l'heure est validé, la calculatrice n'utilise que la plage correspondant à un intervalle d'une seconde. Les valeurs de ces 8 quartets vont donc de #00001FFFh à #00000000h (soit 8192 8192èmes de seconde);
- Si l'affichage n'est pas validé et si une alarme doit avoir lieu dans l'heure qui suit, c'est le nombre de 8192èmes de seconde avant cette alarme qui constitue la valeur de l'horloge;
- Sinon la plage de valeurs utilisée est 0 à 1 heure (soit # 01C20000h à #00000000h), avec remise à 1 heure dès l'appui de touche en mode interactif.

Cette utilisation s'explique par le fait que le passage de l'horloge par la valeur # 00000000h provoque une interruption utilisable par la machine...

La mémoire vive

La mémoire de la HP48 est divisée en plusieurs zones ayant chacune un rôle distinct. Avant de nous pencher en détails sur chacune de ces zones, en voici l'organisation globale:

#70000h	Ram réservée	
(#70551h)	GROBS écran	
(#7056Ah)	Objets temporaires	
(#7056Fh)	Return Stack	
B	Mémoire libre	D*5 quartets
D1	Pile	(#7069Fh) qu.
(#7057Eh)	Ligne de commande	48 quartets min.
(#70583h)	Pile d'undo et variables locales	
(#70588h)	Zone de 5 quartets à 0	5 quartets
(#7058Dh)	Temporary environment	78 quartets
(#70592h)	Variables utilisateur (HOME dir)	
(#70597h)	Sauvegardes en port 0	
(#70669h)		

Toutes ces zones, sauf la mémoire réservée, sont situées à des adresses variables. Ces adresses sont stockées dans la mémoire réservée (ainsi que dans certains registres du microprocesseur).

Pour cette visite guidée de la mémoire vive, nous allons donc détailler la mémoire réservée, et à l'occasion de la description des différents contenus, nous étudierons les zones mémoire correspondantes...

#70000h	CMOS word		5 quartets
#70005h	0000		4 quartets
#70009h	Inhibition arrêt-système		1 quartet
#7000Ah	Type		1 quartet
#7000Bh	Date	WSLOG 1	13 quartets
#70018h	CRC		4 quartets
#7001Ch	Type		1 quartet
#7001Dh	Date	WSLOG 2	13 quartets
#7002Ah	CRC		4 quartets
#7002Eh	Type		1 quartet
#7002Fh	Date	WSLOG 3	13 quartets
#7003Ch	CRC		4 quartets
#70040h	Type		1 quartet
#70041h	Date	WSLOG 4	13 quartets
#7004Eh	CRC		4 quartets
#70052h	Valeur	Offset	13 quartets
#7005Fh	CRC	d'horloge	4 quartets
#70063h	0000000000000		13 quartets
#70070h	FF		2 quartets
#70072h	Auto-test start time		13 quartets
#7007Fh	Auto-test fail time		13 quartets
#7008Ch	Préparation affichage		44 quartets

CMOS word

Les 5 premiers quartets de la mémoire vive sont toujours à la valeur #A5C3Fh. Il s'agit d'un mot de vérification de la validité du contenu de la mémoire. Lorsqu'on essaye de changer ces valeurs, on obtient un arrêt-système.

Inhibition de l'arrêt-système

Le bit 3 quartet situé en #70009h commande l'inhibition de l'arrêt-système ON-C, de l'auto-test manuel ON-D et automatique ON-D. Il rend aussi impossible l'extinction de la machine en forçant le réallumage au bout de quelques secondes. Mettre le bit 3 à 0 provoque cette inhibition.

WSLOG

En #70009h, #7001Ch, #7002Eh et #70040h sont stockées les informations concernant la commande WSLOG. Cette commande (non documentée dans les manuels livrés avec la HP48) renvoie la cause, la date et l'heure du dernier démarrage à chaud de la machine. Les causes possibles sont représentées par un numéro allant de #0h à #Fh suivant le code présenté à la page suivante.

C'est le premier quartet de la zone qui contient ce code.

On trouve ensuite la date à laquelle s'est produit le problème, codée sur 13 quartets, en 8192ème de seconde depuis le premier janvier de l'an 1 !

Enfin, sur 4 quartets, on trouve un checksum permettant de vérifier l'intégrité de la donnée. Ce checksum est calculé sur les 14 quartets précédents.

Ce checksum est plus exactement appelé code de redondance cyclique. Il est calculé en considérant que les bits des quartets de données sont les coefficients d'un polynôme que l'on divise par le polynôme de degré 16 $x^{16}+x^{12}+x^5+1$. Le reste de cette division (un polynôme de degré 15 au plus dont les coefficients sont 0 ou 1) est stocké sur 4 quartets (16 bits). Ce calcul est effectué par un circuit électronique dont l'accès se fait par la ram des entrées-sorties. Le programme `CRC` de la bibliothèque de programmes, réalise un tel calcul de manière purement logicielle...

Code	Signification
0	Allumage de la machine alors que celle-ci était en mode COMA (mise en mode coma par ON-SPC).
1	Batteries très faibles.
2	Problème hardware durant une transmission infra rouge.
3	La machine a effectué un redémarrage (exécution du programme en #00000h).
4	L'offset d'horloge (contrôlé par CRC) a été corrompu.
5	Changement dans les données d'une carte de manière non contrôlée.
6	Non utilisé.
7	Un mot de vérification (sur 5 quartets) en ram ne correspond plus à l'état de la mémoire (la ram est peut-être corrompue).
8	Une anomalie a été détectée lors de la configuration d'un des 5 périphériques (l'un d'entre eux n'est pas configuré ou ne correspond pas à un périphérique valide).
9	Liste d'alarmes corrompue (le CRC n'est plus valide).
A	Non utilisé.
B	Module retiré.
C	Reset système (par le bouton de reset situé sous un des pieds de la machine par exemple).
D	Un gestionnaire d'erreurs RPL est introuvable.
E	Table de configuration corrompue.
F	Carte ram retirée.

Tableau de codage de WSLOG

Offset d'horloge

On trouve en #70052h l'offset d'horloge codé sur 13 bits et suivi de son CRC. Là encore l'offset correspond à une durée (en 8192ème de secondes) écoulée depuis la naissance du Christ.

Autotest start & fail time

Les deux zones de 13 quartets en #70072h et #7007Fh sont utilisées lors de l'auto-test pour stocker respectivement les temps de début du test et d'échec (si un échec a lieu).

Comme ces deux valeurs ne sont que peu importantes, elles ne sont pas validées par un CRC.

Préparation affichage

Enfin, en #7008Ch se trouvent 44 quartets qui servent à préparer les affichages lors de l'utilisation du mini-moniteur (22 caractères).

#700B8h	???...???		35 quartets
#700DBh	Cartes enfichées (bits 0 et 1)		1 quartet
#700DCh			288 quartets
#701FCh	Données	Buffer	512 quartets
#703FCh	BufLen	d'entrée	2 quartets
#703FEh	BufFull	de la prise	1 quartet
#703FFh	BufStart	RS232c	2 quartets
#70401h			43 quartets
#7042Ch	Drapeaux	Informations	1 quartet
#7042Dh	Début	sur la carte	5 quartets
#70432h	Indic. Taille	en port 1	5 quartets
#70437h	Drapeaux	Informations	1 quartet
#70438h	Début	sur la carte	5 quartets
#7043Dh	Indic. Taille	en port 2	5 quartets
#70442h			26 quartets
#7045Ch	Sauvegardes pour les interruptions		103 quartets
#704C3h	Masque de sortie pour test clavier		3 quartets
#704C6h			36 quartets
#704EAh	KeyStart	Buffer du	1 quartet
#704EBh	KeyEnd	Clavier	1 quartet
#704ECh	Codes Touches		32 quartets
#7050Ch			2 quartets

Cartes enfichées (HP48sx)

Les bits 0 et 1 de #700DBh indiquent la présence ou l'absence de carte en port 1 et 2.

Si bit est à un, cela signifie que la carte correspondante est présente, à zéro qu'elle est absente.

Le bit 2 (respectivement 3) est à 1 si la carte en port 1 (respectivement en port 2) est en écriture autorisée.

En résumé, on a le tableau suivant:

Bit 3	Bit 2	Bit 1	Bit 0
2: Ecriture ok	1: Ecriture ok	2: Carte prés.	1: Carte prés.

Par exemple, si le quartet #700DBh vaut #Bh (en binaire: #1011b) cela signifie:

- Carte présente en port 1 (bit 0 à 1);
- Carte présente en port 2 (bit 1 à 1);
- Carte protégée en écriture en port 1 (bit 2 à 0);
- Carte non protégée en écriture en port 2 (bit 3 à 1).

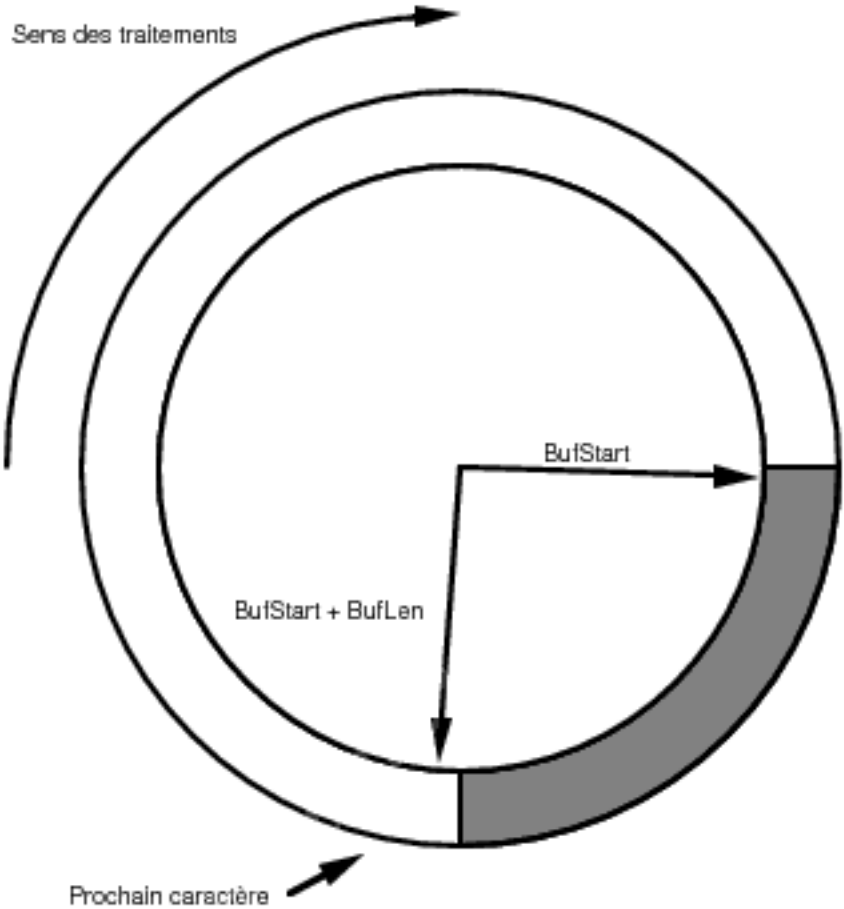
Buffer RS232c

Le buffer d'entrée de la prise RS232c est constitué d'une zone de données de 512 quartets, soient 256 caractères, d'un pointeur de début, BufStart, qui indique le numéro du premier caractère contenu (son adresse est donc #704EAh + 2 * BufStart) et d'un compteur de caractères présents (le dernier caractère reçu se trouve donc en #704EAh + 2 * BufStart + 2 * BufLen -2, le prochain reçu sera stocké en #704EAh + 2 * BufStart + 2 * BufLen).

Un dernier quartet (#704EBh, BufFull) est utilisé pour signaler un dépassement de capacité (buffer plein).

Ce quartet vaut 0 si le buffer est plein, 8 si des informations ont été perdues.

Ce buffer peut donc se représenter par le schéma ci-dessous:



La partie grisée représente la zone contenant des données (contrairement au buffer clavier, les cases mémoires n'ont pas été représentées de manière à ce que le dessin reste lisible).

Informations sur les cartes (HP48sx)

En #7042Ch (respectivement en #70437h) se trouvent des informations sur la carte présente en port 1 (respectivement en port 2).

Cette zone de 11 quartets débute par 1 quartet contenant des informations sur l'utilisation de la carte:

- Le bit 1 indique si la carte est utilisée en extension mémoire (bit à 1 si tel est le cas);

- Le bit 2 indique si la carte est en écriture autorisée (bit à 1 si tel est le cas);
- Le bit 3 indique si la carte est présente (bit à 1 si la carte est bien là).

En résumé, on a le tableau suivant:

Bit 3	Bit 2	Bit 1	Bit 0
Carte présente	Ecriture ok	Carte "mergée"	

Les 5 quartets suivants (#7042Dh ou #70438h) contiennent l'adresse de début de la carte.

En #70432h et #7043Dh se trouve stockée la taille de la carte en complément à 0 (carte de 32 Ko: #F0000h, carte de 128 Ko: #C0000h).

Ces deux valeurs (adresse de début et taille) ne sont valides que si la carte n'est pas fusionnée avec la ram interne.

Sauvegardes pour les interruptions

La zone de 103 quartets située en #7045Ch est utilisée par le système lors des interruptions pour la sauvegarde temporaire du contenu des registres.

Les interruptions (détournement de l'exécution des programmes vers un programmes spécial) sont utilisées par la HP48 pour le traitement du clavier, de la prise RS232c, de l'horloge...

Masque de sortie pour le test clavier

Le masque de sortie situé en #704C3h est utilisé comme argument de OUT=C lors des tests claviers qui ont lieu lors des interruptions. Il est remis à 0 par le système.

Forcer périodiquement ces 3 quartets à #FFFh provoque un grésillement du haut-parleur car des interruptions ont lieu toutes les secondes.

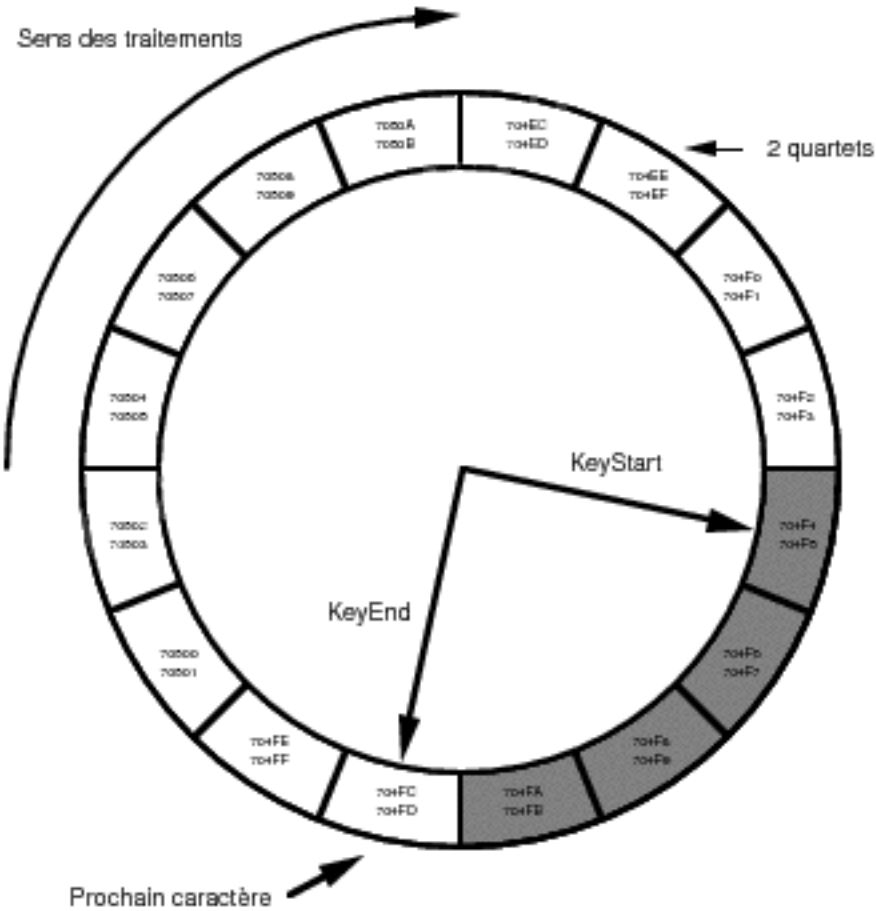
Buffer clavier

Le buffer clavier comporte une zone de données de 32 quartets (16 codes de touches).

Deux pointeurs sont utilisés pour repérer le contenu du buffer:

- KeyStart indiquant le numéro de la case contenant la première touche;
- KeyEnd pointant sur la première case libre.

Les codes des touches à traiter sont donc contenus de #704ECh+2*KeyStart à #704EC+2*KeyEnd, de manière cyclique comme le montre le schéma ci dessous:.



Dans le schéma ci-dessus, KeyStart vaut 4 et KeyEnd 8.

Le codage des touches se fait selon le tableau ci-contre.

A 01	B 02	C 03	D 04	E 05	F 06
MTH 07	PRG 08	CST 09	VAR 0A	↑ 0B	NXT 0C
' 0D	STO 0E	EVAL 0F	← 10	↓ 11	→ 12
SIN 13	COS 14	TAN 15	√x 16	y^x 17	1/x 18
ENTER 19		+/- 1A	EEX 1B	DEL 1C	← 1D
α 80	7 1F	8 20	9 21	÷ 22	
↰ 40	4 24	5 25	6 26	× 27	
↱ C0	1 29	2 2A	3 2B	- 2C	
ON 2D	0 2E	. 2F	SPC 30	+ 31	

Codes buffer des touches du clavier

#7050Eh	Sauveg. @ bit. ecr. (#00120h)	5 quartets
#70513h	Sauveg. marge droite (#00125h)	3 quartets
#70516h	Sauveg. @ bit. menu (#00130h)	5 quartets
#7051Bh	Sauveg. hauteur menu (#00128h)	2 quartets
#7051Dh		52 quartets
#70551h	@ GROB du menu	5 quartets
#70556h	@ GROB de la pile	5 quartets
#7055Bh	@ GROB courant	5 quartets
#70560h	@ GROB graphique	5 quartets
#70565h	@ GROB graphique ?	5 quartets
#7056Ah	@ début des objets temporaires	5 quartets
#7056Fh	@ fin des objets temporaires	5 quartets
#70574h	@ début de memoire libre (B)	5 quartets
#70579h	@ fin de mémoire libre (D1)	5 quartets
#7057Eh	@ ligne de commande	5 quartets
#70583h	@ pile d'undo et variables locales	5 quartets
#70588h	@ zone de 5 quartets à 0 ?	5 quartets
#7058Dh	@ temporary environment	5 quartets
#70592h	@ home directory	5 quartets
#70597h	@ zone de sauvegarde (port 0)	5 quartets
#7059Ch	@ directory courant	5 quartets
#705A1h	@ home directory ?	5 quartets

Sauvegardes

Dans le chapitre sur la ram des entrées-sorties, nous avons vu que plusieurs zones étaient utilisées pour décrire l'afficheur de la HP48 (marge gauche, droite, hauteur du menu...), mais que certains de ces indicateurs ne pouvaient être qu'écrits.

De ce fait, il est nécessaire de les stocker dans une zone-mémoire accessible, en l'occurrence en ram réserve.

C'est pourquoi nous trouvons stocké ici:

- en #7050Eh, la sauvegarde de l'adresse de la bitmap écran (écrite en #00120h);
- en #70513h, la sauvegarde de la marge à droite (écrite en #00125h);
- en #70516h, la sauvegarde de l'adresse de la bitmap du menu (écrite en #00130h);
- en #7051Bh, la sauvegarde de la hauteur de séparation entre l'écran et le menu (écrite en #00128h).

Ces différents paramètres sont donc toujours stockés aux deux endroits (ram réservée et ram des entrées sorties) par les programmes de gestion d'écran de la HP48.

Adresses des objets graphiques

Les 5 adresses suivantes pointent sur les différents objets graphiques utilisés par la machine. Ceux-ci sont stockés en début de mémoire vive (juste après la fin de la mémoire réservée).

Objets temporaires

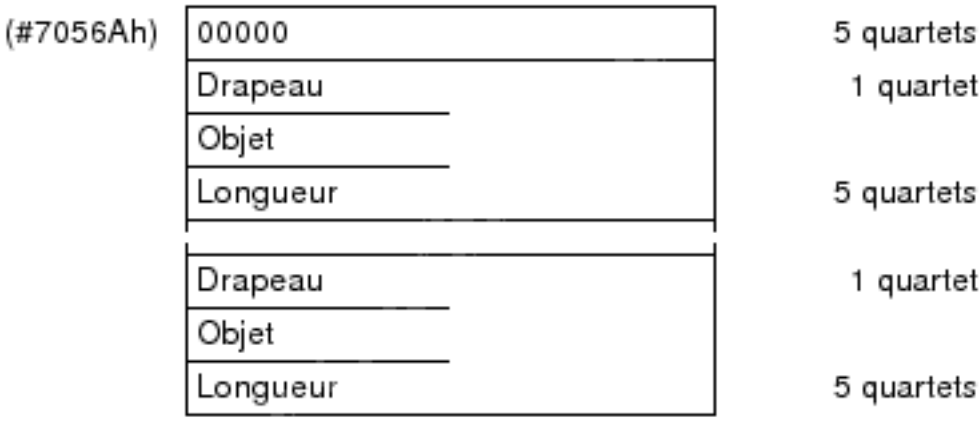
On trouve ensuite les adresses de début et de fin de la zone des objets temporaires. Cette zone est utilisée par la HP48 pour stocker les objets à faible durée de vie (les objets non stockés dans une variable utilisateur). Chacun des objets présents dans cette zone est stocké de la manière suivante:

Drapeau (garbage collector)	1 quartets
Objet	l_Z - 6 quartets
Longueur l_Z de la zone	5 quartets

Lorsqu'on utilise la machine, ces objets s'accumulent dans la zone des objets temporaires. Il faut donc procéder de temps en temps à un nettoyage de manière à détruire les objets non utilisés. Cette opération (qui se produit notamment lors de l'utilisation de la commande `MEM`) est réalisée par un programme nommé le "garbage collector".

Lors de cette opération, la machine repère tous les objets utilisés (à partir des différentes adresses stockées en mémoire). Elle marque chacun de ces objets (dans la zone Drapeau de la structure ci-dessus). Après avoir parcouru chacun de ses pointeurs, la HP48 détruit les objets non marqués (qui ne sont donc pas utilisés).

Cette zone a donc la structure:



Pile des retours

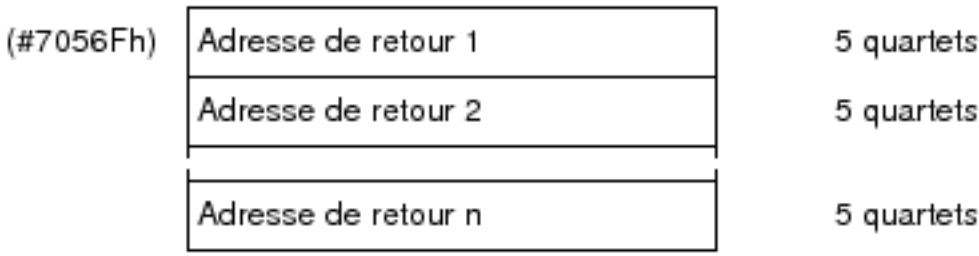
L'adresse de fin des objets temporaires pointe aussi sur le début de la pile des retours du langage de la HP48.

C'est dans cette zone que sont sauvegardées les adresses de retour lors de l'exécution récursive d'objets (programme dans un programme).

Il y a empilage lorsqu'on rencontre le prologue de programme (#02D9Dh) et dépilage lorsque l'on rencontre l'épilogue indiquant la fin de programme (031B2).

La fin de cette zone est pointée par le registre B.

Elle présente donc l'aspect suivant:



l'adresse 1 étant la plus ancienne...

Le registre B pointe sur la fin de cette zone et sur le début de la mémoire libre. Comme la routine de sauvegarde des registres utiles (SAVE_REG) sauve B en #70574h, on trouve souvent la valeur de ce registre à cette adresse.

Mémoire libre

Elle est comprise entre l'adresse contenue dans B (fin de pile des retours) et l'adresse contenue dans D1 (premier étage de la pile, voir plus loin).

La taille de cette mémoire libre est stockée dans le registre D (champ A) en nombre de "cases de 5 quartets" (par exemple, si D champ A vaut #00100h, cela signifie que la zone de mémoire libre fait entre #00500h et #00504h quartets).

Cette indication en terme de "cases de 5 quartets" s'explique par le fait que la pile des retours et la pile-utilisateur (voir plus loin) utilisent des groupes de 5 quartets.

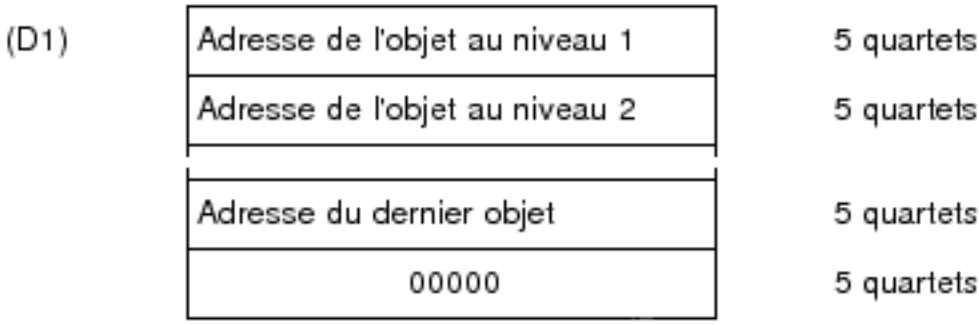
Il est donc facile de savoir s'il reste assez de mémoire libre pour étendre une de ces deux piles (ce qui est une opération très fréquente), puisqu'il suffit de tester si D (champ A) est non nul...

La pile utilisateur

De même que la valeur de B est sauvegardée en #70574h, la valeur de D1, le pointeur de pile, est sauvegardée #70579h.

La pile de la HP48 peut contenir n'importe quel objet. De manière interne la pile ne contient en fait que les adresses de ces objets (car les adresses ont une taille unique: 5 quartets).

Le registre D1 pointe sur le premier étage de la pile, celle-ci se terminant à l'adresse stockée en #7057E:



Pour connaître l'adresse d'un objet situé au niveau n, il suffit de prendre la valeur standard de D1 (sauvée en #70579h par SAVE_REG), d'y ajouter (n-1)*5 (le premier objet est au niveau 1) et de lire 5 quartets à cette adresse.

En particulier, pour réaliser un SWAP (échange des objets aux niveaux 1 et 2), il suffit d'utiliser le programme assembleur suivant:

```
A=DAT1   A       * Adresse de l'objet 1
D1=D1+   S       * Passage au niveau 2
C=DAT1   A       * Adresse de l'objet 2
DAT1=A   A       * On écrit l'adresse de 1
D1=D1-   S       * Passage au niveau 1
DAT1=C   A       * On écrit l'adresse de 2
```

Attention: ce programme n'effectue aucune vérification sur le nombre des objets présents dans la pile...

Ligne de commandes

La ligne de commande commençant à l'adresse contenue en #7057Eh et se terminant à celle stockée en #70583h est la zone utilisée lors de l'édition de ligne de commande.

Cette zone contient les codes ASCII des caractères en cours d'édition et se termine par la caractère 00 (Null) qui sert de délimiteur de fin de ligne (ce qui explique l'impossibilité d'éditer des chaînes contenant ce caractères).

Elle comporte toujours au moins 23 caractères plus le caractère null, les caractères non existants étant remplis par des "nulls":

(#7057Eh)	Caractère 1	2 quartets
	Caractère 2	2 quartets
	...	
	Caractère n (n ≥ 23)	2 quartets
	00	2 quartets
(#70583h)		

La pile d'undo

La mémorisation du contenu de la pile (pile d'undo) et des variables locales se fait dans la même zone mémoire.

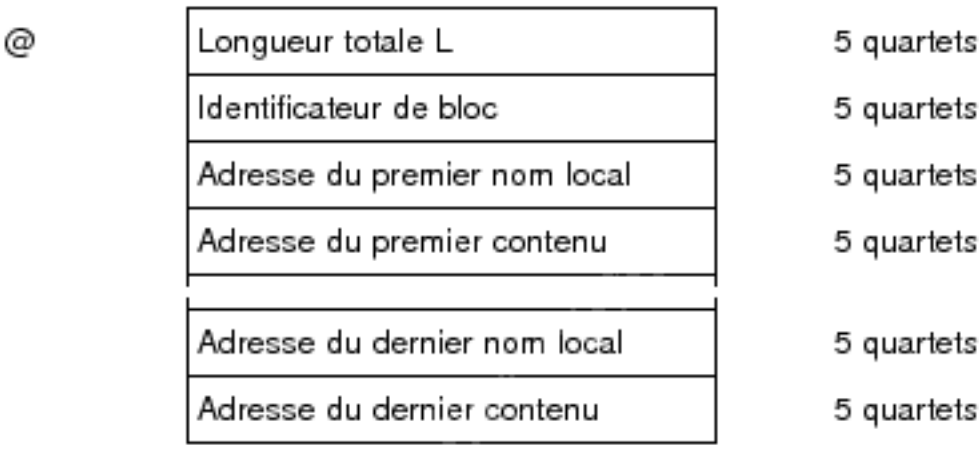
Cette zone possède une structure par blocs:

(#70583h)	Bloc 1	
	Bloc 2	
	...	
	Dernier bloc (undo)	
	00000	5 quartets
(#70588h)		

Le dernier bloc concerne la mémorisation de la pile (UNDO), les autres contenant les différentes variables locales et leur contenu.

Le premier bloc contient les variables les plus récemment crées, l'avant-dernier les plus anciennes...

Chacun de ces blocs est sub-divisé en plusieurs champs, comme le montre le schéma suivant:



Dans le cas d'un bloc de variables locales, l'identificateur de bloc vaut #00000h. Les adresses de noms locaux pointent sur des objets 'local name'. Les adresses de contenu pointent sur l'objet stocké dans la variable locale dont le nom précède.

Pour la pile d'undo, la structure est sensiblement la même (voir schéma, page suivante).

L'identificateur de bloc vaut #00001h s'il n'y a pas de variables locales, #00002h sinon.

Pour des raisons d'homogénéité avec les blocs de variables locales, on trouve encore des pointeurs sur des noms locaux. Ces pointeurs valent tous #61D3Ah, adresse, en mémoire morte, du nom local vide (").

Le champ "nombre n d'éléments dans la pile" contient la profondeur de la pile (codé sur 5 quartets).

@		
	Longueur totale L	5 quartets
	Identificateur de bloc	5 quartets
	Adresse de " (#61D3Ah)	5 quartets
	Nombre n d'éléments dans la pile	5 quartets
	Adresse de " (#61D3Ah)	5 quartets
	Adresse de l'objet au niveau 1	5 quartets
	Adresse de " (#61D3Ah)	5 quartets
	Adresse de l'objet au niveau n	5 quartets

@+L

La pile d'undo

Les champs restants contiennent les adresses des différents objets présents dans la pile d'undo...

Temporary environment

L'environnement temporaire (temporary environment) concerne la gestion des menus.

Cette zone-mémoire contient les adresses nécessaires à l'affichage des touches de menu et à l'exécution des routines associées. Elle est structurée comme le montre le tableau de la page suivante.

Les adresses d'affichage permettent à la HP48 de déterminer le texte à afficher au-dessus des touches ainsi que le texte à placer sur la ligne de commandes en mode PRG ou ALG.

Les adresses d'exécution permettent de retrouver l'adresse du programme associé à une de ces touches.

Lorsque une touche de menu n'a pas de fonction associée, son nom est une chaîne vide (adresse #055DFh) et l'adresse de la

routine d'exécution associée (#3FDD1h) est celle d'un programme émettant un petit "beep".

On peut constater que la place nécessaire à une septième touche a été réservée (soit pour des évolutions futures, soit parce qu'au moment de la programmation des programmes de gestion de cette zone, l'aspect extérieur de la HP48 n'était pas encore complètement décidé).

(#7058Dh)	#07Ch	3 quartets
(#7058Dh)+3h	Adresse d'affichage 1	5 quartets
(#7058Dh)+8h	Adresse d'affichage 2	5 quartets
(#7058Dh)+Dh	Adresse d'affichage 3	5 quartets
(#7058Dh)+12h	Adresse d'affichage 4	5 quartets
(#7058Dh)+17h	Adresse d'affichage 5	5 quartets
(#7058Dh)+1Ch	Adresse d'affichage 6	5 quartets
(#7058Dh)+21h	Adresse d'affichage 7 (réservé)	5 quartets
(#7058Dh)+26h	Adresse d'exécution 1	5 quartets
(#7058Dh)+2Bh	Adresse d'exécution 2	5 quartets
(#7058Dh)+30h	Adresse d'exécution 3	5 quartets
(#7058Dh)+35h	Adresse d'exécution 4	5 quartets
(#7058Dh)+3Ah	Adresse d'exécution 5	5 quartets
(#7058Dh)+3Fh	Adresse d'exécution 6	5 quartets
(#7058Dh)+44h	Adresse d'exécution 7 (réservé)	5 quartets
(#7058Dh)+49h		

Home directory

L'adresse du "home directory", stockée en #70592h, est celle de l'objet "directory" constituant le répertoire principal (celui dans lequel on se trouve après un arrêt-système ou l'exécution de la commande HOME).

Cet objet a été décrit en détails dans le chapitre sur les objets.

En #705A1h, on retrouve cette adresse.

Directory courant

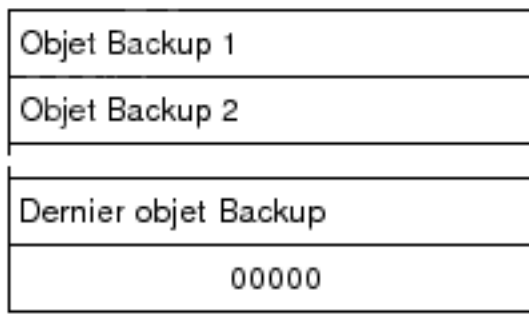
En #7059Ch, est stocké le pointeur sur le répertoire courant qui est lui aussi un objet de type "directory".

Zone de sauvegardes

La HP48 permet d'effectuer des sauvegardes. Celles-ci peuvent être faites sur carte (pour la HP48sx) mais aussi en mémoire interne (dans le port 0).

Cette zone de sauvegarde est organisée de la même manière, quel que soit le port utilisé. Dans le cas de la ram interne (ou de la ram interne et des cartes "mergées" dans le cas de la HP48sx) on trouve l'adresse de début de cette zone en #70597h.

Elle est constituée d'une suite d'objets "backup" (voir le chapitre sur les objets):



#705A6h	@ userkeys		5 quartets
#705ABh	@ alarmes		5 quartets
#705B0h	@ objet à exécuter (D0)		5 quartets
#705B5h	Zone de sauvegarde		5 quartets
#705BAh	@ objet 1		5 quartets
#705BFh	@ objet 2		5 quartets
#705C4h	@ objet 3	Pile de LAST	5 quartets
#705C9h	@ objet 4		5 quartets
#705CEh	@ objet 5		5 quartets
#705D3h	@ grand entier		5 quartets
#705D8h	00000		5 quartets
#705DDh	@Cmde 1	Pile des quatre	5 quartets
#705E2h	@Cmde 2	dernières	5 quartets
#705E7h	@Cmde 3	lignes de	5 quartets
#705ECh	@Cmde 4	commandes	5 quartets
#705F1h	?????		5 quartets
#705F6h	?????		5 quartets
#705FBh	?????		5 quartets
#70600h	@ Error message		5 quartets
#70605h			100 quartets
#70669h	@ de fin de ram		5 quartets
#7066Eh	Mémoire libre (en 5 quartets) (D)		5 quartets

Userkeys et alarmes

En #705A6h et #705ABh, on trouve respectivement les adresses des définitions de touches utilisateur et de la liste des alarmes.

Ces deux tables sont des variables comme celles crées par l'utilisateur et contenues dans un répertoire caché.

Il est en effet possible de "cacher" des objets contenus dans le répertoire utilisateur.

Le principe est simple: si lors du balayage du directory courant que la machine effectue pour déterminer les noms des objets de ce répertoire, elle rencontre un objet dont le nom est vide (''), elle arrête sa recherche.

Pour cacher un objet, on peut donc soit lui donner le nom '' (c'est ce qui est fait pour le répertoire contenant les définitions de touches et la liste d'alarmes), soit le placer après un objet de nom vide. Dans ce second cas, l'objet reste exécutable mais n'apparaît pas comme touche de menu...

Le répertoire caché de la HP48 contient les objets suivants:

- `'Alarms'` qui contient la liste des alarmes;
- `'UserKeys'` qui contient la liste des définitions de touches USR;
- `'UserKeys.CRC'` qui contient la somme de contrôle de `UserKeys` (calculée par: `UserKeys BYTES DROP`).

Pour accéder à ce répertoire caché, il suffit de se placer dans le home-directory et de faire:

```
#15781h SYSEVAL
```

On se retrouve alors dans le directory caché, car ce syseval correspond à l'évaluation du nom vide (`''`).

L'accès aux différents objets cachés est alors possible, mais il est fortement déconseillé de les détruire ou même de les modifier, sous peine de "Memory Lost" !

Pour revenir à la normale: exécuter `HOME`.

Objet à exécuter

#705B0h sert de zone de stockage du registre D0 et pointe donc sur le prochain objet devant être exécuté.

Pile de LAST

La pile de LAST (ARG) est une série de cinq adresses qui correspondent aux objets sauvés (c'est pourquoi le nombre maximal d'objets sauvés par LAST ARG est 5 bien que dans la plupart des cas seuls trois paramètres soient sauvés).

Si moins de 5 objets ont été sauvés, les autres adresses sont mises à la valeur #00000h.

Adresse grand entier

En #705D3h, on trouve l'adresse d'un grand entier (184 chiffres).

Il s'agit probablement d'une table utilisée de manière interne par la HP48.

Cet entier est situé dans la ram des objets temporaires.

Comme il s'agit du premier objet temporaire crée par la HP48, il est toujours le premier objet que l'on rencontre dans cette partie de la mémoire vive.

Pile des lignes de commandes

La pile de commande est constituée selon le même principe que la pile de LAST.

Elle comporte quatre adresses pointant sur des chaînes de caractères contenant ces lignes de commandes.

La plus récente est celle dont l'adresse est stockée en #705DDh, la plus ancienne est #705ECh.

Adresse message d'erreur

En #70600h est stockée l'adresse de la chaîne de caractères contenant le dernier message d'erreur si celui-ci était un message défini par l'utilisateur (par "message" DOERR). Sinon cette adresse vaut #00000h.

Fin de mémoire vive

L'adresse de fin de mémoire vive peut être lue en #70669h.

Pour la HP48sx, la mémoire vive peut être étendue par adjonction d'une ou plusieurs cartes-mémoire (à chaque ajout de carte, la mémoire est reconfigurée de manière à ce que la mémoire-utilisateur forme une zone continue).

Mémoire libre

Les 5 quartets en #7066Eh sont utilisés pour la sauvegarde du registre D. Celui-ci contient une approximation de la mémoire libre, mesurée en nombre de groupes de 5 quartets (voir plus haut).

#70673h	Prochaine erreur à afficher		5 quartets
#70678h			39 quartets
#7069Fh	StackSize		5 quartets
#706A4h	Racine de génération aléatoire		16 quartets
#706B4h			17 quartets
#706C5h	Système	Drapeaux	16 quartets
#706D5h	Utilisateur		16 quartets
#706E5h			26 quartets
#706FFh	Error number		5 quartets
#70704h			15 quartets
#70713h	Prologue		5 quartets
#70718h	Longueur	GROB du	5 quartets
#7071D	Hauteur (6)	caractère sous	5 quartets
#70722h	Largeur (10)	le curseur	5 quartets
#70727h	Pixels		20 quartets
#7073Bh			158 quartets
#707D9h	Nombre de librairies attachées		3 quartets
#707DCh	Numéro	Info première	3 quartets
#707DFh	@ infos	librairie	5 quartets

Numéro	Info dernière	3 quartets
@ infos	librairie	5 quartets

Prochaine erreur à afficher

La zone située en #70673h est utilisée pour stocker le numéro de la prochaine erreur à afficher (lors du retour au mode interactif, cette zone mémoire est regardée et si un message est en attente, il est affiché).

StackSize

Stacksize est la taille de la pile en quartets. Celle-ci comportant toujours au moins 5 quartets à zéro, stacksize vaut 5*(DEPTH+1).

Racine de génération aléatoire

En #706A4h se trouve stockée la racine utilisée par la fonction RAND. Cette racine est un objet "real" privé de son prologue.

Drapeaux

Les drapeaux sont stockés de #706C5h à #706E4h, comme montré dans le tableau page suivante.

Error number

En #706FFh est stocké le numéro de la dernière erreur s'étant produite.

Ce nombre est à #00000h si aucune erreur n'est mémorisée et à #70000h si il s'agit d'un message d'erreur défini par l'utilisateur.

La liste de tous les messages possibles, avec leur numéro, est donnée en annexe.

GROB du caractère sous le curseur

En #70713h, on trouve un objet graphique utilisé pour mémoriser le caractère sous le curseur en mode-édition.

Drapeaux système (-1 à -64):

	Bit 3	Bit 2	Bit 1	Bit 0
#706C5h	-4	-3	-2	-1
#706C6h	-8	-7	-6	-5
#706C7h	-12	-11	-10	-9
#706C8h	-16	-15	-14	-13
#706C9h	-20	-19	-18	-17
#706CAh	-24	-23	-22	-21
#706CBh	-28	-27	-26	-25
#706CCh	-32	-31	-30	-29
#706CDh	-36	-35	-34	-33
#706CEh	-40	-39	-38	-37
#706CFh	-44	-43	-42	-41
#706D0h	-48	-47	-46	-45
#706D1h	-52	-51	-50	-49
#706D2h	-56	-55	-54	-53
#706D3h	-60	-59	-58	-57
#706D4h	-64	-63	-62	-61

Drapeaux utilisateur (1 à 64):

	Bit 3	Bit 2	Bit 1	Bit 0
#706D5h	4	3	2	1
#706D6h	8	7	6	5
#706D7h	12	11	10	9
#706D8h	16	15	14	13
#706D9h	20	19	18	17
#706DAh	24	23	22	21
#706DBh	28	27	26	25
#706DCh	32	31	30	29
#706DDh	36	35	34	33
#706DEh	40	39	38	37
#706DFh	44	43	42	41
#706E0h	48	47	46	45
#706E1h	52	51	50	49
#706E2h	56	55	54	53
#706E3h	60	59	58	57
#706E4h	64	63	62	61

Nombre de librairies attachées

Les 3 quartets en #707D9h contiennent le nombre de librairies attachées.

Chacune de ces librairies est décrite par son numéro, suivie par l'adresse où sont stockées les informations la concernant.

Si ces informations se trouvent en rom cachée, l'adresse correspond à un "system binary" (en mémoire directement accessible) contenant l'adresse en rom cachée.

Dans tous les cas, on obtient une adresse pointant sur la partie déclarative de la librairie (immédiatement située après le nom, en @+n_C*2+Eh pour reprendre les notations de la page 143).

Ce début de librairie contient en effet toutes les informations nécessaires à l'exploitation du contenu de la librairie (messages, commandes...).

Ceci permet en particulier de retrouver facilement les messages d'erreurs, sachant que le numéro d'un tel message se compose de deux parties: le numéro de la librairie contenant la table de messages dans laquelle il est stocké (sur 3 quartets), et son numéro d'ordre dans cette table sur deux quartets, de 1 à 255 (une librairie comporte donc au maximum 256 messages). Le numéro du message est donc:

No de librairie * 256 + numéro d'ordre

A partir du numéro d'erreur, on peut donc facilement déterminer le numéro de la librairie correspondante, et la liste des librairies attachées permet alors de trouver l'adresse de début de la table des messages contenant le libellé correspondant au numéro spécifié.

En particulier, il est possible, en modifiant cette table d'informations, de réécrire la totalité des messages d'erreurs de la HP48 (ce qui permet, par exemple, de les franciser).

Conclusion

La zone de ram réservée s'arrête normalement en #70844h, mais elle peut être étendue si nécessaire. En particulier, des cartes comme la carte HP-solve, réservent de la mémoire supplémentaire pour cette zone (puisqu'entre autres, elle apporte de nouvelles librairies).

Cette description de la mémoire n'est pas complète mais présente la plupart des éléments utiles pour le programmeur désirant réaliser des programmes en langage-machine accédant aux ressources de la HP48.

Après ces chapitres descriptifs de la HP48, nous allons maintenant voir comment réaliser des programmes écrits en assembleur...

Programmer en langage-machine

Dans les chapitres précédents, nous avons étudié les aspects internes de la HP48.

Nous allons à présent utiliser cette connaissance pour accéder à toutes les ressources de la machine... et en particulier à la programmation en langage-machine.

La HP48 ne pouvant gérer que des objets, nous utiliserons l'objet 'code' (dont la structure a été vue dans le chapitre sur les objets) pour contenir un programme en LM.

Le problème est d'arriver à créer cet objet.

D'une manière plus générale, nous allons voir comment créer n'importe quel type d'objet...

Nous avons vu que tout objet peut être représenté par une suite de chiffres hexadécimaux. Nous allons donc réaliser une fonction transformant une séquence hexadécimale en l'objet correspondant.

L'utilisateur entrera une chaîne de caractères contenant ces chiffres qui devront être transformés en la suite quartets correspondante (car dans une chaîne ces chiffres sont stockés par leur code ASCII, par exemple "A", chiffre hexadécimal valant 10 en décimal est stocké par son code ASCII, 41h).

Or il existe un objet simple qui est constitué de chiffres hexadécimaux lorsqu'on l'édite et de quartets lorsqu'il est stocké en mémoire: l'objet "graphic" (le GROB).

La transformation chiffres -> quartets se fera donc grâce à cet objet.

Sa structure est la suivante:

@	Prologue (02B1E)		5 quartets
@+5h	Longueur totale hors prologue l_t		5 quartets
@+Ah	Nombre n_l de lignes (en pixels)		5 quartets
@+Fh	Nombre n_c de colonnes (en pixels)		5 quartets
@+14h	Col. 1 à 8	Pixels de la	1 + 1 quartets
	Derniers pixels	Ligne 1	1 + 1 quartets
	Col. 1 à 8	Pixels de la	1 + 1 quartets
	Derniers pixels	Ligne n_l	1 + 1 quartets
@+l_t+5h			

On voit que la HP48 considère les colonnes 8 par 8. Nous allons donc créer un objet graphique de 8 colonnes sur un nombre de lignes égal au nombre de chiffres à coder divisé par 2 (car 8 pixels correspondent à 2 quartets donc à 2 chiffres hexadécimaux).

Si ce nombre est impair, on arrondira supérieurement.

De cette manière, la taille-mémoire occupée par le GROB sera (sans compter le prologue et les informations de longueur et de taille) au plus égale au nombre de chiffres à coder plus un (en quartets).

Ce codage se fera par une séquence du type:

```
"GROB 8 " OVER SIZE 2 / CEIL + " " + SWAP + OBJ→
```

Ce morceau de programme prépare l'objet graphique dans une chaîne:

- On met le début du grob dans une chaîne (`"GROB 8 "`);
- On calcule le nombre de lignes du grob à créer par la séquence `OVER SIZE 2 / CEIL` et on l'ajoute au début du GROB;
- On ajoute ensuite la liste des codes hexadécimaux à coder (séparés du nombre de lignes par l'ajout de `" "`) par la séquence `" " + SWAP +`;
- Enfin on transforme la chaîne de caractères contenant l'objet graphique par la commande `OBJ→`.

On peut simplifier légèrement ce programme en supprimant l'arrondi supérieur (`CEIL`) car il est automatiquement effectué lors de la transformation de la chaîne en objet graphique par `OBJ→`.

On obtient donc le programme:

```
"GROB 8 " OVER SIZE 2 / + " " + SWAP + OBJ→
```

Cette séquence met dans la pile l'objet graphique contenant l'objet que l'on veut obtenir.

En mémoire, on trouve alors la structure suivante:

@	Prologue (02B1E)	5 quartets
@+5h	Longueur totale hors prologue l_t	5 quartets
@+Ah	Nombre n_l de lignes (en pixels)	5 quartets
@+Fh	Nombre n_c de colonnes (en pixels)	5 quartets
@+14h	Objet à créer	l_t-15 quartets
@+l_t+5h		

On sait que ce sont les adresses des objets qui sont stockées dans la pile. Pour accéder à l'objet à créer, il convient donc de prendre l'adresse @ qui est dans la pile et de la remplacer par @+14h (pour passer le prologue, la longueur, le nombre de colonnes et le nombre de lignes).

Or il existe un appel par SYSEVAL qui réalise cette fonction. Cet appel en #056B6h prend en entrée l'objet et un "system integer" contenant le nombre de cases de 5 quartets à passer et renvoie le nouvel objet ainsi qu'un "external" qui ne nous est pas utile ici.

Nous devons passer 4 cases de 5 quartets, il nous faut donc un "system integer" égal à 4. Un tel objet est stocké en #04017h.

La transformation du GROB en objet se fera donc par la séquence:

```
#4017h SYSEVAL #56B6h SYSEVAL DROP
```

Le premier SYSEVAL rappelle le "system integer" dans la pile, le second effectue la transformation.

La dernière action à faire est de recréer l'objet de manière à ce que la référence dans la pile soit vraiment une référence à un objet en tant que tel (et non plus à un contenu).

Cette recréation se fait très simplement par la fonction NEWOB qui recrée l'objet au niveau 1 de la pile, en modifiant toutes les références le concernant)...

On obtient donc finalement le programme GASS (GASS comme Graphic ASSembleur):

```
GASS
«
 "GROB 8 " OVER SIZE 2 / + " " + SWAP + OBJ→
  #4017h SYSEVAL #56B6h SYSEVAL DROP NEWOB
»
```

Ce programme est très rapide car la transformation des chiffres hexadécimaux en quartets est réalisée par des programmes en mémoire morte écrits en langage machine...

Néanmoins il effectue des vérifications et des calculs qui le ralentissent un peu. Un programme extrêmement rapide écrit entièrement en langage machine est présenté en annexe (ASS).

Essayons à présent le programme GASS pour créer un petit objet...

Pour faciliter la lecture des codes, ceux-ci sont présentés 5 par 5. Ces espaces ne font pas partie de la suite de codes: il faut les entrer sous forme contigüe (sans espace, sans passage à la ligne).

Voici le listing des codes d'un petit objet:

```
C2A20  B1000  7556C  6C602  46F6E  65602  12
```

Pour coder cet objet, procéder de la manière suivante:

- Taper les codes dans une chaîne de caractères (sans espace, sans passage à la ligne):

  ```
  "C2A20B10007556C6C60246F6E6560212"
  ```

- Exécuter GASS...
- Quelque secondes plus tard, l'objet est dans la pile...

Maintenant que nous savons créer n'importe quel objet, nous allons voir comment réaliser des programmes en langage-machine.

Pour écrire de tels programmes, il faut garder à l'esprit quelques points précis:

- Les contenus de certains registres:
  - D0 est le pointeur sur le prochain objet à exécuter (après le programme en langage machine). Pour passer à l'objet suivant on effectue la séquence A=DAT0 A, D0=D0+5, PC=(A) qui se code 142164808C;
  - D1 est le pointeur de pile. Si on effectue A=DAT1 A, A champ A contiendra l'adresse de l'objet au niveau 1. Si on incrémente D1 de 5 (D1=D1+5) on passe au niveau 2 (si on effectue A=DAT1 A, A champ A contiendra alors l'adresse de l'objet au niveau 2);

- B contient l'adresse de fin de pile des retours. Ce registre est peu utile au programmeur;
- D contient la place-mémoire libre en paquets de 5 quartets (c'est à dire en étages de pile);

Ces quatre registres sont à restaurer avant de terminer le programme par 142164808C, sauf si l'on désire volontairement en modifier la valeur. Pour ce faire, il existe deux sous-programmes utiles:

- SAVE_REG, d'adresse #0679Bh (à appeler par GOSBVL #0679B de code 8FB9760) qui sauve ces registres dans la ram réservée;
- LOAD_REG, d'adresse #067D2h (à appeler par GOSBVL #067D2 de code 8F2D760) qui restaure la valeur de ces registres.

- Les structures des objets lorsqu'on cherche à accéder à ceux-ci (pour prendre un argument dans la pile, par exemple) ou lorsque l'on désire encapsuler des objets-codes dans un objet-programme (par exemple pour profiter des fonctions du langage de la HP48);
- La structure de la mémoire vive si l'on doit y accéder.

On peut de plus faire appel aux routines existant en mémoire morte (comme SAVE_REG ou LOAD_REG).

Le meilleur exercice pour appliquer les points exposés dans cette partie est de chercher à comprendre le fonctionnement des programmes en langage-machine présentés en annexe ou, plus difficile, de désassembler certaines routines présentes dans la mémoire morte.

L'étape suivante consiste à concevoir et réaliser des programmes par soi-même, en commençant bien sûr par des idées faciles.

Pour tester la rapidité du langage-machine, vous pouvez par exemple comparer la vitesse d'exécution de deux programmes effectuant des opérations semblables, l'un en langage-machine, l'autre dans le langage de la HP48 (le Reverse Polish Lisp).

Ce test peut être par exemple fait pour deux programmes effectuant 1000 boucles (`1 1000 START NEXT`).

Troisième partie

Bibliothèque de programmes

Avertissement

Cette bibliothèque de programmes comprend de nombreux utilitaires écrits en langage-machine.

Leur compréhension nécessite une lecture approfondie de la partie de ce livre consacrée à ce langage.

Toutefois ils peuvent parfaitement être utilisés sans aucune connaissance particulière, si ce n'est celle de la méthode à utiliser pour les entrer en mémoire, que nous rappelons ici...

Pour des raisons de facilité de lecture les programmes en langage-machine (constitués de chiffres hexadécimaux "0", "1"..."9", "A"..."F") sont présentés par groupes de 5 codes. Par exemple, le programme NOTHING (programme ne faisant rien) serait présenté sous la forme:

```
NOTHING (# B6F7h)
CCD20 F0000 14216 4808C
```

Pour le rentrer en mémoire, il suffit de procéder ainsi:

- Entrer la suite de codes **sans espace ni passage à la ligne** (dans l'exemple ci-dessus, cela donnerait: `"CCD20F00001421648080C"`);
- Après vérification de la somme de contrôle entre parenthèses (optionnelle mais fortement conseillée) exécuter le programme `GASS` (ou `RASS` une fois celui-ci entré) présenté plus loin sur la chaîne de caractères entrée . `GASS` (ou `RASS`) renverra dans la pile l'objet recherché (c'est à dire, dans le cas de programmes en langage machine, le "code" proprement dit [ie: la séquence d'instructions directements compréhensibles par la machine]). Remarques:
  - pour calculer la somme de contrôle il suffit de placer le contenu de l'objet dans la pile et d'exécuter la fonction `BYTES`. Celle-ci renverra la somme de contrôle et la taille de l'objet;
  - Il convient de se placer en mode hexadécimale (par `HEX`) pour effectuer la

comparaison car toutes les sommes de contrôles de ce livres sont données dans cette base;

- Dans le cas des programmes en langage-machine, la somme de contrôle est calculée sur la chaîne de codes avant assemblage par GASS (ou RASS);
- Le programme ALLBYTES présenté plus loin permet de calculer rapidement toutes les sommes de contrôle d'un directory...
- La présence de librairies contenant des commandes de même nom que des programmes utilisés ou la modification de leur nom peuvent conduire à l'obtention d'un checksum différent sans que le programme entré soit faux.

A ce stade, la pile peut contenir un objet inhabituel (contenant souvent le mot Code). **Cet objet ne doit jamais être édité** (sous peine de le détruire).

La dernière chose à faire est alors de stocker le résultat renvoyé par GASS dans la variable correspondant au nom du programme (dans notre exemple, il faut donc faire 'NOTHING' STO).

Pour permettre des vérifications plus faciles, nous avons inclus deux programmes, PAR5 et CLEAN qui permettent respectivement de mettre la chaîne des codes sous une forme identique à celle de ce livre (par groupes de 5 codes, 8 groupes par ligne) et de "nettoyer" une chaîne en enlevant tous les caractères autres que les chiffres hexadécimaux. Pour des raisons de rapidité, ce second programme est lui-même écrit partiellement en langage-machine.

En résumé, avant de taper tout programme en langage-machine, il vous faudra entrer les deux programmes RPL (le langage classique de la HP48) GASS et PAR5, vous entraîner à entrer un programme assembleur avec NOTHING (qui est très court et donc peu susceptible d'erreur) puis entrer le programme-assembleur CLEAN. A ce stade, vous disposerez de tous les outils nécessaires pour accéder à toutes les ressources qui vous sont dévoilées dans ce livre...

Dernière remarque: certains programmes comportent le caractère "↵". Ce symbole représente le retour chariot qui s'obtient en appuyant successivement sur les touches "shift-bleu" ([↱]) et point ([.]).

GASS

GASS est un programme de création d'objets: Il crée n'importe quel objet à partir de son listing en codes hexadécimaux.

Ce programme est expliqué en détail dans le chapitre 8 de la partie précédente.

Il prend une chaîne de caractères contenant la suite de codes hexadécimaux et renvoie l'objet correspondant.

```
GASS (# 1DB3h)
«
  "GROB 8 " OVER SIZE 2 / + " " + SWAP + OBJ→
   #4017h SYSEVAL #56B6h SYSEVAL DROP NEWOB
»
```

Remarque: la création d'objets est une fonction qu'il faut employer avec précaution. En effet, il ne faut pas transformer n'importe quelle suite de codes mais uniquement des listings correspondant à de vrais objets.

Il est donc conseillé de vérifier soigneusement les chaînes de caractères contenant les codes avant de lancer le programme GASS.

ALLBYTES

Le programme ALLBYTES réalise le calcul des sommes de contrôle de tous les objets du répertoire courant.

Il renvoie comme résultat une chaîne de caractères contenant les noms des objets suivis de leur somme de contrôle (hexadécimale).

```
ALLBYTES (# 52FFh)
«
  VARS
  → V
  «
    HEX "␣" 1 V SIZE
    FOR X
      V X GET SWAP OVER →STR 2 OVER SIZE 1 - SUB ":"
      + "              " OVER SIZE 15 SUB + + SWAP
      BYTES DROP " " + +
    NEXT
  »
»
```

PAR5

PAR5 est un petit utilitaire qui permet de présenter les chaînes de codes d'un programme en langage-machine sous une forme identique à celle utilisée dans ce livre (8 groupes de 5 codes par ligne).

Il est utile lorsque l'on recherche une erreur détectée dans une chaîne de codes à l'aide de la somme de contrôle.

Par exemple:

```
"CCD20F0000142164808C" PAR5
```

Renverra:

```
"CCD20 F0000 14216 4808C "
```

Voici donc le programme PAR5:

```
PAR5 (# 74BAh)
«
  → S
  «
    "␣" 0 S SIZE 1 -
    FOR X
      1 40
      FOR Y
        S X Y + DUP 4 + SUB + " " + 5
      STEP
      "␣" + 40
    STEP
  »
»
```

CLEAN

CLEAN réalise la fonction inverse du programme PAR5: il ôte tous les caractères autres que les chiffres hexadécimaux ("0", "1"…"9", "A"…"F").

Il permet de reconstruire une chaîne de codes prête à être utilisée par GASS après utilisation de PAR5 pour vérification...

Ce programme est partiellement écrit en langage-machine, il doit donc être entré conformément aux indications présentes dans l'avertissement situé en début de cette partie...

Voici tout d'abord un listing désassemblé et commenté de CLEAN; le récapitulatif des codes à entrer (sous la forme explicitée dans l'avertissement) se trouve à la suite...

```
          D9D20     CON(5)    PROL_PRGM      Objet programme
          B4E02     CON(5)    STRING_SPC     " "
          76BA1     CON(5)    ADD            +
          CCD20     CON(5)    PROL_CODE      Objet code
début     08000     CON(5)    (fin)-(début)  Longueur code
          8FB976O   GOSBVL    SAVE_REG       Sauvegardes
          143       A=DAT1    A
          130       D0=A                     D0=D1=adresse
          131       D1=A                     objet au niveau 1
          169       D0=D0+    10             D0=adresse contenu
          174       D1=D1+    5
          143       A=DAT1    A              A=taille
          174       D1=D1+    5              D1=adresse contenu
          818F84    A=A-5     A
          819F0     ASRB      A
          D8        B=A       A              B=nbr de carac de
                                             la chaîne
11        8A9       ?B=0      A              Terminé ?
          83        GOYES     14             Oui -> fin !
          14B       A=DAT1    B
          3103      LCHEX     30             Code ASCII de "0"
          9E2       ?A<C      B
          32        GOYES     13             Mauvais carac..
          3193      LCHEX     39             Code ASCII de "9"
          9EA       ?A<=C     B
          41        GOYES     12             Bon carac..
          3114      LCHEX     41             Code ASCII de "A"
          9E2       ?A<C      B
          11        GOYES     13             Mauvais carac..
```

```
          3164      LCHEX     46              Code ASCII de 'F'
          9E6       ?A>C      B
          80        GOYES     13              Mauvais carac..
12        148       DAT0=A    B               Bon carac -> on le
                                               réécrit !
          161       D0=D0+    2               Au suivant..
13        171       D1=D1+    2
          CD        B=B-1     A               Un de moins
          68CF      GOTO      11              On itère !
14        AE0       A=0       B               On marque la fin
          148       DAT0=A    B               par le carac 00
          8F2D760   GOSBVL    LOAD_REG        Récupérations
          142       A=DAT0    A               Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin
          9C2A2     CON(5)    REAL_1
          92CF1     CON(5)    OVER
          C2A20     CON(5)    PROL_STRING     ;
          70000     CON(5)    #0000           ;CHR 00
          00        CON(2)    #00             ;
          4BAC1     CON(5)    POS
          9C2A2     CON(5)    REAL_1
          90DA1     CON(5)    MOINS
          C58C1     CON(5)    SUB
          B2130     CON(5)    EPILOGUE
```

CLEAN (# CD56h)

```
D9D20 B4E02 76BA1 CCD20 08000 8FB97 60143 13013
11691 74143 17481 8F848 19F0D 88A98 314B3 1039E
23231 939EA 41311 49E21 13164 9E680 14816 1171C
D68CF AE014 88F2D 76014 21648 08C9C 2A292 CF1C2
A2070 00000 4BAC1 9C2A2 90DA1 C58C1 B2130
```

PEEK

PEEK permet de connaître le contenu de la mémoire à une adresse donnée. Son utilisation est simple: on lui passe en argument l'adresse à observer et le nombre de quartets à lire et il renvoie un listing des codes hexadécimaux présents à cette adresse sous la forme d'une chaîne de caractères.

Ainsi #0 #5 PEEK renvoie les 5 premiers quartets de la rom de la HP48: "2369B"...

Cette version de PEEK ne permet pas d'accéder au contenu de la rom cachée (mémoire morte en #70000h). Pour accéder à cette zone-mémoire, il convient d'utiliser le programme HRPEEK (Hidden Rom PEEK) présenté plus loin.

Voici tout d'abord le listing commenté du programme assembleur PEEK. Le rappel des codes à entrer se trouve ensuite .

```
         D9D20     CON(5)    PROL_PRGM      Objet programme.
         2ABF1     CON(5)    DUP2           Verification du
         3FBF1     CON(5)    DROP2          nombre d'arguments
         CCD20     CON(5)    PROL_CODE      Objet code
début    3A000     CON(5)    (fin)-(début)  Longueur objet
         8FB9'/60  GOSBVL    SAVE_REG       Sauvegardes
         14'/      C=DAT1    A
         134       D0=C
         169       D0=D0+    10             D0=adresse du
                                            contenu de l'objet
                                            au niveau 1 de la
                                            pile (longueur du
                                            PEEK)
         142       A=DAT0    A              Lecture du nombre
                                            de quartets à lire
         340FFF'/  LCHEX     #'/FFF0        Taille max
         8B6       ?C<A      A
         40        GOYES     10             Taille correcte
         D6        C=A       A              La taille est trop
                                            grande, on la fixe
                                            à la taille max.
10       C6        C=C+C     A              Nombre de quartets
                                            à reserver (2 par
                                            caractère)
         8FD'/B50  GOSBVL    #05B'/D        Reservation ..
```

```
          132       ADDex
          147       C=DAT1    A
          134       D0=C                    D0=adresse de l'
                                            objet au niveau 1
          169       D0=D0+    10
          146       C=DAT0    A             Lecture du contenu
                                            (taille du peek)
          D5        B=C       A
          174       D1=D1+    5
          147       C=DAT1    A
          134       D0=C                    D0=adresse de l'
                                            objet au niveau 2
          169       D0=D0+    10
          146       C=DAT0    A             Lecture du contenu
          135       D1=C
          130       D0=A
l1        8A9       ?B=0      A             Est ce terminé ?
          F2        GOYES     l3            Oui -> fin !
          AE0       A=0       B
          15B0      A=DAT1    1             On lit un quartet
          3103      LCHEX     #30           ;
          A6A       A=A+C     B             ;Transformation
          3193      LCHEX     39            ;en code ASCII
          9EA       ?C>=A     B             ;(0->'1'=48..
          90        GOYES     l2            ;15->'F'=70)
          3170      LCHEX     07            ;
          A6A       A=A+C     B             ;
l2        148       DAT0=A    B             Ecriture dans la
                                            chaîne..
          161       D0=D0+    2             Carac suivant
          170       D1=D1+    1             Quartet suivant
          CD        B=B-1     A             Un de moins
          61DF      GOTO      l1            On itère..
l3        8F2D760   GOSBVL    LOAD_REG      Récupérations
          174       D1=D1+    5             DROP
          E7        D=D+1     A
          118       C=R0
          145       DAT1=C    A             Result -> pile
          142       A=DAT0    A             Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin       B2130     CON(5)    EPILOGUE      Fin programme
```

PEEK (# ED02h)

```
D9D20 2ABF1 3FBF1 CCD20 3A000 8FB97 60147 13416
91423 40FFF 78B64 0D6C6 8FD7B 50132 14713 41691
46D51 74147 13416 91461 35130 8A9F2 AE015 B0310
3A6A3 1939E A9031 70A6A 14816 1170C D61DF 8F2D7
60174 E7118 14514 21648 08CB2 130
```

POKE

Le programme POKE réalise la fonction inverse de PEEK: il permet d'écrire une suite de quartets à une adresse donnée.

Il prend comme arguments un entier égal à l'adresse où 'poker' (au niveau 2 de la pile) et la suite des quartets à écrire en mémoire sous la forme d'une chaîne de caractères contenant les valeurs hexadécimales des quartets à écrire (au niveau 1 de la pile).

ATTENTION: l'usage inconsidéré de POKE peut corrompre le contenu de la mémoire vive et perturber le fonctionnement de la calculatrice... il convient donc de l'employer avec précaution ! Bien entendu les programmes présentés ci-après qui font usage de POKE sont sans danger !

Voici le listing désassemblé et commenté du programme POKE:

```
          D9D20     CON(5)    PROL_PRGM      Objet programme.
          2ABF1     CON(5)    DUP2           Verification du
          3FBF1     CON(5)    DROP2          nombre d'arguments
          CCD20     CON(5)    PROL_CODE      Objet code
début     48000     CON(5)    (fin)-(début)  Longueur objet
          8FB97760  GOSBVL     SAVE_REG       Sauvegardes
          143       A=DAT1    A
          132       AD0ex                    D0=adresse de l'
                                             objet au niveau 1
          164       D0=D0+    5
          146       C=DAT0    A              C=Longueur (5+2*le
                                             nombre de carac de
                                             la chaîne à poker)
          164       D0=D0+    5
          D5        B=C       A
          174       D1=D1+    5
          143       A=DAT1    A
          131       D1=A                     D1=adresse objet 2
                                             (adresse où poker)
          179       D1=D1+    10
          143       A=DAT1    A
          131       D1=A                     D1=adresse où
                                             poker
          34500000  LCHEX     #00005
11        E1        B=B-C     A
          8A9       ?B=0      A              terminé ?
          13        GOYES     11             oui -> fin !
```

```
          14A       A=DAT0    B             Lecture carac
          3103      LCHEX     30            ;
          B6A       A=A-C     B             ;Conversion ASCII
          3190      LCHEX     09            ;en Hexa
          9EA       ?C>=A     B             ;(48="0" -> 0..
          90        GOYES     12            ;"0"="F" -> 15)
          31"0      LCHEX     0"            ;
          B6A       A=A-C     B             ;
12        1590      DAT1=A    1             Ecriture en
                                            mémoire
          161       D0=D0+    2             Carac suivant
          1"0       D1=D1+    1             Quartet suivant
          3420000   LCHEX     #00002
          6DCF      GOTO      11            On itère..
13        8F2D"60   GOSBVL    LOAD_REG      Récupérations
          1"9       D1=D1+    10            ;
          E"        D=D+1     A             ;DROP2
          E"        D=D+1     A             ;
          142       A=DAT0    A             Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin       B2130     CON(5)    EPILOGUE      Fin programme
```

POKE (# 14A5h)

```
D9D20 2ABF1 3FBF1 CCD20 48000 8FB97 60143 13216
41461 64D51 74143 13117 91431 31345 0000E 18A91
314A3 103B6 A3190 9EA90 3170B 6A159 01611 70342
00006 DCF8F 2D760 179E7 E7142 16480 8CB21 30
```

HRPEEK

HRPEEK permet de lire le contenu de la rom cachée qui n'est normalement pas accessible. Pour cela, il détermine où il se trouve en mémoire (ram interne ou module) puis déplace la mémoire vive située en #70000h de manière à laisser libre accès à la rom cachée (zone #70000h à #7FFFFh). La détermination de sa zone de stockage lui permet de déterminer s'il sera ou non impliqué dans ce déplacement...

Son fonctionnement général est le même que PEEK, et sa syntaxe d'appel est la même.

Par exemple #70000h #10h HRPEEK (peek de 16 quartets en #70000h dans la rom cachée) renverra la chaîne de caractères "D21098FFFB108E78".

ATTENTION: il est déconseillé d'utiliser HRPEEK sur une autre zone-mémoire que #70000h #7FFFFh sous peine d'obtenir des résultats erronés (du fait du déplacement de la mémoire vive interne).

Comme HRPEEK effectue des déplacements de la ram interne, l'écran se trouve légèrement "brouillé" pendant l'exécution de ce programme. Ce comportement est normal et ne doit donc pas inquiéter l'utilisateur...

Voici le listing désassemblé et commenté de HRPEEK:

```
         D9D20     CON(5)    PROL_PRGM      Objet programme.
         2ABF1     CON(5)    DUP2           Verification du
         3FBF1     CON(5)    DROP2          nombre d'arguments
         CCD20     CON(5)    PROL_CODE      Objet code
début    D4100     CON(5)    (fin)-(début)  Longueur objet
         8FB97/60  GOSBVL    SAVE_REG       Sauvegardes
         147       C=DAT1    A
         134       D0=C                     D0=adresse de l'
                                            objet au niveau 1
```

```
        169     D0=D0+  10              D0=adresse du
                                        contenu de l'objet
                                        au niveau 1 de la
                                        pile (longueur du
                                        PEEK)
        142     A=DAT0  A               Lecture du nombre
                                        de quartets à lire
        340FFF7 LCHEX   #7FFF0          Taille max
        8B6     ?C<A    A
        40      GOYES   11              Taille correcte
        D6      C=A     A               La taille est trop
                                        grande, on la fixe
                                        à la taille max.
11      C6      C=C+C   A               Nombre de quartets
                                        à reserver (2 par
                                        caractère)
        8FD7B50 GOSBVL  #05B7D          Reservation ..
        132     AD0ex
        147     C=DAT1  A
        134     D0=C                    D0=adresse de l'
                                        objet au niveau 1
        169     D0=D0+  10
        146     C=DAT0  A               Lecture du contenu
                                        (taille du peek)
        10C     R4=C
        174     D1=D1+  5
        147     C=DAT1  A
        134     D0=C                    D0=adresse de l'
                                        objet au niveau 2
        169     D0=D0+  10
        146     C=DAT0  A               Lecture du contenu
        10A     R2=C
        103     R3=A
        84F     ST=0    15              Plus d'int. clav.
12      11C     C=R4
        8AE     ?C#0    A               Est ce terminé ?
        60      GOYES   13
        62D0    GOTO    18              Oui -> fin !
13      CE      C=C-1   A               1 de moins..
        10C     R4=C
        808F    INTOFF
        81B4    A=PC    A               A=adresse en
                                        mémoire de 'here'
here    346CFF7 LC(5)   #80000-(15)+(here)
        8BE     ?C<=A   A               Où est HRPEEK ?
        03      GOYES   15              Dans un module !
        3482000 LC(5)   (14)-(here)
        C2      C=C+A   A               C=adresse en
                                        mémoire de '14'
        2F      p=      15
        80C4    C=P     4               C=adresse de '14'
                                        après déplacement
                                        de la ram interne
                                        en #F0000h
        20      p=      0
```

```
        8FFB620   GOSBVL   #026BF      Déplacement de la
                                       ram interne et
                                       appel de la
                                       routine dont
                                       l'adresse est dans
                                       le champ A de C
        6160      GOTO     16
14      112       A=R2                 ;
        131       D1=A                 ;Lecture d'un
        AE0       A=0      B           ;quartet en R2 et
        15B0      A=DAT1   1           ;sauvegarde dans
        101       R1=A                 ;le registre R1
        01        RTN                  ;
15      340000'   LCHEX    #'0000      ;
        804       UNCNFG               ;
        340000F   LCHEX    #F0000      ;Déplacement ram
        805       CONFIG               ;en #60000h
        3400006   LCHEX    #60000      ;
        805       CONFIG               ;
        112       A=R2
        131       D1=A
        AE0       A=0      B
        15B0      A=DAT1   1           Lecture d'un
                                       quartet
        101       R1=A
        3400006   LCHEX    #60000      ;
        804       UNCNFG               ;
        340000F   LCHEX    #F0000      ;Remise de la ram
        805       CONFIG               ;en #'0000h
        340000'   LCHEX    #'0000      ;
        805       CONFIG               ;
16      8080      INTON                Interruptions ok
        111       A=R1
        3103      LCHEX    30          ;
        A6A       A=A+C    B           ;
        3193      LCHEX    39          ;Conversion du
        9EA       ?C>=A    B           ;quartet lu en
        90        GOYES    17          ;ASCII
        31'0      LCHEX    0'          ;
        A6A       A=A+C    B           ;
17      11B       C=R3
        134       D0=C
        148       DAT0=A   B           Ecriture
        161       D0=D0+   2
        136       CD0ex
        10B       R3=C
        11A       C=R2
        E6        C=C+1    A           Au suivant !
        10A       R2=C
        682F      GOTO     12          On itère..
18      85F       ST=1     15
        8F2D'60   GOSBVL   LOAD_REG    Récupérations
        1'4       D1=D1+   5           ;
        E'        D=D+1    A           ;DROP
        118       C=R0
```

```
          145       DAT1=C    A             Chaîne résultat
                                            dans la pile
          142       A=DAT0    A             Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin       B2130     CON(5)    EPILOGUE      Fin programme
```

HRPEEK (# 4305h)

```
D9D20 2ABF1 3FBF1 CCD20 35100 8FB97 60147 13416
91423 40FFF 78B64 0D6C6 8FD7B 50132 14713 41691
4610C 17414 71341 69146 10A10 384F1 1C8AE 6062D
0CE10 C808F 81B43 46CFF 78BE0 33482 000C2 2F80C
4208F FB620 61601 12131 AE015 B0101 01340 00078
04340 000F8 05340 00068 05112 131AE 015B0 10134
00006 80434 0000F 80534 00007 80580 80111 3103A
6A319 39EA9 03170 A6A11 B1341 48161 13610 B11AE
610A6 82F85 F8F2D 76017 4E711 81451 42164 808CB
2130
```

?ADR

Ce programme permet de connaître l'adresse de l'objet situé au premier niveau de la pile.

Voici le listing désassemblé et commenté de ?ADR:

```
          D9D20     CON(5)    PROL_PRGM      Objet programme.
          E4A20     CON(5)    PROL_INT       Entier nul (où
          A0000     CON(5)    #0000A         sera écrite
          00000     CON(5)    #00000         l'adresse)
          CB2A1     CON(5)    NEWOB          Recréation de
                                             l'entier
          DBBF1     CON(5)    SWAP
          CCD20     CON(5)    PROL_CODE      Objet code
début     62000     CON(5)    (fin)-(début)  Longueur objet
          147       C=DAT1    A              C=@ objet
          174       D1=D1+    5              On enlève l'
          E7        D=D+1     A              objet de la pile
          143       A=DAT1    A
          133       AD1ex
          179       D1=D1+    10
          145       DAT1=C    A              Ecriture @
          131       D1=A
          142       A=DAT0    A              Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin       B2130     CON(5)    EPILOGUE       Fin programme
```

```
?ADR (# 26A0h)
D9D20 E4A20 A0000 00000 CB2A1 DBBF1 CCD20 62000
14717 4E714 31331 79145 13114 21648 08CB2 130
```

SSAG

Ce programme permet de connaître le listing des codes de l'objet situé au premier niveau de la pile. Il réalise donc la fonction inverse de GASS, d'où son nom...

Il utilise les programmes PEEK et ?ADR précédement listés.

Pour déterminer la taille de l'objet, SSAG utilise l'appel syseval #1A1FC qui réalise la même fonction que BYTES, mais quel que soit le type de l'objet donné en argument (la fonction BYTES exécutée avec un nom global comme argument, renvoie la somme de contrôle et la longueur du contenu de ce nom).

L'objet à étudier est d'abord stocké dans la variable globale 'OBJ.TMP' de manière à lui conférer une adresse fixe.

Par exemple "123" SSAG renverra "C2A20B0000132333" qui est bien le listing de l'objet-chaîne contenant les trois caractères "1", "2" et "3" (codes ASCII #31h, #32h et #33h).

SSAG a été écrit par Dominique MOISESCU.

SSAG (# B7AFh)

```
«
  'OBJ.TMP' STO 'OBJ.TMP' RCL DUP ?ADR SWAP
  # 1A1FCh SYSEVAL SWAP DROP 2 * R→B PEEK 'OBJ.TMP'
  PURGE
»
```

RASS

RASS réalise la même fonction que GASS mais est écrit en assembleur...

Voici tout d'abord le listing commenté du programme, suivi de la liste des codes à entrer:

```
          D9D20     CON(5)    PROL_PRGM      Prologue programme
          78BF1     CON(5)    DUP            ;Vérifie présence
          8DBF1     CON(5)    DROP           ;d'au moins 1 elt
          CCD20     CON(5)    PROL_CODE      Prologue code
          BA000     CON(5)    (fin)-(début)  Longueur code
début     8FB978C   GOSBVL    SAVE_REG       Sauvegardes
          147       C=DAT1    A
          137       CD1ex                    D1=adresse chaîne
          109       R1=C
          174       D1=D1+    5
          143       A=DAT1    A              A=longueur chaîne
          34500000  LCHEX     #00005
          8A2       ?C=A      A              Chaîne vide ?
          57        GOYES     15             Oui -> fin
          EA        A=A-C     A
          81C       ASRB                     Nombre de codes
          103       R3=A
          174       D1=D1+    5
          137       CD1ex
          10A       R2=C
          D6        C=A       A
          84A       ST=0      10
11        8F8DA60   GOSBVL    #06AD8         Reservation mem.
          501       GONC      12             Ok !
          8FD3361   GOSBVL    #1633D         Garbage collector
          11B       C=R3
          6BEF      GOTO      11
12        119       C=R1
          135       D1=C
          132       AD0ex
          141       DAT1=A    A              Objet réservé dans
                                             la pile
          130       D0=A
          113       A=R3
          D8        B=A       A
          CD        B=B-1     A
          11A       C=R2
          135       D1=C
13        14B       A=DAT1    B              On lit un code
```

```
          3103      LCHEX     #30           ;
          B6A       A=A-C     B             ;
          3190      LCHEX     #09           ;Code ASCII
          9EA       ?C>=A     B             ;-> hexadécimal
          90        GOYES     14            ;
          31?0      LCHEX     #0?           ;
          B6A       A=A-C     B             ;
14        1580      DAT0=A    1             Ecriture
          160       D0=D0+    1
          1?1       D1=D1+    2
          CD        B=B-1     A             1 de moins
          59D       GONC      13            On continue si
                                            nécessaire
15        8F2D?60   GOSBVL    LOAD_REG      Récupérations
          142       A=DAT0    A             Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin       B2130     CON(5)    EPILOGUE      Fin prgm
```

RASS (# B5D3h)

```
D9D20 78BF1 8DBF1 CCD20 BA000 8FB97 60147 13710
91741 43345 00008 A257E A81C1 03174 13710 AD684
A8F8D A6050 18FD3 36111 B6BEF 11913 51321 41130
113D8 CD11A 13514 B3103 B6A31 909EA 90317 0B6A1
58016 0171C D59D8 F2D76 01421 64808 CB213 0
```

CHK

Ce programme sert à la vérification du nombre et du type des objets dans la pile. En lui-même, il ne présente aucun intérêt mais sera fort utile au programmeur désirant contrôler le nombre et le type des objets passés en paramètres à ses programmes.

CHK prend deux arguments dans la pile: deux entiers qui décrivent les nombre et types d'arguments requis.

Le premier de ces deux entiers est le nombre d'arguments. Il doit être compris entre 0 (aucun argument) et 8. Il se situera au niveau 2 de la pile.

Le second est la description des types. Chaque type est représenté par un nombre hexadécimal à deux chiffres selon la codification présentée dans le tableau page suivante.

Si la codification est mauvaise (nombre d'arguments supérieur à 8 ou type non valide) on obtiendra le message d'erreur "Bad Argument Value".

Si le nombre et les types des arguments aux niveaux 3 et suivants de la pile sont corrects (ils correspondent à la description), il ne se passe rien et les deux entiers de description sont détruits. Dans le cas contraire on obtiendra le message "Too Few Arguments" ou "Bad Argument Value".

Voici quelques exemples:

- Vérifier que la pile contient une chaîne de caractères et un objet quelconque: `#2h #0900h CHK`
- Vérifier que la pile contient deux entiers: `#2h #0A0Ah CHK`
- Vérifier que la pile contient 8 objets quelconques: `#8h #0h CHK`
- Vérifier que la pile contient un nom global et deux réels: `#3h #1A0202h CHK`

Code	Type	Prologue
00	Objet quelconque	
01	System Binary	02911
02	Real	02933
03	Long Real	02955
04	Complex	02977
05	Long Complex	0299D
06	Character	029BF
07	Array	029E8
08	Linked Array	02A0A
09	String	02A2C
0A	Binary Integer	02A4E
0B	List	02A74
0C	Directory	02A96
0D	Algebraic	02AB8
0E	Unit	02ADA
0F	Tagged	02AFC
10	Graphic	02B1E
11	Library	02B40
12	Backup	02B62
13	Library Data	02B88
14	Reserved 1	02BAA
15	Reserved 2	02BCC
16	Reserved 3	02BEE
17	Reserved 4	02C10
18	Program	02D9D
19	Code	02DCC
1A	Global Name	02E48
1B	Local Name	02E6D
1C	XLIB name	02E92

Codage des types d'objets

Voici tout d'abord le listing commenté de CHK, suivi du rappel des codes à entrer.

```
CCD20               CON(5)    PROL_CODE      Objet code
99100     debut     CON(5)    (fin)-(debut)  Longueur code
8FB97'60            GOSBVL    SAVE_REG       Sauvegardes
AF0                 A=0       W              ;Première vérif:
808202              LAHEX     #2             ;les arguments de
AF2                 C=0       W              ;CHK: deux
33A0A0              LCHEX     #0A0A          ;entiers..
'27'0               GOSUB     chk            ;
8F2D'60             GOSBVL    LOAD_REG       Récupération
1'79                D1=D1+    10             ;DROP des deux
E'7                 D=D+1     A              ;entiers
E'7                 D=D+1     A
8FB9'60             GOSBVL    SAVE_REG       Sauvegardes
1C9                 D1=D1-    10
3480000             LCHEX     #00008         Max. args
D5                  B=C       A
147                 C=DAT1    A
134                 D0=C
169                 D0=D0+    10
1567                C=DAT0    W              C(W)=types
174                 D1=D1+    5
143                 A=DAT1    A
130                 D0=A
169                 D0=D0+    10
1527                A=DAT0    W
174                 D1=D1+    5
8B0                 ?A>B      A              Plus de 8 args ?
'71                 GOYES     err1           Oui -> erreur
'7820               GOSUB     chk            Verification
8F2D'60             GOSBVL    LOAD_REG       Récupérations
142                 A=DAT0    A              Retour au RPL
164                 D0=D0+    5
808C                PC=(A)

3430200   err1      LCHEX     #00203         Erreur: Bad Arg.
                                             Value
DAerr               A=C       A
8F2D'60             GOSBVL    LOAD_REG       Récupérations
8D32050             GOVLNG    #05023         Erreur

'7190     chk       GOSUB     chk2           Ce gosub sert à
                                             déterminer
                                             l'adresse de début
                                             de la liste de
                                             prologues qui
                                              suit..

00000               CON(5)    #00000         Objet qcq.
11920               CON(5)    #02911         System Binary
33920               CON(5)    #02933         Real
```

```
55920           CON(5)   #02955      Long Real
77920           CON(5)   #02977      Complex
D9920           CON(5)   #0299D      Long Complex
FB920           CON(5)   #029BF      Character
8E920           CON(5)   #029E8      Array
A0A20           CON(5)   #02A0A      Linked Array
C2A20           CON(5)   #02A2C      String
E4A20           CON(5)   #02A4E      Binary Integer
47A20           CON(5)   #02A74      List
69A20           CON(5)   #02A96      Drectory
8BA20           CON(5)   #02AB8      Algebraic
ADA20           CON(5)   #02ADA      Unit
CFA20           CON(5)   #02AFC      Tagged
E1B20           CON(5)   #02B1E      Graphic
04B20           CON(5)   #02B40      Library
26B20           CON(5)   #02B62      Backup
88B20           CON(5)   #02B88      Library Data
AAB20           CON(5)   #02BAA      Reserved 1
CCB20           CON(5)   #02BCC      Reserved 2
EEB20           CON(5)   #02BEE      Reserved 3
01C20           CON(5)   #02C10      Reserved 4
D9D20           CON(5)   #02D9D      Program
CCD20           CON(5)   #02DCC      Code
84E20           CON(5)   #02E48      Global Name
D6E20           CON(5)   #02E6D      Local Name
29E20           CON(5)   #02E92      XLIB name

D8      chk2    B=A      A           Nombre obj.
AF7             D=C      W           Types
07      L1      C=RSTK               C=adresse de début
                                     de la liste
8A9             ?B=0     A           fini ?
00              RTNYES               Oui !
134             D0=C
06              RSTK=C               Sauvegarde
31D1            LCHEX    #1D
9E7             ?D<C     B           Type ok ?
60              GOYES    L2
693F            GOTO     err1        Non -> erreur
96B     L2      ?D=0     B           ;
C0              GOYES    L3          ;Recherche du
A6F             D=D-1    B           ;prologue
164             D0=D0+   5           ;
64FF            GOTO     L2          ;
147     L3      C=DAT1   A
8AE             ?C#0     A
D0              GOYES    L4
3410200         LCHEX    #00201      Fond de pile
                                     atteint -> erreur
                                     (Too Few
                                     Arguments)
6E1F            GOTO     err
137     L4      CD1EX
143             A=DAT1   A           A=prologue obj.
135             D1=C
```

```
146                 C=DAT0    A
8AA                 ?C=0      A             Objet qcq ?
21                  GOYES     15            Oui -> ok
8A2                 ?A=C      A             Prologue ok ?
D0                  GOYES     15            oui -> 15
3420200             LCHEX     #00202        Le prologue de
                                            l'objet ne
                                            correspond pas au
                                            prologue requis ->
                                            "Bad Argument
                                            Type"..
6DFE                GOTO      err

CD        15        B=B-1     A             Un de moins
BF7               DSR       W             Type suivant
BF7               DSR       W
174               D1=D1+    5             Obj. suivant
689F              GOTO      11            On itère..
          fin
```

CHK (# FD7Ch)

```
CCD20 99100 8FB97 60AF0 80820 2AF23 3A0A0 72708
F2D76 0179E 7E78F B9760 1C934 80000 D5147 13416
91567 17414 31301 69152 71748 B0717 8208F 2D760
14216 4808C 34302 00DA8 F2D76 08D32 05071 90000
00119 20339 20559 20779 20D99 20FB9 208E9 20A0A
20C2A 20E4A 2047A 2069A 208BA 20ADA 20CFA 20E1B
2004B 2026B 2088B 20AAB 20CCB 20EEB 2001C 20D9D
20CCD 2084E 20D6E 2029E 20D8A F7078 A9001 34063
1D19E 76069 3F96B C0A6F 16464 FF147 8AED0 34102
006E1 F1371 43135 1468A A218A 2D034 20200 6DFEC
DBF7B F7174 689F
```

REVERSE

REVERSE renverse les chaînes de caractères. Par exemple "123" REVERSE renverra la chaîne "321".

Ce programme sert à retourner les informations lues avec PEEK.

En effet le microprocesseur SATURN possède la particularité d'écrire les données 'à l'envers'. Il faut donc les renverser pour en obtenir la vraie signification...

Voici tout d'abord le listing désassemblé et commenté de REVERSE. Il est suivi par le rappel des codes à entrer.

```
          D9D20     CON(5)    PROL_PRGM     Objet programme
          FD550     CON(5)    #055DF        Chaîne vide
          76BA1     CON(5)    add           +
          CCD20     CON(5)    PROL_CODE     Objet code
début     86000     CON(5)    (fin)-(début) Longueur code
          8FB976O   GOSBVL    SAVE_REG      Sauvegardes
          143       A=DAT1    A
          131       D1=A                    D1=adresse chaîne
          174       D1=D1+    5
          137       CD1ex
          135       D1=C
          143       A=DAT1    A             A=longueur chaîne
          C2        C=C+A     A
          134       D0=C
          174       D1=D1+    5             D1=adresse premier
                                            caractère
          181       D0=D0-    2             D0=adresse dernier
                                            caractère
          818F84    A=A-5     A
          8A8       ?A=0      A             Chaîne vide ?
          52        GOYES     12            Oui -> fin
11        14B       A=DAT1    B             ;
          14E       C=DAT0    B             ;Echande de deux
          14D       DAT1=C    B             ;caractères
          148       DAT0=A    B             ;
          171       D1=D1+    2
          181       D0=D0-    2
          133       AD1ex
          131       D1=A
          136       CD0ex
          134       D0=C
          8BA       ?C>=A     A             Encore ?
```

```
          FD        GOYES     11
12        8F2D760   GOSBVL    LOAD_REG       Récupérations
          142       A=DAT0    A              Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin       B2130     CON(5)    EPILOGUE       Fin programme
```

REVERSE (# AA7Dh)

```
D9D20 FD550 76BA1 CCD20 86000 8FB97 60143 13117
41371 35143 C2134 17418 1818F 848A8 5214B 14E14
D1481 71181 13313 11361 348BA FD8F2 D7601 42164
808CB 2130
```

CRNAME

CRNAME est un programme permettant la création de noms globaux quelconques (y compris des noms "étranges" ne pouvant pas être entrés au clavier, ou des noms de fonctions déjà existantes).

L'intérêt d'un tel programme est double:

- créer des variables de noms interdits qui sont de ce fait difficiles à détruire, à visiter et à modifier (d'où une certaine sécurité);
- créer des variables ayant le même nom que des fonctions internes de la HP48 dans le but de les remplacer (les programmes utilisateur ayant priorité sur les programmes internes lorsque les noms sont tapés en toutes lettres).

CRNAME (# 11E9h)

```
«
  1 127 SUB 116 CHR 42 CHR + 128 CHR + 228 CHR + 2
  CHR + OVER SIZE CHR + SWAP + 43 CHR + 49 CHR + 0
  CHR + # 4003h SYSEVAL # 56B6h SYSEVAL DROP NEWOB
  1 GET
»
```

Le principe de ce programme est le même que GASS: on crée un objet classique (ici une chaîne de caractères) contenant les codes de l'objet désiré (le nom dans une liste), puis on ôte les informations propres à l'objet contenant pour ne garder que l'objet contenu.

Ici il faut supprimer le prologue et la longueur de la chaîne, soit 2 fois 5 quartets. On utilise pour ce faire la routine en #56B6h qui prend comme argument un "system binary" contenant le nombre de cases de 5 quartets à ôter.

Ce paramètre existe déjà en rom (voir la liste d'objets utiles située en annexe) à l'adresse #4003h. On le rappelle dans la pile par #4003h SYSEVAL. Après le NEWOB, la liste contenant le nom désiré est dans la pile: il ne reste plus qu'à faire 1 GET pour l'obtenir.

CLVAR

L'instruction CLVAR permet de nettoyer la mémoire de la HP48 en détruisant toutes les variables-utilisateur.

Or cette commande est effectuée très simplement en appuyant sur 2 touches ([↱] puis [DEL]).

Entre les mains d'un utilisateur non averti, la HP48 risque donc fortemment l'amnésie...

Il est donc utile d'enlever l'accès à cette commande. Ceci peut se faire grâce au programme CRNAME en procédant de la manière suivante:

- Entrer un programme quelconque. Par exemple:

```
« "No CLVAR Available !" DOERR »
```

- Taper ensuite:

```
"CLVAR" CRNAME STO
```

Il est conseillé d'installer ce faux CLVAR dans le répertoire 'HOME' de manière à ce qu'il soit partout exécutable...

Pour avoir de nouveau accès au véritable CLVAR, il suffit de taper:

```
'CLVAR' PURGE
```

SYSEVAL

L'instruction SYSEVAL permet l'évaluation d'objets situés dans la mémoire de la HP48.

Son utilisation hasardeuse peut conduire à une perte des données-utilisateur... on peut donc la considérer comme dangereuse !

Il est donc intéressant de pouvoir en empêcher l'utilisation...

Cela est très facile: il suffit de créer un programme effectuant une fonction quelconque ayant comme nom 'SYSEVAL'.

Cependant on ne peut entrer directement un tel nom.

Le programme 'CRNAME' (listé précédemment) permet une telle création.

Pour inhiber SYSEVAL, il faut donc procéder ainsi:

- Taper le programme suivant (par exemple):

```
« "No SYSEVAL Available !" DOERR »
```

- Taper ensuite:

```
"SYSEVAL" CRNAME STO
```

Il est conseillé d'installer ce faux SYSEVAL dans le répertoire 'HOME' de manière à ce qu'il soit partout exécutable...

Pour remettre la machine dans un état standard, il faut détruire le programme SYSEVAL en tapant `'SYSEVAL' PURGE` (on peut alors taper directement le nom du programme).

CONTRAST

CONTRAST est un programme utilisant PEEK et POKE qui permet de régler le contraste de l'écran de manière logicielle (ce réglage se fait normalement par les combinaisons de touches [ON] [+] et [ON] [-]).

Il prend en argument un entier compris entre #0h et #1Fh correspondant au contraste désiré (#0h pour un contraste minimal, écran quasiment éteint à #1Fh pour un contraste maximal, écran complétement noir).

Ce programme permet d'atteindre plus de valeurs de contraste que les combinaisons [ON] [+] et [ON] [-] puisque celles-ci se limitent à la plage de valeurs #3h à #13h...

```
CONTRAST (# 7BF1h)
«
  HEX # 101h OVER # Fh AND →STR 3 3 SUB "#" # 102h
  # 1h PEEK + STR→ # Eh AND 4 ROLL 16 / # 1h AND OR
  →STR 3 3 SUB + POKE
»
```

DISPOFF DISPON

DISPOFF et DISPON sont deux programmes utilisant PEEK et POKE qui permettent respectivement d'éteindre et d'allumer l'écran de la HP48.

Il faut noter que DISPOFF inhibe le clavier, en conséquence de quoi DISPOFF et DISPON doivent-ils toujours être utilisés conjointement au sein d'un programme...

Si par mégarde DISPOFF était lancé seul, il deviendrait impossible de rallumer l'écran par un autre moyen que l'arrêt système [ON] [C]...

DISPOFF (# 8EF6h)

```
«
  # 100h "#" OVER # 1h PEEK + STR→ # 7h AND →STR 3 3
  SUB POKE
»
```

DISPON (# 10B7h)

```
«
  # 100h "#" OVER # 1h PEEK + STR→ # 8h OR →STR 3 3
  SUB POKE
»
```

FAST

FAST est un programme permettant d'augmenter la vitesse de calcul de la HP48 de plus de 12%...

Il prend en entrée un programme, le nom d'un programme ou une liste de commandes à effectuer rapidement. Cet argument doit être précédé des arguments nécessaires aux différentes commandes à réaliser.

Ce programme fonctionne en éteignant l'écran (grâce aux programmes DISPOFF et DISPON). Cette extinction "soulage" le bus de la HP48 ce qui a pour effet d'accélérer son fonctionnement.

Comme signalé dans les remarques sur DISPOFF, l'arrêt du programme (par erreur, par exemple) en cours d'exécution, conduirait la HP48 à se retrouver en mode écran éteint, clavier inhibé.

Dans ce cas, il faudrait effectuer un arrêt système ([ON] [C]) pour la remettre dans un état normal...

Exemple: calculer rapidement la dérivée seconde de 'COS(COS(X))':

```
« 'COS(COS(X))' 'X' ∂ 'X' ∂ » FAST
```

```
FAST (# 8EAAh)
«
  DISPOFF EVAL DISPON
»
```

DESASS

Ce programme passionnant ne nécessite pas de commentaire. C'est un programme monstrueux par sa taille mais fort utile puisqu'il est capable de désassembler tout programme en langage-machine.

Il prend deux arguments en entrée:

- au niveau 2 de la pile, une chaîne de caractères contenant la liste des codes hexadécimaux les uns à la suite des autres de la façon habituelle;
- au niveau 1, l'adresse de départ du code. Dans le cas de programmes relogeables (c'est-à-dire indépendants de leur adresse de stockage comme tous les programmes de ce livre) on donnera la valeur #0h à cet argument. Celui-ci n'est utile que dans le cas de désassemblage de programmes pris dans la mémoire morte de la HP48 grâce au programme PEEK. Ainsi pour désassembler la routine située à l'adresse #067B9h, il suffira de faire:

```
#067B9h DUP #100h PEEK SWAP DESASS
```

DESASS est le programme principal, toutes les autres routines ne sont que des sous-programmes utilisés par DESSAS.

L'exécution de DESASS peut se terminer par une erreur lorsqu'il manque des données ou si il rencontre un code invalide (10E par exemple). De plus le désassemblage réalisé est sauvegardé dans la variable 'SOL'.

Il faut noter que DESASS ne désassemble que du langage-machine, et ne prend donc pas en compte les prologues d'objets...

Remarque: les programmes SPC1 et SPC2 sont semblables. Ils ont été dissociés pour permettre un paramétrage de l'aspect des listings obtenus...

```
DESASS (# 489h)
«
  HEX 64 STWS 'ADR' STO 'Z' STO "        - DEBUT -"
  10 CHR + 'SOL' STO 1 'P' STO Z SIZE → S
  «
    DO
      P 'I' STO L READ 1 + GET EVAL + STOS
    UNTIL
      P S >
    END
      "        - FIN - " STOS
  »
»

TAKE (# 7AFDh)
«
  Z P DUP SUB
»

READ (# 3949h)
«
  "#" Z P DUP SUB + STR→ B→R
»

INC (# C417h)
«
  1 'P' STO+
»

STOS (# 3095h)
«
  10 CHR + DUP 1 DISP SOL SWAP + 'SOL' STO INC
»

L (# EB37h)
( A0 A1 A2 A3 A4 A5 A6 A7 A1 A9 AA AB AC AC AC AC )
```

```
A0 (# A89Bh)
«
  INC READ DUP
  IF
    14 ≠
  THEN
    { "RTNSXM" "RTN" "RTNSC" "RTNCC" "SETHEX"
    "SETDEC" "RSTK=C" "C=RSTK" "CLRST" "C=ST"
    "ST=C" "CSTex" "P=P+1" "P=P-1" 14 "RTI" } SWAP
    1 + GET CODE SWAP
  ELSE
    DROP INC READ INC READ
    → x y
    «
      y 8 < 38 CHR 33 CHR IFTE
      → z
      «
        y 8 MOD 2 * 1 + "ABBCCADCBACBACCD"
        → t u
        «
          u t DUP SUB u t 1 + DUP SUB
          → a b
          «
            CODE a "=" a z b + + + + SPC2 +
            IF
              x 15 ==
            THEN
              "A"
            ELSE
              x CH
            END
          »
        »
      »
    »
  END
»
```

```
A1 (# 484Eh)
«
  ( N M ) "18" READ →STR POS GET INC READ 1 + GET
  EVAL
»

N (# 956Ch)
( C0 C0 C0 C0 C4 C4 C6 C6 C6 C9 C9 C9 C6 C9 C9 C9 )

C0 (# 6508h)
«
  TAKE INC CODE "P" 3 ROLL + STR→ READ 1 + GET
»

C6 (# F0DAh)
«
  ( "D0=D0+" "D1=D1+" "D0=D0-" "D1=D1-" ) READ 5 -
  DUP 4 > 3 * - GET INC CODE SWAP SPC2 READ 1 +
  →STR +
»

C9 (# 95A9h)
«
  READ 8 - DUP
  IF
    3 >
  THEN
    4 - "D1=("
  ELSE
    "D0=("
  END
  ( 2 4 5 ) ROT GET SWAP OVER + ")" + SPC2 SWAP 1 -
  → x
  «
    INC Z P DUP x + SUB REVERSE + P x + 'P' STO
    CODE SWAP
  »
»
```

```
C4 (# D7A3h)
«
  READ INC READ
  → x y
  «
    { "DAT0=A" "DAT1=A" "A=DAT0" "A=DAT1" "DAT0=C"
    "DAT1=C" "C=DAT0" "C=DAT1" } y 8 MOD 1 + GET
    SPC2
    IF
      x 4 ==
    THEN
      IF
        y 8 <
      THEN
        "A"
      ELSE
        "B"
      END
    ELSE
      INC READ
      → z
      «
        IF
          y 8 <
        THEN
          z CH
        ELSE
          READ 1 + →STR
        END
      »
    END
    +
  »
  CODE SWAP
»

P0 (# E419h)
{ "R0=A" "R1=A" "R2=A" "R3=A" "R4=A" 5 6 7 "R0=C"
  "R1=C" "R2=C" "R3=C" "R4=C" }
```

```
P1 (# 9F7h)
{ "A=R0" "A=R1" "A=R2" "A=R3" "A=R4" 5 6 7 "C=R0"
  "C=R1" "C=R2" "C=R3" "C=R4" }

P2 (# D1C7h)
{ "AR0ex" "AR1ex" "AR2ex" "AR3ex" "AR4ex" 5 6 7
  "CR0ex" "CR1ex" "CR2ex" "CR3ex" "CR4ex" }

P3 (# 7E1Bh)
{ "D0=A" "D1=A" "AD0ex" "AD1ex" "D0=C" "D1=C"
  "CD0ex" "CD1ex" "D0=AS" "D1=AS" "AD0XS" "AD1XS"
  "D0=CS" "D1=CS" "CD0XS" "CD1XS" }

A2 (# 856Ah)
«
  INC CODE "P=" SPC2 READ →STR +
»

A31 (# 6DCAh)
«
  INC READ
   → x
  «
    SPC2 Z INC P DUP x + DUP 'P' STO SUB REVERSE +
  »
»

A7 (# 1C34h)
«
  "GOSUB" "" 1 3
  START
    INC TAKE +
  NEXT
  # 1000h 4 SAUTREL CODE SWAP
»
```

```
A3 (# DB24h)
«
  "LCHEX " A31 CODE SWAP
»

A4 (# A72Dh)
«
  INC TAKE INC TAKE + DUP
  IF
    "00" ==
  THEN
    DROP "RTNC"
  ELSE
    DUP
    IF
      "20" ==
    THEN
      DROP "NOP3"
    ELSE
      "GOC" SWAP # 100h 1 SAUTREL
    END
  END
  CODE SWAP
»

A5 (# 4081h)
«
  INC TAKE INC TAKE + DUP
  IF
    "00" ==
  THEN
    DROP "RTNNC"
  ELSE
    "GONC" SWAP # 100h 1 SAUTREL
  END
    CODE SWAP
»
```

```
A6 (# A19Ch)
«
  Z INC P DUP 3 + SUB DUP
  IF
    1 3 SUB "300" ==
  THEN
    DROP "NOP4"
  ELSE
    DUP
    IF
      "4000" ==
    THEN
     DROP "NOP5" INC
    ELSE
     1 3 SUB "GOTO" SWAP # 1000h 1 SAUTREL
    END
  END
  INC INC CODE SWAP
»

M (# CC5Ch)
( B0 B1 B1 B3 B4 B4 B6 B6 B6 B6 BA BA BC BC BC BC )

B1 (# 9732h)
«
  "U" TAKE + STR→ INC READ 1 + GET EVAL CODE SWAP
»

B3 (# FA87h)
«
  B1 GOYES
»

B4 (# 5589h)
«
  ( "ST=0" "ST=1" ) READ 3 - GET INC SPC2 READ →STR
  + CODE SWAP
»
```

```
B0 (# E5CDh)
«
  0 U0 INC READ
  → x
  «
    x 1 + GET
    IF
      x 8 ==
    THEN
      DROP2 INC ( 6 7 10 11 ) READ POS V0 READ 1 +
      GET EVAL
    ELSE
      IF
        ( 15 13 12 ) x POS
      THEN
        SPC2 INC TAKE +
      END
    END
    CODE SWAP
    IF
      ROT
    THEN
      GOYES
    END
  »
»

B6 (# 390Bh)
«
  ( "?ST=0" "?ST≠0" "?P≠" "?P=" ) READ 5 - GET INC
  SPC2 READ →STR + CODE SWAP GOYES
»

U0 (# 560Fh)
( "OUT=CS" "OUT=C" "A=IN" "C=IN" "UNCNFG" "CONFIG"
  "C=ID" "SHUTDN" 8 "C+P+1" "RESET" "BUSCC" "C=P"
  "P=C" "SREQ?" "CPex" )
```

```
BA (# 2958h)
«
  READ INC READ
  → x y
  «
    CODE
    IF
      x 10 ==
    THEN
      A
    ELSE
      B
    END
      y 1 + GET SPC2 + "A" GOYES
  »
»

BC (# 2CCCh)
«
  { "GOLONG" 4 "GOVLNG" 5 "GOSUBL" 4 "GOSBVL" 5 }
  READ 2 * 23 - DUP 1 + SUB LIST→ DROP
  → a b
  «
    a Z P 1 + DUP b + 1 - SUB
    IF
      b 5 ==
    THEN
      SWAP SPC2 SWAP REVERSE +
    ELSE
      # 10000h 2 READ 14 == 4 * + SAUTREL
    END
    P b + 'P' STO CODE SWAP
  »
»

V0 (# E524h)
{ "INTON" V01 V02 "BUSCB" V04 V04 V04 V04 V04 V04
  V04 V04 "PC=(A)" "BUSCD" "PC=(C)" "INTOFF" }
```

```
U18 (# 8795h)
«
  READ 8 == INC READ INC READ 1 +
  → t f r
  «
    RA r GET
    IF
      t
    THEN
      DUP "=" SWAP + +
      IF
        r 8 <
      THEN
        "+"
      ELSE
        "-"
      END
      + INC READ 1 + +
    ELSE
      "SRB" +
    END
    f CHA
  »
»

V00 (# 33A5h)
( "ABIT=0" "ABIT=1" "?ABIT=0" "?ABIT=1" "CBIT=0"
  "CBIT=1" "?CBIT=0" "?CBIT=1" )

V01 (# 22D6h)
«
  INC "RSI"
»

V02 (# 2584h)
«
  "LAHEX " A31
»
```

```
V04 (# C703h)
«
  V00 READ 3 - GET SPC2 INC TAKE +
»

U1 (# CFB0h)
{ "ASLC" "BSLC" "CSLC" "DSLC" "ASRC" "BSRC" "CSRC"
  "DSRC" U18 U18 U1A U1B "ASRB" "BSRB" "CSRB" "DSRB"
}

U1A (# BF19h)
«
  INC READ INC READ INC READ 1 +
  → f x r
  «
    RN r GET
    IF
      r 8 <
    THEN
      "A"
    ELSE
      "C"
    END
    IF
      x 2 ==
    THEN
      SWAP "ex" + +
    ELSE
      IF
        x 1 ==
      THEN
        SWAP
      END
        "=" SWAP + +
    END
    f CHA
  »
»
```

```
V1B (# BA48h)
{ 0 1 "PC=A" "PC=C" "A=PC" "C=PC" "APCex" "CPCex" }

U1B (# CC94h)
«
  V1B INC READ 1 + GET SPC2 "A" +
»

RN (# FC36h)
{ "R0" "R1" "R2" "R3" "R4" 5 6 7 "R0" "R1" "R2"
  "R3" "R4" 13 14 15 }

RA (# 8ACEh)
{ "A" "B" "C" "D" 4 5 6 7 "A" "B" "C" "D" 12 13 14
  15 }

U2 (# 5EDBh)
{ 0 "XM=0" "SB=0" 3 "SR=0" 5 6 7 "MP=0" 9 10 11 12
  13 14 "CLRHST" }

U3 (# EA2Ch)
{ 0 "?XM=0" "?SB=0" 3 "?SR=0" 5 6 7 "?MP=0" }

A9 (# 48ADh)
«
  A B NORMAL GOYES
»

AA (# 2CB0h)
«
  C D NORMAL
»

AB (# B467h)
«
  E F NORMAL
»
```

```
AC (# BF15h)
«
  ( C D E F ) READ 11 - GET EVAL INC CODE SWAP READ
  1 + GET SPC2 "A" +
»

A (# DD35h)
( "?A=B" "?B=C" "?C=A" "?D=C" "?A≠B" "?B≠C" "?C≠A"
  "?D≠C" "?A=0" "?B=0" "?C=0" "?D=0" "?A≠0" "?B≠0"
  "?C≠0" "?D≠0" )

B (# 32E9h)
( "?A>B" "?B>C" "?C>A" "?D>C" "?A<B" "?B<C" "?C<A"
  "?D<C" "?A≥B" "?B≥C" "?C≥A" "?D≥C" "?A≤B" "?B≤C"
  "?C≤A" "?D≤C" )

C (# 50AAh)
( "A=A+B" "B=B+C" "C=C+A" "D=D+C" "A=A+A" "B=B+B"
  "C=C+C" "D=D+D" "B=B+A" "C=C+B" "A=A+C" "C=C+D"
  "A=A-1" "B=B-1" "C=C-1" "D=D-1" )

D (# 9930h)
( "A=0" "B=0" "C=0" "D=0" "A=B" "B=C" "C=A" "D=C"
  "B=A" "C=B" "A=C" "C=D" "ABex" "CBex" "CAex"
  "CDex" )

E (# C345h)
( "A=A-B" "B=B-C" "C=C-A" "D=D-C" "A=A+1" "B=B+1"
  "C=C+1" "D=D+1" "B=B-A" "C=C-B" "A=A-C" "C=C-D"
  "A=B-A" "B=C-B" "C=A-C" "D=C-D" )

F (# 7B66h)
( "ASL" "BSL" "CSL" "DSL" "ASR" "BSR" "CSR" "DSR"
  "A=-A" "B=-B" "C=-C" "D=-D" "A=-A-1" "B=-B-1"
  "C=-C-1" "D=-D-1" )

SPC (# EA19h)
"       " (7 espaces)
```

```
SPC1 (# DF86h)
«
  SPC 1 7 4 PICK SIZE - SUB + " " +
»

SPC2 (# DF86h)
«
  SPC 1 7 4 PICK SIZE - SUB + " " +
»

ADRSTR (# 1EF0h)
«
  # 100000h + →STR 4 8 SUB
»

SAUTREL (# D63Eh)
«
  → a b c
  «
    SPC2 ADR I + 1 - c + "#" a REVERSE + OBJ→ DUP
    IF
      b 2 / <
    THEN
      +
    ELSE
      b SWAP - -
    END
    ADRSTR +
  »
»

CODE (# A7D6h)
«
  ADR I 1 - + ADRSTR " " + Z I P SUB SPC1 +
»
```

```
GOYES (# E103h)
«
  + INC P 'I' STO TAKE INC TAKE +
  → a
  «
    10 CHR CODE
    IF
      a "00" ==
    THEN
      "RTNYES"
    ELSE
      "GOYES" a # 100h 0 SAUTREL
    END
    + +
  »
»

NORMAL (# B551h)
«
  → a b
  «
    INC READ INC READ
    → x y
    «
      CODE
      IF
        x 8 <
      THEN
        a
      ELSE
        b
      END
      y 1 + GET SPC2 x CH +
    »
  »
»
```

```
REVERSE (# B227h)
«
  → c
  «
   "" c SIZE 1
    FOR x
      c x DUP SUB + -1
    STEP
  »
»

CH (# 989Eh)
«
  → a
  «
    ( "P" "WP" "XS" "X" "S" "M" "B" "W" ) a 8 MOD 1
    + GET
  »
»

CHR (# FDECh)
«
  → f
  «
    SPC2
    IF
      f 15 ==
    THEN
      "A"
    ELSE
      f CH
    END +
  »
»
```

Manipulation d'entiers système

Ces programmes réalisent des conversions entre entiers système (SB pour System Binary) et différents types de données utilisées par la machine: entiers binaires (B pour Binary Integer), réels (R pour Real) et caractères (C pour Character). Remarques:

- Aucune vérification de type n'est faite par les programmes. Il faut donc être certain de bien appeler le bon (sous peine d'obtenir des résultat faux mais sans danger pour la machine);
- Le dernier type d'objet (character) n'est pas un objet normalement accessible à l'utilisateur. Grâce à ces programmes on peut facilement en générer. Par exemple pour créer le caractère #40h ('A'), il suffit de taper:

  ```
  #40h B→SB SB→C
  ```

  Le caractère correspondant sera affiché `"Character"` par la HP48.

```
B→SB (# A92h)
« # 5A03h SYSEVAL »

SB→B (# C4F4h)
« # 59CCh SYSEVAL »

R→SB (# 41Ch)
« # 18CEAh SYSEVAL »

SB→R (# F1E1h)
« # 18DBFh SYSEVAL »

C→SB (# 2100h)
« # 5A51h SYSEVAL »

SB→C (# 2756h)
« # 5A75h SYSEVAL »
```

ROMRCL

Ce programme, très court, permet de rappeller dans la pile l'objet dont l'adresse est donnée en argument (sous forme d'entier).

Pour ce faire, il utilise le syseval #C612h qui rappelle dans la pile l'objet dont l'adresse est passée sous forme de "system binary".

On utilise donc le programme B->SB qui réalise la transformation d'un entier en ce type d'objet.

Remarques:

- Ce programme permet de rappeler les objets situés en rom cachée en les dupliquant en mémoire vive...
- Ne pas utiliser des adresses "au hasard";
- N'utiliser ROMRCL que sur des adresses en mémoire morte.

```
ROMRCL (# B490h)
«
   B→SB # C612h SYSEVAL
»
```

A->STR et STR->A

A→STR réalise la transformation d'une adresse sous forme d'entier en la chaîne de caractères la représentant (écrite à l'envers).

STR→A, quant à lui, réalise la fonction réciproque.

Ils sont particulièrement utiles lorsque l'on se sert de PEEK et POKE pour lire ou écrire des adresses en mémoire.

Ils utilisent tous deux le programme REVERSE.

Exemples:

#70000h A→STR renvoie "00007".

"0000F" STR→A renvoie #F0000h (en mode hexadécimal).

```
A→STR (# E4F3h)
«
   HEX # 100000h + # 1FFFFFh AND →STR REVERSE 2 6
   SUB
»
```

```
STR→A (# 9287h)
«
  "00000" + 1 5 SUB "h" SWAP + "#" + REVERSE STR→
»
```

SEARCH

Voici 3 programmes de recherche en mémoire: ROMSEARCH, RAMSEARCH et MODUSEARCH. Ils permettent de rechercher la ou les adresse(s) correspondant à une chaîne de codes donnée en entrée, respectivement dans:

- La mémoire morte (y compris la rom cachée) pour ROMSEARCH. Les adresses supérieures à #70000h (objets en rom cachée) seront à utiliser avec le programmes ROMRCL;
- La mémoire vive (y compris les modules "mergés") pour RAMSEARCH;
- Dans les modules (uniquement sur HP48sx) pour MODUSEARCH. Ce dernier prend un argument de plus que les deux autres: en plus de la chaîne à rechercher, il faut lui donner un réel indiquant le numéro du port concerné. Après vérification de la validité d'un tel port (module présent et non "mergé") la recherche sera effectuée.

Remarque: ces trois programmes utilisent le programme SEARCH listé ci-dessous, ainsi que PEEK, HRPEEK (pour ROMSEARCH) et STR→A (pour RAMSEARCH et MODUSEARCH).

Exemples: chercher tous les objets "chaîne de caractère" dans la rom:

```
"C2A20" ROMSEARCH
```

Même recherche dans le module en port 2 (si un tel module est présent):

```
"C2A20" 2 MODUSEARCH
```

```
SEARCH (# EC79h)
«
  → MOTIF AD FIN PRGM
  «
    # 100h DUP MOTIF SIZE + → LEN LENP
    «
      ( )
      DO
        AD DUP 1 DISP LENP PRGM EVAL
        IF
          MOTIF POS AD OVER
        THEN
          + DUP 'AD' STO 1 - DUP
          IF
            FIN ≥
          THEN
            DROP
          ELSE
            DUP 2 DISP 1000 .07 BEEP +
          END
        ELSE
        + LEN + 'AD' STO
        END
      UNTIL
        AD FIN ≥
      END
    »
  »
»


ROMSEARCH (# 5E4Eh)
«
  → MOTIF
  «
    MOTIF # 0h # 70000h 'PEEK' SEARCH MOTIF # 70000h
    # 80000h 'HRPEEK' SEARCH +
  »
»
```

```
RAMSEARCH (# 88ABh)
«
  # 70000h # 70669h # 5h PEEK STR→A 'PEEK' SEARCH
»

MODUSEARCH (# C06Dh)
«
  → PORT
  «
    IF
      PORT 1 ≠ PORT 2 ≠ AND
    THEN
      # Ah DOERR
    END
    # 70421h PORT 11 * + → ADR
    «
      ADR # 1h PEEK STR→A → FLAGS
      «
        IF
          FLAGS # 8h AND # 0h ==
        THEN
          # Ah DOERR
        END
        IF
          FLAGS # 2h AND # 0h ≠
        THEN
          "PORT MERGED-USE RAMS" DOERR
        END
      »
      ADR 1 + # 5h PEEK STR→A DUP # 100000h ADR 6 +
      # 5h PEEK STR→A - + 'PEEK' SEARCH
    »
  »
»
```

CRC

Ce programme réalise le calcul des codes de redondance cyclique (CRC) utilisés par la HP48 pour garantir l'intégrité du contenu de certains de ces objets.

Ce programme prend en entrée une chaîne de codes (comme celle prise en argument par GASS) et renvoie le checksum correspondant. Par exemple:

"123456789ABCDEF0" CRC renverra #A8ECh dans la pile.

```
CRC (# 9D00h)
«
  # 0h
  → S CRC.V
  «
    1 S SIZE
    FOR X
      S X X SUB NUM 48 - DUP 9 > 7 * - # 0h + CRC.V
      16 / SWAP CRC.V XOR # Fh AND # 1081h * XOR
      'CRC.V' STO
    NEXT
  CRC.V
»
```

Voici une version plus rapide du programme précédent, entièrement écrite en assembleur:

```
CRCLM (# D298h)
D9D20 E4A20 A0000 00000 CB2A1 CCD20 CC000 8FB97
60147 13416 91741 43131 17414 7D517 43450 000E1
8A907 D014B 3103B 6A319 09EA9 03170 B6A14 67C50
34F00 000EF 3DB80 82160 C7A6C 5AFCB 80821 40C7A
6C5AF CB142 F4742 0DB14 41713 42000 06E8F 8F2D7
60142 16480 8CD7F E0EF2 DFFC0 EF20E FB01D BBF18
DBF1B 2130
```

CALC

CALC est en fait un ensemble de programmes permettant le calcul sur les grands entiers.

La HP48 peut déjà effectuer des calculs sur les entiers, mais dans le domaine de valeurs 0 à 18446744073709551615.

L'ensemble de programmes présenté ici permet quant à lui des calculs sur des nombres arbitrairement grands, la seule limitation étant liée à la mémoire disponible.

A titre d'exemple, ils ont permis le calcul de factorielle 2000 (plus de 5000 chiffres !) ou de la racine carrée de 2 à 500 décimales...

Les fonctions proposées travaillent sur des entiers positifs représentés sous forme de chaînes de caractères (par exemple: "1234567890" est l'entier 1234567890).

Les fonctions proposées sont les suivantes:

- ADD additionne deux entiers;
- SUBS soustrait deux entiers et renvoie la valeur absolue de la différence;
- MULT multiplie deux entiers;
- BFACT calcule la factorielle de l'entier donné en argument. Il procède par multiplications successives et affiche à l'écran le nombre courant et le nombre à atteindre de manière à ce que l'utilisateur ait une idée du travail effectué;
- POW calcule l'entier au niveau 2 à la puissance de l'entier au niveau 1 (comme ^). Comme pour BFACT une mesure du chemin parcouru est affichée (0 s'affiche lorsque tout est terminé);
- E multiplie l'entier au niveau 2 par la puissance de 10 donnée au niveau 1;
- DIV divise l'entier au niveau 2 par l'entier au niveau 1. La division réalisée est une division euclidienne;

- `MODU` renvoie la valeur du modulo de l'entier au niveau 2 par l'entier au niveau 1 (reste de la division euclidienne);
- `SQR` calcule une valeur approchée de la racine carrée de l'entier donné. Cette valeur est toujours inférieure à la valeur réelle.

Cet ensemble de programmes utilise des sous-routines dont certaines sont écrites en assembleur. Le listing commenté de ces routines en langage-machine est tout d'abord donné.

On trouve plus loin le rappel des codes à entrer sous la forme habituelle (voir la manière de rentrer un programme en langage machine).

Listing de DECODE.LM

Ce programme réalise la conversion d'un entier sous la forme spécifique utilisée par `ADD.LM`, `SUB.LM` et `MULT.LM` en un entier sous forme de chaîne de caractères.

```
          CCD20     CON(5)    PROL_CODE      Objet code
          B6000     CON(5)    (fin)-(début)
          8FB97/60  GOSBVL    SAVE_REG       Sauvegarde des
                                             registres
          143       A=DAT1    A
          132       AD0ex                    D0=adresse de
                                             l'objet au niveau
                                             1 de la pile
          164       D0=D0+    5
          3450000   LCHEX     #00005
          142       A=DAT0    A              longueur de
                                             cet objet
          EA        A=A-C     A
          D8        B=A       A
          164       D0=D0+    5
          174       D1=D1+    5
          143       A=DAT1    A
          133       AD1ex                    D1=adresse de
                                             l'objet au niveau
                                             2 de la pile
          174       D1=D1+    5
          147       C=DAT1    A
          133       AD1ex
          C2        C=C+A     A
          137       CD1ex
          3103      LCHEX     #30
11        8A9       ?B=0      A              Terminé ?
          61        GOYES     12             oui -> fin !
          1C1       D1=D1-    2
          15E0      C=DAT0    1              On lit un chiffre
          15D1      DAT1=C    2
          160       D0=D0+    1
          CD        B=B-1     A              Un chiffre de
                                             moins
          6AEF      GOTO      11
12        8F2D7/60  GOSBVL    LOAD_REG       récupération des
                                             registres
          142       A=DAT0    A              retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin
```

Listing d'ENCODE.LM

Ce programme réalise la fonction inverse de DECODE.LM et convertit l'entier sous forme de chaîne en un entier sous la forme utilisée par les trois programmes d'addition, de soustraction et de multiplication.

```
          CCD20     CON(5)    PROL_CODE      Objet code
début     '6000     CON(5)    (fin)-(début)  Longueur objet
          8FB9'60   GOSBVL    SAVE_REG       Sauvegardes..
          143       A=DAT1    A
          132       AD0ex                    D0=adresse de
                                             l'objet au niveau
                                             1 de la plie
          164       D0=D0+    5
          34500000  LCHEX     #00005
          142       A=DAT0    A              Longueur objet
          EA        A=A-C     A
          D8        B=A       A
          164       D0=D0+    5
          1'4       D1=D1+    5
          143       A=DAT1    A              D1=adresse de
                                             l'objet au niveau
                                             2 de la pile
          133       AD1ex
          1'4       D1=D1+    5
          14'       C=DAT1    A
          133       AD1ex
          C2        C=C+A     A
          13'       CD1ex
11        8A9       ?B=0      A              Est ce terimé ?
          61        GOYES     12             oui -> fin !
          1C1       D1=D1-    2
          15B0      A=DAT1    1              Lecture un chiffre
          1500      DAT0=A    P
          160       D0=D0+    1
          CD        B=B-1     A              Un chiffre de
                                             moins
          6AEF      GOTO      11             On itère..
12        8F2D'60   GOSBVL    LOAD_REG       Récupération des
                                             registres
          142       A=DAT0    A              Retour au RPL..
          164       D0=D0+    5
          808C      PC=(A)
fin
```

Listing de FORMAT.LM

Ce programme permet de donner à l'entier une forme classique en déterminant le nombre de zéros inutiles en début de chaîne (de manière à convertir "00123" en "123", par exemple).

```
          CCD20     CON(5)    PROL_CODE      Objet code
début     5E000     CON(5)    (fin)-(début)  Lobgueur objet
          8FB97/60  GOSBVL    SAVE_REG       Sauvegardes
          143       A=DAT1    A
          130       D0=A                     D0=adresse de
                                             l'objet au niveau
                                             1 de la pile
          169       D0=D0+    10
          17/4      D1=D1+    5
          143       A=DAT1    A
          131       D1=A                     D1=adresse de
                                             l'objet au niveau
                                             2 de la pile
          17/4      D1=D1+    5
          143       A=DAT1    A              longueur objet
          818F84    A=A-5     A
          17/2      D1=D1+    3
          D3        D=0       A              Nombre de zéros
                                             à ôter
11        17/1      D1=D1+    2
          E7/       D=D+1     A
          818F81    A=A-2     A
          8A8       ?A=0      A              terminé ?
          B0        GOYES     12             oui -> fin !
          157/0     C=DAT1    P
          90A       ?C=0      P              Est un zéro ?
          9E        GOYES     11             oui -> on itère
12        DB        C=D       A
          144       DAT0=C    A              On écrit le nombre
                                             de zéros à ôter
          8F2D7/60  GOSBVL    LOAD_REG       Récupérations
          142       A=DAT0    A              Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin
```

Listing de ZERO.LM

Ce programme met à zéro l'entier sous la forme utilisée par ADD.LM, SUB.LM et MULT.LM donné en argument.

```
          CCD20     CON(5)    PROL_CODE      Objet code
début     54000     CON(5)    (fin)-(début)  Longueur objet
          8FB97/60  GOSBVL    SAVE_REG       Sauvegardes..
          143       A=DAT1    A
          131       D1=A                     D1=adresse de
                                             l'objet au niveau
                                             1 de la pile
          174       D1=D1+    5
          143       A=DAT1    A
          174       D1=D1+    5
          C4        A=A+A     A
          F4        ASR       A              A=nombre de
                                             blocs de huit
                                             chiffres de cet
                                             objet
          AF2       C=0       W
l1        8A8       ?A=0      A              terminé ?
          F0        GOYES     l2             oui -> fin !
          15D7      DAT1=C    8              mise à zéro
          177       D1=D1+    8
          CC        A=A-1     A
          61FF      GOTO      l1             on itère
l2        8F2D7/60  GOSBVL    LOAD_REG       récupérations
          142       A=DAT0    A              retour au RPL.
          164       D0=D0+    5
          808C      PC=(A)
fin
```

Listing de ADD.LM

Ce programme réalise l'addition de deux entiers donnés en arguments. Il travaille par blocs de 8 chiffres.

```
          CCD20     CON(5)    PROL_CODE      Objet code
début     5?000     CON(5)    (fin)-(début)  Longueur code
          8FB9?60   GOSBVL    SAVE_REG       Sauvegardes
          143       A=DAT1    A
          130       D0=A
          169       D0=D0+    10             D0=adresse du
                                             contenu de l'objet
                                             au niveau 1 de la
                                             pile
          1?4       D1=D1+    5
          143       A=DAT1    A
          131       D1=A                     D1=adresse de l'
                                             objet au niveau 2
          1?4       D1=D1+    5
          14?       C=DAT1    A
          C6        C=C+C     A
          F6        CSR       A
          D?        D=C       A              D=nombre de blocs
                                             de cet objet
          1?4       D1=D1+    5
          AF0       A=0       W
          20        p=        0              retenue à zéro
11        8AB       ?D=0      A              Est ce terminé ?
          F2        GOYES     12             oui -> fin !
          AF2       C=0       W
          80F0      CPex      0              retenue
          05        SETDEC                   Passage en mode
                                             décimal
          15A?      A=DAT0    8              lecture premier
                                             bloc
          A?2       C=C+A     W              ajout à la
                                             retenue
          15B?      A=DAT1    8              lecture second
                                             bloc
          A?2       C=C+A     W              ajout
          04        SETHEX                   On repasse en
                                             hexadécimal
          15D?      DAT1=C    8              Ecriture du résult
          16?       D0=D0+    8              Blocs suivants
          1?7       D1=D1+    8
          CF        D=D-1     A              Un bloc de moins
          80D8      P=C       8              Retenue -> P
          61DF      GOTO      11             On itère
12        8F2D?60   GOSBVL    LOAD_REG       Récupérations
          142       A=DAT0    A              Retour au RPL..
          164       D0=D0+    5
          808C      PC=(A)
fin
```

Listing de SUB.LM

Ce programme réalise la soustraction de deux entiers donnés en arguments. Il travaille par blocs de 8 chiffres.

```
          CCD20      CON(5)    PROL_CODE      Objet code
début     6'000      CON(5)    (fin)-(début)  Longueur objet
          8FB9'60    GOSBVL    SAVE_REG       Sauvegardes
          143        A=DAT1    A
          130        D0=A                     D0=adresse de
                                              l'objet au niveau
                                              1 de la pile
          169        D0=D0+    10
          1'4        D1=D1+    5
          143        A=DAT1    A
          131        D1=A                     D1=adresse de
                                              l'objet au niveau
                                              2 de la pile
          1'4        D1=D1+    5
          14'        C=DAT1    A
          C6         C=C+C     A
          F6         CSR       A
          D'         D=C       A              D=nombre de blocs
                                              de 8 chiffres
          1'4        D1=D1+    5
          AF0        A=0       W              Pas de retenue..
11        8AB        ?D=0      A              terminé ?
          23         GOYES     12             oui -> fin !
          AF2        C=0       W
          15E'       C=DAT0    8              On lit 1 bloc
          05         SETDEC                   Passage en mode
                                              décimal
          A'A        A=A+C     W              Ajout à la
                                              retenue
          15F'       C=DAT1    8              Bloc à soustraire
          B'2        C=C-A     W              Soustraction
          04         SETHEX                   Retour en hexa
          15D'       DAT1=C    8              Ecriture resultat
          1'7        D1=D1+    8
          16'        D0=D0+    8
          CF         D=D-1     A              1 bloc de moins..
          AF0        A=0       W
          94A        ?C=0      S              Retenue ?
          4D         GOYES     11             Non -> on itere
          B64        A=A+1     B              On enregistre la
                                              retenue
          6ECF       GOTO      11             On itère..
12        8F2D'60    GOSBVL    LOAD_REG       Récupérations
          142        A=DAT0    A              Retour au RPL
          164        D0=D0+    5
          808C       PC=(A)
fin
```

Listing de MULT.LM

Ce programme réalise la multiplication de deux entiers. Il réalise cette opération comme on le fait "à la main" en travaillant successivement sur chaque chiffre de l'entier au niveau 1 de la pile...

```
         CCD20     CON(5)    PROL_CODE      Objet code
début    C1100     CON(5)    (fin)-(début)  Longueur code
         8FB97/60  GOSBVL    SAVE_REG       Sauvegardes
         143       A=DAT1    A
         818F09    A=A+10    A
         101       R1=A                     R1=adresse du
                                            contenu de l'objet
                                            au niveau 1 de la
                                            pile (resultat)
         17/4      D1=D1+    5
         143       A=DAT1    A
         133       ADlex                    D1=adresse de
                                            l'objet au niveau
                                            2 de la pile
         17/4      D1=D1+    5
         AF2       C=0       W
         147       C=DAT1    A
         818FA4    C=C-5     A
         BF2       CSL       W
         BF2       CSL       W
         BF2       CSL       W
         AD7       D=C       M              Nombre de blocs de
                                            l'entier au niveau
                                            2 de la pile
         17/4      D1=D1+    5
         133       ADlex
         103       R3=A                     R3=adresse du
                                            contenu de l'objet
                                            au niveau 2
         17/4      D1=D1+    5
         143       A=DAT1    A
         131       D1=A                     D1=adresse objet 3
         17/4      D1=D1+    5
         AF2       C=0       W
         147       C=DAT1    A
         818FA4    C=C-5     A
         BF2       CSL       W
         BF2       CSL       W
         BF2       CSL       W
         17/4      D1=D1+    5
         133       ADlex
         102       R2=A                     R2=adresse du
                                            contenu objet 3
11       95F       ?D#0      M              Encore du travail?
         60        GOYES     12             Oui -> on continue
         6690      GOTO      17             Non -> on arrete
```

```
12      113     A=R3
        131     D1=A
        AE2     C=0     B
        15F0    C=DAT1  1       On lit un chiffre
        AE7     D=C     B
        170     D1=D1+  1
        133     ADlex
        103     R3=A
        A5F     D=D-1   M       un ch. de moins
        112     A=R2
        131     D1=A
        ADD     CBex    M
        AD9     C=B     M
        111     A=R1
        130     D0=A
        E4      A=A+1   A
        101     R1=A
        96B     ?D=0    B       Mult. par  zéro ?
        1C      GOYES   11      Oui -> fini !
        AE2     C=0     B
13      95D     ?B#0    M       Encore ?
        A0      GOYES   14      Oui..
        15C0    DAT0=C  1       Non: on écrit
                                la retenue finale
        62BF    GOTO    11      Et on itère..
14      06      RSTK=C          Sauvegade C
        15B0    A=DAT1  1       On lit un chiffre
        AEB     C=D     B
        05      SETDEC          Passage en décimal
        AE1     B=0     B
15      822     SB=0            ;
        81900   ASRB    P       ;
        832     ?SB=0           ;
        50      GOYES   16      ; Multiplication
        A61     B=B+C   B       ;
16      A66     C=C+C   B       ;
        90C     ?A#0    P       ;
        AE      GOYES   15      ;
        07      C=RSTK
        A69     C=C+B   B       Ajout retenue
        AE0     A=0     B
        15A0    A=DAT0  1
        A62     C=C+A   B       Ajout à l'existant
        15C0    DAT0=C  1       Ecriture resultat
        04      SETHEX          Retour en hexa
        160     D0=D0+  1
        170     D1=D1+  1
        A5D     B=B-1   M
        BE6     CSR     B       Mise a jour ret.
        6BAF    GOTO    13      On itère
17      8F2D760 GOSBVL  LOAD_REG Récupérations
        142     A=DAT0  A       Retour au RPL
        164     D0=D0+  5
        808C    PC=(A)
fin
```

Listing de DIV.LM

Ce programme réalise la division de deux entiers. Il renvoie le reste et le quotient de la division euclidienne.

```
          CCD20     CON(5)    PROL_CODE      Objet code
début     '6100     CON(5)    (fin)-(début)  Longueur code
          8FB9'60   GOSBVL    SAVE_REG       Sauvegardes
          143       A=DAT1    A
          130       D0=A                     D0=adresse de
                                             l'objet au niveau
                                             1 de la pile
          164       D0=D0+    5
          142       A=DAT0    A
          818F84    A=A-5     A
          819F0     ASRB      A
          103       R3=A                     R3=nbr de chiffres
          164       D0=D0+    5
          132       AD0ex
          102       R2=A                     R2=adresse du
                                             contenu de l'objet
                                             au niveau 1
          1'74      D1=D1+    5              Objet suivant
          143       A=DAT1    A              A=adresse de
                                             l'objet au niveau
                                             2 de la pile
          818F0'    A=A+8     A
          D8        B=A       A
          1'74      D1=D1+    5              Objet suivant
          143       A=DAT1    A              A=adresse de
                                             l'objet au niveau
                                             3 de la pile
          130       D0=A
          16'       D0=D0+    8
          1'74      D1=D1+    5              Objet suivant
          143       A=DAT1    A              A=adresse de
                                             l'objet au niveau
                                             4 de la pile
          818F04    A=A+5     A
          131       D1=A
          14'       C=DAT1    A              Longueur obj 4
          CA        A=A+C     A
          818F81    A=A-2     A
          100       R0=A
          818FA4    C=C-5     A
          819F2     CSRB      A              Nbr de chiffres
                                             de l'objet 4
          109       R1=C
          AC3       D=0       5
l1        113       A=R3
          8AC       ?A#0      A              Encore ?
          60        GOYES     l2
          61B0      GOTO      l9             Non -> fin !
```

```
12      CC      A=A-1   A       Un chiffre de
                                moins..
        103     R3=A
        AC2     C=0     S
13      111     A=R1            ;
        DE      CAex    A       ;
        D7      D=C     A       ;
        DE      CAex    A       ;
        136     CD0ex           ;
        C2      C=C+A   A       ;
        C2      C=C+A   A       ;
        C8      B=B+A   A       ;Initialisations
        C8      B=B+A   A       ;
        DD      CBex    A       ;
        136     CD0ex           ;
        B47     D=D+1   S       ;
        110     A=R0            ;
        131     D1=A            ;
        AE2     C=0     B       Pas de retenue
14      8AF     ?D#0    A       Encore ?
        60      GOYES   15
        6740    GOTO    17      Non -> suite
15      CF      D=D-1   A
        05      SETDEC          Passage en décimal
        AE0     A=0     B
        15B0    A=DAT1  1       Lecture 1 chiffre
        A62     C=C+A   B       Ajout retenue
        15A0    A=DAT0  1
        EE      C=A-C   A       soustraction
        04      SETHEX
        DC      ABex    A
        132     AD0ex
        15C0    DAT0=C  1       Réécriture
        181     D0=D0-  2
        132     AD0ex
        DC      ABex    A
        181     D0=D0-  2       Un chiffre de
                                moins
        1C1     D1=D1-  2
        A82     C=0     P
        96A     ?C=0    B       Retenue ?
        A0      GOYES   16
        3110    LCHEX   01      Oui -> retenue !
        6DBF    GOTO    14
16      AE2     C=0     B       Pas de retenue !
        66BF    GOTO    14
17      96E     ?C#0    B       Retenue en fin ?
        90      GOYES   18      Oui -> arret
        B46     C=C+1   S       Incrément quotient
        658F    GOTO    13      On itère
18      161     D0=D0+  2
        136     CD0ex
        DD      CBex    A
        136     CD0ex
        161     D0=D0+  2
```

```
         B47       D=D+1     S
         112       A=R2
         131       D1=A
         1554      DAT1=C    S              Ecriture quotient
         171       D1=D1+    2
         133       AD1ex
         102       R2=A
         694F      GOTO      11             On itère..
19       8F2D760   GOSBVL    LOAD_REG       Récupérations
         822       SB=0
         81943     DSRB      S
         832       ?SB=0                    Faut il modifier
                                            l'ordre des objets
                                            dans la pile ?
         A1        GOYES     110            Non -> fin
         174       D1=D1+    S              ;
         143       A=DAT1    A              ;
         174       D1=D1+    S              ;
         147       C=DAT1    A              ;Echange objets
         141       DAT1=A    A              ;aux niveaux 2 et
         1C4       D1=D1-    S              ;3 de la pile
         145       DAT1=C    A              ;
         1C4       D1=D1-    S              ;
110      142       A=DAT0    A              Retour au RPL
         164       D0=D0+    S
         808C      PC=(A)
fin
```

DECODE.LM (# D620h)

```
CCD20 B6000 8FB97 60143 13216 43450 00014 2EAD8
16417 41431 33174 14713 3C213 73103 8A961 1C115
E015D 1160C D6AEF 8F2D7 60142 16480 8C
```

ENCODE.LM (# B0A9h)

```
CCD20 76000 8FB97 60143 13216 43450 00014 2EAD8
16417 41431 33174 14713 3C213 78A96 11C11 5B015
00160 CD6AE F8F2D 76014 21648 08C
```

FORMAT.LM (# E27Dh)

```
CCD20 5E000 8FB97 60143 13016 91741 43131 17414
3818F 84172 D3171 E7818 F818A 8B015 7090A 9EDB1
448F2 D7601 42164 808C
```

ZERO.LM (# 69AAh)

```
CCD20 54000 8FB97 60143 13117 41431 74C4F 4AF28
A8F01 5D717 7CC61 FF8F2 D7601 42164 808C
```

ADD.LM (# E74Ch)

```
CCD20 57000 8FB97 60143 13016 91741 43131 17414
7C6F6 D7174 AF020 8ABF2 AF280 F0051 5A7A7 215B7
A7204 15D71 67177 CF80D 861DF 8F2D7 60142 16480
8C
```

SUB.LM (# C14h)

```
CCD20 67000 8FB97 60143 13016 91741 43131 17414
7C6F6 D7174 AF08A B23AF 215E7 05A7A 15F7B 72041
5D717 7167C FAF09 4A4DB 646EC F8F2D 76014 21648
08C
```

MULT.LM (# ACDBh)

```
CCD20 C1100 8FB97 60143 818F0 91011 74143 13317
4AF21 47818 FA4BF 2BF2B F2AD7 17413 31031 74143
13117 4AF21 47818 FA4BF 2BF2B F2174 13310 295F6
06690 11313 1AE21 5F0AE 71701 33103 A5F11 2131A
DDAD9 11113 0E410 196B1 CAE29 5DA01 5C062 BF061
5B0AE B05AE 18228 19008 3250A 61A66 90CAE 07A69
AE015 A0A62 15C00 41601 70A5D BE66B AF8F2 D7601
42164 808C
```

DIV.LM (# AD61h)

```
CCD20 76100 8FB97 60143 13016 41428 18F84 819F0
10316 41321 02174 14381 8F07D 81741 43130 16717
41438 18F04 13114 7CA81 8F811 00818 FA481 9F210
9AC31 138AC 6061B 0CC10 3AC21 11DED 7DE13 6C2C2
C8C8D D136B 47110 131AE 28AF6 06740 CF05A E015B
0A621 5A0EE 04DC1 3215C 01811 32DC1 811C1 A8296
AA031 106DB FAE26 6BF96 E90B4 6658F 16113 6DD13
6161B 47112 13115 54171 13310 2694F 8F2D7 60822
81943 832A1 17414 31741 47141 1C414 51C41 42164
808C
```

DIV.C (# B5C2h)

```
«
  FORMAT "0" SWAP + SWAP FORMAT "0" SWAP +
  IF
    OVER "00" ==
  THEN
    DROP2 # 305h DOERR
  ELSE
    DUP NEWOB DUP 1 OVER SIZE 6 PICK SIZE - 1 +
    SUB DIV.LM SWAP ROT DROP DUP SIZE DUP 5 ROLL
    SIZE - 1 + SWAP SUB
  END
»
```

```
MULT.C (# 7E7Ch)
«
  DUP2 + ZERO.LM MULT.LM 3 ROLLD DROP2
»

PREPARE (# 18D6h)
«
  FORMAT SWAP FORMAT → N1 N2
  «
    IF
      N1 SIZE N2 SIZE DUP2 >
    THEN
      DROP2 N2 N1
    ELSE
      IF
        <
      THEN
        N1 N2
      ELSE
        N1 N2
        IF
          DUP2 >
        THEN
          SWAP
        END
      END
    END
    ENCODE SWAP ENCODE DUP2 SIZE SWAP SIZE SWAP
    - 0 CHR
    WHILE
      DUP2 SIZE >
    REPEAT
      DUP +
    END
    1 ROT SUB +
  »
»
```

```
DECODE (# A04Dh)
«
  DUP DUP + SWAP DECODE.LM DROP FORMAT
»

ENCODE (# 19ADh)
«
  "0000000" SWAP + DUP SIZE 8 MOD 1 + OVER SIZE SUB
  DUP 1 OVER SIZE 2 / SUB ENCODE.LM SWAP DROP
»

FORMAT (# E1B2h)
«
  "0" SWAP + # FFFFFh NEWOB FORMAT.LM B→R OVER SIZE
  SUB
»

MODU (# FB90h)
«
  IF
    FORMAT DUP "0" ==
  THEN
    DROP
  ELSE
    DIV.C SWAP DROP FORMAT
  END
»

DIV (# 600Ah)
«
  DIV.C DROP FORMAT
»
```

```
E (# 5A9Eh)
«
  →STR STR→ DUP
  → N
  «
    "0"
    WHILE
      N DUP 2 / IP 'N' STO
    REPEAT
      DUP +
    END
    1 ROT SUB +
  »
»

POW (# D4DBh)
«
  →STR STR→
  → N
  «
    ENCODE 1 ENCODE
    WHILE
      N DUP 1 DISP 0 ≠
    REPEAT
      IF
        N 2 / DUP IP 'N' STO FP
      THEN
        OVER MULT.C
      END
      SWAP DUP MULT.C SWAP
    END
    SWAP DROP DECODE
  »
»
```

```
SQR (# C265h)
«
  "00" + FORMAT DUP 1 OVER SIZE 2 / SUB
  → A X
  «
    DO
      X A OVER DIV ADD 2 DIV
    UNTIL
      X OVER 'X' STO ==
    END
    X 1 OVER SIZE 1 - SUB
  »
»

BFACT (# 23E5h)
«
  →STR STR→ DUP 2 DISP 1 ENCODE 1 ROT
  FOR X
    X DUP 1 DISP ENCODE MULT.C
  NEXT
  DECODE
»

MULT (# EC5Fh)
«
  ENCODE SWAP ENCODE MULT.C DECODE
»

SUBS (# 204Fh)
«
  PREPARE SUB.LM DROP DECODE
»

ADD (# 701Ch)
«
  PREPARE 0 CHR DUP + DUP + ROT OVER + 3 ROLLD +
  ADD.LM DROP DECODE
»
```

Factorielle 2000

Le résultat suivant a été obtenu grâce à l'ensemble de programmes 'CALC' précédemment listé.

PI

Le calcul des décimales de π est un problème qui a toujours passionné les mathématiciens. Aujourd'hui, avec les progrès croissants de l'informatique, il est possible de calculer des millions de décimales de ce chiffre mythique...

Grâce à l'ensemble de programmes CALC, nous allons pouvoir nous lancer, nous aussi, dans un tel calcul. Cependant, du fait de la place-mémoire limitée et des capacités de calculs de la machine, nous ne pourrons dépasser quelques milliers de décimales...

Le principe de calcul est simple. Il existe une formule bien connue:

$$\frac{\pi}{4} = \mathrm{Atan}\left(\frac{1}{2}\right) + \mathrm{Atan}\left(\frac{1}{5}\right) + \mathrm{Atan}\left(\frac{1}{8}\right)$$

De plus, nous savons que Atan peut se calculer par:

$$\mathrm{Atan}(x) = \sum_{n=0}^{\infty} (-1)^n . \frac{x^{2n+1}}{2n+1}$$

qui converge d'autant plus vite que x est petit.

Nous aurons donc:

$$\pi = 4 . \left(\sum_{n=0}^{\infty} (-1)^n . \frac{\left(\frac{1}{2}\right)^{2n+1} + \left(\frac{1}{5}\right)^{2n+1} + \left(\frac{1}{8}\right)^{2n+1}}{2n+1} \right)$$

Comme le calculateur CALC ne gère que des nombres entiers positifs, il nous faudra tout multiplier par une puissance de 10 (de manière à rendre les nombres entiers) et à gérer le signe du terme général de la série "à la main".

Le programme PI réalise ce calcul. Il prend en entrée un réel représentant le nombre de chiffres significatifs désirés.

Il affiche en permanence le numéro de l'étape courante (2n+1) et le nombre de décimales restant à calculer.

Il faut compter environ une dizaine de secondes par décimale (mais cette mesure dépend de nombreux facteurs: mémoire libre disponible, nombre de décimales demandées...). Voici quelques décimales calculées à l'aide de 'PI':

```
3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117867982
14808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881
09756659334461284756482337867831652712019091456485669234603486104543266482133936072602491412737245870066
06315588174881520920962829254091715364367892590360011330530548820466521384146951941511609433057270365759
59195309218611738193261179310511854807446237996274956735188575272489122793818301194912983367336244865664
38860213949463952247371907021798609437027705392171762931767523846748184676694051320005681271452635608277
85771342757789609173637178721468440901224953430146549585371050792279689258923542019956112129021960864034
41815981362977477130996051870721134999999837297804995105973173281609631859502445945534690830264252230825
33446850352619311881710100031378387528865875332083814206171776691473035982534904287554687311595628638823
```

```
P1 (# CF6Dh)
«
  → P
  « 1 P 1 + E DUP 2 DIV OVER 5 DIV ROT 8 DIV 1 0 0
    → A B C N T S
    «
      A B ADD C ADD
      DO
        T SWAP
        IF
          S
        THEN
          SUBS
        ELSE
          ADD
        END
        'T' STO 1 S - 'S' STO A 4 DIV DUP 'A' STO B
        25 DIV DUP 'B' STO C 64 DIV DUP 'C' STO ADD
        ADD N 2 ADD DUP 'N' STO DUP 1 DISP DIV DUP
        SIZE 2 DISP
      UNTIL
        DUP "0" ==
      END
      DROP T 4 MULT 2 P SUB "3." SWAP +
    »
  »
»
```

VAL

Ce programme calcule la valeur d'un polynôme donné sous forme de vecteur en un point donné.

Il prend en arguments le vecteur décrivant le polynôme et le point considéré et renvoie la valeur du polynôme en ce point.

Remarque: le second argument peut être un réel ou un complexe, ainsi que tout objet algébrique ou nom...

Exemple: calculer la valeur de x^2+2x+1 en 2:

```
[ 1 2 1 ] 2 VAL
```

```
VAL (# 2681h)
«
  → V X
  «
    V SIZE LIST→ DROP → A
    «
      0 1 A
      FOR Y
        V Y GET X A Y - ^ * +
      NEXT
    »
  »
»
```

A->V et V->A

Ces deux programmes permettent respectivement la conversion d'un polynôme sous forme algébrique ('3*X^2+2X+1' par exemple) en sa forme-vecteur ([3 2 1]) ainsi que l'opération réciproque.

C'est en effet avec la seconde forme que travaille la plupart des programmes présentés ici...

Remarque: le programme V→A utilise le programme VAL précédemment listé...

```
A→V (# 60Dh)
«
  ( ) 0 'I' STO
  DO
    0 'X' STO OVER EVAL I FACT / 1 →LIST SWAP + SWAP
    'X' DUP PURGE ∂ 1 'I' STO+ SWAP
  UNTIL
    OVER 0 SAME
  END
  SWAP DROP 'I' PURGE LIST→ 1 →LIST →ARRY
»


V→A (# 4E46h)
«
  'X' VAL
»
```

DER

Ce programme effectue la dérivation d'un polynôme écrit sous forme de vecteur.

Exemple: pour dériver le polynôme $3X^2+2X+1$, taper:

```
[ 3 2 1 ] DER
```

```
DER (# 9063h)
«
  ARRY→ LIST→ - → A
  « DROP
    IF
      A 0 ==
    THEN
      [ 0 ]
    ELSE
      1 A
      FOR X
        X * A ROLLD
      NEXT
      A 1 →LIST →ARRY
    END
  »
»
```

DIVP

Ce programme réalise la division euclidienne de deux polynômes écrits sous forme de vecteurs.

Par exemple, pour diviser le polynôme X^2+2X+1 par le polynôme X+1, il suffit de taper:

```
[ 1 2 1 ] [ 1 1 ] DIVP
```

Il renvoie le quotient au niveau 2 de la pile, et le reste au niveau 1.

```
DIVP (# 28E3h)
«
   DUP2 → A B
  «
    B 1 GET A SIZE 1 GET B SIZE 1 GET DUP2 -
    → c n p q
    «
      IF
        p 1 ==
      THEN
        DROP2 A c / [ 0 ]
      ELSE
        0 q
        FOR x
          OVER 1 GET c / DUP 4 ROLLD * n x - 1 →LIST
          RDM - ARRY→ 1 GET 1 - →ARRY SWAP DROP B
        NEXT
        DROP q 2 + ROLLD q 1 + →ARRY SWAP
      END
    »
  »
»
```

PCAR

PCAR permet de calculer le polynôme caractéristique d'une matrice carrée quelconque. Le résultat renvoyé est un vecteur contenant les différents coefficients du polynôme.

Ce vecteur peut ensuite être utilisé avec le programme LAGU afin d'en trouver toutes les racines, ce qui permet donc de calculer facilement toutes les valeurs propres de la matrice...

Exemple:

`3 IDN PCAR` renvoie `[ 1 -3 3 -1 ]` (x^3-3x^2+3x-1)

```
PCAR (# DB94h)
«
  DUP IDN DUP SIZE LIST→ DROP2 → M I N
  «
    0 N
    FOR X
      M I X * - DET
    NEXT
    N 1 + 1 →LIST →ARRY N 1 + IDN 0 N
    FOR X
      0 N
      FOR Y
        X 1 + N Y - 1 + 2 →LIST X Y ^ PUT
      NEXT
    NEXT ✓
  »
»
```

LAGU

Ce programme trouve toutes les racines réelles et complexes d'un polynôme quelconque (à coefficients réels ou complexes).

Pour l'utiliser, il suffit de placer dans la pile le polynôme sous forme de vecteur en puissances décroissantes des x^i: [a_n ... a_0], le coefficient a_i correspond au coefficient devant le terme x^i, et de lancer `LAGU`.

Le programme vous informera des différentes étapes du calcul et renverra dans la pile la liste des racines du polynôme.

L'algorithme employé est celui de Laguerre:

On fixe Z_0 quelconque (à défaut d'une approximation de la racine, on peut mettre 0 ou la valeur de la racine précédemment trouvée, ce qui provoque un gain de temps important dans le cas d'une racine multiple) et on calcule $Z_{k+1} = Z_k + S_k$ où S_k est l'étape de Laguerre valant:

$$Sk = \frac{-n.P(Z_k)}{P'(Z_k) + E.\sqrt{((n-1).P'(Z_k))^2 - n.(n-1).P(Z_k).P''(Z_k)}}$$

Où

- n étant le degré du polynome;
- P le polynome;
- P' sa dérivée première;
- P'' sa dérivée seconde.

E étant à choisir égal à +1 ou -1 pour rendre le dénominateur le plus grand possible, de manière à ce que l'étape de Laguerre soit la plus petite possible.

Attention: Si le polynôme possède des racines de grande multiplicité, le procédé va osciller sans jamais converger.

Les approximations sont très bonnes pour un polynôme de degré inférieur à 7 et de multiplicité maximale 4.

LAGU utilise les programmes VAL DER et DIVP listés précédemment...

Exemple: pour trouver les racines du polynôme suivant:

$$X^6-14.X^4+49.X^2-36$$

Il faut:

- Placer le polynôme vecteur correspondant au niveau 1 de la pile:

```
[ 1 0 -14 0 49 0 -36 ]
```

- Lancer LAGU.

Quelques instants plus tard, on obtient la liste des six racines du polynôme précédent:

```
{ 1 2 3 -1 -2 -3 }
```

```
LAGU (# 15BCh)
«
  IF
    DUP SIZE { 1 } ==
  THEN
    DROP { }
  ELSE
    CLLCD { } 'SOL' STO 0 'Z' STO
    DO
      DUP DUP2 'P' STO 1 DISP 'Z' VAL SWAP DER DUP
      'Z' VAL SWAP DER 'Z' VAL P SIZE LIST→ - DUP 1
      - DUP SQ 3 PICK 3 PICK * NEG
      → P0 P1 P2 N M A B
      «
        "Racine No " N + 2 DISP Z
        WHILE
          DUP 'Z' STO 3 DISP P0 EVAL DUP ABS
          .0000000001 >
```

```
        REPEAT
          P1 EVAL P2 EVAL
          → R S T
          «
            S A S SQ * B R T * * + ∫ DUP2 DUP2 + ABS
            3 ROLLD - ABS ≥ 2 * 1 - * + DUP
            IF
              ABS 0 ==
            THEN
              50 .1 BEEP DROP RAND 40 * 20 - RAND 40
              * 20 - R→C "⁄0 Nouveau Z0" 2 DISP
            ELSE
              N NEG R * SWAP ⁄ Z +
            END
          »
        END
        DROP
      »
      SOL Z + 'SOL' STO P 1 Z NEG ( 2 ) →ARRY DIVP
      DROP
    UNTIL
      DUP SIZE LIST→ ≤
    END
    DROP ( Z P ) PURGE SOL
  END
»
```

PMAT

Ce programme calcule l'image d'une matrice carrée quelconque par un polynôme.

Il prend en arguments la matrice et le polynôme (sous forme vectorielle).

Exemple: pour calculer l'image de la matrice identité d'ordre 3 par le polynôme $3X^2+2X+1$, il suffit de taper:

```
3 IDN [ 3 2 1 ] PMAT
```

```
PMAT (# 844Ch)
«
  SWAP OVER SIZE 1 GET → V X L
  «
    X 0 CON X IDN L 1
    FOR Y
      DUP V Y 1 →LIST GET * ROT + SWAP X * -1
    STEP DROP
  »
»
```

µSOLVER

µSOLVER permet la résolution d'un système de plusieurs équations non linéaires à plusieurs inconnues... Pour résoudre un tel système, il utilise l'algorithme de Newton-Raphson.

Mode d'emploi du programme: après avoir entré les deux objets ci-dessous, µSOLVER s'utilise comme suit:

- Placer la liste les différentes équations à résoudre dans une liste, la stocker dans `'MEQ'`. Exemple:

  ```
  ( 'SQ(X)+SQ(Y)=1' 'SQ(X-1)+SQ(Y)=1' )
  ```

  (intersection de deux cercles).

- Placer les noms des inconnues dans une liste et la stocker dans `'MVAR'`. Pour l'exemple ci-dessus:

  ```
  ( X Y )
  ```

- Stocker des valeurs approchées dans les variables. Cette étape est optionnelle mais permet d'accélérer la recherche. Par exemple 1 dans `'X'` et dans `'Y'`;

- Placer la précision recherchée dans la pile (qui correspond à un majorant de la norme euclidienne des erreurs). Par exemple `0.00001`;

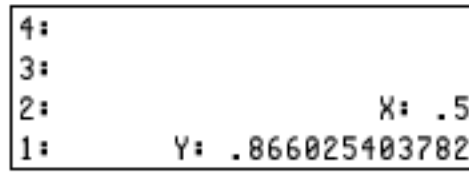
- Lancer µSOLVER.

Durant toute la recherche, il affichera l'erreur de calcul courante.

En cas d'erreur (division par zéro...), la reprise sera automatique.

En fin de recherche, il place les différentes valeurs approchées dans la pile, "tagguées" par le nom de la variable correspondante.

Dans le cas de notre exemple, nous obtiendrions:



De plus, µSOLVER possède deux particularités:

- Il permet la recherche de racines complexes. Pour effectuer une telle recherche, il convient de prendre des nombres complexes comme valeurs approchées de départ;
- Il comporte de nombreux IFERR... END. De ce fait, il peut être difficile d'interrompre le programme en appuyant sur [ON]. Pour l'arrêter, il convient d'appuyer deux fois très rapidement sur cette touche (ce qui peut conduire à des affichages incohérents).

Il est composé de deux programmes: µSOLVER, programme principal, et JACOB qui calcule le jacobien du système.

µSOLVER a été écrit par Christophe DUPONT DE DINECHIN.

```
JACOB (# B41h)
«
  ( ) 1 4 PICK SIZE
  FOR e
    1 3 PICK SIZE
    FOR υ
      3 PICK e GET 3 PICK υ GET ∂ +
    NEXT
  NEXT
  ROT ROT DROP2
»
```

```
µSOLVER (# 86BAh)
«
  MEQ MVAR DUP LIST→ DUP 1 + 1 ROT
  START
    DUP ROLL EVAL DUP TYPE 1 > 0 ROT IFTE SWAP
  NEXT
  1 - →ARRY 3 ROLLD DUP2 DUP PURGE JACOB SWAP
  4 ROLL
  DO
    OVER LIST→ DUP 2 + ROLL SWAP 1
    FOR i
      DUP i GET ROT STO -1
    STEP
    4 PICK LIST→ DUP 1 + 1 ROT
    START
      DUP ROLL
      IFERR
        →NUM
      THEN
        "Erreur fonction:↵" ERRM + 1 DISP 0
      END
      SWAP
    NEXT
    1 - →ARRY SWAP DUP2 SIZE OVER SIZE SWAP + 6
    PICK LIST→ 2 + DUP ROLL SWAP 3 OVER
    START
      ROT
      IFERR
        →NUM
      THEN
        "Erreur Jacobien:↵" ERRM + 1 DISP 0
      END
      OVER ROLLD
    NEXT
    DROP →ARRY
    IFERR
      ⁄
    THEN
      "Systeme singulier:↵" ERRM + 1 DISP DROP
```

```
      RAND *
    END
    - SWAP ABS "Erreur actuelle:␣" OVER + 1 DISP
  UNTIL
    6 PICK ≤
  END
  5 ROLLD 5 ROLLD DROP DROP2 1 3 PICK SIZE
  FOR x
    DUP2 x GET SWAP x GET DUP2 STO →TAG 3 ROLLD
  NEXT
  DROP2
»
```

LABY

Dans le jeu LABY, vous êtes perdu au milieu d'un labyrinthe... et vous devez en sortir le plus rapidement possible.

Pour jouer, il faut bien évidement commencer par entrer tous les programmes qui suivent. Ensuite placez-vous dans le menu CST (appuyer sur la touche portant ce nom, située à gauche de la touche VAR).

Les 6 touches du menu sont alors actives. Elles ont les rôles suivants:

- `INIT` permet de lancer le jeu. Cette touche déclenche un choix d'un labyrinthe, y place le joueur et affiche la vue correspondante; Le "x" représente votre position;
- `VUE` permet de réafficher la vue courante;
- Les quatre flèches permettent de se déplacer dans le labyrinthe...

Dans le listing qui suit, un seul labyrinthe est présent. Il est possible d'en ajouter autant que désiré.

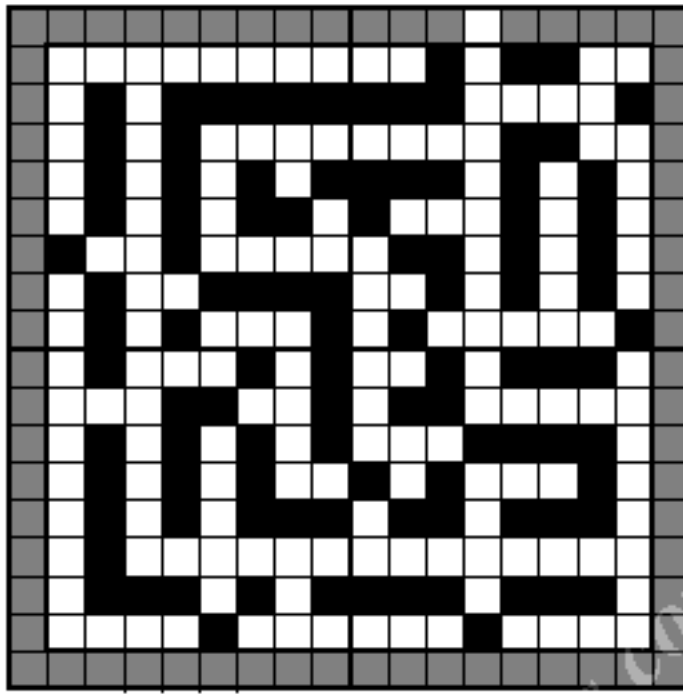
En effet, les différents labyrinthes sont contenus dans la liste `'LABS'`. Cette liste contient elle-même des listes (une par labyrinthe) constituées comme suit:

- Un complexe représentant les coordonnées de la sortie;
- Une liste de quatre entiers représentant le plan du labyrinthe.

Le codage d'un plan se fait de la manière suivante: chaque labyrinthe est construit sur un damier de 16 cases sur 16 cases. Chacune de ces cases peut être soit un passage (0) soit un mur (1).

Ce plan est ensuite converti en 4 entiers (4 fois 64 bits) représentant chacun un quart du labyrinthe.

Un exemple d'un tel codage est donné page suivante...



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
15	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	15
14	0	1	0	1	1	1	1	1	1	1	1	0	0	0	0	1	14
13	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	0	13
12	0	1	0	1	0	1	0	1	1	1	1	0	1	0	1	0	12
11	0	1	0	1	0	1	1	0	1	0	0	0	1	0	1	0	11
10	1	0	0	1	0	0	0	0	0	1	1	0	1	0	1	0	10
9	0	1	0	0	1	1	1	1	0	0	1	0	1	0	1	0	9
8	0	1	0	1	0	0	0	1	0	1	0	0	0	0	0	1	8
7	0	1	0	0	0	1	0	1	0	0	1	0	1	1	1	0	7
6	0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	0	6
5	0	1	0	1	0	1	0	1	0	1	1	1	1	0	0	0	5
4	0	1	0	1	0	1	0	0	0	1	0	0	0	1	0	1	4
3	0	1	0	1	0	1	1	1	0	1	1	1	0	1	1	0	3
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
1	0	1	1	1	0	1	0	1	1	1	1	0	1	1	1	0	1
0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

Le schéma du haut représente le plan du labyrinthe, une case blanche représentant un passage, noire, un mur.

Les cases grisées représentent les "murs implicites", c'est à dire correspondant à des cases "hors-labyrinthe".

Seule une de ces cases reste blanche, c'est la sortie, située aux coordonnées 11 16.

Le tableau du bas représente le codage logique du plan, les murs étant codés en "1" et les passages en "0".

Ce tableau se transforme en quatre entiers binaires dans l'ordre suivant:

2	4
1	3

Ce qui donne:

```
#1010001010011000101010100010101011101010000000101010111000010000b
#0000000011111010000010101010101001101010000010011111001010001010b
#0111010000000110011110000100010101110110000000000011101110000100b
#0011010010000111001100000101011101010001010101100101010010000010b
```

qui, traduits en hexadécimal, donnent la liste suivante:

```
( # A298AA2AEA02AE10h # FA0AAA6A09F28Ah
  # 7406784576007708h # 3487305751565482h )
```

D'où le codage du premier élément de la variable 'LABS' (voir plus bas).

Voici le listing des différents objets à entrer:

```
AL (# 5B7Ah)
«
  0 RDZ 16 RAND * IP
»

BL1 (# 4998h)
"▒" (127 CHR)
```

```
BL2 (# 3D27h)
" " (32 CHR)

TS (# 3E54h)
«
  R→C SO ≠
»

TV (# 5A0Dh)
«
  TVP SWAP TVP +
»

TVP (# 115Fh)
«
  DUP 0 < SWAP 15 > +
»

ETAT (# 85Ah)
«
  DUP2
  IF
    TV
  THEN
    TS
  ELSE
    DUP2 8 / IP SWAP 8 / IP 2 * + 1 + LAB SWAP GET 3
    ROLLD 8 MOD SWAP 8 MOD SWAP 16 SWAP ^ DUP # 1h *
    * SWAP 2 SWAP ^ * AND # 0h >
  END
»

I4 (# AC1h)
«
  X 1 - Y
»
```

```
I3 (# 8E47h)
«
  X 1 + Y
»

I2 (# D48h)
«
  X Y 1 -
»

I1 (# E9E3h)
«
  X Y 1 +
»

TEST (# 29CDh)
«
  1 'COUP' STO+ DUP2
  IF
    TS
  THEN
    DUP2
    IF
      ETAT
    THEN
      "MUR" 1 DISP DROP2
    ELSE
      'Y' STO 'X' STO VUE
    END
  ELSE
    "BRAVO" 1 DISP DROP2 1400 .1 BEEP
  END
  3 FREEZE
»
```

```
CH (# C52Dh)
«
  ETAT 95 * 32 + CHR
»

LABS (# 38FBh)
{
  {
    (11,16)
    { # A298AA2AEA02AE10h # FA0AAA6A09F28Ah
      # 7406784576007708h # 3487305751565482h }
  }
}

CST (# 9F27h)
{
 INIT VUE
 { "←" GA } { "↑" AV } { "↓" AR } { "→" DR }
}

AR (# D13Dh)
«
  I2 TEST
»

AV (# A255h)
«
  I1 TEST
»

DR (# 7EAh)
«
  I3 TEST
»
```

```
GA (# 37EDh)
«
  I4 TEST
»

VUE (# EF45h)
«
  "          " → S
  «
    CLLCD BL1 I1 CH BL1 + + I4 CH BL2 I3 CH + + BL1
    I2 CH BL1 + + S SWAP + 5 DISP S SWAP + 4 DISP S
    SWAP + 3 DISP "COUP No " COUP + 1 DISP 3 FREEZE
  »
»

INIT (# 4961h)
«
  LABS DUP SIZE RAND * 1 + IP GET LIST→ DROP 'LAB'
  STO 'SO' STO 1 'COUP' STO 0 0
  DO
    DROP2 AL AL DUP2 ETAT NOT
  UNTIL
  END
  'Y' STO 'X' STO VUE
»
```

MASTER

MASTER vous permettra de jouer au jeu bien connu du Master-Mind.

Il permet de jouer à chercher des combinaisons des chiffres "0" à "9". La longueur du code à trouver est quelconque. Pour la régler (ce qu'il est nécessaire de faire la première fois), il suffit de taper le nombre désiré et de lancer le programme `'STOL'` (qui initialisera le jeu).

Pour les parties suivantes, il suffira d'initialiser le jeu en exécutant le programme `'INIT'`.

Pour jouer, il faut alors proposer une combinaison sous forme de chaîne de caractères contenant les chiffres proposés et de lancer le programme `'MASTER'`.

Le programme affichera alors le nombre de chiffres exacts (placés et trouvés).

Par exemple, si la longueur du code est 4, que la solution est 8548 alors taper:

```
"8834" MASTER
```

provoquera l'affichage de:

```
Coup No x

8834

Placés=    1
Trouvés=   2
```

Puisque le premier "8" est à sa place, que le second et le "4" font partie du code mais ne sont pas à leur place...

A vous de jouer... après avoir entré les programmes qui suivent !

```
STOL (# CF28h)
«
  DUP
  IF
    TYPE 0 ==
  THEN
    'L' STO INIT
  ELSE
    514 DOERR
  END
»

INIT (# 49F5h)
«
  0 'CO' STO 1 L
  START
    RAND 10 * IP
  NEXT
  L →LIST 'SOL' STO
»

MASTER (# 28D7h)
«
  DUP
  IF
    TYPE 2 == DUP
  THEN
    DROP DUP SIZE L ==
  END
  IF
  THEN
    CLLCD DUP 3 DISP STL PROG 7 FREEZE
  ELSE
    514 DOERR
  END
»
```

```
PROG (# DD75h)
«
  0 0
  → PR CP CT
  «
    1 'CO' STO+ "Coup No " CO + 1 DISP SOL PR 1 L
    FOR X
      DUP X GET 3 PICK X GET
      IF
        ==
      THEN
        X -2 PUT SWAP X -1 PUT SWAP 1 CP + 'CP' STO
      END
    NEXT
    'PR' STO "Placés=    " CP + 5 DISP 1 L
    FOR X
      DUP X GET DUP
      IF
        -1 >
      THEN
        1 L
        FOR Y
          PR Y DUP2 GET 4 PICK
          IF
            ==
          THEN
            -2 PUT 'PR' STO 1 CT + 'CT' STO 4 'Y'
            STO
          ELSE
            DROP2
          END
        NEXT
      END
      DROP
    NEXT
    DROP "Trouvés=   " CT + 6 DISP
  »
»
```

```
STL (# 4DBCh)
«
  → S
  «
    ( ) 1 S SIZE
    FOR X
      S X X SUB STR→ +
    NEXT
  »
»
```

ANAG

Ce programme prend une chaîne de caractères en argument, et affiche tous les anagrammes possibles...

Par exemple:

```
"ABC" ANAG
```

provoquera l'affichage successif des chaînes de caractères:

```
"ABC" "ACB" "BAC" "BCA" "CAB" "CBA"
```

Voici les différents programmes à entrer:

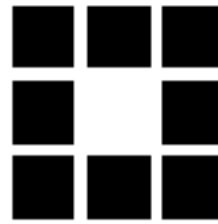
```
PRANAG (# A68Dh)
«
  IF
    B 0 >
  THEN
    -1 'B' STO+ PRDEPTH DUP B -
    FOR X
      X ROLL PRANAG X ROLLD -1
    STEP
    1 'B' STO+
  ELSE
    PRDEPTH DUPN PRDEPTH 2 / 1 - 1
    START
      + -1
    STEP
    4 DISP
  END
»
```

```
PRDEPTH (# EAFFh)
«
  DEPTH C -
»

ANAG (# 1F82h)
«
  → A
  «
    CLLCD A SIZE 'B' STO DEPTH 'C' STO 1 B
    FOR X
      A X DUP SUB
    NEXT
    PRANAG PRDEPTH DROPN ( B C ) PURGE
  »
»
```

CARRE

Le but de ce jeu est de reconstituer le "carré magique", c'est à dire la figure suivante:



Pour réussir cette tâche, le joueur peut appuyer sur les différentes cases (par l'intermédiaire des chiffres "1" à "9"). En effet, l'appui sur une des cases inverse sa couleur... ainsi que celle de certaines des cases voisines !

Pour jouer, il suffit de rentrer les différents programmes et de lancer 'CARREM'...

```
KEYS (# 2CE6h)
( 82 83 84 72 73 74 62 63 64 )

MESS (# A73Ch)
"CALCUL EN COURS..."

T (# 6459h)
{
  { 1 2 4 5 }   { 1 2 3 } { 2 3 5 6 } { 1 4 7 }
  { 2 4 5 6 8 } { 3 6 9 } { 4 5 7 8 } { 7 8 9 }
  { 5 6 8 9 }
}

M (# EE2h)
{ "     789   →" "     456   →" "     123   →" }
```

```
CALC (# E30Ah)
«
  "Press a key..." 1 DISP T 1
  DO
    DROP KEYS
    DO
    UNTIL
      KEY
    END
  UNTIL
    POS DUP
  END
    1000 .05 BEEP MESS 1 DISP GET DUP 1 DUP ROT SIZE
  START
    GETI 1 - DUP 3 MOD 1 +
    WHILE
      DUP 3 >
    REPEAT
      3 -
    END
    SWAP 3 / IP 1 + SWAP 2 →LIST CAR SWAP DUP2 GET
    NOT PUT 'CAR' STO
  NEXT
  DROP2
»

SOL (# C888h)
[[ 1 1 1 ]
 [ 1 0 1 ]
 [ 1 1 1 ]]

CAR (# C888h)
[[ 1 1 1 ]
 [ 1 0 1 ]
 [ 1 1 1 ]]
```

```
VISU (# E530h)
«
  DO
    CAR ( 1 1 ) 1 3
    FOR X
    "" 1 3
      START
        3 ROLLD GETI 95 * 32 + CHR 4 ROLL SWAP +
      NEXT
      M X GET SWAP + 142 CHR + 3 ROLLD
    NEXT
    DROP2 2 4
    FOR X
      X 1 + DISP
    NEXT
    CALC
  UNTIL
    CAR SOL ==
  END
    "       Bravo..." 1 DISP 1 3
  START
    1000 .2 BEEP
  NEXT
»

CARREM (# 2DC2h)
«
  CLLCD MESS 1 DISP 0 RDZ CAR
  DO
    ( 1 1 ) 1 9
    START
      RAND .5 > PUTI
    NEXT
    DROP DUP
  UNTIL
    SOL ≠
  END
  'CAR' STO VISU
»
```

PR40

Ce programme permet d'imprimer des chaînes de caractères sur l'imprimante HP82240A en 40 caractères par ligne au lieu de 24.

La chaîne donnée en entrée peut comporter des retours-chariot (↵), les lignes de plus de 40 caractères sont scindées (comme le fait la fonction PR1)

Le principe du programme est simple: il prend la chaîne donnée et la découpe:

- Par tranches délimitées par des retours-chariot ;
- Puis en morceaux d'au plus 40 caractères.

Chacun des morceaux obtenus est transformé en objet graphique du texte écrit dans la petite police (par `1 →GROB`) et imprimé grâce à la fonction `PR1`. De ce fait les lettres minuscules sont transformées en majuscules.

Ce programme est particulièrement utile pour imprimer les listings obtenus avec le désassembleur.

```
PR40 (# 7B55h)
 « "↵" + → S
   «
     WHILE S SIZE
     REPEAT S DUP "↵" POS DUP2 1 + OVER SIZE SUB 'S'
       STO 1 SWAP 1 - SUB → T
       «
         WHILE T SIZE
         REPEAT T 1 40 SUB 1 →GROB PR1 DROP T 41
           OVER SIZE SUB 'T' STO
         END
       »
     END
   »
 »
```

DSP

Ces deux programmes permettent d'utiliser l'écran de la HP48 en mode 33 colonnes.

De plus, l'affichage défile, ligne par ligne, afin de permettre une vision globale d'un texte en train de s'afficher.

Les deux programmes réalisent les fonctions suivantes:

- INITSCR efface l'écran et initialise la mémoire d'écran utilisée pour réaliser le défilement ;
- DSP qui affiche le message ligne par ligne en décalant vers le haut les textes déjà affichés.

Pour obtenir un texte en petits caractères, on utilise la fonction →GROB en petite police.

On crée un objet graphique pour chacune des lignes à afficher et on les conserve dans la variable SCREEN sous forme de liste.

On les ajoute (par OR sur un GROB blanc) puis on affiche le résultat grâce à →LCD.

Il peut être utilisé avec le programme DESASS (programme de désassemblage automatique) pour la visualisation du listing des programmes assembleurs décodés (au lieu de la classique fonction DISP).

Pour ce faire, il suffit de remplacer "1 DISP" dans la fonction STOS de DESASS par DSP et de rajouter INITSCR au début du programme DESASS.

```
INITSCR (# 424h)
«
  ( ) 'SCREEN' STO CLLCD
»

DSP (# 70A4h)
«
  IF
    "⌟" OVER DUP SIZE DUP SUB OVER ≠
  THEN
    +
  ELSE
    DROP
  END
  → TXT
  «
    WHILE
      TXT 1 OVER "⌟" POS DUP
    REPEAT
      3 DUPN SWAP + OVER SIZE SUB 'TXT' STO 1 -
      SUB 1 →GROB SCREEN + 1 9 SUB 'SCREEN' STO
      # 83h # 40h BLANK 1 SCREEN SIZE DUP # 6h *
      → O
      «
        FOR X
          # 0h O # 6h X * - 2 →LIST SCREEN X GET GOR
        NEXT
      »
      →LCD
    END
    3 DROPN
  »
»
```

MUSICLM

MUSICLM permet de jouer un air de musique sans interruption entre les notes.

Il s'agit d'un programme en langage machine devant être paramétré. Ce paramétrage étant quelque peu délicat, il a été automatisé par le programme RPL MUSICLM listé ci-dessous.

Ce programme prend en entrée la liste des notes (sous la forme de la suite fréquence durée...) et crée le programme-machine correspondant.

Il utilise pour cela les deux programmes GASS et A->STR déjà vus.

Exemple: `{ 1400 .1 2800 .1 1400 .1 } MUSICLM EVAL`

Remarque: l'objet `'Code'` (dans la pile avant le EVAL) peut être stocké dans une variable pour être réutilisé ensuite...

Voici tout d'abord MUSICLM (le listing désassemblé de la routine en langage machine se trouve page suivante).

```
MUSICLM (# EC8h)
«
  → L
  «
    "CCD20" # 4Fh L SIZE 2 + 5 * + A→STR +
    "8FB97608E" + L SIZE 2 + 5 * A→STR 1 4 SUB + 1 L
    SIZE
    FOR X
      L X GET A→STR + L X 1 + GET 1000 * A→STR + 2
    STEP
    "0000000000071351470717414317413706D68AAD08F6A7"
    "1069DF078F2D760142164808C" + + GASS
  »
»
```

```
          CCD20     CON(5)    PROL_CODE      Objet code
début     *****     CON(5)    (fin)-(début)  Longueur code
          8FB9'/60  GOSBVL    SAVE_REG       Sauvegardes
          8E****    GOSUBL    l1

LISTE DES NOTES Fréquence / Durée (en millisecondes)

          CON(5)    #00000                   Fin des notes
          CON(5)    #00000
l1        0'/       C=RSTK
          135       D1=C
          14'/      C=DAT1    A              Lecture fréquence
          D'/       D=C       A
          1'/4      D1=D1+    5
          143       A=DAT1    A              Lecture durée
          1'/4      D1=D1+    5
          13'/      CD1ex
          06        RSTK=C
          D6        C=A       A
          8AA       ?C=0      A              Terminé ?
          D0        GOYES     l2
          8F6A'/10  GOSBVL    BEEP_LM        On beepe..
          69DF      GOTO      l1             On itère
l2        0'/       C=RSTK
          8F2D'/60  GOSBVL    LOAD_REG       Récupérations
          142       A=DAT0    A              Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin
```

MODUL

Ce programme en langage-machine permet l'émission de sons modulés entre deux fréquences. On le règle en fixant une fréquence de départ (DEPART), une fréquence de fin (FIN), un incrément de fréquence (INCREMENT) et une durée de palier (PALIER).

Ces réglages sont automatisés par le programme RPL MODUL (listé plus loin) qui crée automatiquement le programme-assembleur en fonction des 4 paramètres déjà cités.

Ce programme utilise GASS et A->STR (listés précédement).

Exemple d'utilisation: `1400 2800 50 .01 MODUL EVAL`

Remarque: l'objet `'Code'` (dans la pile avant le `EVAL`) peut être stocké dans une variable pour être réutilisé ensuite...

Voici tout d'abord le listing commenté de la routine-assembleur crée par MODUL. Les étoiles (*) représentent les codes dépendant des 4 paramètres.

```
          CCD20     CON(5)    PROL_CODE      Objet code
début     15000     CON(5)    (fin)-(début)  Longueur code
          8FB9'/60  GOSBVL    SAVE_REG       Sauvegardes
          34*****   LCHEX     DEPART         Frequ de départ
          D'/       D=C       A
l1        DB        C=D       A
          06        RSTK=C
          34*****   LCHEX     PALIER         (en millisecondes)
          8F6A'/10  GOSBVL    BEEP_LM        On beepe !
          0'/       C=RSTK
          D'/       D=C       A
          34*****   LCHEX     INCREMENT      On incrémente
          C3        D=D+C     A
          34*****   LCHEX     FIN            Fréquence de fin
          ***       ?D<=C     A              ou ?D>=C A
          '/D       GOYES     l1             Si encore...
          8F2D'/60  GOSBVL    LOAD_REG       Récupérations
          142       A=DAT0    A              Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin
```

```
MODUL (# E9CEh)
«
  → D F I P
  «
    IF
      P 1000 * DUP 'P' STO # 0h + # 0h ==
    THEN
      "PALIER NUL..." DOERR
    END
    IF
      I # 0h + # 0h ==
    THEN
      "INCREMENT NUL..." DOERR
    END
    "CCD20150008FB976034" D A→STR + "D7DB0634" + P
    A→STR + "8F6A71007D734" + I A→STR +
    IF
      D F <
    THEN
      "C3"
    ELSE
      "E3"
    END
    + "34" + F A→STR +
    IF
      D F <
    THEN
      "8BF"
    ELSE
      "8BB"
    END
    + "7D8F2D760142164808C" + GASS
  »
»
```

RABIP

Ce petit programme génère des sons aléatoires, à des fréquences comprises entre 0 et 4400 Hz, pour des durées de 0 à 0,1 seconde.

Il s'arrête par appui sur une touche quelconque.

Il peut servir à signaler, d'une manière originale, la fin de l'exécution d'un programme long...

```
RABIP (# A75Bh)
«
  DO
    4400 RAND * .1 RAND * BEEP
  UNTIL
    KEY
  END
  DROP
»
```

JINGLE

Ce programme joue le petit air de musique, dont les notes sont contenues dans la liste SOUNDS (dont un exemple est donné ci-après).

A noter que dans SOUNDS, les notes sont stockées en ordre inverse: la dernière paire fréquence/durée correspond à la première note jouée...

```
JINGLE (# 83E1h)
«
  SOUNDS LIST→ 1 SWAP 2 / MEM DROP
  START
    BEEP
  NEXT
»

SOUNDS (# 9A73h)
{
  390 .75 440 .15 275 .15 350 .075 350 .15 390 .075
  690 .15 565 .15 390 .15 465 .15 565 .15 590 .075
  390 .075 390 .15 565 .3 390 .3 350 .15 390 .15 515
  .075 390 .075 390 .15 465 .3 390 .3
}
```

RENAME

Ce programme permet de renommer un objet. Il prend en arguments l'ancien nom de l'objet et son nouveau nom.

Remarque: l'objet est renommé sans modification de sa position dans le répertoire...

```
RENAME (# 1A24h)
«
  OVER RCL SWAP STO VARS DUP2 SWAP POS 2 SWAP SUB
  ORDER PURGE
»
```

AUTOST

AUTOST est un exemple de programme en démarrage automatique. Il peut être enrichi au gré de l'utilisateur.

Il sera assigné à la touche [OFF]. Le programme suivant réalise automatiquement l'installation (assignation de la touche [OFF] et passage en mode USER).

```
AUTOST (# BCE5h)
«
  «
    CLLCD OFF 1400 .07 BEEP "HP48 : READY" 1 DISP
    1000 .01 BEEP .5 WAIT
  »
  91.3 ASN -62 SF
»
```

CAL

CAL permet d'afficher un calendrier sur un mois. Il prend en entrée une liste de deux réels spécifiant le mois en question: cette liste doit contenir le numéro du mois désiré (entre 1 et 12) ainsi que l'année (entre 1583 et 9999).

Cependant, pour simplifier l'utilisation de ce programme, il est prévu que:

- Si la liste ne contient qu'un élément, celui-ci est considéré comme un numéro de mois, et l'année utilisée est l'année courante;
- Si la liste est vide, on affiche le calendrier du mois courant.

Voici le listing de CAL:

```
CAL (# 282h)
«
  CLLCD # 4E2CFh SYSEVAL RCLF
  → F
  «
    -42 SF ( ) + DATE FP 100 * SWAP OVER IP + SWAP
    FP 10000 * + DUP DUP SIZE 2 MOD 2 + GET SWAP 1
    GET
    → Y M
    «
      1.0119 1 M 100 / + Y 1000000 / + DDAYS 7 MOD
      → S
      «
        { "JANVIER" "FEVRIER" "MARS" "AVRIL" "MAI"
        "JUIN" "JUILLET" "AOUT" "SEPTEMBRE"
        "OCTOBRE" "NOVEMBRE" "DECEMBRE" } M GET
        " " + Y + "            " 1 22 4 PICK SIZE -
        2 / SUB SWAP + 1 DISP
        " LU MA ME JE VE SA DI" 2 DISP { 31 28 31 30
        31 30 31 31 30 31 30 31 } M GET M 2 == Y 4
```

```
          MOD 0 == Y 100 MOD 0 == - Y 1000 MOD 0 == +
          AND +
          → N
          «
            0 5
            FOR L
              "" 1 7
              FOR C
                L 7 * C + S - "    " SWAP
                IF
                  DUP 0 > OVER N ≤ AND
                THEN
                  +
                ELSE
                  DROP
                END
                DUP SIZE DUP 2 - SWAP SUB +
              NEXT
              L 16 * # 1247Bh + SYSEVAL
            NEXT
            7 FREEZE
          »
        »
      »
    »
  »
```

CIRCLE

CIRCLE réalise un tracé rapide de cercles à l'écran. Il utilise pour ce faire l'algorithme de BRESENHAM. Il prend deux arguments en entrée:

- Un réel représentant le diamètre du cercle à tracer. Si ce diamètre est négatif, c'est un cercle blanc de diamètre égal à la valeur absolue de ce paramètre, qui sera tracé;
- Un complexe donnant les coordonnées de son centre.

Remarques:

- Ces deux arguments sont laissés dans la pile. Si ils sont inutiles pour la suite, il conviendra de les détruire (par DROP2);
- Ce programme s'automodifiant, il ne faut pas l'utiliser sous forme backup (sauvegarde dans un port).

Trois programmes de démonstration (TEST1, TEST2 et TEST3) vous permettront d'admirer la vitesse de tracé...

Le listing désassemblé de CIRCLE étant fort long, il ne sera pas donné ici.

CIRCLE a été écrit par Christophe NGUYEN.

```
TEST1 (# D683h)
«
  ERASE ( # 0h # 0h ) PVIEW 1 1000
  START
    RAND 20 * RAND 131 * 65 - RAND 64 * 32 - R→C
    CIRCLE DROP2
  NEXT
»
```

```
TEST2 (# 58EEh)
«
  ERASE ( # 0h # 0h ) PVIEW 10 (0,0) 1 20
  START
    CIRCLE DUP2 RAND 10 * 5 - RAND 10 * 5 - R→C +
    CIRCLE
  NEXT
  1 1000
  START
    DUP2 RAND 10 * 5 - RAND 10 * 5 - R→C + CIRCLE
    DEPTH ROLL -1 * DEPTH ROLL CIRCLE DROP2
  NEXT
»

TEST3 (# 35EFh)
«
  INIT DEG
  DO
    -180 180
    FOR T
      5 T * COS 60 * 7 T * SIN 30 * R→C 3 OVER
      CIRCLE DROP2 DEPTH ROLL -3 SWAP CIRCLE DROP2 2
    STEP
  UNTIL
    0
  END
»

INIT (# 50F1h)
«
  ERASE 1 20
  START
    (100,100)
  NEXT
  ( # 0h # 0h ) PVIEW
»
```

CIRCLE (# 9965h)

```
D9D20 2ABF1 3FBF1 CCD20 99300 8FB97 60201 37135
06147 13517 41371 35067 42110 B0713 517F7 51110
C0706 13517 41471 35174 13713 575F0 10807 13517
41471 35174 17E20 15719 1A511 10F81 00808 21716
A0080 82190 70000 7D534 4D200 C9137 1491B 56507
14213 216EA F0142 8A867 70000 7D534 A6200 C9137
14181 C1CD1 41AF0 142CC 81C81 CE4E4 81C1C F1C01
CF141 184AF 01428 A8721 CD141 81C1C D1411 6917F
17F17 41321 417BB 0208F 2D760 14216 4808C 13713
51341 6ED21 53332 0059B E0332 4009B 2A032 003AB
AAB61 7D133 EA131 80D0D 21571 6900D 22001 23D1D
71F81 C0013 37F20 1FC90 00133 71201 F6000 01337
310DB E9152 19184 0FA20 01A8B 90AB0 C8A0E 65FF0
D01D0 10111 0102C 43430 000E2 10811 111AE A2430
F9026 06160 7B601 10243 0F906 D1119 C6C6C A2034
60000 CA100 69201 1AD51 19E9C 6C6CA 2034A 0000C
A1001 1ACE1 0A119 E6109 6F8F1 1111A 8A600 72000
18407 54011 9D511 A109D 910A7 130F9 D910A 76201
19FA1 0AF9D 91097 310F9 D9109 87000 8506A BF201
1A808 24020 00EA1 1CEA3 40400 08BE0 0C434 11000
D7340 3217C F480C 268FF 13711 13414 000CA 11BCA
34380 008BE 00D6A F0DAA E781C 81C13 7C213 5A64A
64AEB B62AE A301A 6C490 A1666 FF153 10E1E 15110
11531 B9C0E 1EB9C 15110 1B213 0
```

BANNER

Le programme BANNER permet de faire défiler un texte à l'écran en caractères géants.

Remarques:

- les caractères acceptés sont ceux dont le code ASCII est compris entre 31 et 90 (en particulier les chiffres, la ponctuation et les lettres majuscules);
- BANNER utilise une table pour dessiner les caractères. Cette table est générée automatiquement par le programme MKT listé plus loin. De ce fait, la méthode d'entrée des codes varie quelque peu... Pour entrer BANNER il faut procéder de la manière suivante:
  - Entrer les codes BANNER1, sous forme d'une chaîne de caractères, en une seule ligne, sans espace; placer la chaîne dans la pile;
  - Entrer et exécuter le programme MKT (qui produira une chaîne de 2100 caractères);
  - Entrer les codes de BANNER2, sous forme d'une chaîne de caractères, en une seule ligne, sans espace; placer la chaîne dans la pile;
  - Concaténer les trois chaînes (en appuyant deux fois sur [+]);
  - Exécuter GASS (ou RASS) et stocker le résultat (0 CHR + CLLCD Code DROP) dans 'BANNER'.

Pour utiliser BANNER, il suffit de lui donner une chaîne de caractères en argument et d'admirer le résultat...

Exemple: "VOYAGE AU CENTRE DE LA HP48..." BANNER

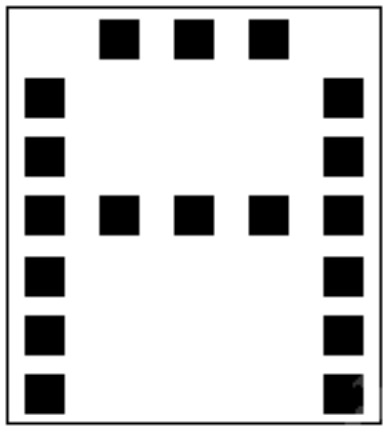
Vous trouverez tout d'abord le listing désassemblé et commenté de BANNER, suivi du rappel des codes à entrer (BANNER1 et BANNER2) ainsi que le programme MKT.

BANNER a été écrit par Christophe NGUYEN.

```
         D9D20     CON(5)    PROL_PRGM
         4B2A2     CON(5)    #2A2B4          Réel nul.
         66BC1     CON(5)    #1CB66          CHR
         76BA1     CON(5)    #1AB67          Addition
         858A1     CON(5)    #1A858          CLLCD
         CCD20     CON(5)    PROL_CODE
début    23A00     CON(5)    (fin)-(début)
         8FB976O   GOSBVL    SAVE_REG
         1BE0507   D0=(5)    70505E
         142       A=DAT0    A               A=@ bitmap écran
         3412000   LCHEX     #00021
         C2        C=C+A     A
         134       D0=C
         10A       R2=C                      Position courante
         137       CD1ex
         135       D1=C
         06        RSTK=C
         AE0       A=0       B
         8082180   LAHEX     #08             Hauteur gros pixel
         100       R0=A
         AE0       A=0       B
         8082120   LAHEX     #02             Largeur gros pixel
         101       R1=A                      (2 fois 4 qu.)
         07        C=RSTK
         135       D1=C
         143       A=DAT1    A               A=@ chaîne
         131       D1=A
         179       D1=D1+    10              D1=@premier carac.
         137       CD1ex
         135       D1=C
         06        RSTK=C
Boucle   1BE0507   D0=(5)    70505E
         142       A=DAT0    A
         130       D0=A                      D0=@ bitmap écran
         16F       D0=D0+    16
         16F       D0=D0+    16
         160       D0=D0+    1               Position ds écran
         07        C=RSTK
         135       D1=C                      D1=@ carac
         D0        A=0       A
         14B       A=DAT1    B               Lecture 1 carac.
         96C       ?A#0      B               CHR(0) ?
         80        GOYES     Cont            On continue..!
         8CD990    GOLONG    Bye_bye         Terminé..
Cont     34F1000   LCHEX     #0001F          ;
         EE        C=A-C     A               ;Calcul de l'
         DA        A=C       A               ;offset pour
         C6        C=C+C     A               ;retrouver la
         C6        C=C+C     A               ;représentation du
         C6        C=C+C     A               ;caractère.
         C6        C=C+C     A               ;1 carac= 35 data
         C6        C=C+C     A               ;Carac entre 31 et
         C2        C=C+A     A               ;90 donc:
         C2        C=C+A     A               ;offset=(c-31)*35
```

```
          C2        C=C+A     A              ;
          DA        A=C       A              ;
          8E4380    GOSUBL    Get_code       Gosub après les
                                             datas (pour en
                                             déterminer
                                             l'adresse)
```

```
* Fin de BANNER1 et début du codage des caractères..
*
* Ce codage s'effectue 1 quartet par pixel (0 ou F) par des
* raisons de rapidité d'exécution.. Il se fait colonne par
* colonne. Exemple, pour la lettre A:
*
```



Æ

0	F	F	F	0
F	0	0	0	F
F	0	0	0	F
F	F	F	F	F
F	0	0	0	F
F	0	0	0	F
F	0	0	0	F

```
*
* D'où le codage de "A":
* 0FFFFFFF00F000F00F000F00F0000FFFFFF..
*
* Fin du codage des caractères et début de BANNER2

Get_code  07        C=RSTK                    C=@ des data
          CA        A=A+C     A               Ajout offset
          171       D1=D1+    2               @carac. suivant
          137       CD1ex
          06        RSTK=C                    Sauvegarde
          131       D1=A
          305       LCHEX     #5              5 colonnes..
          A97       D=C       WP
Suivant   A1F       D=D-1     WP
          560       GONC      Déb_col         si pas terminé..
          6F70      GOTO      Blanc           sinon -> blanc
Déb_col   3170      LCHEX     #07             7 lignes..
          AE5       B=C       B
Ecr_col   A6D       B=B-1     B
          472       GOC       Fin_col         terminé..
          1531      A=DAT1    WP              lecture pixel
          118       C=R0                      Hauteur gros pixel
Repète_H  A1E       C=C-1     WP
          431       GOC       Fin_répH        Terminé..
          1501      DAT0=A    WP              écriture
          16F       D0=D0+    16              ; Passage à la
```

```
          16F       D0=D0+    16             ; ligne suivante
          161       D0=D0+    2              ;
          6CEF      GOTO      Repète_H
Fin_répH  1?0       D1=D1+    1              Gros pixel suivant
          68DF      GOTO      Ecr_col
Fin_col   119       C=R1                     On a écrit à
          AE5       B=C       B              droite de l'écran:
          1BE0S0?   D0=(5)    ?0S0E          il faut à présent
          146       C=DAT0    A              décaler vers la
          134       D0=C                     gauche..
          118       C=R0                     Recalcul du nombre
          AEA       A=C       B              de lignes à
          A64       A=A+A     B              décaler..
          A64       A=A+A     B
          A64       A=A+A     B
          B6E       C=A-C     B
Répète_L  A6D       B=B-1     B              Agrandissement en
          4?1       GOC       Col_suiv       largeur..
          1BE0S0?   D0=(5)    ?0S0E
          142       A=DAT0    A
          130       D0=A
          ?D50      GOSUB     Gauche         Décalage..
          68EF      GOTO      Répète_L
Col_suiv  11A       C=R2
          134       D0=C
          6D?F      GOTO      Suivant

Blanc     11A       C=R2                     Affichage du blanc
          134       D0=C                     entre deux
          118       C=R0                     caractères..
          AEA       A=C       B
          A64       A=A+A     B
          A64       A=A+A     B
          A64       A=A+A     B
          B6E       C=A-C     B
          AE5       B=C       B
          A90       A=0       WP
Parti     A6E       C=C-1     B              Ecriture d'une
          431       GOC       Blgauche       colonne blanche
          1501      DAT0=A    WP
          16F       D0=D0+    16
          16F       D0=D0+    16
          161       D0=D0+    2
          6CEF      GOTO      Parti
Blgauche  AE9       C=B       B              Décalage
          1BE0S0?   D0=(5)    ?0S0E
          142       A=DAT0    A
          130       D0=A
          ?600      GOSUB     Gauche
          8C896F    GOLONG    Boucle

Gauche    06        RSTK=C                   Décalage de la
          AE4       A=B       B              partie visible
          AE5       B=C       B              de l'affichage..
          AE6       C=A       B              C=nbr lignes
```

```
          06        RSTK=C                    D0=@ bitmap écran
Bcle_gau  A6D       B=B-1     B
          5?1       GONC      Suitegau
          0?        C=RSTK
          AE5       B=C       B
          34BB000   LCHEX     #000BB         Temporisation pour
Wait      CE        C=C-1     A              ralentir le
          5DF       GONC      Wait           décalage..
          0?        C=RSTK
          01        RTN                      Ok, terminé !
Suitegau  2F        p=        15             Décalage d'une
          1521      A=DAT0    WP             seule ligne
          B94       ASR       WP
          1501      DAT0=A    WP
          16E       D0=D0+    15
          1521      A=DAT0    WP
          B94       ASR       WP
          1501      DAT0=A    WP
          16E       D0=D0+    15
          142       A=DAT0    A
          F4        ASR       A
          1503      DAT0=A    X
          20        p=        0
          163       D0=D0+    4              Ligne suivante
          68BF      GOTO      Bcle_gau       On continue

Bye_bye   8F2D?60   GOSBVL    LOAD_REG       Récupérations
          142       A=DAT0    A              Retour au RPL
          164       D0=D0+    5
          808C      PC=(A)
fin       8DBF1     CON(5)    DROP
          B2130     CON(5)    EPILOGUE
```

Voici le récapitulatif des codes à entrer (attention, la méthode d'entrée est légèrement différente de la méthode standard: lire à ce sujet l'avertissement énoncé plus haut).

BANNER1 (# 4C06h)

```
D9D20 4B2A2 66BC1 76BA1 858A1 CCD20 23A00 8FB97
601BE 05071 42341 2000C 21341 0A137 13506 AE080
82180 100AE 08082 12010 10713 51431 31179 13713
5061B E0507 14213 016F1 6F160 07135 D014B 96C80
8CD99 034F1 000EE DAC6C 6C6C6 C6C2C 2C2DA 8E438
0
```

MKT (# DF20h)

```
«
  "" ( # 0h # 0h ) PVIEW 31 90
  FOR A
    PICT ( # 0h # 0h ) A CHR 2 →GROB REPL 0 4
    FOR X
      0 6
      FOR Y
        IF
          X R→B Y R→B 2 →LIST PIX?
        THEN
          "F"
        ELSE
          "0"
        END
        +
      NEXT
    NEXT
  NEXT
»
```

BANNER2 (# B995h)

```
07CA1 71137 06131 305A9 7A1F5 606F7 03170 AE5A6
D4721 53111 8A1E4 31150 116F1 6F161 6CEF1 7068D
F119A E51BE 05071 46134 118AE AA64A 64A64 B6EA6
D4711 BE050 71421 307D5 068EF 11A13 46D7F 11A13
4118A EAA64 A64A6 4B6EA E5A90 A6E43 11501 16F16
F1616 CEFAE 91BE0 50714 21307 6008C 896F0 6AE4A
E5AE6 06A6D 57107 AE534 BB000 CE5DF 07012 F1521
B9415 0116E 1521B 94150 116E1 42F41 50320 16368
BF8F2 D7601 42164 808C8 DBF1B 2130
```

Annexes

Réponses aux exercices

I) Première partie: la HP48

1) Chapitre 1

A-1-1: [↰] [0]

A-1-2: [↱] [STO]

2) Chapitre 2

A-2-1: Une séquence possible est:

[5] [ENTER] [3] [ENTER] [1] [+] [9] [ENTER] [5] [-] [*] [/]

(certaines fonctions, comme [+], [-], [*]... effectuent un [ENTER] automatique avant de s'exécuter...).

A-2-2: Par exemple [SWAP] [ROT]

A-2-3: COS((3*5)-11)/4-1) ce qui donne 1 (COS(0)).

3) Chapitre 3

A-3-1: Taper [↱] ['] ['] [E] [X] [O] [ENTER] [↰] [VAR] [CRDIR] [VAR] [EXO] [1] ['] [A] [STO] [2] ['] [B] [STO] [3] ['] [C] [STO]

A-3-2: 6 (PARTS, PROB, HYP, MATR, VECTR et BASE).

4) Chapitre 4

A-4-1: « → A B « A B + » » peut additionner deux chaînes de caractères.

A-4-2: Il calcule la fraction (A+B)/(A*B) où A et B sont deux réels pris dans la pile.

A-4-3: Par exemple:

```
FIBO (# 5B7Eh)
«
  → N
  «
    IF
      N 1 ≤
    THEN
      1
    ELSE
      N 1 - FIBO N 2 - FIBO +
    END
  »
»
```

II) Deuxième partie: le langage machine

1) Chapitre 1

B-1-1: 1h, Ah, 19h, FFFFh, BEBEh.

B-1-2: 291, 16, 256, 2898, 3.

2) Chapitre 2

B-2-1: B73, AFB.

B-2-2: Pour **P**: B03 et A8B ; Pour **WP**: B13 et A9B.

B-2-3: A13 D=D+C WP, A73 D=D+C W, D=0 P, D=0 WP.

B-2-4: 411.

B-2-5: 41.

B-2-6: C411.

B-2-7: #70080h:0, #70081h:1, #70082h:2.

B-2-8: C champ X contient 210, C champ B contient 10 et C champ XS contient 2.

B-2-9: 3 (les quartets 0, 1 et 2).

3) Chapitre 3

B-3-1: Le programme se code de la manière suivante:

```
CCD20           CON(5)    #02DCC
45000    debut  CON(5)    (fin)-(debut)

6310            GOTO      l1

CC       subl   A=A-1     A
3454321         LCHEX     #12345
CE       l2     C=C-1     A
5DF             GONC      l2
03              RTNCC

3450000  l1     LCHEX     #00005
DA              A=C       A
?CEF     l3     GOSUB     l2
8AC             ?A#0      A
9F              GOYES     l3

3410000         LCHEX     #00001
DA              A=C       A
?110            GOXUB     l4
8A8             ?A=0      A
40              GOYES     l5
CC              A=A-1     A
142      l5     A=DAT0    A
164             D0=D0+    5
808C            PC=(A)

8AA      l4     ?C=0      A
00              RTNYES
D2              C=0       A
E4              A=A+1     A
01              RTN
         fin
```

D'où le listing des codes:

```
CCD20 45000 6310C C3454 321CE 5DF03 34500 00DA7
CEF8A C9F34 10000 DA711 08A84 0CC14 21648 08C8A
A00D2 E401
```

B-3-2: Le listing se décode en:

```
143             A=DAT1   A
133             AD1EX
179             D1=D1+   10
1577            C=DAT1   W
B76             C=C+1    W
1557            DAT1=C   W
131             D1=a
142             A=DAT0   A
164             D0=D0+   5
808C            PC=(A)
```

4) Chapitre 4

B-4-1: Le Short Integer <54321h>.

B-4-2: 11920 EDCBA.

B-4-3: 11920 B7000.

B-4-4: 33920 100 000000000021 0.

B-4-5: -77345.

B-4-6: Elle perd de la précision et le coderait:

55920 51000 543210987654321 0.

B-4-7: -1E-2 (-0,01).

B-4-8: 77920 000 000000000001 0 10000 0000000002 0

B-4-9: (-33,33).

B-4-10: D9920 00000 000000000000000 0 00000...
...000000000000000 0

B-4-11: Le complexe étendu:

(1.23456789012345,-543210987654321)

B-4-12: FB920 34

B-4-13: Le caractère 'D' (code ASCII 44h).

B-4-14: 8E920 67200 11920 30000 30000 50000 80000...

B-4-15: Elle contient des chaînes de caractères.

B-4-16: C2A20 B1000 84 56 C6 C6 F6 02 75 F6 27 C6 46

B-4-17: "Bravo !"

B-4-18: E4A20 51000 1BF7935000000000

B-4-19: #54321h

B-4-20: 47A20B2130

B-4-21: { OK }

B-4-22: 69A20 FF7 12000 00000 10 44 10 C2A207000033 21000 10 14 10 C2A207000043

B-4-23: 69A20 100 321 03EF7 00000 12000 00000 10 44 10 C2A207000033 21000 10 14 10 C2A207000043

B-4-24: 8BA20 84E201014 84E201024 76BA1 B2130

B-4-25: 'A*(B-C)'

B-4-26: ADA20 3392000000000000000210 C2A2070000D6 68B01 B2130

B-4-27: 5,1 m^3

B-4-28: CFA20 20 55E4 84E2030451474

B-4-29: OK:CORRAL

B-4-30: GROB 4 1 F0

B-4-31: #6FFh

B-4-22: 'VIDE'

B-4-33: non.

B-4-34: 'BCKP'

B-4-35: #62D6h

B-4-36: Par #2361Eh et #23639h.

B-4-37: Par #1AB67h.

B-4-38: CCD20 50000

B-4-39: CCD20 F0000 142 164 808C. Il s'agit du programme:

```
142             A=DAT0   A
164             D0=D0+   5
808C            PC=(A)
```

C'est à dire un programme ne faisant que passer à l'objet suivant...

B-4-40: 84E20 50 84 56 C6 C6 F6.

B-4-41: Un nom vide.

B-4-42: 4.

B-4-43: 'Name'.

B-4-44: 29E20 654 321

B-4-45: Librairie #001h, commande #002h.

Informations diverses

Voici quelques informations utiles...

- Déterminer le numéro de version de sa machine: allumer la HP48, appuyer sur [ON] et en continuant à le presser, appuyer sur [D]. Relâcher [D] puis [ON]. Trois barres noires apparaissent à l'écran. Appuyer alors sur la touche Backspace (←, DROP, CLR), le texte "705D9:1B8DA178E5A111B6" s'affiche en haut de l'écran. Presser alors [EVAL]. Un texte s'affiche lors un court instant:

  ```
  Version HP48-?
  Copyright HP 1989
  ```

  Où '?' est le numéro de version (A, B, C, D...). Pour revenir à l'état normal presser sur les touches [ON] [C] (de la même manière que pour [ON] [D]).

- En cas de plantage de la machine: dans le cas où la HP48 ne réagit plus à vos sollicitations, essayez, dans cet ordre, les différentes réinitialisations possibles:
  - [ON] qui interrompt la plupart des programmes en cours, sans danger pour le contenu de la mémoire;
  - [ON] [C] qui est beaucoup plus fort et qui respecte aussi l'intégrité de la mémoire (sauf la pile);
  - [ON] [A] [F] qui nettoie la mémoire. Deux solutions sont possibles: répondre oui (yes) ou non (no) à la proposition de tentative de restauration de celle-ci. Il faut noter que cette tentative de restauration peut échouer dans le cas de graves problèmes... Le temps de restauration peut alors tendre vers l'infini, et il faut alors utiliser le point suivant:
  - Sous la HP48 on trouve 4 patins en caoutchouc qui ne sont pas collés: on peut les ôter facilement. Sous celui situé en haut (côté trappe d'accès aux cartes) à gauche ou à droite (selon les modèles), se trouve un petit

trou, à côté duquel est écrit la lettre 'R' (comme Reset). En enfonçant doucement un objet fin (trombone par exemple) dans ce logement, on provoque une remise à zéro de la machine. Si le temps d'appui est court, les données stockées par l'utilisateur seront préservées. En appuyant plus longtemps (une à deux secondes), la réinitialisation sera totale; ATTENTION: ce bouton est fragile, il convient de ne l'utiliser qu'en cas de nécessité extrême...

- En désespoir de cause on peut enlever les piles de la HP48. Celle-ci possédant une sauvegarde de puissance par condensateur, il est donc nécessaire de les décharger. Deux solutions sont possibles: attendre quelques heures (!) ou placer les piles à l'envers (+ à la place de moins) pendant quelques secondes (sans danger, la HP48 étant protégée par des diodes). Remplacer ensuite les piles dans le sens normal;
- Si aucune de ces méthodes n'a eu de succès, la meilleure solution est d'aller demander conseil à un revendeur Hewlett-Packard agréé...

- Quand et où votre HP48 a-t-elle été fabriquée ?

  La réponse est simple: il suffit de décoder le numéro de série gravé au dos de votre calculatrice (juste au-dessus du compartiment des piles.

  Celui-ci se décompose de la manière suivante:

  - Les deux premiers chiffres indiquent le nombre d'années écoulées depuis 1960;
  - On trouve ensuite deux chiffres représentant le numéro de la semaine concernée;
  - Puis une lettre indiquant le pays où la machine a été fabriquée (A pour Amérique, B pour Brésil, S pour Singapour...);
  - Enfin les 5 derniers chiffres indiquent le numéro d'ordre de la machine dans la semaine.

  La calculatrice dont le numéro de série est 3007A01051 est donc la 1051ème machine fabriquée la 7ème semaine de 1990 (1960+30) en Amérique...

Binaire, hexadécimal et autres barbaries...

Voici quelques notions qu'il est nécessaire de connaître parfaitement lorsque l'on désire accéder aux ressources cachées de la HP48 et de toute autre machine en général.

Notion de "base"

En mathématiques, une base est le nombre de symboles que l'on a à sa disposition pour compter.

Usuellement, nous comptons tous en base 10: les symboles à notre disposition sont les chiffres classiques "0" à "9".

Si nous désirons compter en base 4, nous devrons avoir 4 symboles à notre disposition (par exemple "0", "1", "2", et "3").

Lorsque nous comptons en base 10 (notre base habituelle), nous procédons ainsi:

- Nous partons de zéro (0);
- Pour passer au suivant nous remplaçons le chiffre le plus à droite par le symbole suivant ("0" devient "1", "1" devient "2"...);
- Lorsque le symbole le plus à droite est le dernier ("9"), nous le remplaçons par le premier ("0") et nous passons au symbole suivant pour le chiffre situé à gauche (si celui-ci était absent, on considère qu'il valait "0").

En fait, le principe est le même dans toutes les bases, la seule différence étant le jeu de symboles utilisé... Par exemple, pour compter en base 4, nous aurons successivement:

0, 1, 2, 3, 10, 11, 12, 13, 20, 21, 22, 23, 30, 31, 32, 33, 100, 101...

Ce qui, en base 10 correspond à la séquence:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

Remarque: le nombre 102 se lit "un-zéro-deux" et non "cent-deux", diction réservée à la base dix.

Deux bases sont couramment employées en informatique, la base 2 que l'on nomme le BINAIRE et la base 16 que l'on nomme l'HEXADECIMAL.

Le binaire

Un ordinateur a une mémoire constituée par une série de circuits électroniques qui ne peuvent être que dans deux états possibles, que l'on symbolise par 1 ou 0 (il y a du courant ou non).

Pour examiner le contenu de sa mémoire à un endroit précis l'ordinateur regarde s'il y a du courant ou non à cet endroit.

Comme seuls deux états sont possibles, toute la science des ordinateurs repose sur des calculs en base 2. On appelle ce type de calcul, du calcul binaire ou booléen (du nom du mathématicien Georges Boole qui developpa cette logique à deux états en 1846).

Dans cette base on compte donc de la manière suivante:

0, 1, 10, 11, 100, 101, 110, 111, 1000...

Cette notion conduit à une autre: celle de bit...

Le bit

On appel BIT une unité binaire qui peut valoir 0 ou 1, ce qui correspond donc à l'unité d'information élémentaire de l'ordinateur...

Souvent, ces bits sont groupés ensembles: quelquefois par quatre (pour former des quartets), le plus souvent par huit (pour former des octets).

Attention: dans ces groupes, l'ordre des bits compte...

Les quartets

La HP48 groupe les bits par paquets de 4 . Un tel paquet s'appelle un QUARTET (la traduction anglaise de ce terme est "nibble").

Il y a 16 quartets possibles: 0000, 0001, 0010, 0011, 0100, 0101, 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111.

Les octets

D'autres ordinateurs travaillent sur des paquets de 8 BITS, c'est-à-dire sur des octets. Il y a 256 octets possibles:

00000000, 00000001, 00000010... 11111110, 11111111

Comme vous pouvez le constater, la base binaire n'est pas très commode à manier, puisqu'elle conduit à manipuler des nombres très longs...

Il faut donc utiliser une base comportant plus de symboles. Comme l'unité élémentaire (le bit) conduit à utiliser la base 2, il est intéressant d'utiliser une base multiple de ce nombre...

En l'occurrence, c'est la base 16, ou hexadécimal, qui a été choisie...

L'hexadécimal

L'hexadécimal est le nom de la base 16, nous avons donc besoin de 16 symboles pour compter dans cette base...

Il n'y a donc pas assez des dix symboles couramment employés: Il faut donc en rajouter 6 qui sont habituellement "A", "B", "C", "D", "E" et "F".

Bien sur, les symboles importent peu et vous pouvez construire toutes les mathématiques sur un ensemble de symboles quelconques.

Par exemple l'ensemble { 6 è $ } où "6", "é" et "$" sont les trois symboles de votre base, peut parfaitement faire l'affaire. Vous comptez alors en base trois et fondez de nouvelles mathématiques ainsi:

6, é, $, é6, éé, é$, $6, $é, $$, é66, é6é, é6$, éé6, ééé, éé$...

Cela sera peut-être clair pour vous, mais pour les autres... C'est pourquoi il vaut mieux accepter les conventions existantes et utiliser les 6 symboles supplémentaires "A", "B", "C", "D", "E" et "F" comme tout le monde ! Avec ces symboles nous comptons donc ainsi:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11...19, 1A, 1B, 1C...

Un quartet peut alors prendre les valeurs:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F

Un octet:

00, 01, 02, 03, 04... 0A, 0B, 0C, 0D, 0E, 0F, 10... FE et FF

Ce qui est plus facile à manipuler que des nombres composés uniquement de "0" et de "1" !

Correspondance entre les bases

Le programme ci-dessous génère une table de convertionentre les bases binaire, décimale et hexadécimale. Cette table comprend les états 0 à 255, les plus utiles au programmeur... Chaque ligne comprend, dans cet ordre, les représentations binaire, décimale et hexadécimale d'un même nombre...

```
CONV (# A709h)
«
  "␣" 0 255
  FOR X
    X 1 DISP X R→B SWAP BIN OVER →STR 3 OVER SIZE 1 - SUB
    "       " SWAP + DUP SIZE 7 - 999 SUB + DEC OVER →STR 3
    OVER SIZE 1 - SUB "   " SWAP + DUP SIZE 3 - 999 SUB +
    HEX SWAP →STR 3 OVER SIZE 1 - SUB "  " SWAP + DUP SIZE 2
    - 999 SUB + "␣" +
  NEXT
»
```

Routines utiles

Voici quelques routines en langage-machine situées en mémoire morte qui réalisent des fonctions utiles, et permettent d'alléger les programmes assembleur:

- SAVE_REG (# 0679Bh) réalise la sauvegarde des registres D0, D1, B et D dans une zone particulière de la mémoire vive;
- LOAD_REG (# 067D2h) récupère les valeurs des registres sauvées par SAVE_REG;
- TRDN (# 0670Ch) copie C champ A quartets pointés par D0 à l'adresse contenue dans D1 (adresses de débuts des deux zones). Cette routine est à utiliser si D1 < D0 (transfert vers le bas);
- TRUP (# 066B9h) copie C champ A quartets pointés par D0 à l'adresse contenue dans D1 (adresses de fins des deux zones). Cette routine est à utiliser si D1 > D0 (transfert vers le haut);
- RES_ROOM (# 039BEh) réserve C champ A quartets. L'adresse de la zone ainsi réservée sera stockée dans D0. Si la mémoire libre est trop faible pour réaliser la réservation, il y aura nettoyage de la mémoire (garbage collector). Si ce nettoyage n'est pas suffisant, il y aura arrêt du programme et émission d'un message d'erreur;
- GARB_COLL (# 0613Eh) nettoie la mémoire de la HP48 en détruisant tous les objets inutiles (objets de la ram des objets temporaires non référencés);
- RES_STR (# 05B7Dh) réserve une chaîne de caractères dont la longueur (en quartets) est dans C champ A. Cette routine renvoie l'adresse de la chaîne dans R0 champ A et l'adresse de son contenu dans D0. Si la mémoire libre est trop faible pour réaliser la réservation, il y aura nettoyage de la mémoire (garbage collector). Si ce nettoyage n'est pas suffisant, il y aura arrêt du programme et émission d'un message d'erreur;
- PUSH_R0 (# 06537h) place dans la pile la valeur du champ A de R0 sous forme d'un entier-système. ATTENTION: les registres D1 et D doivent avoir été sauvés par appel à SAVE_REG;

- PUSH_R0_R1 (# 06529h) place dans la pile les valeur des champs A de R0 et R1 sous forme de deux entiers-système. R0 sera placé au niveau 2, R1 au niveau 1. ATTENTION: les registres D1 et D doivent avoir été sauvés par appel à SAVE_REG;
- POP_C (# 06641h) lit la valeur d'un system binary dans la pile. La valeur est stockée dans le champ A de C et l'entier est ôté de la pile. ATTENTION: les valeurs des registres D1 et D doivent être celle du système (pointeur de pile et mémoire libre). Si tel n'est pas le cas, il convient de les restaurer (par appel à LOAD_REG si la sauvegarde a été faite par SAVE_REG). De plus leur valeur est modifiée par cette routine (l'objet au niveau 1 est enlevé);
- POP_C_A (# 03F5Dh) prend les valeurs de deux entiers courts dans la pile. Comme pour la routine précédente, les deux entiers sont ôtés et les valeurs de D1 et D sont modifiées. Le champ A de C contiendra la valeur de l'entier au niveau 1, le champ A de A, celle de l'entier au niveau 2;
- DIV5 (# 06A8Eh) divise le contenu de C champ A par 5. Cette routine utilise les 10 premiers quartets des registres A C et D. Il s'agit en fait d'une multiplication de C par 3355444 suivie d'une division par 16777216, ce qui correspond, de manière approchée, à une division par 5;
- MULT (# 03991h) réalise la multiplication des champs A de A et C et place le résultat dans B champ A;
- BEEP (# 017A6h) émet un son dont la fréquence se trouve dans D champ A et la durée, en millisecondes, dans C champ A. Cette routine prend en compte le drapeau -56;
- ERROR (# 05023h) affiche le message d'erreur dont le numéro est dans A champ A. ATTENTION: cette routine doit être appelée par saut et non par appel de sous-programme et après appel à LOAD_REG. Elle provoque l'arrêt du programme en cours d'exécution;
- STOP (# 10FDBh) appelée par saut, arrête le programme en cours d'exécution. Il s'agit en fait de l'erreur #123h qui n'est pas récupérable par IFERR et qui provoque un retour au mode interactif. Cette sortie doit être précédée d'un appel à LOAD_REG.

Liste exhaustive des messages d'erreurs

Voici la liste exhaustive des messages d'erreurs que la HP48 (sans librairie supplémentaires) est susceptible d'émettre.

Ils sont classés par ordre de leur code (donné en valeur décimale et hexadécimale).

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

Décimal	Hexadécimal	Messages
257	# 101h	"No Room to Save Stack"
258	# 102h	"Can't Edit Null Char."
259	# 103h	"Invalid User Function"
260	# 104h	"No Current Equation"
262	# 106h	"Invalid Syntax"
263	# 107h	"Real Number"
264	# 108h	"Complex Number"
265	# 109h	"String"
266	# 10Ah	"Real Array"
267	# 10Bh	"Complex Array"
268	# 10Ch	"List"
269	# 10Dh	"Global Name"
270	# 10Eh	"Local Name"
271	# 10Fh	"Program"
272	# 110h	"Algebraic"
273	# 111h	"Binary Integer"
274	# 112h	"Graphic"
275	# 113h	"Tagged"
276	# 114h	"Unit"
277	# 115h	"XLIB Name"
278	# 116h	"Directory"
279	# 117h	"Library"
280	# 118h	"Backup"
281	# 119h	"Function"
282	# 11Ah	"Command"
283	# 11Bh	"System Binary"
284	# 11Ch	"Long Real"
285	# 11Dh	"Long Complex"
286	# 11Eh	"Linked Array"
287	# 11Fh	"Character"
288	# 120h	"Code"

Décimal	Hexadécimal	Messages
289	# 121h	"Library Data"
290	# 122h	"External"
292	# 124h	"LAST STACK Disabled"
293	# 125h	"LAST CMD Disabled"
294	# 126h	"HALT Not Allowed"
295	# 127h	"Array"
296	# 128h	"Wrong Argument Count"
297	# 129h	"Circular Reference"
298	# 12Ah	"Directory Not Allowed"
299	# 12Bh	"Non-Empty Directory"
300	# 12Ch	"Invalid Definition"
301	# 12Dh	"Missing Library"
302	# 12Eh	"Invalid PPAR"
303	# 12Fh	"Non-Real Result"
304	# 130h	"Unable to Isolate"
305	# 131h	"No Room to Show Stack"
306	# 132h	"Warning:"
307	# 133h	"Error:"
308	# 134h	"Purge?"
309	# 135h	"Out of Memory"
310	# 136h	"Stack"
311	# 137h	"Last Stack"
312	# 138h	"Last Commands"
313	# 139h	"Key Assignments"
314	# 13Ah	"Alarms"
315	# 13Bh	"Last Arguments"
316	# 13Ch	"Name Conflict"
317	# 13Dh	"Command Line"

Décimal	Hexadécimal	Messages
513	# 201h	"Too Few Arguments"
514	# 202h	"Bad Argument Type"
515	# 203h	"Bad Argument Value"
516	# 204h	"Undefined Name"
517	# 205h	"LASTARG Disabled"
518	# 206h	"IncompleteSubexpression"
519	# 207h	"Implicit () off"
520	# 208h	"Implicit () on"
769	# 301h	"Positive Underflow"
770	# 302h	"Negative Underflow"
771	# 303h	"Overflow"
772	# 304h	"Undefined Result"
773	# 305h	"Infinite Result"
1281	# 501h	"Invalid Dimension"
1282	# 502h	"Invalid Array Element"
1283	# 503h	"Deleting Row"
1284	# 504h	"Deleting Column"
1285	# 505h	"Inserting Row"
1286	# 506h	"Inserting Column"

Décimal	Hexadécimal	Messages
1537	# 601h	"Invalid ∑Data"
1538	# 602h	"Nonexistent ∑DAT"
1539	# 603h	"Insufficient ∑Data"
1540	# 604h	"Invalid ∑PAR"
1541	# 605h	"Invalid ∑Data LN(Neg)"
1542	# 606h	"Invalid ∑Data LN(0)"
1543	# 607h	"Invalid EQ"
1544	# 608h	"Current equation:"
1545	# 609h	"No current equation."
1546	# 60Ah	"Enter eqn, press NEW"
1547	# 60Bh	"Name the equation,press ENTER"
1548	# 60Ch	"Select plot type"
1549	# 60Dh	"Empty catalog"
1550	# 60Eh	"undefined"
1551	# 60Fh	"No stat data to plot"
1552	# 610h	"Autoscaling"
1553	# 611h	"Solving for "
1554	# 612h	"No current data. Enter"
1555	# 613h	"data point, press ∑+"
1556	# 614h	"Select a model"
1557	# 615h	"No alarms pending."
1558	# 616h	"Press ALRM to create"
1559	# 617h	"Next alarm:"
1560	# 618h	"Past due alarm:"
1561	# 619h	"Acknowledged"
1562	# 61Ah	"Enter alarm, press SET"
1563	# 61Bh	"Select repeat interval"
1564	# 61Ch	" I/O setup menu"
1565	# 61Dh	"Plot type: "
1566	# 61Eh	""""
1567	# 61Fh	" (OFF SCREEN)"

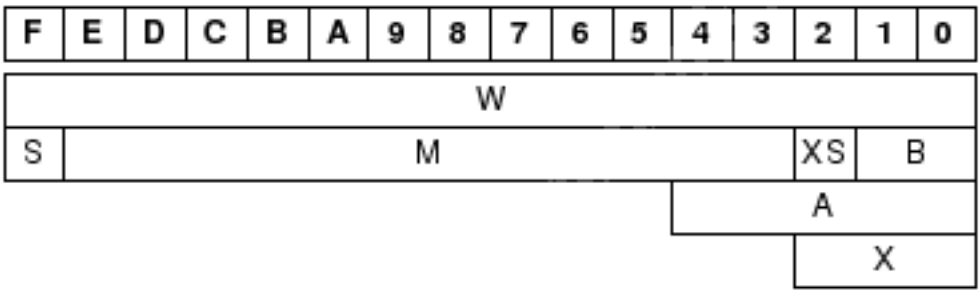
Décimal	Hexadécimal	Messages
1567	# 620h	"Invalid PTYPE"
1569	# 621h	"Name the stat data,press ENTER"
1570	# 622h	"Enter value (zoom outif >1), press ENTER"
1571	# 623h	"Copied to stack"
1572	# 624h	"x axis zoom w/AUTO."
1573	# 625h	"x axis zoom."
1574	# 626h	"y axis zoom."
1575	# 627h	"x and y axis zoom."
1576	# 628h	"IR/wire: "
1577	# 629h	"ASCII/binary: "
1578	# 62Ah	"baud: "
1579	# 62Bh	"parity: "
1580	# 62Ch	"checksum type: "
1581	# 62Dh	"translate code:"
1582	# 62Eh	"Enter matrix, then NEW"
2561	# A01h	"Bad Guess(es)"
2652	# A02h	"Constant?"
2563	# A03h	"Interrupted"
2564	# A04h	"Zero"
2565	# A05h	"Sign Reversal"
2566	# A06h	"Extremum"
2567	# B01h	"Invalid Unit"
2568	# B02h	"Inconsistent Units"

Décimal	Hexadécimal	Messages
3073	# C01h	"Bad Packet Block Check"
3074	# C02h	"Timeout"
3075	# C03h	"Receive Error"
3076	# C04h	"Receive Buffer Overrun"
3077	# C05h	"Parity Error"
3078	# C06h	"Transfer Failed"
3079	# C07h	"Protocol Error"
3080	# C08h	"Invalid Server Cmd."
3081	# C09h	"Port Closed"
3082	# C0Ah	"Connecting"
3083	# C0Bh	"Retry #"
3084	# C0Ch	"Awaiting Server Cmd."
3085	# C0Dh	"Sending "
3086	# C0Eh	"Receiving "
3087	# C0Fh	"Object Discarded"
3088	# C10h	"Packet #"
3089	# C11h	"Processing Command"
3090	# C12h	"Invalid IOPAR"
3091	# C13h	"Invalid PRTPAR"
3092	# C14h	"Low Battery"
3093	# C15h	"Empty Stack"
3094	# C16h	"Row "
3095	# C17h	"Invalid Name"
3329	# D01h	"Invalid Date"
3330	# D02h	"Invalid Time"
3331	# D03h	"Invalid Repeat"
3332	# D04h	"Nonexistent Alarm"
458752	#70000h	Dernier message utilisateur (message DOERR)

Instructions du LM

Les instructions des pages suivantes sont données dans l'ordre des codes. Ceci complète le manuel HDS de HP qui les donne dans l'ordre alphabétique (en particulier cette liste est très utile pour désassembler des programmes en Rom...). Pour des raisons de commodités d'usage, la totalité des instructions se trouve décrite sur deux pages en vis-à-vis, de manière à ce que le lecteur ait accès à la totalité des instructions sans avoir de page à tourner...

Les explications détaillées de ces instructions se trouvent dans la partie 2 (Le langage machine), chapitres 2 et 3. Concernant les champs des registres, voici les deux tableaux récapitulatifs déjà vus:



Champ	a	f	b
P	0	0	8
WP	1	1	9
XS	2	2	A
X	3	3	B
S	4	4	C
M	5	5	D
B	6	6	E
W	7	7	F
A		F	

00	RTNSXM	
01	RTN	
02	RTNSC	
03	RTNCC	
04	SETHEX	
05	SETDEC	
06	RSTR=C	
07	C=RSTR	
08	CLRST	
09	C=ST	
0A	ST=C	
0B	CSTEX	
0C	P=P+1	
0D	P=P-1	
0Ef0	A=A&B	f
0Ef1	B=B&C	f
0Ef2	C=C&A	f
0Ef3	D=D&C	f
0Ef4	B=B&A	f
0Ef5	C=C&B	f
0Ef6	A=A&C	f
0Ef7	C=C&D	f
0Ef8	A=A|B	f
0Ef9	B=B|C	f
0EfA	C=C|A	f
0EfB	D=D|C	f
0EfC	B=B|A	f
0EfD	C=C|B	f
0EfE	A=A|C	f
0EfF	C=C|D	f
0F	RTI	

100	R0=A	
101	R1=A	
102	R2=A	
103	R3=A	
104	R4=A	
108	R0=C	
109	R1=C	
10A	R2=C	
10B	R3=C	
10C	R4=C	
110	A=R0	
111	A=R1	
112	A=R2	
113	A=R3	
114	A=R4	
118	C=R0	
119	C=R1	
11A	C=R2	
11B	C=R3	
11C	C=R4	
120	AR0EX	
121	AR1EX	
122	AR2EX	
123	AR3EX	
124	AR4EX	
128	CR0EX	
129	CR1EX	
12A	CR2EX	
12B	CR3EX	
12C	CR4EX	
130	D0=A	
131	D1=A	
132	AD0EX	
133	AD1EX	
134	D0=C	
135	D1=C	
136	CD0EX	
137	CD1EX	
138	D0=AS	
139	D1=AS	
13A	AD0XS	
13B	AD1XS	
13C	D0=CS	
13D	D1=CS	
13E	CD0XS	
13F	CD1XS	
140	DAT0=A	A
141	DAT1=A	A
142	A=DAT0	A
143	A=DAT1	A
144	DAT0=C	A
145	DAT1=C	A
146	C=DAT0	A
147	C=DAT1	A
148	DAT0=A	B
149	DAT1=A	B
14A	A=DAT0	B
14B	A=DAT1	B
14C	DAT0=C	B
14D	DAT1=C	B
14E	C=DAT0	B
14F	C=DAT1	B
150m	DAT0=A	m
151m	DAT1=A	m
152m	A=DAT0	m
153m	A=DAT1	m
154m	DAT0=C	m
155m	DAT1=C	m
156m	C=DAT0	m
157m	C=DAT1	m
158x	DAT0=A	x+1
159x	DAT1=A	x+1
15Ax	A=DAT0	x+1
15Bx	A=DAT1	x+1
15Cx	DAT0=C	x+1
15Dx	DAT1=C	x+1
15Ex	C=DAT0	x+1
15Fx	C=DAT1	x+1
16n	D0=D0+	n+1
17n	D1=D1+	n+1
18n	D0=D0-	n+1
19pq	D0=|2|	qp
1Apqrs	D0=|4|	srqp
1Bpqrst	D0=|5|	tsrqp
1Cn	D1=D1-	n+1
1Dpq	D1=|2|	qp
1Epqrs	D1=|4|	srqp
1Fpqrst	D1=|5|	tsrqp

2n	P=	n

3xh0.hx	LCHEX	@hx.h0

400	RTNSC	
420	NOP3	
4yz	GOC	zy

500	RTNNC	
5yz	GONC	zy

6300	NOP4	
64000	NOP5	
6yzt	GOTO	tzy

7yzt	GOSUB	tzy

800	OUT=CS	
801	OUT=C	
802	A=IN	
803	C=IN	
804	UNCNFG	
805	CONFIG	
806	C=ID	
807	SHUTDN	
8080	INTON	
8081.0	RSI	
8082xh0.hx	LAHEX	@hx.h0
8083	BUSCB	
8084d	ABIT=0	d
8085d	ABIT=1	d
8086d	?ABIT=0	d
8087d	?ABIT=1	d
8088d	CBIT=0	d
8089d	CBIT=1	d
808Ad	?CBIT=0	d
808Bd	?CBIT=1	d
808C	PC=|A|	
808D	BUSCD	
808E	PC=|C|	
808F	INTOFF	
809	C+P+1	
80A	RESET	
80B	BUSCC	
80Cx	C=P	x
80Dx	P=C	x
80E	SREQ?	
80Fx	CPEX	x
810	ASLC	
811	BS	
812	CS	
813	DS	
814	AS	
815	BS	
816	CS	
817	DS	
818f0x	A=	
818f1x	B=	
818f2x	C=	
818f3x	D=	
818f8x	A=	
818f9x	B=	
818fAx	C=	
818fBx	D=	
819f0	AS	
819f1	BS	
819f2	CS	
819f3	DS	
81AE00	R0	
81AE01	R1	
81AE02	R2	
81AE03	R3	
81AE04	R4	
81AE08	R0	
81AE09	R1	
81AE0A	R2	
81AE0B	R3	
81AE0C	R4	
81AE10	A=	
81AE11	A=	
81AE12	A=	
81AE13	A=	
81AE14	A=	
81AE18	C=	
81AE19	C=	
81AE1A	C=	
81AE1B	C=	
81AE1C	C=	
81AE20	AR	
81AE21	AR	
81AE22	AR	
81AE23	AR	
81AE24	AR	
81AE28	CR	
81AE29	CR	
81AE2A	CR	
81AE2B	CR	
81AE2C	CR	
81B2	PC	
81B3	PC	
81B4	A=	
81B5	C=	
81B6	AP	
81B7	CP	
81C	AS	
81D	BS	
81E	CS	
81F	DS	
821	XM	
822	SB	
824	SR	
828	MP	
82F	CL	
831	XS	
832	TS	
834	IS	
838	IM	
84d	ST	
85d	ST	
86d	TS	
87d	TS	
88n	IP	
89n	IP	
8A0	IE	
8A1	IC	
8A2	IA	
8A3	IC	
8A4	IE	
8A5	IC	
8A6	IA	
8A7	ID	
8A8	IA	
8A9	IE	
8AA	IC	
8AB	ID	

A≠0	A
B≠0	A
C≠0	A
D≠0	A
A>B	A
B>C	A
C>A	A
D>C	A
A<B	A
B<C	A
C<A	A
D<C	A
A≥B	A
B≥C	A
C≥A	A
D≥C	A
A≤B	A
B≤C	A
C≤A	A
D≤C	A
OLONG	srop
OVLNG	tsrop
OSUBL	srop
OSBVL	tsrop

A=B	a
B=C	a
C=A	a
C=D	a
A≠B	a
B≠C	a
C≠A	a
D≠C	a
A=0	a
B=0	a
C=0	a
D=0	a
A≠0	a
B≠0	a
C≠0	a
D≠0	a
A>B	b
B>C	b
C>A	b
D>C	b
A<B	b
B<C	b
C<A	b
D<C	b
A≥B	b
B≥C	b
C≥A	b
D≥C	b
A≤B	b
B≤C	b
C≤A	b
D≤C	b

=A+B	a
=B+C	a
=C+A	a
=D+C	a
=A+A	a
=B+B	a
=C+C	a
=D+D	a
=B+A	a
=C+B	a
=A+C	a
=C+D	a
=A-1	a
=B-1	a
=C-1	a
=D-1	a
=0	b
=0	b
=0	b
=0	b
=B	b
=C	b
=A	b
=C	b
=A	b
=B	b
=C	b

AbB	C=D	b
AbC	ABEX	b
AbD	BCEX	b
AbE	ACEX	b
AbF	CDEX	b

Ba0	A=A-B	a
Ba1	B=B-C	a
Ba2	C=C-A	a
Ba3	D=D-C	a
Ba4	A=A+1	a
Ba5	B=B+1	a
Ba6	C=C+1	a
Ba7	D=D+1	a
Ba8	B=B-A	a
Ba9	C=C-B	a
BaA	A=A-C	a
BaB	C=C-D	a
BaC	A=B-A	a
BaD	B=C-B	a
BaE	C=A-C	a
BaF	D=C-D	a
Bb0	ASL	b
Bb1	BSL	b
Bb2	CSL	b
Bb3	DSL	b
Bb4	ASR	b
Bb5	BSR	b
Bb6	CSR	b
Bb7	DSR	b
Bb8	A=-A	b
Bb9	B=-B	b
BbA	C=-C	b
BbB	D=-D	b
BbC	A=-A-1	b
BbD	B=-B-1	b
BbE	C=-C-1	b
BbF	D=-D-1	b

C0	A=A+B	A
C1	B=B+C	A
C2	C=C+A	A
C3	D=D+C	A
C4	A=A+A	A
C5	B=B+B	A
C6	C=C+C	A
C7	D=D+D	A
C8	B=B+A	A
C9	C=C+B	A
CA	A=A+C	A
CB	C=C+D	A
CC	A=A-1	A
CD	B=B-1	A
CE	C=C-1	A
CF	D=D-1	A

D0	A=0	A
D1	B=0	A
D2	C=0	A
D3	D=0	A
D4	A=B	A
D5	B=C	A
D6	C=A	A
D7	D=C	A
D8	B=A	A
D9	C=B	A
DA	A=C	A
DB	C=D	A
DC	ABEX	A
DD	BCEX	A
DE	ACEX	A
DF	CDEX	A

E0	A=A-B	A
E1	B=B-C	A
E2	C=C-A	A
E3	D=D-C	A
E4	A=A+1	A
E5	B=B+1	A
E6	C=C+1	A
E7	D=D+1	A
E8	B=B-A	A
E9	C=C-B	A
EA	A=A-C	A
EB	C=C-D	A

EC	A=B-A	A
ED	B=C-B	A
EE	C=A-C	A
EF	D=C-D	A

F0	ASL	A
F1	BSL	A
F2	CSL	A
F3	DSL	A
F4	ASR	A
F5	BSR	A
F6	CSR	A
F7	DSR	A
F8	A=-A	A
F9	B=-B	A
FA	C=-C	A
FB	D=-D	A
FC	A=-A-1	A
FD	B=-B-1	A
FE	C=-C-1	A
FF	D=-D-1	A

Liste des commandes par leur nom

Voici la liste de toutes les commandes de la HP48 classées par leur nom (dans l'ordre du manuel de référence HP). Cette liste est divisée en deux parties correspondant aux deux librairies (respectivement #002h et #700h).

Certaines commandes n'ont pas de nom. Elles ne réalisent aucune fonction et sont probablement réservées pour un usage futur par HP.

Chaque ligne est constituée, dans cet ordre, par le nom de la fonction, son numéro de commande en hexadécimal puis en décimal et son adresse d'appel (par SYSEVAL).

Par exemple ABS est la commande #03Dh (61) et peut être appelé par #1AA1Fh SYSEVAL.

Ces adresses peuvent être utilisées dans des objets programme. Par exemple on peut réaliser une fonction dupliquant 3 fois l'objet au niveau 1 (objet → objet objet objet objet) est assemblant l'objet programme contenant les instructions DUP DUP2. On trouve dans le tableau:

```
|DUP          #10Dh 269 #1FB87h|
|DUP2         #10Eh 270 #1FBA2h|
```

Les deux commandes ont pour adresse respective #1FB87h et #1FBA2h. L'objet-programme les contenant a #02D9Dh comme prologue et #0312B comme épilogue. L'objet désiré se code donc:

"D9D2078BF12ABF1B2130"

Ce qui économise 10 quartets par rapport au programme contenant les deux délimiteurs (« et ») tout en réalisant exactement la même fonction...

Ces tables sont aussi utiles au lecteur désirant désassembler une commande particulière (elles correspondent à des objets programme situés en mémoire morte).

La première d'entre elles correspond à la librairie #002h.

ABS	#03Dh	61	#1AA1Fh
ACK	#015h	21	#1987Eh
ACKALL	#014h	20	#19863h
ACOS	#058h	88	#1B72Fh
ACOSH	#05Bh	91	#1B830h
ALOG	#060h	96	#1BA3Dh
AND	#0E5h	229	#1E783h
APPLY	#102h	258	#1F55Dh
APPLY	#103h	259	#1F55Ch
ARC	#0D8h	216	#1E5D2h
ARCHIVE	#160h	352	#2125Ah
ARG	#04Dh	77	#1B2DBh
ARRY→	#0ABh	171	#1D092h
→ARRY	#0AAh	170	#1D009h
ASIN	#057h	87	#1B6A4h
ASINH	#05Ah	90	#1B7EBh
ASN	#17Bh	379	#224F4h
ASR	#000h	0	#19578h
ATAN	#059h	89	#1B79Ch
ATANH	#05Ch	92	#1B8A2h
ATTACH	#165h	357	#21448h
AUTO	#0C0h	192	#1E1ABh
AXES	#0BAh	186	#1E0BEh
BAR	#0E3h	227	#1E741h
BARPLOT	#13Ch	316	#20133h
BAUD	#172h	370	#2200Ch
B→R	#00Ah	10	#196BBh
BEEP	#034h	52	#1A5C4h
BESTFIT	#143h	323	#2025Eh
BIN	#090h	144	#1C559h
BINS	#138h	315	#2010Eh
BLANK	#0D1h	209	#1E416h
BOX	#0D0h	208	#1E3ECh
BUFLEN	#176h	374	#22087h
BYTES	#026h	38	#1A1D9h
C→PX	#0C7h	199	#1E29Ah
C→R	#09Fh	159	#1C98Eh
CEIL	#068h	104	#1BC0Fh
CENTR	#0BBh	187	#1E0E8h
CF	#084h	132	#1C2D5h
%CH	#07Eh	126	#1C149h
CHR	#0A5h	165	#1CB66h
CKSM	#171h	369	#21FECh
CLEAR	#11Ah	282	#1FCEBh
CLKADJ	#018h	24	#198DEh
CLLCD	#038h	56	#1A858h
CLOSEIO	#16Ah	362	#21ED5h
CLΣ	#11Ch	284	#1FD2Bh
CLUSR	#15Ah	346	#210FCh
CLVAR	#15Ah	346	#210FCh
CNRM	#077h	119	#1BFBEh
COLCT	#14Dh	333	#20A15h
COLΣ	#138h	312	#2009Ah

COMB	#081h	129	#1C1F6h
CON	#0ADh	173	#1D186h
CONIC	#0DDh	221	#1E681h
CONJ	#03Eh	62	#1AA6Eh
CONT	#03Ah	58	#1A8BBh
CONVERT	#00Bh	11	#196DBh
CORR	#121h	289	#1FDC1h
COS	#052h	82	#1B505h
COSH	#055h	85	#1B606h
COV	#122h	290	#1FDDCh
CR	#0F3h	243	#1EEA4h
CRDIR	#020h	32	#1A105h
CROSS	#07Ah	122	#1C01Eh
DATE	#011h	17	#19812h
→DATE	#016h	22	#1989Eh
DATE→	#01Fh	31	#199D2h
D→R	#070h	112	#1BEC8h
DDAYS	#01Eh	30	#199B2h
DEC	#091h	145	#1C574h
DECR	#14Ch	332	#209AAh
DEFINE	#156h	342	#20D65h
DEG	#087h	135	#1C399h
DELALARM	#01Ch	28	#19972h
DELAY	#0F5h	245	#1EF43h
DELKEYS	#17Dh	381	#22548h
DEPND	#0C4h	196	#1E22Bh
DEPTH	#114h	276	#1FC44h
DET	#078h	120	#1BFDEh
DETACH	#166h	358	#2147Ch
DISP	#032h	50	#1A584h
DOERR	#02Ah	42	#1A339h
DOT	#079h	121	#1BFFEh
DRAW	#0BFh	191	#1E190h
DRAX	#0C1h	193	#1E1C6h
DROP	#110h	272	#1FBD8h
DROP2	#111h	273	#1FBF3h
DROPN	#115h	277	#1FC64h
DTAG	#180h	384	#22633h
DUP	#10Dh	269	#1FB87h
DUP2	#10Eh	270	#1FBA2h
DUPN	#116h	278	#1FC7Fh
ENG	#08Ch	140	#1C452h
EQ→	#0A8h	168	#1CEE3h
ERASE	#0C5h	197	#1E25Fh
ERR0	#02Bh	43	#1A36Dh
ERRM	#02Dh	45	#1A3A3h
ERRN	#02Ch	44	#1A388h
EVAL	#02Eh	46	#1A3BEh
EXP	#05Dh	93	#1B905h
EXPAN	#14Eh	334	#20A49h
EXPFIT	#141h	321	#201FBh
EXPM	#062h	98	#1BAC2h
e	#042h	66	#1AB23h

FACT	#064h	100	#1BB41h
FC?	#086h	134	#1C360h
FC?C	#08Fh	143	#1C520h
FINDALARM	#01Bh	27	#19948h
FINISH	#16Fh	367	#21FB6h
FIX	#08Ah	138	#1C3EAh
FLOOR	#067h	103	#1BBD9h
FP	#066h	102	#1BBA3h
FREE	#163h	355	#213D1h
FREEZE	#033h	51	#1A5A4h
FS?	#085h	133	#1C313h
FS?C	#08Eh	142	#1C4A1h
FUNCTION	#0DCh	220	#1E661h
GET	#0B2h	178	#1D7C6h
GETI	#0B3h	179	#1D8C7h
GOR	#0D3h	211	#1E456h
GRAD	#089h	137	#1C3CFh
GRAPH	#0C8h	200	#1E2BAh
→GROB	#0D7h	215	#1E5ADh
GXOR	#0D4h	212	#1E4E4h
→H	#0BDh	189	#1E150h
HEX	#092h	146	#1C58Fh
HISTOGRAM	#0E2h	226	#1E721h
HISTPLOT	#13Dh	317	#20167h
HMS→	#074h	116	#1BF5Eh
HMS-	#075h	117	#1BF7Eh
HMS+	#073h	115	#1BF3Eh
→HMS	#072h	114	#1BF1Eh
HOME	#022h	34	#1A140h
IDN	#0AEh	174	#1D2DCh
IFT	#030h	48	#1A4CDh
IFTE	#02Fh	47	#1A3FEh
IM	#09Bh	155	#1C819h
INCR	#14Bh	331	#208F4h
INDEP	#0B7h	183	#1E04Ah
INPUT	#17Ah	378	#224CAh
INV	#04Ch	76	#1B278h
IP	#065h	101	#1BB6Dh
ISOL	#150h	336	#20A93h
i	#043h	67	#1AB45h
KERRM	#175h	373	#2206Ch
KEY	#039h	57	#1A873h
KGET	#16Ch	364	#21F24h
KILL	#028h	40	#1A303h
LABEL	#0C9h	201	#1E2D5h
LAST	#036h	54	#1A604h
LASTARG	#036h	54	#1A604h
LCD→	#0D5h	213	#1E572h
→LCD	#0D6h	214	#1E58Dh
LIBS	#164h	356	#2142Dh
LINE	#0CEh	206	#1E398h
ΣLINE	#13Ah	314	#200F3h
LINFIT	#13Fh	319	#201B1h
LIST→	#09Eh	158	#1C95Ah

→LIST	#098h	152	#1C783h
LN	#05Eh	94	#1B94Fh
LNP1	#061h	97	#1BA8Ch
LOG	#05Fh	95	#1B9C6h
LOGFIT	#140h	320	#201D6h
LR	#12Eh	302	#1FF20h
MANT	#06Fh	111	#1BE9Ch
↑MATCH	#109h	265	#1FA59h
↓MATCH	#10Ah	266	#1FA8Dh
MAX	#06Ah	106	#1BC71h
MAXΣ	#128h	296	#1FE7Eh
MAXR	#040h	64	#1AADFh
MEAN	#129h	297	#1FE99h
MEM	#158h	344	#20FAAh
MENU	#15Ch	348	#21196h
MERGE	#162h	354	#2137Fh
MIN	#06Bh	107	#1BCE3h
MINΣ	#12Ah	298	#1FEB4h
MINR	#041h	65	#1AB01h
MOD	#06Eh	110	#1BE4Dh
NEG	#03Ch	60	#1A995h
NEWOB	#027h	39	#1A2BCh
NOT	#0E7h	231	#1E88Fh
NΣ	#120h	288	#1FDA6h
NUM	#0A4h	164	#1CB46h
→NUM	#035h	53	#1A5E4h
OBJ→	#0A9h	169	#1CF7Bh
OCT	#093h	147	#1C5AAh
OFF	#029h	41	#1A31Eh
OLDPRT	#0EFh	239	#1EE38h
OPENIO	#169h	361	#21EB5h
OR	#0E6h	230	#1E809h
ORDER	#159h	345	#20FD9h
OVER	#113h	275	#1FC29h
PARAMETRIC	#0DFh	223	#1E6C1h
PARITY	#173h	371	#2202Ch
PATH	#021h	33	#1A125h
PDIM	#0C3h	195	#1E201h
PERM	#082h	130	#1C236h
PGDIR	#15Fh	351	#2123Ah
PICK	#117h	279	#1FC9Ah
PICT	#0D2h	210	#1E436h
PIX?	#0CDh	205	#1E36Eh
PIXOFF	#0CCh	204	#1E344h
PIXON	#0CBh	203	#1E31Ah
PKT	#179h	377	#220DDh
PMAX	#0B9h	185	#1E09Eh
PMIN	#0B8h	184	#1E07Eh
POLAR	#0DEh	222	#1E6A1h
POS	#0A1h	161	#1CAB4h
PR1	#0F0h	240	#1EE53h
PREDV	#12Fh	303	#1FF7Ah
PREDX	#131h	305	#1FFBAh
PREDY	#130h	304	#1FF9Ah

PRLCD	#0F6h	246	#1EF63h
PRST	#0F2h	242	#1EE89h
PRSTC	#0F1h	241	#1EE6Eh
PRVAR	#0F4h	244	#1EEBFh
PURGE	#157h	343	#20EFEh
PUT	#0B0h	176	#1D407h
PUTI	#0B1h	177	#1D5DFh
PVARS	#15Eh	350	#211FCh
PVIEW	#0CAh	202	#1E2F0h
PWRFIT	#142h	322	#20220h
PX→C	#0C6h	198	#1E27Ah
→Q	#107h	263	#1F9C4h
→Q□	#108h	264	#1F9E9h
QUAD	#151h	337	#20AB3h
QUOTE	#101h	257	#1F500h
RAD	#088h	136	#1C3B4h
RAND	#07Fh	127	#1C1B9h
RATIO	#10Ch	268	#1FB5Dh
R→B	#009h	9	#1969Bh
R→C	#099h	153	#1C79Eh
R→D	#071h	113	#1BEF4h
RCEQ	#0F9h	249	#1F133h
RCL	#154h	340	#20B40h
RCLALARM	#01Ah	26	#19928h
RCLF	#096h	150	#1C619h
RCLKEYS	#17Eh	382	#22586h
RCLMENU	#15Dh	349	#211E1h
RCLΣ	#11Dh	285	#1FD46h
RCWS	#095h	149	#1C5FEh
RDM	#0ACh	172	#1D0DFh
RDZ	#080h	128	#1C1D4h
RE	#09Ah	154	#1C7CAh
RECN	#16Dh	365	#21F62h
RECV	#16Eh	366	#21F96h
REPL	#09Dh	157	#1C8EAh
RES	#0BCh	188	#1E126h
RESTORE	#161h	353	#2133Ch
RL	#001h	1	#1959Bh
RLB	#002h	2	#195BBh
RND	#06Ch	108	#1BD55h
RNRM	#076h	118	#1BF9Eh
ROLL	#118h	280	#1FCB5h
ROLLD	#119h	281	#1FCD0h
ROOT	#0FBh	251	#1F16Eh
ROT	#112h	274	#1FC0Eh
RR	#003h	3	#195DBh
RRB	#004h	4	#195FBh
RSD	#07Bh	123	#1C03Eh
RULES	#14Fh	335	#20A7Dh
SAME	#0E4h	228	#1E761h
SBRK	#178h	376	#220C2h
SCALE	#0C2h	194	#1E1E1h
SCATRPLOT	#13Eh	318	#2018Ch
SCATTER	#0E1h	225	#1E701h
SCI	#08Bh	139	#1C41Eh
SCLΣ	#139h	313	#200C4h
SCONJ	#146h	326	#203CCh
SDEV	#12Bh	299	#1FECFh
SEND	#16Bh	363	#21EF0h
SERVER	#170h	368	#21FD1h
SF	#083h	131	#1C274h
SHOW	#152h	338	#20AD3h
SIGN	#04Eh	78	#1B32Ah
SIN	#051h	81	#1B4ACh
SINH	#054h	84	#1B5B7h
SINV	#144h	324	#202CEh
SIZE	#0A0h	160	#1C9B8h
SL	#005h	5	#1961Bh
SLB	#006h	6	#1963Bh
SNEG	#145h	325	#2034Dh
SQ	#050h	80	#1B426h
SR	#007h	7	#1965Bh
SRB	#008h	8	#1967Bh
SRECV	#168h	360	#21E95h
STD	#08Dh	141	#1C486h
STEQ	#0FAh	250	#1F14Eh
STIME	#177h	375	#220A2h
STO	#155h	341	#20CCDh
STO*	#14Ah	330	#20753h
STO+	#147h	327	#2044Bh
STO-	#148h	328	#20538h
STO/	#149h	329	#2060Ch
STOALARM	#019h	25	#198FEh
STOF	#097h	151	#1C67Fh
STOKEYS	#17Ch	380	#22514h
STOΣ	#11Bh	283	#1FD0Bh
STR→	#0A3h	163	#1CB26h
→STR	#0A2h	162	#1CB0Bh
STWS	#094h	148	#1C5C5h
SUB	#09Ch	156	#1C85Ch
SWAP	#10Fh	271	#1FBBDh
SYSEVAL	#031h	49	#1A52Eh
%T	#07Dh	125	#1C0D7h
→TAG	#17Fh	383	#225BEh
TAN	#053h	83	#1B55Eh
TANH	#056h	86	#1B655h
TAYLR	#153h	339	#20B20h
TEXT	#0D9h	217	#1E606h
TICKS	#012h	18	#1982Dh
TIME	#010h	16	#197F7h
→TIME	#017h	23	#198BEh
TLINE	#0CFh	207	#1E3C2h
TMENU	#15Bh	347	#2115Dh
TOT	#12Ch	300	#1FEEAh
TRANSIO	#174h	372	#2204Ch
TRN	#0AFh	175	#1D392h
TRNC	#06Dh	109	#1BDD1h
TRUTH	#0E0h	224	#1E6E1h

TSTR	#01Dh	29	#19992h
TVARS	#025h	37	#1A1AFh
TYPE	#0A6h	166	#1CB86h
UBASE	#00Eh	14	#19771h
UFACT	#00Fh	15	#197A5h
→UNIT	#00Dh	13	#1974Fh
UPDIR	#023h	35	#1A15Bh
UTPC	#134h	308	#2001Ah
UTPF	#136h	310	#2005Ah
UTPN	#135h	309	#2003Ah
UTPT	#137h	311	#2007Ah
UVAL	#00Ch	12	#19718h
VAR	#12Dh	301	#1FF05h
VARS	#024h	36	#1A194h
V→	#0B4h	180	#1DD06h
→V2	#0B5h	181	#1DE66h
→V3	#0B6h	182	#1DEC2h
VTYPE	#0A7h	167	#1CE28h
→W	#0BEh	190	#1E170h
WAIT	#037h	55	#1A71Fh
WSLOG	#013h	19	#19848h
ΣX	#123h	291	#1FDF7h
ΣX^2	#125h	293	#1FE2Dh
XCOL	#132h	306	#1FFDAh
XMIT	#167h	359	#21E75h
XOR	#0E8h	232	#1EBF6h
XPON	#069h	105	#1BC45h
XRNG	#0DAh	218	#1E621h
XROOT	#04Ah	74	#1B185h
ΣX*Y	#127h	295	#1FE63h
ΣY	#124h	292	#1FE12h
ΣY^2	#126h	294	#1FE48h
YCOL	#133h	307	#1FFFAh
YRNG	#0DBh	219	#1E641h
+	#044h	68	#1AB67h
+	#045h	69	#1ACDDh
-	#046h	70	#1AD09h
*	#047h	71	#1ADEEh
/	#048h	72	#1AF05h
^	#049h	73	#1B02Dh
<	#0EBh	235	#1EBBEh
≤	#0EDh	237	#1ECFCh
>	#0ECh	236	#1EC5Dh
≥	#0EEh	238	#1ED9Bh
=	#03Bh	59	#1A8D8h
==	#0E9h	233	#1E972h
≠	#0EAh	234	#1EA9Dh
!	#063h	99	#1BB02h
∫	#0FCh	252	#1F1D4h
∫	#0FDh	253	#1F223h
∂	#0F7h	247	#1EF7Eh
∂	#0F8h	248	#1EFD2h
‰	#07Ch	124	#1C060h
π	#03Fh	63	#1AABDh

Σ	#0FEh	254	#1F2C9h
Σ+	#11Eh	286	#1FD61h
Σ-	#11Fh	287	#1FD8Bh
∫	#04Fh	79	#1B374h
|	#0FFh	255	#1F354h
|	#100h	256	#1F3F3h
_	#108h	267	#1FAEBh
	#104h	260	#1F640h
	#105h	261	#1F996h
	#106h	262	#1F9AEh

Pour la librairie #700h:

C$	#01Bh	27	#23813h
CASE	#019h	25	#2378Dh
DIR	#01Bh	27	#23813h
DO	#007h	7	#230C3h
ELSE	#002h	2	#22FB5h
END	#003h	3	#22FD5h
END	#016h	22	#23694h
END	#017h	23	#236B9h
FOR	#00Ah	10	#231A0h
GROB	#01Bh	27	#23813h
HALT	#00Eh	14	#23472h
IF	#000h	0	#22EC3h
IFERR	#00Dh	13	#233DFh
NEXT	#00Bh	11	#2324Ch
PROMPT	#01Ch	28	#23824h
REPEAT	#006h	6	#2305Dh
START	#009h	9	#23103h
STEP	#00Ch	12	#23380h
THEN	#001h	1	#22EFAh
THEN	#018h	24	#2371Fh
THEN	#01Ah	26	#237A8h
UNTIL	#008h	8	#230EDh
WHILE	#005h	5	#23033h
XLIB	#01Bh	27	#23813h
«	#012h	18	#2361Eh
»	#011h	17	#235FEh
»	#013h	19	#23639h
→	#004h	4	#22FEBh
→	#010h	16	#234C1h
'	#014h	20	#23654h
'	#015h	21	#23679h
	#00Fh	17	#2349Ch

Liste des commandes par leur numéro

Cette seconde liste présente les commandes de la HP48 par leur numéro de commande (une commande est entièrement définie par son numéro de librairie et son numéro de commande).

De la même manière qu'il existait des noms en double dans la liste précédente (commandes de même nom mais définies par leur contexte, comme par exemple « qui peut être le début de programme ou le début de zone de définition d'une variable locale introduite par →), il existe des commandes de mêmes numéros.

Ceci s'explique par le fait que ce sont des instructions de présentation (DIR, C$...) et non de vraies commandes: elles réalisent toutes la même fonction, celle de délimiteur d'objets.

La première table correspond à la librairie #002h.

0	#000h	#1957Bh	ASR
1	#001h	#1959Bh	RL
2	#002h	#195BBh	RLB
3	#003h	#195DBh	RR
4	#004h	#195FBh	RRB
5	#005h	#1961Bh	SL
6	#006h	#1963Bh	SLB
7	#007h	#1965Bh	SR
8	#008h	#1967Bh	SRB
9	#009h	#1969Bh	R→B
10	#00Ah	#196BBh	B→R
11	#00Bh	#196DBh	CONVERT
12	#00Ch	#1971Bh	UVAL
13	#00Dh	#1974Fh	→UNIT
14	#00Eh	#19771h	UBASE
15	#00Fh	#197A5h	UFACT
16	#010h	#197F7h	TIME
17	#011h	#19812h	DATE
18	#012h	#1982Dh	TICKS
19	#013h	#19848h	WSLOG
20	#014h	#19863h	ACKALL
21	#015h	#1987Eh	ACK
22	#016h	#1989Eh	→DATE
23	#017h	#198BEh	→TIME
24	#018h	#198DEh	CLKADJ
25	#019h	#198FEh	STOALARM
26	#01Ah	#19928h	RCLALARM
27	#01Bh	#19948h	FINDALARM
28	#01Ch	#19972h	DELALARM
29	#01Dh	#19992h	TSTR
30	#01Eh	#199B2h	DDAYS
31	#01Fh	#199D2h	DATE+
32	#020h	#1A105h	CRDIR
33	#021h	#1A125h	PATH
34	#022h	#1A140h	HOME
35	#023h	#1A158h	UPDIR
36	#024h	#1A194h	VARS
37	#025h	#1A1AFh	TVARS
38	#026h	#1A1D9h	BYTES
39	#027h	#1A2BCh	NEWOB
40	#028h	#1A303h	KILL
41	#029h	#1A31Eh	OFF
42	#02Ah	#1A339h	DOERR
43	#02Bh	#1A36Dh	ERR0
44	#02Ch	#1A388h	ERRN
45	#02Dh	#1A3A3h	ERRM
46	#02Eh	#1A3BEh	EVAL
47	#02Fh	#1A3FEh	IFTE

```
 48 #030h #1A4CDh IFT
 49 #031h #1A52Eh SYSEVAL
 50 #032h #1A584h DISP
 51 #033h #1A5A4h FREEZE
 52 #034h #1A5C4h BEEP
 53 #035h #1A5E4h →NUM
 54 #036h #1A604h LAST
 54 #036h #1A604h LASTARG
 55 #037h #1A71Fh WAIT
 56 #038h #1A858h CLLCD
 57 #039h #1A873h KEY
 58 #03Ah #1A8BBh CONT
 59 #03Bh #1A8D8h =
 60 #03Ch #1A995h NEG
 61 #03Dh #1AA1Fh ABS
 62 #03Eh #1AA6Eh CONJ
 63 #03Fh #1AABDh ¥
 64 #040h #1AADFh MAXR
 65 #041h #1AB01h MINR
 66 #042h #1AB23h e
 67 #043h #1AB45h i
 68 #044h #1AB67h +
 69 #045h #1ACDDh +
 70 #046h #1AD09h -
 71 #047h #1ADEEh *
 72 #048h #1AF05h /
 73 #049h #1B02Dh ^
 74 #04Ah #1B185h XROOT
 76 #04Ch #1B278h INV
 77 #04Dh #1B2DBh ARG
 78 #04Eh #1B32Ah SIGN
 79 #04Fh #1B374h ∫
 80 #050h #1B426h SQ
 81 #051h #1B4ACh SIN
 82 #052h #1B505h COS
 83 #053h #1B55Eh TAN
 84 #054h #1B5B7h SINH
 85 #055h #1B606h COSH
 86 #056h #1B655h TANH
 87 #057h #1B6A4h ASIN
 88 #058h #1B72Fh ACOS
 89 #059h #1B79Ch ATAN
 90 #05Ah #1B7E8h ASINH
 91 #05Bh #1B830h ACOSH
 92 #05Ch #1B8A2h ATANH
 93 #05Dh #1B905h EXP
 94 #05Eh #1B94Fh LN
 95 #05Fh #1B9C6h LOG
 96 #060h #1BA3Dh ALOG
 97 #061h #1BA8Ch LNP1
 98 #062h #1BAC2h EXPM
 99 #063h #1BB02h !
100 #064h #1BB41h FACT
101 #065h #1BB6Dh IP
```

```
102 #066h #1BBA3h FP
103 #067h #1BBD9h FLOOR
104 #068h #1BC0Fh CEIL
105 #069h #1BC45h XPON
106 #06Ah #1BC71h MAX
107 #06Bh #1BCE3h MIN
108 #06Ch #1BD55h RND
109 #06Dh #1BDD1h TRNC
110 #06Eh #1BE4Dh MOD
111 #06Fh #1BE9Ch MANT
112 #070h #1BEC8h D→R
113 #071h #1BEF4h R→D
114 #072h #1BF1Eh →HMS
115 #073h #1BF3Eh HMS→
116 #074h #1BF5Eh HMS→
117 #075h #1BF7Eh HMS-
118 #076h #1BF9Eh RNRM
119 #077h #1BFBEh CNRM
120 #078h #1BFDEh DET
121 #079h #1BFFEh DOT
122 #07Ah #1C01Eh CROSS
123 #07Bh #1C03Eh RSD
124 #07Ch #1C060h %
125 #07Dh #1C0D7h %T
126 #07Eh #1C149h %CH
127 #07Fh #1C1B9h RAND
128 #080h #1C1D4h RDZ
129 #081h #1C1F6h COMB
130 #082h #1C236h PERM
131 #083h #1C274h SF
132 #084h #1C2D5h CF
133 #085h #1C313h FS?
134 #086h #1C360h FC?
135 #087h #1C399h DEG
136 #088h #1C3B4h RAD
137 #089h #1C3CFh GRAD
138 #08Ah #1C3EAh FIX
139 #08Bh #1C41Eh SCI
140 #08Ch #1C452h ENG
141 #08Dh #1C486h STD
142 #08Eh #1C4A1h FS?C
143 #08Fh #1C520h FC?C
144 #090h #1C559h BIN
145 #091h #1C574h DEC
146 #092h #1C58Fh HEX
147 #093h #1C5AAh OCT
148 #094h #1C5C5h STWS
149 #095h #1C5FEh RCWS
150 #096h #1C619h RCLF
151 #097h #1C67Fh STOF
152 #098h #1C783h →LIST
153 #099h #1C79Eh R→C
154 #09Ah #1C7CAh RE
155 #09Bh #1C819h IM
```

```
156 #09Ch #1C85Ch SUB
157 #09Dh #1C8EAh REPL
158 #09Eh #1C95Ah LIST→
159 #09Fh #1C98Eh C→R
160 #0A0h #1C9B8h SIZE
161 #0A1h #1CAB4h POS
162 #0A2h #1CB0Bh →STR
163 #0A3h #1CB26h STR→
164 #0A4h #1CB46h NUM
165 #0A5h #1CB66h CHR
166 #0A6h #1CB86h TYPE
167 #0A7h #1CE28h VTYPE
168 #0A8h #1CEE3h EQ→
169 #0A9h #1CF7Bh OBJ→
170 #0AAh #1D009h →ARRY
171 #0ABh #1D092h ARRY→
172 #0ACh #1D0DFh RDM
173 #0ADh #1D186h CON
174 #0AEh #1D2DCh IDN
175 #0AFh #1D392h TRN
176 #0B0h #1D407h PUT
177 #0B1h #1D5DFh PUTI
178 #0B2h #1D7C6h GET
179 #0B3h #1D8C7h GETI
180 #0B4h #1DD06h V→
181 #0B5h #1DE66h →V2
182 #0B6h #1DEC2h →V3
183 #0B7h #1E04Ah INDEP
184 #0B8h #1E07Eh PMIN
185 #0B9h #1E09Eh PMAX
186 #0BAh #1E0BEh AXES
187 #0BBh #1E0E8h CENTR
188 #0BCh #1E126h RES
189 #0BDh #1E150h *H
190 #0BEh #1E170h *W
191 #0BFh #1E190h DRAW
192 #0C0h #1E1ABh AUTO
193 #0C1h #1E1C6h DRAX
194 #0C2h #1E1E1h SCALE
195 #0C3h #1E201h PDIM
196 #0C4h #1E228h DEPND
197 #0C5h #1E25Fh ERASE
198 #0C6h #1E27Ah PX→C
199 #0C7h #1E29Ah C→PX
200 #0C8h #1E2BAh GRAPH
201 #0C9h #1E2D5h LABEL
202 #0CAh #1E2F0h PVIEW
203 #0CBh #1E31Ah PIXON
204 #0CCh #1E344h PIXOFF
205 #0CDh #1E36Eh PIX?
206 #0CEh #1E398h LINE
207 #0CFh #1E3C2h TLINE
208 #0D0h #1E3ECh BOX
209 #0D1h #1E416h BLANK
```

```
210 #0D2h #1E436h PICT
211 #0D3h #1E456h GOR
212 #0D4h #1E4E4h GXOR
213 #0D5h #1E572h LCD→
214 #0D6h #1E58Dh →LCD
215 #0D7h #1E5ADh →GROB
216 #0D8h #1E5D2h ARC
217 #0D9h #1E606h TEXT
218 #0DAh #1E621h XRNG
219 #0DBh #1E641h YRNG
220 #0DCh #1E661h FUNCTION
221 #0DDh #1E681h CONIC
222 #0DEh #1E6A1h POLAR
223 #0DFh #1E6C1h PARAMETRIC
224 #0E0h #1E6E1h TRUTH
225 #0E1h #1E701h SCATTER
226 #0E2h #1E721h HISTOGRAM
227 #0E3h #1E741h BAR
228 #0E4h #1E761h SAME
229 #0E5h #1E783h AND
230 #0E6h #1E809h OR
231 #0E7h #1E88Fh NOT
232 #0E8h #1E8F6h XOR
233 #0E9h #1E972h ==
234 #0EAh #1EA9Dh ≠
235 #0EBh #1EBBEh <
236 #0ECh #1EC5Dh >
237 #0EDh #1ECFCh ≤
238 #0EEh #1ED9Bh ≥
239 #0EFh #1EE38h OLDPRT
240 #0F0h #1EE53h PR1
241 #0F1h #1EE6Eh PRSTC
242 #0F2h #1EE89h PRST
243 #0F3h #1EEA4h CR
244 #0F4h #1EEBFh PRVAR
245 #0F5h #1EF43h DELAY
246 #0F6h #1EF63h PRLCD
247 #0F7h #1EF7Eh ∂
248 #0F8h #1EFD2h ∂
249 #0F9h #1F133h RCEQ
250 #0FAh #1F14Eh STEQ
251 #0FBh #1F16Eh RDOT
252 #0FCh #1F1D4h ∫
253 #0FDh #1F223h ∫
254 #0FEh #1F2C9h Σ
255 #0FFh #1F354h |
256 #100h #1F3F3h |
257 #101h #1F500h QUOTE
258 #102h #1F550h APPLY
259 #103h #1F55Ch APPLY
260 #104h #1F640h
261 #105h #1F996h
262 #106h #1F9AEh
263 #107h #1F9C4h →Q
```

264	#108h	#1F9E9h →Q■
265	#109h	#1FA59h ↑MATCH
266	#10Ah	#1FA8Dh ↓MATCH
267	#10Bh	#1FAEBh _
268	#10Ch	#1FB5Dh RATIO
269	#10Dh	#1FB87h DUP
270	#10Eh	#1FBA2h DUP2
271	#10Fh	#1FBBDh SWAP
272	#110h	#1FBD8h DROP
273	#111h	#1FBF3h DROP2
274	#112h	#1FC0Eh ROT
275	#113h	#1FC29h OVER
276	#114h	#1FC44h DEPTH
277	#115h	#1FC64h DROPN
278	#116h	#1FC7Fh DUPN
279	#117h	#1FC9Ah PICK
280	#118h	#1FCB5h ROLL
281	#119h	#1FCD0h ROLLD
282	#11Ah	#1FCE8h CLEAR
283	#11Bh	#1FD0Bh STOΣ
284	#11Ch	#1FD2Bh CLΣ
285	#11Dh	#1FD46h RCLΣ
286	#11Eh	#1FD61h Σ+
287	#11Fh	#1FD8Bh Σ-
288	#120h	#1FDA6h NΣ
289	#121h	#1FDC1h CORR
290	#122h	#1FDDCh COV
291	#123h	#1FDF7h ΣX
292	#124h	#1FE12h ΣY
293	#125h	#1FE2Dh ΣX^2
294	#126h	#1FE48h ΣY^2
295	#127h	#1FE63h ΣX*Y
296	#128h	#1FE7Eh MAXΣ
297	#129h	#1FE99h MEAN
298	#12Ah	#1FEB4h MINΣ
299	#12Bh	#1FECFh SDEV
300	#12Ch	#1FEEAh TOT
301	#12Dh	#1FF05h VAR
302	#12Eh	#1FF20h LR
303	#12Fh	#1FF7Ah PREDV
304	#130h	#1FF9Ah PREDY
305	#131h	#1FFBAh PREDX
306	#132h	#1FFDAh XCOL
307	#133h	#1FFFAh YCOL
308	#134h	#2001Ah UTPC
309	#135h	#2003Ah UTPN
310	#136h	#2005Ah UTPF
311	#137h	#2007Ah UTPT
312	#138h	#2009Ah COLΣ
313	#139h	#200C4h SCLΣ
314	#13Ah	#200F3h ΣLINE
315	#13Bh	#2010Eh BINS
316	#13Ch	#20133h BARPLOT
317	#13Dh	#20167h HISTPLOT
318	#13Eh	#2018Ch SCATRPLOT
319	#13Fh	#201B1h LINFIT
320	#140h	#201D6h LOGFIT
321	#141h	#201FBh EXPFIT
322	#142h	#20220h PWRFIT
323	#143h	#2025Eh BESTFIT
324	#144h	#202CEh SINV
325	#145h	#2034Dh SNEG
326	#146h	#203CCh SCONJ
327	#147h	#2044Bh STO+
328	#148h	#20538h STO-
329	#149h	#2060Ch STO/
330	#14Ah	#20753h STO*
331	#14Bh	#208F4h INCR
332	#14Ch	#209AAh DECR
333	#14Dh	#20A15h COLCT
334	#14Eh	#20A49h EXPAN
335	#14Fh	#20A7Dh RULES
336	#150h	#20A93h ISOL
337	#151h	#20AB3h QUAD
338	#152h	#20AD3h SHOW
339	#153h	#20B20h TAYLR
340	#154h	#20B40h RCL
341	#155h	#20CCDh STO
342	#156h	#20D65h DEFINE
343	#157h	#20EFEh PURGE
344	#158h	#20FAAh MEM
345	#159h	#20FD9h ORDER
346	#15Ah	#210FCh CLUSR
346	#15Ah	#210FCh CLVAR
347	#15Bh	#21150h TMENU
348	#15Ch	#21196h MENU
349	#15Dh	#211E1h RCLMENU
350	#15Eh	#211FCh PVARS
351	#15Fh	#2123Ah PGDIR
352	#160h	#2125Ah ARCHIVE
353	#161h	#2133Ch RESTORE
354	#162h	#2137Fh MERGE
355	#163h	#213D1h FREE
356	#164h	#2142Dh LIBS
357	#165h	#21448h ATTACH
358	#166h	#2147Ch DETACH
359	#167h	#21E75h XMIT
360	#168h	#21E95h SRECV
361	#169h	#21EB5h OPENIO
362	#16Ah	#21ED5h CLOSEIO
363	#16Bh	#21EF0h SEND
364	#16Ch	#21F24h KGET
365	#16Dh	#21F62h RECN
366	#16Eh	#21F96h RECV
367	#16Fh	#21FB6h FINISH
368	#170h	#21FD1h SERVER
369	#171h	#21FECh CKSM
370	#172h	#2200Ch BAUD

```
371 #173h #2202Ch PARITY
372 #174h #2204Ch TRANSIO
373 #175h #2206Ch KERRM
374 #176h #22087h BUFLEN
375 #177h #220A2h STIME
376 #178h #220C2h SBRK
377 #179h #220DDh PKT
378 #17Ah #224CAh INPUT
379 #17Bh #224F4h ASN
380 #17Ch #22514h STOKEYS
381 #17Dh #22548h DELKEYS
382 #17Eh #22586h RCLKEYS
383 #17Fh #225BEh →TAG
384 #180h #22633h DTAG
```

Pour la librairie #700h:

```
  0 #000h #22EC3h IF
  1 #001h #22EFAh THEN
  2 #002h #22FB5h ELSE
  3 #003h #22FD5h END
  4 #004h #22FEBh →
  5 #005h #23033h WHILE
  6 #006h #2305Dh REPEAT
  7 #007h #230C3h DO
  8 #008h #230EDh UNTIL
  9 #009h #23103h START
 10 #00Ah #231A0h FOR
 11 #00Bh #2324Ch NEXT
 12 #00Ch #23380h STEP
 13 #00Dh #233DFh IFERR
 14 #00Eh #23472h HALT
 17 #00Fh #2349Ch
 16 #010h #234C1h →
 17 #011h #235FEh »
 18 #012h #2361Eh «
 19 #013h #23639h »
 20 #014h #23654h '
 21 #015h #23679h '
 22 #016h #23694h END
 23 #017h #236B9h END
 24 #018h #2371Fh THEN
 25 #019h #2378Dh CASE
 26 #01Ah #237A8h THEN
 27 #01Bh #23B13h C$
 27 #01Bh #23B13h DIR
 27 #01Bh #23B13h GROB
 27 #01Bh #23B13h XLIB
 28 #01Ch #23B24h PROMPT
```

Liste d'objets en rom

Voici la liste des adresses d'objets situés en mémoire morte. Cette liste n'est pas exhaustive mais présente les objets succeptibles d'être utiles: au lieu de coder l'objet dont il a besoin, le programmeur n'aura qu'à faire référence aux adresses correspondantes...

La première colonne de chaque tableau contient l'adresse de l'objet qui est décrit dans la ou les autres colonnes (plusieurs formes sont parfois données pour un même objet).

Remarque: les adresses supérieures à #70000h correspondent à des objets situés en rom cachée et ne peuvent donc pas être utilisées directement (il faut utiliser le programme ROMRCL qui fait partie de la bibliothèque de programmes)...

System Binary

#03FEFh	<0h>	0
#03FF9h	<1h>	1
#04003h	<2h>	2
#0400Dh	<3h>	3
#04017h	<4h>	4
#04021h	<5h>	5
#0402Bh	<6h>	6
#04035h	<7h>	7
#0403Fh	<8h>	8
#04049h	<9h>	9
#04053h	<Ah>	10
#0405Dh	<Bh>	11
#04067h	<Ch>	12
#04071h	<Dh>	13
#0407Bh	<Eh>	14
#04085h	<Fh>	15
#0408Fh	<10h>	16
#04099h	<11h>	17
#040A3h	<12h>	18
#040ADh	<13h>	19
#040B7h	<14h>	20
#040C1h	<15h>	21
#040CBh	<16h>	22

#040D5h	<17h>	23
#040DFh	<18h>	24
#040E9h	<19h>	25
#040F3h	<1Ah>	26
#040FDh	<1Bh>	27
#04107h	<1Ch>	28
#04111h	<1Dh>	29
#0411Bh	<1Eh>	30
#04125h	<1Fh>	31
#0412Fh	<20h>	32
#04139h	<21h>	33
#04143h	<22h>	34
#0414Dh	<23h>	35
#04157h	<24h>	36
#04161h	<25h>	37
#0416Bh	<26h>	38
#04175h	<27h>	39
#0417Fh	<28h>	40
#04189h	<29h>	41
#04193h	<2Ah>	42
#0419Dh	<2Bh>	43
#64B12h	<2Ch>	44
#64B1Ch	<2Dh>	45

#64B26h	<2Eh>	46
#64B30h	<2Fh>	47
#64B3Ah	<30h>	48
#64B44h	<31h>	49
#64B4Eh	<32h>	50
#64B58h	<33h>	51
#64B62h	<34h>	52
#64B6Ch	<35h>	53
#64B76h	<36h>	54
#64B80h	<37h>	55
#64B8Ah	<38h>	56
#64B94h	<39h>	57
#64B9Eh	<3Ah>	58
#64BA8h	<3Bh>	59
#64BB2h	<3Ch>	60
#64BBCh	<3Dh>	61
#64BC6h	<3Eh>	62
#64BD0h	<3Fh>	63
#64BDAh	<40h>	64
#64BE4h	<41h>	65
#64BEEh	<42h>	66
#64BF8h	<43h>	67
#64C02h	<44h>	68
#64C0Ch	<45h>	69
#64C16h	<46h>	70
#64C20h	<4Ah>	74
#2D96Ah	<4Bh>	75
#6B5C4h	<4Dh>	77
#64C2Ah	<4Fh>	79
#64C34h	<50h>	80
#64C3Eh	<51h>	81
#64C48h	<52h>	82
#64C52h	<53h>	83
#64C5Ch	<54h>	84
#64C66h	<55h>	85
#64C70h	<56h>	86
#64C7Ah	<57h>	87
#3A215h	<58h>	88
#64C84h	<5Bh>	91
#1CCD8h	<5Fh>	95
#64C8Eh	<60h>	96
#64C98h	<61h>	97
#6B696h	<61h>	97
#64CA2h	<62h>	98
#64CACh	<64h>	100
#64CB6h	<65h>	101
#6D98Ch	<68h>	104
#3A20Bh	<6Eh>	110
#64CC0h	<6Fh>	111
#64CCAh	<70h>	112
#64CD4h	<71h>	113
#64CDEh	<72h>	114
#64CE8h	<73h>	115
#64CF2h	<74h>	116

#64CFCh	<75h>	117
#64D06h	<7Ah>	122
#39980h	<7Dh>	125
#6C947h	<7Fh>	127
#64D10h	<80h>	128
#64D1Ah	<82h>	130
#64D24h	<83h>	131
#46BE3h	<84h>	132
#2F4A2h	<86h>	134
#64D2Eh	<8Fh>	143
#64D38h	<91h>	145
#64D42h	<92h>	146
#6C68Bh	<93h>	147
#64D4Ch	<9Ah>	154
#64D56h	<9Eh>	158
#64D60h	<9Fh>	159
#64D6Ah	<A0h>	160
#64D74h	<A1h>	161
#64D7Eh	<A2h>	162
#64D88h	<A5h>	165
#64D92h	<A6h>	166
#64D9Ch	<A7h>	167
#64DA6h	<A9h>	169
#64DB0h	<AAh>	170
#64DBAh	<AEh>	174
#1CD69h	<AFh>	175
#64DC4h	<B1h>	177
#17F4Ah	<B8h>	184
#64DCEh	<BBh>	187
#64DD8h	<C0h>	192
#20D2Ch	<C8h>	200
#64DE2h	<CCh>	204
#64DECh	<D0h>	208
#64DF6h	<E1h>	225
#64E00h	<EAh>	234
#64E0Ah	<EEh>	238
#64E14h	<F0h>	240
#64E1Eh	<FDh>	253
#64E28h	<FFh>	255
#64E32h	<100h>	256
#64E3Ch	<102h>	258
#50E45h	<104h>	260
#64E46h	<106h>	262
#64E50h	<107h>	263
#64E5Ah	<110h>	272
#64E64h	<111h>	273
#15E0Bh	<112h>	274
#15D6Fh	<117h>	279
#15DABh	<118h>	280
#64E6Eh	<123h>	291
#64E78h	<124h>	292
#31C5Eh	<127h>	295
#33CA1h	<12Fh>	303
#64E82h	<131h>	305

#64E8Ch	<132h>	306
#64E96h	<133h>	307
#64EA0h	<134h>	308
#64EAAh	<135h>	309
#64EB4h	<136h>	310
#64EBEh	<137h>	311
#64EC8h	<138h>	312
#64ED2h	<139h>	313
#64EDCh	<13Ah>	314
#64EE6h	<13Bh>	315
#64EF0h	<13Dh>	317
#64EFAh	<13Eh>	318
#64F04h	<151h>	337
#64F0Eh	<200h>	512
#4ECFDh	<201h>	513
#4ED25h	<202h>	514
#2E7D6h	<204h>	516
#64F18h	<205h>	517
#673B7Bh	<206h>	518
#1B147h	<304h>	772
#64F22h	<311h>	785
#1C930h	<313h>	787
#64F2Ch	<411h>	1041
#64F36h	<412h>	1042
#64F40h	<444h>	1092
#64F4Ah	<451h>	1105
#64F54h	<452h>	1106
#37DE7h	<501h>	1281
#472C8h	<502h>	1282
#472D2h	<503h>	1283
#472DCh	<504h>	1284
#472E6h	<505h>	1285
#472F0h	<506h>	1286
#64F5Eh	<510h>	1296
#64F68h	<511h>	1297
#1C93Fh	<515h>	1301
#64F72h	<550h>	1360
#50E4Fh	<602h>	1538
#4A320h	<605h>	1541
#4A32Ah	<606h>	1542
#4A334h	<607h>	1543
#4A33Eh	<608h>	1544
#4A348h	<609h>	1545
#4A352h	<60Ah>	1546
#4A35Ch	<60Bh>	1547
#4A366h	<60Ch>	1548
#4A370h	<60Dh>	1549
#4A37Ah	<60Eh>	1550
#4A384h	<60Fh>	1551
#4A38Eh	<610h>	1552
#4A398h	<611h>	1553
#4A3A2h	<612h>	1554
#4A3ACh	<613h>	1555
#4A3B6h	<614h>	1556

#4A3C0h	<615h>	1557
#4A3CAh	<616h>	1558
#4A3D4h	<617h>	1559
#4A3DEh	<618h>	1560
#4A3E8h	<619h>	1561
#4A3F2h	<61Ah>	1562
#4A3FCh	<61Bh>	1563
#4A406h	<61Ch>	1564
#4A410h	<61Dh>	1565
#4A41Ah	<61Eh>	1566
#4A424h	<61Fh>	1567
#4A42Eh	<620h>	1568
#4A438h	<621h>	1569
#4A442h	<622h>	1570
#4A44Ch	<623h>	1571
#4A456h	<624h>	1572
#4A460h	<628h>	1576
#4A46Ah	<629h>	1577
#4A474h	<62Ah>	1578
#4A47Eh	<62Bh>	1579
#4A488h	<62Ch>	1580
#4A492h	<62Dh>	1581
#4A49Ch	<62Eh>	1582
#20496h	<644h>	1604
#64F86h	<650h>	1616
#64F90h	<700h>	1792
#1D448h	<710h>	1808
#1D42Fh	<750h>	1872
#08E14h	<7FFh>	2047
#4B32Dh	<800h>	2048
#1C8A7h	<822h>	2082
#1E548h	<82Ch>	2092
#1C898h	<855h>	2133
#1C912h	<85Ch>	2140
#1E49Ch	<85Ch>	2140
#1E52Ah	<85Ch>	2140
#64F9Ah	<861h>	2145
#64FA4h	<862h>	2146
#64FAEh	<865h>	2149
#64FB8h	<86Eh>	2158
#20D4Ah	<8F1h>	2289
#20D3Bh	<9F1h>	2545
#33CBFh	<A01h>	2561
#33CD3h	<A02h>	2562
#34301h	<A03h>	2563
#33D91h	<A04h>	2564
#33C29h	<A05h>	2565
#33C83h	<A06h>	2566
#64FCCh	<A11h>	2577
#64FD6h	<A12h>	2578
#64FE0h	<A1Ah>	2586
#64FEAh	<A21h>	2593
#64FF4h	<A22h>	2594
#64FFEh	<A2Ah>	2602

#65008h	<A61h>	2657
#65012h	<A62h>	2658
#6501Ch	<A65h>	2661
#65026h	<A6Eh>	2670
#65030h	<AA1h>	2721
#6503Ah	<AA2h>	2722
#65044h	<AAAh>	2730
#28AE?h	<B01h>	2817
#2D5C3h	<C02h>	3074
#6504Eh	<C06h>	3078
#65058h	<C07h>	3079
#65062h	<C08h>	3080
#6506Ch	<C0Ah>	3082
#650?6h	<C0Bh>	3083
#2DA?Ch	<C0Ch>	3084
#2F0E0h	<C0Dh>	3085
#2F8EEh	<C0Eh>	3086
#2FA9Ch	<C0Fh>	3087
#2F04Ah	<C10h>	3088
#2DD6Fh	<C11h>	3089
#2EC39h	<C12h>	3090
#3182Ch	<C15h>	3093
#31BF1h	<C16h>	3094
#6?385h	<C17h>	3095
#1C889h	<C22h>	3106
#1E50Ch	<C2Ch>	3116
#1C8?Ah	<C55h>	3157

#1E4EEh	<C5Ch>	3164
#26289h	<CFFh>	3327
#65080h	<DFFh>	3583
#65088h	<E00h>	3584
#1E5DCh	<2111h>	8465
#03F8Bh	<2933h>	10547
#03FDBh	<2955h>	10581
#03F95h	<2977h>	10615
#03F9Fh	<2A74h>	10868
#03FC?h	<2A96h>	10902
#191?3h	<2A96h>	10902
#03FBDh	<2AB8h>	10936
#03FE5h	<2ADAh>	10970
#25C3?h	<2B1Eh>	11038
#03FB3h	<2D9Dh>	11677
#03FA9h	<2E48h>	11848
#03FD1h	<2E6Dh>	11885
#397?4Dh	<3039h>	12345
#16AD6h	<4000h>	16384
#16AE5h	<5000h>	20480
#16AF4h	<8000h>	32768
#16B03h	<9000h>	36864
#16B21h	<D000h>	53248
#16B12h	<E000h>	57344
#65094h	<70000h>	458752
#6?D12h	<80000h>	524288
#65096h	<FFFFFh>	1048575

Real

#2A48?h	-9.99999999999E499
#2A1D?h	-4.77451811461E441
#2B139h	-260
#2A42Eh	-9
#2A419h	-8
#2A404h	-7
#2A3EFh	-6
#2A3DAh	-5
#2A3C5h	-4
#2A3B0h	-3
#2A39Bh	-2
#2A386h	-1
#650D2h	-0.5
#2A4B1h	-1E-499
#2A2B4h	0
#2A49Ch	1E-499
#52C?2h	1E-12
#4B864h	3.49065850399E-2
#494B4h	0.1
#?D2??h	4.34294481904E-1
#650BDh	0.5
#49618h	0.15
#2A2C9h	1

#31F4Fh	1.8
#2A2DEh	2
#1A223h	2.5
#650A8h	2.71828182846
#2A2F3h	3
#2A443h	3.14159265359
#2A308h	4
#2A31Dh	5
#2A332h	6
#514EBh	6.28318530718
#2A34?h	7
#2A35Ch	8
#2A3?1h	9
#650E?h	10
#1CC03h	11
#1CC1Dh	12
#1CC3?h	13
#1CC51h	14
#1CC85h	15
#1CD3Ah	16
#1CD54h	17
#1CDF2h	18
#1CE0?h	19

```
#1CC6Bh  20
#1CCA4h  21
#1CCC3h  22
#1CCE2h  23
#1CD01h  24
#1CD20h  25
#1CD73h  26
#1CD8Dh  27
#49161h  40
#320B1h  80
#415F1h  100
#650FCh  180
#65111h  200
#2B0CEh  260
#65126h  360
```

```
#6513Bh  400
#4C035h  499
#4C068h  499
#22352h  1200
#2236?h  2400
#223?Ch  4800
#0EFEEh  8192
#1A?CEh  8192
#22391h  9600
#0F003h  491520
#0F018h  29491200
#0F02Dh  707788800
#0F042h  4954521600
#2A4?2h  9.99999999999E499
```

Long Real

```
#2B1D6h  -1E-10000
#2B31Fh  -495.920119017593
#2B36Ch  -76.5594818140208
#2B3B9h  -1.21142857142857
#2A4C6h  0
#2B1BCh  1E-10000
#2A62Ch  1.74532925199433E-2
#10ED0h  7.95774715459477E-2
#2A562h  0.1
#2B3DDh  0.4
#2A5?Ch  0.5
#10E68h  5.55555555555556E-1
#52A2Fh  0.7
#2B410h  9.18938533204673E-1
#2A4E0h  1
#2A4FAh  2
#5230Fh  2.30258509299405
```

```
#2A514h  3
#2A458h  3.14159265358979
#2A52Eh  4
#2A548h  5
#0F688h  6.28318530717959
#2B1FFh  7
#2B390h  9.33584905660377
#2A596h  10
#2B2DCh  12
#2B343h  30.3479606073615
#0F547?h  32
#2B300h  60
#2C1C5h  100
#10E9Ch  273.15
#10EB6h  459.67
#2B0F2h  1E10000
```

Complex

```
#4AB2Ah  (0,  0)
#524AFh  (0,  0)
#5196Ah  (-1,  0)
```

```
#524F?h  (1,  0)
#526?Fh  (0,  1)
#526AEh  (0,  -1)
```

Long Complex

```
#5193Bh  (0,  0)
```

Character

03918h	' '
05127h	'A'
05A9Eh	'A'
0D318h	' '
0D333h	' '
0FA62h	'k'
1016Fh	' '
1918Ch	'I'
44457h	'A'
450BEh	'·'
45174h	'-'
6202Ch	'λ'
6472Fh	' '
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	'+'
65640h	'<'
65647h	'>'
6564Eh	'∠'
65655h	'∂'
6565Ch	'∫'
65663h	'('
6566Ah	'⌟'
65671h	'π'
65678h	')'
6567Fh	'Σ'
65686h	' '
6568Dh	'_'
65694h	'['
6569Bh	']'
656A2h	'{'
656A9h	'}'
656B0h	'≤'
656B7h	'≥'
656BEh	'≠'
6947Ch	'' '
69483h	'@'
6D2B1h	'?'
7A929h	'α'
7A930h	'%'
7A937h	'' '
7A93Eh	'≤'
7A945h	'≥'
7A94Ch	'≠'

7A961h	'~'
7A968h	'β'
7A96Fh	'∘'
7A976h	'δ'
7A97Dh	'ε'
7A984h	'θ'
7A98Bh	'γ'
7A992h	'η'
7A999h	'i'
7A9A0h	'∞'
7A9A7h	'l'
7A9AEh	'↑'
7A9B5h	'λ'
7A9BCh	'μ'
7A9C3h	'▶'
7A9CAh	'Ω'
7A9D1h	'←'
7A9D8h	'↓'
7A9DFh	'ρ'
7A9E6h	'σ'
7A9EDh	'τ'
7A9F4h	'¿'
7A9FBh	'υ'
7AA02h	'ϰ'
7AA09h	'±'
7AA10h	'π'
7AA17h	'¢'
7AA1Eh	'£'
7AA25h	'¥'
7AA2Ch	'ϖ'
7AA33h	'°'
7AA3Ah	'!'
7AA41h	'?'
7AA48h	'&'
7AA4Fh	'@'
7AA56h	'$'

Array

```
#72000h  {  "Insufficient Memory" "Directory Recursion"
            "Undefined Local Name" "Undefined XLIB Name"
            "Memory Clear" "Power Lost" "Warning:"
            "Invalid Card Data" "Object In Use"
            "Port Not Available" "No Room in Port"
            "Object Not in Port" "Recovering Memory"
            "Try To Recover Memory?" "Replace RAM, Press ON"
            "No Mem To Config All" }
#72281h  {  "Bad Guess(es)" "Constant?" "Interrupted" "Zero"
            "Sign Reversal" "Extremum" }
#7232Ch  {  "Bad Packet Block Check" "Timeout"
            "Receive Error" "Receive Buffer Overrun"
            "Parity Error" "Transfer Failed"
            "Protocol Error" "Invalid Server Cmd."
            "Port Closed" "Connecting" "Retry #"
            "Awaiting Server Cmd." "Sending " "Receiving "
            "Object Discarded" "Packet #"
            "Processing Command" "Invalid IOPAR"
            "Invalid PRTPAR" "Low Battery" "Empty Stack"
            "Row " "Invalid Name" }
#7260Ah  {  "Invalid Date" "Invalid Time" "Invalid Repeat"
            "Nonexistent Alarm" }
#726A5h  {  "Invalid Unit" "Inconsistent Units" }
#72704h  {  "No Room to Save Stack" "Can't Edit Null Char."
            "Invalid User Function" "No Current Equation" ""
            "Invalid Syntax" "Real Number" "Complex Number"
            "String" "Real Array" "Complex Array" "List"
            "Global Name" "Local Name" "Program" "Algebraic"
            "Binary Integer" "Graphic" "Tagged" "Unit"
            "XLIB Name" "Directory" "Library" "Backup"
            "Function" "Command" "System Binary" "Long Real"
            "Long Complex" "Linked Array" "Character" "Code"
            "Library Data" "External" ""
            "LAST STACK Disabled" "LAST CMD Disabled"
            "HALT Not Allowed" "Array"
            "Wrong Argument Count" "Circular Reference"
            "Directory Not Allowed" "Non-Empty Directory"
            "Invalid Definition" "Missing Library"
            "Invalid PPAR" "Non-Real Result"
            "Unable to Isolate" "No Room to Show Stack"
            "Warning:␣" "Error:" "Purge?" "Out of Memory"
            "Stack" "Last Stack" "Last Commands"
            "Key Assignments" "Alarms" "Last Arguments"
            "Name Conflict" "Command Line" "" }
#72DCFh  {  "Too Few Arguments" "Bad Argument Type"
            "Bad Argument Value" "Undefined Name"
            "LASTARG Disabled" "Incomplete␣Subexpression"
            "Implicit () off" "Implicit () on" }
#72F1Eh  {  "Positive Underflow" "Negative Underflow"
            "Overflow" "Undefined Result" "Infinite Result" }
```

```
#72FE6h [  "Invalid Σ Data" "Nonexistent ΣDAT"
           "Insufficient Σ Data" "Invalid ΣPAR"
           "Invalid Σ Data LN(Neg)" "Invalid Σ Data LN(0)"
           "Invalid EQ" "Current equation:"
           "No current equation." "Enter eqn, press NEW"
           "Name the equation,␣press ENTER"
           "Select plot type" "Empty catalog" "undefined"
           "No stat data to plot" "Autoscaling"
           "Solving for " "No current data. Enter"
           "data point, press Σ+" "Select a model"
           "No alarms pending." "Press ALRM to create"
           "Next alarm:" "Past due alarm:" "Acknouledged"
           "Enter alarm, press SET"
           "Select repeat interval" "      I/O setup menu"
           "Plot type: " "" "" " (OFF SCREEN)"
           "Invalid PTYPE"
           "Name the stat data,␣press ENTER"
           "Enter value (zoom out␣if >1), press ENTER"
           "Copied to stack" "x axis zoom w/AUTO.␣"
           "x axis zoom.␣" "y axis zoom.␣"
           "x and y axis zoom.␣" "IR/wire:          "
           "ASCII/binary:   " "baud:              "
           "parity:           " "checksum type: "
           "translate code:" "Enter matrix, then NEW" ]
#736F9h [  "Invalid Dimension" "Invalid Array Element"
           "Deleting Row" "Deleting Column" "Inserting Row"
           "Inserting Column" ]
#7AA5Dh [  <3FFDBh>  <3FFF4h>  <4000Dh>  <654CDh>  <65583h>
           <7A929h> ]
#7AA94h [  <4003Fh>  <40026h>  <40058h>  <654D4h>  <6558Ah>
           <7A968h> ]
#7AACBh [  <40071h>  <4008Ah>  <400A3h>  <654DBh>  <65591h>
           <7A96Fh> ]
#7AB02h [  <400BCh>  <400D5h>  <400EEh>  <654E2h>  <65598h>
           <7A976h> ]
#7AB39h [  <40107h>  <40120h>  <40139h>  <654E9h>  <6559Fh>
           <7A97Dh> ]
#7AB70h [  <40152h>  <4016Bh>  <40184h>  <654F0h>  <655A6h>
           <7A984h> ]
#7ABA7h [  <3AD57h>  <3AE33h>  <1EE53h>  <654F7h>  <655ADh>
           <7A98Bh> ]
#7ABDEh [  <3AE1Ah>  <3A6E5h>  <21FD1h>  <654FEh>  <65584h>
           <7A992h> ]
#7AC15h [  <3AB72h>  <3ADA2h>  <3AD8Bh>  <65505h>  <6558Bh>
           <7A9A0h> ]
#7AC4Ch [  <3AF37h>  <3AD70h>  <3AD89h>  <6550Ch>  <655C2h>
           <7A9A7h> ]
#7AC83h [  <3A93Dh>  <3AE6Fh>  <3B00Eh>  <65513h>  <655C9h>
           <7A9AEh> ]
#7ACBAh [  <3A71Ch>  <3A735h>  <3B211h>  <6551Ah>  <655D0h>
           <7A9B5h> ]
#7ACF1h [  <3A69Ah>  <1A15Bh>  <1A140h>  <65521h>  <655D7h>
           <7A937h> ]
```

```
╋'/AD28h  [  <3A992h>  <20D65h>  <20840h>  <65528h>  <6550Eh>
            <7A9BCh>  ]
╋'/AD5Fh  [  <1A3BEh>  <1F9C4h>  <1A5E4h>  <6552Fh>  <655E5h>
            <7A9CAh>  ]
╋'/AD96h  [  <3A834h>  <1E2BAh>  <3AFE6h>  <65536h>  <655ECh>
            <7A9D1h>  ]
╋'/ADCDh  [  <3A645h>  <3AE4Ch>  <3AF69h>  <6553Dh>  <655F3h>
            <7A9D8h>  ]
╋'/AE04h  [  <3A8DEh>  <1F8BDh>  <3A80Ch>  <65544h>  <655FAh>
            <7A9DFh>  ]
╋'/AE3Bh  [  <1B4ACh>  <1B6A4h>  <1EF7Eh>  <6554Bh>  <65601h>
            <7A9E6h>  ]
╋'/AE72h  [  <1B505h>  <1B72Fh>  <1F1D4h>  <65552h>  <65608h>
            <7A9EDh>  ]
╋'/AEA9h  [  <1B55Eh>  <1B79Ch>  <1F2C9h>  <65559h>  <6560Fh>
            <7A930h>  ]
╋'/AEE0h  [  <1B374h>  <1B426h>  <1B185h>  <65560h>  <65616h>
            <7A961h>  ]
╋'/AF17h  [  <1B02Dh>  <1BA3Dh>  <1B9C6h>  <65567h>  <6561Dh>
            <7A9F8h>  ]
╋'/AF4Eh  [  <1B278h>  <1B905h>  <1B94Fh>  <6556Eh>  <65624h>
            <7AA02h>  ]
╋'/AF85h  [  <3A7F3h>  <3AF50h>  <3B068h>  <3A7F3h>  <7AA48h>
            <7AA4Fh>  ]
╋'/AFBCh  [  <3AAB2h>  <3A7C6h>  <3B12Bh>  <65575h>  <6562Bh>
            <7AA09h>  ]
╋'/AFF3h  [  <3AC3Ah>  <3B15Dh>  <3B18Fh>  <6557Ch>  <65632h>
            <7AA10h>  ]
╋'/B02Ah  [  <3AB09h>  <3B1DFh>  <3AAEBh>  <3AB09h>  <7AA3Ah>
            <7A999h>  ]
╋'/B061h  [  <3A5F0h>  <1F8D8h>  <1FCEBh>  <3A5F0h>  <7AA41h>
            <7A9F4h>  ]
╋'/B098h  [  <3AA0Ah>  <3B0DBh>  <3B081h>  <3B036h>  <3E5EEh>
            <3E5AFh>  ]
╋'/B0CFh  [  <65495h>  <3AE88h>  <3BE54h>  <65495h>  <3F167h>
            <3F178h>  ]
╋'/B106h  [  <6549Ch>  <3A0EDh>  <48FD6h>  <6549Ch>  <3F18Fh>
            <3F1A3h>  ]
╋'/B13Dh  [  <654A3h>  <3A859h>  <47B5Ah>  <654A3h>  <3F187h>
            <3F1C8h>  ]
╋'/B174h  [  <1AF05h>  <3A6CCh>  <3AC08h>  <6545Dh>  <3A6CCh>
            <65433h>  ]
╋'/B1ABh  [  <3A893h>  <3A8ACh>  <3A893h>  <3AA37h>  <3AA50h>
            <3AA37h>  ]
╋'/B1E2h  [  <65480h>  <3A7A3h>  <47AE7h>  <65480h>  <7AA56h>
            <7AA17h>  ]
╋'/B219h  [  <65487h>  <3AEB5h>  <3AEE2h>  <65487h>  <7AA1Eh>
            <7AA25h>  ]
╋'/B250h  [  <6548Eh>  <3AF05h>  <3AF1Eh>  <6548Eh>  <7AA2Ch>
            <7AA33h>  ]
╋'/B287h  [  <1A0EEh>  <3A8A4h>  <3A6B3h>  <6543Ah>  <3ABA4h>
            <6568Dh>  ]
╋'/B2BEh  [  <3A8C5h>  <3A8C5h>  <3A8ACh>  <3AA69h>  <3AA69h>
            <3AA50h>  ]
```

```
#7B2F5h  [  <6546Bh>  <3B1CBh>  <3B1B7h>  <6546Bh>  <7A953h>
            <7A94Ch>  ]
#7B32Ch  [  <65472h>  <3ACF8h>  <1A604h>  <65472h>  <6548Bh>
            <654C6h>  ]
#7B363h  [  <65479h>  <3ACBCh>  <3A6FEh>  <65479h>  <7A93Eh>
            <7A945h>  ]
#7B39Ah  [  <1AD09h>  <3ABD6h>  <3AC21h>  <6544Fh>  <3AB8Dh>
            <6542Ch>  ]
#7B3D1h  [  <3A5CDh>  <1A8BBh>  <3A9CEh>  <3A5CDh>  <1A8BBh>
            <3A9CEh>  ]
#7B408h  [  <65464h>  <1A8D8h>  <22FE8h>  <65464h>  <6548Fh>
            <65639h>  ]
#7B43Fh  [  <3A753h>  <3A778h>  <3ADD4h>  <3A753h>  <3A778h>
            <3ADD4h>  ]
#7B476h  [  <65686h>  <1AABDh>  <6564Eh>  <65686h>  <65671h>
            <6564Eh>  ]
#7B4ADh  [  <1AB67h>  <3AB8Bh>  <3ABEFh>  <65441h>  <3AB8Bh>
            <654AAh>  ]
#7B4E4h  [  <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>  ]
```

String

```
#0S5DFh  ""
#0E524h  ""
#0E5C6h  ""
#0FA69h  "9"
#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  " × "
#15E47h  "GROB  "
#15F23h  "C♯  "
#15FB5h  "C#  "
```

```
#161B2h  "XLIB  "
#16BB2h  "("
#16BEBh  ")"
#16C42h  "DIR"
#16CEDh  "END"
#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  "RDM"
#218C6h  "SYSRAM"
```

```
#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"
#29705Ah  "Invalid  Expression"
#2DF88h  ":␣"
#2E4F0h  "␣␣"
#2E877Bh  "F"
#2E8877h  "G"
#2E8B1h  "R"
#2F162h  "%%HP:"
#31D26h  "    "
#32341h  " "
#34105h  "+  "
#34115h  "-  "
#3412Fh  "?  "
#39771Dh  "HALT"
#3977D9h  "!USR"
#3977EBh  "USER"
#3982Ah  "ALG"
#39862h  "PRG"
#3998Ah  " }"
#39F09h  "1:  "
#39F28h  "   "
#39FF2h  "..."
#3B29Dh  "PARTS"
#3B2C0h  "PROB"
#3B2E1h  "HYP"
#3B300h  "MATRX"
#3B323h  "VECTR"
#3B346h  "BASE"
#3B55Bh  "STK"
#3B577Ah  "OBJ"
#3B599h  "DSPL"
#3B5BAh  "CTRL"
#3B5DBh  "BRCH"
#3B5FCh  "TEST"
#3B65Eh  "DRPN"
#3B77ECh  "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"
#3CB7AAh  "EDITΣ"
#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  "Å"
#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"
#3D339h  "in^3"
#3D34Bh  "l"
#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"
#3D7A6h  "Pa"
#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  "ren"
#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  "NKEQ"
#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  "▶"
#44243h  ":"
#46017h  "EDIT"
#4604Ch  "→WID"
#4606Dh  "WID→"
#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  "▶"
#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  "Extrn"
#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  "RF"
#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  "[ "
#65216h  "["
#65176h  "("
#65182h  ")"
#6518Eh  "#"
#6519Ah  "_"
#651A6h  "$"
#651B2h  "&"
#651BEh  " "
#651CAh  ">"
#651D6h  "<"
#651E2h  "E"
#651EEh  "≤"
#651FAh  "Σ"
#65206h  "∫"
```

```
#65212h  "                "
#65238h  "␣"
#65244h  "der"
#65254h  " "
#65260h  "UNKNOWN"
#652?8h  "'"
#65284h  "'"
#65290h  ","
#6529Ch  "."
#652A8h  "¡"
#652B4h  "("
#652C0h  ")"
#652CCh  "^"
#652D8h  "•"
#652E4h  "/"
#652F0h  "+"
#652FCh  "-"
#65308h  "="
#65314h  "ſ"
#65320h  "ð"
#6532Ch  "GROB"
#6533Eh  "C$"
#6534Ch  "0"
#65358h  "1"
#65364h  "2"
#653?0h  "3"
#653?Ch  "4"
#65388h  "5"
```

```
#65394h  "6"
#653A0h  "7"
#653ACh  "8"
#653B8h  "9"
#656C5h  "R<<"
#656D5h  "R<Z"
#656E5h  "XYZ"
#656F5h  "<>"
#65?03h  "()"
#65?11h  "[]"
#65?1Fh  "''"
#65?2Dh  "::"
#65?3Bh  "()"
#65?49h  "***"
#65?5?h  "ECHO"
#65?69h  "EXIT"
#65??Bh  "Undefined"
#65?9?h  "RAD"
#65?A?h  "GRAD"
#6?365h  "d"
#69692h  "  RATIO  "
#6A5??h  "RULES"
#6A59Fh  "EDIT"
#6A5C0h  "EXPR"
#6A5E1h  "SUB"
#6A600h  "REPL"
#6B413h  "NOT  "
```

Binary Integer

```
#10E34h  #0000000000000000h
#10E4Eh  #0000000100000000h
#1A215h  #0000h
#1A4?1h  #0526260410h
#1A9F9h  #8010h
#1AC?5h  #70107h
#1AEDEh  #80108h
#1B013h  #0504108h
#1B104h  #90109h
#1B113h  #0504109h
#1B3E?h  #C010h
#1BB2Ah  #010Ah
#1BE84h  #80108h
#1E?CEh  #50105h
#1E854h  #40104h
#1E8CBh  #5010h
```

```
#1EA21h  #60106h
#1F00Eh  #0524310h
#1F241h  #053626060410h
#1F319h  #052606073410h
#1F40Ch  #05B0734122h
#1F523h  #052410h
#1F5D9h  #059063410h
#1FB1Dh  #72109h
#26DD2h  #0590410h
#26DE3h  #050410h
#26DF3h  #05060410h
#26E05h  #0506060410h
#26E19h  #050606060410h
#26E2Fh  #05060606060410h
#26E4?h  #00700h
```

List

```
#05SE9h  { }
#0BSA8h  { }
#0E47Sh  { M N }
#0FAS3h  { '1_kg' '1_m' '1_A' '1_s' '1_K' '1_cd' '1_mol'
           '1_?' }
#101D5h  { }
#10B5Eh  { }
#10B68h  { }
#10B72h  { }
#10B7Ch  { }
#10B86h  { }
#14396h  { 'halt }
#155EFh  { 'nohalt }
#19A91h  { 0 "" 0 }
#19EFAh  { 4954521600 707788800 29491200 491520 8192 1 }
#19F6Bh  { " ueek(s)" " day(s)" " hour(s)" " minute(s)"
           " second(s)" " ticks" }
#1F960h  { 'num }
#1FF2Fh  { "Intercept" "Slope" }
#221F4h  { "none " "odd " "even " "mark " }
#2234Dh  { 1200 2400 4800 9600 }
#223C9h  { 1 2 3 }
#22400h  { 0 1 2 3 4 }
#22441h  { 0 1 2 3 }
#23754h  { 'noname 'stop }
#23879h  { 'ioinprogress }
#23903h  { st ofs tok }
#23989h  { }
#24A28h  { i j }
#25699h  { <2h> <8h> <Ah> <10h> }
#258EEh  { }
#25A06h  { '1 '2 '3 }
#272C8h  { 'ttt 'str 'ofs 'tok 'rbv 'idfflg 'tmpop 'tmppdat
           'ploc 'bv 'unbound }
#28077Ah  { }
#281D3h  { i * + }
#28BB4h  { ( ) }
#2C756h  { 1 2 0 0 LINFIT }
#2D4DBh  { }
#2D5F5h  { 'PACKET 'RETRY 'MaxR }
#2D839h  { 'KP 'PKNO }
#2D86Bh  { 'LNAME 'RETRY 'KMODE 'KRM }
#2DD56h  { 'ERRMSG }
#2DF01h  { 'LNAME 'KMODE 'KRM }
#2E6CDh  { 'KP 'PKNO }
#2E8E5h  { 'KP 'PKNO }
#2E90Dh  { 'LNAME 'KMODE 'KRM }
#2E99Eh  { 9600 0 0 0 3 1 }
#2E9F0h  { }
#2EED3h  { 'LNAME 'OBJ 'PACKET 'KRM }
```

```
#2F1FDh  {  'KLIST  'DPOS  'KML  }
#2F6F0h  {  'RETRY  }
#31C32h  {  'IWrap  }
#31CA9h  {  1  }
#31F4Ah  {  1.8  "  80  "␣"  }
#32F9Fh  {  'nohalt  }
#3304Eh  {  <Eh>  }
#33306Ch  {  }
#34D2Bh  {  }
#36796h  {  <3h>  }
#368CCh  {  3  }
#36BF1h  {  #a  #b  }
#36CEAh  {  #b  }
#36D5Ah  {  }
#36D82h  {  }
#38A39h  {  'SavedUI  }
#3B298h  {  "PARTS"  {  ABS  SIGN  CONJ  ARG  RE  IM  MIN  MAX  MOD  %
            %CH  %T  MANT  XPON  IP  FP  FLOOR  CEIL  RND  TRNC  MAXR
            MINR  }  }
#3B2DCh  {  "HYP"  {  SINH  ASINH  COSH  ACOSH  TANH  ATANH  EXPM
            LNP1  }  }
#3B2FBh  {  "MATRX"  {  CON  IDN  TRN  RDM  DET  RSD  ABS  RNRM  CNRM  }
            }
#3B36Ch  {  ABS  SIGN  CONJ  ARG  RE  IM  MIN  MAX  MOD  %  %CH  %T  MANT
            XPON  IP  FP  FLOOR  CEIL  RND  TRNC  MAXR  MINR  }
#3B420h  {  SINH  ASINH  COSH  ACOSH  TANH  ATANH  EXPM  LNP1  }
#3B452h  {  CON  IDN  TRN  RDM  DET  RSD  ABS  RNRM  CNRM  }
#3B556h  {  "STK"  {  OVER  ROT  ROLL  ROLLD  PICK  DEPTH  DUP  DUP2
            DUPN  DROP2  {  "DRPN"  DROPN  }  }  }
#3B575h  {  "OBJ"  {  OBJ→  EQ→  →ARRY  →LIST  →STR  →TAG  R→C  C→R
            DTAG  →UNIT  TYPE  VTYPE  SIZE  POS  REPL  SUB  NUM  CHR
            PUT  GET  PUTI  GETI  }  }
#3B5F7h  {  "TEST"  {  AND  OR  XOR  NOT  SAME  TYPE  ==  ≠  <  >  ≤  ≥  SF
            CF  FS?  FC?  FS?C  FC?C  }  }
#3B622h  {  OVER  ROT  ROLL  ROLLD  PICK  DEPTH  DUP  DUP2  DUPN
            DROP2  {  "DRPN"  DROPN  }  }
#3B659h  {  "DRPN"  DROPN  }
#3B67Fh  {  OBJ→  EQ→  →ARRY  →LIST  →STR  →TAG  R→C  C→R  DTAG  →UNIT
            TYPE  VTYPE  SIZE  POS  REPL  SUB  NUM  CHR  PUT  GET  PUTI
            GETI  }
#3B90Eh  {  AND  OR  XOR  NOT  SAME  TYPE  ==  ≠  <  >  ≤  ≥  SF  CF  FS?
            FC?  FS?C  FC?C  }
#3B972h  {  PR1  PRST  PRSTC  PRLCD  PRVAR  CR  DELAY  OLDPRT  }
#3BC8Dh  {  ASN  STOKEYS  RCLKEYS  DELKEYS  MENU  CST  TMENU
            RCLMENU  STOF  RCLF  SF  CF  FS?  FC?  FS?C  FC?C  }
#3BCE7h  {  MEM  BYTES  VARS  ORDER  PATH  CRDIR  TVARS  PVARS  NEWOB
            LIBS  ATTACH  DETACH  MERGE  FREE  ARCHIVE  RESTORE
            PGDIR  }
#3BD46h  {  STO+  STO-  STO*  STO/  INCR  DECR  SINV  SNEG  SCONJ  }
#3C03Eh  {  FUNCTION  CONIC  POLAR  PARAMETRIC  TRUTH  BAR  {
            "HIST"  HISTOGRAM  }  SCATTER  }
#3C061h  {  "HIST"  HISTOGRAM  }
#3C483h  {  COLCT  EXPAN  ISOL  QUAD  SHOW  TAYLR  ↑MATCH  ↓MATCH  |
            APPLY  QUOTE  →Q¤  }
```

```
#3C9BDh  {  "→DATE"  →DATE  }
#3C9E0h  {  "→TIME"  →TIME  }
#3CD9Bh  {  {  "LIN"  LINFIT  }  {  "LOG"  LOGFIT  }  {  "EXP"  EXPFIT
            }  {  "PWR"  PWRFIT  }  {  "BEST"  BESTFIT  }  }
#3CDA0h  {  "LIN"  LINFIT  }
#3CDBFh  {  "LOG"  LOGFIT  }
#3CDDEh  {  "EXP"  EXPFIT  }
#3CDFDh  {  "PWR"  PWRFIT  }
#3CE1Ch  {  "BEST"  BESTFIT  }
#3D096h  {  "m"  "cm"  "mm"  "yd"  "ft"  "in"  "Mpc"  "pc"  "lyr"
            "au"  "kn"  "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"  }
#3DAF2h  {  CONVERT  UBASE  UVAL  UFACT  →UNIT  }
#4132Dh  {  "0"  "1"  "2"  "&"  }
#433DBh  {  ""  <0h>  <1h>  <1h>  <2h>  <0h>  }
#45716h  {  <1h>  <1h>  }
#47B73h  {  9  15  }
#47BBEh  {  3  15  }
#49199h  {  }
#4A7CFh  {  LINFIT  LOGFIT  EXPFIT  PWRFIT  }
#4C93Fh  {  'xmax  'N  }
#4CF50h  {  'EnvOK  'EXITFCN  }
#4FF98h  {  'xe  'ye  'x  'y  'xc  'yc  'r2  'left  'up  'exit  }
#50D39h  {  'PlotEnv  }
#52D26h  {  }
#5456Ah  {  'tcls  'fcls  }
#547ABh  {  }
#549CCh  {  'dvar  }
#54DCBh  {  'xSYMfcn  'xfcn  }
#55666h  {  'oth  }
#557EEh  {  'scl  'xSYMfcn  'xfcn  }
#557FBh  {  'xSYMfcn  'xfcn  }
```

```
#569?1h  {  'sumexpr  'sumvar  }
#56BCDh  {  2  ^  /  }
#56E43h  {  ¥  180  /  *  }
#56E61h  {  ¥  200  /  *  }
#56EA2h  {  180  ¥  /  *  }
#56EC0h  {  200  ¥  /  *  }
#56EFCh  {  'dv  }
#5?1F2h  {  'dv  'op  'nm  }
#5?6F8h  {  ¥  i  *  }
#5??2Fh  {  2  ¥  *  i  *  }
#5?8CEh  {  'ni  'ns  }
#5?EA3h  {  '*s  }
#580F9h  {  '+s  }
#58DB1h  {  'fl  }
#59110h  {  'nmls  }
#592BBh  {  'c  'b  'a  }
#59512h  {  'n  'prog  }
#59641h  {  'n  }
#59919h  {  1  +  1  -  }
#5A60Fh  {  'piflag  }
#5A660h  {  'd  'r  }
#5A?5Ch  {  'd  'R  'est  'X  'T  }
#5AAE0h  {  'bnds  'dvar  }
#5D6?8h  {  'which  'op1  'op2  }
#5DD42h  {  2  ^  -  ∫  i  }
#5DE1Eh  {  ¥  /  *  }
#5FDBCh  {  'ct  'pp  'ep  }
#600F3h  {  }
#60806h  {  'reg  'sur  'cts  'sun  'nlg  'ckd  'prd  'prp  'rhs  }
#60BE4h  {  'patternls  'compos  'varls  }
#60E?3h  {  (  )  (  )  (  )  (  )  &1  &2  &3  &4  }
#6419Ah  {  <13Eh>  <123h>  <DFFh>  }
#68484h  {  }
#69A92h  {  'Radix  'KeysOK?  'ExprLit  'BuffW  'BuffH  'SaveBlank
            'ManOp  'nohalt  'AppMode  'NameGrab  'EXITFCN
            'FontGauge  'LE  'LB  'TE  'FormEnvOK  'prow  'pcol
            'cursy  'cursx  'ttt  'source  'ofs  'tok  'rbv  'idfflg
            'tmpop  'tmppdat  'ploc  'bv  'unbound  }
```

Unit

```
#0FA58h  '1_kg'
#0FA84h  '1_m'
#0FAA4h  '1_A'
#0FAC4h  '1_s'
```

```
#0FAE4h  '1_K'
#0FB04h  '1_cd'
#0FB26h  '1_mol'
#0FB4Ah  '1_?'
```

Graphic

```
#13DB4h  GROB  6  10  F1F1F1F1F1F1F1F1F100

#39B2Dh  GROB  131  2
         FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0
         00000000000000000000000000000000000
#3A337h  GROB  21  8
         FFFFFF0000000000000000000000000000000
         0000000000000
#3A399h  GROB  21  8
         FFFFFF0000000000000000E00000E00000E
         0000000000000
#3A3FBh  GROB  21  8
         1CFFFF0000000000000000000000000000000
         0000000000000
#3A45Dh  GROB  21  8
         FFFFFF1000011000011000011000011000
         1100001FFFFFF
#505C3h  GROB  5  5  4040F14040
#505EAh  GROB  5  5  11A040A011
#5058Bh  GROB  0  0
#66EA5h  GROB  6  10  F1111111111111F100

#66ECDh  GROB  6  8  F11111111111F100

#66EF1h  GROB  4  6  F090909090F0

#66F11h  GROB  6  8  0000000000000000
```

```
#66F35h  GROB  6  10  00000000000000000000
#66F5Dh  GROB  7  5  F7778605F7F7
#66F7Dh  GROB  5  4  F1B151F1
```

Global Name

```
#0DF01h  'Alarms'
#0DF28h  'Alarms'
#15776Ch  'EQ'
#157781h  ''
#19A72h  'ALRMDAT'
#19B1Fh  'ALRMDAT'
#19DBEh  'ALRMDAT'
#211B4h  'CST'
#225A4h  'S'
#2C1FDh  'ΣDAT'
#2C738h  'ΣPAR'
#2E9D5h  'IOPAR'
#2EA59h  'IOPAR'
#31F87h  'PRTPAR'
#31FB8h  'PRTPAR'
#34DBBh  ''symb'
#3FACFh  'SKEY'
#4093Bh  'αENTER'
#409DFh  'βENTER'
#41A43h  'UserKeys'
#41A69h  'UserKeys.CRC'
#41BD7h  'S'
#43536h  'ALG'
#4358Ah  'α'
#435CEh  'V'
#47459h  'X'
#48D4Bh  'ALRMDAT'
#4A145h  'X'
#4A19Eh  'X'
#4A1DEh  'X'
#4A22Dh  'X'
#4A25Eh  'X'
#4AB1Ch  'X'
#4AB59h  'Y'
#50FCEh  'X'
#50FE6h  'Y'
#51288h  'PPAR'
#51436h  'si'
#56859h  'IERR'
#57793Fh  'n0'
#57795Dh  's0'
#59304h  'si'
```

Local Name

```
#0E477Ah  'M'
#0E483h  'N'
#0E4A0h  'M'
#0E4AEh  'N'
#0E4C1h  'M'
#1439Bh  ''halt'
#14483h  ''nohalt'
#1F96Fh  ''num'
#1F977Eh  ''fcn'
#23772Eh  ''stop'
#23773Fh  ''noname'
#23877Eh  ''ioinprogress'
#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  ''rbu'
#27318h  ''idfflg'
#2732Dh  ''tmpop'
#27340h  ''tmppdat'
#2735Fh  ''ploc'
#27368h  ''bu'
#2737Eh  ''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'
#54560h  ''tcls'
#54580h  ''fcls'
#54600h  ''tcls'
#54624h  ''fcls'
#5465Dh  ''tcls'
#5466Eh  ''fcls'
#549DBh  ''dvar'
#54DD0h  ''xSYMfcn'
#54DE7h  ''xfcn'
#55660h  ''oth'
#55783h  ''scl'
#55792h  ''xSYMfcn'
#557A9h  ''xfcn'
#55800h  ''xSYMfcn'
#55817h  ''xfcn'
#56976h  ''sumexpr'
#56980h  ''sumvar'
#56F0Bh  ''dv'
#57208h  ''nn'
#57218h  ''op'
#578E2h  ''ni'
#578EFh  ''ns'
#57EF3h  ''+s'
#58149h  ''+s'
#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  ''uhich'
#5D690h  ''opl'
```

```
#5D69Fh  ''op2'
#5FDC1h  ''ct'
#5FDCEh  ''pp'
#5FDDBh  ''ep'
#6080Bh  ''reg'
#6081Ah  ''sur'
#60829h  ''cts'
#60838h  ''sun'
#6084'/h  ''mlg'
#60856h  ''ckd'
#60865h  ''prd'
#608'/4h  ''prp'
#60883h  ''rhs'
#60BE9h  ''patternls'
#60C04h  ''compos'
#60C19h  ''varls'
#60C4Fh  ''patternls'
#60C6Ah  ''compos'
#60CCAh  ''compos'
#60D'/Fh  ''varls'
#60E8Ch  '&1'
#60E9'/h  '&2'
#60EA2h  '&3'
#60EADh  '&4'
#61D3Ah  ''
#69A9'/h  ''Radix'
#69AAAh  ''KeysOK?'
#69AC1h  ''ExprLit'
```

```
#69AD8h  ''BuffW'
#69AEBh  ''BuffH'
#69AFEh  ''SaveBlank'
#69B19h  ''ManOp'
#69B2Ch  ''nohalt'
#69B41h  ''AppMode'
#69B58h  ''NameGrob'
#69B'/1h  ''EXITFCN'
#69B88h  ''FontGauge'
#69BA3h  ''LE'
#69BB0h  ''LB'
#69BBDh  ''TE'
#69BCAh  ''FormEnvOK'
#69BE5h  ''prou'
#69BF6h  ''pcol'
#69C0'/h  ''cursy'
#69C1Ah  ''cursx'
#69C2Dh  ''ttt'
#69C3Ch  ''source'
#69C51h  ''ofs'
#69C60h  ''tok'
#69C6Fh  ''rbu'
#69C'/Eh  ''idfflg'
#69C93h  ''tmpop'
#69CA6h  ''tmppdat'
```

```
#69CBDh  ''ploc'
#69CCEh  ''bu'
#69CDBh  ''unbound'
```

Glossaire

Adresse	Entier compris entre 0 et FFFFF (hexadécimal) représentant l'endroit où se trouve une information donnée.
Assemblage	Action d'assembler.
Assembler	Traduire un programme-assembleur en un programme en langage-machine.
Assembleur	Langage symbolique représentant le langage-machine, programme réalisant la traduction d'un programme-assembleur en langage machine.
Bank-switching	Technique consistant à faire cohabiter deux mémoires à la même adresse. L'une des deux est visible, l'autre est cachée. Pour accéder à la mémoire cachée, on change l'adresse de la mémoire visible.
BCD	Terme anglais pour DCB.
Bit	Mémoire prenant la valeur 0 ou 1 et constituant les quartets.
Buffer	Zone-mémoire destinée au stockage temporaire d'informations en attente de traitement (par exemple les codes des touches frappées, ou les données provenant de l'interface RS232c).
Byte	Terme anglais pour octet.
Champ	Partie d'un registre.
DCB	Décimal Codé Binaire: manière de stocker un nombre décimal en le représentant par le nombre hexadécimal identique (ex: 20, décimal, sera représenté par 20h [hexadécimal] qui vaut en fait 32 [décimal]).

Desassemblage	Action de désassembler.
Desassembler	Traduire un programme-machine en un programme-assembleur.
Desassembleur	Programme réalisant un désassemblage.
Drapeau	Mémoire ayant la valeur 0 ou 1 et servant d'indicateur.
Garbage Collector	Opération réalisée dès que la machine ne dispose plus d'assez de place-mémoire. Cette opération consiste en la destruction des objets temporaires devenus inutiles. On peut déclencher le garbage-collector en utilisant la commande MEM.
Hexadécimal	Base 16 (les "chiffres" sont: 0 1 2 3 4 5 6 7 8 9 A B C D E F).
Indicateur	Un des symboles pouvant s'allumer au sommet de l'afficheur de la HP48 et indiquant l'état de la machine (Rad, Alpha...), plus généralement mémoire ou symbole pouvant prendre 2 états.
Kilo-Octet	Unité de mesure de la taille d'une mémoire. Correspond à 1024 (2^{10}) octets.
Langage machine	Suite de codes représentant une suite d'instructions élémentaires compréhensibles par le microprocesseur.
LCD	Liquid Crystal Display, afficheur à cristaux liquides.
Mémoire	Organe de stockage de l'information. Voir Ram et Rom.
Mémoire morte	Voir Rom.
Mémoire vive	Voir Ram.
Objet	Tout élément avec lequel travaille le RPL. Par exemple: un réel.

Octet	Information sur 8 bits, unité de mesure de la taille d'une mémoire.
Peek	Instruction ou programme servant à lire le contenu d'une ou plusieurs cases-mémoire situées à partir d'une adresse donnée.
Pile	Source d'énergie de la HP48 mais aussi système de stockage temporaire de données dont le contenu est affiché sur la partie centrale de l'écran et sur lequel est basé le principe de la notation polonaise inversée.
Poke	Instruction ou programme servant à écrire une ou plusieurs cases-mémoire situées à partir d'une adresse donnée.
Processeur	Voir microprocesseur.
Prologue	Groupe de 5 quartets identifiant l'objet débutant par ces quartets.
Quartet	Case-mémoire élémentaire de 4 bits (valeur comprise entre 0 et 16 [décimal], entre 0 et F [hexadécimal]).
RAM	Random Acces Memory ou mémoire-vive: circuit électronique pouvant stocker des informations (modifiables), zone-mémoire contenant des informations modifiables.
Registre	Une mémoire du microprocesseur. Ne contient que des entiers positifs ou nuls.
ROM	Read Only Memory ou mémoire morte: circuit électronique stockant des informations (non modifiables), zone-mémoire contenant des informations non-modifiables (la Rom contient les programmes machine des instructions RPL).
RS232c	Standard d'échange de données.

Index

@ 64
?ADR 226
π 287
µSOLVER 49, 298

0

02911 (prologue) 121, 122, 150
02933 (prologue) 121, 123
02955 (prologue) 121, 124
02977 (prologue) 121, 125
0299D (prologue) 121, 126
029BF (prologue) 121, 127
029E8 (prologue) 121, 128, 147
02A0A (prologue) 121, 130
02A2C (prologue) 121, 131, 147
02A4E (prologue) 121, 132, 145, 148
02A74 (prologue) 121, 133
02A96 (prologue) 121, 134, 136
02AB8 (prologue) 121, 138
02ADA (prologue) 121, 139
02AFC (prologue) 121, 141
02B1E (prologue) 121, 142
02B40 (prologue) 121, 143
02B62 (prologue) 121, 149, 150
02B88 (prologue) 121, 151
02BAA (prologue) 121, 152
02BCC (prologue) 121, 152
02BEE (prologue) 121, 152
02C10 (prologue) 121, 152
02D9D (prologue) 121, 153
02DCC (prologue) 121, 154
02E6D (prologue) 121, 156
02E48 (prologue) 121, 155
02E92 (prologue) 121, 157
0312B (épilogue) 133, 138, 139, 153

A

A 75, 76
A->STR 262
A->V 290
ADD 267
Additions 81, 95
Adresse 75, 64, 226, 405
Adresse alarmes 194
Adresse bitmap ecran 184
Adresse bitmap menu 184
Adresse de fin de ram 194
Adresse de fin des objets temporaires 184
Adresse de la ligne de commande 184
Adresse de la pile d'undo et des variables locales 184
Adresse sauvegardes 184
Adresse directory courant 184
Adresse Error message 194
Adresse fin mémoire libre 184
Adresse GROB courant 184
Adresse GROB de la pile 184
Adresse GROB du menu 184
Adresse GROB graphique 184
Adresse home directory 184
Adresse memoire libre 184
Adresse message d'erreur 197
Adresse objet à exécuter 194
Adresse objets temporaires 184
Adresse temporary environment 184
Adresse userkeys 194
Adresse bitmap écran 171
Adresse de base de la ram interne 164, 169
Adresse Hash-Table 134
Adresse Liste Messages 134
Adresse d'un objet situé au niveau n 188
Adresses d'objets structurés 157
Affectations 81, 82
Alert 164
Algebraic 121, 138
Alpha 164
ANAG 313
Appels absolus 82, 110
Appels de sous-programmes 82, 109
Appels relatifs 82, 110
Arborescence des menus 29
Areuh 69
Array 121, 128, 387
ASN 48
Assemblage 68, 405
Assembler 405
Assembleur 67, 68, 405
Assignation de fonctions 47
Auto-test fail time 174
Auto-test start time 174
AUTOST 328
Autotest start & fail time 177

B

B 75, 76, 173, 184, 187, 208
B->SB 260
BACKUP 52, 121, 144, 149, 193
Bank-switching 159, 405
BANNER 334
Base 351
Batteries 176
BCD 405
BEEP 71
BFACT 267
Binaire 352
Binary Integer 121, 132, 395
Bit 69, 352, 405
Bitmap des menus 170, 185
Booléen 352
Buffer 178, 180, 405
Buffer clavier 182
Buffer RS232c 179
BufFull 178
BufLen 178
BufStart 178
Busy 164
Buzzer 72
Byte 75, 405

C

C 75
CAL 329
CALC 267
Calcul numérique 56
Calcul symbolique 55
Calculateur de codes de redondance cyclique 164
Calculateur de CRC 165
Calcul binaire 352
Calcul de fonction 56
Caractère 127, 131
CARRE 315
CARRY 74
Carte présente dans un des deux port 167
Carte mémoire 51, 159, 176, 178, 179
Champ 75, 76, 405
Character 121, 127, 386
Chargements de constantes 81, 82
Chargements de valeurs 81, 85
Checksum 144, 165, 175, 266
CHK 230
CIRCLE 331
Clavier 15, 178
CLEAN 216
CLVAR 238
CMOS word 174, 175
Codage des touches 183
Code 121, 154, 203
Code de contrôle 150
Code de redondance cyclique 144, 149, 150, 166, 175, 266
Codes en langage-machine 157
COMA 176
Commandes du Bus 82, 117
Commentaires 49
Comparaisons avec des constantes 82, 111
Comparaisons de registres 82, 113
Complémentations à 2 81, 102
Complex 121, 125, 385
Config Object 143
Configuration 144
CONTRAST 165, 240
Contraste 164, 165
Correspondance de bases 354
CRDIR 31
CRNAME 237
CST 46

D

D 75, 173, 187, 208
D0 74, 75, 194, 207
D1 74, 75, 173, 184, 187, 188, 207
Décimal codé binaire 69, 123, 405
Décrémentations 81, 96
DER 291
Dérivation 55
DESASS 243
Desassemblage 68, 406
Desassembler 406
Desassembleur 406
Développement limité 55
Directory 121, 134
Directory courant 193
Display 164, 165
DISPOFF 241
DISPON 241
DIV 267
Divisions par 2 81, 103
Divisions par 16 81, 105
DIVP 292
Données 27
Drapeau 406
Drapeaux 178, 198, 199
Drapeaux système 200
Drapeaux utilisateur 200
DSP 319

E

E 267
Echanges de contenus de registres 81, 91
Ecran 16, 159, 160
EEPROM 51
Entier 132
Entrée infra-rouge 169
Entrée ok 168
Entrée RS232c 164, 168
Entrées-sorties 159, 163
Epilogue 133, 138, 139, 153
EPROM 51
Erreur 176, 357
Error number 198, 199
"Et" logiques 81, 99
Exponent 76
Exponent-Sign 76
Exposant 123, 124, 125, 126
External module missing 74

F

Factorielle 2000 286
FAST 242
Fin de mémoire vive 197
F 76

G

Garbage collector 185, 186, 406
GASS 206, 213
Gérer la pile 22
Gestion du temps 57
Global Name 121, 155, 401
Graphic 121, 142, 400
Graphisme 56
GROB du caractère sous le curseur 199
GROBS écran 173

H

Hardware internal Design Specifications 77
Hash Table 134, 135, 143 à 145
Hauteur menu 170,171, 184, 185
Hexadécimal 69, 353, 406
HOME 30, 31
Home directory 193
Horloge 159, 170, 172, 174, 176
HP71 71
HRPEEK 222
HST 74

I

Icônes 47
Incrémentations 81, 94
Indicateur 406
Indicateurs 164, 166
Informations sur les cartes enfichées 180
Informations sur le contenu des ports 164, 167
Infra Rouge 54, 159, 176, 169
Inhibition arrêt-système 174
Inhibition de l'arrêt-système 175
INPUT 71
Instructions de contrôle 82, 118
Instructions sans effet 82, 119
Interruptions RS232c 164
Interruption 78, 168, 178
IR 54, 164
IR in 164
IR in mem. 164

J

JINGLE 326

K

KERMIT 53
KeyEnd 178, 182
KeyStart 178, 182
Kilo-Octet 406

L

LABY 302
LAGU 294
Langage machine 406
Langage 67
Langage évolué 67
LAST 194
LCD 406
Left Shift 164
Librairie 134
Library 121, 135, 143
Library Data 121, 151
Library number 143
Ligne de commandes 173, 188
Link-table 143, 144, 148
Linked Array 121, 130
LISP 33
Lisp Polonais Inversé 33
List 121, 133, 396
Liste d'alarmes 176
Liste Messages 134
LOAD_REG 208
Local Name 121, 156, 401
Long Complex 121, 126, 385
Long Real 121, 124, 385

M

M 75, 76
Manipulations du compteur de programme 82, 109
Mantisse 75, 123, 124, 125, 126
Marge 164, 165, 170
Marge à droite 171, 184, 185
Masque de sortie pour le test clavier 181
MASTER 309
MEM 27
Mémoire libre 173, 187, 194, 197
Mémoire 159, 406
Mémoire cachée 160
Mémoire morte 159, 263, 406
Mémoire vive 263, 406
Mémoire vive interne 159
Menu 17, 27
Menu "VAR" 30
MERGE 52
Message Array 143
Message 147
Message-table 144
Messages d'erreurs 357
Méthodes de programmation 34
Microprocesseur 69
Mini-éditeur de mémoire 160
MODU 268
MODUL 323
Module 176
Module pulled 74
MODUSEARCH 263
MP 74
MULT 267
Multiplications par 2 81, 102
Multiplications par 16 81, 104
MUSICLM 321

N

NEXT 27
Nombre de librairies attachées 134
Nombre de messages-tables attachées 201
Nombre de table de messages attachées 198
"Non" logiques 81, 101
Notation polonaise inversée 19
Notes diverses 77
NOTHING 211
Null 189
Numéro de la librairie 135

O

Objet 406
Objet à exécuter 196
Objets temporaires 173, 185
Objet de configuration 144
Objet sauvegardé 149
Objets 33, 121
Objets graphiques 185
Octet 353, 407
Offset d'horloge 177
Opérations mathématiques 81, 94
OUTPUT 71
"Ou" logiques 81, 100

P

P 76
PAR5 214
PATH 31
PC 74
PCAR 293
PEEK 218, 407
Périphériques 159
PI 288
Pile d'undo et variables locales 173
Pile de LAST 196
Pile des lignes de commandes 196
Pile 16, 19, 173
Pile des retours 186
Pile d'undo 189
Pile utilisateur 187
Pixel 142
PMAT 297
Pointeur de champs 71, 75
Pointeurs 71, 74
POKE 220, 407
Port 180
Port 0 184
Port 1 160, 178
Port 2 160, 178
POW 267
PR40 318
Préparation affichage 174, 177
PREVIOUS 27
Processeur 407
Prochaine erreur à afficher 198, 199
Program 121, 153
Programme 34
Prologue 121, 407
PROM 51
Pseudo-opérations 82, 119

Q

Quartet 353, 407

R

R0 74
R1 74
R2 74
R3 74
R4 74
RABIP 325
Racine de génération aléatoire 198, 199
RAM 51, 52, 159, 407
RAMSEARCH 263
Ram réservée 173
Ram des entrées-sorties 163
Ram I/O 160
Ram interne 160, 169
RASS 228
Real 121, 123, 384
Recherche d'extrema 56
Récursivité 41
Redéfinition des touches 47
Registre 407
Registres 71
Registres d'entrées/sorties 71
Registres de calcul 71, 75
Registres de sauvegarde 71, 74
Registres utilisés par la HP48 78
Registres-drapeaux 71, 74
RENAME 327
Reserved 1 121
Reserved 1, 2, 3 et 4 152
Reserved 2 121
Reserved 3 121
Reserved 4 121
Reset 176
Résolution d'équation 56
Retours de sous-programmes 82, 110
Return stack 74, 173
REVERSE 235
Reverse Polish Lisp 33
Reverse Polish Notation 19
Right Shift 164
ROM 51, 159, 407
ROMRCL 261
ROMSEARCH 263
Rom cachée 169, 222, 263
Rotations des quartets vers la droite 81, 106
Rotations des quartets vers la gauche 81, 106
RPL 33
RPN 19
RS232c 53, 164, 167, 168, 178, 179
RSTK 74

S

S 75, 76
SATURN 71
Sauts 82, 107
Sauts absolus fixes 82, 108
Sauts absolus par valeur 82, 108
Sauts relatifs conditionnels 82, 108
Sauts relatifs inconditionnels 82, 107
Sauvegardes 173, 185
Sauvegardes et récupérations (Rn et RSTK) 81
Sauvegardes et récupérations (Rn et RSTK) 86
Sauvegardes pour les interruptions 181
SAVE_REG 208
SB 74
SEARCH 263
service request 74
Signe 75, 123, 124, 125, 126
Sortie infra-rouge 169
Sortie ok 168
Sortie RS232c 164, 168
Soustractions 81, 98
SQR 268
SR 74
SSAG 227
ST 74
StackSize 198, 199
Statistiques 56
STATUS 74
Sticky bit 74
STO 30
Stocker des données 30
STR->A 262
String 121, 131, 390
SUBS 267
Symbole Adresse (@) 64
SYSEVAL 239
System Binary 121, 122, 381

T

Table des messages 147
Tableaux décrivant la mémoire 64
TAG 49, 141
Tagged 121, 141
Tagguer 49
Temporary environment 173, 191
Tests 82, 111
Touche 72, 178, 183
Transferts avec l'extérieur 81, 90
Transferts avec la mémoire 81, 89
Transmitting 164

U

Unit 121, 139, 399
Unités 57
unité 139
UPDIR 31
Userkeys et alarmes 195

V

V->A 290
VAL 289
VAR 30
Variables utilisateur 173
Variable 40
Variable globale 39, 40
Variable locale 38, 40, 189, 190
Vitesse RS232c 166

W

W 75, 76
Wide 76
Wide-P 76
WP 76
WSLOG 160, 174, 175, 176

X

X 75, 76, 77
XLIB name 121, 157
XM 74
XS 75, 76

Z

Zone de sauvegardes 193

Notes personnelles

Ouvrage d'initiation, cours de perfectionnement, document de référence... "Voyage au centre de la HP48" est tout cela à la fois !
Cet ouvrage est donc l'outil idéal pour tous ceux qui désirent aller jusqu'au bout des possibilités offertes par cette petite machine... Possibilités qui sont bien supérieures à ce que l'on peut imaginer ! Que vous soyez lycéen, étudiant, ingénieur ou tout simplement informaticien dans l'âme, cet ouvrage vous entraînera à travers le monde merveilleux de la programmation avancée tout en vous offrant une vaste bibliothèque de programmes variés. Plus de 50 programmes sont au rendez-vous, parmi lesquels des programmes mathématiques performants (calcul en précision infinie, résolution de systèmes non linéaires...), des utilitaires, des exemples de programmes en langage assembleur...
Autorisée aux examens et concours, calculatrice de poche plus que performante, la HP48 devient, grâce aux explications contenues dans ce livre, un véritable ordinateur de poche, aux possibilités inégalées sous un si faible volume...

ISBN 9782878920031