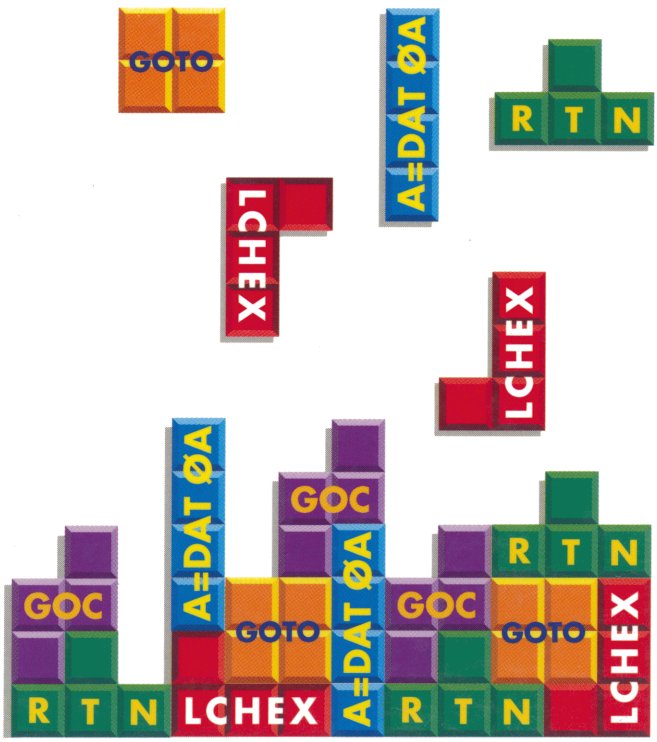
CALCULATRICES

Loïc Fieux

HP 48
faites vos jeux en assembleur



L'assembleur HP-48 enfin accessible à tous !

avec disk !

Le guide de référence

DUNOD

CALCULATRICES

Loïc Fieux

HP 48

faites vos jeux en assembleur

DUNOD

Ce livre n'est pas le manuel des calculatrices Hewlett-Packard HP-48. Le contenu de ce livre n'engage pas la société Hewlett-Packard, ni ses distributeurs.

Cet ouvrage est destiné aux utilisateurs des calculatrices Hewlett-Packard HP-48 versions S, SX, G et GX.

Ce pictogramme mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit, particulièrement dans le domaine de l'édition technique et universitaire, le développement massif du **photocopillage**.

Le Code de la propriété intellectuelle du 1er juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établissements d'enseignement supérieur, provoquant une baisse brutale des achats de livres et de revues, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.

Nous rappelons donc que toute reproduction, partielle ou totale, de la présente publication est interdite sans autorisation du Centre français d'exploitation du droit de copie (**CFC**, 3 rue Hautefeuille, 75006 Paris).

Ouvrage mis en pages par l'auteur

Illustrations de couverture : Rachid Maraï

© DUNOD, Paris, 1995

ISBN 2 10 002358 6

Code 042358

Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de l'auteur, ou de ses ayants droit, ou ayants cause, est illicite (loi du 11 mars 1957, alinéa 1er de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. La loi du 11 mars 1957 n'autorise, aux termes des alinéas 2 et 3 de l'article 41, que les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective d'une part, et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration.

Table des matières

Avant-propos 9
Avertissement 10

Première partie : initiation

1 • Vite ! Mon premier programme ! 13
1 • Quelques notions indispensables 13
2 • Mon premier programme... Vite ! 29
3 • Les structures de base 45
4 • Des fragments de jeux 60
5 • Quelques compléments 76

2 • Le binaire naturel 77
1 • La base 2 77
2 • Conversions 82
3 • Poids faible et poids fort 85
4 • Nombres rationnels 86

3 • Calculs en base 2 89
1 • L'addition binaire 89
2 • Soustraction et nombres négatifs 90
3 • Le complément à 2 92
4 • Multiplication binaire 97
5 • Division binaire 99
6 • Pour aller plus loin... 100

4 • Base 8, base 16 et codage des quartets 103
1 • La base 8 103
2 • La base 16 105
3 • Coder les quartets 108

5 • Le décimal codé binaire 111
1 • Le décimal codé binaire (DCB) 111
2 • Calculer en DCB 113

6 • Logique 115
1 • Fonctions logiques élémentaires 115
2 • Algèbre de Boole 119

7 • Le code ASCII 125
1 • L'ASCII 125
2 • Les codes ASCII 126

8 • Introduction au microprocesseur 131
1 • Introduction 131
2 • Fonctionnement du microprocesseur 135

9 • Introduction au SATURN 137
1 • Le SATURN et ses rôles 137
2 • Généralités et spécificités du SATURN 139
3 • Les registres du SATURN 142

10 • Configuration matérielle du SATURN 149
1 • Fonctionnement du SATURN 149
2 • La mémoire de la HP-48 152

11 • Les registres du SATURN et leurs champs 159
1 • Rappels et appellations 159
2 • Les champs des registres 161
3 • Remarques et astuces 162
4 • Les champs des registres de calcul 163

12 • Plages d'adresses des différentes mémoires 165
1 • Hiérarchie des mémoires 165
2 • HP-48 G 167
3 • HP-48 GX sans carte 168
4 • HP-48 GX (32 ko en port 1) 169
5 • HP-48 GX (128 ko en port 1) 170
6 • HP-48 GX (128 ko en ports 1 et 2) 171
7 • Remarques 172

13 • Introduction au jeu d'instructions 173

14 • Instructions : traitement des registres de travail 177
1 • Affectation d'une valeur 177
2 • Traitements 181

15 • Instructions : sauvegarde dans un registre et pointeurs 193
1 • Les registres de sauvegarde 193
2 • Les pointeurs 199

16 • Instructions : sauts et tests 209
1 • Les sauts 209
2 • Les tests 219

17 • Instructions : mode, registres spéciaux, interruptions, etc. 229
1 • Les registres spéciaux 229
2 • Modes, interruptions, bus, etc. 235

18 • Accès à la mémoire masquée 239
1 • La mémoire masquée 239
2 • Utiliser CONFIG et UNCNFG 240
3 • Les pointeurs étendus 245

Deuxième partie : références

19 • Les instructions classées par mnémoniques 249

20 • Les instructions classées par codes 269

21 • Les masques de clavier 289
1 • Les registres IN et OUT 289
2 • Les masques de touches 290

22 • Les objets en mémoire 291
1 • Introduction 291
2 • Binaire système 294
3 • Nombre réel 295
4 • Nombre réel double précision 296
5 • Nombre complexe 297
6 • Nombre complexe double précision 298
7 • Caractère 299
8 • Tableau 300

9 • Tableau indexé 302
10 • Chaîne de caractères 303
11 • Entier 304
12 • Liste 305
13 • Répertoire 306
14 • Expression algébrique 310
15 • Objet avec unité 311
16 • Objet « taggué » 312
17 • Objet graphique 313
18 • Bibliothèque 314
19 • Sauvegarde 320
20 • Données de bibliothèque 321
21 • Pointeur étendu 322
22 • Objets réservés 322
23 • Programme RPL 323
24 • Code exécutable 324
25 • Nom global 325
26 • Nom local 326
27 • Accès à une bibliothèque 327
28 • Primitives RPL 328
29 • Remarques importantes 329

23 • Structure de la mémoire vive des entrées/sorties 331
1 • Vue d'ensemble 331
2 • Adresses utiles de la RAM I/O 332

24 • La mémoire vive réservée au système 339
1 • Liste des champs 339
2 • Description de certains champs 353

25 • Quelques routines en mémoire morte 365
1 • Introduction aux routines 365
2 • Manipulation de registres 366
3 • La gestion du clavier 367
4 • Ecran et graphisme 369
5 • Opérations à partir de répertoires 371
6 • Applications mathématiques 372
7 • Gestion de la mémoire 374
8 • Horloge, coma, interruptions, etc. 378
9 • Transmission de données 380
10 • Bibliothèques et ports 381

26 • Codes WSLOG 385

Troisième partie : listings commentés

27 • Présentation des jeux 389
1 • Les jeux 389
2 • Exploitation des listings 390
3 • Présentation des listings 398
4 • Utiliser la disquette jointe 400

28 • Alpha pour HP-48 S et SX 403
1 • Le jeu 403
2 • Les objets 404
3 • Exécuter le programme 426
4 • Comment jouer ? 427
5 • Retrouver les fichiers sur la disquette 429

29 • Star pour HP-48 S et SX 431
1 • Le jeu 431
2 • Les objets 432
3 • Exécuter le programme 452
4 • Comment jouer ? 452
5 • Créer vos propres tableaux 453
6 • Retrouver les fichiers sur la disquette 454

30 • Xalk pour HP-48 S et SX 455
1 • Le jeu 455
2 • Les objets 456
3 • Exécuter le programme 482
4 • Comment jouer ? 482
5 • Retrouver les fichiers sur la disquette 483

31 • Seth pour HP-48 S et SX 485
1 • Le jeu 485
2 • Les objets 486
3 • Exécuter le programme 536
4 • Comment jouer ? 537
5 • Retrouver les fichiers sur la disquette 538

32 • Alpha pour HP-48 G et GX 539
1 • Le jeu 539
2 • Les objets 540
3 • Exécuter le programme 564
4 • Comment jouer ? 565
5 • Retrouver les fichiers sur la disquette 566

33 • Star pour HP-48 S et SX 567
1 • Le jeu 567
2 • Les objets 568
3 • Exécuter le programme 586
4 • Comment jouer ? 586
5 • Retrouver les fichiers sur la disquette 587

34 • Xalk pour HP-48 G 589
1 • Le jeu 589
2 • Les objets 590
3 • Exécuter le programme 616
4 • Comment jouer ? 616
5 • Retrouver les fichiers sur la disquette 617

35 • Seth pour HP-48 G 619
1 • Le jeu 619
2 • Les objets 620
3 • Exécuter le programme 670
4 • Comment jouer ? 670
5 • Retrouver les fichiers sur la disquette 671

36 • Structure des programmes 673
1 • Les listings pour HP-48 S et SX 673
2 • Les listings pour HP-48 G et GX 677
3 • Les listings pour HP-48 G 678

Bibliographie 681
Utilisation de la disquette 683
Les clubs d'utilisateurs 685
Glossaire 687
Index 695
Assistance technique 702

Avant-propos

Parce qu'elle exige une maîtrise totale d'une foule de notions, la programmation en langage assembleur est difficile. Ecrire un livre pour le débutant est donc une vaste entreprise. Il faut à la fois apporter au lecteur les très nombreuses informations dont il a besoin tout en l'encourageant. Cela explique la structure de la première partie de cet ouvrage. Nous avons choisi de guider la réalisation du premier programme en assembleur dès le premier chapitre. Celui-ci regroupe aussi des exemples de programme qui pourront être transposés lors de la réalisation de jeux. Le reste de la première partie est organisé comme un véritable cours. Toutes les bases indispensables sont traitées avant qu'il puisse être question de l'architecture du SATURN puis de son jeu d'instructions.

La deuxième partie de ce livre est consacrée aux indispensables références dont a besoin le programmeur en langage assembleur. Elle est certainement la moins originale puisque rien ne peut y avoir été inventé. Sa rédaction a été facilitée par la consultation des ouvrages cités dans la bibliographie (voir en fin d'ouvrage).

La troisième partie est consacrée à des listings de jeux commentés. Dès que le contenu de la première partie du livre est maîtrisé, vous pouvez entreprendre l'étude de jeux en assembleur. L'amélioration possible de ces programmes constitue un excellent exercice pour le débutant.

L'épaisseur de ce livre se justifie par notre désir de vous apporter le maximum d'explications. Cela dit, si vous rencontrez un problème, vous pouvez soumettre celui-ci au Club Calculatrices qui a mis en place un service d'assistance technique aux programmeurs (voir *Assistance technique* en fin d'ouvrage).

Bon courage à tous !

Avertissement

Afin de faciliter la lecture, le texte est balisé d'icônes :

Attention ! Ce pictogramme signale une erreur à éviter...

Chouette ! Une astuce !

Cet éléphant signale les paragraphes dont le contenu est à apprendre (une mémoire d'éléphant sera la bienvenue...).

CCD 20 signale un exemple pratique. Pourquoi CCD20 ? Parce que c'est le prologue inversé d'un objet de type code exécutable bien sûr !

Un petit commentaire...

Cet expert donne parfois un petit conseil. Il signale aussi des informations à réserver aux initiés.

Les noms des touches à presser sont placés entre crochets. Ainsi, [ENTER] signifie « Appuyez sur la touche nommée ENTER ».

Placez votre HP-48 en mode hexadécimal en tapant [MTH] [F] [A].

Ni l'auteur, ni l'éditeur ne pourront être tenus responsables des dommages liés à une utilisation détournée de ce livre (en particulier, destruction de l'écran par blocage du balayage).

Quelques notations à connaître :

- @ représente une adresse en mémoire ,
- #*a*h est équivalent à a_{16},
- #*b*d est équivalent à b_{10}.
- R(c) se note aussi à Rc ou R:c (R est un registre, c, un champ).

Quelques unités utilisées :

- 1 bit est un nombre binaire (0 ou 1),
- un quartet est un groupe de 4 bits,
- un octet est un groupe de 8 bits,
- un kilo octet rassemble 1024 octets.

En cas de problème, utilisez le service d'assistance technique (Voir *Assistance technique* en fin d'ouvrage).

1

Initiation

Le premier chapitre vous permet de découvrir l'architecture de la HP-48 et de programmer très rapidement en langage assembleur. Vous trouverez dans ce premier chapitre de nombreux exemples réutilisables dans vos propres créations.

Les chapitres 2 à 6 constituent un cours de numération hexadécimale et de logique. Le chapitre 5 est consacré au DCB, la maîtrise de celui-ci est indispensable si vous utilisez l'instruction `SETDEC` qui fait travailler le SATURN en DCB.

Les chapitres 8 à 11 décrivent le microprocesseur SATURN. Le chapitre 11 est dédié aux registres du SATURN.

Le chapitre 12 met en évidence le masquage des mémoires sur les HP-48 G et GX. L'accès à la mémoire masquée s'obtient en reconfigurant la mémoire (voir le chapitre 18).

Les chapitres 13 à 17 présentent le jeu d'instructions du SATURN. Le contenu du chapitre 13 doit absolument être connu avant d'aborder les chapitres 14 à 17.

Le chapitre 18 vous explique comment accéder à la mémoire masquée en reconfigurant les modules de mémoire. Il est nécessaire de maîtriser le contenu des chapitres 9, 10 et 12 avant d'aborder le chapitre 18.

1

Vite ! Mon premier programme !

Vous voulez de l'assembleur ? Vous allez en avoir ! Ce premier chapitre est là pour vous mettre le pied à l'étrier. Nous commencerons par exposer rapidement quelques notions essentielles. Ensemble, nous réaliserons ensuite plusieurs petits programmes ayant une fonction utile pour la conception de jeux. Cependant, il doit être parfaitement clair pour tous que la programmation en assembleur exige la maîtrise de notions très diverses et pas toujours simples. Celles-ci sont expliquées dans les autres chapitres de cette première partie.

Ce qu'il faut connaître pour programmer en assembleur sur HP-48 :

- la numération binaire et hexadécimale,
- les opérations logiques et les manipulations de bits,
- la structure du microprocesseur et les rôles de chacun de ses registres,
- les champs des registres,
- la structure de la mémoire, sa configuration, ses différents modules, ses plages d'adresses,
- les instructions que comprend le microprocesseur,
- et une foule d'autres choses qui ont su décourager plus d'un programmeur courageux !!!

Le glossaire en fin d'ouvrage vous aidera à comprendre le petit vocabulaire nécessaire. Une lecture raisonnable de ce livre consiste à lire une première fois ce premier chapitre d'initiation, étudiez ensuite avec attention tous les autres chapitres de cette première partie, car ce sont eux qui vous apporteront toutes les bases qu'exige l'assembleur, puis, revenez à ce chapitre qui vous semblera enfin limpide ! Vous pourrez alors étudier et exploiter les exemples de programmes ludiques de la troisième partie, la deuxième partie de ce livre étant consacrée aux indispensables références.

Attention ! En cas de maladresse, la programmation en assembleur peut provoquer la perte du contenu de la mémoire. En effet, toutes les protections qu'offrent l'utilisation normale de la machine sont ici inactives. Il est donc très vivement recommandé d'effectuer une sauvegarde de toutes vos données sur un micro-ordinateur, via la liaison série.

Vive les trombones ! Ayez toujours à portée de la main un trombone déplié, car, en cas de gros plantage, vous devrez probablement insérer celui-ci dans l'orifice prévu à cet effet sous le patin supérieur droit de votre HP-48 préférée...

1

Quelques notions indispensables

1.1

Programmation, quartet et hexadécimal

Toutes les notions que nous allons ici vous présenter très rapidement sont expliquées en détail dans les chapitres suivants de cette première partie. Nous nous contenterons donc, pour l'instant, d'un survol de quelques notions et définitions indispensables car nous présumons que vous êtes impatients de rédiger vos premières lignes d'assembleur !

■ Les trois niveaux de programmation de la HP-48

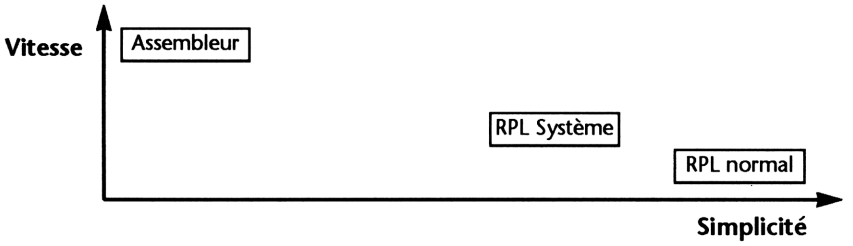
Dans le cadre de son utilisation normale, la HP-48 est prévue pour être programmée en RPL. Ce langage, apparenté au FORTH et au LISP, est facilement accessible à celui qui a quelques notions de BASIC ou de PASCAL et qui connaît la notation postfixe (ou RPN, notation polonaise inversée). Bref, le RPL est un langage accessible sans trop d'effort à l'utilisateur moyen, c'est d'ailleurs le langage le plus simple qui puisse être employé pour programmer la HP-48. Le RPL est un langage sûr qui effectue de nombreux tests pour garantir une exécution normale du programme. En particulier, les types et la présence des éventuels arguments sont contrô-

lés. Aucun risque de perte de donnée n'est donc à craindre en RPL. Cela dit, la sécurité que procurent les tests permanents effectués automatiquement en RPL se paye par une exécution très lente des programmes écrits dans ce langage.

Chaque instruction du langage RPL est composée d'une succession d'instructions élémentaires effectuant une tâche précise. Ces instructions élémentaires du langage RPL sont appelées, external, primitive codes ou syseval selon le cas. Il est donc possible de créer ses propres commandes RPL en combinant ces primitives codes. Le RPL système est né ! En RPL système, on fait appel à ces fractions d'instructions RPL pour construire un programme. On oublie volontairement toutes les fractions d'instructions chargées d'assurer la sécurité du programme (vérification de la présence et du type des arguments, etc.), on gagne ainsi beaucoup de temps.

Oublions le RPL et le RPL système car pour contrôler la machine au plus bas niveau, le seul langage valable est l'assembleur. En assembleur on commande directement le microprocesseur à l'aide d'instruction représentative de ce qu'il est capable de faire, autrement dit, très peu de choses ! En effet, le microprocesseur ne sait que lire dans la mémoire, y écrire et effectuer des opérations mathématiques et logiques très élémentaires. Le microprocesseur ne sait absolument pas calculer sin 45° et n'a pas la moindre idée de ce qu'est une pression sur une touche ou l'affichage d'un simple caractère sur l'écran. Il faudra donc lui expliquer tout cela avec une immense précision et une infinie patience ! Seules des opérations extrêmement simples sont possibles en assembleur, la réalisation du moindre calcul ou l'obtention d'un résultat compliquent terriblement le travail du programmeur qui doit se débrouiller avec un jeu d'instructions très limité. Heureusement, l'assembleur, dont la « simplicité » fait de lui le langage le plus compliqué, a pour lui un avantage qu'aucun autre langage ne peut remettre en cause : une rapidité exceptionnelle qui correspond très exactement aux capacités maximales de la machine.

En résumé, le RPL est le langage standard et officiel de la HP-48. Il est sûr, relativement facile à maîtriser mais lent et limité aux instructions prévues par HP. Le RPL système repose sur le « découpage » d'instructions RPL. On gagne en rapidité la sécurité perdue. Celui qui maîtrise le RPL saura programmer en RPL système à condition de savoir où se trouvent en ROM les fractions d'instructions dont il a besoin. La programmation en assembleur est sans lien avec la programmation en RPL ou en RPL système. Toutes les instructions usuelles devront donc être réinventées. L'assembleur est un langage très complexe car il ne permet que des actions élémentaires mais il offre, en échange de quelques nuits blanches, les programmes les plus rapides qui soient et l'accès à toutes les ressources de la machine.



Ce schéma résume ce qui vient d'être dit à propos des trois niveaux de programmation.

■ Les trois niveaux de la programmation en assembleur

Trois niveaux de programmation peuvent être distingués en assembleur :

- le langage machine,
- les codes hexadécimaux,
- les mnémoniques de l'assembleur.

Le langage machine est le seul que comprenne le microprocesseur. Malheureusement, il n'est constitué que d'une succession de 0 et de 1, il est donc absolument incompréhensible. On ne manipule jamais le langage machine sous sa forme pure.

Comme nous allons le voir, ces 0 et ces 1 peuvent être groupés par paquets de quatre chiffres pour former un quartet. Chaque quartet peut être représenté par un chiffre hexadécimal (c'est-à-dire un chiffre en base 16). On groupe donc par paquets de quatre les 0 et les 1 du langage machine. On obtient ainsi une suite de quartets faciles à représenter à l'aide de chiffres hexadécimaux. Cette chaîne de codes hexadécimaux n'est donc qu'une forme facile d'emploi du langage machine.

Cela dit, le microprocesseur ne sait faire qu'un petit nombre de chose. On gagne donc à associer une représentation lisible et mnémotechnique à chacune de ses activités élémentaires. A chaque instruction compréhensible par le microprocesseur correspond donc un mnémonique. Le programmeur utilise ces mnémoniques pour la création de son programme. Des tables de correspondances permettent de transformer ces mnémoniques en codes hexadécimaux. Un petit programme très simple permet ensuite d'obtenir le programme exécutable... en langage machine !

Le programmeur développe donc un code source à l'aide de mnémoniques. Ce code source est la base de travail du programmeur en langage assembleur. Il suffit ensuite d'une simple traduction de ce code source

en code hexadécimal puis en langage machine pour obtenir le programme qui sera effectivement exécuté par la machine. Le programme effectivement exécuté (toujours en langage machine !) est appelé code objet ou code exécutable.

En résumé :

- le langage machine est le seul que comprenne le microprocesseur. Constitué uniquement de 0 et de 1, il est illisible et n'est pas manipulé directement,
- le code hexadécimal est une traduction lisible du langage machine dont les 0 et les 1 sont groupés par paquets de quatre,
- les mnémoniques sont des noms associés à chaque instruction du langage machine. C'est à partir de mnémoniques que l'on écrit le code source du programme. Il suffit de traduire ce code source en codes hexadécimaux puis en langage machine pour obtenir le code exécutable (ou code objet) qui sera effectivement exécuté par la machine.

On confond souvent le langage machine et l'assembleur. Il est vrai que l'un n'est que la traduction directe de l'autre. Cependant, et puisqu'il convient d'être précis, retenez que le langage machine correspond au code exécutable uniquement et n'est constitué que de 0 et de 1. Le langage assembleur fait appel quant à lui aux mnémoniques. Les codes hexadécimaux se situent à mi-chemin entre le langage machine et les mnémoniques du langage assembleur, néanmoins, nous venons de dire que le langage machine se limite au code exécutable, nous assimilerons donc les codes hexadécimaux au langage assembleur. Le langage assembleur peut donc prendre deux formes que sont les mnémoniques d'une part et les codes hexadécimaux d'autre part. Par ailleurs, notez que le langage assembleur est aussi appelé assembleur et langage d'assemblage.

Pour vous convaincre de la nécessité des mnémoniques, considérez une instruction courante comme `LAHEX` (tel est son mnémonique) qui peut, selon son paramètre, être représentée par 2^{64} codes hexadécimaux différents, c'est-à-dire environ 2.10^{18} !

Définition du langage assembleur donnée par *Le Dictionnaire de l'informatique* (J. Woodcock, DUNOD, 1992) : *Type de langage de programmation de très bas niveau dans lequel chaque instruction correspond à une instruction machine unique. Un langage d'assemblage est donc spécifique d'un certain processeur. Après écriture d'un programme en langage d'assemblage, le programmeur doit utiliser l'assembleur spécifique du processeur pour traduire son programme en instruction machine. Le langage d'assemblage donne une maîtrise totale de l'ordinateur mais les programmes écrits pour un type d'or-*

dinateur doivent être réécrits pour tourner sur un autre type. Le langage d'assemblage peut être préféré à un langage de haut niveau dans les cas suivants : vitesse, contrôle ou préférence personnelle. Les programmes écrits en langage d'assemblage tournent plus vite et occupent moins de place que ceux produits par un compilateur. Ils donnent au programmeur la possibilité d'agir directement sur le matériel (processeur, mémoire, affichage et ports d'entrées/sorties).

On appelle langage de haut niveau un langage évolué et facilement compréhensible par l'homme : le RPL en est un. Au contraire, un langage de bas niveau sera proche de ce que sait effectivement faire la machine. Le langage machine est un langage de bas niveau. L'assembleur l'est aussi puisqu'il n'est qu'une traduction du langage machine.

■ Programme assembleur et langage assembleur

On appelle programme assembleur un programme capable de transformer un code source écrit en langage assembleur en un code exécutable en langage machine. Le programme assembleur constitue donc un outil de programmation et peut lui-même avoir été écrit à l'aide de n'importe quel langage de programmation.

Ne confondez pas programme assembleur et programme écrit en langage assembleur !

■ Le processus de développement d'un programme en assembleur

Un programme en langage d'assemblage ne se crée pas sur la machine mais sur une feuille de papier (avec une bonne gomme !). Les différentes étapes de développement du programme consistent à :

- analyser ce que doit faire le programme avec une précision maximale,
- décomposer le programme en sous-ensembles (par exemple, l'un pour le calcul de telle fonction, l'autre pour l'affichage, etc.),
- décomposer chaque tâche en petites tâches élémentaires jusqu'à ce que chaque tâche élémentaire corresponde à une instruction du microprocesseur,
- traduire en mnémoniques,
- traduire en codes hexadécimaux,
- traduire en langage machine.

Retenez qu'un programme en assembleur naît sur le papier et y retourne généralement régulièrement pour y être corrigé !

■ Les bits, les quartets et l'hexadécimal

Nous avons dit que le langage machine se compose d'une suite de 0 et de 1. Cela est lié au fait qu'un microprocesseur n'est capable de travailler qu'avec ce type d'informations. Ces 0 et ces 1 sont appelés bits. Un groupe de quatre bits est appelé quartet.

L'hexadécimal est le système de numération de base 16. Notre système de numération habituel est décimal, il emploie la base 10 et les dix chiffres 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9. « 10 » représente la base quel que soit le système de numération. « 10 » représente donc 10 en base 10 alors qu'il représente 16 en base 16... Comme vous venez de le comprendre il est indispensable de préciser le système de numération. On ajoute donc un indice à la suite du nombre, par exemple « $_{10}$ » et « $_{16}$ » pour les bases 10 et 16. On peut aussi encadrer le nombre avec « # » et une lettre minuscule représentant la base. 2 en base 10 pourra donc être noté 2_{10} ou #2d (d pour décimal, c'est-à-dire base 10). De même 16 en base 10 est noté 16_{10} ou #16d. Converti en hexadécimal, ce nombre se note 10_{16} ou #10h (h pour hexadécimal, c'est-à-dire base 16).

La base 10 emploie dix chiffres, de même, la base 16 emploie seize chiffres qui sont : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F. Le tableau ci-dessous présente quelques équivalences entre la base 10 et la base 16.

Base 10	Base 16	Base 10	Base 16	Base 10	Base 16
0_{10}	#0h	11_{10}	#Bh	26_{10}	#1Ah
1_{10}	#1h	12_{10}	#Ch	30_{10}	#1Dh
2_{10}	#2h	13_{10}	#Dh	31_{10}	#1Eh
3_{10}	#3h	14_{10}	#Eh	32_{10}	#1Fh
4_{10}	#4h	15_{10}	#Fh	255_{10}	#FFh
5_{10}	#5h	16_{10}	#10h	256_{10}	#100h
6_{10}	#6h	17_{10}	#11h	257_{10}	#101h
7_{10}	#7h	18_{10}	#12h	4095_{10}	#FFFh
8_{10}	#8h	19_{10}	#13h	4096_{10}	#1000h
9_{10}	#9h	20_{10}	#14h	65535_{10}	#FFFFh
10_{10}	#Ah	25_{10}	#19h	65536_{10}	#10000h

Voyons maintenant comment quatre bits, un quartet donc, sont équivalents à un chiffre hexadécimal. Considérons quatre bits quelconques, par exemple 1110. Plaçons-les dans un tableau et numérotons ses colonnes.

Numéros des bits	3	2	1	0
Bits	1	1	1	0

Remarquez que l'on numérote toujours à partir de zéro. Lorsque 16 valeurs doivent être numérotées, elles le sont donc de 0 à 15, c'est-à-dire, de #0h à #Fh !

Considérons maintenant que les numéros des bits sont les exposants du nombre 2_{10}.

Poids des bits	2^3	2^2	2^1	2^0
Numéros des bits	**3**	**2**	**1**	**0**
Bits	1	1	1	0

On peut bien sûr calculer les puissances de 2...

Poids des bits	$8=2^3$	$4=2^2$	$2=2^1$	$1=2^0$
Numéros des bits	**3**	**2**	**1**	**0**
Bits	1	1	1	0

Nous venons d'associer une valeur à chaque bit du quartet 1110. Ce quartet n'est autre qu'un nombre binaire (c'est-à-dire en base 2) formé de quatre chiffres, binaires eux aussi. Les nombres binaires sont suivis de l'indice « $_2$ ». Il est maintenant facile d'obtenir la valeur dénotée par ce quartet en multipliant chacun des bits par leur poids. On pose :

$$1110_2 = 1\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0$$

D'où :

$$1101_2 = 1\times8 + 1\times4 + 1\times2 + 0\times1$$
$$1101_2 = 8 + 4 + 2 + 0$$
$$1101_2 = 14_{10}$$
$$1101_2 = \#Dh$$

Il est donc possible de convertir un quartet en un chiffre hexadécimal. Un bit peut prendre deux formes, 0 et 1. Un quartet se compose de quatre bits. Un quartet peut donc prendre 2^4=16 formes différentes. A chaque quartet correspond donc un chiffre hexadécimal, et inversement. Le tableau ci-dessous présente les correspondances.

Quartet sous forme de groupe de 4 bits (exprimé en base 2)	**Quartet sous forme de chiffre hexadécimal (exprimé en base 16)**	**Quartet sous forme de groupe de 4 bits (exprimé en base 2)**	**Quartet sous forme de chiffre hexadécimal (exprimé en base 16)**
0000 (0000_2)	0 (#0h)	1000 (1000_2)	8 (#8h)
0001 (0001_2)	1 (#1h)	1001 (1001_2)	9 (#9h)
0010 (0010_2)	2 (#2h)	1010 (1010_2)	A (#Ah)
0011 (0011_2)	3 (#3h)	1011 (1011_2)	B (#Bh)
0100 (0100_2)	4 (#4h)	1100 (1100_2)	C (#Ch)
0101 (0101_2)	5 (#5h)	1101 (1101_2)	D (#Dh)
0110 (0110_2)	6 (#6h)	1110 (1110_2)	E (#Eh)
0111 (0111_2)	7 (#7h)	1111 (1111_2)	F (#Fh)

On symbolisera désormais un quartet par le chiffre hexadécimal qu'il représente. Par extension, on peut considérer qu'un chiffre hexadécimal est un quartet. Ainsi, #CCD20h est un nombre hexadécimal à cinq chiffres, il s'agit donc de cinq quartets !

A retenir :

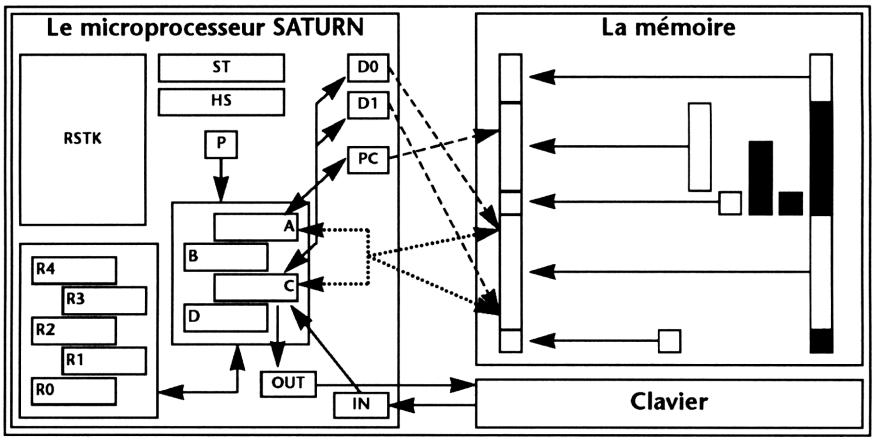
- la base 16, ou système de numération hexadécimal, emploie seize chiffres de 0 à 9 et de A à F. En base 16, « 10 » représente 16_{10}, autrement dit, « 10 » représente la base,
- un microprocesseur, comme tout autre circuit d'électronique numérique, ne manipule que des 0 et des 1,
- ces 0 et ces 1 sont des bits (contraction de binary digits), autrement dit, des nombres binaires, ou encore, des nombres en base 2,
- un quartet est un groupe de quatre bits. Chaque quartet est représenté par un chiffre hexadécimal. On peut donc considérer qu'un chiffre hexadécimal est le quartet qu'il représente.

Tout ceci est expliqué et approfondi dans les autres chapitres de cette première partie. Néanmoins, il nous a semblé indispensable d'aborder une première fois ces notions avant d'entamer la réalisation de votre premier programme en langage d'assemblage.

1.2

HP-48 à cœur ouvert !

Ci-dessous, le schéma de principe du fonctionnement de la HP-48.



Ce schéma met en évidence deux grands ensembles :

- la mémoire,
- le microprocesseur.

La programmation en assembleur repose sur des échanges de données entre la mémoire et le microprocesseur où ces mêmes données font l'objet de traitements élémentaires.

■ La mémoire

La mémoire est chargée de conserver les données. Celles-ci peuvent être de deux types :

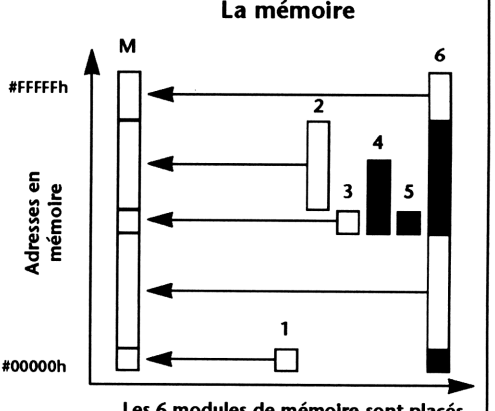
- les données traitantes, par exemple, les programmes,
- les données traitées qui sont les arguments et les résultats des programmes.

La mémoire de la HP-48 est divisée en 6 modules de mémoire. La mémoire contient des quartets, chacun de ces quartets se trouve à une adresse précise. A chaque quartet correspond donc une adresse. Le microprocesseur SATURN de la HP-48 ne peut distinguer que $2^{20} \approx 1000000$ d'adresses numérotées de #00000h à #FFFFFh en hexadécimal et sur 5 quartets. Malheureusement, il y a plus de quartets en mémoire que d'adresses disponibles. Il est donc nécessaire d'employer une même adresse pour deux quartets distincts. Etant donné que la mémoire est divisée en 6 modules, l'astuce consiste à créer une hiérarchie entre ces modules. Ainsi, lorsque l'on utilise une adresse appartenant à deux quartets différents, l'adresse désignera le quartet appartenant au module le plus prioritaire parmi ceux où existe l'adresse.

Les 6 modules de mémoire sont représentés par des rectangles. Les plages d'adresses directement accessibles correspondent à des rectangles blancs alors que les zones masquées correspondent à des rectangles noirs.

On remarquera que certains modules prioritaires (1, 2, 3) sont visibles dans leur intégralité, d'autres, moins prioritaires, sont complètement masqués (4, 5). Le module 6 (qui correspond à la mémoire morte des HP-48 G/GX) est partiellement masqué.

Quoi qu'il en soit, la mémoire (M) semble toujours linéaire de #00000h à #FFFFFh. Spécifier une adresse revient à spécifier indirectement un module de mémoire.



Il va de soi qu'une même adresse ne peut pas être employée à deux reprises dans un même module car il n'est alors plus possible de s'appuyer sur la hiérarchie entre les modules pour retrouver un quartet.

Certaines plages d'adresses de certains modules, peu prioritaires, sont donc masquées par les mêmes plages d'adresses utilisées par des modules plus prioritaires. La mémoire apparaît comme continue de #00000h à #FFFFFh, la plage d'adresses utilisées détermine le module utilisé.

Bien entendu, il est possible d'accéder à la mémoire masquée. Il suffit pour cela de reconfigurer la mémoire en affectant aux modules prioritaires une adresse de départ et une taille telles qu'ils ne masquent plus le module moins prioritaire auquel on souhaite accéder.

A retenir :

- la mémoire est chargée de conserver les données,
- la mémoire est le terme employé pour désigner un ensemble de 6 modules de mémoire ayant chacun leurs rôles et spécificités,
- la mémoire peut être vue comme une suite linéaire d'adresses allant de #00000h à #FFFFFh,
- spécifier une adresse revient à accéder au quartet se trouvant à ladite adresse dans le module le plus prioritaire parmi les modules où existe cette adresse,
- il n'existe pas suffisamment d'adresses pour tous les quartets en mémoire. Il est donc nécessaire d'employer une même adresse pour plusieurs quartets appartenant à différents modules. La hiérarchie des modules permet de rendre visible ou de masquer les plages d'adresses où des adresses identiques sont employées. Avec ce principe et avec une configuration donnée de la mémoire, un unique quartet correspond à chaque adresse en mémoire,
- les modules de mémoire masquée sont accessibles en modifiant la configuration de la mémoire.

■ Le microprocesseur

Le microprocesseur est chargé de traiter les données. Il exécute les ordres exprimés à l'aide des données traitantes (programme) pour modifier les données traitées.

On trouve, au sein du microprocesseur, de minuscules mémoires appelées registres. Dans un premier temps, et de façon très approximative, on peut

séparer les registres en deux catégories en fonction de la nature de leur contenu :

- certains registres sont chargés de désigner une adresse en mémoire. Ces registre sont appelés pointeurs,
- d'autres registres sont chargés d'accueillir les données traitées.

Les registres D0, D1 et PC sont des registres pointeurs. Ils contiennent une adresse en mémoire. Nous avons déjà dit que les adresses sont codées sur 5 quartets, ces registres pointeurs ont donc une capacité de 5 quartets. L'un d'eux peut donc contenir une donnée comme #00100h ou #CFFFFh.

Le registre pointeur PC a un rôle capital, c'est lui qui contient l'adresse en mémoire de l'instruction à exécuter (celle-ci est une donnée traitante).

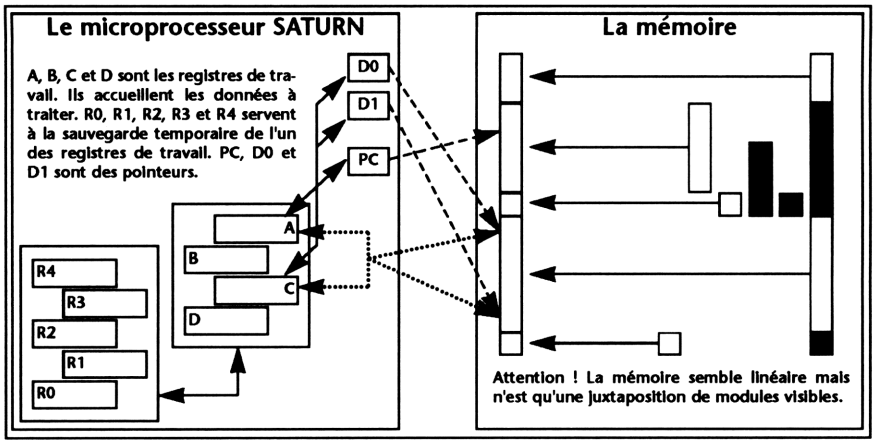
Parmi les registres chargés d'accueillir les données traitées, on peut distinguer 2 types de registres :

- les registres de travail, aussi appelés registres de calcul. Toute donnée que doit modifier le microprocesseur passe par un registre de travail car c'est le contenu d'un registre de travail qui fait l'objet des traitements (opération logique, décalage de bits, etc.). Les 4 registres de travail sont nommés A, B, C et D,
- les registres de sauvegarde sont chargés de la sauvegarde temporaire du contenu de l'un des registres de travail lorsque celui-ci doit être libéré pour accueillir une nouvelle donnée. Pour remplir ce rôle, les registres de sauvegarde ont bien entendu la même capacité que les registres de travail, c'est-à-dire 16 quartets (64 bits). Les registres de sauvegarde s'appellent R0, R1, R2, R3 et R4. Ils ne sont pas associés à un registre de travail particulier et peuvent donc sauvegarder le contenu de n'importe quel registre de travail.

Le rôle des pointeurs D0 et D1 est lié au contenu des registres de travail. En effet, D0 et D1 contiennent des adresses qui sont celles où seront lus ou écrits les quartets des registres de travail.

A retenir :

- c'est au sein des registres A, B, C et D que les données sont traitées,
- si une sauvegarde temporaire s'avère nécessaire, l'un des registres de sauvegarde R0, R1, R2, R3 ou R4 peut accueillir la donnée à conserver temporairement,
- le registre pointeur PC contient l'adresse en mémoire de l'instruction à exécuter,
- D0 et D1 désignent les adresses en mémoire utilisées pour les échanges de données entre un registre de travail et la mémoire.



A condition d'avoir conscience que la mémoire n'est qu'une juxtaposition de portions visibles de modules de mémoire prioritaires, on peut la considérer comme linéaire. Elle apparaît donc comme une suite de #100000h quartets numérotés de #00000h à #FFFFFh.

Les flèches du schéma imposent quelques commentaires :

- PC, D0 et D1 contiennent une adresse en mémoire. Pour attribuer une adresse à l'un de ces pointeurs, il faut transférer des quartets depuis A ou C,
- les quartets se trouvant en mémoire à partir de l'adresse désignée par D0 ou D1 peuvent être copiés dans A ou C. Inversement, le contenu de A ou de C peut être copié en mémoire à partir de l'adresse indiquée par D0 ou D1,
- PC contient l'adresse en mémoire de l'instruction à exécuter, il désigne donc toujours une donnée traitante en mémoire,
- les registres de sauvegardes R0, R1, R2, R3 et R4 se contentent de conserver temporairement le contenu des registres de travail A, B, C et D. On peut donc copier des quartets depuis un registre de travail vers un registre de sauvegarde puis effectuer une copie en sens inverse lorsque cela devient nécessaire.

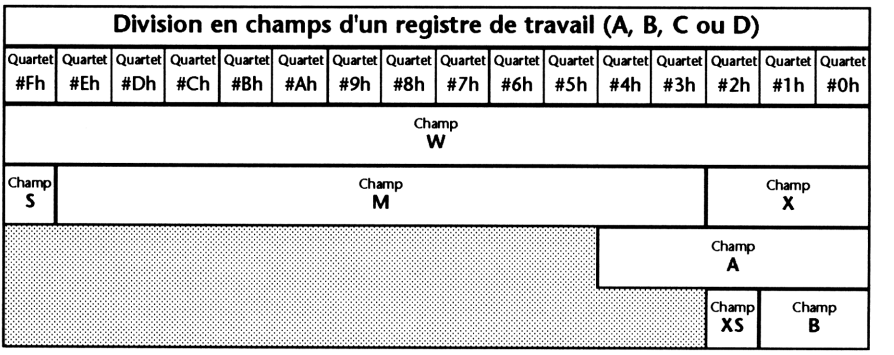
■ Le découpage en champs des registres de travail

Les quartets que contient un registre de travail sont traités par le microprocesseur. Les registres de travail ont une capacité de 16 quartets (64 bits). Il est très souvent indispensable de réduire l'application d'un traitement à une partie seulement d'un registre de travail. Un champ est une partie d'un registre de travail. Il existe des champs prédéfinis mais il est possible de définir ses propres champs.

Commençons par numéroter de #0h à #Fh les 16 quartets d'un registre de travail.

Numérotation des 16 quartets d'un registre de travail (A, B, C ou D)															
Quartet #Fh	Quartet #Eh	Quartet #Dh	Quartet #Ch	Quartet #Bh	Quartet #Ah	Quartet #9h	Quartet #8h	Quartet #7h	Quartet #6h	Quartet #5h	Quartet #4h	Quartet #3h	Quartet #2h	Quartet #1h	Quartet #0h

Un champ est une portion de registre constituée de quartets contigus. Le tableau ci-dessous présente les champs prédéfinis.



Quelques champs fréquemment utilisés :

- le champ W (W comme Word, mot binaire) correspond à l'intégralité du registre. Dire qu'une instruction s'applique à l'intégralité d'un registre revient donc à dire qu'elle s'applique à son champ W,
- le champ A (A comme Address, adresse en mémoire) a une capacité de 5 quartets (quartets n°#0h à #4h). Il est très utile pour l'affectation d'une adresse à un registre pointeur ou pour le travail à partir d'une adresse. Rappelons que celles-ci sont codées sur 5 quartets,
- le champ B (B comme Byte, octet) a une capacité de 2 quartets (quartets n°#0h et n°#1h),
- le champ X a une capacité de 3 quartets (quartets n°#0h à n°#2h).

A retenir :

- les registres de travail (et seulement eux) peuvent être découpés en champs,
- il est souvent commode ou nécessaire de limiter un traitement à une partie seulement d'un registre de travail. Le découpage en champs est donc indispensable,
- les champs W, A, B, XS, X, M, S sont des champs prédéfinis et fixes.

■ Le registre P détermine les champs P et WP

Outre les sept champs prédéfinis, le programmeur peut créer des champs personnalisés pour répondre à ses besoins. Il utilise pour cela le registre P.

Le registre P est un petit registre très particulier. Il ne peut contenir qu'un quartet. La valeur qu'il contient est donc comprise entre #0h et #Fh. Cette valeur permet de définir deux champs :

- le champ P d'un registre de travail se résume à un unique quartet et correspond au quartet n°#Ph de ce registre de travail. Autrement dit, le quartet d'un registre de travail dont le numéro se trouve dans le registre P constitue le champ P du registre de travail,
- le champ WP d'un registre de travail correspond à tous les quartets de ce registre depuis le quartet #0h jusqu'au quartet dont le numéro est contenu dans le registre P.

Ne confondez surtout pas le champ P d'un registre de travail avec le registre P !

Par exemple, l'unique quartet du registre P vaut #3h. Le champ P d'un quelconque registre de travail correspond alors à son quartet n°#3h. Quant à lui, le champ WP correspond aux quartets n°#0h à #3h de ce même registre de travail.

Le registre P, lorsqu'il n'est pas utilisé pour définir un champ particulier, peut trouver une foule d'autres emplois fort commodes. Il constitue en outre un excellent petit compteur.

■ Vers une vision cohérente du microprocesseur SATURN

Nous connaissons déjà :

- les registres de travail A, B, C et D,
- les registres de sauvegarde R0, R1, R2, R3 et R4,
- le registre P qui permet de définir les champs personnalisés P et WP au sein des registres de travail,
- les registres pointeurs D0 et D1 qui déterminent l'adresse des quartets en mémoire avec lesquels A et C peuvent échanger des données,
- le registre pointeur PC, appelé compteur ordinal et chargé d'indiquer où se trouve l'instruction à exécuter.

La connaissance de ces registres peut suffire pour une première approche mais sachez qu'il existe d'autres registres au sein du SATURN.

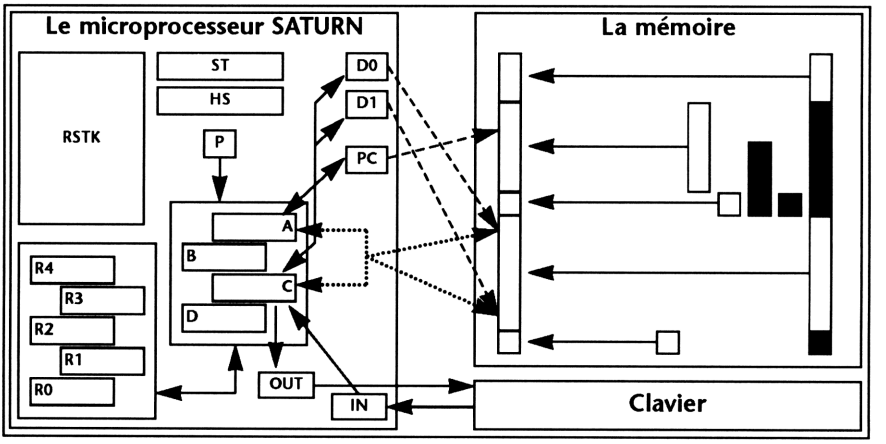
Quelques registres peu utilisés par le débutant :

- RSTK contient 8 adresses qui sont les adresses de retour de sous-programme (nous verrons cela plus tard !),
- ST contient 4 quartets dont chaque bit a un rôle de drapeau,
- les 4 bits de HS ont un état représentatif de l'activité du microprocesseur.

Les registres IN et OUT servent à la détection d'une pression sur le clavier.

■ Les échanges entre la mémoire, le clavier et le microprocesseur

Nous avons décrit successivement tous les éléments de ce schéma. Vous devriez maintenant mieux comprendre les flux de données entre les registres du microprocesseur d'une part, entre le microprocesseur, la mémoire et le clavier, d'autre part.



Le registre pointeur PC désigne l'adresse de l'instruction à exécuter. PC indique donc au microprocesseur où se trouve le programme qu'il doit exécuter.

D0 et D1 désignent les adresses où A et C pourront lire ou écrire leur contenu. Deux définitions s'imposent :

- une lecture en mémoire est un transfert de quartets depuis la mémoire vers un registre du microprocesseur. Le contenu de la mémoire reste inchangé mais le contenu des registres est « écrasé » par les quartets en provenance de la mémoire,
- une écriture est un transfert de quartets depuis l'un des registres du microprocesseur vers la mémoire. Le contenu du registre concerné reste inchangé mais le contenu de la mémoire est remplacé par les quartets en provenance du registre.

La détection d'une pression sur une touche du clavier est obtenue en plaçant dans le registre OUT (dont le contenu est copié depuis le registre C) quelques quartets représentatifs de la touche que l'on veut tester. On recueille ensuite dans le registre IN des signaux émis par le clavier. Le contenu de IN est renvoyé dans le registre de travail C. Une opération logique (une comparaison de bits dans le cas présent) permet de savoir si la touche testée a effectivement été pressée.

■ **Qu'est-ce que la programmation en assembleur ?**

En assembleur, on se contente de rapatrier vers les registres de travail des données en provenance de la mémoire ou du clavier. On traite les données se trouvant dans les registres de travail puis on les replace dans la mémoire... C'est tout !

2

Mon premier programme... Vite !

Que ceux qui n'ont toujours pas sauvegardé l'intégralité de la mémoire de leur HP-48 sur le disque dur de leur ordinateur ou sur une carte RAM le fassent tout de suite ! Les autres peuvent aller prendre un petit café car les choses sérieuses vont commencer...

2.1

Contentons-nous de ne rien faire !

■ **Le programme ASS**

Le programme ASS, déjà utilisé dans le livre *HP-48 : Mathématiques en prépa* (DUNOD), sert à la conversion d'une chaîne de codes hexadécimaux en code exécutable (en langage machine). ASS est un petit programme écrit en RPL qui se contente d'appeler deux adresses syseval en mémoire morte.

Nous n'allons pas expliquer avec beaucoup de précision la manipulation de la pile car l'utilisation de base de la HP-48 est considérée comme acquise. Si vous ne vous sentez pas sûr de vous, nous vous conseillons la lecture préalable de *HP-48 : permis de conduire* (DUNOD) qui vous expliquera, simplement et précisément, comment maîtriser votre HP-48 G/GX.

Il est vivement conseillé de créer un répertoire de travail dans lequel nous rangerons le programme ASS ainsi que tous les exemples de programmes que nous utiliserons. Ce répertoire sera directement accessible depuis HOME, nous le nommerons 'EXEMPLES'. Pour le créer, saisir :

- [↱] ['] [VAR] pour se placer dans le répertoire HOME,
- ['] [α] [α] puis le nom EXEMPLES suivi de [ENTER]
- [α] [α] CRDIR [ENTER] pour créer le répertoire,
- [A] (première touche en haut à gauche) pour se placer dans notre nouveau répertoire.

Avant de saisir ASS, placez-vous en mode hexadécimal en saisissant [MTH] [F] [A]. Commencez ensuite la saisie de ASS dont le listing figure ci-dessous. N'oubliez pas d'utiliser les délimiteurs de programme RPL accessibles par [↰] [-]. Rappelons que les instructions d'un programme RPL peuvent être saisies caractère par caractère ou obtenues en sélectionnant un article de l'un des sous-menus accessibles par la touche [PRG]. Par ailleurs, il n'est absolument pas nécessaire de revenir à la ligne pendant la saisie, contentez-vous de placer un espace entre les instructions et les paramètres.

Le listing de ASS :

CCD 20

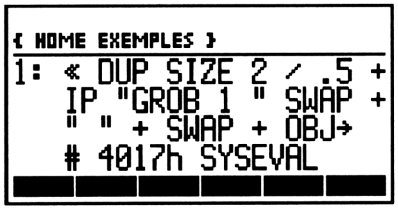
```
« DUP SIZE 2 / .5
+ IP "GROB 1 "
SWAP + " "
+ SWAP + OBJ→
#4017h SYSEVAL
#56B6h SYSEVAL
DROP NEWOB »
```

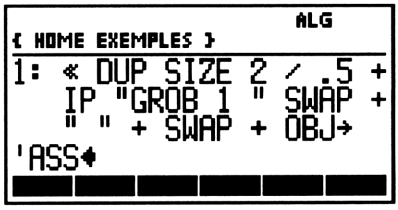
Une fois le programme introduit, appuyez sur [ENTER] puis saisissez ['] [α] [α] ASS [α] [STO]. Le programme RPL saisi se trouve maintenant sauvegardé dans la variable ASS du répertoire EXEMPLES.

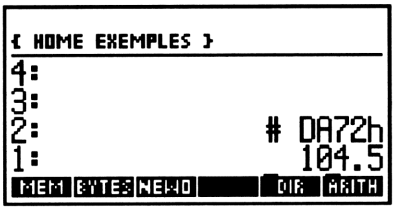
Il nous faut maintenant vérifier que le programme a été saisi correctement. Pour cela, on le renvoie dans la pile sans l'évaluer en saisissant [↱] [A] (car [A] est la touche qui permet ici de sélectionner l'article ASS du menu), puis on saisit [↰] [VAR] [B] pour activer BYTES chargée de renvoyer la longueur de l'objet et son checksum.

Vous devez normalement obtenir #DA72h au niveau 2 de la pile et 104.5 au niveau 1. #DA72h est le checksum (« somme de contrôle » ou CRC, code de redondance cyclique) alors que 104.5 est l'espace occupé en mémoire par l'objet, cet espace est exprimé en octets. ASS occupe donc ici 209 quartets en mémoire (209 = 2×104,5).

Si vous n'obtenez pas les valeurs #DA72h et 104.5, cela signifie que vous avez commis une erreur de saisie. Corrigez alors le programme en saisissant [VAR] [↱] [A] [↓]. Une fois les corrections effectuées, n'oubliez pas de sauvegarder à nouveau le programme sous le nom ASS qui, sinon, restera associé à la version défectueuse du programme.







■ Le checksum (ou CRC)

A l'aide de la commande BYTES accessible par [↰] [VAR] [B], nous venons de calculer le checksum (ou CRC) du programme ASS. Pour calculer le CRC d'un objet, il suffit de l'envoyer dans la pile sans l'évaluer puis d'appeler la commande BYTES avec [↰] [VAR] [B]. Rappelons que pour placer dans la pile un objet sans l'évaluer, il suffit d'appuyer sur [↱] avant d'appuyer sur la touche de sélection associée à l'article de menu représentant la variable qui contient l'objet. Le checksum (ou CRC, ou somme de contrôle) est un moyen simple et fiable de vérifier qu'un objet a été correctement saisi. Nous y aurons donc recours systématiquement.

■ Saisir et contrôler une chaîne de codes hexadécimaux

Notre premier programme n'est pas une farce. Il existe mais ne fait rien. Il se contente de se déclarer puis de rendre la main au RPL.

Ce que notre programme, qui ne fait rien, va quand même faire :

- se déclarer en tant que code exécutable (1) en annonçant le prologue 02DCC (les objets et leurs prologues sont présentés au chapitre 22),
- déclarer sa longueur (2), toujours sur 5 quartets,
- rendre la main au RPL en chargeant le prologue de l'objet qui le suit en mémoire dans le registre A (3),
- faire pointer D0 sur les premiers quartets suivant le prologue de cet objet (4),
- évaluer cet objet (5).

Nous venons d'énoncer les spécifications de notre programme. En programmation, les spécifications définissent l'environnement opérationnel et les caractéristiques offertes par le nouveau programme.

Nous présenterons les programmes dans des tableaux tels que celui-ci :

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1) Signale un code exécutable
#00h	XXXXX	*Longueur hp*	(2) Longueur hors prologue
#05h	142	A=DAT0 A	(3)
#08h	164	D0=D0+ 5	(4)
#0Bh	808C	PC=(A)	(5) On quitte le programme

Nous remplirons les tableaux de la façon suivante :

- on commence par traduire les spécifications en mnémoniques, la colonne des mnémoniques est donc la première remplie,
- puis, on traduit ces mnémoniques en codes hexadécimaux en se référant aux chapitres 13 à 17 (vous utiliserez le chapitre 19 quand vous serez expérimenté !),
- il n'est malheureusement pas possible d'effectuer cette traduction en parcourant les mnémoniques du premier au dernier. En effet, certains codes comme celui de la longueur hors prologue ou ceux représentant les décalages d'adresses pour les sauts (nous verrons cela bientôt) ont des valeurs inconnues avant la traduction de tous les autres mnémoniques. Bien que ces quartets soient inconnus, on connaît leur nombre avec précision. Par exemple, on sait que la longueur est codée sur 5 quartets, on remplace donc temporairement ces quartets par autant de « X »,
- on numérote les codes hexadécimaux en affectant le numéro #0h (ou #00h, ou #000h... selon votre humeur) au premier quartet codant la longueur hors prologue. Les mnémoniques étant représentés par quelques quartets, l'adresse signalée en

face de ce groupe de quartets est celle du premier d'entre eux c'est-à-dire celui qui se trouve le plus à gauche,
• on peut ensuite remplacer les quartets inconnus représentés par des « X » par leurs valeurs,
• l'étape suivante consistera à juxtaposer tous les codes hexadécimaux dans une chaîne de caractères.

Dans le cadre de notre exemple, le seul champ inconnu au premier abord est la longueur hors prologue du code hexadécimal. Nous savons juste que cette longueur est toujours codée sur 5 quartets. Nous l'avons donc remplacée par XXXXX. Une fois tous les mnémoniques traduits en codes hexadécimaux ou en « X », il nous est possible d'obtenir la longueur totale hors prologue. Il suffit pour cela de compter tous les quartets (dont les éventuels quartets « X » encore inconnus) à partir du premier quartet codant la longueur totale hors prologue jusqu'au dernier quartet du code. Insistons sur le fait qu'il s'agit d'une longueur hors prologue, les 5 quartets codant le prologue sont donc ignorés. Dans le cadre de notre exemple, la longueur hors prologue est #Fh. Notons qu'elle ne correspond pas à l'adresse du dernier code hexadécimal.

Nous avons parlé de champ inconnu au sein du code hexadécimal. Il ne s'agit en aucun cas d'un champ de registre ! Un champ est ici une portion de code ayant un rôle déterminé. Les 5 quartets codant le prologue sont un champ, de même les 5 quartets codant la longueur totale hors prologue constituent aussi un champ, etc.

Notez qu'il nous faut inverser les champs. Ainsi, le prologue 02DCC sera utilisé sous la forme CCD20. De même, la longueur totale hors prologue de notre premier exemple est #Fh, ce qui correspond sur 5 quartets à #0000Fh. Une fois ce champ inversé, on obtient F0000. Nous pouvons donc achever le codage de notre exemple :

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1) Signal
#00h	F0000	*Longueur hp*	(2) Longueur hors prologue
#05h	142	A=DAT0 A	(3)
#08h	164	D0=D0+ 5	(4)
#0Bh	808C	PC=(A)	(5) On quitte le programme

A retenir :
• un programme en assembleur se déclare en tant que code exécutable à l'aide du prologue 02DCC,
• lors de la traduction des mnémoniques en codes hexadécimaux, on remplace les quartets inconnus par autant de « X »,

- tous les champs de données sont inversés,
- la longueur hors prologue correspond au nombre de quartets du code moins les 5 quartets du prologue (02DCC qui devient `CCD20` dans le code),
- un programme en assembleur commence toujours par le prologue 02DCC suivi de la longueur du code hors prologue toujours codée sur 5 quartets,
- « rendre la main au RPL » revient à quitter l'exécution du programme. Pour cela, on évalue l'objet suivant. Un programme écrit en assembleur se finit donc le plus souvent par `A=DAT0 A` suivi de `D0=D0+ 5` et `PC=(A)`. Les chapitres 13 à 17 vous renseigneront sur le rôle précis de ces instructions.

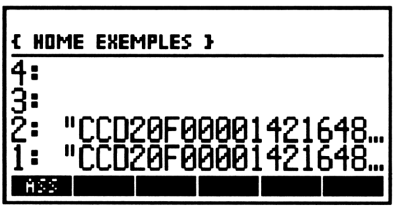
Nous allons maintenant saisir notre programme. Si nous juxtaposons les codes hexadécimaux, nous obtenons : **CCD20F0000142164808C**. Pour pouvoir être introduite et traitée, cette chaîne doit être placée entre les délimiteurs d'une chaîne de caractères que sont " et ".

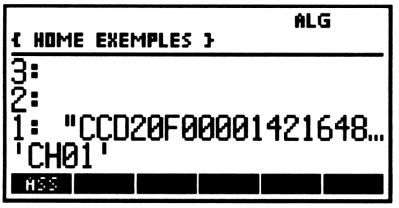
CCD 20

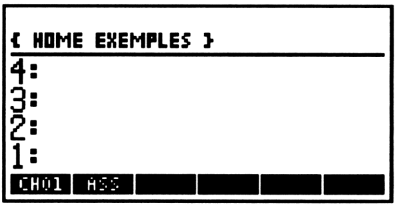
Placez les délimiteurs avec [↱] [-] puis appuyez à deux reprises sur [α] pour bloquer le clavier en mode alphabétique. Saisissez ensuite, sans espace ni saut de ligne, la chaîne :

"CCD20F0000142164808C"

Une fois la chaîne saisie, appuyez deux fois sur [ENTER] pour que la chaîne figure en deux exemplaires dans la pile. Le second exemplaire va servir à la vérification du checksum. Activez la commande BYTES en saisissant [↰] [VAR] [B]. Vous devez normalement obtenir #B6F7h pour le CRC et 25 octets pour la longueur de l'objet chaîne de caractères. Si tel n'est pas le cas, corrigez la chaîne jusqu'à ce que vous obteniez ces valeurs.







Nous allons enregistrer cette première chaîne de codes hexadécimaux dans une variable nommée 'CH01' (chaîne hexadécimale n°01). Pour cela, on commence par réafficher les variables en appuyant sur [VAR] on supprime les deux valeurs renvoyées par BYTES en saisissant [↶] [⇐] [↶] [⇐] (deux utilisations successives de DROP). On introduit ensuite le nom en saisissant ['] [α] [α] CH01 [α]. Une pression sur [STO] associe ce nom à la chaîne de codes au niveau 2 de la pile. La variable CH01 contenant la chaîne réapparaît dans le menu présentant les variables ([VAR]).

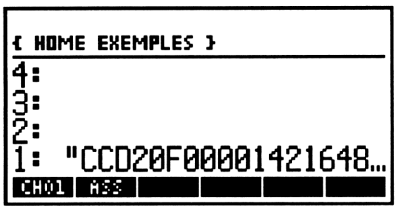
■ Créer le code exécutable à partir de la chaîne

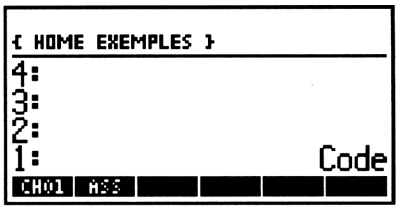
Nous disposons pour l'instant d'une chaîne de codes hexadécimaux (variable CH01) et d'un programme convertissant cette chaîne en code exécutable (programme ASS).

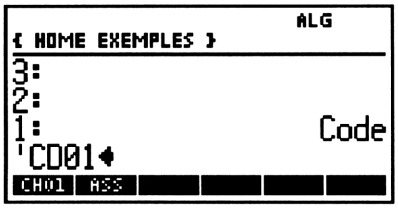
Pour obtenir le code exécutable relatif à une chaîne de codes hexadécimaux, il suffit de donner cette chaîne en argument au programme ASS. Pour cela, il suffit d'appuyer sur la touche de sélection d'article du menu se trouvant en face de l'article CH01 puis sur celle se trouvant en face de ASS. Il est aussi possible de saisir les noms de variables caractères en entrant [α] [α] CH01 [ENTER] [α] [α] ASS [ENTER].

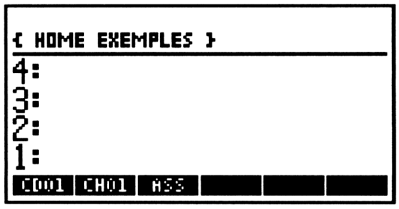
Dans les deux cas, on obtient Code au niveau 1 de la pile. Il s'agit du code exécutable. Nous pourrions à ce stade appuyer sur [EVAL] pour exécuter notre programme mais nous allons d'abord enregistrer le code exécutable dans une variable. Nous appellerons cette variable 'CD01' (code exécutable relatif à la chaîne n°01). On saisit donc ['] [α] [α] CD01 [α] [STO].

Il suffit maintenant d'évaluer CD01 pour exécuter notre programme !









Une pression sur la touche de sélection se trouvant au-dessous de l'article CD01 du menu provoque l'exécution du programme. Le sablier (témoin d'exécution) s'allume puis s'éteint et la machine est à nouveau utilisable normalement. Le programme s'est exécuté, sa seule fonction consistant à quitter sa propre exécution ! Cela dit, il s'agit d'un véritable programme en langage assembleur. Cet exemple était chargé de vous montrer comment un programme doit commencer et se finir...

Nous savons maintenant comment saisir une chaîne de codes hexadécimaux, trouver son CRC et la transformer en code exécutable à l'aide de ASS. L'exécution du programme écrit en langage d'assemblage se résume ensuite à l'évaluation de la variable relative au code exécutable.

Cependant, il est parfois indispensable de protéger le programme pour éviter son exécution à la suite d'une erreur de manipulation. C'est le cas de certains des jeux présentés dans la troisième partie de cet ouvrage. Il est alors impossible d'exécuter le programme en évaluant la variable relative au code exécutable. L'emploi d'un « lanceur », généralement un programme écrit en RPL, se charge de préparer la machine (contenu de la pile, etc.) à l'exécution du programme en assembleur. Le lanceur provoque ensuite l'évaluation (exécution) du code exécutable.

2.2

Prenons de bonnes habitudes

Nous savons maintenant qu'un programme écrit en assembleur commence par un prologue (02DCC) suivi de la longueur totale hors prologue (toujours codée sur 5 quartets). Nous savons aussi que le programme doit s'achever par `A=DAT0 A`, suivi de `D0=D0+ 5` et de `PC=(A)` ce qui revient à l'évaluation de l'objet suivant. Rappelons que tous les champs de paramètres sont inversés.

Un programme écrit en assembleur permet de contrôler la machine au plus bas niveau ce qui, vous vous en doutez, peut semer un certain désordre. Pour que la machine soit rendue au RPL dans un état utilisable, il faut que :

- les registres de travail B et D ainsi que D0 et D1 n'aient pas été modifiés,
- le registre P (registre pointeur de champ) ait la valeur #0h en fin de programme,
- le SATURN soit en mode hexadécimal.

Nous venons de dire que les registres B, D, D0 et D1 ne doivent pas être modifiés. En limitant énormément les capacités de nos programmes, nous pourrions presque nous passer des registres B et D. Par contre, il nous est impossible de nous passer de D0 et de D1. Ne pas modifier ces registres, revient à les restituer tels que nous les avons trouvés. L'astuce consiste donc à sauvegarder ces quatre registres dès le début du programme pour les récupérer à la fin du programme. Le programme pourra ainsi les utiliser sans perturber le fonctionnement de la HP-48.

Il existe dans la mémoire morte de la HP-48 une foule de petits programmes appelés « routines ». Un programme écrit en assembleur peut utiliser l'une de ces routines comme un sous-programme. Il suffit pour cela de préciser l'adresse en mémoire morte de la routine. La routine placée en #0679Bh sauvegarde les registres B, D , D0 et D1 alors que la routine se trouvant en #067D2h récupère ces sauvegardes et rend donc aux registres leurs valeurs d'origine. L'instruction dont le mnémonique est `GOSBVL` permet un appel de sous-programme ; ici, le sous-programme est une routine.

Nous placerons donc `GOSBVL 0679B` en début de programme et `GOSBVL 067D2` en fin de programme. Ainsi, nous pourrons utiliser librement B, D, D0 et D1.

Le registre P, appelé pointeur de champ, doit valoir #0h à la fin du programme. L'instruction `P= 0` devra donc figurer en fin de programme. Le SATURN doit être restitué en mode hexadécimal en fin de programme. L'instruction `SETHEX` devra donc aussi figurer en fin de programme.

Les interruptions sont un ensemble de processus pris en charge par le microprocesseur lorsque survient un événement (une pression sur la touche [ON], par exemple). Le traitement des interruptions est pris en charge par une routine très gourmande en ressources de la machine. Il est donc utile d'interdire les interruptions en début de programme mais il ne faudra surtout pas oublier de les autoriser à nouveau en fin de programme. On utilise donc `INTOFF` en début de programme et `INTON` en fin de programme.

Pour acquérir des habitudes de programmeur soigneux et ordonné, créons un programme prenant en charge toutes les opérations de « rangement » dont il vient d'être question. Les spécifications de ce programme sont énumérées ci-dessous :

- déclaration d'un code exécutable (1),
- déclaration de la longueur totale hors prologue (2),
- appel de la routine de sauvegarde de B, D, D0 et D1 (3),
- interdiction des interruptions (4),

- c'est ici, entre les étapes 4 et 5 de l'algorithme, que se trouve normalement le corps utile du programme (notre programme se contente de faire le ménage...),
- restitution des registres B, D, D0 et D1 par appel de la routine en #067D2h (5),
- chargement de #0h dans le registre P (6),
- autorisation des interruptions (7),
- mode hexadécimal du SATURN (8),
- rendre la main en évaluant l'objet suivant (9-10-11).

Nous allons donc :

- traduire les spécifications en mnémoniques,
- traduire les mnémoniques en codes hexadécimaux. Les quartets temporairement inconnus (la longueur hors prologue par exemple), étant remplacés par des « X »,
- on remplace ensuite les quartets inconnus par leurs valeurs.

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1) Signale un code exécutable
#00h	92000	*Longueur hp*	(2) Longueur hors prologue
#05h	8FB9760	GOSBVL 0679B	(3) Appel d'une routine en ROM
#0Ch	808F	INTOFF	(4) Interdiction des interruptions
			Corps utile du programme...
#10h	8F2D760	GOSBVL 067D2	(5) Appel d'une routine en ROM
#17h	20	P= 0	(6) Chargement de #0h dans P
#19h	8080	INTON	(7) Autorisation des interruptions
#1Dh	04	SETHEX	(8) Mode hexadécimal
#1Fh	142	A=DAT0 A	(9) On rend la main
#22h	164	D0=D0+ 5	(10)
#25h	808C	PC=(A)	(11) Branchement de l'exécution

Le code hexadécimal a une longueur hors prologue de 41 quartets, c'est-à-dire #29h. Sur 5 quartets, cette longueur devient #00029h, les champs étant inversés, on incorpore 92000 dans le code. Rappelons que les chapitres 13 à 17 présentent chaque instruction en détail.

Exactement comme nous l'avons déjà fait pour notre premier exemple, saisissons entre " et ", sans espace ni saut de ligne, la chaîne suivante :

```
"CCD20920008FB9760808F8F2D760208080041421648O8C"
```

A l'issue de la saisie de cette chaîne hexadécimale, appuyez deux fois sur [ENTER] pour obtenir deux exemplaires de la chaîne hexadécimale. Le second exemplaire va servir d'argument à BYTES que vous allez appeler

grâce à [←] [VAR] [B]. Normalement, vous devez obtenir le CRC #664Ch et une longueur de 51 octets. Si ce n'est pas le cas, corrigez la chaîne de codes hexadécimaux.

Enregistrez la chaîne dans une variable que vous nommerez CH02. Evaluez CH02 pour placer la chaîne dans la pile. Exécutez ASS pour obtenir le code exécutable que vous pouvez exécuter en appuyant sur [EVAL] ou sauvegarder dans une variable que vous nommerez CD02. L'évaluation de la variable CD02 provoque l'exécution de notre programme. Celui-ci ne produit rien mais est la base d'un programme sain qui laisse la HP-48 dans un état normal de fonctionnement à l'issue de son exécution.

2.3
Utiliser D1 pour un SWAP de la pile

Immédiatement après le début du programme, le registre D1 contient une adresse à laquelle se trouve l'adresse du contenu du niveau 1 de la pile. D1 contient donc l'adresse de l'adresse du contenu du niveau 1 de la pile.

La commande SWAP permet d'inverser les contenus des niveaux 1 et 2 de la pile. Nous allons recréer cette commande en assembleur. Ce qu'il faut savoir :

- on ne manipule pas directement les objets des différents niveaux de la pile mais on manipule les adresses des objets des différents niveaux de la pile,
- les adresses sont codées sur 5 quartets. On utilisera donc le champ A des registres de travail car sa capacité est adaptée,
- D1 contenant au départ l'adresse de l'adresse de l'objet du niveau 1 et les adresses étant codées sur 5 quartets, `D1=D1+ 5` fait pointer D1 sur l'adresse de l'adresse du niveau 2 de la pile (`D1=D1+ 10` pour le niveau 3 et ainsi de suite). De même, `D1=D1- 5` permet de redescendre d'un niveau (D1 désignera ainsi l'adresse de l'adresse de l'objet de niveau 1 s'il désignait auparavant l'adresse de l'adresse de l'objet de niveau 2).

Les spécifications du programme que nous allons créer sont les suivantes :

- déclaration d'un code exécutable (1),
- déclaration de la longueur totale hors prologue (2),
- appel de la routine de sauvegarde de B, D, D0 et D1 (3),
- interdiction des interruptions (4),

- copie dans le champ A du registre A des 5 quartets dont l'adresse est précisée par D1, c'est donc l'adresse du contenu du niveau 1 de la pile qui est ainsi placée dans le champ A du registre A (5),
- ajout de #5h au contenu de D1 pour que D1 contienne l'adresse de l'adresse du contenu du niveau 2 de la pile (6),
- copie dans le champ A du registre C de l'adresse du contenu du niveau 2 de la pile (7),
- D1 contient pour l'instant l'adresse de l'adresse du niveau 2 de la pile. On copie à cette adresse l'adresse de l'objet qui était à l'origine au niveau 1 (8),
- retrait de #5h au contenu de D1 pour que celui-ci contienne à nouveau l'adresse de l'adresse du contenu du niveau 1 de la pile (9),
- D1 contient pour l'instant l'adresse de l'adresse du niveau 1 de la pile. On copie à cette adresse l'adresse de l'objet qui était à l'origine au niveau 2. Les niveaux 1 et 2 sont donc maintenant inversés (10),
- restitution des registres B, D, D0 et D1 par appel de la routine en #067D2h (11),
- chargement de #0h dans le registre P (12),
- autorisation des interruptions (13),
- mode hexadécimal du SATURN (14),
- rendre la main en évaluant l'objet suivant (15-16-17).

Traduisons maintenant les spécifications en mnémoniques puis en codes hexadécimaux.

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1) Signale un code exécutable
#00h	B3000	*Longueur hp*	(2) Longueur hors prologue
#05h	8FB9760	GOSBVL 0679B	(3) Appel d'une routine en ROM
#0Ch	808F	INTOFF	(4) Interdiction des interruptions
			Début de la partie utile
#10h	143	A=DAT1 A	(5) Adresse niveau 1 dans A(A)
#13h	174	D1=D1+ 5	(6) Adresse du niveau 2
#16h	147	C=DAT1 A	(7) Adresse niveau 2 dans C(A)
#19h	141	DAT1=A A	(8) Ecriture en mémoire
#1Ch	1C4	D1=D1- 5	(9) Retour au niveau 1
#1Fh	145	DAT1=C A	(10) Ecriture en mémoire
			Fin de la partie utile
#22h	8F2D760	GOSBVL 067D2	(11) Appel d'une routine en ROM

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
#29h	20	P= 0	(12) Chargement de #0h dans P
#2Bh	8080	INTON	(13) Interruptions autorisées
#2Fh	04	SETHEX	(14) Mode hexadécimal
#31h	142	A=DAT0 A	(15) On rend la main
#34h	164	D0=D0+ 5	(16)
#37h	808C	PC=(A)	(17) Branchement de l'exécution

Soit un registre *R* et son champ *ch*, on note *R(ch)* le registre *R* limité au champ *ch*. A(A) désigne donc le champ A du registre A. Nous avons utilisé cette notation pour les commentaires du programme.

Les quartets étant numérotés à partir de #00h (#00h est le numéro du premier quartet codant la longueur totale hors prologue), la longueur totale hors prologue est obtenue en ajoutant le nombre de quartets du dernier code hexadécimal à son adresse. Dans le cadre de cet exemple, on a bien #37h + #4h = #3Bh. Après inversion des champs, on insère `B3000` dans le code à la suite du prologue.

Attention ! Ce programme ne vérifie pas que la pile contient au moins deux objets. Si vous exécutez ce programme alors que la pile ne contient pas au moins deux objets, des pertes de mémoire sont à prévoir !

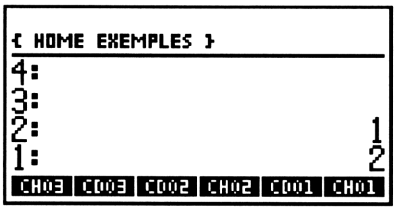
Sans espace ni saut de ligne, saisissez entre " et " la chaîne ci-dessous :

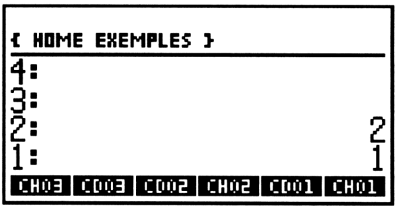
```
"CCD20B30008FB9760808F1431741471411C41458F2D760208080
04142164808C"
```

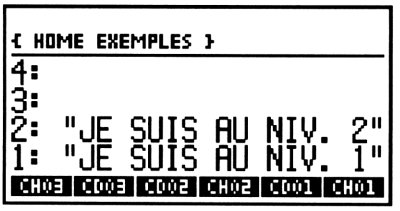
Une fois la chaîne introduite, appuyez deux fois sur [ENTER] pour obtenir deux exemplaires de la chaîne hexadécimale. Le second exemplaire sert d'argument à la commande BYTES qui doit renvoyer #C600h et 69. Si ce n'est pas le cas, vérifiez la chaîne hexadécimale.

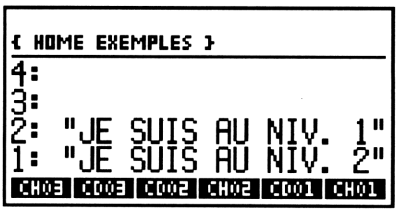
Enregistrez la chaîne dans une variable nommée 'CH03', placez la chaîne dans la pile, exécutez ASS et stockez le code exécutable obtenu dans une variable que vous nommerez 'CD03'.

Placez impérativement deux objets quelconques aux niveaux 1 et 2 de la pile. Evaluez CD03 pour exécuter le programme. Celui-ci a le même effet sur la pile que la commande SWAP mais ne vérifie pas la présence de deux objets aux deux premiers niveaux de la pile. Il est donc très dangereux d'utiliser ce programme si la pile ne remplit pas les conditions requises car vous risquez de provoquer des pertes de mémoire.









Cet exemple élémentaire illustre l'utilisation qui peut être faite du contenu de D1. Insistons sur le fait que D1 ne contient pas l'adresse du contenu du niveau 1 de la pile mais l'adresse de l'adresse du niveau 1 de la pile. Vous pouvez maintenant transposer cet exemple pour reproduire toutes les commandes de manipulation de la pile.

2.4

Modifier un objet de la pile

Sauf si nous modifions le contenu du registre D1, celui-ci contient l'adresse où se trouve l'adresse du contenu du niveau 1 de la pile. Cette particularité de D1 nous a permis de recréer le SWAP. Nous allons maintenant modifier le contenu du niveau 1 de la pile.

Notre nouveau programme va prendre en argument une chaîne de caractères (contenant au moins un caractère). Son premier caractère sera remplacé par le caractère de code ASCII immédiatement supérieur (voir le chapitre 7 traitant du code ASCII).

Nous savons qu'un code exécutable est un objet commençant par le prologue 02DCC suivi du codage sur 5 quartets de sa longueur hors prologue. De même, une chaîne de caractères est un objet. Son prologue est 02A2C, suivi de sa longueur hors prologue codée sur 5 quartets, elle-même suivie des codes ASCII de chaque caractère de la chaîne de caractères. Notre programme va lire le code ASCII (codé sur 2 quartets) du premier caractère de la chaîne et lui ajouter 1.

Les spécifications du programme :

- prologue 02DCC (1),
- longueur totale hors prologue (2),
- sauvegarde des registres (3),
- interdiction des interruptions (4),
- copie dans A(A) de l'adresse de l'objet du niveau 1 de la pile (5),
- cette adresse étant dans A(A), on lui ajoute #Ah pour sauter le prologue et la longueur totale hors prologue de la chaîne de caractères. L'adresse placée dans A(A) est donc celle du code ASCII du premier caractère de la chaîne de caractères placée au niveau 1 de la pile (6),
- copie de A(A) dans D1 pour que D1 contienne l'adresse du code ASCII du premier caractère (7),
- lecture du code ASCII qui est placé dans A(B) (8),
- ajout de 1 au code ASCII (9),
- écriture en mémoire du nouveau code ASCII (10)
- restitution des registres B, D, D0 et D1 par appel de la routine en #067D2h (11),
- chargement de #0h dans le registre P (12),
- autorisation des interruptions (13),
- mode hexadécimal du SATURN (14),
- rendre la main en évaluant l'objet suivant (15-16-17).

Attention ! Si l'objet au niveau 1 de la pile n'est pas une chaîne de caractères, le programme fera n'importe quoi car il ne vérifie pas qu'il y a un objet au niveau 1 et que celui-ci est une chaîne de caractères.

Traduisons les spécifications en mnémoniques puis en codes hexadécimaux.

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1) Signale un code exécutable
#00h	E3000	*Longueur hp*	(2) Longueur hors prologue
#05h	8FB9760	GOSBVL 0679B	(3) Appel d'une routine en ROM
#0Ch	808F	INTOFF	(4) Interdiction des interruptions
			Début de la partie utile
#10h	143	A=DAT1 A	(5) Adresse niveau 1 dans A(A)
#13h	818F09	A=A+#Ah A	(6) Saut vers adresse 1er caractère
#19h	131	D1= A	(7) D1 pointe sur le code ASCII
#1Ch	14B	A=DAT1 B	(8) Code ASCII copié dans A(B)
#1Fh	B64	A=A+1 B	(9) Incrémentation code ASCII
#22h	149	DAT1=A B	(10) Ecriture du nouveau caractère
			Fin de la partie utile

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
#25h	8F2D760	GOSBVL 067D2	(11) Appel d'une routine en ROM
#2Ch	20	P= 0	(12) Chargement de #0h dans P
#2Eh	8080	INTON	(13) Interruptions autorisées
#32h	04	SETHEX	(14) Mode hexadécimal
#34h	142	A=DAT0 A	(15) On rend la main
#37h	164	D0=D0+ 5	(16)
#3Ah	808C	PC=(A)	(17) Branchement de l'exécution

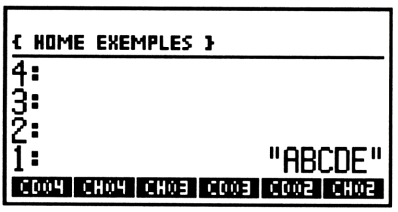
Saisissez sans espace ni saut de ligne et entre " et " la chaîne ci-dessous :

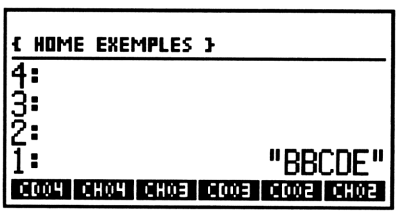
"CCD20E30008FB9760808F143818F0913114BB641498F2D760208 08004142164808C"

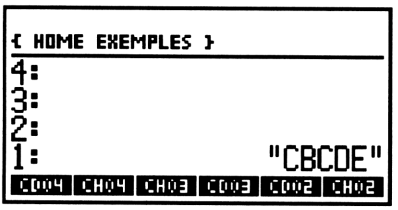
Après avoir introduit cette chaîne de caractères, appuyez deux fois sur [ENTER] pour en obtenir deux exemplaires. Le second sert d'argument à la fonction BYTES qui doit renvoyer #F508h et 72. Si tel n'est pas le cas, corrigez la chaîne. Enregistrez ensuite la chaîne de caractères sous le nom 'CH04'. Placez-la dans la pile et exécutez ASS. Enregistrez le code exécutable obtenu dans une variable que vous nommerez CD04.

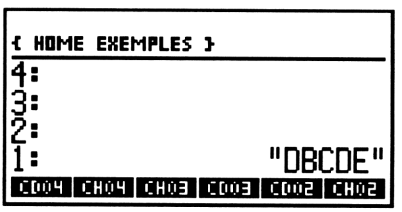
Il suffira d'évaluer CD04 pour exécuter le programme. Attention ! Ce programme écrit en mémoire. L'objet placé au niveau 1 de la pile doit impérativement être une chaîne de caractères. Rappelons qu'une chaîne de caractères est placée entre " et ".

Vous connaissez maintenant le principe d'une modification d'objet dans la pile. Connaissant l'adresse de l'objet, il suffit de placer sa partie à modifier dans un registre de travail puis d'écrire le champ qui vient d'être modifié









à la place de celui qui devait être modifié. Pour cela, le chapitre 22 où sont décrites les structures des objets est un outil indispensable.

3
Les structures de base

Les instructions d'un programme sont exécutées en séquence, c'est-à-dire les unes après les autres. Cependant, les sauts permettent de bouleverser cet ordre en faisant reprendre l'exécution à partir d'un point de branchement déterminé par le programmeur (voir le chapitre 16 sur les sauts et tests). On distingue les sauts absolus où le point de branchement est défini par son adresse, et les sauts relatifs ou le point de branchement est défini par un décalage d'adresses à partir de l'instruction de saut.

3.1
Les tests

Reprenons le programme stocké dans la variable CD04 (la chaîne correspondante est dans CH04). Ce programme incrémente le code ASCII du premier caractère d'une chaîne de caractères. Malheureusement, il ne vérifie pas qu'un objet est bien présent dans la pile et que cet objet est une chaîne de caractères. Nous allons donc ajouter des tests pour n'autoriser l'exécution du programme que si une chaîne se trouve au niveau 1 de la pile et qu'à condition que cette chaîne soit non vide.

Le programme va effectuer trois tests successifs. Le premier vérifie que la pile n'est pas vide, le deuxième vérifie que l'objet au niveau 1 est une chaîne de caractères alors que le troisième vérifie que cette chaîne n'est pas vide. Si l'une de ces vérifications détecte un cas à éviter, l'exécution reprend à la fin du programme et celui-ci prend fin sans avoir modifier la mémoire.

Un test est toujours suivi de `GOYES` ou de `RTNYES`. Ces instructions déterminent ce qui devra être fait si le test renvoie la valeur « vrai ». `GOYES` pro-

voque un saut relatif limité à 128 quartets vers le début ou la fin du programme. `RTNYES` provoque un retour de sous-programme. Un test réussi provoque donc, soit un saut, soit un retour de sous-programme.

Les spécifications du programme :

- prologue 02DCC (1),
- longueur totale hors prologue (2),
- sauvegarde des registres (3),
- interdiction des interruptions (4),
- copie dans A(A) de l'adresse de l'objet du niveau 1 (5),
- cette adresse est-elle nulle ? (6),
- si oui, cela signifie que le niveau 1 est vide, on saute alors vers le label Fin (7),
- copie dans D1 de l'adresse de l'objet du niveau 1 (8),
- copie dans A(A) du prologue de l'objet du niveau 1 (9),
- chargement dans C(A) du prologue d'un objet de type chaîne de caractères (10),
- le prologue de l'objet du niveau 1 de la pile est-il différent du prologue d'un objet de type chaîne de caractères ? (11)
- si oui, l'objet n'est pas une chaîne de caractères, on saute alors vers le label Fin (12),
- on saute les 5 quartets du prologue pour que D1 pointe sur le longueur totale hors prologue de la chaîne de caractères (13),
- copie dans A(A) de la longueur totale hors prologue de la chaîne de caractères (14),
- mise à zéro de C(A) (15),
- la longueur totale hors prologue de la chaîne de caractères est-elle nulle ? (16)
- si oui, la chaîne est vide (chaîne `""`), on saute alors vers le label Fin (17),
- on saute les 5 quartets de la longueur totale hors prologue pour que D1 pointe sur le premier code ASCII de la chaîne de caractères (18),
- copie dans A(B) du code ASCII (19),
- incrémentation du code ASCII (20),
- remplacement en mémoire du code ASCII du premier caractère de la chaîne (21),
- label Fin. Restitution des registres B, D, D0 et D1 par appel de la routine en #067D2h (22),
- chargement de #0h dans le registre P (23),
- autorisation des interruptions (24),
- mode hexadécimal du SATURN (25),
- rendre la main en évaluant l'objet suivant (26-27-28).

Les étapes 15 et 16 de l'algorithme peuvent être remplacées par `?A=0 A`.

Traduisons les spécifications en mnémoniques puis en codes hexadécimaux. Nous sommes obligés de remplacer les 5 quartets codant la longueur totale hors prologue par des « XXXXX » tant que nous ne connaissons pas le nombre de quartets du code. De même, chaque instruction GOYES devra être remplacée par « XX ». Nous ne pourrons calculer les valeurs des « XX » que lorsque les adresses des codes de chaque instruction seront connues.

CCD 20

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1) Signale un code exécutable
#00h	C5000	*Longueur hp*	(2) Longueur hors prologue
#05h	8FB9760	GOSBVL 0679B	(3) Appel d'une routine en ROM
#0Ch	808F	INTOFF	(4) Interdiction des interruptions
			Le niveau 1 est-il vide ?
#10h	143	A=DAT1 A	(5) Adresse niveau 1 dans A(A)
#13h	8A8	?A=0 A	(6) Test : adresse nulle ?
#16h	D2	Goyes FIN	(7) Si oui, sauter vers FIN
			Y a-t-il une chaîne au niveau 1 ?
#18h	131	D1=A	(8) D1 pointe sur le prologue
#1Bh	143	A=DAT1 A	(9) Copie du prologue dans A(A)
#1Eh	34C2A20	LC 4 02A2C	(10) Chargement dans C(A)
#25h	8A6	?A≠C A	(11) Test : égalité des prologues ?
#28h	B1	Goyes FIN	(12) Non, alors sauter vers FIN
			Cette chaîne est-elle vide ?
#2Ah	174	D1=D1+5	(13) D1 pointe sur longueur hp
#2Dh	143	A=DAT1 A	(14) Copie dans A de longueur hp
#30h	D2	C=0 A	(15) Initialisation à 0 de C(A)
#32h	8A2	?A=C A	(16) Test : A(A) est-il nul ?
#35h	E0	Goyes FIN	(17) Si oui, sauter vers FIN
			Incrémentation du code ASCII
#37h	174	D1=D1+5	(18) D1 pointe sur le caractère 1
#3Ah	14B	A=DAT1 B	(19) Copie dans A(B) du car. 1
#3Dh	B64	A=A+1 B	(20) Incrémentation
#40h	149	DAT1=A B	(21) Ecriture du nouveau car.
			Fin de la partie utile
		FIN (label)	Point de branchement
#43h	8F2D760	GOSBVL 067D2	(22) Appel d'une routine en ROM
#4Ah	20	P= 0	(23) Chargement de #0h dans P
#4Ch	8080	INTON	(24) Interruptions autorisées
#50h	04	SETHEX	(25) Mode hexadécimal
#52h	142	A=DAT0 A	(26) On rend la main
#55h	164	D0=D0+ 5	(27)
#58h	808C	PC=(A)	(28) Branchement de l'exécution

L'instruction GOYES est toujours placée immédiatement après un test. Elle n'est codée que par les deux quartets déterminant le décalage d'adresses du saut relatif. Dans le cadre de cet exemple, le label FIN correspond au branchement de l'exécution sur le premier quartet de 8F2D760. Insistons sur le fait qu'un label n'est en aucun cas une instruction (le label n'est pas exécutable). Il s'agit juste d'un point de repère dans le programme. Ici, le label FIN correspond au branchement de l'exécution sur le quartet d'adresse #43h. Il suffit de calculer le décalage d'adresses entre le point de branchement et l'instruction GOYES pour savoir comment coder les instructions GOYES.

Toujours dans le cadre de cet exemple, les trois instructions GOYES FIN se trouvent aux adresses #16h, #28h et #35h. FIN étant à l'adresse #43h, on calcule #43h-#16h=#2Dh, #43h-#28h=#1Bh et #43h-#35h=#Eh. Après codage sur 2 quartets et inversion des champs, les trois GOYES FIN se codent respectivement D2, B1 et E0.

Il s'agit ici d'un cas simple puisque les sauts se font vers la fin du programme. Dans le cas d'un saut vers le début du programme, il faut faire appel au complément à 2 pour le codage du décalage (voir le chapitre 16).

Il est très important de suivre l'évolution des contenus des différents registres au cours de l'exécution pour comprendre le fonctionnement du programme. Lorsqu'un test est « vrai », on effectue un saut relatif, autrement dit, l'exécution reprend *n* quartets avant ou après l'instruction de test.

Pour introduire le programme, saisissez, sans espace ni saut de ligne et entre " et ", la chaîne ci-dessous :

```
"CCD20C50008FB9760808F1438A8D213114334C2A208A6B117414
3D28A2E017414BB641498F2D760208080041421648O8C"
```

Une fois la chaîne hexadécimale introduite, appuyez deux fois sur [ENTER] pour en obtenir deux exemplaires dans la pile. Le second exemplaire sert d'argument à la commande BYTES qui doit renvoyer #3D87h et 102. Si ce n'est pas le cas, corrigez la chaîne hexadécimale.

Enregistrez ensuite la chaîne dans une variable que vous nommerez 'CH05'. Envoyez la chaîne dans la pile, exécutez ASS et stockez le code hexadécimal obtenu dans une variable que vous nommerez 'CD05'.

Il suffit d'évaluer CD05 pour exécuter le programme. Si une chaîne non vide se trouve au niveau 1 de la pile, le code ASCII de son premier carac-

tère est incrémenté. Si la pile est vide, si elle contient un objet autre qu'une chaîne ou si elle contient une chaîne vide, alors le programme reste sans effet.

3.2
Les sauts relatifs inconditionnels

GOTO est l'une des instructions de saut relatif inconditionnel. Lorsqu'un GOTO est rencontré, l'exécution reprend à partir d'une adresse définie par un décalage d'adresses à partir de celle de GOTO.

Le programme que nous allons développer comporte une boucle. A chaque exécution de la boucle le premier caractère de la chaîne de caractères placée au niveau 1 de la pile est remplacé par le caractère de code ASCII immédiatement supérieur (avec retour au code #00h lorsque #FFh est atteint).

Les spécifications :

- prologue, longueur totale hors prologue (1),
- sauvegarde des registres indispensables et interdiction des interruptions (2),
- chargement dans A(A) de l'adresse de la chaîne de caractères se trouvant au niveau 1 de la pile (3),
- copie dans D1 de cette adresse (4),
- ajout de #Ah à D1 pour qu'il pointe sur le premier caractère de la chaîne (5),
- copie dans A(B) du code ASCII du premier caractère (6),
- initialisation du compteur de P à la valeur #0h (7),
- début de la boucle et ajout de #1h au compteur P (8),
- ajout de #1h à A(B) (9),
- P vaut-il 5 ? (10),
- si oui, reprendre l'exécution à partir du label FIN (11),
- dans la négative reprendre l'exécution au début de la boucle (12),
- label FIN et écriture en mémoire de A(B) à la place du code ASCII du premier caractère de la chaîne (13),
- récupération des sauvegardes de registres (14),
- remise à zéro de P (15),
- autorisation des interruptions (16),
- mode hexadécimal (17),
- on quitte le programme (18).

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	***Prologue***	(1) Signale un code exécutable
#00h	84000	***Longueur hp***	
#05h	8FB9760	GOSBVL 0679B	(2) Sauvegarde des registres puis
#0Ch	808F	INTOFF	interdiction des interruptions
#10h	143	A=DAT1 A	(3) Adresse de la chaîne dans A(A)
#13h	131	D1=A	(4) Adresse de la chaîne dans D1
#16h	179	D1=D1+ #Ah	(5) D1 pointe sur le 1er caractère
#19h	14B	A=DAT1 B	(6) Code ASCII dans A(B)
#1Ch	20	P= 0	(7) Initialisation du compteur
		BOUCLE	Label BOUCLE
#1Eh	0C	P=P+1	(8) Incrémentation du compteur
#20h	B64	A=A+1 B	(9) Incrémentation du code ASCII
#23h	895	?P=5	(10) Test : 5 boucles ?
#26h	60	Goyes FIN	(11) Si oui, saut vers FIN
#28h	65FF	Goto BOUCLE	(12) Retour en début de boucle
		FIN	Label FIN
#2Ch	149	DAT1=A B	(13) Ecriture du nouveau caractère
			Fin de la partie utile
#2Fh	8F2D760	GOSBVL 067D2	(14) Récupération des registres
#36h	20	P= 0	(15) Initialisation de P
#38h	8080	INTON	(16) Autorisation des interruptions
#3Ch	04	SETHEX	(17) Mode hexadécimal
#3Eh	142	A=DAT0 A	(18) On rend la main
#41h	164	D0=D0+ 5	
#44h	808C	PC=(A)	

Le registre P est utilisé comme compteur. Il est initialisé à #0h et augmenté de #1h à chaque exécution des instructions de la boucle. On quitte la boucle lorsque P atteint #5h.

Tant que P est inférieur à #5h, `Goyes FIN` ne peut pas être exécuté car la condition `?P=5` n'est pas remplie, `Goyes FIN` est donc ignoré. On passe dans ce cas à `Goto BOUCLE` qui s'exécute normalement. `Goyes` est conditionnel car il est directement soumis à une condition. `Goto` est inconditionnel car il suffit qu'il soit rencontré au cours de l'exécution pour qu'il soit exécuté.

Lorsque P est égal à #5h, la condition nécessaire à l'exécution de `Goyes FIN` est remplie. Ce branchement de l'exécution fait que

`Goto BOUCLE` n'est pas rencontré et n'est donc pas exécuté. Alors que l'instruction `P= 0` en fin de programme était inutile dans nos autres programmes, elle est ici absolument indispensable car nous avons modifié le registre P.

Comme d'habitude nous remplaçons lors de la traduction en codes hexadécimaux les quartets inconnus par des « X ». La longueur totale hors prologue s'obtient en ajoutant à l'adresse du dernier champ (#44h) la longueur de celui-ci (#4h). #44h+#4h = #48h, on insère donc #84000h dans le code après inversion des champs. Pour coder `Goyes FIN`, on calcule le décalage entre l'adresse de `Goyes FIN` (#26h) et l'adresse du premier quartet suivant le label FIN (#2Ch). #2Ch-#26h=#Eh, donc, sur 2 quartets et après inversion des champs, `Goyes FIN` se code `60`. Le codage de `Goto BOUCLE` est un peu plus compliqué puisque le décalage d'adresses est négatif. L'adresse de `Goto BOUCLE` est #28h, celle du label BOUCLE est #1Eh. On calcule #28h+#1h-#1Eh=#Bh. #1h est lié au fait que `Goto`, hors le codage du décalage, est codé sur un quartet. La décalage étant négatif, on calcule #0h-#Bh, on supprime les #Fh sur la gauche pour ne conserver que trois quartets à droite, on obtient ainsi #FF5h. On ajoute `6` à gauche du champ inversé, on obtient ainsi `65FF` qui est le codage de Goto BOUCLE. Voir le chapitre 16 pour plus de détails.

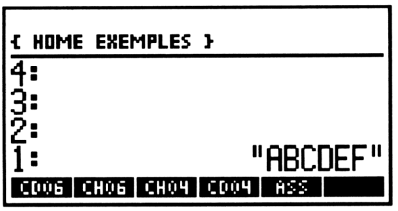
Sans espace ni saut de ligne, saisissez entre " et " la chaîne ci-dessous :

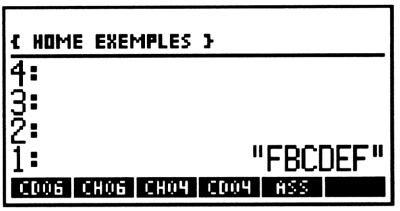
```
"CCD20840008FB9760808F14313117914B200CB648956065FF149
8F2D7602080800414216 4808C"
```

Une fois la chaîne introduite, appuyez deux fois sur [ENTER] pour placer dans la pile deux exemplaires de la chaîne. Le second exemplaire servira d'argument à BYTES qui doit renvoyer #D87h et 82. Si ce n'est pas le cas, vérifiez la chaîne. Enregistrez-la ensuite dans une variable que vous nommerez 'CH06'. Envoyez cette chaîne au niveau 1 de la pile puis exécutez ASS. Stockez le code exécutable obtenu dans une variable que vous nommerez 'CD06'.

Pour exécuter ce programme, placez une chaîne de caractères (entre " et ") au niveau 1 de la pile. Evaluez ensuite CD06. Le premier caractère de la chaîne ASCII se trouvant au niveau 1 de la pile voit 5 ajouté à son code ASCII. Par exemple, A sera remplacé par F.

Attention ! Contrairement au précédent, ce programme ne vérifie pas que le niveau 1 de la pile contient une chaîne non vide. N'exécutez surtout pas ce programme si le niveau 1 de la pile ne contient pas une chaîne non vide.





3.3
Les sauts relatifs conditionnels

Il existe plusieurs méthodes pour soumettre un saut relatif à une condition. Un test suivi de `Goyes` en est une que nous avons déjà utilisée.

Le SATURN comporte une sorte de mini registre limité à 1 bit et appelé bit de retenue ou carry. Ce bit peut être armé (mis à 1) ou désarmé (mis à 0) par certaines instructions.

Nous allons reprendre le programme précédent en décrémentant le registre P initialisé à #4h. L'instruction `P=P-1` armera le bit de retenue lorsqu'il ne sera plus possible de décrémenter P (quand il vaut déjà 0). La valeur de la retenue sera exploitée par `Gonc` qui provoque un saut vers le début de la boucle tant que le bit de retenue vaudra 0. Cela dit, nous utiliserons aussi au sein de la boucle l'instruction `A=A+1 B` qui modifie aussi la retenue. Cette instruction désarme la retenue dans le cas normal et l'arme en cas de dépassement de capacité.

L'exécution reprend à partir du début de la boucle en fonction de la valeur du bit de retenue. La boucle comporte deux instructions modifiant le bit de retenue. L'ordre de ces instructions n'est donc pas quelconque !

Description de notre programme :

- prologue, longueur totale hors prologue (1),
- sauvegarde des registres indispensables et interdiction des interruptions (2),
- chargement dans A(A) de l'adresse de la chaîne de caractères se trouvant au niveau 1 de la pile (3),
- copie dans D1 de cette adresse (4),
- ajout de #Ah à D1 pour qu'il pointe sur le premier caractère de la chaîne (5),
- copie dans A(B) du code ASCII du premier caractère (6),

- initialisation du compteur de P à la valeur #4h (7),
- début de la boucle et ajout de #1h à A(B) (8),
- décrémentation de P (9), la retenue est armée si P vaut déjà #0h avant la décrémentation,
- si la retenue n'est pas armée, retour vers 8 (10),
- écriture en mémoire de A(B) à la place du code ASCII du premier caractère de la chaîne (11),
- récupération des sauvegardes de registres (12),
- remise à zéro de P (13),
- autorisation des interruptions (14),
- mode hexadécimal (15)
- on quitte le programme (16).

CCD 20

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1) Signale un code exécutable
#00h	24000	*Longueur hp*	
#05h	8FB9760	GOSBVL 0679B	(2) Sauvegarde des registres puis
#0Ch	808F	INTOFF	interdiction des interruptions
#10h	143	A=DAT1 A	(3) Adresse de la chaîne dans A(A)
#13h	131	D1=A	(4) Adresse de la chaîne dans D1
#16h	179	D1=D1+ #Ah	(5) D1 pointe sur le 1er caractère
#19h	14B	A=DAT1 B	(6) Code ASCII dans A(B)
#1Ch	24	P= 4	(7) Initialisation du compteur
		BOUCLE	Label BOUCLE
#1Eh	B64	A=A+1 B	(8) Incrémentation du code ASCII
#20h	0D	P=P-1	(9) Décrémentation du compteur
#23h	5AF	GONC BOUCLE	(10) Retour en début de boucle
#26h	149	DAT1=A B	(11) Ecriture du nouveau caractère
			Fin de la partie utile
#29h	8F2D760	GOSBVL 067D2	(12) Récupération des registres
#30h	20	P= 0	(13) initialisation de P
#32h	8080	INTON	(14) Autorisation des interruptions
#36h	04	SETHEX	(15) Mode hexadécimal
#38h	142	A=DAT0 A	(16) On rend la main...
#3Bh	164	D0=D0+ 5	
#3Eh	808C	PC=(A)	

L'utilisation du bit de retenue rend ce programme plus simple que le précédent. Cependant, il faut savoir que de nombreuses instructions modifient le bit de retenue. Ainsi, un test arme le bit de retenue s'il est vérifié,

une instruction d'incrémentation, de décrémentation, d'addition ou de soustraction arme le bit de retenue en cas de dépassement de capacité du champ concerné et le désarme sinon.

Lorsque la valeur du bit de retenue est utilisée, il convient d'ordonner avec soin les différentes instructions modifiant le bit de retenue.

L'instruction GONC est codée sur 3 quartets : 5 suivi de 2 quartets codant le décalage. Il s'agit d'un saut vers le début du programme. On ajoute #1h à l'adresse de GONC puis on soustrait l'adresse du point de branchement. On calcule donc #23h+#1h-#1Eh=#6h. On calcule ensuite #0h-#6h=#FAh car il s'agit d'un décalage d'adresses. Après inversion des champs GONC BOUCLE est codé 5AF. Voir le chapitre 16 pour plus de détails sur la technique de calcul du décalage d'adresses.

Pour introduire ce programme, saisissez sans espace ni saut de ligne et entre " et ", la chaîne ci-dessous :

```
"CCD20240008FB9760808F14313117914B24B640D5AF1498F2D76
020808004142164808C"
```

Une fois la chaîne introduite, appuyez deux fois sur [ENTER] pour en obtenir deux exemplaires dans la pile. Le second exemplaire sert d'argument à la fonction BYTES qui doit renvoyer #184Ch et 76. Corrigez la chaîne si ce n'est pas le cas. Ensuite, enregistrez-la dans une variable que vous nommerez 'CH07'. Envoyez la chaîne dans la pile, exécutez ASS et enregistrez le code exécutable obtenu dans une variable que vous nommerez 'CD07'.

Pour exécuter le programme, placez une chaîne quelconque au niveau 1 de la pile (une chaîne de caractères est délimitée par " et "). Evaluez ensuite CD07.

Attention ! Ce programme n'effectue aucune vérification. Si le niveau 1 de la pile ne contient pas une chaîne non vide, le programme écrira des données aberrantes en mémoire. Nous avons déjà commenté la réalisation d'un programme vérifiant que le niveau 1 de la pile contient une chaîne non vide. Vous devriez maintenant être capable d'ajouter au programme CD07 les tests nécessaires...

Nous savons qu'un test est associé à un saut relatif conditionnel. Malheureusement, ce saut relatif est limité à 128 quartets. L'astuce consiste donc à tester la condition inverse pour pouvoir utiliser une instruction de saut relatif non conditionnel.

Ainsi :

```
?A=C  A
Goyes très_loin
```

...n'est pas acceptable si le label *très_loin* se trouve à plus de 128 quartets, mais...

```
?A≠C  A
Goyes tout_près
Goto très_loin
Label tout_près
```

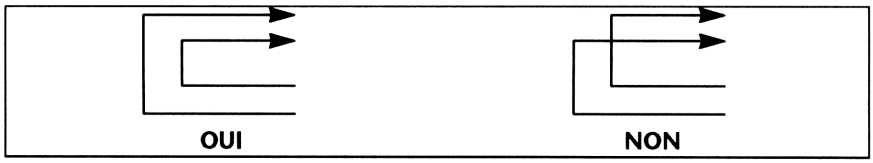
...nous permet de faire des sauts relatifs longs selon le résultat du test. Bien que Goto ne soit pas conditionnel, il l'est ici indirectement puisque l'instruction `Goto très_loin` ne peut être exécutée que si A(A) et C(A) sont égaux.

3.4
Les boucles

Une boucle peut être assimilée à un saut répété vers le début du programme. Nous avons déjà utilisé plusieurs types de boucles se distinguant par leurs sorties respectives. En effet, c'est la condition de sortie d'une boucle qui fait son originalité. Pour sortir d'une boucle, on peut utiliser :

- un test dont le Goyes associé ne branchera plus l'exécution sur le début de la boucle lorsque la condition requise ne sera plus satisfaite,
- un compteur (le registre P si 16 valeurs sont suffisantes), la valeur de P contrôle alors la sortie de la boucle. Voir l'exemple CD06 où `?P=5, Goyes FIN, Goto BOUCLE` peuvent être remplacés par `?P≠5, Goyes BOUCLE`,
- la valeur du bit de retenue. Voir l'exemple CD07.

Les boucles peuvent être imbriquées à condition de ne pas utiliser la même condition de sortie mais ne doivent pas se chevaucher.



3.5
Les sauts absolus

Nous faisons appel à des sauts lorsque la prochaine instruction à exécuter n'est pas celle se trouvant immédiatement après celle qui vient d'être exécutée. Tous les sauts que nous avons employés jusqu'à présent sont des sauts relatifs. Autrement dit, on s'écarte de *n* quartets à partir de l'instruction de saut.

Il est aussi possible d'indiquer une adresse précise en mémoire. On parle alors de saut absolu.

Vous avez sans doute remarqué que tous nos programmes s'achèvent par `A=DAT0 A`, `D0=D0+5` et `PC=(A)`. Ces trois instructions provoquent l'évaluation de l'objet suivant. On quitte ainsi le programme écrit en assembleur.

Il y a à l'adresse #12002h de la mémoire morte (ROM) un fragment de programme qui correspond très exactement à `A=DAT0 A`, `D0=D0+5` et `PC=(A)`. Il suffit donc de sauter vers l'adresse #12002h pour quitter le programme !

`GOVLNG 12002` branche l'exécution sur l'adresse #12002h de la ROM. Ainsi, notre premier programme (CD01) qui se contentait de quitter sa propre exécution, devient :

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	***Prologue***	(1) Signal
#00h	**C0000**	***Longueur hp***	(2) Longueur hors prologue
#05h	**8D20021**	**Govlng 12002**	(3) On quitte le programme

Ce saut long nous permet de remplacer trois instructions par une seule. Nous l'utiliserons donc désormais en fin de programme.

Introduisez sans espace ni saut de ligne et entre " et " la chaîne ci-dessous :

"CCD20C00008D20021"

Une fois cette chaîne introduite, appuyez deux fois sur [ENTER]. Utilisez le second exemplaire de la chaîne comme argument de la fonction BYTES qui doit normalement renvoyer #11F6h et 22. Si ce n'est pas le cas, cor-

rigez la chaîne. Enregistrez-la ensuite dans une variable que vous nommerez 'CH08'. Placez cette chaîne dans la pile et exécutez ASS. Mémorisez le code exécutable obtenu dans une variable que vous nommerez CD08. Il suffit ensuite d'évaluer CD08 pour exécuter le programme. Celui-ci a exactement le même effet que CD01, autrement dit, aucun effet !

3.6
Les appels de sous-programmes

Certaines portions d'un programme qui doivent être appelées depuis plusieurs points du programme gagnent à être placées dans un sous-programme. Lorsqu'un sous-programme est appelé, une adresse de retour est placée au sommet de la pile RSTK (pile des retours). L'adresse de retour est l'adresse de l'instruction se trouvant immédiatement après l'instruction commandant l'appel du sous-programme. Aucune instruction particulière ne signale un début d'un sous-programme, par contre, une instruction ordonne le retour. L'exécution reprend alors à partir de l'adresse de retour.

Les instructions `GOSBVL` que nous utilisons depuis nos premiers programmes sont des appels de sous-programmes (des routines) par saut absolu en mémoire morte. L'utilisateur appelle ses propres sous-programmes au moyen de sauts relatifs (`GOSUB`) car il ne connaît pas l'adresse à laquelle se trouve son programme en mémoire, par contre il peut compter le nombre de quartets entre l'adresse de retour (adresse de l'instruction suivant l'instruction d'appel de sous-programme) et l'adresse du premier quartet de la première instruction du sous-programme.

Nous allons reprendre le programme CD07 en plaçant les instructions de sa boucle dans un sous-programme. Cette modification ne présente pas d'intérêt particulier dans le cadre de cet exemple, mais l'utilisation de sous-programmes devient incontournable lorsqu'une même procédure doit être utilisée à partir d'un point quelconque du programme.

Description de notre programme :

- prologue, longueur totale hors prologue, sauvegarde des registres B, D, D0 et D1 (1),
- comme pour CD07, on charge dans A(B) le code ASCII du premier caractère de la chaîne se trouvant au niveau 1 de la pile (2),
- initialisation du registre P à 4 (3),
- début de la boucle, appel du sous-programme (4),

- retour au début de la boucle si la retenue n'est pas armée (5),
- écriture en mémoire du nouveau premier caractère de la chaîne (6),
- récupération des registres, mise à zéro de P, autorisation des interruptions (ici inutile puisqu'elles n'ont pas été interdites !), mode hexadécimal (7),
- évaluation de l'objet suivant pour quitter le programme (8),
- début du sous-programme et incrémentation de A(B) (9),
- décrémentation de P avec armement de la retenue si P vaut déjà 0 quand on tente de le décrémenter (10),
- retour inconditionnel de sous-programme (11).

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1) Signale un code exécutable
#00h	14000	*Longueur hp*	
#05h	8FB9760	GOSBVL 0679B	
#0Ch	143	A=DAT1 A	(2)
#0Fh	131	D1=A	
#12h	179	D1=D1+ #Ah	
#15h	14B	A=DAT1 B	
#18h	24	P=4	(3) Initialisation du compteur
		BOUCLE	Label BOUCLE
#1Ah	7C10	Gosub SPRG	(4) Appel du sous-programme
#1Eh	5BF	Gonc BOUCLE	(5) Saut conditionnel selon retenue
#21h	149	DAT1=A B	(6) Ecriture en mémoire
#24h	8F2D760	GOSBVL 067D2	(7)
#2Bh	20	P=0	
#2Dh	8080	INTON	
#31h	04	SETHEX	
#33h	8D20021	GOVLNG 12002	(8) On quitte le programme
			Fin du programme principal
		SPRG	Début du sous-programme SPRG
#3Ah	B64	A=A+1 B	(9)
#3Dh	0D	P=P-1	(10)
#3Fh	01	RTN	(11)

Nous n'avons pas jugé utile de commenter à nouveau les parties du programme déjà vues dans CD06 et CD07.

Ce programme rassemble deux appels de sous-programmes en ROM (`GOSBVL 0679B` et `GOSBVL 067D2`), un saut absolu (`GOVLNG 12002`),

un saut relatif conditionnel (GONC BOUCLE) et appel de sous-programme par saut relatif (GOSUB SPRG).

Il nous faut calculer deux décalages d'adresses, l'un pour GONC, l'autre pour GOSUB. Les techniques de calcul des décalages sont exposées au chapitre 16, nous ne ferons ici qu'appliquer les formules. GONC se trouve à l'adresse #1Eh, le label BOUCLE correspond au quartet d'adresse #1Ah. On calcule donc #1Eh+#1h-#1Ah=#5h, puis, le décalage étant négatif, #0h-#5h=#FBh. GONC BOUCLE sera donc codé 5BF. Pour coder l'instruction GOSUB SPRG, il faut connaître l'adresse de retour de sous-programme (ici, #1Eh) et l'adresse du premier quartet du sous-programme (ici, #3Ah). En calculant #3Ah-#1Eh on obtient #1Ch qui est la valeur du décalage associé à GOSUB SPRG qui sera donc codé 7C10.

Remarquez que les instructions de fin de programme (celles qui servent à le quitter), ne sont pas les dernières instructions du programme. La dernière instruction est ici une instruction de retour de sous-programme.

Pour introduire le programme, saisissez sans espace ni saut de ligne et entre " et " la chaîne ci-dessous :

"CCD20140008FB976014313117914B247C105BF1498F2D7602080
80048D20021B640D01"

Une fois cette chaîne introduite appuyez deux fois sur [ENTER] pour en obtenir deux exemplaires. Le second exemplaire sert d'argument à BYTES qui doit renvoyer #E4CBh et 75. Si ce n'est pas le cas, corrigez la chaîne. Enregistrez ensuite la chaîne dans une variable que vous nommerez 'CH09'. Placez la chaîne dans la pile et exécutez ASS. Stockez le code exécutable obtenu dans une variable que vous nommerez 'CD09'.

Pour exécuter le programme, il suffit de placer une chaîne de caractères non vide au niveau 1 de la pile et d'évaluer CD09.

Attention ! Ce programme ne vérifie pas que la pile comporte une chaîne non vide au niveau 1. N'exécutez pas CD09 si une chaîne de caractères non vide ne se trouve pas au niveau 1 de la pile.

Nous venons de passer en revue les structures élémentaires utilisées pour la programmation en assembleur :

- les tests,
- les sauts relatifs et absolus,
- les appels de sous-programmes par sauts absolus,
- les appels de sous-programmes par sauts relatifs.

4
Des fragments de jeux

Un jeu vidéo, ce n'est jamais plus qu'un affichage renouvelé, des détections de pressions sur le clavier et quelques sons. Nous allons voir comment réaliser tout cela.

4.1
Détecter une touche

Pour détecter une pression sur une touche, il faut :

- placer dans le registre OUT le masque OUT de la touche à tester,
- recueillir dans le registre IN les signaux en provenance du clavier,
- si ces signaux correspondent au masque IN de la touche, alors la touche a été pressée.

On utilise normalement les instructions `OUT=C` et `C=IN`. Cependant, `C=IN` pose parfois des problèmes liés à une implémentation en adresse impaire. On remplace donc `C=IN` par `GOSBVL 01160`. En effet, on trouve à l'adresse #01160h de la ROM une routine équivalente à `C=IN`.

Il existe aussi en ROM une routine remplaçant le couple d'instructions `OUT=C` et `C=IN`. Elle se trouve à l'adresse #01EECh de la ROM.

Les masques des touches (masque IN et masque OUT de chaque touche) sont présentés au chapitre 21.

Nous allons créer un programme chargé de détecter une pression sur la touche [VAR]. Le programme ne pourra prendre fin que lorsque cette touche sera pressée. Décrivons le programme :

- prologue, longueur totale hors prologue (1),
- sauvegarde des registres par appel d'une routine spécialisée en ROM et interdiction des interruptions (2),
- on place les 3 quartets de poids faibles du masque IN de [VAR] dans le champ X de A (3),

- au début de la boucle de test on place le masque OUT de [VAR] dans C(X) (4),
- copie de C(X) dans OUT (5),
- copie de IN dans C par appel de la routine en #1160h (6),
- « ET » logique entre les champs X des registres A et C pour ne maintenir dans C(X) que les bits armés communs à A(X) et C(X) (7),
- après le « ET » logique, tous les bits de C(X) sont-ils désarmés ? (8),
- si oui, retour vers le début de la boucle (9),
- récupération des registres, mise à 0 de P, autorisation des interruptions masquables, mode hexadécimal (10),
- on quitte le programme (11).

CCD 20

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	(1)
#00h	64000	*Longueur hp*	
#05h	8FB9760	GOSBVL 0679B	(2)
#0Ch	808F	INTOFF	
#10h	80822400	LAHEX 004	(3) Masque IN de [VAR] → A(X)
		Test_Var	Label (début de la boucle)
#18h	32080	LCHEX 080	(4) Masque OUT de [VAR] → C(X)
#1Dh	801	OUT=C	(5) Masque OUT de [VAR] → OUT
#20h	8F06110	GOSBVL 01160	(6) Equivalent à C=IN
#27h	0E32	C=C&A X	(7) « ET » logique
#2Bh	93A	?C=0 X	(8) C(X) est-il nul ?
#2Eh	AE	Goyes Test_Var	(9) Si oui, boucle !
			Fin de la partie utile du programme
#30h	8F2D760	GOSBVL 067D2	(10) Petit ménage habituel
#37h	20	P= 0	avant de quitter...
#39h	8080	INTON	
#3Dh	04	SETHEX	
#3Fh	8D20021	GOVLNG 12002	(11) On quitte le programme...

Notez que nous nous sommes contentés de charger les 3 quartets de poids faibles du masque IN car le quartet de poids fort de ce masque est toujours nul (sauf pour la touche [ON]. On limite ainsi les opérations aux champs X des registres A et C. Voir le chapitre 11 sur les registres.

Attention ! Placez les tests de touche dans une boucle pour que la machine vérifie régulièrement l'état du clavier. Utilisez des appels de sous-pro-

grammes pour traiter les différents événements qui doivent déclencher des pressions sur des touches du clavier.

Dans une boucle de test du clavier, il est généralement indispensable de guetter des pressions sur différentes touches. C'est en particulier le cas des programmes ludiques. L'astuce consiste à placer dans le registre OUT la somme logique (« OU » logique) des masques OUT des différentes touches à tester. On recueille ensuite dans le registre IN un masque IN identifiant la touche.

Par extension, il est possible d'attendre une pression sur une touche quelconque du clavier. Il suffit pour cela de placer la somme logique des différents masques OUT dans le registre OUT et d'attendre un masque IN quelconque dans le registre IN.

La touche [ON] peut être testée comme n'importe quelle autre mais il est indispensable d'interdire préalablement les interruptions (`INTOFF`).

Sans espace ni saut de ligne, saisissez la chaîne ci-dessous entre " et " :

```
"CCD20640008FB9760808F8082240032080801 8F061100E3293AA
E8F2D760208080048D20021"
```

Une fois la chaîne introduite, appuyez deux fois sur [ENTER] pour en obtenir deux exemplaires. Le second exemplaire sert d'argument à BYTES qui doit renvoyer #97B2h et 80. Si ce n'est pas le cas, corrigez la chaîne puis enregistrez-la dans une variable que vous nommerez 'CH10'. Placez la chaîne dans la pile, exécutez ASS et stockez le code exécutable obtenu dans une variable que vous nommerez 'CD10'.

4.2
Emettre un son

Le vibreur est activé en plaçant #800h dans OUT, il est désactivé en plaçant #000h dans ce même registre. Une activation suivie d'une désactivation provoque un « clic ». On peut produire un son en provoquant une suite de clics avec une fréquence déterminée.

Le programme que nous allons réaliser se fonde sur trois boucles imbriquées. La première utilise le registre D comme compteur, la seconde utilise B et la troisième utilise A. La seule nouveauté introduite par ce pro-

gramme est l'utilisation du registre OUT pour émettre un son. Tout le reste est déjà connu, nous ne détaillerons donc pas le fonctionnement.

CCD 20

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	En-tête classique...
#00h	45000	*Longueur hp*	
#05h	8FB9760	Gosbvl 0679B	
#0Ch	808F	INTOFF	
#10h	D2	C=0	Initialisation de C
#12h	303	LCHEX 3	3 dans C
#15h	22	P= 2	Initialisation à 2 de P
#17h	D7	D=C A	3 dans D(A)
		Boucle 1	Label de branchement Boucle 1
#19h	304	LCHEX 4	4 dans C
#1Ch	D5	B=C A	4 dans B(A)
		Boucle 2	Label de branchement Boucle 2
#1Eh	308	LCHEX 8	8 dans C
#21h	801	OUT=C	Activation du vibreur
#24h	D2	C=0 A	0 dans C
#26h	801	OUT=C	Désactivation du vibreur
#29h	D4	A=B A	Affectation d'une valeur à A(A)
		Boucle 3	Label de branchement Boucle 3
#2Bh	CC	A=A-1 A	Boucle d'attente pour la
#2Dh	5DF	GONC Boucle 3	fréquence du son
#30h	818F9F	B=B-#10h A	Boucle pour la durée du son
#36h	57E	GONC Boucle 2	
#39h	CF	D=D-1 A	Boucle répétition du son
#3Bh	5DD	GONC Boucle 1	
#3Eh	8F2D760	GOSBVL 067D2	
#45h	20	P= 0	Ici indispensable !
#47h	8080	INTON	
#4Bh	04	SETHEX	
#4Dh	8D20021	GOVLNG 12002	On quitte le programme

En exploitant l'inversion des champs, on peut se permettre de ne charger que #8h dans C au lieu de #800h. Cela n'est valable que si A(C) est nul.

Sans espace ni saut de ligne, introduisez entre " et " la chaîne ci-dessous :

```
"CCD20450008FB9760808FD230322D7304D5308801D2801D4CC5D
F818F9F57ECF5DD8F2D760208080048D20021"
```

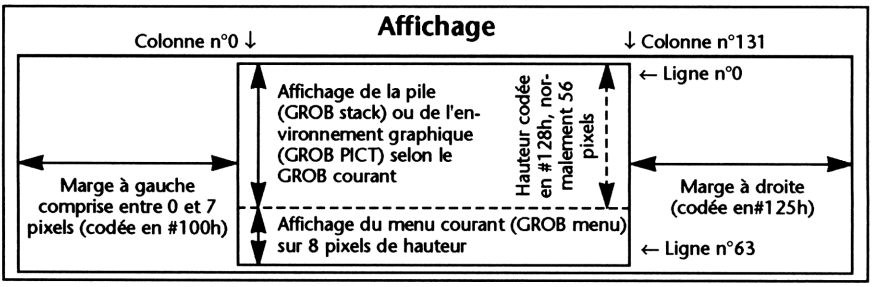
Une fois la chaîne introduite, appuyez deux fois sur [ENTER] pour obtenir deux exemplaires de la chaîne. Le second exemplaire sert d'argument à BYTES qui doit renvoyer #5FCh et 94. Si ce n'est pas le cas, corrigez la chaîne puis enregistrez-la dans une variable que vous nommerez 'CH11'. Placez la chaîne dans la pile puis exécutez ASS. Placez le code exécutable ainsi obtenu dans une variable que vous nommerez 'CD11'.

Pour exécuter le programme, il suffit d'évaluer CD11. Tendez l'oreille et écoutez votre HP-48 ronronner !

4.3
Inversion vidéo

L'écran doit être perçu comme une grille de points. Chacun d'eux peut être allumé ou éteint. L'état d'un point de l'écran est représenté par un bit qui peut être armé ou désarmé selon l'état du pixel (pixel est la contraction de *picture element*, un pixel est un point de l'écran). Un bit est donc associé à chaque pixel de l'écran. Un ensemble de bits décrit donc le contenu de l'écran ligne par ligne. 34 quartets sont nécessaires pour coder les états des pixels d'une ligne de l'écran.

En assembleur, modifier un pixel de l'écran revient à modifier l'état du bit qui lui est associé. Des zones de la mémoire vive sont consacrées à la sauvegarde des écrans. Ce sont les « bitmaps » écran. Il existe une bitmap écran pour la pile, une pour le menu courant, une pour l'environnement graphique, etc. Il existe dans la RAM système des champs où sont écrites les adresses des différentes bitmaps.



Pendant le processus d'affichage la HP-48 lit en #120h (RAM I/O) l'adresse à laquelle se trouve la bitmap à afficher. Vous pouvez écrire en #120h mais pas y lire. Les trois premiers bits en #100h déterminent la marge à gauche alors qu'en #125h est codée la marge à droite. Remarquez que l'écran physique est déterminé à partir d'un écran virtuel. L'adresse en #120h doit, de préférence, être paire. Si elle est impaire c'est l'adresse paire immédiatement inférieure qui sera considérée. Lorsque l'adresse de la « bitmap » est impaire, on donne la valeur 4 à la marge à gauche (en fait, on peut lui donner une valeur comprise entre 4 et 7). Inversement, la marge à gauche doit valoir 0 (en fait, entre 0 et 3) si l'adresse de la bitmap est paire. En #128h est codé le nombre de lignes du GROB principal à afficher à partir de la ligne n°0. Cette valeur est limitée à #3Fh (64 lignes de 0 à 63), notons que l'orsqu'elle vaut #3Fh, le GROB menu est masqué. Il est très vivement recommandé d'étudier adresse par adresse les différents champs de la mémoire vive des entrées-sorties (RAM I/O) et de la mémoire vive réservée au système (RAM système). Voir les chapitres 23 et 24 ainsi que le chapitre 22 pour la structure des objets graphiques.

Quelques adresses utiles où vous ne pouvez qu'écrire :

- en #120h est écrite l'adresse du premier bit du GROB à afficher,
- en #128h se trouve le numéro (entre 0 et 63) de la ligne séparant le GROB principal (pile ou PICT) et le GROB menu,
- en #130h est écrite l'adresse du premier bit du GROB du menu à afficher à partir de la ligne dont le numéro se trouve en #128h.

Quelques adresses en ROM et en RAM système :

- #8068Dh (G/GX) et #7050Eh (S/SX) contiennent l'adresse du premier bit de l'écran courant (GROB stack ou GROB PICT). On peut donc y une copie du contenu de #120h (la lecture est impossible en #120h,)
- on trouve en #13299h l'adresse de l'adresse du premier bit codant l'écran courant. En #13299h on trouve donc #8068Dh sur G/GX et #7050Eh sur S/SX. Cette adresse est très utile pour l'adaptation des programmes aux différentes versions de HP-48,
- #806D0h (G/GX) et #70551h (S/SX) contiennent l'adresse du prologue du GROB menu (8 pixels de hauteur),
- #806D5h (G/GX) et #70556h (S/SX) contiennent l'adresse du prologue du GROB stack (en général, 56 pixels de hauteur),
- #806E4h (G/GX) et #70565h (S/SX) contiennent l'adresse du prologue du GROB PICT (en général, 56 pixels de hauteur).

Nous allons développer un programme inversant l'état de tous les bits décrivant le GROB courant, on appelle cela une inversion vidéo. Ce programme et les suivants lisent en #131F8h car à cette adresse se trouve #8068Dh ou #7050Eh selon le type de la machine. On lit ensuite à l'une

de ces deux adresses pour récupérer l'adresse du codage de la bitmap. Il s'agit ici de l'adresse du premier bit décrivant le premier pixel et non pas adresse du prologue du GROB concerné. Nous allons utiliser deux boucles imbriquées. La première parcourt les 56 lignes de l'objet graphique alors que la seconde est chargée de parcourir les 34 quartets codant chaque ligne de l'écran.

Description de notre programme :

- récupération dans D0 de l'adresse du premier bit de la bitmap courante (1),
- on fait pointer D0 sur l'adresse du premier quartet décrivant cette bitmap (2),
- initialisation du compteur C(B) de la première boucle à 56 (#56d=#37h) (3),
- initialisation du compteur P de la seconde boucle à 16 (#16d=#Fh) (4),
- écriture dans A(B) de 8 bits lus à partir de D0 (5),
- inversion de ces bits par complément à 1 (6),
- écriture des 8 bits modifiés à partir de D0 (7),
- passage à l'octet suivant (8),
- décrémentation du compteur de la seconde boucle (9),
- décrémentation du compteur de la première boucle (10),

Nous traitons les quartets par lots de deux (champ B de A). Nous devons donc traiter 17 lots par ligne. Le compteur P ici utilisé nous limite à 16 itérations, cela explique que le 17e lot soit traité hors de la seconde boucle (on répète une fois hors de la boucle les instructions de celle-ci).

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	En-tête classique...
#00h	26000	*Longueur hp*	
#05h	8FB9760	Gosbvl 0679B	
#0Ch	808F	INTOFF	
#10h	1B8F131	D0=131F8	(1) En #131F8h, on récupère
#17h	146	C=DAT0 A	l'adresse de l'adresse du 1er bit
#1Ah	134	D0=C	décrivant le GROB courant
#1Dh	146	C=DAT0 A	(2) lecture de l'adresse de ce 1er
#20h	134	D0=C	bit (D0 pointe sur ce 1er bit)
#23h	3173	LCHEX 37	(3) #37h=#56d itérations
		Boucle 1	Label de début de la boucle 1
#27h	2F	P=F	(4) #Fh=#16d itérations
		Boucle 2	Label de début de la boucle 2
#29h	14A	A=DAT0 B	(5) Lecture d'un octet
#2Ch	BEC	A=-A-1 B	(6) Inversion d'un octet

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
#2Fh	148	DAT0=A B	(7) Ecriture de l'octet modifié
#32h	161	D0=D0+2	(8) Passage à l'octet suivant
#35h	0D	P=P-1	(9) Décrémentation boucle 2
#37h	51F	GONC BOUCLE 2	Fin de la boucle 2
#3Ah	14A	A=DAT0 B	Les 4 étapes de la boucle 2 sont
#3Dh	BEC	A=-A-1 B	reprises une 17e fois car P
#40h	148	DAT0=A B	est limité à 16 alors qu'il y a
#43h	161	D0=D0+2	17×2 quartets à traiter
#46h	A6E	C=C-1 B	(10) Décrémentation boucle 1
#49h	5DD	GONC BOUCLE 1	Fin de la boucle 1
#4Ch	8F2D760	GOSBVL 067D2	La fin est bien connue !
#53h	20	P= 0	
#55h	8080	INTON	
#59h	04	SETHEX	
#5Bh	8D20021	GOVLNG 12002	On quitte le programme

Comme d'habitude, introduisez sans espace ni saut de ligne et entre " et " cette chaîne de caractères :

```
"CCD20260008FB9760808F1B8F131146134146134317 32F14ABEC
1481610D51F14ABEC148161A6E5DD8F2D760208080048D20021"
```

Une fois la chaîne introduite, appuyez deux fois sur [ENTER]. Le second exemplaire de la chaîne sert d'argument à BYTES qui doit renvoyer #6302h et 108. Si ce n'est pas le cas, corrigez la chaîne. Enregistrez-la ensuite dans une variable que vous nommerez 'CH12'. Placez à nouveau la chaîne dans la pile, exécutez ASS puis enregistrez le code exécutable obtenu dans une variable que vous nommerez 'CD12'. Il suffit d'évaluer CD12 pour inverser l'état de tous les bits de l'écran. Comme vous le constaterez, l'inversion puis le retour à l'état normal ont lieu avec une vitesse que seul l'assembleur est en mesure d'offrir.

4.4

Allumer un point isolé

Nous allons maintenant créer un programme capable d'allumer un point de l'écran. Comme vous allez le constater, une opération déroutante de simplicité en environnement PICT ou dans un programme RPL est ici d'une savoureuse complexité !

Le principe de base est pourtant assez simple. La bitmap écran décrit pixel par pixel le contenu de l'écran. La bitmap écran est donc une suite linéaire de bits. Connaissant les coordonnées sur l'écran du pixel à allumer, il nous faut retrouver le bit correspondant dans la bitmap écran.

Nous avons déjà vu à l'occasion du programme précédent comment trouver l'adresse du premier quartet de la bitmap écran. Chaque ligne de l'écran est décrite par 34 quartets. Connaissant l'ordonnée du point à allumer, il suffit de multiplier cette ordonnée par 34 puis de l'ajouter à l'adresse du premier quartet codant l'écran pour obtenir l'adresse du premier quartet codant la ligne où se trouve le pixel à allumer. Une division entière de l'abscisse par 4 (4 pixels sont codés sur 1 quartet) fournit – par rapport au premier quartet codant la ligne concernée – l'adresse du quartet où se trouve le bit représentant le pixel à modifier.

L'adresse du quartet où se trouve le bit à modifier est donc la somme de l'adresse du premier quartet codant la ligne et du numéro du quartet dans cette même ligne de l'écran. Il faut ensuite déterminer celui des quatre bits qui devra être inversé, on utilise pour cela le reste de la division de l'abscisse par 4. Pour obtenir ce reste, on soustrait le quotient entier multiplié par 4 à l'abscisse. On charge ensuite un sous-programme d'allumer le pixel voulu en armant le bit associé à ce pixel.

Connaissant les coordonnées du pixel, on procède en trois étapes pour identifier le bit à modifier :

- recherche de la « ligne » à partir de l'ordonnée. Chaque ligne est codée par 34 quartets. On cherche donc l'adresse du premier quartet du groupe de 34 où se trouve le quartet à modifier,
- recherche du quartet à partir de l'abscisse,
- recherche du bit à modifier au sein du quartet.

Description du programme :

- chargement de l'abscisse (65_{10}) dans A(A) (1),
- sauvegarde de A(A) dans B(A) (2),
- chargement de l'ordonnée (15_{10}) dans C(A) (3),
- sauvegarde de C(A) dans D(A) (4),
- comme pour le programme CD12, on récupère l'adresse du premier quartet codant l'écran et on fait pointer D0 dessus (5),
- à partir de l'ordonnée on détermine – par rapport à l'adresse du premier quartet codant l'écran – l'adresse du premier quartet de la ligne concernée (6),
- on ajoute l'adresse du début de l'écran à celle que nous venons de calculer puis on fait pointer D0 dessus (7),
- deux décalages d'un bit vers la droite divisent par 4 l'abscisse et fournissent l'adresse – par rapport à l'adresse du premier

quartet codant la ligne concernée – du quartet où se trouve le bit à modifier (8),
• au sein de ce quartet, le numéro du bit à modifier s'obtient en soustrayant le numéro du quartet multiplié par 4 à l'abscisse du pixel (9),
• le bit à modifier étant identifié, un sous-programme est chargé d'écrire en mémoire la modification (10),
• une fois le pixel modifié, la machine est bloquée tant que la touche [VAR] n'est pas pressée (11). Voir le programme CD10.

CCD 20

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	Comme d'habitude...
#00h	3F000	*Longueur hp*	Longueur totale hors prologue
#05h	8FB9760	Gosbvl 0679B	Sauvegarde des registres
#0Ch	808F	INTOFF	Interdiction des interruptions
#10h	8082414000	LAHEX 00041	(1) Abscisse du pixel dans A(A)
#1Ah	D8	B=A A	(2) Sauvegarde de l'abscisse
#1Ch	34F0000	LC0000F	(3) Ordonnée du pixel dans C(A)
#23h	D7	D=C A	(4) Sauvegarde de l'ordonnée
#25h	1B99231	D0=13299	(5) Récupération de l'adresse du
#2Ch	142	A=DAT0 A	premier quartet de la bitmap
#2Fh	130	D0=A	courante
#32h	142	A=DAT0 A	
#35h	130	D0=A	
#38h	C7	D=D+D A	(6) Recherche verticale à partir de
#3Ah	DB	C=D A	l'ordonnée
#3Ch	23	P=3	
		BOUCLE 1	
#3Eh	C7	D=D+D A	
#40h	0D	P=P-1	
#42h	5BF	Gonc BOUCLE 1	
#45h	CB	C=C+D A	
#47h	132	AD0EX	(7) Recherche verticale
#4Ah	CA	A=A+C A	D0 pointe finalement sur le
#4Ch	130	D0=A	premier quartet de la ligne
#4Fh	D4	A=B A	(8) Recherche horizontale
#51h	819F0	ASRB A	2 décalages d'un bit vers la droite
#56h	819F0	ASRB A	provoquent une division par 4
#5Bh	136	CD0EX	
#5Eh	C2	C=C+A A	D0 pointe finalement sur le
#60h	134	D0=C	quartet où se trouve le bit
#63h	C4	A=A+A A	(9) Recherche du bit. On multi-
#65h	C4	A=A+A A	plie par 4 le quotient entier.
#67h	E8	B=B-A A	*Abscisse - 4×quotient entier = reste*

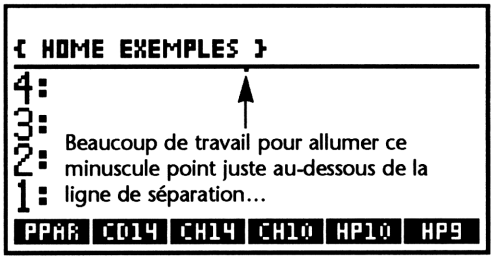
CCD 20

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
#69h	AE4	A=B B	Le reste est copié dans B
#6Ch	7A30	Gosub PIXEL	Appel du sous-programme
#70h	1540	DAT0=C P	Ecriture en mémoire du bit armé
#74h	80822400	LAHEX 004	Masque IN de [VAR] dans A(X)
		Test_VAR	Boucle de test de la touche [VAR]
#7Ch	32080	LCHEX 080	Masque OUT de [VAR] dans C(X)
#81h	801	OUT=C	On teste le clavier
#84h	8F06110	Gosbvl 01160	Equivalent à C=IN
#8Bh	0E32	C=C&A X	Est-ce le masque IN de [VAR] ?
#8Fh	93A	?C=0 X	
#92h	AE	Goyes Test_var	Non ? Alors boucle Test_VAR
#94h	8F2D760	Gosbvl 067D2	Récupération des registres
#9Bh	20	P= 0	Mise à zéro de P
#9Dh	8080	INTON	Autorisation des interruptions
#A1h	04	SETHEX	Mode hexadécimal
#A3h	8D20021	Govlng 12002	On quitte le programme
			Fin du programme principal
		PIXEL	*Début du sous-programme PIXEL*
#AAh	20	P= 0	Initialisation de P
#ACh	3110	LCHEX 01	
#B0h	962	?A=C B	Le reste vaut-il 1 ?
#B3h	F1	Goyes Pix_1	
#B5h	3120	LCHEX 02	
#B9h	962	?A=C B	Le reste vaut-il 2 ?
#BCh	21	Goyes Pix_2	
#BEh	3130	LCHEX 03	
#C2h	962	?A=C B	Le reste vaut-il 3 ?
#C5h	32	Goyes Pix_3	
#C7h	1560	C=DAT0 P	Cas d'un reste nul
#CBh	80890	CBIT=1 0	
#D0h	01	RTN	
		Pix_1	
#D2h	1560	C=DAT0 P	
#D6h	80891	CBIT=1 1	Armement du bit 1
#DBh	01	RTN	
		Pix_2	
#DDh	1560	C=DAT0 P	
#E1h	80892	CBIT=1 2	Armement du bit 2
#E6h	01	RTN	
		Pix_3	
#E8h	1560	C=DAT0 P	
#ECh	80893	CBIT=1 3	Armement du bit 3
#F1h	01	RTN	

Introduisez cette chaîne sans espace ni saut de ligne et entre " et" :

```
"CCD203F0008FB9760808F8082414000D834F0000D71B99231142
130142130C7DB23C70D5BFCB132CA130D4819F0819F0136C2134C
4C4E8AE47A3015408082240032080801 8F061100E3293AAE8F2D7
60208080048D20021203110962F131209622131309623215608 08
900115608089101156080892011560808 9301"
```

Une fois la chaîne introduite, appuyez deux fois sur [ENTER] pour obtenir dans la pile deux exemplaires de la chaîne. L'un sert d'argument à BYTES qui doit renvoyer #CA27h et 253. Corrigez la chaîne si ce n'est pas le cas. Enregistrez-la ensuite dans une variable que vous nommerez 'CH13'. Envoyez la chaîne dans la pile, exécutez ASS puis enregistrez le code exécutable ainsi obtenu dans une variable que vous nommerez 'CD13'. Il suffit d'évaluer CD13 pour exécuter le programme. Appuyez sur [VAR] pour quitter le programme.



4.5
Affichage d'un nombre

Le programme que nous allons maintenant développer fait référence à une table de codage des symboles propre au programme. La représentation graphique de chaque symbole y est codée sur 5 quartets. Ce programme s'inspire d'un article paru dans le numéro 11 du bulletin *Haute Performance* (voir la liste des clubs d'utilisateurs à la fin de l'ouvrage).

Nous allons ici réutiliser les procédures de détection d'une pression sur le clavier (CD10) et la méthode de chargement dans D0 de l'adresse du premier quartet codant la bitmap écran (CD11 et CD12). Décrivons notre programme :

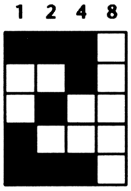
- chargement dans A(A) du nombre à afficher (1),
- recherche de l'adresse du premier pixel de l'écran (2a) (2b),

• la partie principale du programme se trouve dans le sous-programme AFF_CH (3a) (3b),
• le sous-programme AFF_CH est en fait une grande boucle (BOUCLE 1) répétée 5 fois car il y a 5 chiffres à afficher. B(B) est le compteur de boucle (4),
• une tentative de décrémentation provoque un retour de sous-programme si B(B) vaut déjà #00h (5),
• le registre pointeur de champ P est initialisé à #0h (6),
• C(A) est initialisé à #00000h (7),
• copie dans C(P) (le quartet n°0 de C) de A(P). On place donc le quartet de poids faible du nombre à afficher dans C(P) (8),
• C(A) est multiplié par #5h (9),
• `GOSUB DECOD` ne sert qu'à placer au sommet de la pile RSTK l'adresse du premier quartet codant le premier chiffre dans la table de symbole. Il s'agit d'une adresse physique et non pas d'une adresse relative par rapport au début du programme (10),
• les représentations graphiques des 16 chiffres hexadécimaux sont codées dans une table où 5 quartets codent un symbole. Chaque quartet code une ligne d'un symbole (11),
• on copie C(A) dans D(A). D(A) contient donc maintenant le chiffre à représenter multiplié par 5. Cela permet de retrouver l'adresse du premier quartet codant son symbole au sein de la table où chaque symbole est codé sur 5 quartets (12),
• `C=RSTK` permet de récupérer dans C(A) l'adresse du premier quartet codant le premier symbole dans la table (13),
• on ajoute l'adresse du premier quartet de la table des symboles au nombre de quartets dont il faut se décaler pour atteindre le premier quartet codant le symbole recherché (14),
• D1 pointe sur le premier quartet codant la représentation graphique du symbole (15),
• au sein de BOUCLE 2, le registre C tient deux rôles. Il sert à la fois de compteur de boucle (16a) (16b) (16c) (16d) chargé de produire 5 itérations, et de registre de travail chargé d'échanges entre D0 et D1. On fait appel à R0 pour conserver C en tant que compteur. C récupère le codage du symbole à l'adresse pointée par D1 (17) puis l'écrit sur la bitmap écran à l'adresse pointée par D0 (18),
• au sein de BOUCLE 2, D1 et D0 seront incrémentés à cinq reprises. D1 augmente par pas de 1 pour désigner le quartet suivant dans la table de codage (19) alors que D0 avance par pas de 34 quartets pour passer à la ligne suivante sur l'écran (une ligne de l'écran est codée sur 34 quartets) (20),
• on récupère dans C la valeur de D0 qui désigne un quartet de la bitmap écran (21a). On retire 5×34 quartets pour remonter de 5 lignes puis on retire un quartet pour ce décaler de 4 pixels

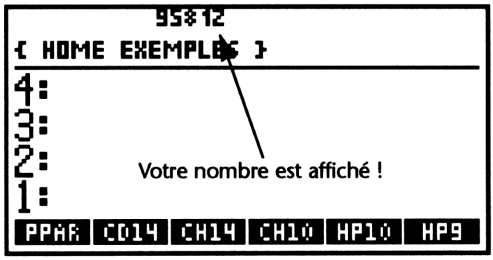
sur la gauche du dernier chiffre affiché (21b) (21c) (21d) (21e). On revient donc finalement 171 quartets en arrière dans la bitmap écran pour que D0 puisse pointer sur le quartet codant la première ligne de pixels du prochain caractère (22),

• A contient le nombre à représenter. Pour obtenir le prochain chiffre à représenter dans le quartet n°0 de A, il suffit de décaler A d'un quartet vers la droite (23).

Codage des symboles



Les symboles sont codés sur 5 quartets. A chaque quartet correspond une ligne du symbole. Ici, « 2 » sera codé #74217h. La colonne 8 reste vide pour séparer chaque symbole.



La recherche de l'adresse du premier pixel de la bitmap écran est confiée au sous-programme AD_EC. Il lit en #13299h l'adresse où est écrite l'adresse du premier pixel de l'écran affiché. La donnée, lue en #13299h, varie selon le type de machine, cette lecture en #13299h rend donc le programme utilisable sur HP-48 S/SX et HP-48 G/GX. Deux lectures successives sont nécessaires puisqu'on passe d'une première adresse où l'on lit une seconde adresse aux données pointées à cette seconde adresse. A la fin de AD_EC, `D0=D0+ #Eh` permet de se placer sur le pixel n°#Ehx#4h.

`Gosub Decod` est ici utilisé dans le seul but de placer au sommet de la pile RSTK l'adresse du premier quartet codant le premier symbole. L'instruction `C=RSTK` récupère cette adresse et la retire de la pile RSTK.

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
	CCD20	*Prologue*	Prologue
#000h	62100	*Longueur hp*	Longueur totale hors prologue
#005h	8FB9760	Gosbvl 0679B	Sauvegarde des registres
#00Ch	808F	INTOFF	Interdiction des interruptions
#010h	8082421859	LAHEX 95812	(1) Nombre à afficher dans A(A)

CCD 20

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
#01Ah	7A30	Gosub AD_EC	(2a)
#01Eh	7E40	Gosub AFF_CH	(3)
#022h	80822400	LAHEX 004	*Voir CD10*
		Test_Var	
#02Ah	32080	LCHEX 080	
#02Fh	801	OUT=C	
#032h	8F06110	Gosbvl 01160	
#039h	0E32	C=C&A X	
#03Dh	93A	?C=0 X	
#040h	AE	Goyes Test_Var	
#042h	8F2D760	Gosbvl 067D2	Récupération des registres
#049h	20	P= 0	
#04Bh	8080	Inton	Autorisation des interruptions
#04Fh	04	SETHEX	Mode hexadécimal
#051h	8D20021	Govlng 12002	On quitte le programme
		AD_EC	(2b) Sous-programme AD_EC
#058h	1B99231	D0=13299	Adresse commune aux S et G
#05Fh	146	C=DAT0 A	
#062h	134	D0=C	
#065h	146	C=DAT0 A	
#068h	134	D0=C	D0→ 1er quartet de la bitmap
#06Bh	16D	D0=D0+ #Eh	On s'écarte de 15×4 pixels
#06Eh	01	RTN	Retour de AD_EC
		AFF_CH	Sous-programme AFF_CH
#070h	3150	LCHEX 05	(4) 5 itérations
#074h	AE5	B=C B	
		BOUCLE 1	Label BOUCLE 1
#077h	A6D	B=B-1 B	(5)
#07Ah	400	RTNC	Retour de sous-programme
#07Dh	20	P= 0	(6)
#07Fh	D2	C=0 A	(7)
#081h	A86	C=A P	(8)
#084h	D7	D=C A	(9) Multiplication par 5 de C(A)
#086h	C6	C=C+C A	
#088h	C6	C=C+C A	
#08Ah	CB	C=C+D A	
#08Ch	7050	Gosub DECOD	(10)
#090h	75557		(11) Codage de « 0 »
#095h	46444		Codage de « 1 »
#09Ah	74217		Codage de « 2 »
#09Fh	74247		Codage de « 3 »

CCD 20

Adresses	Codes hexadécimaux	Mnémoniques et labels	Commentaires
#0A4h	11172		Codage de « 4 »
#0A9h	71647		Codage de « 5 »
#0AEh	71757		Codage de « 6 »
#0B3h	74211		Codage de « 7 »
#0B8h	25252		Codage de « 8 »
#0BDh	75747		Codage de « 9 »
#0C2h	25755		Codage de « A »
#0C7h	35357		Codage de « B »
#0CCh	61116		Codage de « C »
#0D1h	35553		Codage de « D »
#0D6h	71317		Codage de « E »
#0DBh	71311		Codage de « F »
		DECOD	Label DECOD
#0E0h	D7	D=C A	(12)
#0E2h	07	C=RSTK	(13) Récupération d'adresse
#0E4h	CB	C=C+D A	(14)
#0E6h	135	D1=C	(15)
#0E9h	3140	LCHEX 04	(16a)
		BOUCLE 2	Label BOUCLE 2
#0EDh	108	R0=C B	(16b)
#0F0h	1570	C=DAT1 P	(17)
#0F4h	1540	DAT0=C P	(18)
#0F8h	170	D1=D1+ 1	(19)
#0FBh	16F	D0=D0+ 16	(20)
#0FEh	16F	D0=D0+ 16	
#101h	161	D0=D0+ 2	
#104h	118	C=R0 B	(16c)
#107h	A6E	C=C-1 B	(16d)
#10Ah	52E	Gonc BOUCLE 2	
#10Dh	136	CD0EX	(21a) Récupération de D0 dans C
#110h	D7	D=C A	(21b)
#112h	34BA000	LC000AB	(21c) 171 dans C(A)
#119h	E3	D=D-C A	(21d)
#11Bh	DB	C=D A	(21e)
#11Dh	134	D0=C	(22)
#120h	F4	ASR A	(23) Passage au chiffre suivant
#122h	645F	Goto BOUCLE 1	Fin de l'affichage d'un chiffre

Saisissez la chaîne ci-dessous sans espace ni saut de ligne et entre " et ".

```
"CCD20621008FB9760808F80824218597A307E408082240032080
8018F061100E3293AAE8F2D760208080048D200211B9923114613
414613416D013150AE5A6D40020D2A86D7C6C6CB7050755574644
```

```
474217742471117271647717577421125252757472575535357 61
1163555371317713 11D707CB13531401081570154017016F16F16
1118A6E52E136D734BA000E3DB134F4645F"
```

Appliquez BYTES à une copie de cette chaîne. Vous devez obtenir #948Dh et 304. Enregistrez la chaîne dans une variable que vous nommerez 'CH14'. Donnez la chaîne en argument à ASS, le code exécutable ainsi obtenu doit être sauvegardé dans une variable que vous appellerez 'CD14'. Evaluez CD14 pour exécuter le programme. Pour arrêter le programme, appuyez sur [VAR].

5
Quelques compléments

Les instructions qui arment le bit de retenue lorsqu'une certaine condition est remplie, le désarment quand ladite condition n'est pas remplie.

Notez que ASS vous permet de créer vos propres objets (même ceux interdits à l'utilisateur !). Il suffit de placer dans une chaîne de caractères l'objet à créer (voir le chapitre 22), puis d'exécuter ASS.

La conversion des mnémoniques en codes hexadécimaux est fastidieuse mais peut être confiée à un programme assembleur. Il en existe de nombreux dans le domaine public (AsmFlash, HP Saturn Assembler, etc.). Prenez garde, chaque assembleur a sa propre syntaxe.

Vous allez maintenant entreprendre la lecture de la suite de cet ouvrage. Il vous est conseillé d'étudier avec attention tous les chapitres de la première partie, puis parcourez les chapitres 21 à 24 de la deuxième partie. Vous serez ensuite capable d'analyser les jeux de la troisième partie et pourrez entreprendre vos premières créations ludiques en assembleur.

2
Le binaire naturel

Les ordinateurs affichent tous les caractères de la création sur leurs écrans. Cependant, il faut savoir que la machine ne manipule que des « 0 » et des « 1 ». Ces deux chiffres représentent les deux seuls états reconnus en électronique numérique : un niveau « haut » (le courant passe) et un niveau « bas » (le courant ne passe pas). Nous allons maintenant décrire le code binaire de base qu'est le "binaire naturel".

1
La base 2

1.1
Introduction

Les nombres que nous manipulons au quotidien sont exprimés en base 10. Il y a à cela une raison bien simple : nous avons dix doigts. Il existe de nombreuses autres bases, par exemple, la base 60 (calculs sexagésimaux) utilisée pour les minutes et les secondes.

L'ENIAC, construit à la fin de la seconde guerre mondiale, est souvent considéré comme la première machine informatique électronique.

L'ENIAC, en utilisant le système de numération décimal n'a fait que démontrer que ce système de numération n'est pas utilisable « à l'intérieur » d'une machine électronique. En effet, l'emploi de la base dix impose la reconnaissance de dix états distincts qui correspondent aux dix chiffres (de « 0 » à « 9 »). Or, des phénomènes physiques et électriques que nous ne décrirons pas dans ce livre rendent très peu fiable la reconnaissance de dix états distincts. Tous les ordinateurs, et par extension, toutes les calculatrices, se contentent de reconnaître la présence de courant (c'est un état « haut », interprété comme un « 1 »), ou l'absence de courant (c'est un état « bas » interprété comme un « 0 »). Les deux chiffres « 0 » et « 1 » sont ceux utilisés en base 2 (calculs binaires).

Une calculatrice va donc recevoir vos instructions et vos données exprimées à l'aide de caractères divers, ceux-ci seront convertis en suites de nombres binaires et traités sous cette forme. Le résultat du traitement subira une conversion inverse afin de rendre le résultat lisible (la conversion est dite « transparente » pour l'utilisateur final).

En décimal, c'est-à-dire en base 10, nous utilisons quotidiennement les chiffres de « 0 » à « 9 ». Dix symboles sont donc utilisés. Remarquez que la base 10 implique l'emploi de dix symboles (il existe dix chiffres). Par ailleurs, la base, ici 10, est représentée par « 1 » suivi de « 0 ». Cette remarque peut sembler ridicule au premier abord mais vous ne tarderez pas à comprendre son intérêt !

En binaire, c'est-à-dire en base 2, seuls deux chiffres - 0 et 1 - seront employés. Tous les nombres binaires seront donc des suites de « 0 » et de « 1 ». Comment compter en base 2 ? Le zéro décimal habituel demeure représenté par « 0 », tout aussi simplement, le 1 décimal sera représenté par « 1 ». Comment représenter le 2 décimal en binaire ? Ne disposant que de deux chiffres, 2 sera représenté par 10 en base 2 que l'on notera 10_2. Nous l'avions annoncé, « 10 » représente toujours la base.

En cas d'équivoque, la base dans laquelle est exprimé le nombre est placée en indice. Par exemple 11_2 n'est autre que 3_{10}...

Peut-être vous demandez-vous pourquoi la base 1 n'est jamais utilisée. Cette base n'a aucun intérêt puisque chaque nombre serait constitué d'un nombre de chiffres égal à la valeur qu'il représente. Par exemple, pour représenter 1000_{10} en base 1, il faut mille chiffres. En base 10 quatre chiffres suffisent pour représenter 1000_{10} ; une dizaine de chiffres est nécessaire en base 2.

C'est donc la numération binaire qui apporte le compromis recherché entre le nombre d'états que doit distinguer la machine et les contraintes

d'utilisation. La numération binaire a permis le développement d'une algèbre très particulière, l'algèbre de Boole dont nous parlerons prochainement... Mais revenons à la représentation des nombres en base 2. Comme nous l'avons dit 0_{10} est représenté par 0_2, 1_{10} est représenté par 1_2 et 2_{10} est représenté par 10_2. Pour représenter 3_{10} en binaire, il nous suffit d'ajouter 1_2 (soit, 1_{10}) à 10_2 (soit, 2_{10}), ce qui donne 11_2. Pour représenter 4_{10}, on ajoutera 1_2 (soit, 1_{10}) à 11_2 (soit, 3_{10}). 1_2+11_2 donne 100_2. 4_{10} est donc représenté par 100_2 en binaire.

En base 10, le carré de la base, c'est-à-dire 10^2, est représenté par 100 (ici, 100_{10}). En base 2, le carré de la base, c'est-à-dire $2_{10}{}^2=4_{10}$ est aussi représenté par 100 (ici, 100_2). 100 représente donc toujours le carré de la base. De même, 1000 représente toujours le cube de base, etc.

Le tableau ci-dessous résume les équivalences entre les notations en décimal et les notations en binaire pour les nombres compris entre 0_{10} et 16_{10}.

Décimal	Binaire	Remarques
0	0	*Notations identiques*
1	1	*Notations identiques*
2	10	*10 représente la base*
3	11	
4	100	*Base 2 puissance 2*
5	101	
6	110	
7	111	
8	1000	*Base 2 puissance 3*
9	1001	
10	1010	*10 représente la base*
11	1011	
12	1100	
13	1101	
14	1110	
15	1111	
16	10000	*Base 2 puissance 4*

Remarquez qu'avec un chiffre binaire, vous pouvez représenter deux ($2=2_{10}{}^1$) nombres qui sont 0_2 et 1_2. Avec deux chiffres binaires, vous représenterez quatre ($4=2_{10}{}^2$) nombres : $00_2=0_2$, $01_2=1_2$, 10_2 et 11_2. Avec trois chiffres binaires vous représenterez huit ($8=2_{10}{}^3$) nombres binaires : $000_2=0_2$, $001_2=1_2$, $010_2=10_2$, $011_2=11_2$, 100_2, 101_2, 110_2 et 111_2. Avec

quatre chiffres binaires, vous représenterez seize ($16=2_{10}^4$) nombres binaires : 0000_2, 0001_2, 0010_2, 0011_2, 0100_2, 0101_2, 0110_2, 0111_2, 1000_2, 1001_2, 1010_2, 1011_2, 1100_2, 1101_2, 1110_2 et 1111_2. Tout ceci pour vous faire comprendre qu'avec *n* chiffres binaires il est possible de représenter 2^n nombres binaires.

Ainsi, avec huit chiffres binaires, on représente 256 (2^8) valeurs différentes. Avec seize chiffres binaires, ce sont 65536 (2^{16}) nombres différents que nous représenterons.

Notez qu'il est parfaitement inutile d'apprendre par cœur une table de correspondance entre les valeurs décimales et leurs équivalents binaires. Il existe des méthodes de calcul simples permettant de passer d'une base à l'autre.

1.2
Grouper les bits

Commençons par préciser que les mots « numérique » et « digital » doivent être considérés comme synonymes. Quel que soit le mode de numération, un chiffre quelconque est une unité numérique, c'est donc aussi une unité digitale. Traduit en anglais ce « digital unit » se contracte pour donner naissance au mot « digit ». Un digit est donc un chiffre, c'est-à-dire une valeur numérique représentée à l'aide d'un unique symbole.

Un nombre binaire est donc un digit binaire. Traduit en anglais, le digit binaire devient un « binary digit » qui a été contracté en « bit ». Un bit est donc un chiffre binaire. Un bit est un 0 ou un 1.

En binaire il est nécessaire d'employer plusieurs bits pour représenter une valeur supérieure ou égale à 2_{10},. Un groupe de digits, et donc, un groupe de bits, constitue un mot. On parle de mot binaire lorsque le mot est constitué de bits. Multiplet est un synonyme de mot quelle que soit la base de numération. Par contre il existe deux termes qui ne s'emploient qu'en base 2 :

- un quartet est un mot de quatre bits,
- un octets est un mot de huit bits.

Un quartet peut prendre $2^4_{10} = 16$ valeurs différentes. On l'appelle parfois tétrade ou nibble. Un octet peut prendre $2^8_{10} = 256$ valeurs différentes.

En anglais, on emploie « byte » à la place d'octet. Méfiez-vous des confusions entre bit et byte !

On appelle souvent mot, un mot constitué de deux octets juxtaposés (16 bits). Un double mot désigne souvent un groupe de quatre octets juxtaposés (32 bits). A défaut de texte officiel, ces deux dernières définitions ne reflète qu'une habitude de langage. Notez qu'il est fréquent que la définition du mot varie en fonction des constructeurs !

1.3
Forme canonique d'un nombre

En analysant le mot décimal 7872, on peut décomposer ce dernier en une somme : le chiffre des milliers multiplié par 10^3, auquel on ajoute le chiffre des centaines multiplié par 10^2, auxquels on ajoute le chiffre des dizaines multiplié par 10^1, auxquels on ajoute le chiffre des unités multiplié par 10^0. On a donc :

$$7872 = 7\times10^3 + 8\times10^2 + 7\times10^1 + 2\times10^0$$

Plus généralement :

$$abcd_{10} = a\times10^3 + b\times10^2 + c\times10^1 + d\times10^0$$

En se fondant sur l'exposant de la base, ici dix, on peut attribuer un rang à chaque digit du mot :

- *d* occupe le rang 0,
- *c* occupe le rang 1,
- *b* occupe le rang 2,
- *a* occupe le rang 3.

Généralisons en appelant *n* le plus haut rang et en notant chaque coefficient α :

$$\alpha_n10^n + \alpha_{n-1}10^{n-1} + \alpha_{n-2}10^{n-2} + \alpha_{n-3}10^{n-3} + \ldots + \alpha_0 10^0$$

N'étant plus limité par les quatre digits de notre exemple de départ, donnons la forme générale d'un nombre entier en base 10 :

$$N_{10} = \alpha_n10^n + \alpha_{n-1}10^{n-1} + \alpha_{n-2}10^{n-2} + \ldots + \alpha_0 10^0$$

Généralisons encore en nous affranchissant de la base 10. Dans l'expression qui suit, le nombre N est exprimé dans la base B :

$$N_B = \alpha_n B^n + \alpha_{n-1} B^{n-1} + \alpha_{n-2} B^{n-2} + \ldots + \alpha_0 B^0$$

Cette formulation correspond à la forme canonique d'un nombre, et ce, quelle que soit sa base.

Bien entendu la forme canonique peut être adaptée à la base 2 :

$$N_2 = \alpha_n 2^n + \alpha_{n-1} 2^{n-1} + \alpha_{n-2} 2^{n-2} + \ldots + \alpha_0 2^0$$

α représente un digit et dépend donc de la base de numération. En binaire, α ne peut donc prendre que deux valeurs : 0 ou 1.

La forme canonique des nombres est le point de départ des méthodes de conversion entre bases de numération.

2
Conversions

2.1
Du binaire vers le décimal

Considérons un octet, c'est-à-dire un groupe de 8 bits ou de deux quartets juxtaposés. Chaque bit peut se voir attribuer un rang. Le bit de droite a pour rang 0 alors que le bit de gauche a 7 pour rang. Soit l'octet 10110001, voyons les rangs de chacun des huit bits :

Rangs	7	6	5	4	3	2	1	0
Bits	1	0	1	1	0	0	0	1

A chaque bit d'un mot binaire correspond donc un rang. En binaire la base est 2_{10}, en utilisant le rang d'un bit comme exposant de la base, on obtient le poids de chaque bit d'un mot.

Rangs	7	6	5	4	3	2	1	0
Poids	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Bits	1	0	1	1	0	0	0	1

Le poids de chaque bit est ici exprimé en base 10. L'octet 10110001_2 peut donc très facilement être converti en son équivalent décimal en se référant au tableau ci-dessus. On pose :

$$10110001_2 = 1\times2^7 + 0\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 0\times2^2 + 0\times2^1 + 0\times2^0$$

En calculant les puissances de 2_{10}, on obtient :

$$10110001_2 = 1\times128_{10}+0\times64_{10}+1\times32_{10}+1\times16_{10}+0\times8_{10}+0\times4_{10}+0\times2_{10}+0\times1_{10}$$

D'où :

$$10110001_2 = 128_{10}+32_{10}+16_{10}+1_{10}$$

Finalement :

$$10110001_2 = 177_{10}$$

Et voilà votre incompréhensible octet traduit en décimal ! Un octet, constitué de huit bits, peut prendre 2^8=256 valeurs distinctes. Vérifiez vous-même que $0000\ 0000_2 = 0_{10}$ et que $1111\ 1111_2 = 255_{10}$. Ce sont donc bien 256 valeurs que permet de représenter un octet. Remarquez qu'on groupe les bits par quatre pour faciliter la lecture. Ces groupes de quatre bits sont des quartets qui, comme nous le verrons bientôt, correspondent chacun à un nombre hexadécimal (en base 16).

C'est donc en multipliant chaque bit d'un mot binaire par son poids qu'on obtient l'équivalent décimal d'un nombre binaire. Ainsi quelle que soit la longueur d'un mot binaire, on écrit au-dessus de chaque bit le poids de celui-ci en commençant par 2^0=1 à gauche jusqu'à 2^n à droite si le mot est constitué de *n* bits.

La progression des poids de la gauche vers la droite correspond aux termes d'une suite géométrique de raison 2. Le tableau ci-dessous donne quelques correspondances entre rang et poids.

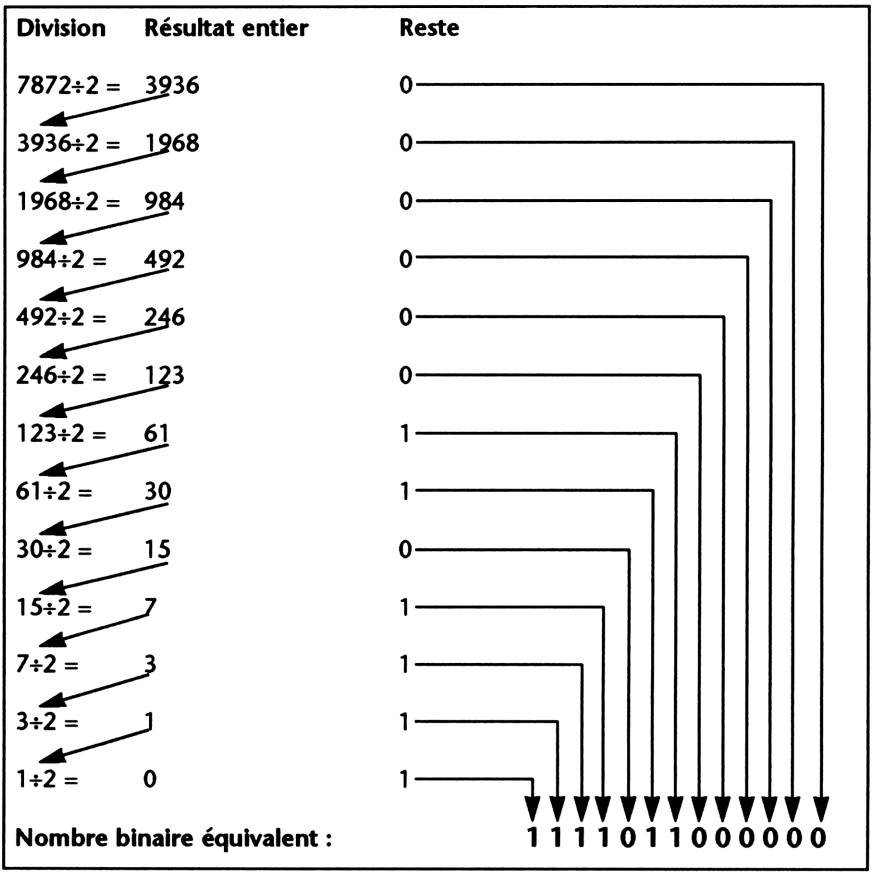
Rang	Poids	Rang	Poids	Rang	Poids	Rang	Poids
0	1	3	8	6	64	9	512
1	2	4	16	7	128	10	1024
2	4	5	32	8	256	11	2048

2.2
Du décimal vers le binaire

CCD 20

Pour convertir un nombre décimal en son équivalent binaire, on emploie généralement la méthode dite "des divisions successives". Elle consiste en succession de divisions par 2_{10} dont on retient les restes. Compte tenu du diviseur, ici 2_{10}, le reste ne peut valoir que 0_{10} ou 1_{10}.

Convertissons 7872_{10} en binaire :



On effectue donc des divisions successives par 2 jusqu'à obtenir un résultat nul. On aligne les restes, toujours égaux à 0 ou à 1, de ces divisions sur une colonne. On transforme cette colonne en ligne (voir tableau ci-dessus) pour obtenir le nombre binaire.

Il n'est pas difficile de vérifier l'exactitude du résultat. Il suffit pour cela d'effectuer la conversion inverse :

$$
\begin{aligned}
1111011000000_2 &= 1 \times 2^{12} \quad (= 1 \times 4096_{10}) \\
&+ 1 \times 2^{11} \quad (= 1 \times 2048_{10}) \\
&+ 1 \times 2^{10} \quad (= 1 \times 1024_{10}) \\
&+ 1 \times 2^{9} \quad (= 1 \times 512_{10}) \\
&+ 0 \times 2^{8} \quad (= 0 \times 256_{10}) \\
&+ 1 \times 2^{7} \quad (= 1 \times 128_{10}) \\
&+ 1 \times 2^{6} \quad (= 1 \times 64_{10}) \\
&+ 0 \times 2^{5} \quad (= 0 \times 32_{10}) \\
&+ 0 \times 2^{4} \quad (= 0 \times 16_{10}) \\
&+ 0 \times 2^{3} \quad (= 0 \times 8_{10}) \\
&+ 0 \times 2^{2} \quad (= 0 \times 4_{10}) \\
&+ 0 \times 2^{1} \quad (= 0 \times 2_{10}) \\
&+ 0 \times 2^{0} \quad (= 0 \times 1_{10})
\end{aligned}
$$

D'où :

$$1111011000000_2 = 4096_{10} + 2048_{10} + 1024_{10} + 512_{10} + 128_{10} + 64_{10}$$

On a bien :

$$1111011000000_2 = 7872_{10}$$

Il existe d'autres méthodes de conversion, par exemple, la méthode dite des « soustractions successives ». N'oubliez pas que votre HP-48 est capable d'effectuer toutes les conversions dont vous avez besoin ! Cependant il est indispensable de maîtriser une méthode de calcul avant de substituer la HP-48 à notre propre réflexion.

3

Poids faible et poids fort

Nous avons vu qu'au sein d'un mot binaire chaque bit possède un rang et qu'à chaque rang est associé un poids. Le bit le plus à droite a 0 pour rang alors que le bit le plus à gauche à *n-1* pour rang (dans le cas d'un mot binaire de *n* bits). On en déduit que les poids associés à chaque bit varient

entre 2^0 et 2^{n-1}. Le bit le plus à droite est le « bit de plus faible poids » alors que le bit de gauche est le « bit de plus fort poids ». On utilise les sigles américains LSB et MSB :

- LSB signifie Least Significant Bit ou bit le moins significatif (bit de plus faible poids),
- MSB signifie Most Significant Bit ou bit le plus significatif (bit de plus fort poids).

La place du LSB et du MSB dans un octet :

	MSB							**LSB**
Rang	7	6	5	4	3	2	1	0

Bien entendu, il existe aussi un MSB et un LSB au sein d'un quartet :

	MSB			**LSB**
Rang	3	2	1	0

De même l'un des deux quartets qui constitue un octet est de poids fort alors que l'autre est de poids faible.

	Quartet de poids fort				**Quartet de poids faible**			
Rang	7	6	5	4	3	2	1	0

4
Nombres rationnels

Les nombres rationnels, constitués d'une partie entière et d'une partie décimale séparées par une virgule, pourront être représentés en généralisant une fois de plus la forme canonique d'un nombre. Utilisant la base de numération B, la partie décimale d'un nombre rationnel sera représentée par :

$$N_B : \alpha_{-1}B^{-1} + \alpha_{-2}B^{-2} + \alpha_{-3}B^{-3} + \ldots + \alpha_{-n}B^{-n}$$

En spécialisant cette formule à la base 2, on obtient :

$$N_2 : \alpha_{-1}2^{-1} + \alpha_{-2}2^{-2} + \alpha_{-3}2^{-3} + \ldots + \alpha_{-n}2^{-n}$$

Retenez que $B^{-n} = 1/B^n$, d'où $2^{-1} = 1/2^1 = 1/2$ et $2^{-2} = 1/2^2 = 1/4$, etc.

Le tableau ci-dessous résume les premières puissances négatives de 2 :

n	2^n	2^{-n}	10^n	10^{-n}
0	1	**1**	1	1
1	2	**0,5**	10	0,1
2	4	**0,25**	100	0,01
3	8	**0,125**	1000	0,001
4	16	**0,0625**	10000	0,0001
5	32	**0,03125**	100000	0,00001
6	64	**0,015625**	1000000	0,000001
7	128	**0,0078125**	10000000	0,0000001
8	256	**0,003906...**	100000000	0,00000001
9	512	**0,001953...**	1000000000	0,000000001
10	1024	**0,000976...**	10000000000	0,0000000001

On calcule le poids des bits à droite de la virgule exactement comme le poids des bits de la partie entière :

Rang	0	-1	-2	-3	-4	-5
Poids	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
	=	=	=	=	=	=
	1	$1/2^1$	$1/2^2$	$1/2^3$	$1/2^4$	$1/2^5$
		=	=	=	=	=
		1/2	1/4	1/8	1/16	1/32
		=	=	=	=	=
	1	0,5	0,25	0,125	0,0625	0,03125

A titre d'exemple, convertissons $0,11010_2$ en décimal :

Nombre	0,	1	1	0	1	0
Rang	0	-1	-2	-3	-4	-5
Poids	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
Valeur	0×1	1×0,5	1×0,25	0×0,125	1×0,0625	0×0,03125

On pose :

$$0{,}11010_2 = 0{,}5_{10} + 0{,}25_{10} + 0{,}0625_{10}$$

D'où $0{,}11010_2 = 0{,}8125_{10}$.

La conversion inverse, de décimal à binaire s'effectue à l'aide de méthodes très proches de celles déjà utilisées pour les entiers. La méthode dite "des multiplications successives par 2" a l'avantage de la rapidité.

3

Calculs en base 2

Nous allons maintenant décrire les règles qui permettent l'addition, la soustraction, la multiplication et la division avec des nombres binaires, qui, rappelons-le sont ceux qui seront effectivement manipulés par la machine informatique. Ce chapitre vous permettra de comprendre comment un microprocesseur procède pour effectuer nos calculs. L'arithmétique octale et hexadécimale se déduit des règles présentées ici avec des calculs en base 2.

1 L'addition binaire

L'addition binaire peut se résumer en quatre cas de base :

0 + 0 = 0	→	Pas de retenue
0 + 1 = 1	→	Pas de retenue
1 + 0 = 0	→	Pas de retenue
1 + 1 = 0	→	retenue

Par exemple :

```
      10101 , 01
+      1001 , 1
=    11110 , 11
```

Soit la somme de deux octets. En partant des bits de droite (bit de poids le plus faible), on additionne les bits d'une même colonne en reportant une éventuelle retenue sur la colonne placée immédiatement à gauche.

L'addition fractionnée permet, en groupant les bits par paires et en effectuant des sommes de paires en parallèle, de diminuer notablement le temps de calcul. Même si la machine manipule des mots d'un octet (ou plus), il lui faudra effectuer autant d'additions qu'il y a de colonnes de bits, et ce, afin de tenir compte d'une éventuelle retenue. Pour ajouter deux octets, huit additions consécutives sont nécessaires. Avec la méthode dite de « l'addition fractionnée », vous pouvez réduire le nombre d'additions consécutives. Il suffit pour cela de diviser chaque octet en quatre groupes de deux bits. Les quatre groupes de deux fois deux bits seront additionnés en parallèle à l'aide de quatre circuits indépendants. Les retenues « sortantes » qui passent d'un groupe de bits à un autre groupe de bits sont mémorisées puis transmises au groupe de bits qui doit les accueillir. Les retenues sont ajoutées aux résultats partiels déjà obtenus ce qui va peut-être entraîner l'apparition de nouvelles retenues qui nécessiteront un nouveau cycle d'addition.

2
Soustraction et nombres négatifs

2.1
Utilisation d'un signe

Pour représenter un nombre binaire négatif nous pouvons nous contenter de le faire précéder du signe « moins » habituel (« - »). Les notations sont alors les suivantes :

- 4_{10} est noté +0100 (codage sur un quartet) qui peut se simplifier en 0100,
- -4_{10} est noté -0100.

Cette notation consistant à ajouter un signe devant la valeur absolue est un codage en « valeur absolue plus le signe ». Nous avons déjà dit que la

machine ne reconnaissait que des « 0 » et des « 1 ». Les habituels signes « + » et « - » ne sont donc pas reconnus par la machine. Il va donc falloir coder les « + » et « - » à l'aide de bits.

2.2
Le binaire signé

En utilisant la notation avec « la valeur absolue plus le signe », la plus grande valeur que l'on puisse représenter avec un octet est (+)1111 1111 qui vaut 255_{10}. Toujours avec cette notation, la plus petite valeur que l'on puisse représenter avec un octet est (-)1111 111, c'est-à-dire -255_{10}. Or, les signes « + » et « - » ne sont pas compréhensibles par la machine d'où l'obligation de les remplacer par des bits.

On décide donc que la valeur du bit de poids le plus fort (MSB) déterminera le signe du mot binaire. On utilise généralement les conventions suivantes :

- quand le MSB vaut 0, le nombre est positif,
- quand le MSB vaut 1, le nombre est négatif.

L'un des bits de l'octet étant réservé au signe, il ne reste plus que sept bits sur huit pour exprimer la valeur absolue du nombre binaire. Une telle notation est appelée binaire signé. Ici, le MSB est le bit de signe.

La plus grande valeur que peut prendre un octet exprimé en binaire signé est 0111 1111, ce qui correspond à $+127_{10}$. La plus petite valeur qu'il peut prendre est 1111 111, soit -127_{10}. Nous avons toujours dit qu'un octet (huit bits) peut prendre 2^8=256 valeurs distinctes. Remarquez qu'ici, il ne peut prendre que 255 valeurs différentes, en effet, entre -127_{10} et $+127_{10}$ il n'y a que 255 valeurs entières !

En fait, notre octet exprimé en binaire signé peut toujours prendre 256 formes différentes car il existe avec ce type de notation deux zéros :

- 0000 0000 est le zéro positif,
- 1000 0000 est le zéro négatif.

Binaire sur un octet	Signe	Valeur décimale
0000 0000	positif	« + » 0
1000 0000	négatif	« - » 0
0000 0001	positif	1

2.3
Soustraction binaire

Les règles de la soustraction bit à bit sont résumées ci-dessous.

1 - 1 = 0	
1 - 0 = 1	
0 - 0 = 0	
0 - 1 = -1	→ résultat négatif

Rappelons la forme générale d'une soustraction :

	Diminuende
-	Diminuteur
=	Différence

On exécute une soustraction binaire comme une soustraction décimale. Si l'on rencontre un 0-1, on suppose que le diminuende est 10_2 et non 0, la soustraction fournit alors 10-1=1 comme résultat mais il convient de reporter une retenue vers la gauche.

L'algorithme usuel permettant de calculer la soustraction s'applique mal aux calculs binaires effectués par des circuits numériques. Pour faire une soustraction, on préfère donc calculer une addition utilisant le complément à 2 dont il va maintenant être question.

3
Le complément à 2

Les circuits logiques à la base de l'électronique numérique ne sont que rarement intéressants lorsque nous leur soumettons nos méthodes usuelles de calcul. Une soustraction, par exemple, sera exécutée par un circuit numérique en exploitant ce que ce dernier sait très bien faire : inverser des bits et les ajouter.

3.1
Complément à 1 et complément à 2

Le complément restreint, ou complément à 1, d'un mot binaire est ce mot binaire une fois que tous ses bits ont été inversés. Tous les 0 deviennent des 1 alors que tous les 1 deviennent des 0. Par exemple, le complément à 1 du quartet 1101 est le quartet 0010.

Le complément vrai d'un mot binaire, aussi appelé complément à 2, est son complément restreint auquel on a ajouté 1. Par exemple, le complément à 2 du quartet 1101 est 0010 + 1, c'est-à-dire 0011.

3.2
Soustraction par complément à 2

Grâce au complément à 2, une soustraction va se transformer en une addition. La soustraction 7-2=5 en décimal devient 0111-0010 en binaire (les nombres binaires étant ici codés sur un quartet). On pose :

```
    0111
-   0010
=   0101
```

Calculons le complément à 2 du diminuteur, ici 2_{10} ou 0010_2 :

```
    0010
    1101  → Complément à 1 de 0010
+      1  → On ajoute 1
=   1110  → Complément à 2 de 0010
```

Pour effectuer notre soustraction nous allons ajouter le complément à 2 du diminuteur au diminuende :

```
    0111  → 7 en décimal
+   1110  → Complément à 2 de 0010 (2₁₀)
=   0101  → On « perd » la dernière retenue
```

Attention ! La méthode soustraction par addition du complément à 2 du diminuteur fonctionne dans tous les cas à condition d'ignorer volontairement la dernière retenue.

3.3
Codage par complément à 2

Nous avons expliqué comment obtenir le complément à 2 d'un nombre binaire mais nous n'avons pas expliqué le principe du codage en complément à 2 indispensable pour distinguer les nombres positifs des nombres négatifs.

Dans plusieurs ouvrages traitant de la numération binaire on trouve un exemple proche de celui que nous allons maintenant développer. Vous achetez une voiture neuve, en sortant du garage, le compteur de celle-ci indique 000000 km. Après un premier kilomètre, il indiquera 000001 puis 000002, etc. Admettons maintenant que vous rouliez en marche arrière. Dans ce cas, le compteur indique 000000 km puis 999999 km (qui signifie ici -1), puis 999998 km (qui signifie ici -2), etc.

Reprenons maintenant notre exemple avec un compteur kilométrique affichant des nombres binaires. Au départ, notre compteur indique toujours 0000 0000. Si nous partons en marche avant, il affichera 0000 0001 puis 0000 0010, etc. Par contre, si nous partons en marche arrière, le compteur indiquera 1111 1111 après un kilomètre puis 1111 1110 après deux kilomètres. 1111 1111 et 1111 1110 représentent donc respectivement -1_{10} et -2_{10}.

Remarquez qu'avec une telle notation les nombres négatifs commencent par 1 (leur MSB vaut 1). Les nombres positifs devront donc commencer par 0 pour être distingués des nombres négatifs. D'où la convention :

- le bit de poids le plus fort (bit de gauche ou MSB) vaut 1 si le nombre est négatif,
- le bit de poids le plus fort vaut 0 si le nombre est positif.

En codant les nombres binaires sur un octet, on constate que les nombres positifs sont ici compris entre 0000 0000 (0_{10}) et 0111 1111(127_{10}) alors que les nombres négatifs, qui commencent tous par 1, sont compris entre 1111 1111 (-1_{10}) et 1000 0000 (-128_{10}). Remarquez qu'avec ce type de notation, il n'y a plus de « +0 » ou de « -0 », on retrouve bien 256 valeurs

possibles pour un octet. Elles sont comprises entre 1000 0000 (-128_{10}) et 0111 1111 (127_{10}).

Cette notation en complément à 2 sur un octet interdit désormais d'ôter 1_2 à 1000 0000_2 (-128_{10}), car on obtiendrait alors un nombre positif (on obtient 0111 1111_2 c'est-à-dire $+127_{10}$) ce qui est aberrant comme résultat du calcul de $-128_{10}-1_{10}$!

De même, il n'est pas possible d'effectuer la somme 0111 $1111_2 + 1_2$ car alors cette somme qui correspond à $127_{10} + 1_{10}$ renverrait 1000 0000_2 comme résultat. 1000 0000_2 correspond, en notation en complément à 2, à -128_{10} et non à $+128_{10}$!

En binaire signé et sur un octet, nous écrivons :

- $+1_{10}$ = 0000 0001,
- -1_{10} = - 0000 0001.

Calculons le complément à 2 de - 0000 0001 :

	0000 0001	→ *|(-)1$_{10}$|*
	1111 1110	→ *Complément à 1 de 0000 0001*
+	1	→ *On ajoute 1*
=	1111 1111	→ *Complément à 2 de 0000 0001*

Le complément à 2 de -0000 0001_2 (valeur absolue plus le signe) correspond à la façon dont le compteur kilométrique de l'exemple précédent affichait -1. Le complément à 2 fournit donc des valeurs successives décroissantes. Le calcul du complément à 2 de -2_{10} (-0000 0010 en binaire signé) correspond bien à 1111 1110. De même, le complément à 2 de -128_{10} (-1000 0000 en binaire signé) est bien 1000 0000 :

	1000 0001	→ *|(-)128$_{10}$|*
	0111 1111	→ *Complément à 1 de 1000 0001*
+	1	→ *On ajoute 1*
=	1000 0000	→ *Complément à 2 de 1000 0001*

Nous ne faisons ici que vérifier ce que notre compteur kilométrique binaire nous a annoncé depuis le début de cet exemple.

En utilisant la notation en complément à 2, un octet permet de coder 256 valeurs comprises entre -128_{10} et $+127_{10}$. Avec cette notation un signe n'est plus nécessaire. En binaire signé comme en notation en complément à 2, le premier bit détermine le signe. Cependant, nous insistons sur les différences qui existent entre le binaire signé et la notation en complé-

ment à 2. Il existe deux zéros en binaire signé alors qu'il n'y en a qu'un en notation en complément à 2. En notation en complément à 2 l'octet peut coder 256 valeurs distinctes (et non 255 comme en binaire signé si on considère que -0 = +0). Le codage des nombres positifs est le même en binaire signé et en notation en complément à 2. C'est le codage des nombres négatifs qui est différent.

Décimal	Binaire, complément à 2	Binaire signé
+127	0111 1111	0111 1111
+1	0000 0001	0000 0001
0	0000 0000	0000 0000 *ou* 1000 0000
-1	1111 1111	1000 0001
-127	1000 0001	1111 1111
-128	1000 0000	impossible sur un octet

La notation en complément à 2 va-t-elle résoudre tous nos problèmes de notation des nombres binaires négatifs ? Oui, à condition de connaître quelques pièges que nous réserve cette notation ! La soustraction effectuée par complément à 2 peut bien sûr fournir un résultat négatif. Celui-ci est tout de suite détecté puisque son bit de plus fort poids vaut 1. Ce résultat négatif n'est pas directement exploitable. Il faut calculer son complément à 2 pour obtenir la valeur absolue du résultat et ne pas oublier que celui-ci est négatif (on obtient un nombre positif qui est la valeur absolue d'un nombre négatif...). Par exemple, calculons 9_{10}-16_{10}. Pour cela on commence par chercher le complément à 2 de 16_{10} :

	0001 0000	→ *16_{10}*
	1110 1111	→ *Complément à 1 de 0001 0000*
+	1	→ *On ajoute 1*
=	1111 0000	→ *Complément à 2 de 0001 0000*

Pour effectuer la soustraction 9_{10}-16_{10} on ajoute 9_{10} (0000 1001_2) au complément à 2 de 0001 0000_2 (16_{10}) que nous venons de calculer.

	0000 1001	→ *9_{10}*
+	1111 0000	→ *Complément à 2 de 0001 0000*
=	1111 1001	→ *MSB vaut 1... Résultat négatif !*
	0000 0110	→ *Complément à 1 de 1111 1001*
+	1	→ *On ajoute 1*
=	0000 0111	→ *Complément à 2 de 1111 1001*

0000 0111_2 correspond à 7_{10}.

Dans le cadre de cet exemple, le bit de plus fort poids du résultat de la soustraction vaut 1. Ainsi, nous savons que le résultat est négatif, et donc, non exploitable directement. Il faut calculer le complément à 2 du résultat pour obtenir la valeur absolue de celui-ci. On obtient 7_{10}. Le résultat étant négatif, on sait maintenant que le résultat est -7_{10}. Vérifiez vous-même que $9_{10}-16_{10} = -7_{10}$.

4
Multiplication binaire

L'algorithme de multiplication utilisé en base 10 peut aussi être utilisé en binaire. On utilise alors des additions et des décalages de bits.

	1 1011	→ 27_{10}
×	101	→ 5_{10}
	1 1011	→ *Premier produit partiel*
+	00 000	→ *Deuxième produit partiel*
+	110 11	→ *Troisème produit partiel*
=	1000 0111	→ *Produit (135_{10})*

La multiplication par additions et décalages est décrite au paragraphe 4.2.

4.1
Multiplication par additions successives

La méthode la plus simple et la plus lourde consiste à procéder par additions successives du multiplicande à lui-même autant de fois que l'indique le multiplicateur. Cette méthode est difficilement adaptable aux nombres rationnels et occupe le processeur pendant un temps beaucoup plus important que les méthodes que nous allons maintenant présenter. Cela dit, un circuit logique peut, sans aucun problème, effectuer une multiplication par additions successives.

4.2
Multiplication par décalage et addition

Reprenons l'exemple cité en introduction :

	1 1011	→ *Multiplicande* (27_{10})
×	101	→ *Multiplicateur* (5_{10})
	1 1011	→ 1×11011
+	00 000	→ 0×11011
+	110 11	→ 1×11011
=	1000 0111	→ *Produit* (135_{10})

Cette multiplication a été effectuée comme une multiplication classique à ceci près que nous sommes ici en base 2. Remarquez que nous avons procédé par décalages puis par additions. Un microprocesseur effectue très facilement un décalage ou une addition de bits, cette méthode est donc appropriée.

Comme nous l'avons dit, cette multiplication a été calculée comme une multiplication classique en base 2 :

- le multiplicateur est parcouru bit par bit à partir de la droite,
- lorsque l'on rencontre un bit à 1 dans le multiplicateur, on reporte intégralement le multiplicande,
- lorsque l'on rencontre un bit à 0 dans le multiplicateur, on reporte juste un zéro,
- à chaque fois que l'on considère un nouveau bit du multiplicateur, on décale d'un bit vers la droite le nouveau produit partiel,
- lorsque le multiplicateur a été parcouru dans sa totalité, on calcule la somme des produits partiels. On obtient ainsi le produit.

On se contente donc d'observer un bit du multiplicateur, de décaler un mot binaire et de calculer une somme de nombres binaires (en utilisant une éventuelle retenue).

Remarquez les effets d'un décalage :

- un décalage d'un bit vers la gauche revient à multiplier par 2_{10} le mot binaire (multiplication par 10_2),
- un décalage d'un bit vers la droite revient à diviser par 2_{10} le mot binaire (division par 10_2).

4.3
Vers une multiplication plus rapide

Pour optimiser l'algorithme de calcul que nous venons de décrire et qui n'est autre que celui que nous utilisons « sur le papier », il est de bon ton d'exploiter l'une des facilités offertes par la numération binaire. Nous allons donc détecter les suites de 0 et les suites de 1 dans le multiplicateur.

La rencontre d'un 0 dans le multiplicateur est synonyme de décalage d'un rang vers la gauche. Les suites de 0 gagnent donc à être traitées en une seule fois. On effectue donc un unique décalage sur autant de rangs qu'il y a de 0.

Dans le cas d'une suite de 1, différentes simplifications sont envisageables l'une des plus évidentes se fonde sur l'égalité suivante :

$$1111_2 = 1\ 0000_2 - 1_2$$

Par suite :

$$N_2 \times 1111_2 = (N_2 \times 1\ 0000) - N_2$$

On évite alors une succession de produits intermédiaires. On se contente de décaler de plusieurs rangs et de reporter le multiplicande.

D'autres méthodes sont envisageables, on peut fractionner la multiplication et la calculer en "parallèle" ou encore utilisée une "retenue anticipée", etc.

5
Division binaire

Avec des nombres en base 2, la division, comme la multiplication, peut très facilement être calculée avec l'algorithme "classique". On obtient un résultat de la forme *diviseur* × *quotient* = *dividende* + *reste*.

Calculons $156_2 \div 12_2$, c'est-à-dire $1001110_2 \div 1100_2$:

	1001110	1100
-	0000	0
	10011	01
-	01100	
	001111	011
-	001100	
	0000110	0110

On obtient : $1001110_2 = 1100_2 \times 110_2 + 110_2$, c'est-à-dire, une fois revenu en décimal, $78_{10} = 6_{10} \times 12_{10} + 6_{10}$.

On procède donc par soustraction (dans cet exemple, nous ne vous avons pas infligé la soustraction par complément à 2...) et par décalage à droite. Par ailleurs, il faut à chaque stade comparer le diviseur au reste.

6

Pour aller plus loin...

6.1

Précision et longueur du mot binaire

Un mot binaire constitué de *n* bits peut représenter 2^n valeurs différentes, ainsi :

- un bit peut prendre deux valeurs, 0 et 1,
- un quartet (4 bits) peut prendre $2^4 = 16$ valeurs,
- un octet (8 bits) peut prendre $2^8 = 256$ valeurs,
- un mot de 16 bits peut prendre $2^{16} = 65\,536$ valeurs,
- un mot de 32 bits peut prendre $2^{32} = 4\,294\,967\,296$ valeurs,
- un mot de 64 bits peut prendre $2^{64} = 1{,}844 \times 10^{19}$ valeurs,etc.

Le mot binaire standard que traite un ordinateur a une longueur limitée. Cela limite l'étendue des valeurs qu'est capable de traiter la machine. On

contourne ce problème en juxtaposant des mots binaires de base. Ainsi, si le mot binaire de base de la machine comporte 16 bits, il sera possible de travailler en « double précision » en juxtaposant deux mots de base. 32 bits seront alors disponibles. Par suite, il est possible de juxtaposer trois mots de base, on parlera alors de « triple précision ».

Dans le cas de la double précision, deux mots binaires de base ont été juxtaposés. On distingue alors au sein du « double mot » un mot de « poids fort » à gauche, et un mot de « poids faible » à droite.

6.2
Synthèse des notations négatives

Le tableau ci-dessous résume les différentes notations négatives dont nous disposons. Remarquez que le complément à 1 peut aussi être utilisé pour coder des nombres négatifs.

Décimal	Valeur absolue plus signe	Binaire signé	Complément à 1	Complément à 2
+255	1111 1111	-	-	-
+128	1000 0000	-	-	-
+127	0111 1111	0111 1111	0111 1111	0111 1111
+1	0000 0001	0000 0001	0000 0001	0000 0001
+0	0000 0000	0000 0000	0000 0000	0000 0000
-0	- 0000 0000	1000 0000	1111 1111	0000 0000
-1	- 0000 0001	1000 0001	1111 1110	1111 1111
-2	- 0000 0010	1000 0010	1111 1101	1111 1110
-127	- 0111 1111	1111 1111	1000 0000	1000 0001
-128	- 1000 0000	-	-	1000 0000
-255	- 1111 1111	-	-	-

6.3
Virgule fixe et virgule flottante

La représentation d'un nombre rationnel impose la représentation de la partie entière et de la partie décimale de celui-ci. Deux cas peuvent se pré-

senter. Dans le cas le plus simple, on peut envisager une virgule fixe. On détermine à l'avance le nombre de bits affectés à la partie entière et le nombre de bits affectés à la partie décimale. Mais on peut aussi envisager l'emploi d'une virgule flottante. Cette virgule dite « flottante » vient alors se placer entre deux bits quelconques du mot binaire en fonction de la valeur à représenter.

L'une des méthodes envisageables afin de marquer la position de la virgule flottante est la réservation de quelques bits du mot binaire afin de leur affecter une valeur qui correspond au numéro du bit après lequel doit se placer la virgule.

6.4
Notation scientifique

La notation dite "scientifique" autorise la représentation de valeurs beaucoup plus grandes que le nombre de valeurs que peut normalement représenter un mot binaire donné. Considérons un mot binaire de 16 bits. Celui-ci ne pourra prendre que 65536 valeurs différentes. Il nous faut une méthode permettant la représentation de valeurs supérieures à 65536 ! La notation scientifique consiste en une multiplication de la mantisse par un facteur 10 élevé à une puissance déterminée. Ainsi, dans le cadre de la notation scientifique, affectons les 16 bits de notre mot binaire :

- 1 bit pour le signe de la mantisse ("+" ou "-"),
- 1 bit pour l'exposant du facteur 10 ("+" ou "-"),
- 4 bits pour coder l'exposant du facteur 10 (de 10^0 à 10^{15}),
- 10 bits pour coder la mantisse qui comportera donc trois chiffres significatifs puisqu'elle peut représenter des nombres jusqu'à $2^{10} = 1024$.

Bien entendu, en double précision (ici, 32 bits), la mantisse et l'exposant seraient beaucoup plus précis.

4

Base 8, base 16 et codage des quartets

Avec l'introduction de l'hexadécimal, nous allons pouvoir grouper les bits par quartets (groupes de quatre bits) et représenter chaque quartet par l'un des seize chiffres hexadécimaux. Ce chapitre commence par l'étude de la numération en base 8 chargée d'introduire la base 16.

Rappelons qu'un indice placé après un nombre indique la base dans laquelle le nombre est exprimé, ainsi, $15_{10} = 1111_2 = 17_8 = A_{16}$! Plus particulièrement dans le cas des nombres décimaux, on a pris l'habitude de faire suivre chaque nombre de la première lettre du nom de la base, cela dit, les notation A_{16} et AH ou Ah ou encore #Ah sont équivalentes d'un point de vue mathématique.

1 La base 8

Aujourd'hui tombé en désuétude, le calcul en base 8 a beaucoup simplifié la programmation des microprocesseurs manipulant des mots binaires d'une longueur multiple de 12. En base 8, on utilise sept chiffres de 0 à 7. Comme nous l'avons déjà dit, 10 représente toujours la base, donc $10_8 = 8_{10}$ et $11_8 = 9_{10}$. On ne manipule en informatique que des 0 et des 1, la base 2 s'avère donc incontournable. Par commodité, on la remplace par la base 8 ou la base 16 ($8=2^3$ et $16=2^4$). La base 10 ne peut pas se substituer à la base 2 car il n'existe pas de nombre entier n tel que $2^n=10$.

Le tableau ci-dessous présente quelques équivalences entre les nombres binaires, octaux et décimaux.

Nombres en base 10	Nombres en base 8	Nombres en base 2
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	10	1000
9	11	1001
10	12	1010
11	13	1011
12	14	1100
13	15	1101
14	16	1110
15	17	1111
16	20	10000
...	...	...
$64 = 8^2$	100	1000000
$512 = 8^3$	1000	1000000000
$4096 = 8^4$	10000	1000000000000

La forme canonique d'un nombre octal entier est :

$$N_8 = \alpha_n 8^n + \alpha_{n-1} 8^{n-1} + \alpha_{n-2} 8^{n-2} + \ldots + \alpha_0 8^0$$

Comme $8 = 2^3$, on peut poser :

$$N_8 = \alpha_n 2^{3\times n} + \alpha_{n-1} 2^{3\times n-1} + \alpha_{n-2} 2^{3\times n-2} + \ldots + \alpha_0 2^{3\times 0}$$

Fort de ce résultat, observez attentivement les trois dernières lignes du tableau ci-dessus. Vous constaterez qu'un zéro octal correspond ici à trois zéros binaires...

Un nombre octal peut prendre $2^3 = 8$ valeurs différentes, c'est-à-dire autant de valeurs que trois nombres binaires. Un nombre octal comporte donc trois fois moins de chiffres que son équivalent binaire. Ainsi, chaque nombre octal peut remplacer un groupe de trois bits, alors qu'un nombre décimal ne représente pas un groupe de *n* bits tout simplement parce

qu'il n'existe pas d'entier *n* tel que $2^n=10$ (alors que $2^3=8$, ce qui justifie l'emploi de l'octal comme représentation du binaire).

Considérons un processeur traitant des mots binaires de 12 bits. Soit l'un de ces mots : 100101110111 (2423_{10}). On peut découper ce mot en tranches de trois bits puis représenter chacune de ces tranches à l'aide d'un nombre octal :

100 101 110 111
↓ ↓ ↓ ↓
4 5 6 7

On a ici $100101110111_2 = 4567_8 = 2423_{10}$. Donc, pour convertir un nombre binaire en un nombre octal, il suffit de le découper en tranches de trois bits depuis la gauche vers la droite. Chaque groupe de trois bits correspond à un chiffre octal (chiffres de 0 à 7) qu'il suffit de reporter pour reconstituer le nombre octal. Inversement, pour convertir un nombre octal en un nombre en base 2, il suffit de traduire chaque chiffre par les trois bits qui lui correspondent en base 2.

Selon un principe similaire nous allons maintenant voir comment quatre bits vont être remplacés par un unique chiffre hexadécimal.

2
La base 16

Jusqu'à maintenant, nous avons décrit la forme canonique d'un nombre, la base 2 et la base 8 dans le but d'introduire maintenant la base 16 ou hexadécimal (ou encore, hexa). La maîtrise de l'hexadécimal est indispensable à quiconque prétend programmer en assembleur, car, comme nous allons le voir, un quartet est représenté par un chiffre hexadécimal.

En hexadécimal 16_{10} est représenté par 10_{16}. Se pose alors le problème de la représentation de 10_{10}, 11_{10}, 12_{10}, 13_{10}, 14_{10} et 15_{10} et hexadécimal, base dans laquelle il nous faut disposer de 16 chiffres. En plus des dix chiffres habituels (0 à 9), l'hexadécimal a recours aux six premières lettres de l'alphabet (A à F). Notez que ces lettres sont toujours écrites en majuscules au sein des nombres hexadécimaux.

Le tableau ci-dessous présente quelques équivalences entre les nombres binaires, octaux, décimaux et hexadécimaux.

Nombres en base 10	Nombres en base 16	Nombres en base 8	Nombres en base 2
0	0	0	0
1	1	1	1
2	2	2	10
3	3	3	11
4	4	4	100
5	5	5	101
6	6	6	110
7	7	7	111
8	8	10	1000
9	9	11	1001
10	A	12	1010
11	B	13	1011
12	C	14	1100
13	D	15	1101
14	E	16	1110
15	F	17	1111
16	10	20	10000
...	...	...	...
$64=8^2=4\times16$	40	100	1000000
$512=2\times2\times16$	200	1000	1000000000
$4096=16^3=8^4$	1000	10000	1000000000000

La forme canonique d'un nombre hexadécimal entier est :

$$N_{16} = \alpha_n 16^n + \alpha_{n-1} 16^{n-1} + \alpha_{n-2} 16^{n-2} + \ldots + \alpha_0 16^0$$

Comme $16 = 2^4$, on pose :

$$N_{16} = \alpha_n 2^{4\times n} + \alpha_{n-1} 2^{4\times n-1} + \alpha_{n-2} 2^{4\times n-2} + \ldots + \alpha_0 2^{4\times 0}$$

Pour retrouver l'équivalent décimal d'un nombre hexadécimal, il suffit de remplacer les α_n de la forme canonique par les chiffres du nombre hexadécimal. On se réfère ensuite à une table des puissances entières de 16 telle que celle que nous vous proposons ci-dessous.

$16^0 = 1$	$16^3 = 4096$	$16^6 = 16777216$
$16^1 = 16$	$16^4 = 65536$	$16^7 = 268435456$
$16^2 = 256$	$16^5 = 1048576$	$16^8 = 4294967296$

Par exemple, convertissons $5A2B_{16}$ en décimal :

$$5A2B_{16} = 5_{16}\times16^3 + A_{16}\times16^2 + 2_{16}\times16^1 + B_{16}\times16^0$$

Par suite,

$$5A2B_{16} = 5_{10}\times16^3 + 10_{10}\times16^2 + 2_{10}\times16^1 + 11_{10}\times16^0$$

$$5A2B_{16} = 20480_{10} + 2560_{10} + 32_{10} + 11_{10}$$

$$5A2B_{16} = 23083_{10}$$

Pour convertir un nombre décimal en hexadécimal, on peut procéder par divisions successives par 16 en retenant les restes. Vous pouvez aussi utiliser une table de conversion entre base 10 et base 16.

Octet 3				Octet 2				Octet 1			
Quartet 6		Quartet 5		Quartet 4		Quartet 3		Quartet 2		Quartet 1	
Hexa.	*Decimal*	*Hexa.*	*Decimal*	*Hexa.*	*Decimal*	*Hexa.*	*Decimal*	*Hexa.*	*Decimal*	*Hexa.*	*Decimal*
0	0	0	0	0	0	0	0	0	0	0	0
1	1 048 576	1	65 536	1	4 096	1	256	1	16	1	1
2	2 097 152	2	131 072	2	8 192	2	512	2	32	2	2
3	3 145 728	3	196 608	3	12 288	3	768	3	48	3	3
4	4 194 304	4	262 144	4	16 384	4	1 024	4	64	4	4
5	5 242 880	5	327 680	5	20 480	5	1 280	5	80	5	5
6	6 291 456	6	393 216	6	24 576	6	1 536	6	96	6	6
7	7 340 032	7	458 752	7	28 672	7	1 792	7	112	7	7
8	8 388 608	8	524 288	8	32 768	8	2 048	8	128	8	8
9	9 437 184	9	589 824	9	36 864	9	2 304	9	144	9	9
A	10 485 760	A	655 360	A	40 960	A	2 560	A	160	A	10
B	11 534 336	B	720 896	B	45 056	B	2 816	B	176	B	11
C	12 582 912	C	786 432	C	49 152	C	3 072	C	192	C	12
D	13 631 488	D	851 968	D	53 248	D	3 328	D	208	D	13
E	14 680 064	E	917 504	E	57 344	E	3 584	E	224	E	14
F	15 728 640	F	983 040	F	61 440	F	3 840	F	256	F	15

Par soustractions successives et en exploitant le tableau de conversion, nous allons convertir 44092_{10} en hexadécimal :

- on recherche dans le tableau la valeur immédiatement inférieure à 44092_{10}, cette valeur est 40960_{10},
- à 40960_{10} correspond A_{16} dans la colonne « quartet 4 ». Un chiffre hexadécimal codant un quartet et le premier chiffre hexadécimal trouvé étant dans la quatrième colonne, on en déduit

que le nombre hexadécimal recherché comporte quatre chiffres. A ce stade, le nombre recherché est $A???_{16}$,

- on calcule 44092_{10}-40960_{10} = 3132_{10}. Puis on cherche dans le tableau le nombre immédiatement inférieur à 3132_{10}. On trouve 3072_{10} dans la colonne « quartet 3 », le nombre recherché devient $AC??_{16}$ (3072_{10} correspond C_{16}),
- de même on calcule 3132_{10}-3072_{10} = 60_{10}. En considérant 48_{10} dans la colonne « quartet 2 », le nombre recherché devient $AC3?_{16}$,
- on calcule ensuite 60_{10}-48_{10} = 12_{10}. En considérant 12_{10} dans la colonne « quartet 1 », on obtient le nombre recherché, c'est-à-dire 44092_{10} = $AC3C_{16}$.

Rassurez-vous, la plupart des calculatrices scientifiques disposent de fonctions de conversion entre les bases 2, 8, 10 et 16 ! Dans le cadre de la programmation en assembleur sur HP-48, il est très utile d'avoir avec soi une petite calculatrice scientifique facile d'emploi (TI-52, fx 4000P, etc.) qui ne servira qu'aux conversions entre bases. Ainsi, il ne sera pas nécessaire de quitter l'édition d'un programme sur HP-48 pour effectuer à l'aide de celle-ci un simple calcul de conversion.

3
Coder les quartets

Un quartet est un groupe de quatre bits, il peut donc prendre 2^4=16 valeurs. Un nombre hexadécimal peut, lui aussi, prendre 16 valeurs, pour des raisons de lisibilité, on peut donc remplacer chaque quartet par un nombre hexadécimal. Comme nous le verrons, la HP-48 travaille à partir de quartets. Le tableau ci-dessous rappelle les équivalences entre quartets et chiffres hexadécimaux.

Quartet	Chiffre hexa.	Quartet	Chiffre hexa.	Quartet	Chiffre hexa.
0000	0	0110	6	1100	C
0001	1	0111	7	1101	D
0010	2	1000	8	1110	E
0011	3	1001	9	1111	F
0100	4	1010	A	0001 0000	10
0101	5	1011	B	0001 0001	11

Il n'existe que seize quartets différents. A chacun d'eux nous pouvons substituer l'un des seize chiffres hexadécimaux. Cette substitution ne présente que des avantages :

- on code les mêmes informations mais avec quatre fois moins de symboles,
- le binaire est très peu lisible et est donc une source permanente d'erreurs. L'hexadécimal est moins propice aux fautes de saisie que le binaire et permet une recherche plus commode des erreurs.

Attention ! En assembleur, un chiffre hexadécimal n'est qu'une représentation d'un quartet binaire. Ainsi, un calcul tel que le complément restreint d'un nombre hexadécimal ne sera, en fait, que le calcul du complément restreint du nombre binaire représenté par le nombre hexadécimal.

Par exemple, convertissons $A0F1_{16}$ en binaire. On considère l'un après l'autre tous les chiffres hexadécimaux du nombre puis on leur substitue leur équivalent binaire après avoir consulté le tableau d'équivalence de la page précédente, ainsi :

A	0	F	1
↓	↓	↓	↓
1010	0000	1111	0001

On peut aussi effectuer la conversion de 1010 0000 1111 00001 en hexadécimal, toujours à partir du tableau de la page précédente :

1010	0000	1111	0001
↓	↓	↓	↓
A	0	F	1

On peut bien sûr convertir un nombre binaire en hexadécimal même si le nombre binaire considéré ne comporte pas un nombre de bits multiple de 4. Il suffit pour cela d'ajouter de un à trois zéros à gauche du nombre binaire pour qu'il comporte un nombre de bits multiple de 4. Le découpage en quartets est alors facile.

L'octal étant abandonné, la programmation en assembleur s'effectue aujourd'hui en hexadécimal avec la quasi totalité des microprocesseurs. Remarquons qu'une représentation en base 256, permettant de représenter un octet avec un unique symbole, n'a pas d'intérêt, car l'excès de symboles devient une nouvelle source d'erreurs. La base 16 constitue donc le compromis attendu entre lisibilité et utilité pratique.

Le SATURN de la HP-48 traite l'information quartet par quartet, cependant la plupart des microprocesseurs traite des groupes d'octets. Chaque octet est alors codé à l'aide de deux chiffres hexadécimaux.

5
Le décimal codé binaire

Le microprocesseur ne manipule que des bits qu'il convient de grouper par quartets. Bien qu'un quartet corresponde à un chiffre hexadécimal, on peut adapter le codage des quartets de façon à ne représenter que des nombres en base 10 à l'aide de quartets. Comme nous le verrons, le SATURN peut calculer à l'aide de nombres hexadécimaux ou de nombres décimaux. Dans ce dernier cas on fait appel au « décimal codé binaire » ou DCB (appelée BCD par les anglo-saxons).

1
Le décimal codé binaire (DCB)

Un quartet (un groupe de 4 bits) peut prendre seize valeurs. En DCB on se limite à l'utilisation des dix premières valeurs, on reste ainsi en décimal.

Décimal	Binaire sur 4 bits	DCB	Décimal	Binaire sur 4 bits	DCB
0	0000	0000	8	1000	1000
1	0001	0001	9	1001	1001
2	0010	0010	10	1010	0001 0000
3	0011	0011	11	1011	0001 0001
4	0100	0100	12	1100	0001 0010
5	0101	0101	13	1101	0001 0011
6	0110	0110	14	1110	0001 0100
7	0111	0111	15	1111	0001 0101

En DCB, on réserve quatre nombres binaires (quatre bits) pour coder un nombre décimal. Par conséquent, certains quartets représentant A_{16} à F_{16} sont interdits en DCB.

Décimal	Hexadécimal	Quartets interdits en DCB	Equivalence en DCB
10	A	1010	0001 0000
11	B	1011	0001 0001
12	C	1100	0001 0010
13	D	1101	0001 0011
14	E	1110	0001 0100
15	F	1111	0001 0101

Pour traduire 45_{10} en DCB, on parcourt le nombre, chiffre par chiffre, et on reporte pour chaque chiffre son équivalent en DCB :

```
  4     5
  ↓     ↓
0100  0101
```

Attention ! Le DCB emploie des bits mais ceux-ci ne doivent pas être traités comme ceux d'un mot binaire classique. Ainsi, 4510 est représenté par 0100 0101 en DCB et par 0010 1101 en binaire sur huit bits.

Le code DCB usuel est celui que nous venons de présenter (on code un digit décimal sur quatre digits binaires). Ce codage DCB usuel porte le nom de DCB compacté par opposition au DCB étendu.

En DCB étendu chaque chiffre décimal est codé sur un octet (8 bits). Les quatre bits de poids fort (quartet de poids fort) de l'octet sont donc toujours 0000. L'utilisation de ce code est donc peu performante.

En DCB étendu un octet code des valeurs comprises entre 0 et 9 alors qu'en DCB compacté un octet code des valeurs comprises entre 0 et 99 puisque chacun des deux quartets de l'octet peut coder un nombre compris entre 0 et 9.

Décimal	DCB compacté	DCB étendu
9	1001	00001001
10	0001 0000	00000001 00000000
11	0001 0001	00000001 00000001
12	0001 0010	00000001 00000010

2
Calculer en DCB

Comme tous les autres moyens de codage des nombres, le DCB peut servir à tous les types de calcul. Cependant, certains quartets étant interdits (ils n'ont de sens qu'en hexadécimal), des précautions particulières doivent être prises. Le SATURN de la HP-48, est un microprocesseur capable de calculer directement en DCB, il est donc capable de prévenir les erreurs liées au mélange du DCB avec le binaire naturel (à condition d'avoir été placé en mode DCB !). Cela dit, nous allons étudier quelques erreurs de calcul en DCB, et ce, afin de bien comprendre le calcul en DCB.

■ Dépassement de la capacité d'un quartet

Effectuons la somme $6_{10}+6_{10}$:

$$\begin{array}{rll} & 0000\ 0110 & \rightarrow\ 6_{10}\ \textit{en DCB} \\ + & 0000\ 0110 & \rightarrow\ 6_{10}\ \textit{en DCB} \\ \hline = & 0000\ 1100 & \rightarrow\ \textit{Quartet interdit en DCB} \end{array}$$

Or 1100 est l'un des quartets interdits en DCB ! Ici, le dépassement a lieu sur le quartet de poids faible. Pour corriger l'erreur et obtenir un résultat correct formulé en DCB, on ajoute 6_{10} au résultat afin de « sauter » les six quartets interdits en DCB :

$$\begin{array}{rll} & 0000\ 1100 & \rightarrow\ 12_{10}\ \textit{en binaire naturel} \\ + & 0000\ 0110 & \rightarrow\ \textit{on ajoute}\ 6_{10} \\ \hline = & 0001\ 0010 & \rightarrow\ 12_{10}\ \textit{en DCB} \end{array}$$

0001 0010 représente, en DCB, le résultat correct (12_{10}). Notez que lorsqu'un quartet « interdit » apparaît sur le quartet de poids fort on ajoute 60_{10} et non 6_{10} afin de corriger l'erreur.

■ Propagation d'une retenue

Outre le dépassement de la capacité de chaque quartet (apparition de l'un des six quartets interdits en DCB), il convient de surveiller la propagation des retenues d'un quartet vers un autre quartet plus significatif (de

poids plus fort, donc vers la gauche...). Par exemple, calculons $9_{10} + 9_{10}$, une retenue va passer du quartet de droite à celui de gauche :

$$\begin{array}{lll} & 0000\ 1001 \rightarrow 9_{10} & \\ + & 0000\ 1001 \rightarrow 9_{10} & \\ \hline = & 0001\ 0010 \rightarrow 12_{10} & \textit{en DCB} \end{array}$$

Or 0001 0010 représente, en DCB, 12_{10} qui est un résultat inexact en DCB. La correction consiste à ajouter 6_{10} au résultat :

$$\begin{array}{lll} & 0001\ 0010 \rightarrow 12_{10} & \textit{en DCB} \\ + & 0000\ 0110 \rightarrow 6_{10} & \textit{en DCB} \\ \hline = & 0001\ 1000 \rightarrow 18_{10} & \textit{en DCB} \end{array}$$

0001 1000 représente, en DCB, 18_{10} qui est le résultat correct. La retenue se propageant du quartet de poids faible vers les quartet de poids fort est appelée retenue intermédiaire ou demi-retenue.

Il est aussi possible qu'une retenue sorte du quartet de poids fort. Par exemple, calculons $91_{10} + 91_{10}$:

$$\begin{array}{lll} & 1001\ 0001 \rightarrow 91_{10} & \textit{en DCB} \\ + & 1001\ 0001 \rightarrow 91_{10} & \textit{en DCB} \\ \hline = & 1\ 0010\ 0010 \rightarrow 122_{10} & \textit{en DCB} \end{array}$$

Le résultat est 0001 0010 0010 en DCB ce qui correspond à 122_{10} et est un résultat faux. La correction consiste à ajouter 60_{10} au résultat :

$$\begin{array}{lll} & 0001\ 0010\ 0010 \rightarrow 122_{10} & \textit{en DCB} \\ + & 0000\ 0110\ 0010 \rightarrow 60_{10} & \textit{en DCB} \\ \hline = & 0001\ 1000\ 0010 \rightarrow 182_{10} & \textit{en DCB} \end{array}$$

6 Logique

A partir de fonctions logiques élémentaires (OU, ET, inversion), il est possible de construire des circuits numériques combinatoires ou séquentiels réalisant des tâches très complexes. L'une des nombreuses applications des fonctions logiques à la programmation en assembleur est la comparaison de données en entrée à celles d'un masque de clavier afin de détecter une pression sur l'une des touches du clavier.

1 Fonctions logiques élémentaires

1.1 Introduction

Créée en 1847 par Georges Boole, l'algèbre booléenne n'a pas connu d'application pratique jusqu'en 1938 lorsque Claude Shannon l'a appliquée à la conception des circuits électriques. Toute l'électronique numérique (électronique digitale) se fonde sur l'algèbre de Boole, par suite, les ordinateurs et les microprocesseurs exploitent l'algèbre de Boole qui est adaptée à la manipulation de bits.

L'algèbre de Boole ne s'applique qu'à deux situations :

- un état dit « vrai », noté 1, qui correspond au passage du courant électrique dans un circuit,
- un état dit « faux », noté 0, pour lequel le courant ne passe pas.

Attention ! Nous notons 1 l'état « vrai » car nous utilisons une logique positive dans le cadre de laquelle l'état vrai correspond, d'un point de vue électrique, à un potentiel supérieur à celui de l'état « faux ». En général, l'état « vrai » correspond à un potentiel de +5 V au point considéré du circuit alors que l'état « faux » correspond à 0 V.

L'algèbre de Boole fait appel au binaire, on retrouve la notion de complément restreint déjà vue :

- 0 est le complément (complément restreint) de 1,
- 1 est le complément de 0.

Il existe trois fonctions logiques de base qui permettent, par combinaison, de construire toutes les autres :

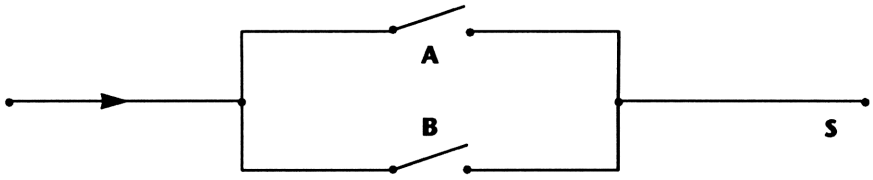
Nom français	Nom anglais	Notation
ET	AND	·
OU	OR	+
NON ou Inversion	NOT ou Invert	¯

L'inversion logique se note en surlignant la valeur logique donnée en argument. Ainsi l'inversion logique de A sera notée $\bar{A}$ qui se lit "A-barre" ou "non A".

En fait, seules deux fonctions logiques sont indispensables car elles permettent d'élaborer la troisième, et, par suite, toutes les combinaisons des trois fonctions logiques de base.

1.2
Ou logique (somme logique)

Le OU logique a une interprétation électrique élémentaire. Elle consiste en deux interrupteurs, notés A et B, montés en parallèle. Il suffit que l'un des interrupteurs soit fermé pour que le courant passe.



On représente par 1 un interrupteur fermé permettant le passage du courant (la variable A vaut 1 si l'interrupteur A est fermé, A=0 si A est ouvert). L'état de la sortie S est fonction des états des interrupteurs.

Comme pour toutes les fonctions logiques, on peut établir pour le OU logique une table de vérité. Il s'agit d'un tableau résumant l'état de la sortie S en fonction des entrées, ici, A et B.

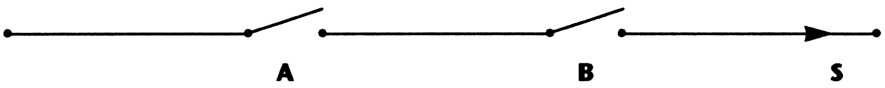
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

S=A+B est l'équation caractéristique qui se lit « S égale A ou B ».

Attention ! On appelle somme logique le *OU* logique mais cela n'a rien à voir avec l'addition abordée dans le chapitre traitant du calcul en base 2.

1.3
Et logique (produit logique)

Le ET logique a, lui aussi, une interprétation électrique qui consiste en deux interrupteurs A et B montés en série. L'état de la sortie S est fonction des états des interrupteurs A et B. Le ET logique est appelé produit logique mais il ne faut pas le confondre avec la multiplication classique. L'équation caractéristique de l'opération ET logique est S=A·B où l'on pourra omettre le point s'il n'y a pas d'équivoque.



La table de vérité du ET logique vous est présentée ci-dessous.

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

Avec la fonction *ET* logique, la sortie est à 1 si, et seulement si, toutes les variables représentant les entrées sont elles-aussi à 1.

1.4
Inversion

L'inversion logique consiste juste à inverser le bit donné en argument. L'équation caractéristique de l'inversion logique est $S = \bar{A}$ qui se lit « S égale non-A » ou « S égale A-barre ». La table de vérité de cette fonction est présentée ci-dessous.

A	S
0	1
1	0

1.5
OU-NON (NOR) et ET-NON (NAND)

Les fonctions logiques peuvent être associées à volonté. Deux associations courantes sont les fonctions OU-NON (NOR) et ET-NON (NAND). Les tables de vérité obtenues pour ces deux fonctions sont celles des fonctions OU et ET inversées. Avec la fonction OU-NON la sortie vaut 1 à la condition que toutes les entrées soient à 0. Vous constaterez que c'est l'inverse

pour la fonction ET-NON avec laquelle la sortie ne passe à 0 que si toutes les entrées sont à 1.

OU-NON a pour équation caractéristique $S=\overline{A+B}$, sa table de vérité est présentée ci-dessous.

A	B	S
0	0	1
0	1	0
1	0	0
1	1	0

ET-NON a pour équation caractéristique $S=\overline{A \cdot B}$, sa table de vérité est présentée ci-dessous.

A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

2
Algèbre de Boole

La tableau ci-dessous résume les fonctions logiques que nous connaissons et que nous allons apprendre à simplifier.

Nom français	Nom anglais	Equation caractéristique
OU	OR	$S = A+B$
ET	AND	$S = A \cdot B$
NON ou Inversion	NOT ou Invert	$S = \overline{A}$
OU-NON	NOR	$S = \overline{A+B}$
ET-NON	NAND	$S = \overline{A \cdot B}$

2.1
Simplification des expressions

Rappelons le principe de la complémentation :

$$\overline{0} = 1$$
$$\overline{1} = 0$$

Par suite,

$$\overline{\overline{0}} = 0$$
$$\overline{\overline{0}} = \overline{1}$$

Deux inversions s'annulant mutuellement, on ne devra jamais trouver plus d'une barre au-dessus d'une expression simplifiée. On dit que $\overline{A}$ est le complément de A, de même, $\overline{A \cdot B}$ est le complément de A·B. Rappelons que les notations A·B et AB sont équivalentes.

Puisqu'une variable booléenne (un bit) ne peut prendre que deux valeurs, on a :

$$A + \overline{A} = 1$$
$$A \cdot \overline{A} = 0$$

Après application au produit et à la somme logiques, on obtient :

$$AB + \overline{AB} = 1$$
$$AB \cdot \overline{AB} = 0$$

Par ailleurs, quelles que soient les variables booléennes A et B, on a :

$$A + B + 1 = 1$$
$$A \cdot B \cdot 0 = 0$$

On constate donc que 0 est l'élément absorbant du produit logique. Ci-après, quelques autres propriétés simples :

$$A + 0 = A$$
$$A \cdot 1 = A$$
$$A + A + A\ldots + A = A$$
$$A \cdot A \cdot A\ldots \cdot A = A$$
$$(A + B) + (A + B)\ldots + (A + B) = A + B$$

2.2
Axiomatique et relations fondamentales

Certaines des propriétés que nous allons maintenant énoncer sont directement issues de l'algèbre classique. Par contre, d'autres sont propres à l'algèbre de Boole. Dans les exemples qui suivent A, B et C désignent des variables booléennes (qui sont pour nous des bits).

■ Symétrie et commutativité

Si on a l'égalité A = B, alors on peut poser B = A. Dans le cas d'un produit logique, on a A·B = B·A.

■ Transitivité

Si on a les égalités A = B et B = C, alors on peut poser A = C.

■ Associativité

Dans le cas du produit logique on a l'égalité (A·B)·C = A·(B·C) = ABC. De même, la somme logique A+(B+C) est équivalente à (A+B)+C ainsi qu'à A+B+C.

■ Distributivité

On a l'égalité (A·B) + (A·C) = A·(B+C) qu'il convient d'utiliser pour de très nombreuses simplifications.

■ Absorption

Les égalités A·(A+B) = A et A+(A·B) = A sont parmi les plus déconcertantes mais autorisent de très intéressantes simplifications.

■ Relations fondamentales

Nous allons maintenant passer en revue une dizaine de relations fondamentales. Elles font suite aux propriétés que nous venons de citer.

Quelques propriétés fondamentales :

$$\overline{\overline{A}} = A$$
$$A + A = A$$
$$A \cdot A = A$$
$$A + 1 = 1$$
$$A \cdot 0 = 0$$
$$A + (\overline{A} \cdot B) = A + B$$
$$B + (\overline{B} \cdot A) = A + B$$
$$A \cdot (\overline{A} + B) = A \cdot B$$
$$B \cdot (\overline{B} + A) = A \cdot B$$
$$A{\cdot}B + B{\cdot}C + \overline{A}{\cdot}C = A{\cdot}B + \overline{A}{\cdot}C$$

La première égalité correspond au théorème d'involution, les deux suivantes sont relatives au théorème d'idempotance. Deux autres égalités sont capitales, elles sont issues des théorèmes de De Morgan.

2.3
Théorèmes de De Morgan

■ Premier théorème de De Morgan

Il s'énonce ainsi : « le complément d'une somme de variables booléennes est égal au produit des compléments de chacun de ses termes ». Pour deux et trois variables, on pose :

$$\overline{A+B} = \overline{A} \cdot \overline{B}$$
$$\overline{A+B+C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$$

■ Second théorème de De Morgan

Il s'énonce ainsi : « le complément d'un produit de variables booléennes est égal à la somme des compléments de ces variables booléennes ». Pour deux et trois variables, on pose :

$$\overline{A{\cdot}B} = \overline{A} + \overline{B}$$
$$\overline{A{\cdot}B{\cdot}C} = \overline{A} + \overline{B} + \overline{C}$$

Bien sûr, ces théorèmes s'appliquent aussi à plus de trois variables.

2.4
Min-terme et max-terme

On dispose de n variables booléennes (ce sont des bits qui peuvent prendre 0 ou 1 pour valeur). On peut affirmer que toutes les variables qui ne valent pas 1 valent 0, et inversement. Considérant cet ensemble des n variables, on introduit deux nouvelles notions :

- le min-terme est le produit logique des n variables,
- le max-terme est la somme logique des n variables.

Chaque variable étant représentée par sa forme vraie ou par sa forme complémentée, on établit le tableau suivant pour quatre variables booléennes A, B, C et D :

A B C D	Min-terme	Max-terme
0 0 0 0	$m_0 = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}$	$M_0 = \bar{A} + \bar{B} + \bar{C} + \bar{D}$
0 0 0 1	$m_1 = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D$	$M_1 = \bar{A} + \bar{B} + \bar{C} + D$
0 0 1 0	$m_2 = \bar{A} \cdot \bar{B} \cdot C \cdot \bar{D}$	$M_2 = \bar{A} + \bar{B} + C + \bar{D}$
0 0 1 1	$m_3 = \bar{A} \cdot \bar{B} \cdot C \cdot D$	$M_3 = \bar{A} + \bar{B} + C + D$
0 1 0 0	$m_4 = \bar{A} \cdot B \cdot \bar{C} \cdot \bar{D}$	$M_4 = \bar{A} + B + \bar{C} + \bar{D}$
0 1 0 1	$m_5 = \bar{A} \cdot B \cdot \bar{C} \cdot D$	$M_5 = \bar{A} + B + \bar{C} + D$
0 1 1 0	$m_6 = \bar{A} \cdot B \cdot C \cdot \bar{D}$	$M_6 = \bar{A} + B + C + \bar{D}$
0 1 1 1	$m_7 = \bar{A} \cdot B \cdot C \cdot D$	$M_7 = \bar{A} + B + C + D$
1 0 0 0	$m_8 = A \cdot \bar{B} \cdot \bar{C} \cdot \bar{D}$	$M_8 = A + \bar{B} + \bar{C} + \bar{D}$
1 0 0 1	$m_9 = A \cdot \bar{B} \cdot \bar{C} \cdot D$	$M_9 = A + \bar{B} + \bar{C} + D$
1 0 1 0	$m_{10} = A \cdot \bar{B} \cdot C \cdot \bar{D}$	$M_{10} = A + \bar{B} + C + \bar{D}$
1 0 1 1	$m_{11} = A \cdot \bar{B} \cdot C \cdot D$	$M_{11} = A + \bar{B} + C + D$
1 1 0 0	$m_{12} = A \cdot B \cdot \bar{C} \cdot \bar{D}$	$M_{12} = A + B + \bar{C} + \bar{D}$
1 1 0 1	$m_{13} = A \cdot B \cdot \bar{C} \cdot D$	$M_{13} = A + B + \bar{C} + D$
1 1 1 0	$m_{14} = A \cdot B \cdot C \cdot \bar{D}$	$M_{14} = A + B + C + \bar{D}$
1 1 1 1	$m_{15} = A \cdot B \cdot C \cdot D$	$M_{15} = A + B + C + D$

Respectant les notations utilisées dans le tableau, on constate que le complément d'un max-terme est un min-terme, et inversement, on pose :

$$\bar{m}_i = M_{2^n-1-i}$$

où :

- n est le nombre de variables ($n = 4$ dans notre tableau),
- i est le rang de m et de M,
- 2^n-1 est la plus grande valeur de i qui varie entre 0 et 2^n-1.

On démontre que toutes les fonctions logiques peuvent être transformées en :

- un produit de max-termes, c'est-à-dire un produit de sommes logiques,
- une somme de min-termes, c'est-à-dire une somme de produits logiques.

7
Le code ASCII

1
L'ASCII

Comme nous le savons, 8 bits permettent de coder 2^8=256 valeurs différentes. La table ASCII fait correspondre à chaque valeur un caractère ou un symbole. Elle permet donc de passer d'un codage par bits - compréhensible par la machine informatique - à un codage par symboles « humains » que nous serons capables d'interpréter.

Le code ASCII initial n'utilisait que 7 bits, il n'autorisait donc que 128 caractères et symboles et est encore utilisé, entre autres, par le minitel. Le code ASCII le plus utilisé aujourd'hui est l'ASCII étendu (codage sur 8 bits).

Le dictionnaire de l'informatique (DUNOD, 1992) donne la définition suivante : « ASCII Sigle de American Standard Code for Information Interchange. se prononce « aski ». Schéma de codage qui attribue des valeurs numériques aux lettres, chiffres, marques de ponctuation et autres caractères. En normalisant le codage, ASCII permet l'échange d'informations entre ordinateurs. ASCII fournit 256 codes, divisés en deux jeux de 128 chacun, l'un dit standard, l'autre étendu. ces jeux représentent toutes les combinaisons possibles de 7 ou 8 bits, ce dernier cas correspondant à l'octet. Le jeu ASCII standard utilise 7 bits pour chaque code, allant de 0 à 127 (codes hexa #00h à #7Fh). Le jeu dit étendu utilise 8 bits pour chaque code, allant de 128 à 255 (codes hexa #80h à #FFh). Dans le jeu standard ASCII, les premiers 32 caractères sont assignés à des codes de contrôle pour la communication et pour l'impression

(en particulier les caractères non imprimables qui représentent un saut de ligne, un saut de feuille, une tabulation, etc.) qui servent dans les transferts d'information entre ordinateurs ou entre ordinateurs et imprimantes. Les 96 caractères restants codent les signes de ponctuation, chiffres de 0 à 9 et les lettres majuscules et minuscules de l'alphabet romain. Les codes du jeu ASCII étendu, qui vont de 128 à 255, sont attribués à des ensemble de caractères variés par les divers constructeurs et développeurs de logiciels. Ces codes ne sont pas aussi interchangeables entre programmes et ordinateurs que ne le sont caractères du jeu standard. IBM, par exemple, emploie un ensemble de caractères ASCII étendus, appelé jeu IBM étendu, pour ses micro-ordinateurs personnels. Apple utilise un jeu analogue, mais différent, pour les caractères utilisables sur ses ordinateurs Macintosh. Ainsi, là où le jeu ASCII standard est universel pour les matériels et les logiciels de la micro-informatique, les caractères du jeu ASCII étendu ne peuvent être correctement interprétés que si le programme, l'ordinateur ou l'imprimante sont adaptés à cette tâche.

La consultation du dictionnaire de l'informatique (DUNOD, 1992) est conseillée à ceux qui s'intéressent à l'informatique, quel que soit leur niveau. En effet, il s'agit d'un ouvrage de référence incontournable.

2
Les codes ASCII

2.1
Les caractères de contrôle en ASCII standard

Caractères codés	Codes ASCII (en décimal)	Codes ASCII (en hexadécimal)
Caractère nul	0	#00h
SOH (Start Of Heading)	1	#01h
STX (Start of TeXt)	2	#02h
ETX (End of TeXt)	3	#03h
EOT (End Of Transmission)	4	#04h
ENQ (ENQuiry)	5	#05h

Caractères codés	Codes ASCII (en décimal)	Codes ASCII (en hexadécimal)
ACK (ACKnowledge)	6	#06h
BEL (BELL)	7	#07h
BS (BackSpace)	8	#08h
HT (Horizontal Tab)	9	#09h
LF (Line Feed)	10	#0Ah
VT (Vertical Tab)	11	#0Bh
FF (FormFeed)	12	#0Ch
CR (Carriage Return)	13	#0Dh
SO (Shift Out)	14	#0Eh
SI (Shift In)	15	#0Fh
DLE (Data Link Escape)	16	#10h
DC1 (Device Control #1)	17	#11h
DC2 (Device Control #2)	18	#12h
DC3 (Device Control #3)	19	#13h
DC4 (Device Control #4)	20	#14h
NAK (Negative AcKnowledge)	21	#15h
SYN (SYNchronous idle)	22	#16h
ETB (End Transmission Block)	23	#17h
CAN (CANcel)	24	#18h
EM (End of Medium)	25	#19h
SUB (SUBstitute)	26	#1Ah
ESC (ESCape)	27	#1Bh
FS (File Separator)	28	#1Ch
GS (Group Separator)	29	#1Dh
RS (Record Separator)	30	#1Eh
US (Unit Separator)	31	#1Fh
« espace »	32	#20h

Rappelons que ces codes sont ceux utilisés par les compatibles IBM PC. Une autre table ASCII est à utiliser avec la HP-48.

2.2

Caractères et symboles en ASCII standard

Les codes ASCII de la table standard (7 bits) s'étendent de #00h à #7Fh. Les 32 premiers codes (#00h à #20h) sont des caractères de contrôle. Les

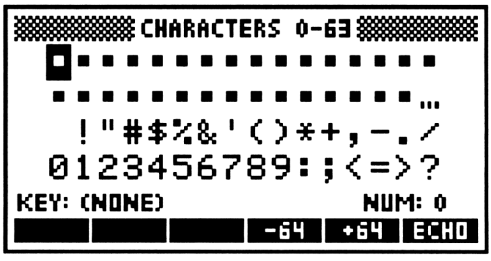
suivants, de #21h à #7Fh, correspondent aux symboles aux caractères et aux symboles classiques. Les significations des codes ASCII #21h à #7Fh sont communes à tous les codes ASCII.

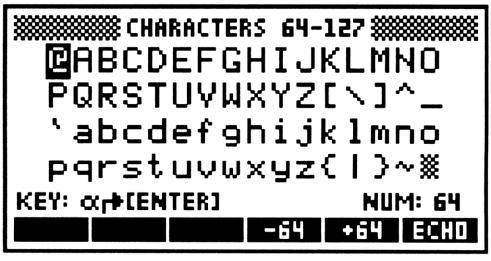
Caractères codés	Codes ASCII (base 10)	Codes ASCII (base 16)	Caractères codés	Codes ASCII (base 10)	Codes ASCII (base 16)
!	33	#21h	E	69	#45h
"	34	#22h	F	70	#46h
#	35	#23h	G	71	#47h
$	36	#24h	H	72	#48h
%	37	#25h	I	73	#49h
&	38	#26h	J	74	#4Ah
'	39	#27h	K	75	#4Bh
(	40	#28h	L	76	#4Ch
)	41	#29h	M	77	#4Dh
*	42	#2Ah	N	78	#4Eh
+	43	#2Bh	O	79	#4Fh
,	44	#2Ch	P	80	#50h
-	45	#2Dh	Q	81	#51h
.	46	#2Eh	R	82	#52h
/	47	#2Fh	S	83	#53h
0	48	#30h	T	84	#54h
1	49	#31h	U	85	#55h
2	50	#32h	V	86	#56h
3	51	#33h	W	87	#57h
4	52	#34h	X	88	#58h
5	53	#35h	Y	89	#59h
6	54	#36h	Z	90	#5Ah
7	55	#37h	[	91	#5Bh
8	56	#38h	\	92	#5Ch
9	57	#39h	]	93	#5Dh
:	58	#3Ah	^	94	#5Eh
;	59	#3Bh	_	95	#5Fh
<	60	#3Ch	`	96	#60h
=	61	#3Dh	a	97	#61h
>	62	#3Eh	b	98	#62h
?	63	#3Fh	c	99	#63h
@	64	#40h	d	100	#64h
A	65	#41h	e	101	#65h
B	66	#42h	f	102	#66h
C	67	#43h	g	103	#67h
D	68	#44h	h	104	#68h

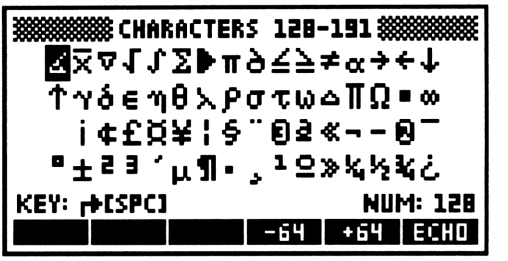
Caractères codés	Codes ASCII (base 10)	Codes ASCII (base 16)	Caractères codés	Codes ASCII (base 10)	Codes ASCII (base 16)
i	105	#69h	u	117	#75h
j	106	#6Ah	v	118	#76h
k	107	#6Bh	w	119	#77h
l	108	#6Ch	x	120	#78h
m	109	#6Dh	y	121	#79h
n	110	#6Eh	z	122	#7Ah
o	111	#6Fh	{	123	#7Bh
p	112	#70h	|	124	#7Ch
q	113	#71h	}	125	#7Dh
r	114	#72h	~	126	#7Eh
s	115	#73h	*DEL*	127	#7Fh
t	116	#74h			

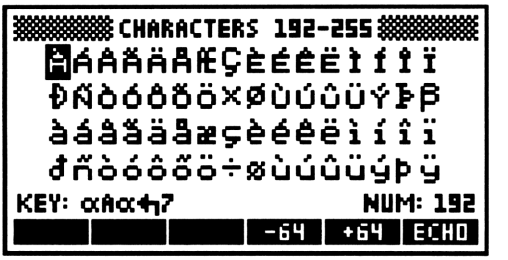
2.3
L'ASCII utilisé par la HP-48

En tapant [I→] [PRG] (HP-48 G/GX), vous activez les écrans de sélection CHARS qui présentent tous les caractères disponibles sur HP-48.









En appuyant sur les touches correspondant aux articles *+64* et *-64* du menu, vous ferez défiler 4 écrans présentant chacun 64 caractères. Vous pouvez choisir un caractère en le sélectionnant à l'aide des touches de déplacement. La combinaison de touches donnant accès au caractère sélectionné est affiché en bas à gauche alors que le code ASCII correspondant (en décimal) l'est en bas à droite. En sélectionnant l'article *ECHO* du menu, vous envoyez le caractère sélectionné dans la ligne de commande de la pile.

Les écrans « CHARS » présentent donc tous les caractères et symboles utilisables sur HP-48. C'est aux codes ASCII indiqués sur ces écrans qu'il vous faudra vous référer, notamment pendant la création d'objets de type caractère, chaîne de caractères, nom global ou local, etc. Attention ! Ce code ASCII est présenté en décimal et devra être converti en hexadécimal pour certaines de ces utilisations, lors d'une création d'objets en particulier.

Comparez le code ASCII de la HP-48 (voir les copies d'écrans ci-dessus) au code ASCII utilisé par les compatibles IBM PC. Notez les différences (sur la partie ASCII étendu) et les points communs (sur la partie ASCII standard).

8

Introduction au microprocesseur

1 Introduction

1.1 Données traitantes et données traitées

Un microprocesseur est un circuit numérique complexe capable de traiter des données en fonction des instructions de traitement qui lui sont fournies. Nous avons dit et répété qu'au niveau des circuits d'électronique numérique toutes les informations sont sous forme de 0 et de 1. Ainsi, un texte, une image ou un programme ne sont manipulés que sous forme binaire. Le microprocesseur ne manipule que deux types de données :

- les données traitantes, appelées couramment « programme », sont décodées par le microprocesseur et déterminent les traitements effectués,
- les données traitées font l'objet du traitement commandé par les données traitantes.

Le microprocesseur ne traite que des bits, il sait aller les chercher en mémoire, les y remettre, effectuer des opérations logiques, des décalages... C'est à peu près tout mais Il nous faudra pourtant nous débrouiller avec cela pour programmer en assembleur !

1.2
Algorithme et programme

Les notions d'algorithme et de programme sont liées. Un algorithme est la décomposition d'une tâche en primitives (actions élémentaires indivisibles) d'un exécutant donné. Un programme n'est autre que la spécialisation d'un algorithme à une machine donnée.

Par machine, il faut ici comprendre «machine virtuelle». En effet, les machines que vous utilisez lorsque vous programmez en RPL sur HP-48 et en langage d'assemblage (langage assembleur), toujours sur HP-48, ne sont pas les mêmes ! La HP-48 offre, selon le mode d'utilisation, une interface et des fonctions différentes, la machine virtuelle que vous utilisez est donc différente ! Une machine virtuelle est donc qu'une sur-couche logicielle déterminant nos rapports avec la machine physique.

Un algorithme étant une suite de primitives, remarquons que la notion de primitive est fonction de la machine virtuelle que nous utilisons. En RPL, on peut considérer le calcul d'une racine carrée ou le tracé d'une représentation graphique comme une primitive. Le RPL est un langage dit de « haut niveau », il s'éloigne de la machine physique qui est complètement masquée par les facilités offertes par les instructions prêtes à l'emploi du langage RPL.

Au contraire, le langage d'assemblage (langage assembleur) nous permet de dialoguer avec une machine virtuelle très proche de la machine physique. Autrement dit, nous pouvons faire faire à la machine tout ce qui lui est possible car nous intervenons directement au niveau de ses circuits électroniques. Les primitives du langage assembleur se résument à des ordres de lecture ou d'écriture en mémoire, à des opérations logiques, à des décalages de bits ou à des complémentations.

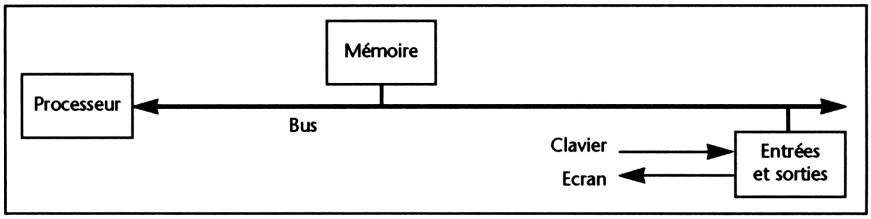
Tout est permis en assembleur, puisque toutes les ressources de la machine au plus bas niveau nous sont accessibles. L'assembleur est un langage dit de « bas niveau » puisqu'il nous confronte à une machine virtuelle très proche de la machine physique.

Avec un langage de haut niveau tel que le RPL, la machine offre des facilités à l'utilisateur, en particulier, elle propose des fonctions qui sont celles que manipule l'homme (fonctions scientifiques, etc.). Au contraire, un langage de bas niveau tel que l'assembleur oblige l'homme à s'adapter à la machine physique qui ne connaît que des bits.

1.3
Architecture de base

Un ordinateur, quelle que soit sa taille, dispose toujours des éléments suivants :

- un processeur chargé des traitements,
- une mémoire contenant les données traitantes et traitées,
- des dispositifs d'introduction des données et de restitution des résultats (circuits d'entrées/sorties).



L'unité centrale de traitement (UCT ou CPU « central processing unit ») est chargée de modifier les données traitées selon les directives des données traitantes (le programme). L'unité centrale de traitement est un microprocesseur lorsque le système informatique est un micro-ordinateur. La HP-48, qui n'est autre qu'un micro-ordinateur spécialisé, a un processeur SATURN pour UCT.

La mémoire morte (ROM « read only memory ») du système informatique contient des informations gravées une fois pour toutes en usine. Il est donc impossible de modifier le contenu de la mémoire morte. Celle-ci contient au moins un programme appelé « moniteur » et chargé de faire démarrer la machine. La ROM de la HP-48 occupe 512 ko et accueille une foule de programmes et de données (interpréteur RPL, EqLib, solveurs, fonctions scientifiques, etc.).

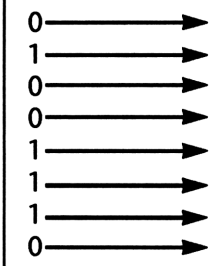
La mémoire vive (ou mémoire centrale ou RAM « random access memory ») est occupée par les données en cours de traitement et par les programmes introduits par l'utilisateur. La RAM des HP-48 S, SX et G a une capacité de 32 ko alors que celle de la HP-48 GX offre 128 ko. La mémoire vive est volatile. En effet, son contenu est perdu si l'alimentation électrique est interrompue. Les calculatrices, telles que la HP-48, disposent d'une mémoire vive qui doit remplir à la fois les fonctions de la mémoire vive classique et celles de la mémoire de masse. La mémoire de masse est normalement chargée de conserver - généralement sur disque - les don-

nées lorsque l'ordinateur est éteint, cependant, les calculatrices ne disposent pas de mémoire de masse au sens propre. La mémoire vive des calculatrices ne perd donc pas son contenu à l'extinction (une pile se charge de l'alimenter pour éviter les pertes de données).

Les dispositifs d'entrées/sorties permettent le dialogue avec la machine. Le clavier est un périphérique d'entrée permettant l'introduction des informations. L'écran, au contraire, est un périphérique de sortie permettant de visualiser les résultats renvoyés par la machine.

Les différents éléments que nous venons de citer sont reliés les uns aux autres par un bus. Imaginez une ligne d'autobus dont chaque station correspondrait à l'un des éléments cités. Il suffit alors de mettre une donnée « dans le bus » et d'indiquer la station où celle-ci doit descendre. Le bus des ordinateurs est une série de voies électriques parallèles, tous les éléments reliés par le bus sont identifiés par une adresse logique. Même si le nom de bus a été adopté par analogie au transport en commun, le bus est plutôt une autoroute à *n* fils électriques capable de véhiculer des mots de *n* bits.

Transmission en série et en parallèle de l'octet 01001110



La mémoire vive (la RAM) et la mémoire morte (la ROM) sont des ensembles de « cellules » élémentaires de mémoire. Toutes les cellules ont la même capacité (un quartet dans le cas de la HP-48, chaque cellule mémorise donc un chiffre hexadécimal). Chaque cellule est identifiée par son adresse qui n'est autre que son numéro d'ordre. Ainsi, les mémoires vives et mortes peuvent être représentées par une longue rue dont chaque maison accueille un chiffre hexadécimal (codé par un quartet). Chaque « maison » est identifiée par son numéro dans la rue, autrement dit, par son adresse. En pratique, l'adresse en mémoire est un nombre hexadécimal à cinq chiffres. L'adresse #00000h correspond au début de la mémoire.

2
Fonctionnement du microprocesseur

2.1
Exécution d'une instruction

Nous avons déjà fait la distinction entre données traitées et données traitantes. Les programmes sont des ensembles de données traitantes, c'est-à-dire des suites d'instructions. Chaque instruction est rangée dans une série de cellules de la mémoire, ces cellules étant bien sûr rangées dans l'ordre des adresses croissantes.

Le microprocesseur doit à chaque instant savoir quelle sera la prochaine instruction à exécuter. Le microprocesseur dispose de plusieurs petites mémoires d'une capacité de quelques quartets et appelées « registres ». L'un de ces registres contient en permanence l'adresse en mémoire de la prochaine instruction à exécuter. Ce registre porte le nom de compteur ordinal, pointeur d'instruction ou program counter (PC).

On décompose l'exécution d'une instruction en six étapes dont les quatre dernières s'exécutent sans intervention du programmeur :

- le compteur ordinal (PC) indique l'emplacement en mémoire de la prochaine instruction à exécuter,
- cette instruction voyage sur le bus et arrive sur l'un des registres du microprocesseur,
- l'instruction va être décodée par les circuits de décodage du microprocesseur,
- les circuits de décodage transmettent à l'unité de séquencement et de commandes un ensemble d'instructions pour le traitement. L'unité de séquencement et de commande voit son débit cadencé par une horloge elle-même tributaire de la fréquence d'un quartz (2 ou 4 MHz selon les versions de HP-48),
- les circuits de commande exécutent l'ordre en faisant appel à l'unité arithmétique et logique (UAL) qui est chargée d'exécuter les opérations arithmétiques et logiques,
- le résultat apparaît sur l'un des registres du microprocesseur et peut faire l'objet d'un autre traitement ou d'une copie en mémoire vive.

2.2
Cycles du microprocesseur

Une simple addition est, aux yeux du microprocesseur, une opération complexe qui peut être décomposée en opérations élémentaires. L'exécution de l'opération complexe occupera le microprocesseur pendant un cycle d'instruction alors que l'exécution de chaque opération élémentaire nécessitera un microcycle. Un microcycle, aussi appelé état, correspond à une impulsion du quartz de l'horloge. Ainsi, dans le cas d'une HP-48G dont le quartz a une fréquence de 4 MHz, le microprocesseur reçoit 4 millions d'impulsions par seconde. Un microcycle dure alors un quart de millionième de seconde puisque la période ou durée d'un état est l'inverse de la fréquence.

Retenez donc qu'un microprocesseur cadencé à 4 MHz n'exécute pas 4 millions d'instructions du langage assembleur en une seconde. Il exécute en une seconde 4 millions d'instructions encore plus élémentaires que celles utilisées en assembleur. L'exécution d'une instructions du langage assembleur exige plusieurs microcycles.

2.3
Registres et drapeaux

Comme nous l'avons déjà dit, il y a au sein d'un microprocesseur plusieurs petites mémoires appelées « registres ». L'un de ces registres est le registre d'état et accueille plusieurs drapeaux (flags) qu'il ne faut pas confondre avec les drapeaux RPL. Un drapeau est un bit donné du registre d'état. Par exemple, considérons le drapeau de retenue. Le bit correspondant à cette retenue vaudra 0 ou 1 si une retenue doit ou ne doit pas être reportée sur la suite du calcul. On dit que la retenue est « armée » ou « désarmée ».

Les bits des registres ST et HS (HST), le bit de retenue (ou Carry) et bien d'autres bits, fonctionnent comme des drapeaux.

9

Introduction au SATURN

1 Le SATURN et ses rôles

Le SATURN est un microprocesseur dit « 4 bits » conçu spécialement par Hewlett-Packard pour ses calculateurs de poche (en fait, le SATURN est plus un « 64 bits » qu'un « 4 bits »...). Les différentes générations du SATURN ont équipé les HP-71 (version 1LF2 du SATURN), puis les HP-28 (version 1LK7 du SATURN) et enfin les HP-48. Notez que le SATURN est un microprocesseur très particulier à de nombreux points de vue.

■ L'horloge du SATURN

Comme tous les microprocesseurs, le SATURN comporte une unité de séquencement et de commande chargée d'organiser le travail du microprocesseur. Cette unité permet au microprocesseur de passer d'un état à l'état suivant. Tous les états ont la même durée ; ce sont des états électriques, ou plutôt, logiques. L'unité de séquencement et de commande déclenche le passage à l'état suivant à chaque fois qu'elle reçoit une impulsion en provenance de l'horloge chargée de cadencer le travail du microprocesseur.

Cette horloge envoie ses impulsions avec une fréquence déterminée par un quartz. La fréquence d'horloge des HP-48 S et SX est de ≈2 MHz. Le SATURN a alors deux millions d'états différents par seconde et chaque état

dure 0,5 µs. Dans le cas des HP-48 G et GX, la fréquence passe à ≈4 MHz. Le SATURN a ici quatre millions d'états différents par seconde, chacun durant 0,25 µs.

■ Le SATURN décode les instructions

Toutes les instructions données au microprocesseur SATURN doivent d'abord être décodées par des circuits spécialisés. Toute exécution d'une commande passe donc nécessairement par une phase de décodage.

■ Le SATURN échange des données

L'une des fonctions les plus importantes d'un microprocesseur est l'échange de données. Le SATURN échange des données avec les mémoires vive et morte ainsi qu'avec les dispositifs dits « périphériques ». Le clavier ou l'écran sont à considérer comme des périphériques. Le SATURN pourra, par exemple, aller chercher un objet de type chaîne à une adresse donnée de la mémoire vive et contrôler son affichage dans la pile (visualisée sur l'écran).

■ Le SATURN dispose d'une unité arithmétique et logique

L'unité arithmétique et logique du SATURN (UAL) prend en charge les opérations logiques (complémentation, OU logique, ET logique, etc.) ainsi que diverses opérations arithmétiques (addition, etc.). En organisant convenablement ces différentes fonctions, on reconstitue des algorithmes de calcul exploitables, par exemple, celui calculant la racine carrée.

■ Le SATURN réagit aux « interruptions » pendant un processus

Pendant l'exécution d'un programme, d'un calcul ou de tout autre processus, le SATURN doit pouvoir réagir à divers signaux en provenance de l'extérieur, par exemple, une pression sur une touche. Une interruption provoque l'arrêt temporaire du travail en cours pour traiter l'événement qui a déclenché l'interruption, on achève le travail interrompu dès la fin du traitement de l'interruption. Notez qu'il est possible d'interdire les interruptions afin d'accélérer un programme écrit en assembleur.

Remarquez que le SATURN prend aussi en charge l'organisation de la mémoire vive. La mémoire peut être gérée en « temps réel » (comme aiment le dire les informaticiens industriels !).

2

Généralités et spécificités du SATURN

Le SATURN a, comme nous allons maintenant le voir, quelques particularités. Vous devrez sans cesse avoir à l'esprit les quelques points que nous allons préciser.

■ Le SATURN calcule en hexadécimal et en DCB

Tous les microprocesseurs calculent au moins en hexadécimal. Ces calculs en base 16 n'étant qu'une représentation commode des bits directement manipulés par l'électronique numérique du microprocesseur. Comme nous l'avons déjà dit, un chiffre hexadécimal représente un quartet, autrement dit, une suite de quatre bits.

Nous calculons habituellement en base 10. Nous pouvons demander au SATURN de faire de même, et ce, pour simplifier certains programmes. Le SATURN utilise le DCB que nous avons déjà décrit pour représenter les nombres en base 10.

■ Les types de registre du SATURN

Rappelons qu'un registre est une petite mémoire du microprocesseur. Un registre a une capacité de quelques bits seulement. On peut établir trois grandes catégories de registres :

- les registres dits « pointeurs »,
- les registres de travail (aussi appelés registres de calcul),
- les drapeaux.

Les registres pointeurs contiennent une adresse en mémoire. C'est à cette adresse que le microprocesseur pourra aller chercher une donnée, il s'agira dans ce cas d'une lecture en mémoire. Il peut aussi copier un résultat à cette adresse, on parle alors d'écriture en mémoire.

Le contenu d'un registre de travail peut faire l'objet de divers traitements. Le SATURN dispose d'instructions particulières prévues à cet effet.

Le contenu des drapeaux (flags) varie selon le contexte. Un drapeau pourra donc être armé ou désarmé (retenue, bits de statut, etc.).

■ L'unité d'information du SATURN

Avec le SATURN, la plus petite quantité d'information que nous puissions manipuler directement est le quartet (une série de 4 bits). Un quartet permet le codage de 2^4=16 valeurs différentes. Les drapeaux constituant un cas à part, chaque registre est divisible en un nombre entier de quartets.

■ Les adresses

L'adressage est la désignation par le microprocesseur d'une adresse en mémoire (emplacement en mémoire). Les adresses que peut désigner le SATURN sont codées sur 20 bits, autrement dit, sur 5 quartets. Un quartet représentant un nombre hexadécimal, les adresses « adressables » sont comprises entre 00000_{16} et $FFFFF_{16}$.

Reprenant la notation des objets de type entier système, 00000_{16} devient #00000h alors que $FFFFF_{16}$ sera noté #FFFFFh où h indique qu'il s'agit d'un nombre hexadécimal.

L'espace adressable du SATURN :

$$FFFFF_{16} = 2_{10}{}^{20} = 1048576_{10} \text{ adresses}$$

Le SATURN peut donc désigner 1 048 576 cellules de mémoire distinctes. Chacune contient un quartet. Le SATURN peut donc adresser une mémoire de 1 048 576 quartets, c'est-à-dire 524 288 octets (512 ko).

Quelques astuces permettent de dépasser cette limitation et de travailler avec plus de 512 ko (masquage temporaire de zones de la mémoire).

Très important ! L'unité d'information étant le quartet, à chaque adresse en mémoire correspond un unique quartet. Cependant, un objet de type quelconque occupe généralement de nombreux quartets, on considère donc que l'adresse de l'objet est celle de son premier quartet.

Les objets sont stockés en mémoire sous forme de suites de quartets, on parle alors d'enregistrement en séquence. En effet, l'accès à l'objet - qui peut être un programme - est séquentiel, car, pour le parcourir intégralement, il faut lire chacun de ses quartets un par un du premier au dernier. On commence donc la lecture au niveau du premier quartet ; l'adresse de ce dernier peut donc être considérée comme l'adresse de l'objet.

Par suite, l'exécution d'un programme n'est autre que la lecture par le SATURN de tous les quartets du programme en partant du premier.

■ L'inversion des quartets

Il va maintenant être question de l'une des particularités les plus déconcertantes du SATURN. Ce dernier nous oblige à inverser les quartets pendant les opérations de lecture et d'écriture en mémoire. Rappelons qu'une lecture est un transfert de quartets depuis la mémoire vers un registre alors qu'une écriture est un transfert depuis un registre vers la mémoire vive.

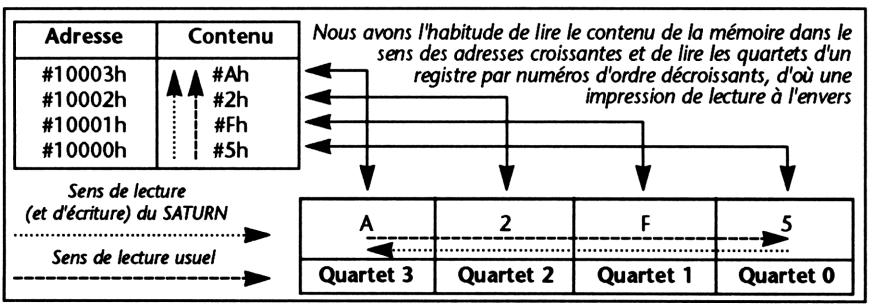
La capacité d'un registre du SATURN est un nombre entier de quartets. Les plus gros registres du SATURN accueillent jusqu'à 16 quartets, c'est-à-dire 64 bits (un Intel iAPX 386 se contente de 32 bits...). Certains registres spécialisés dans la désignation d'une adresse en mémoire, ont une capacité de 5 quartets, établissez vous-même un lien entre cette capacité et l'espace adressable du SATURN.

Chaque quartet d'un registre est numéroté de *0* à *n-1* (pour un registre de *n* quartets). Le quartet n°0 est le quartet de poids le plus faible (le plus à droite) alors que le quartet n°*n-1* est le quartet de poids le plus fort (le plus à gauche).

Nous avons l'habitude de lire et d'écrire de la gauche vers la droite. Le SATURN, lui, effectue ses transferts en commençant par les quartets de poids faible (quartet n°0 puis les suivants). Il « lit » et « écrit » donc de la droite vers la gauche puisque le premier quartet qu'il considère est le quartet de droite.

Soit un registre imaginaire ayant une capacité de 4 quartets. Ce registre contient #A2F5h (qui peut aussi être noté $A2F5_{16}$). Sachant qu'à chaque adresse en mémoire correspond un quartet mémorisé, si on copie le contenu de ce quartet à l'adresse imaginaire #10000h, on obtient :

- le quartet #5h à l'adresse #10000h,
- le quartet #Fh à l'adresse #10001h,
- le quartet #2h à l'adresse #10002h,
- le quartet #Ah à l'adresse #10003h.



Vous avez sans doute remarqué que les adresses sont représentées par cinq chiffres hexadécimaux, par exemple #2AEF1h (qui peut aussi être noté $2AEF1_{16}$). Le SATURN manipule des adresses codées sur 5 quartets, et donc, sur 20 bits. Dans le cadre de la programmation en assembleur, il est évidemment plus commode de représenter les adresses telles qu'elles seront finalement manipulées par le SATURN plutôt que de les représenter par des nombres décimaux auxquels nous sommes habitués.

3
Les registres du SATURN

3.1
Les familles de registres

Les registres, petites mémoires internes au microprocesseur SATURN, peuvent être rangés en plusieurs catégories. Tous les microprocesseurs disposent de registres de travail, de registres pointeurs et de drapeaux. Voyons maintenant ce que nous propose le SATURN.

■ Les registres de travail (ou registres de calcul)

Les registres A, B, C et D sont les quatre registres de travail. Ce sont les données qu'ils contiennent qui feront l'objet d'un traitement. Ces registres ont une capacité de 64 bits, c'est-à-dire 16 quartets. A et C peuvent servir à l'écriture ou à lecture en mémoire. Sans intervention du programmeur, le registre D contient une valeur représentant l'espace disponible en mémoire alors que B contient l'adresse de la première portion de mémoire disponible. A, B, C et D sont aussi appelés registres de calcul.

■ Les registres de sauvegarde

Les registres R0, R1, R2, R3 et R4 sont les cinq registres de sauvegarde. Ils peuvent accueillir 64 bits (16 quartets). Cette capacité est justifiée par le

fait qu'ils sont voués à la sauvegarde temporaire des registres de travail A, B, C et D. Notez que les registres de sauvegarde ont le même rôle que la mémoire vive, ils sont cependant plus intéressants puisqu'une écriture (sauvegarde) dans un registre de sauvegarde est beaucoup plus rapide qu'une écriture en mémoire vive. De même la copie du contenu d'un registre de sauvegarde vers un registre de travail est beaucoup plus rapide qu'un accès à la mémoire vive.

■ Les registres pointeurs

Un pointeur désigne un emplacement en mémoire vive. Les registres D0 et D1 sont les deux registres pointeurs de données du SATURN. Celui-ci traite des adresses codées sur 5 quartets, D0 et D1 ont donc chacun une capacité de 5 quartets (20 bits). Le SATURN va lire ou écrire des données en mémoire vive à l'adresse spécifiée par l'un des registres D0 ou D1. C'est l'un des registres de travail qui recevra les données lues à l'adresse spécifiée ou qui copiera son contenu à l'adresse spécifiée.

Sans intervention du programmeur, D1 contient l'adresse du niveau 1 de la pile (pile utilisateur).

■ Le compteur ordinal

Le registre PC (Program Counter) correspond au compteur ordinal. L'adresse de la prochaine instruction à exécuter se trouve dans le registre PC. Destiné à accueillir une adresse codée sur 5 quartets, le registre PC a bien entendu une capacité de 5 quartets (20 bits). Le passage à l'instruction exécutable suivante implique une modification du contenu du registre PC. Le programmeur peut affecter une valeur au registre PC afin d'effectuer un saut.

■ Les entrées et sorties

Le microprocesseur échange des données avec l'extérieur via des dispositifs périphériques tels que le clavier. Le SATURN dispose de deux registres chargés de ces échanges. Le registre IN (input) est le registre d'entrée et accueille 4 quartets (16 bits). Le registre OUT (output) est le registre de sortie et a une capacité de 3 quartets (12 bits).

Pour tester, à l'aide d'un programme écrit en assembleur, l'appui sur une touche, on compare un « masque de touche » au signal envoyé par le clavier et recueilli dans le registre IN.

■ RSTK, la pile des retours

Une pile est une structure de données de type « LIFO » (Last In First Out, le dernier entré est le premier sorti). Cette pile se comporte comme la pile utilisateur RPL. Lors d'un appel de sous-programme, la pile RSTK mémorise l'adresse à laquelle il faudra reprendre l'exécution du programme principal lorsque l'exécution du sous-programme sera achevée.

La pile RSTK peut être assimilée à une série de 8 registres de 5 quartets chacun car elle comporte 8 niveaux contenant chacun une adresse codée sur 5 quartets. Il est possible d'emboîter 6 niveaux de sous-programmes si l'on considère que 2 niveaux de la pile RSTK doivent être réservés à la gestion des interruptions.

■ Les drapeaux

Les drapeaux sont des bits spéciaux qui peuvent, comme tous les bits, valoir 0 ou 1. On dit d'un drapeau qu'il est « armé » lorsqu'il vaut 1, le désarmer revient à le faire repasser à 0.

Le bit de retenue, ou bit CARRY, est un drapeau isolé du SATURN (hors registre). Le bit CARRY passe à 1 dès qu'une retenue doit être considérée pendant un calcul arithmétique. Par ailleurs, le bit CARRY passe à 1 lorsqu'un test renvoie « vrai » (« true ») pour résultat. Lorsqu'un test renvoie « faux » (« false »), le bit CARRY vaut 0 (drapeau désarmé).

Le registre ST (STatus bits) comporte 4 quartets (16 bits numérotés de 0 à 15). Les bits 0 à 11 sont utilisables par l'utilisateur alors que les bits 12 à 15 sont gérés par et pour le SATURN.

Le registre HST (Hardware STatus bits, ou HS) ne comporte qu'un quartet (4 bits). Il est déconseillé au novice de manipuler ces bits car le SATURN s'en sert pour traiter des événements hardware particuliers. Les quatre bits du registre HST :

- XM (eXternal Module missing) est le bit 0 du registre HST,
- SB (Sticky bit) est le bit 1 du registre HST,
- SR (Service Request) est le bit 2 du registre HST,
- MP (Module Pulled) est le bit 3 du registre HST.

Le seul bit de HST qui puisse être armé par l'utilisateur est XM à l'aide de l'instruction assembleur correspondant au mnémonique `RTNSXM`. Le débutant devra se contenter du bit SB qui est le seul exploitable par un programmeur peu aguerri. Le bit SB est armé lorsqu'un décalage à droite

(sur un registre de travail bien sûr !) implique l'éjection sur la droite d'un bit valant 1. Ce bit est donc armé lorsqu'il y a perte d'information significative. Notez qu'un 0 en tant que MSB - bit le plus significatif - n'est pas une information significative.

■ Le pointeur de champ

Il est possible de ne travailler que sur une portion d'un registre de travail, cette portion de registre est appelée « champ ». Un registre spécial, le pointeur de champ, permet de désigner la portion d'un registre de travail qui doit faire l'objet d'un traitement.

P est le registre pointeur de champ. Il peut contenir un quartet (4 bits) et peut donc coder 16 valeurs différentes ce qui lui permet de désigner l'un des 16 quartets de l'un des registres de travail A, B, C ou D. On appelle champ P le quartet n°P du registre de travail considéré. On appelle champ WP la portion d'un registre de travail allant du quartet n°0 au quartet n°P.

Le registre P peut prendre 16 valeurs et pourra être utilisé comme compteur de boucle si sa fonction de pointeur de champ est temporairement laissée de côté et si 16 valeurs sont suffisantes. Cette utilisation de P est particulièrement intéressante si l'on considère que les opérations de lecture et d'écriture dans le registre P sont particulièrement rapides.

■ En résumé

Noms des registres	Types	Capacités
A, B, C, D	Registres de travail	16 quartets (64 bits)
R0, R1, R2, R3, R4	Registres de sauvegarde	16 quartets (64 bits)
D0, D1	Registres pointeurs	5 quartets (20 bits)
PC	Compteur ordinal	5 quartets (20 bits)
IN	Entrées	4 quartets (16 bits)
OUT	Sorties	3 quartets (12 bits)
RSTK	Pile des retours	8x5 quartets (8x20 bits)
CARRY	Drapeau	1 bit
ST	Drapeaux	4 quartets (16 bits)
HST (XM, SB, SR, MP)	Drapeaux	1 quartet (4 bits)
P	Pointeur de champ	1 quartet (4 bits)
WP	*Champ*	*P quartets (Px4 bits)*

WP apparaît dans notre tableau de synthèse alors qu'il n'est qu'une portion d'un registre de travail déterminée par le contenu du registre P.

Un registre de 16 quartets est un registre de travail ou un registre de sauvegarde (d'un registre de travail). Un registre de 5 quartets contient une adresse en mémoire. Un drapeau correspond toujours à un unique bit.

3.2
Les champs des registres de calcul

Seul le contenu d'un registre de travail (A, B, C ou D) peut faire l'objet d'un traitement. Il est cependant possible de limiter ce traitement à une portion d'un registre de travail. Cette portion de registre est un champ. Retenez que seuls les registres de travail sont divisibles en champs.

Certains champs ont été prévus pendant la conception du SATURN, d'autres pourront être créés par le programmeur à l'aide du registre P. Le tableau ci-dessous présente les champs fixes W, S, M, X, XS, A et B d'un registre de travail (16 quartets, 64 bits).

Division en champs des 16 quartets d'un registre de travail (A, B, C ou D)

Quartet 15	Quartet 14	Quartet 13	Quartet 12	Quartet 11	Quartet 10	Quartet 9	Quartet 8	Quartet 7	Quartet 6	Quartet 5	Quartet 4	Quartet 3	Quartet 2	Quartet 1	Quartet 0
Champ **W**															
Champ **S**	Champ **M**												Champ **X**		
											Champ **A**				
													Champ **XS**	Champ **B**	

Vous devrez avoir à l'esprit ce tableau car de nombreuses instructions du SATURN ne s'appliquent qu'à un champ d'un registre de calcul.

■ Le champ W

Le champ W (Word) n'est autre que le registre de travail pris intégralement (64 bits). Le champ W s'étend donc du quartet 0 au quartet 15.

■ Le champ S

Le champ S (mantissa Sign, signe de la mantisse) ne comprend que le quartet de poids le plus fort qu'est le quartet 15.

■ Le champ M

Le champ M (Mantissa, mantisse) comprend 12 quartets et s'étend du quartet 3 au quartet 14.

■ Le champ X

Le champ X (eXponent) comprend 3 quartets et s'étend du quartet 0 au quartet 2.

■ Le champ XS

Le champ XS (eXponent Sign) se résume au quartet 2. Le champ XS est une portion du champ X dont il reprend le quartet de poids le plus fort.

■ Le champ B

Le champ B (Byte) correspond aux deux quartets de plus faible poids que sont les quartets 0 et 1 d'un registre de calcul. Rappelons qu'un « byte » n'est autre qu'un octet. Les quartets 0 et 1 représentent l'octet de poids faible du registre de calcul. On utilisera donc le champ B lorsque les manipulations de bits doivent se limiter à un octet.

■ Le champ A

Le champ A (Address) comprend 5 quartets et s'étend du quartet 0 au quartet 4, il inclut donc les 5 quartets de poids le plus faible. La capacité du champ A correspond à la taille d'une adresse en mémoire toujours codée sur 5 quartets (20 bits).

■ Le champ S+M+X

La HP-71 utilisait l'une des premières versions du SATURN. Les champs S+M+X (16 quartets au total) évoquent la représentation des nombres

réels sur HP-71. Le quartet du champ S est consacré au signe de la mantisse elle-même représentée par les 12 quartets (48 bits) du champ M. L'exposant est codé sur les 3 quartets du champ X. Le signe de l'exposant est codé sur le champ XS incorporé au champ X.

■ En résumé

Le tableau ci-dessous vous permettra de mémoriser facilement les champs fixes des registres de travail.

Champs	Contenu prévu initialement	Capacité du champ	Quartets concernés
W	Tout le registre !	16 quartets (64 bits)	0 à 15
S	Signe de la mantisse	1 quartet (4 bits)	15
M	Mantisse	12 quartets (48 bits)	3 à 14
X	Exposant	3 quartets (12 bits)	0 à 2
XS	Signe de l'exposant	1 quartet (4 bits)	2
S + M + X (dont XS)	*Nombre réel*	*16 quartets (64 bits)*	*0 à 15*
B	Octet	2 quartets (8 bits)	0 et 1
A	Adresse en mémoire	5 quartets (20 bits)	0 à 4

A retenir :

- les registres de travail sont divisibles en champs,
- le champ B comme Byte (octet) est le premier octet du registre,
- le champ A comme Adresse correspond aux 5 premiers quartets (les adresses en mémoire sont codées sur 5 quartets),
- le champ W comme Word (mot binaire manipulé par le SATURN) correspond au registre dans sa totalité,
- les champs S+M+X correspondent à la totalité du registre. Le champ XS est une portion du champ X,
- ne pas confondre les champs A et B avec les registres A et B.

■ Les champs définis par l'utilisateur

Le contenu du registre P est un nombre compris entre 0_{10} et 15_{10} que choisit le programmeur. Le champ P d'un registre de travail est son quartet n°*P*. Le champ WP d'un registre de travail comprend tous les quartets d'un registre de travail du quartet n°0 au quartet n°*P*. En définissant le contenu du registre P, l'utilisateur définit les champs P et WP des registres de calcul.

10 Configuration matérielle du SATURN

1 Fonctionnement du SATURN

Nous allons décrire différentes notions indispensables à la compréhension du fonctionnement du microprocesseur SATURN de la HP-48.

■ Le SATURN émet des adresses sur son bus

Un microprocesseur ne comprend que deux niveaux logiques correspondant à la présence ou à l'absence de courant électrique. Nous notons 0 et 1 ces deux niveaux logiques. Chaque emplacement en mémoire peut contenir un quartet et est identifié par son adresse codée par 5 quartets. Pour lire ou écrire une donnée en mémoire, le SATURN doit connaître l'adresse de la donnée. Le SATURN désigne l'emplacement choisi en mémoire en envoyant l'adresse de cet emplacement sur le bus (ici, bus d'adresses). Le bus est un ensemble de voies électriques parallèles permettant l'échange de données entre les différents éléments de la HP-48 (microprocesseur SATURN, mémoires, périphériques, etc.).

■ Les registres A et C permettent les échanges avec la mémoire

L'unité d'information du SATURN est le quartet, par ailleurs, à chaque adresse en mémoire correspond un quartet. Rappelons qu'un quartet est

une série de 4 bits pouvant être représentée par un nombre hexadécimal. Les registres A et C sont des registres de travail (ou registres de calcul) permettant des échanges avec la mémoire. Une information à copier en mémoire devra être placée dans l'un de ces deux registres. Les informations à copier en mémoire passent du SATURN à la mémoire en passant par le bus (ici, bus de données). Etant donné la capacité des registres A et C, il est possible d'échanger jusqu'à 16 quartets (64 bits) avec la mémoire en une seule opération.

■ Mémoire vive, mémoire morte et zones masquées de la mémoire

Pour désigner un quartet en mémoire, on place son adresse dans un registre pointeur de données (D0 ou D1) ; l'adresse voyage ensuite vers la mémoire en utilisant le bus (ici, bus d'adresses). Une HP-48 S/SX dispose de 256 ko de mémoire morte (ROM), de 32 ko de mémoire vive (RAM) « soudée » qui peut être complétée par 256 ko de mémoire sur cartes amovibles sur la HP-48 SX. Le SATURN doit donc ici pouvoir adresser 544 ko (256 ko + 32 ko + 256 ko).

Les registres pointeurs de données du SATURN peuvent contenir des adresses codées sur 20 bits.

$$2^{20} = 1\ 048\ 576 \text{ adresses}$$
$$1\ 048\ 576 \text{ quartets} \div 2 = 524\ 288 \text{ octets}$$
$$524\ 288 \div 2^{10} = 524\ 288 \div 1024 = 512 \text{ ko}$$

Les 20 bits des registres pointeurs de données permettent de distinguer 2^{20} = 1 048 576 adresses. A chaque adresse correspond un quartet en mémoire. La capacité maximale théorique de la mémoire est donc 1 048 576 quartets, c'est-à-dire 524 288 octets. On divise ce résultat par 2^{10} = 1024 pour le convertir en kilo octets. On obtient 512 ko. Les 20 bits des registres pointeurs de données peuvent donc désigner 512 ko en mémoire ce qui est différent des 544 ko nécessaires.

En fait, une partie de la mémoire morte (ROM) est cachée. On l'appelle ROM cachée et elle a une capacité de 32 ko (S/SX). En retirant ces 32 ko des 544 ko à adresser on obtient 512 ko ce qui correspond à l'espace adressable par le SATURN. La mémoire accessible (non masquée) est toujours limitée à 512 ko par les capacités d'adressage du SATURN.

Les données de la ROM cachée se trouvent aux mêmes adresses logiques que d'autres données « non cachées ». Les règles de priorités entre les mémoires permettent de faire cohabiter à une même adresse deux données distinctes. Lors d'un accès à une zone cachée de la mémoire, il sera

nécessaire de reconfigurer les modules de mémoire pour que le module à atteindre ne soit plus masqué.

■ L'unité arithmétique et logique

Le SATURN confie tous les traitements à son UAL (unité arithmétique et logique). L'UAL n'est active que lorsqu'un ordre lui est donné par l'unité de séquencement et de commande. Ce sont les registres de calcul A, B, C et D qui sont traités par l'UAL.

■ L'assembleur, une affaire de registres !

La programmation en assembleur se résume à des calculs arithmétiques et logiques à partir des registres de travail du microprocesseur. Les échanges de données ne se font qu'à travers des adresses logiques placées dans des registres spécialisés du microprocesseur.

■ L'unique bus du SATURN

Un microprocesseur classique comporte trois bus, un pour les adresses, un autre pour les données et un troisième pour les commandes. Le SATURN ne dispose que d'un seul bus qui est donc chargé de transporter les commandes, les données et les adresses. Le bus [0:3] (c'est son nom) de notre HP-48 est multiplexé puisque la nature de l'information qu'il transporte dépend de l'instant considéré.

■ Lecture en mémoire utilisant un bus multiplexé

Lorsque le SATURN doit lire une donnée en mémoire, voilà ce qui se passe au niveau du bus :

- le SATURN émet un signal LOAD DP (ou LOAD PC) sur le bus il signale ainsi son aptitude à la réception de données,
- le périphérique qui doit envoyer des données place l'adresse de l'information à émettre dans l'un de ses propres registres,
- le périphérique dont le contenu doit être lu (la mémoire par exemple) signale ensuite sa disponibilité,
- les données lues arrivent ensuite au SATURN via le bus.

Le bus [0:3] a donc servi à l'envoi d'une commande et d'une adresse puis à la réception d'une donnée.

2
La mémoire de la HP-48

2.1
Définitions

La mémoire vive est aussi appelée RAM (Random Access Memory, mémoire à accès aléatoire). Il est possible de lire et d'écrire des données en mémoire vive. Dans le cas de la HP-48, la mémoire vive tient aussi le rôle de la mémoire de masse puisqu'elle est chargée de conserver les données lorsque la HP-48 est éteinte.

La mémoire morte est aussi appelée ROM (Read Only Memory, mémoire lisible uniquement). Seule la lecture de la ROM est possible. Le contenu de la ROM a été figée en usine. Autant il est possible d'effacer volontairement ou de perdre accidentellement le contenu de la RAM, autant il est impossible d'effacer le contenu de la ROM.

Le tableau ci-dessous résume les capacités des mémoires pour les versions S, SX, G et GX de la HP-48.

	RAM standard	ROM standard	RAM maximale	ROM maximale
HP-48 G	32 ko	512 ko	32 ko	512 ko
HP-48 GX	128 ko	512 ko	4,256 Mo	4,64 Mo
HP-48 S	32 ko	256 ko	32 ko	256 ko
HP-48 SX	32 ko	256 ko	288 ko	512 ko

2.2
Structure de la mémoire

La HP-48 G dispose de 512 ko de mémoire morte et de 32 ko de mémoire vive (au total, 544 ko). La HP-48 GX dispose de 512 ko de mémoire morte et de 128 ko de mémoire vive extensible à 4,256 Mo. Nous avons

déjà dit que la taille des registres pointeurs de données du SATURN ne permettait pas l'adressage de plus de 512 ko, nous avions alors parlé de priorités entre les mémoires pour en masquer certaines. Un autre problème d'adressage est posé par les cartes de 2^n×128 ko de la HP-48 GX.

■ Diviser la mémoire en partitions de 128 ko sur HP-48 GX

Dans la mémoire des HP-48 G/GX, les données sont groupées en « banques de données » (« banks » que nous appellerions « partitions de mémoire » en bon français...). Le port 2 de la HP-48 GX accepte des cartes d'une capacité de 4 Mo. Pour être utilisable, cette carte est divisée en tranches de 128 ko, on crée ainsi 32 tranches de 128 ko numérotées de 0 à 31 (on a bien 128 × 32 = 4096 ko = 4 Mo). Sur le port 2, ces tranches de 128 ko définissent 32 partitions notées 2:0 à 2:31.

■ Accéder à la mémoire en port 2

Sur un port donné, une seule partition sera visible à un instant donné. Pour accéder à la partition *n* du port 2 il suffit de lire en mémoire l'information se trouvant à l'adresse #7F040h + 2.*n* en utilisant l'instruction `C=DAT0 B`. Ainsi, pour sélection la partition 3 de la carte en port 2, il suffit d'exécuter `D0=7F046` puis `C=DAT0 B`.

■ Accéder à la mémoire en port 1

Le port 1 de la HP-48 GX ne peut normalement recevoir que des cartes compatibles avec la HP-48 SX et donc limitées à 128 ko. Il existe cependant de #7F000h à #7F03Eh une table de sélection pour le port 1. Dans le cadre d'une utilisation normale seule l'utilisation de la partition 1:0 (première partition du port 1) est cohérente. On notera qu'il est possible d'accroître la mémoire vive de la HP-48 G en débloquant ses ports 1 et 2 inutilisés et en autorisant l'utilisation des partitions de ce port. Remarquez que la zone de sélection du port 1 s'étend de #7F000h à #7F03Eh et que la zone de sélection du port 2 commence juste après en #7F040h pour prendre fin en #7F07Eh.

■ Sélection d'une partition à l'aide du Bank Driver

Le Bank Driver des HP-48 G/GX sert à la sélection d'une banque de données (ou partition de la mémoire). Le Bank Driver occupe 2 ko (c'est-à-dire 4096 quartets) et a été configuré entre les adresses #7F000h et #80000h.

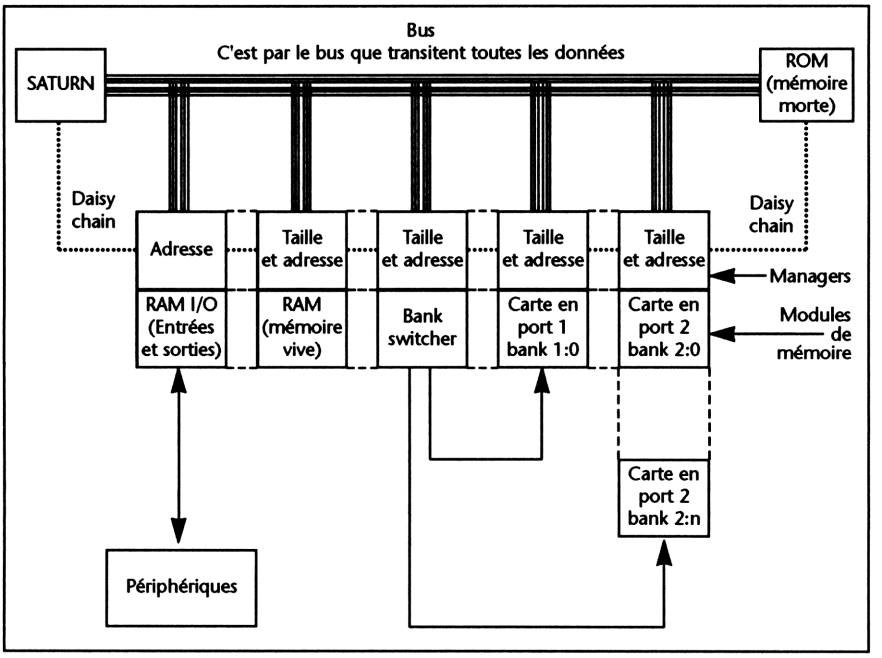
■ Adresses de la mémoire vive interne sur HP-48 GX

La mémoire vive interne de la HP-48 GX occupe les adresses #80000h à #C0000h. De #80000h à #C0000h on trouve $40000_{16} = 262144_{10}$ adresses pour 262144_{10} quartets, autrement dit, 128 ko.

2.3
Hiérarchie des mémoires

La HP-48 comporte plusieurs mémoires reliées au SATURN par le bus. Il faut savoir que chaque mémoire dispose de ses propres pointeurs, de ses propres compteurs ordinaux (PC) ainsi que de drapeaux spécifiques.

Le schéma ci-dessous résume l'architecture de la HP-48.



Le SATURN échange des données avec les autres composants de la HP-48 en utilisant le bus. A chaque module de mémoire est associé un gestionnaire (ou manager). Chaque gestionnaire se fonde sur l'adresse de départ et la taille du module qu'il gère pour savoir si les données en transit sur le

bus lui sont destinées. Lorsque deux modules ont la même adresse de départ, le module le plus proche du SATURN sur le bus est prioritaire. On a donc la hiérarchie suivante par ordre décroissant :

- RAM I/O (mémoire vive des entrées/sorties ou display driver),
- RAM interne 32 ko pour HP-48 S/SX/G, 128 ko pour HP-48 GX,
- le gestionnaire de partitions ou gestionnaire de bancs ou bank-switcher ou encore bank-manager,
- carte en port 1 (HP-48 SX/GX),
- carte en port 2 (HP-48 SX/GX),
- ROM (mémoire morte).

Seule l'adresse de départ du gestionnaire de la RAM I/O peut être ajusté, la taille de la RAM I/O est fixée à 64 quartets (32 octets). Le gestionnaire de partitions (bank-switcher) permet la sélection de la partition active sur les cartes en ports 1 et 2. Il n'existe pas de gestionnaire pour la mémoire morte, notez cependant que celle-ci débute à l'adresse #00000h pour une capacité de 512 ko sur HP-48 G/GX (256 ko sur HP-48 S/SX).

■ Le rôle des gestionnaires de modules de mémoire

Le gestionnaire d'un module de mémoire configure ce dernier en fixant :

- l'adresse de départ du module,
- la taille du module, la RAM I/O étant un cas particulier.

L'adresse de départ d'un module ainsi que sa taille doivent être des multiples de 4096 quartets (#1000h quartets ou 2048 octets ou 2 ko). La RAM I/O constitue ici encore un cas particulier.

Si vous configurez un module avec une taille différente de sa taille réelle, deux cas sont envisageables. Si la taille réelle est supérieure à la taille déclarée, seul un volume d'information égal à la taille déclarée sera accessible. Si la taille réelle est *n* fois plus petite que la taille déclarée, alors vous accéderez à des informations identiques aux adresses @, 2.@, 3.@, 4.@, 5.@... *n*.@.

Même si un module de mémoire est absent (cartes, etc.) son gestionnaire doit être configuré. On lui attribue alors une adresse non utilisée associée à une taille minimale de 4096 quartets.

■ La daisy-chain gère les priorités entre modules de mémoire

A chaque module de mémoire est associé un gestionnaire (manager). Tous les gestionnaires sont reliés entre eux par la « daisy-chain » chargée de gérer les priorités entre les mémoires.

Quand une information est placée sur le bus, le SATURN émet un signal en direction du premier gestionnaire rencontré sur la daisy-chain. Ce signal ordonne à ce premier gestionnaire de « consulter » l'information en transit sur le bus. Si l'information est destinée au module de mémoire associé au gestionnaire considéré, cette information sera traitée par ce module de mémoire. Dans le cas contraire, le gestionnaire envoie un signal vers le gestionnaire suivant, via la daisy-chain. Ce deuxième gestionnaire devra à son tour consulter l'information en transit sur le bus, la traiter si elle lui est destinée ou envoyer un signal vers le gestionnaire suivant dans le cas contraire, et ainsi de suite jusqu'à ce que le module de mémoire destinataire reçoive l'information ou que la mémoire morte - qui ne dispose pas de gestionnaire - soit atteinte.

Lorsque deux mémoires ont été configurées à la même adresse, l'accès aux données de la mémoire la moins prioritaire rend indispensable la déconfiguration de la mémoire la plus prioritaire.

Nous allons maintenant aborder les mécanismes de configuration et de déconfiguration des mémoires au moyen de la daisy-chain. Il faut savoir que chaque gestionnaire de module de mémoire est relié à la daisy-chain par une entrée (daisy-in) et une sortie (daisy-out). L'entrée recevra le signal en provenance d'un gestionnaire de module plus prioritaire, alors que la sortie enverra un signal vers un gestionnaire moins prioritaire.

Les entrées (daisy-in) et les sorties (daisy-out) peuvent se trouver à l'état haut (1) ou à l'état bas (0). Les états des entrées et sorties révèlent la configuration ou déconfiguration du module de mémoire associé :

- la sortie d'un module déconfiguré est à l'état bas,
- l'entrée et la sortie d'un module configuré sont à l'état haut.

Le gestionnaire d'un module de mémoire configuré de façon matérielle (hard configured) et non logicielle a sa sortie au même état que son entrée.

Le tableau ci-dessous présente les entrées et sorties des modules lorsque toutes les mémoires sont déconfigurées. La première flèche en pointillés arrive du SATURN, la seconde va vers la ROM.

	RAM I/O	RAM	Bank Switcher	Carte en port 1	Carte en port 2
Daisy-in	⇢ 1	0	0	0	0
Daisy-out	0	0	0	0	0 ⇢

La commande de bus RESET permet de déconfigurer tous les modules. Au contraire, la commande de bus CONFIG sert à configurer un module. CONFIG configure le module de mémoire déconfiguré dont l'entrée est au niveau haut (ou 1), en effet, un module de mémoire est configuré si ses entrées et sorties ne sont pas au même niveau.

Ainsi, on obtient successivement :

	RAM I/O	RAM	Bank Switcher	Carte en port 1	Carte en port 2
Daisy-in	1	1	0	0	0
Daisy-out	1	0	0	0	0

	RAM I/O	RAM	Bank Switcher	Carte en port 1	Carte en port 2
Daisy-in	1	1	1	0	0
Daisy-out	1	1	0	0	0

	RAM I/O	RAM	Bank Switcher	Carte en port 1	Carte en port 2
Daisy-in	1	1	1	1	0
Daisy-out	1	1	1	0	0

	RAM I/O	RAM	Bank Switcher	Carte en port 1	Carte en port 2
Daisy-in	1	1	1	1	1
Daisy-out	1	1	1	1	0

La carte en port 2 sera donc configurée en dernier. La commande de bus CONFIG configure successivement la RAM I/O (ou display driver), la mémoire vive (RAM), le bank-switcher (ou bank driver ou sélecteur de partition), la carte en port 1 puis la carte en port 2. Rappelons que la ROM se trouve en bout de daisy-chain et est dépourvue de gestionnaire.

Lors d'une déconfiguration, on observe le même phénomène en cascade que pendant la configuration. Ainsi, après la déconfiguration d'un module, le prochain module déconfiguré sera celui qui se trouve à sa gauche sur la daisy-chain telle que nous l'avons représentée jusqu'à maintenant.

■ Cas de la HP-48 G

D'un point de vue matériel, la HP-48 G est très proche de la HP-48 GX, seuls lui manquent les connecteurs pour cartes. Disposant de tous les gestionnaires de la HP-48 GX, la HP-48 G peut gérer le même volume de mémoire. Il est donc possible d'étendre la mémoire vive de la HP-48 G endébloquant son port 1 (ou son port 2) et en connectant de la RAM aux gestionnaires non utilisés.

Pour plus d'informations, voir le chapitre consacré à l'accès à la mémoire masquée.

11

Les registres du SATURN et leurs champs

Ce chapitre rappelle et complète ce qui a déjà été dit à propos des registres dans le chapitre « Introduction au SATURN » dont le contenu doit être maîtrisé avant d'aborder le présent chapitre. En particulier, les notions suivantes doivent être maîtrisées :

- notion de registre,
- nature et utilisation des registres de travail (ou de calcul),
- nature et utilisation des registres de sauvegarde,
- nature et utilisation des registres pointeurs,
- nature et utilisation du compteur ordinal,
- nature et utilisation des registres d'entrée/sortie,
- nature et utilisation de la pile des retours,
- nature et utilisation des drapeaux,
- nature et utilisation du pointeur de champ,

1 Rappels et appellations

Différentes appellations sont utilisées pour désigner un même registre. Nous avons tenté de rassembler les plus courantes :

- un registre de travail n'est autre qu'un registre de calcul,
- un registre pointeur est parfois appelé registre d'adresse,
- le compteur ordinal est aussi appelé pointeur d'instruction,

- le registre d'entrée IN est aussi appelé registre INPUT,
- le registre de sortie OUT est aussi appelé registre OUTPUT,
- la pile des retours est la « return stack » ou RSTK,
- un registre drapeau s'appelle plus simplement « drapeau » ou « flag »,

A ne pas oublier :

- un champ est une portion de registre,
- le pointeur de champ est un registre particulier qui défini un champ,
- un champ et un pointeur de champ sont deux choses différentes bien que le registre pointeur de champ P définisse un champ P correspondant au quartet n°*P* d'un registre de calcul,
- un champ correspond à un groupe de quartets,
- le registre HST se divise en 4 bits (XM, SB, SR et MP).

Le tableau ci-dessous vous a déjà été présenté, il doit absolument être connu par cœur !

Noms des registres	Types	Capacités
A, B, C, D	Registres de travail	16 quartets (64 bits)
R0, R1, R2, R3, R4	Registres de sauvegarde	16 quartets (64 bits)
D0, D1	Registres pointeurs	5 quartets (20 bits)
PC	Compteur ordinal	5 quartets (20 bits)
RSTK	Pile des retours	8x5 quartets (8x20 bits)
IN	Entrées	4 quartets (16 bits)
OUT	Sorties	3 quartets (12 bits)
CARRY	Drapeau	1 bit
ST	Drapeaux	4 quartets (16 bits)
HST (XM, SB, SR, MP)	Drapeaux	1 quartet (4 bits)
P	Pointeur de champ	1 quartet (4 bits)
champ WP	*Champ*	*P quartets (Px4 bits)*

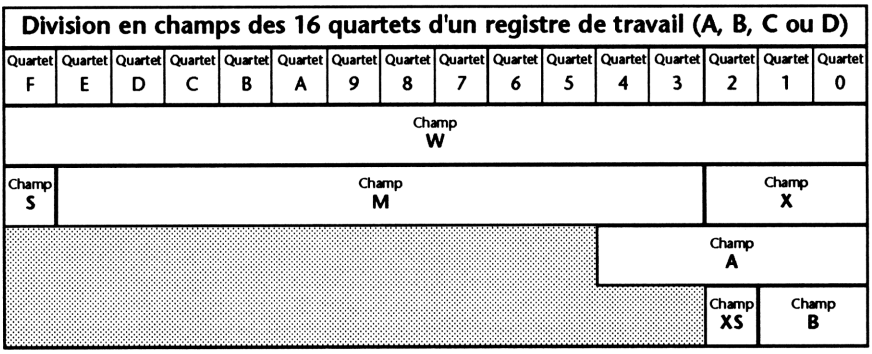
Le tableau ci-dessus vous présente dans l'ordre :

- les registres de travail longs de 16 quartets (ou 64 bits),
- les registres de sauvegarde chargés de mémoriser temporairement le contenu des registres de travail,
- les registres contenant des adresses en mémoire,
- les registres d'entrée et de sortie (échanges),
- les drapeaux,
- le pointeur de champ et le champ WP.

Attention ! Bien qu'il soit présent dans le tableau précédent, WP n'est pas un registre. WP est un champ défini par le registre P.

2
Les champs des registres

Les registres de travail (registres de calcul) peuvent être segmentés en champs. Un champ est un groupe de quartets contigus. La taille d'un champ correspond à un nombre entier de quartets compris entre 1 et 16. Le schéma ci-dessous présente les champs d'un registre de calcul. Notez que les 16 quartets y sont numérotés de #0h (0_{10}) à #Ah (15_{10}).



Il est facile de mémoriser les champs si on connaît l'origine de leurs sigles.

Champs	Contenu prévu initialement (HP-71)	Capacité du champ	Quartets concernés
W	Tout le registre !	16 quartets (64 bits)	0 à 15
S	Signe de la mantisse	1 quartet (4 bits)	15
M	Mantisse	12 quartets (48 bits)	3 à 14
X	Exposant	3 quartets (12 bits)	0 à 2
XS	Signe de l'exposant	1 quartet (4 bits)	2
S + M + X (dont XS)	*Nombre réel*	*16 quartets (64 bits)*	*0 à 15*
B	Octet	2 quartets (8 bits)	0 et 1
A	Adresse en mémoire	5 quartets (20 bits)	0 à 4

Les noms de certains champs sont hérités de leurs fonctions au sein du SATURN équipant le HP-71 (révisez le chapitre « Introduction au SATURN » !).

Il existe une foule de notations différentes pour représenter le champ *c* d'un registre *R* (ces noms sont des exemples), par exemple :

- R(C),
- R.c,
- R:C.

Où *R* est le nom du registre, et *c*, celui du champ.

3

Remarques et astuces

Les différents champs sont indépendants. Modifier un champ d'un registre ne modifie donc pas le contenu des autres champs du même registre à condition bien sûr qu'il ne s'agisse pas de champs superposés ! Faites très attention au contenu de P - même et surtout si vous ne l'utilisez pas ! - qui a le don de perturber certaines opérations sur A et C.

Bien que la pile des retours (RSTK) comporte 8 niveaux, n'en utilisez jamais plus de 6. Le traitement d'une interruption - à un instant quelconque - utilise 2 niveaux. Si ces 2 niveaux ne sont pas disponibles, vous risquez le traditionnel « Memory Lost » (Merci au trombone déplié dans le palpeur du RESET !!!).

Les bits 0 à 11 du registre ST permettent la sauvegarde de résultats de tests. Le bit CARRY (retenue) est modifié si souvent que l'information qu'il contient doit être sauvegardée si elle n'est pas exploitée immédiatement.

La sauvegarde des registres (en particulier B, D et les registres d'adresse D0 et D1) en début de programme doit être un réflexe. En fin de programme, le SATURN doit se trouver en mode hexadécimal et le registre P doit impérativement valoir 0.

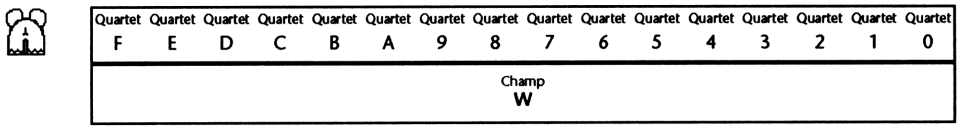
Des routines bien connues prennent en charge la sauvegarde (en début de programme) et la restauration (en fin de programme) des registres.

4

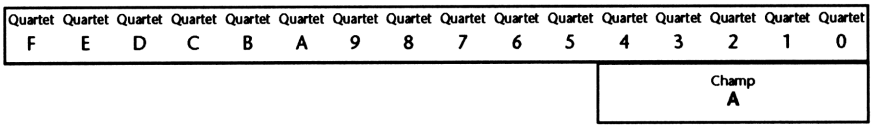
Les champs des registres de calcul

Un registre de calcul (ou registre de travail) a une capacité de 64 bits groupés en 16 quartets numérotés de #0h à #Ah (de 0 à 15). Nous allons représenter un par un chaque champ d'un registre de calcul.

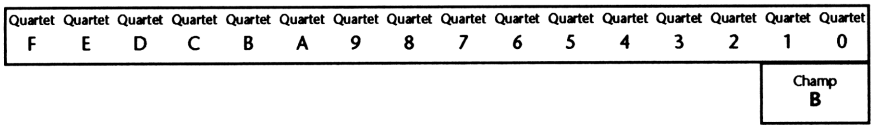
■ Champ W, comme Word (mot), le mot binaire complet



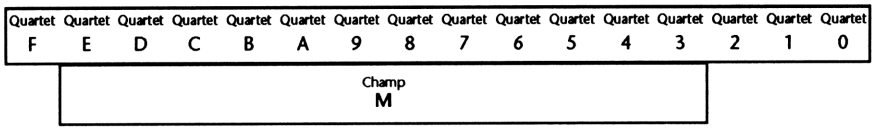
■ Champ A : les 5 premiers quartets... Comme une adresse !



■ Champ B : le premier octet (byte en anglais)

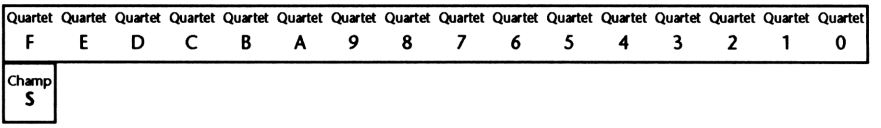


■ Champ M, comme une mantisse de réel (sur HP-71)

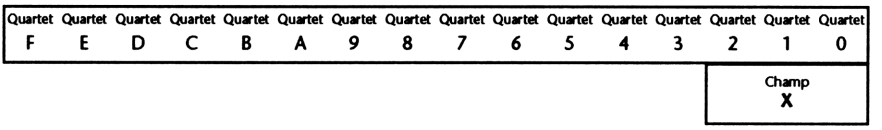


Sur HP-71, lui aussi doté d'un SATURN, le champ M correspondait à la mantisse des nombres réels.

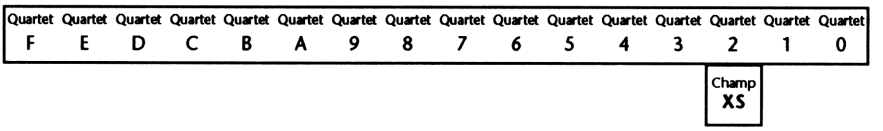
■ Champ S, comme le signe de la mantisse d'un réel (sur HP-71)



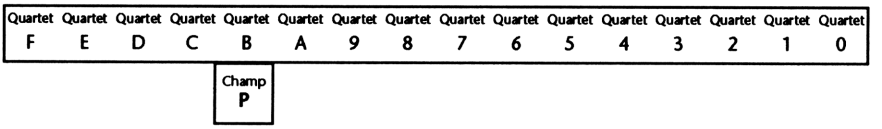
■ Champ X, comme l'eXposant d'un réel (sur HP-71)



■ Champ XS, comme le Signe de l'eXposant d'un réel (sur HP-71)

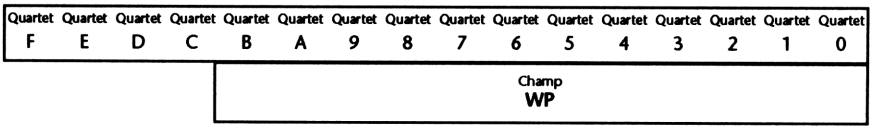


■ Champ P : le quartet dont le numéro se trouve dans le registre P



Ceci n'est qu'un exemple ! Ici, le registre P contient #Bh, donc le champ P correspond au douzième quartet (12_{10}=#Bh=B_{16}).

■ Champ WP : du quartet 0 au quartet dont le numéro se trouve dans le registre P



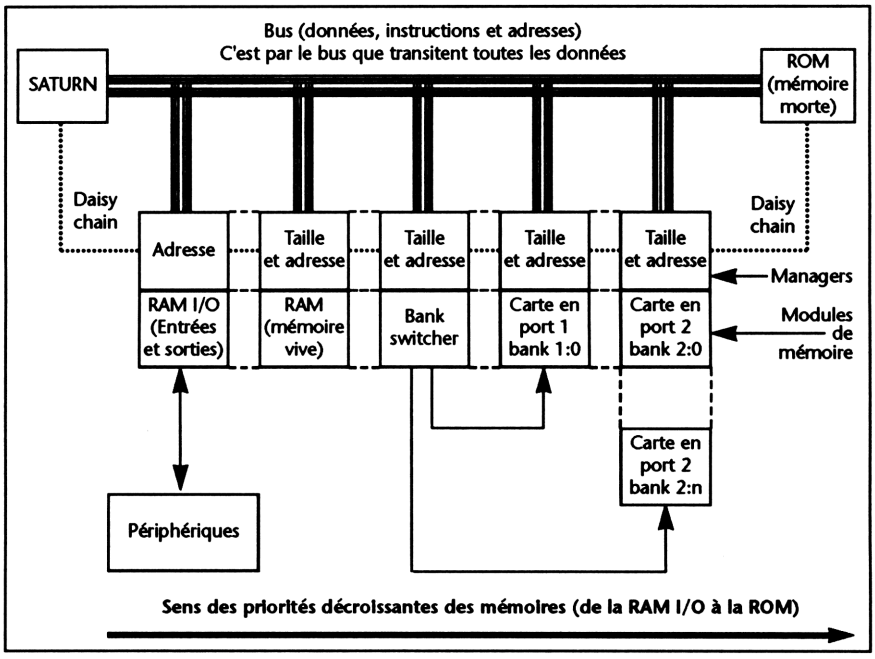
Ceci n'est qu'un exemple ! Ici, le registre P contient #Bh, donc le champ P correspond aux douze premiers quartets (quartets #0h à #Bh).

12

Plages d'adresses des différentes mémoires

1 Hiérarchie des mémoires

La daisy-chain met en évidence la hiérarchie entre les mémoires.



Sur HP-48, une adresse est codée sur 5 quartets, c'est-à-dire sur 20 bits. Il est donc possible de désigner 2^{20} = 1 048 576 adresses en mémoire. L'unité d'information étant, sur HP-48, le quartet, ces 1 048 576 adresses correspondent à 1 048 576 quartets, c'est-à-dire 524 288 octets ou 512 ko puisque 1 ko = 2^{10} octets = 1024 octets. Sur 5 quartets, on code les adresses de #00000h à #FFFFFh.

La mémoire morte (ROM) des HP-48 G/GX a une capacité de 512 ko et a donc besoin de toutes les adresses disponibles de #00000h à #FFFFFh. Il faut donc utiliser une même adresse avec différentes mémoires. Certaines mémoires sont donc plus prioritaires que d'autres.

Les différentes mémoires citées dans l'ordre des priorités décroissantes :

- la mémoire vive des entrées/sorties, appelée RAM I/O,
- la mémoire vive interne (« normale »), appelée RAM,
- le gestionnaire de partitions (ou de bancs) ou bank-switcher,
- la carte en port 1,
- la carte en port 2,
- la mémoire morte, appelée ROM.

Une même adresse peut désigner différents emplacements dans plusieurs mémoires. C'est la mémoire la plus prioritaire qui s'attribue l'adresse. Il s'agit d'un masquage de la mémoire moins prioritaire par la mémoire plus prioritaire obtenu à l'aide de la daisy chain. L'accès à une mémoire masquée est possible en reconfigurant les mémoires ou en fixant des adresses de départ différentes (d'un point de vue logique). Sur HP-48 GX, la mémoire vive interne (RAM) masque la ROM entre les adresses #80000h et #BFFFFh. Pour accéder à cette zone de la ROM, il suffit de fixer le début de la mémoire vive interne en #C0000h.

En cas de reconfiguration de la mémoire, vous devrez obligatoirement interdire les interruptions. En effet, le traitement d'une interruptions débute par la vérification de l'état de la mémoire qui doit se trouver dans son état « normal » .

Configuration normale de la mémoire des HP-48 G/GX :

Mémoires	Adresses
Mémoire vive des entrées/sorties (RAM I/O)	de #100h à #13Fh
Mémoire vive normale (interne) sur HP-48G	de #80000h à #8FFFFh
Mémoire vive normale (interne) sur HP-48GX	de #80000h à #BFFFFh
Gestionnaire de partitions (bank-switcher)	de 7F000h à 7FFFFh
Port carte n°1 vide (HP-48 G ou GX sans carte)	de 7E000h à 7EFFFh
Port carte n°2 vide (HP-48 G ou GX sans carte)	de 7E000h à 7EFFFh
Port carte n°1 avec carte sur HP-48 GX	à partir de C0000h
Port carte n°2 avec carte sur HP-48 GX	à partir de C0000h

2

HP-48 G

Adresses	Mémoire vive des entrées et sorties (RAM I/O)	Mémoire vive interne (RAM)	Gestion des partitions (Bank switcher)	Port 1	Port 2	Mémoire morte (ROM)
de #0h à #FFh 256 quartets						256 quartets
de #100h à #13Fh 64 quartets	64 quartets					Masqué par RAM I/O 64 quartets
de #140h à #7DFFFh 515776 quartets						515776 quartets (=252 ko)
de #7E000h à #7EFFFh 4096 quartets				Port vide 2 ko	Masqué par port 1 Port vide 2 ko	Masqué par ports 1 & 2 2 ko
de #7F000h à #7FFFFh 4096 quartets			2 ko			Masqué par bank switcher 2 ko
de #80000h à #8FFFFh 65536 quartets		32 ko				Masqué par IRAM interne 32 ko
de #90000h à #BFFFFh 196608 quartets						224 ko
de #C0000h à #CFFFFh 65536 quartets						
de #D0000h à #FFFFFh 196608 quartets						

3

HP-48 GX sans carte

Adresses	Mémoire vive des entrées et sorties (RAM I/O)	Mémoire vive interne (RAM)	Gestion des partitions (Bank switcher)	Port 1	Port 2	Mémoire morte (ROM)
de #0h à #FFh 256 quartets						256 quartets
de #100h à #13Fh 64 quartets	64 quartets					Masqué par RAM I/O 64 quartets
de #140h à #7DFFFh 515776 quartets						515776 quartets (=252 ko)
de #7E000h à #7EFFFh 4096 quartets				Port vide 2 ko	Masqué par port 1 Port vide 2 ko	Masqué par ports 1 & 2 2 ko
de #7F000h à #7FFFFh 4096 quartets			2 ko			Masqué par bank switcher 2 ko
de #80000h à #8FFFFh 65536 quartets		128 ko				Masqué par RAM interne 128 ko
de #90000h à #BFFFFh 196608 quartets						
de #C0000h à #CFFFFh 65536 quartets						128 ko
de #D0000h à #FFFFFh 196608 quartets						

4

HP-48 GX (32 ko en port 1)

Adresses	Mémoire vive des entrées et sorties (RAM I/O)	Mémoire vive interne (RAM)	Gestion des partitions (Bank switcher)	Port 1	Port 2	Mémoire morte (ROM)
de #0h à #FFh 256 quartets						256 quartets
de #100h à #13Fh 64 quartets	64 quartets					Masqué par RAM I/O 64 quartets
de #140h à #7DFFFh 515776 quartets						515 776 quartets (=252 ko)
de #7E000h à #7EFFFh 4096 quartets					Port vide 2 ko	Masqué par port 2 2 ko
de #7F000h à #7FFFFh 4096 quartets			2 ko			Masqué par bank switcher 2 ko
de #80000h à #8FFFFh 65536 quartets		128 ko				Masqué par RAM interne 128 ko
de #90000h à #BFFFFh 196608 quartets						
de #C0000h à #CFFFFh 65536 quartets				Carte 32 ko (carte 48SX par exemple)		Masqué par carte en port 1 32 ko
de #D0000h à #FFFFFh 196608 quartets						96 ko

5
HP-48 GX (128 ko en port 1)

Adresses	Mémoire vive des entrées et sorties (RAM I/O)	Mémoire vive interne (RAM)	Gestion des partitions (Bank switcher)	Port 1	Port 2	Mémoire morte (ROM)
de #0h à #FFh 256 quartets						256 quartets
de #100h à #13Fh 64 quartets	64 quartets					Masqué par RAM I/O 64 quartets
de #140h à #7DFFFh 515776 quartets						515 776 quartets (=252 ko)
de #7E000h à #7EFFFh 4096 quartets					Port vide 2 ko	Masqué par port 2 2 ko
de #7F000h à #7FFFFh 4096 quartets			2 ko			Masqué par bank switcher 2 ko
de #80000h à #8FFFFh 65536 quartets		128 ko				Masqué par RAM interne 128 ko
de #90000h à #BFFFFh 196608 quartets						
de #C0000h à #CFFFFh 65536 quartets				Carte de type GX ou SX en port 1 128 ko		Masqué par carte en port 1 128 ko
de #D0000h à #FFFFFh 196608 quartets						

6

HP-48 GX (128 ko en ports 1 et 2)

Adresses	Mémoire vive des entrées et sorties (RAM I/O)	Mémoire vive interne (RAM)	Gestion des partitions (Bank switcher)	Port 1	Port 2	Mémoire morte (ROM)
de #0h à #FFh 256 quartets						256 quartets
de #100h à #13Fh 64 quartets	64 quartets					Masque par RAM I/O 64 quartets
de #140h à #7DFFFh 515776 quartets						519 872 quartets (=253 ko)
de #7E000h à #7EFFFh 4096 quartets						
de #7F000h à #7FFFFh 4096 quartets			2 ko			Masque par bank switcher 2 ko
de #80000h à #8FFFFh 65536 quartets		128 ko				Masque par RAM interne 128 ko
de #90000h à #BFFFFh 196608 quartets						
de #C0000h à #CFFFFh 65536 quartets				Carte type SX ou GX 128 ko	Masque par Carte en port 1 Carte type GX (*n*·128 ko) 128 ko	Masque par cartes en port 1 et 2 128 ko
de #D0000h à #FFFFFh 196608 quartets						

7
Remarques

Ces différents schémas mettent en évidence le masquage de certaines parties de la mémoire, en particulier, le masquage de la mémoire morte (ROM) dont certaines parties sont toujours masquées en configuration normale. Au contraire certaines parties de la mémoire morte sont, en configuration normale, toujours accessibles. C'est notamment le cas des deux zones suivantes :

- de #0h à #FFh,
- de #140h à #7DFFFh.

Le dernier cas abordé (HP-48 GX avec deux cartes) correspond aussi à la configuration d'une HP-48 GX ayant dans son port 2 une carte d'une capacité supérieure à 128 ko. En effet, une carte de plus de 128 ko est segmentée en partitions de 128 ko. Une seule partition est rendue visible à un instant donné. Le gestionnaire de partitions (bank switcher) est chargé de la sélection des « tranches » de 128 ko (partitions).

13

Introduction au jeu d'instructions

Ce chapitre introduit les quatre suivants. Ceux-ci vous présentent le jeu d'instructions du microprocesseur SATURN. Il s'agit donc de l'ensemble des ordres qui peuvent être donnés à la machine en langage d'assemblage (en « assembleur »). Dans cette partie du livre, les chapitres consacrés au jeu d'instructions sont chargés de vous présenter le rôle de chaque instruction. Vous trouverez dans la partie « références » du présent ouvrage deux chapitres permettant une recherche rapide dans une liste d'un mnémonique à partir de son code ou d'un code à partir de son mnémonique.

■ Présentation des instructions

Les instructions sont regroupées par familles et vous sont présentées au moyen de tableaux tels que celui-ci :

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A-B *ch*	E0	A	oui	7
	Ba0	P, WP, XS, X, S, M, B, W		3+*n*
B=B-C *ch*	E1	A	oui	7
	Ba1	P, WP, XS, X, S, M, B, W		3+*n*

La première colonne contient les mnémoniques. Il est utile de bien connaître ceux-ci pour programmer rapidement. La deuxième colonne présente le code hexadécimal qui correspond au mnémonique. En fait, le code hexadécimal à utiliser peut varier en fonction du champ du registre sur lequel agit l'instruction. La troisième colonne contient les noms des champs sur lesquels agit un code donné. Dans notre exemple, à un même mnémonique correspondent deux codes hexadécimaux. En général, l'un

est à utiliser si l'instruction s'applique au champ A, l'autre est à employer si l'instruction s'applique à un champ autre que A (P, WP, etc.). Enfin, la quatrième colonne signale une modification de la retenue (« carry bit ») par l'instruction alors que la cinquième colonne indique le nombre de cycles nécessaires à l'exécution de l'instruction. Ces cycles sont généralement appelés « microcycles » et ont une durée égale à l'inverse de la fréquence de fonctionnement du microprocesseur. Ainsi, la HP-48 G étant dotée d'un microprocesseur fonctionnant à peu près à 4 MHz, un microcycle sur HP-48 G dure environ 0,00000025 seconde (0,25 µs). Par exemple, si 8 microcycles sont nécessaires à l'exécution d'une instruction donnée, le SATURN travaillera ici pendant 2 µs pour ladite instruction.

■ Notations utilisées dans les tableaux

Dans la première colonne, *ch* désigne un champ, *ch* est donc à remplacer selon vos besoins par A, P, WP, XS, X, etc. Les adresses et les décalages d'adresse sont notés *pq*, *pqr*, *pqrs* ou *pqrst*. Chacune de ces lettres représente un chiffre hexadécimal (un quartet). Les champs étant inversés, *pq*, *pqr*, *pqrs* et *pqrst* deviennent respectivement *qp*, *rqp*, *srqp* et *tsrqp* dans les mnémoniques. Le caractère *n* désigne un quartet variable, *n* est donc un nombre compris entre #0h et #Fh.

Dans la deuxième colonne du tableau les caractères *a* et *b* représentent un quartet variant en fonction du champ auquel s'applique l'instruction. Le tableau ci-dessous présente les correspondances entre les valeurs de *a* ou de *b* (selon l'instruction) et le champ auquel s'applique l'instruction.

Champ auquel s'applique l'instruction	Valeur de *a*	Valeur de *b*
P	0	8
WP	1	9
XS	2	A
X	3	B
S	4	C
M	5	D
B	6	E
W	7	F

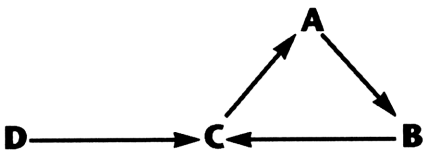
Par exemple, codons l'instruction `A=A-B` appliquée aux champs B des registres concernés (instruction `A=A-B B`). La tableau de la page précédente indique que le code à utiliser est `Ba0`. Le tableau ci-dessus, nous indique que *a* prend pour valeur #6h lorsque le champ B doit être désigné. On obtient ainsi le code hexadécimal `B60` pour `A=A-B B`.

Ne confondez pas les chiffres hexadécimaux A et B (#Ah et #Bh) avec les quartets variables `a` et `b` qui peuvent prendre les valeurs citées dans le tableau.

Les troisièmes colonnes des tableaux de présentation des instructions permettent de choisir le code hexadécimal en fonction du champ auquel s'applique l'instruction représentée par son mnémonique. Les cinquièmes colonnes contiennent parfois un élément variable généralement noté *m* ou *n* dont le rôle est précisé au cas par cas.

■ Utilisation des registres de travail

Ayez à l'esprit le schéma ci-dessous :



Il résume les manipulations possibles entre les registres de travail que sont les registres A, B, C et D. Lorsque l'on utilise deux registres de travail (pour un calcul, un échange de quartets, etc.), on manipule donc l'un des quatre couples possibles de registres de travail. Ces couples sont (A,B), (A,C), (B,C) et (C,D).

■ Structure des chapitres consacrés au jeu d'instructions

Les quatre chapitres qui suivent aborderont successivement :

- le traitement des registres de travail,
  - initialisation à zéro,
  - affectation d'une valeur initiale,
  - incrémentation et décrémentation,
  - copie ou échange de champs,
  - complémentation, rotation et division par décalage,
  - opérations arithmétiques et logiques élémentaires,
- la manipulation des registres de sauvegarde et des pointeurs,
  - échange ou copie entre les registres de sauvegarde et les registres de travail,
  - lecture et écriture dans les registres D0 et D1,
  - lecture et écriture à l'adresse désignée par D0 ou D1,

- les sauts et les tests,
  - sauts relatifs conditionnels ou inconditionnels,
  - sauts absolus et indirects,
  - appel d'un sous-programme,
  - retour d'un sous-programme,
  - test de l'état d'un bit,
  - test de la nullité d'un registre,
  - comparaison entre registres,
  - gestion du résultat d'un test,
- les autres instructions,
  - manipulation des registres P, RSTK, IN, OUT, ST et HS,
  - choix du mode décimal ou hexadécimal,
  - instructions relatives au bus,
  - instructions nulles.

■ Instructions équivalentes

Lorsqu'il est possible d'écrire une même instruction à l'aide de plusieurs mnémoniques différents, seul le mnémonique le plus courant vous est proposé, et ce, afin de n'encombrer ni votre mémoire ni les tableaux de présentation des mnémoniques. Ainsi, `A=A&B` *ch* vous est présenté mais pas `A=B&A` *ch* qui n'est qu'une autre forme du même mnémonique, de même pour `CAEX` et `ACEX`. Rare exception, les différentes formes de `LAHEX` et `LCHEX` sont citées car le débutant ne peut pas deviner que `LA` et `LC` sont des formes abrégées de ces instructions. Par contre, lorsque plusieurs codes hexadécimaux peuvent être associés à un même mnémonique, tous ces codes seront cités (voir les listes d'instructions dans la partie « Références » du présent ouvrage).

■ Mémoriser les mnémoniques

Comme son nom l'indique, un mnémonique est un moyen mnémotechnique, ici utilisé pour retenir une instruction du langage assembleur. Dans la mesure du possible, le mnémonique tente de représenter à la fois simplement et explicitement l'action d'une instruction. Les noms des registres (et de leurs champs) concernés par l'instruction apparaissent dans le mnémonique ainsi que certaines abréviations, par exemple, EX (« exchange ») pour un échange, R (« right ») pour un décalage à droite, L (« left ») pour un décalage à gauche, etc. Essayez de détecter ces abréviations pour mieux mémoriser les mnémoniques. Cela dit, une mise en garde s'impose car il n'existe pas de véritable normalisation des mnémoniques de l'assembleur SATURN, vous pourrez donc rencontrer, en fonction des auteurs, plusieurs mnémoniques différents pour une même instruction...

14

Instructions : traitement des registres de travail

1

Affectation d'une valeur

1.1

Chargement d'une valeur initiale

■ Chargement d'une constante dans les registres A et C

Il est indispensable de pouvoir charger une valeur initiale dans les registres A et C. On utilise pour cela les instructions LAHEX et LCHEX qui sont généralement notées LA et LC par soucis de simplification. LA $nh_n..h_0$ et LC $nh_n..h_0$ placent les quartets (nombres hexadécimaux) h_0 à h_n dans les quartets n°p à n°$p+n$ (p étant le contenu du registre P). Ces instructions permettent le transfert de $n+1$ quartets vers le registre A ou le registre C. Le paramètre n varie entre 0 (#0h) et 15 (#Fh), n=0 correspond donc au transfert d'un unique quartet (h_0) vers le quartet n°p du registre A ou C. Notez que le registre P devra se voir affecter au préalable le numéro du quartet dans le registre A ou C à partir duquel seront transférés les quartets h_0 à h_n. L'affectation d'une valeur au registre P est abordée dans le chapitre « *Instructions : mode, registres spéciaux, interruptions, etc.* ».

CCD 20

Par exemple, le contenu initial du registre A étant FEDCBA9876543210, on considère l'instruction LA $nh_n..h_0$. Si n vaut 2, on transfert $n+1$ quar-

tets, autrement dit, 3 quartets notés h_0, h_1, h_2. Si les quartets h_0=#Ah, h_1=#Bh et h_2=#Ch doivent être copiés à partir du troisième quartet du registre A, il faut :

- affecter au registre P la valeur #2h, en effet, les 16 quartets du registre A étant numérotés de #0h à #Fh, le troisième quartet a donc #2h pour numéro. Comme nous l'avons dit, l'affectation d'une valeur particulière au registre P est traitée dans un autre chapitre,
- coder LAnh$_n$..h$_0$ avec les valeurs utilisées dans cet exemple (n=2, h_0=#Ah, h_1=#Bh et h_2=#Ch). Pour obtenir le code hexadécimal correspondant à LA2CBA, LA se code 8082, n est codé par sa valeur et h$_n$..h$_0$ sont codés #ABCh en étant réordonnés de h$_0$ à h$_n$. On obtient ainsi le code 80822ABC.

Le registre A avait pour valeur initiale FEDCBA9876543210. Après la manipulation que nous venons de décrire, il prend pour nouvelle valeur les 16 quartets FEDCBA98765**CBA**10.

A retenir : LAHEX et LCHEX (LA et LC) transfèrent n+1 quartets à partir du quartet n°p, p étant la valeur chargée au préalable dans le registre P. Le quartet h_0 se place sur le quartet n°p, le quartet h_1 est copié sur le quartet n°p+1 et ainsi de suite jusqu'à ce que le quartet h_n soit placé sur le quartet n°p+n.

Il existe une circularité des affectations de valeurs aux quartets. Ainsi, si vous tentez d'affecter des valeurs à 5 quartets en partant du quartet #Eh du registre (l'avant dernier quartet), vous placerez h_0 dans le quartet n°#Eh, h_1 dans le quartet #Fh mais h_2, h_3 et h_4 seront copiés dans les quartets n°#0h, n°#1h et n°#2h.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
LAnh$_n$..h$_0$	8082nh$_0$..h$_n$	Défini par le registre P	non	6+n
LCnh$_n$..h$_0$	3nh$_0$..h$_n$	Défini par le registre P	non	4+n

La retenue (carry bit) n'est pas modifiée par LAHEX (LA) et LCHEX (LC).

Le nombre de cycles nécessaires est fonction du paramètre n. Ainsi, pour le transfert d'un quartet vers le registre C, 6 cycles sont nécessaires. Notez que le chargement du registre A est plus long que celui du registre C et que ces opérations de chargement ne sont possibles que sur A et C.

LAHEX et LCHEX sont parfois notées LA et LC mais elles peuvent aussi être notées LA(n+1) et LC(n+1) pour vous rappeler que même si n vaut 0, un quartet sera transféré, le nombre de quartets transférés valant n+1.

■ Initialisation à zéro d'un champ d'un registre de travail

Ces instructions remplacent par des #0h le champ spécifié du registre indiqué. Le paramètre *ch* est à remplacer par un nom de champ. Le paramètre *b* prend une valeur liée à *ch*.

Ainsi, pour mettre à zéro l'ensemble du registre B, il suffit de coder l'instruction `B=0 W`. Le tableau ci-dessous nous indique qu'il faut utiliser le code `Ab1`. Au chapitre « *Introduction au jeu d'instructions* » il nous est précisé que *b* doit être remplacé par #Fh lorsque l'instruction concerne le champ W. On obtient ainsi le code `AF1` pour `B=0 W`.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=0 *ch*	D0	A	non	7
	A*b*0	P, WP, XS, X, S, M, B, W		3+*n*
B=0 *ch*	D1	A	non	7
	A*b*1	P, WP, XS, X, S, M, B, W		3+*n*
C=0 *ch*	D2	A	non	7
	A*b*2	P, WP, XS, X, S, M, B, W		3+*n*
D=0 *ch*	D3	A	non	7
	A*b*3	P, WP, XS, X, S, M, B, W		3+*n*

Le nombre de cycles nécessaires à l'exécution est fonction du nombre *n* de quartets constituant le champ concerné par l'instruction.

■ Manipulation d'un bit isolé

Vous pouvez affecter « manuellement » une valeur à l'un des 16 premiers bits des registres A et C (bits des 4 premiers quartets des registres A et C). On « arme un bit » lorsqu'on lui attribue 1 pour valeur. Inversement, on le « désarme » lorsqu'il reprend la valeur 0. Les instructions `ABIT` et `CBIT` concernent respectivement les registres A et C.

Par exemple, pour mettre à 1 le deuxième bit du deuxième quartet (il s'agit donc du bit n°#5h) du registre C on codera ici `CBIT=1 5` (`CBIT` parce qu'il s'agit du registre C, `=1` parce qu'on arme et #5h pour le numéro du bit). On obtient ainsi le code 80895.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
ABIT=0 *n*	8084*n*	bit isolé	non	7
ABIT=1 *n*	8085*n*	bit isolé	non	7
CBIT=0 *n*	8088*n*	bit isolé	non	7
CBIT=1 *n*	8089*n*	bit isolé	non	7

1.2
Opérations entre registres

■ Echange des contenus des champs de deux registres

Vous pouvez échanger le contenu d'un champ d'un registre donné avec le contenu du champ correspondant d'un autre registre. Par exemple, vous pouvez échanger le champ X de A avec le champ X de C.

Par exemple, pour échanger le contenu de A(M) (champ M du registre A) avec celui de C(M), on utilisera l'instruction `CAEX M` qui sera codée par `ADE`.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
ABEX *ch*	DC	A	non	7
	A*b*C	P, WP, XS, X, S, M, B, W		3+*n*
BCEX *ch*	DD	A	non	7
	A*b*D	P, WP, XS, X, S, M, B, W		3+*n*
CAEX *ch*	DE	A	non	7
	A*b*E	P, WP, XS, X, S, M, B, W		3+*n*
DCEX *ch*	DF	A	non	7
	A*b*F	P, WP, XS, X, S, M, B, W		3+*n*

Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets constituant le champ auquel est appliquée l'instruction.

Attention ! Plusieurs notations peuvent être employées pour ces instructions qui ont une action symétrique sur les deux registres qu'elles concernent. Ainsi, `CAEX` peut aussi être notée `ACEX` et `ABEX`, `BCEX` et `DCEX` pourront respectivement être notées `BAEX`, `CBEX` et `CDEX`. Il existe donc deux notations pour chaque instruction. Ces deux notations sont absolument équivalentes.

■ Copie d'un registre dans un autre registre

Nous venons de voir qu'il est possible d'échanger les contenus de deux champs de même type appartenant à deux registres de travail différents. Il est aussi possible de copier le contenu d'un champ d'un registre dans le champ correspondant dans un autre registre. Par exemple, vous pourrez ainsi copier A(M) (champ M du registre A) dans B(M).

Codons par exemple la copie de B(X) (champ X du registre B) dans A(X). `B=A ch` est codé `Ab8` la table de correspondance du chapitre « *Introduction au jeu d'instructions* » nous précise que `b` doit valoir #Bh pour représenter le champ X. On obtient ainsi le code `AB8`.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=B ***ch***	D4	A	non	7
	*Ab*4	P, WP, XS, X, S, M, B, W		3+*n*
B=A ***ch***	D8	A	non	7
	*Ab*8	P, WP, XS, X, S, M, B, W		3+*n*
B=C ***ch***	D5	A	non	7
	*Ab*5	P, WP, XS, X, S, M, B, W		3+*n*
C=B ***ch***	D9	A	non	7
	*Ab*9	P, WP, XS, X, S, M, B, W		3+*n*
C=A ***ch***	D6	A	non	7
	*Ab*6	P, WP, XS, X, S, M, B, W		3+*n*
A=C ***ch***	DA	A	non	7
	*Ab*A	P, WP, XS, X, S, M, B, W		3+*n*
D=C ***ch***	D7	A	non	7
	*Ab*7	P, WP, XS, X, S, M, B, W		3+*n*
C=D ***ch***	DB	A	non	7
	*Ab*B	P, WP, XS, X, S, M, B, W		3+*n*

Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets constituant le champ auquel est appliquée l'instruction.

2
Traitements

2.1
Les additions

■ Incrémentation

L'incrémentation d'un champ d'un registre consiste simplement à ajouter #1h au contenu dudit champ.

Mnémoniques		Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A+1	***ch***	E4	A	oui	7
		Ba4	P, WP, XS, X, S, M, B, W		3+*n*
B=B+1	***ch***	E5	A	oui	7
		Ba5	P, WP, XS, X, S, M, B, W		3+*n*
C=C+1	***ch***	E6	A	oui	7
		Ba6	P, WP, XS, X, S, M, B, W		3+*n*
D=D+1	***ch***	E7	A	oui	7
		Ba7	P, WP, XS, X, S, M, B, W		3+*n*

Par exemple, incrémentons C(B), autrement dit, ajoutons #1h au contenu du premier octet du champ C. Il faut pour cela coder `C=C+1  B`. Dans `Ba6`, *a* vaut #6h puisqu'il s'agit du champ B. On obtient le code `B66`.

Ici, la retenue (« carry bit ») vaut 1 si l'incrémentation provoque un dépassement de la capacité du champ qui vaut alors #0h. Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution est le nombre de quartets constituant le champ auquel est appliquée l'instruction.

■ Addition d'un champ

Les instructions citées ci-après permettent d'ajouter le contenu d'un champ d'un registre donné au même champ d'un autre registre. Par exemple, le champ A(X) peut se voir ajouter le contenu du champ B(X). A(X) vaudra finalement A(X) + B(X). Par exemple, l'ajout de B(X) à C(X) correspond au mnémonique `C=B+C  X` qui est codé `A39`.

Mnémoniques		Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A+B	***ch***	C0	A	oui	7
		Aa0	P, WP, XS, X, S, M, B, W		3+*n*
B=A+B	***ch***	C8	A	oui	7
		Aa8	P, WP, XS, X, S, M, B, W		3+*n*
B=B+C	***ch***	C1	A	oui	7
		Aa1	P, WP, XS, X, S, M, B, W		3+*n*
C=B+C	***ch***	C9	A	oui	7
		Aa9	P, WP, XS, X, S, M, B, W		3+*n*
C=C+A	***ch***	C2	A	oui	7
		Aa2	P, WP, XS, X, S, M, B, W		3+*n*
A=C+A	***ch***	CA	A	oui	7
		AaA	P, WP, XS, X, S, M, B, W		3+*n*
D=D+C	***ch***	C3	A	oui	7
		Aa3	P, WP, XS, X, S, M, B, W		3+*n*
C=D+C	***ch***	CB	A	oui	7
		AaB	P, WP, XS, X, S, M, B, W		3+*n*

Les instructions présentées ci-dessous sont très proches de celles que nous venons de voir et permettent de doubler la valeur d'un champ en ajoutant un champ à lui-même.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A+A ***ch***	C4	A	oui	7
	A*a*4	P, WP, XS, X, S, M, B, W		3+*n*
B=B+B ***ch***	C5	A	oui	7
	A*a*5	P, WP, XS, X, S, M, B, W		3+*n*
C=C+C ***ch***	C6	A	oui	7
	A*a*6	P, WP, XS, X, S, M, B, W		3+*n*
D=D+D ***ch***	C7	A	oui	7
	A*a*7	P, WP, XS, X, S, M, B, W		3+*n*

Pour toutes les instructions des deux tableaux précédents, la retenue (« carry bit ») vaut 1 en cas de dépassement de la capacité du champ (« overflow »). Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets constituant le champ auquel est appliquée l'instruction.

■ Addition d'une constante

Pour ajouter #1h au contenu d'un champ d'un registre, il suffit d'utiliser les instructions d'incrémentation que nous avons déjà exposées. Nous abordons ici une série d'instructions permettant d'ajouter à un champ des valeurs comprises entre #2h et #10h. Ces instructions ajoutent *n*+1 au contenu du champ, ainsi, pour ajouter #2h à un champ, il faut donner à *n* la valeur #1h.

Exceptionnellement, tous les champs habituels ne sont pas concernés par cette série d'instructions. En particulier, les champs P, WP, XS et S ne sont pas utilisables. La retenue (« carry bit ») vaut 1 si le calcul provoque une retenue à reporter.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A+(n+1) ***ch***	818F0*n*	A	oui	6+*m*
	818*a*0*n*	X, M, B, W		
B=B+(n+1) ***ch***	818F1*n*	A	oui	6+*m*
	818*a*1*n*	X, M, B, W		
C=C+(n+1) ***ch***	818F2*n*	A	oui	6+*m*
	818*a*2*n*	X, M, B, W		
D=D+(n+1) ***ch***	818F3*n*	A	oui	6+*m*
	818*a*3*n*	X, M, B, W		

Le paramètre *m* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets constituant le champ auquel est appliquée l'instruction.

N'oubliez pas que chaque quartet d'un registre a un « poids » dépendant de son numéro d'ordre. Le quartet n°#0h « pèse » 16^0 alors que le quartet n°#15h « pèse » 16^{15}. Ajouter #2h au quartet n°#2h revient à ajouter 512 (#200h) au registre puisque ce quartet a pour poids 16^2... La constante à ajouter à un champ d'un registre de travail est limitée à #Fh cependant il n'y a pas là de véritable limitation puisque tout dépend du poids du quartet qui sera augmenté !

2.2
Les soustractions et complémentations

■ Décrémentation

La décrémentation d'un champ d'un registre consiste à ôter #1h au contenu dudit champ.

Par exemple, décrémentons C(B), autrement dit, retirons #1h au contenu du premier octet du champ C. Il faut pour cela coder `C=C-1 B`. Dans `AaE`, *a* vaut #6h puisqu'il s'agit du champ B. On obtient le code `A6E`.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A-1 *ch*	CC	A	oui	7
	AaC	P, WP, XS, X, S, M, B, W		3+*n*
B=B-1 *ch*	CD	A	oui	7
	AaD	P, WP, XS, X, S, M, B, W		3+*n*
C=C-1 *ch*	CE	A	oui	7
	AaE	P, WP, XS, X, S, M, B, W		3+*n*
D=D-1 *ch*	CF	A	oui	7
	AaF	P, WP, XS, X, S, M, B, W		3+*n*

Ici, la retenue (« carry bit ») vaut 1 si le champ que l'on tente de décrémenter vaut #0h. Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets constituant le champ auquel est appliquée l'instruction.

Il existe d'autres instructions permettant d'ôter à un registre une valeur pouvant atteindre #Fh.

■ Soustraction

Les instructions citées ci-après permettent de soustraire le contenu d'un champ d'un registre donné au même champ d'un autre registre. Par exemple, le champ A(X) peut se voir soustraire le contenu du champ B(X). A(X) vaudra finalement A(X) - B(X). Par exemple, la soustraction de B(X) à A(X) correspond au mnémonique `A=A-B X` qui est codé `B30`.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A-B *ch*	E0	A	oui	7
	Ba0	P, WP, XS, X, S, M, B, W		3+*n*
B=B-C *ch*	E1	A	oui	7
	Ba1	P, WP, XS, X, S, M, B, W		3+*n*
C=C-A *ch*	E2	A	oui	7
	Ba2	P, WP, XS, X, S, M, B, W		3+*n*
D=D-C *ch*	E3	A	oui	7
	Ba3	P, WP, XS, X, S, M, B, W		3+*n*
B=B-A *ch*	E8	A	oui	7
	Ba8	P, WP, XS, X, S, M, B, W		3+*n*
C=C-B *ch*	E9	A	oui	7
	Ba9	P, WP, XS, X, S, M, B, W		3+*n*
A=A-C *ch*	EA	A	oui	7
	BaA	P, WP, XS, X, S, M, B, W		3+*n*
C=C-D *ch*	EB	A	oui	7
	BaB	P, WP, XS, X, S, M, B, W		3+*n*
A=B-A *ch*	EC	A	oui	7
	BaC	P, WP, XS, X, S, M, B, W		3+*n*
B=C-B *ch*	ED	A	oui	7
	BaD	P, WP, XS, X, S, M, B, W		3+*n*
C=A-C *ch*	EE	A	oui	7
	BaE	P, WP, XS, X, S, M, B, W		3+*n*
D=C-D *ch*	EF	A	oui	7
	BaF	P, WP, XS, X, S, M, B, W		3+*n*

Pour toutes ces instructions, la retenue (« carry bit ») vaut 1 si la différence est négative. Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets constituant le champ auquel est appliquée l'instruction.

■ Soustraction d'une constante

Pour retirer #1h au contenu d'un champ d'un registre, il suffit d'utiliser les instructions de décrémentation déjà vues. Nous abordons ici une série d'instructions permettant de soustraire à un champ des valeurs comprises entre #2h et #10h.

Ces instructions retirent n+1 au contenu du champ, ainsi, pour retirer #3h à un champ, il faut donner à n la valeur #2h.

Exceptionnellement, tous les champs habituels ne sont pas concernés par cette série d'instructions. En particulier, les champs P, WP, XS et S ne sont pas utilisables.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A-(n+1) *ch*	818F8*n*	A	oui	6+*m*
	818*a*8*n*	X, M, B, W		6+*m*
B=B-(n+1) *ch*	818F9*n*	A	oui	6+*m*
	818*a*9*n*	X, M, B, W		6+*m*
C=C-(n+1) *ch*	818FA*n*	A	oui	6+*m*
	818*a*A*n*	X, M, B, W		6+*m*
D=D-(n+1) *ch*	818FB*n*	A	oui	6+*m*
	818*a*B*n*	X, M, B, W		6+*m*

La retenue (« carry bit ») vaut 1 si le calcul provoque une retenue à reporter (ici, un résultat négatif). Le paramètre m dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets constituant le champ auquel est appliquée l'instruction.

Le fait de ne pouvoir soustraire que des valeurs limitées à #Fh n'est pas vraiment une limitation car, plus que la valeur soustraite, c'est le poids du quartet qui devra supporter la soustraction qui importe.

■ Complément à 2

Le résultat renvoyé par ces instructions dépend du mode de fonctionnement du SATURN. Si le SATURN fonctionne en mode décimal, c'est le complément à 10_{10} qui sera renvoyé comme résultat.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=-A *ch*	F8	A	oui	7
	B*b*8	P, WP, XS, X, S, M, B, W		3+*n*
B=-B *ch*	F9	A	oui	7
	B*b*9	P, WP, XS, X, S, M, B, W		3+*n*
C=-C *ch*	FA	A	oui	7
	B*b*A	P, WP, XS, X, S, M, B, W		3+*n*
D=-D *ch*	FB	A	oui	7
	B*b*B	P, WP, XS, X, S, M, B, W		3+*n*

La retenue (« carry bit ») vaut 1 si le champ auquel est appliquée l'instruction n'a pas #0h pour valeur initiale. Le paramètre n dont dépend le

nombre de cycles nécessaires à l'exécution correspond au nombre de quartets constituant le champ auquel est appliquée l'instruction.

■ Complément à 1

Le résultat renvoyé par ces instructions dépend du mode de fonctionnement du SATURN. Si le SATURN fonctionne en mode décimal, c'est le complément à 9_{10} qui sera renvoyé comme résultat. Le complément à 1 correspond à l'inversion de l'état de tous les bits du champs, on obtient le complément à 1 en mode hexadécimal.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=-A-1 *ch*	FC	A	oui	7
	B*b*C	P, WP, XS, X, S, M, B, W		3+*n*
B=-B-1 *ch*	FD	A	oui	7
	B*b*D	P, WP, XS, X, S, M, B, W		3+*n*
C=-C-1 *ch*	FE	A	oui	7
	B*b*E	P, WP, XS, X, S, M, B, W		3+*n*
D=-D-1 *ch*	FF	A	oui	7
	B*b*F	P, WP, XS, X, S, M, B, W		3+*n*

Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution est le nombre de quartets constituant le champ auquel est appliquée l'instruction.

2.3
Les opérations logiques

■ Somme logique (« OU » logique)

Le « OU » logique, noté « ! », fait appel à une comparaison bit à bit de deux champs de même type appartenant à deux registres distincts. Prenons l'exemple de l'instruction `A=A!B X`. Les champs A(X) et B(X) seront comparés et le résultat sera placé dans A(X). La fonction « OU »

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A!B *ch*	0EF8	A	non	4+*n*
	0E*a*8	P, WP, XS, X, S, M, B, W		
B=B!C *ch*	0EF9	A	non	4+*n*
	0E*a*9	P, WP, XS, X, S, M, B, W		

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
C=C!A *ch*	0EFA	A	non	4+*n*
	0EaA	P, WP, XS, X, S, M, B, W		
D=D!C *ch*	0EFB	A	non	4+*n*
	0EaB	P, WP, XS, X, S, M, B, W		
B=A!B *ch*	0EFC	A	non	4+*n*
	0EaC	P, WP, XS, X, S, M, B, W		
C=B!C *ch*	0EFD	A	non	4+*n*
	0EaD	P, WP, XS, X, S, M, B, W		
A=C!A *ch*	0EFE	A	non	4+*n*
	0EaE	P, WP, XS, X, S, M, B, W		
C=D!C *ch*	0EFF	A	non	4+*n*
	0EaF	P, WP, XS, X, S, M, B, W		

logique a été présentée dans les chapitres d'initiation à la logique. La retenue (« carry bit ») n'est pas modifiée par ces instructions. Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution est le nombre de quartets constituant le champ auquel est appliquée l'instruction.

■ Produit logique (« ET » logique)

Le « ET » logique, ici noté « & », a déjà été expliqué dans les chapitres d'initiation à la logique. Les deux champs donnés en argument à la fonction logique « ET » sont comparés bit par bit. Il s'agit de deux champs de même type appartenant à deux registres distincts. Prenons l'exemple de l'instruction `A=A&B X`. Les champs A(X) et B(X) seront comparés et le résultat sera placé dans A(X).

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=A&B *ch*	0EF0	A	non	4+*n*
	0Ea0	P, WP, XS, X, S, M, B, W		
B=B&C *ch*	0EF1	A	non	4+*n*
	0Ea1	P, WP, XS, X, S, M, B, W		
C=C&A *ch*	0EF2	A	non	4+*n*
	0Ea2	P, WP, XS, X, S, M, B, W		
D=D&C *ch*	0EF3	A	non	4+*n*
	0Ea3	P, WP, XS, X, S, M, B, W		
B=A&B *ch*	0EF4	A	non	4+*n*
	0Ea4	P, WP, XS, X, S, M, B, W		
C=B&C *ch*	0EF5	A	non	4+*n*
	0Ea5	P, WP, XS, X, S, M, B, W		
A=C&A *ch*	0EF6	A	non	4+*n*
	0Ea6	P, WP, XS, X, S, M, B, W		
C=D&C *ch*	0EF7	A	non	4+*n*
	0Ea7	P, WP, XS, X, S, M, B, W		

La retenue (« carry bit ») n'est pas modifiée par ces instructions. Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution est le nombre de quartets constituant le champ auquel est appliquée l'instruction.

2.4
Les décalages et rotations

■ Division et multiplication par 16 (#10h) par décalage d'un quartet

Il existe huit instructions permettant de décaler un registre vers la droite ou vers la gauche. Un décalage d'un quartet vers la gauche correspond à une multiplication par 16 (#10h) alors qu'un décalage d'un quartet vers la droite est une division par 16. Lors d'un décalage vers la gauche, tous les quartets sont décalés d'un rang vers la gauche, le quartet n°#Fh étant perdu alors qu'est inséré un quartet nul à la place de l'ancien quartet n°#0h. De même, lors d'un décalage vers la droite, le quartet n°#0h est perdu alors qu'un quartet nul remplace l'ancien quartet n°#Fh.

Les mnémoniques de ces instructions commencent par le nom du registre concerné (A, B, C ou D) et finissent par l'orientation du décalage (L pour un décalage à gauche, R pour un décalage à droite).

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
ASL *ch*	F0	A	non	7
	B*b*0	P, WP, XS, X, S, M, B, W		3+*n*
BSL *ch*	F1	A	non	7
	B*b*1	P, WP, XS, X, S, M, B, W		3+*n*
CSL *ch*	F2	A	non	7
	B*b*2	P, WP, XS, X, S, M, B, W		3+*n*
DSL *ch*	F3	A	non	7
	B*b*3	P, WP, XS, X, S, M, B, W		3+*n*
ASR *ch*	F4	A	non	7
	B*b*4	P, WP, XS, X, S, M, B, W		3+*n*
BSR *ch*	F5	A	non	7
	B*b*5	P, WP, XS, X, S, M, B, W		3+*n*
CSR *ch*	F6	A	non	7
	B*b*6	P, WP, XS, X, S, M, B, W		3+*n*
DSR *ch*	F7	A	non	7
	B*b*7	P, WP, XS, X, S, M, B, W		3+*n*

Bien que le bit de retenue ne soit pas modifié par ces instructions, un décalage d'un quartet vers la droite fait passer à 1 le bit SB si le quartet perdu sur la droite pendant le décalage est non nul. Le paramètre `n` dont dépend le nombre de cycles nécessaires à l'exécution est le nombre de quartets constituant le champ auquel est appliquée l'instruction.

■ Rotation d'un quartet vers la droite ou vers la gauche

Les noms de ces instructions commencent par la lettre représentant le registre de travail concerné (A, B, C ou D). La troisième lettre désigne le sens de la rotation (L pour une rotation vers la gauche, R pour une rotation vers la droite). Par exemple, l'instruction ordonnant un décalage vers la droite de l'intégralité du registre C est `CSRC`.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
ASLC	810	Tous les quartets	non	21
BSLC	811	Tous les quartets	non	21
CSLC	812	Tous les quartets	non	21
DSLC	813	Tous les quartets	non	21
ASRC	814	Tous les quartets	non	21
BSRC	815	Tous les quartets	non	21
CSRC	816	Tous les quartets	non	21
DSRC	817	Tous les quartets	non	21

Dans le cas d'une rotation vers la droite, le quartet n°#1h devient le quartet n°#0h, l'ancien quartet n°#0h devenant le n°#Fh. Inversement, dans le cas d'une rotation vers la gauche, le quartet n°#Eh devient le quartet #Fh alors que ce dernier devient le quartet n°#0h. Remarquez que contrairement à un décalage, une rotation n'entraîne pas de perte d'information (aucun quartet n'est perdu puisque les quartets sont juste déplacés). Uniquement dans le cas d'une rotation vers la droite, le bit SB est mis à 1 si les quatre bits passant du quartet n°#0h au quartet n°#Fh sont nuls.

■ Décalage d'un bit vers la droite (division par 2)

Tous les bits du registre sont ici décalés d'un rang.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
ASRB	81C	Tous les quartets	non	20
BSRB	81D	Tous les quartets	non	20
CSRB	81E	Tous les quartets	non	20
DSRB	81F	Tous les quartets	non	20

La première lettre du mnémonique représente le registre de travail concerné par l'instruction. Ainsi, BSRB correspond au décalage vers la droite de tous les bits du registre B.

Indépendamment des champs et des quartets, tous les bits d'un registre de travail sont décalés par les instructions ASRB (registre A), BSRB (registre B), CSRB (registre C) et DSRB (registre D). Rappelons qu'un registre de travail comporte 64 bits. Ces bits sont décalés d'un rang vers la droite à chaque appel des instructions que nous venons de citer. Lors du décalage le bit le plus à droite est perdu alors qu'un bit nul est inséré à gauche du registre. Lorsque le bit perdu sur la droite vaut 1, la valeur de SB passe à 1.

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

Les instructions que nous venons de voir décalent bit par bit l'ensemble d'un registre. Cela dit, il est aussi possible de limiter le décalage bit à bit à un champ d'un registre de travail. Les mnémoniques des instructions réalisant cette opération de décalage bit à bit sur un champ sont les mêmes que ceux que nous venons de voir mais sont suivis du suffixe « .F » afin de rappeler qu'ils concernent un champ (« field ») et non un registre. Le décalage bit à bit vers la droite divise par #2h le contenu d'un champ.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
ASRB.F *ch*	819F0	A	non	14
	819*a*0	P, WP, XS, X, S, M, B		9+*n*
BSRB.F *ch*	819F1	A	non	14
	819*a*1	P, WP, XS, X, S, M, B		9+*n*
CSRB.F *ch*	819F2	A	non	14
	819*a*2	P, WP, XS, X, S, M, B		9+*n*
DSRB.F *ch*	819F3	A	non	14
	819*a*3	P, WP, XS, X, S, M, B		9+*n*

Nous n'avons pas fait figurer dans la liste des champs concernés le registre W (le registre dans sa totalité) car il existe des instructions (ASRB, BSRB, etc.) qui se chargent du décalage bit par bit vers la droite de la totalité d'un registre.

Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets sur lequel porte le décalage bit à bit. Bien que le bit de retenue ne soit pas modifié par ces instructions on notera que si le bit perdu vers la droite lors du décalage vaut 1, alors le bit SB passe à 1. Bien entendu, lors du décalage, le bit inséré sur la gauche du champ vaut 0.

Comme nous l'avons dit, un décalage d'un bit vers la droite revient à une division entière par #2h du champ auquel est appliquée l'instruction de décalage. Pour une multiplication par #2h, ne disposant pas d'une fonction de décalage bit à bit vers la gauche, on utilisera une somme. En effet, nous avons vu qu'il est possible d'ajouter un champ à lui-même. Par exemple, `C=C+C A` provoquera la multiplication par #2h du champ A du registre C. Nous n'avons donc pas à regretter l'absence d'une instruction permettant le décalage bit à bit vers la gauche !

15

Instructions : sauvegarde dans un registre et pointeurs

1

Les registres de sauvegarde

1.1

Copies vers et depuis un registre de sauvegarde

Ici encore nous remarquerons que les registres de travail A et C sont les « registres rois » auxquels tout est permis. Pour effectuer une copie entre un registre de sauvegarde et un registre de travail tels que B ou D, il est nécessaire de faire transiter les quartets par A ou C.

Il existe cinq registres de sauvegarde : R0, R1, R2, R3 et R4. Il n'existe pas de priorité particulière entre eux, ces cinq registres sont donc équivalents.

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

Les instructions de type R_n=A ou R_n=C (où R_n est un registre de sauvegarde quelconque) sont pour nous l'occasion de réviser les affectations. Le signe « = » est ici un opérateur d'affectation. En effet, à son membre gauche, ici R_n, est affectée la valeur de son membre droit qui est ici le registre A ou le registre C. Inversement, C=R_n affectera au registre de travail C le contenu du registre de sauvegarde spécifié.

Retenez donc que lors d'une affectation le membre gauche adopte la valeur du membre droit qui, lui, demeure inchangé. Des mnémoniques tels que `A=A+B` ou `A=B+A` sont équivalents (« + » est commutatif) alors que R_n=A ou A=R_n ne sont pas du tout équivalents puisque le sens du transfert de données est changé. Notez que si R_n=A ou A=R_n ne sont pas équivalents, `?A=B` et `?B=A`, eux, sont équivalents car il s'agit dans ce dernier cas d'un test et non d'une affectation... Le signe « = » autorise une inversion des membres droit et gauche lorsqu'il sert de comparateur pour un test (`?A=B`, par exemple) mais une inversion des membres autour de « = » lors d'une affectation inverse le processus d'affectation (les deux opérations ne sont pas du tout équivalentes).

Les instructions ci-dessous sont composées de la façon suivante : à gauche du signe « = » se trouve le nom du registre où sera copié le contenu du registre dont le nom figure à droite du signe « = ». Ainsi, `R0=C` entraîne la copie dans le registre R0 de l'intégralité du registre C.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
R0=A	100	Tous les quartets	non	19
R1=A	101	Tous les quartets	non	19
R2=A	102	Tous les quartets	non	19
R3=A	103	Tous les quartets	non	19
R4=A	104	Tous les quartets	non	19
R0=C	108	Tous les quartets	non	19
R1=C	109	Tous les quartets	non	19
R2=C	10A	Tous les quartets	non	19
R3=C	10B	Tous les quartets	non	19
R4=C	10C	Tous les quartets	non	19

■ Copie d'un champ d'un registre de travail vers le même champ d'un registre de sauvegarde

Il est possible de limiter la copie d'un registre de travail vers un registre de sauvegarde à un champ spécifié. Bien que ce ne soit pas tout à fait exact, on peut considérer que les registres de sauvegarde sont découpés en champs selon le même schéma que les registres de travail.

Comme pour la précédente série d'instructions, on trouve dans le mnémonique un signe « = » d'affectation encadré, à sa gauche par le nom du registre de sauvegarde qui doit recevoir la copie , et à sa droite par le nom du registre de travail contenant l'original à copier. Le mnémonique est complété par le suffixe « `.F` » rappelant que seul un champ précis (*ch*) est concerné par l'opération de copie.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
R0=A.F *ch*	81AF00	A	non	9+*n*
	81A*a*00	P, WP, XS, X, S, M, B, W		
R1=A.F *ch*	81AF01	A	non	9+*n*
	81A*a*01	P, WP, XS, X, S, M, B, W		
R2=A.F *ch*	81AF02	A	non	9+*n*
	81A*a*02	P, WP, XS, X, S, M, B, W		
R3=A.F *ch*	81AF03	A	non	9+*n*
	81A*a*03	P, WP, XS, X, S, M, B, W		
R4=A.F *ch*	81AF04	A	non	9+*n*
	81A*a*04	P, WP, XS, X, S, M, B, W		
R0=C.F *ch*	81AF08	A	non	9+*n*
	81A*a*08	P, WP, XS, X, S, M, B, W		
R1=C.F *ch*	81AF09	A	non	9+*n*
	81A*a*09	P, WP, XS, X, S, M, B, W		
R2=C.F *ch*	81AF0A	A	non	9+*n*
	81A*a*0A	P, WP, XS, X, S, M, B, W		
R3=C.F *ch*	81AF0B	A	non	9+*n*
	81A*a*0B	P, WP, XS, X, S, M, B, W		
R4=C.F *ch*	81AF0C	A	non	9+*n*
	81A*a*0C	P, WP, XS, X, S, M, B, W		

Par exemple, la copie du champ A du registre C dans le registre R3 est codée `81AF0B` et correspond au mnémonique `R3=C.F A`. De même, la copie dans le registre de sauvegarde R4 du champ X du registre A est codée `81A304` et correspond au mnémonique `R4=A.F X`.

Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets transférés depuis le registre de travail vers le registre de sauvegarde.

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

Les registres de sauvegarde R0, R1, R2, R3 et R4 ne permettent pas de traiter les valeurs qu'ils contiennent. Il est donc indispensable de pouvoir rapatrier vers un registre de travail (registre de calcul) le contenu d'un registre de sauvegarde.

Il existe donc dix instructions permettant la copie de l'intégralité de l'un des cinq registres de sauvegarde vers le registre A ou le registre C. On trouve dans les mnémoniques de ces instructions un signe « = » d'affectation. A la gauche de celui-ci se trouve le nom du registre (A ou C) devant accueillir le contenu du registre de sauvegarde dont le nom est précisé à droite de ce même signe « = ».

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=R0	110	W	non	19
A=R1	111	W	non	19
A=R2	112	W	non	19
A=R3	113	W	non	19
A=R4	114	W	non	19
C=R0	118	W	non	19
C=R1	119	W	non	19
C=R2	11A	W	non	19
C=R3	11B	W	non	19
C=R4	11C	W	non	19

Par exemple, `A=R3` transfère l'intégralité du registre de sauvegarde R3 vers le registre de travail A. Ce mnémonique correspond au code `113`.

■ Copie d'un champ d'un registre de sauvegarde vers le même champ d'un registre de travail

Il est possible de limiter la copie d'un registre de sauvegarde vers un registre de travail à un champ spécifié. Comme pour la précédente série d'instructions, on trouve dans le mnémonique un signe « = » d'affectation encadré, à sa gauche par le nom du registre de travail qui doit recevoir la copie, et à sa droite par le nom du registre de sauvegarde contenant l'original à copier. Le mnémonique est complété par le suffixe « `.F` » rappelant que seul un champ précis (*ch*) est concerné par l'opération de copie. Toutefois, l'application au champ W est ici inintéressante.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=R0.F *ch*	81AF10	A	non	9+*n*
	81Aa10	P, WP, XS, X, S, M, B, W		
A=R1.F *ch*	81AF11	A	non	9+*n*
	81Aa11	P, WP, XS, X, S, M, B, W		
A=R2.F *ch*	81AF12	A	non	9+*n*
	81Aa12	P, WP, XS, X, S, M, B, W		
A=R3.F *ch*	81AF13	A	non	9+*n*
	81Aa13	P, WP, XS, X, S, M, B, W		
A=R4.F *ch*	81AF14	A	non	9+*n*
	81Aa14	P, WP, XS, X, S, M, B, W		
C=R0.F *ch*	81AF18	A	non	9+*n*
	81Aa18	P, WP, XS, X, S, M, B, W		
C=R1.F *ch*	81AF19	A	non	9+*n*
	81Aa19	P, WP, XS, X, S, M, B, W		
C=R2.F *ch*	81AF1A	A	non	9+*n*
	81Aa1A	P, WP, XS, X, S, M, B, W		

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
C=R3.F *ch*	81AF1B	A	non	*9+n*
	81Aa1B	P, WP, XS, X, S, M, B, W		
C=R4.F *ch*	81AF1C	A	non	*9+n*
	81Aa1C	P, WP, XS, X, S, M, B, W		

Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets transférés depuis le registre de sauvegarde vers le registre de travail.

Par exemple, le copie du registre R4 vers le registre C limitée au champ WP correspond au mnémonique `C=R4.F WP` qui est codé `81A11C` (puisque l'élément variable `a` vaut #1h lorsqu'il doit désigner le champ WP).

1.2
Echanges avec un registre de sauvegarde

■ Echange du contenu d'un registre de travail avec celui d'un registre de sauvegarde

Vous pouvez demander l'échange du contenu d'un registre de travail avec celui d'un registre de sauvegarde. Les trois premiers caractères des mnémoniques de ces instructions représentent les deux registres dont les contenus sont échangés. Attention ! A chacune de ces instructions peuvent correspondre deux mnémoniques. En effet, `AR0EX` est absolument équivalent à `R0AEX`. Cela dit, la première forme est, et de loin, la plus employée par les programmeurs.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
AR0EX	120	W	non	19
AR1EX	121	W	non	19
AR2EX	122	W	non	19
AR3EX	123	W	non	19
AR4EX	124	W	non	19
CR0EX	128	W	non	19
CR1EX	129	W	non	19
CR2EX	12A	W	non	19
CR3EX	12B	W	non	19
CR4EX	12C	W	non	19

Par exemple, CR1EX correspond au transfert du contenu de C vers R1 ainsi qu'au transfert simultané du contenu de R1 vers C.

■ Echange du contenu d'un champ d'un registre de travail avec le contenu du même champ d'un registre de sauvegarde

Nous venons de voir qu'un registre de travail et un registre de sauvegarde peuvent échanger leurs contenus. Les instructions que nous allons maintenant présenter permettent de limiter cet échange à un champ donné. Les mnémoniques de la précédente série sont complétés par le suffixe « .F » précisant que l'action de l'instruction est limitée à un champ (« field »). Le nom du champ auquel est appliquée l'instruction est notée *ch* à la suite de la partie principale du mnémonique.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
AR0EX.F *ch*	81AF20	A	non	9+*n*
	81A*a*20	P, WP, XS, X, S, M, B, W		
AR1EX.F *ch*	81AF21	A	non	9+*n*
	81A*a*21	P, WP, XS, X, S, M, B, W		
AR2EX.F *ch*	81AF22	A	non	9+*n*
	81A*a*22	P, WP, XS, X, S, M, B, W		
AR3EX.F *ch*	81AF23	A	non	9+*n*
	81A*a*23	P, WP, XS, X, S, M, B, W		
AR4EX.F *ch*	81AF24	A	non	9+*n*
	81A*a*24	P, WP, XS, X, S, M, B, W		
CR0EX.F *ch*	81AF28	A	non	9+*n*
	81A*a*28	P, WP, XS, X, S, M, B, W		
CR1EX.F *ch*	81AF29	A	non	9+*n*
	81A*a*29	P, WP, XS, X, S, M, B, W		
CR2EX.F *ch*	81AF2A	A	non	9+*n*
	81A*a*2A	P, WP, XS, X, S, M, B, W		
CR3EX.F *ch*	81AF2B	A	non	9+*n*
	81A*a*2B	P, WP, XS, X, S, M, B, W		
AR4EX.F *ch*	81AF2C	A	non	9+*n*
	81A*a*2C	P, WP, XS, X, S, M, B, W		

L'élément variable *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets transférés entre les registres.

Bien que les mnémoniques cités soient les plus couramment employés et sachant que les trois premiers caractères des mnémoniques désignent ici les deux registres intervenant dans l'échange, ils existent aussi d'autres mnémoniques pour ces instructions. Par exemple, `AR4EX.F` peut aussi être noté `R4AEX.F`.

Considérons par exemple l'échange des registres R1 et C limité à leurs champs X. Le mnémonique utilisé est `CR1EX.F X`. Le code correspondant est `81A329`.

2
Les pointeurs

Les registres D0 et D1 ont une capacité de 5 quartets, ce qui correspond au format des adresses en mémoire utilisé sur HP-48. D0 et D1 sont chargés de désigner un point précis en mémoire (en fait, une adresse), c'est pourquoi on les appelle les « registres pointeurs » ou, plus simplement, les « pointeurs ». Nous allons à nouveau constater que les registres de travail A et C ont un rôle essentiel puisqu'ils sont les seuls capables de transférer des quartets vers et depuis les registres D0 et D1.

2.1
Affectation de données à D0 et à D1

■ Chargement de 2, 4 ou 5 quartets dans D0 ou D1

Il est possible d'affecter une valeur de votre choix aux pointeurs D0 et D1. Comme nous l'avons déjà vu, les instructions `LAHEX` et `LCHEX` sont utilisées pour charger une valeur de votre choix dans les registres A et C. Ici, des instructions du type `D0=(`*n*`)` ou `D1=(`*n*`)` (où *n* vaut 2, 4 ou 5) vont permettre d'affecter une valeur de votre choix aux deux premiers, aux quatre premiers ou aux cinq quartets de D0 ou de D1.

Notez que l'affectation de deux, quatre ou cinq quartets de votre choix à un registre pointeur commence toujours par le premier quartet dudit registre pointeur. Le chargement commence donc toujours par le quartet de poids le plus faible (quartet de droite dans la représentation courante).

Par exemple, l'instruction `D0(4)` *pqrs* (où *p*, *q*, *r* et *s* sont des quartets arbitraires) que l'on code `1A`*srqp* entraîne l'affectation de la valeur *s*

au quartet n°#0h, de la valeur *r* au quartet n°#1h, de la valeur *q* au quartet n°#2h et de la valeur *p* au quartet n°#3h. Le quartet n°#4h (le cinquième et dernier quartet du registre pointeur) n'est pas modifié par `D0=(4) pqrs` mais le serait par une instruction du type `D0=(5) pqrst`.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
D0=(2) *pq*	19*qp*	2 premiers quartets	non	4
D0=(4) *pqrs*	1A*srqp*	4 premiers quartets	non	6
D0=(5) *pqrst*	1B*tsrqp*	tout le registre	non	7
D1=(2) *pq*	1D*qp*	2 premiers quartets	non	4
D1=(4) *pqrs*	1E*srqp*	4 premiers quartets	non	6
D1=(5) *pqrst*	1F*tsrqp*	tout le registre	non	7

Dans le mnémonique, le nombre entre parenthèses est le nombre de quartets à modifier au sein du pointeur alors que *p*, *q*, *r*, *s* et *t* sont les valeurs (ce sont des nombres hexadécimaux compris entre #0h et #Fh) à affecter aux quartets du pointeur. Notez l'inversion du champ contenant les quartets à charger.

■ Addition ou soustraction d'une constante à D0 ou à D1

Il est possible de modifier le contenu de D0 ou de D1 sans avoir recours aux instructions d'affectation de 2, 4 ou 5 quartets que nous venons de décrire. En effet, vous pouvez ajouter ou soustraire à D0 ou à D1 une valeur comprise entre #1h et #10h (16_{10}). Pour cela il suffit de préciser la valeur d'un quartet *n* qui complétera les mnémoniques des instructions d'addition ou de soustraction à D0 ou à D1. Comme tous les quartets, *n* peut varier entre #0h et #Fh. Les instructions d'addition ou de soustractions travaillent à partir de *n*+1. Donc, pour ajouter #1h à la valeur d'un pointeur il faudra affecter la valeur #0h au quartet *n*. C'est ainsi qu'on peut augmenter le contenu d'un pointeur de #10h alors que *n* est limité à #Fh.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
D0=D0+ *(n+1)*	16*n*	tout le registre	oui	7
D0=D0- *(n+1)*	18*n*	tout le registre	oui	7
D1=D1+ *(n+1)*	17*n*	tout le registre	oui	7
D1=D1- *(n+1)*	1C*n*	tout le registre	oui	7

Par exemple, pour ajouter #2h à D0 on utilisera `D0=D0+ (n+1)`. Ici, *n*+1 vaut #2h, donc *n* vaut #1h. `D0=D0+ (1+1)` est codé `161`.

Dans les mnémoniques, le nom du registre concerné (D0 ou D1) vous est rappelé et est suivi du type d'opération effectuée (addition ou soustrac-

tion). L'instruction peut concerner tout le registre pointeur. En effet, #1h ajouté à #0FFFFh vaut #10000h ce qui modifie tous les quartets du registre pointeur.

■ Copie du champ A de A ou de C vers D0 ou D1

Le champ A des registres de travail comporte 5 quartets, il est donc prédisposé à l'accueil d'adresses (codées sur 5 quartets). Les champs A des registres de travail A et C peuvent transférer leurs contenus vers l'un des registres pointeurs D0 ou D1.

Par exemple, `D0=C` copie le champ A du registre C dans le pointeur D0. Bien qu'aucun champ ne soit précisé, les instructions `D0=A`, `D1=A`, `D0=C` et `D1=C` ne concernent que le champ A des registres A et C.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
D0=A	130	D0 en entier et champ A de A	non	8
D0=C	134	D0 en entier et champ A de C	non	8
D1=A	131	D1 en entier et champ A de A	non	8
D1=C	135	D1 en entier et champ A de C	non	8

■ Copie des quatre premiers quartets de A ou C dans D0 ou D1

Les instructions `D0=A`, `D0=C`, `D1=A` et `D1=C` permettent d'affecter à un pointeur D0 et D1 le contenu du champ A du registre A ou du registre C. Les instructions que nous allons découvrir maintenant permettent le même type d'échange, à ceci près, qu'on se limite aux quatre premiers quartets. Il s'agit donc ici du transfert des quatre premiers quartets du registre A ou du registre C vers l'un des registres pointeurs. Autrement dit, les quatre quartets de poids faible du registre de travail migrent vers les quatre quartets de poids faible du pointeur, alors que le quartet de poids fort de celui-ci demeure inchangé. Les mnémoniques utilisés sont presque les mêmes que ceux que nous venons de voir, seul un « s » leur a été ajouté. Cette série d'instructions s'emploie comme la précédente, la seule différence étant que le quartet de poids fort du pointeur reste inchangé.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
D0=AS	138	4 premiers quartets	non	7
D0=CS	13C	4 premiers quartets	non	7
D1=AS	139	4 premiers quartets	non	7
D1=CS	13D	4 premiers quartets	non	7

■ Echange des contenus des champs A avec les registres D0 ou D1

Comme nous l'avons déjà dit, les champs A des registres de travail ont la même capacité que les registres pointeurs D0 et D1. Il existe quatre instructions permettant d'échanger le contenu d'un registre pointeur avec le contenu du champ A du registre A ou C. Comme toute les instructions relatives aux échanges de contenus, `AD0EX`, `AD1EX`, `CD0EX` et `CD1EX` peuvent être notées de deux façons différentes. Les notations `D0AEX`, `D1AEX`, `D0AEX` et `D1CEX` sont donc acceptables mais peu utilisées et donc, déconseillées.

Les trois premières lettres du mnémonique correspondent aux noms des deux registres concernés par l'échange. Par exemple, `AD1EX` échange le contenu de D1 avec celui du champ A du registre A. Bien qu'aucun champ ne soit mentionné, ces instructions ne concernent que le champ A des registres de travail mais modifient l'intégralité des pointeurs D0 et D1.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
AD0EX	132	D0 en entier et champ A de A	non	8
AD1EX	133	D1 en entier et champ A de A	non	8
CD0EX	136	D0 en entier et champ A de C	non	8
CD1EX	137	D1 en entier et champ A de C	non	8

■ Echange des contenus des quatre premiers quartets de A ou de C avec les quatre premiers quartets du registre D0 ou D1

Nous venons de décrire l'échange du contenu d'un pointeur avec le contenu du champ A d'un registre de travail (A ou C). Les instructions que nous allons décrire maintenant permettent de limiter cette opération aux quatre premiers quartets des registres concernés en laissant donc inchangé le quartet de poids fort du pointeur (D0 ou D1). Comme pour la précédente série d'instructions, chaque mnémonique peut prendre deux formes. Ainsi, `AD0XS` peut aussi être notée `D0AXS` mais ce type de notation inhabituelle est déconseillé. Les trois premiers caractères du mnémoniques représentent les registres concernés par l'échange.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
AD0XS	13A	4 premiers quartets	non	7
AD1XS	13B	4 premiers quartets	non	7
CD0XS	13E	4 premiers quartets	non	7
CD1XS	13F	4 premiers quartets	non	7

Ces instructions sont comparables à celles de la précédente série. La seule différence réside dans le fait que seuls les quatre premiers quartets des registres sont concernés.

2.2
Lecture et écriture en mémoire

Nous avons vu comment modifier le contenu des registres pointeurs D0 et D1, en particulier, nous avons vu comment charger une valeur dans D0 ou D1 en « important » celle-ci depuis un registre de travail (A ou C). Il ne faut pourtant pas perdre de vue que le rôle principal de D0 et de D1 consiste à « pointer » sur une adresse en mémoire. D0 et D1 servent donc à désigner l'emplacement en mémoire où doit être lue ou écrite une donnée. Les registres de travail A et C - encore eux ! - peuvent envoyer leurs données vers l'adresse contenue dans l'un des registres pointeurs, il s'agit alors d'une opération d'écriture en mémoire. Inversement, A et C peuvent recevoir les données se trouvant à l'adresse désignée par D0 ou D1, il s'agit alors d'une opération de lecture en mémoire.

■ Lecture en mémoire et chargement d'un champ de A ou de C

Sachant qu'en anglais une « data » est une donnée, le mnémonique `A=DAT0 M` peut être traduit par « *copier dans le champ M du registre A les données se trouvant en mémoire à partir de l'adresse désignée par le pointeur D0* ». La famille d'instructions dont il va maintenant être question sert

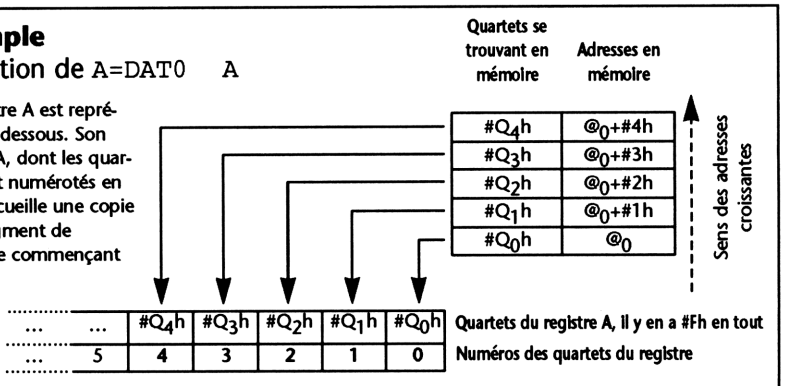
Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=DAT0 ***ch***	142	A	non	18
	14A	B		15
	152*a*	P, WP, XS, X, S, M, B, W		17+*n*
A=DAT1 ***ch***	143	A	non	18
	14B	B		15
	153*a*	P, WP, XS, X, S, M, B, W		17+*n*
C=DAT0 ***ch***	146	A	non	18
	14E	B		15
	156*a*	P, WP, XS, X, S, M, B, W		17+*n*
C=DAT1 ***ch***	147	A	non	18
	14F	B		15
	157*a*	P, WP, XS, X, S, M, B, W		17+*n*

donc à copier un segment de mémoire dans un champ du registre A ou du registre C. On affecte donc à A ou à C une valeur en provenance de la mémoire. Le schéma ci-dessous résume le principe de la lecture en mémoire.

Exemple

Exécution de `A=DAT0 A`

Le registre A est représenté ci-dessous. Son champ A, dont les quartets sont numérotés en gras, accueille une copie d'un segment de mémoire commençant en $@_0$.



$@_0$ est l'adresse contenue dans le pointeur D0 utilisé par l'instruction en cours d'exécution. $\#Q_0h$ est le quartet qui se trouve en mémoire à l'adresse $@_0$. $\#Q_1h$, $\#Q_2h$, et les quartets suivants se trouvent aux adresses $@_0+\#1h$, $@_0+\#2h$ et suivantes. La lecture s'effectue dans le sens adresses croissantes. Le champ du registre de travail qui doit recevoir les quartets lus est rempli à partir de son quartet de poids le plus faible.

Dans les mnémoniques de ces instructions, le nom du registre de travail dont l'un des champs doit recevoir les données est à gauche du signe « = », « DAT0 » signifie qu'il s'agit d'un transfert de données (« data ») depuis l'adresse précisée par le pointeur D0 (de même, « DAT1 » indique que le pointeur D1 est utilisé). Attention ! Il ne s'agit en aucun cas du transfert vers un registre de travail du contenu d'un pointeur mais du transfert des données se trouvant en mémoire à l'adresse qu'indique le pointeur. En fait, il s'agit des données se trouvant en mémoire *à partir* de l'adresse indiquée par le pointeur...

Bien entendu, il est indispensable de placer au préalable l'adresse où doit s'effectuer la lecture dans le pointeur (D0 ou D1) qui sera utilisé par l'instruction de lecture choisie (`A=DAT0`, `A=DAT1`, `C=DAT0` et `C=DAT1`).

On dispose pour cette série d'instructions de codes simplifiés qui sont propres à l'application de la fonction au champ B du registre A ou C. Bien que cela soit inintéressant (car plus long), on peut aussi utiliser la forme générale du code en remplaçant *a* par la valeur #6h qu'il prend lorsqu'il doit représenter le champ B. Le champ B a une capacité de deux quartets, il accueille donc un octet (d'où son nom puisqu'en anglais, un octet est appelé « byte »). Un caractère est codé sur deux quartets (voir le chapitre consacré à l'ASCII), lorsqu'un caractère doit être lu en mémoire et transféré vers un registre, il se place logiquement dans le registre B. Dans le

cadre de la lecture en mémoire de caractères ASCII il est donc bien utile de disposer de codes hexadécimaux simplifiés se contentant de charger le champ B avec les deux quartets se trouvant à partir de l'adresse désignée par un pointeur.

Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution est le nombre de quartets constituant le champ auquel est appliquée l'instruction.

Nous aborderons bientôt des instructions d'écriture dans la mémoire (transfert depuis un registre vers la mémoire) qui fonctionnent de façon similaire.

■ Lecture en mémoire et chargement d'un nombre déterminé de quartets dans A ou C

Il est aussi possible de lire à partir de l'adresse contenue dans le pointeur utilisé (D0 ou D1) un nombre de quartets compris entre #1h et #10h, ces quartets étant copiés dans A ou dans C à partir du quartet n°#0h de ces registres. Le quartet variable n varie entre #0h et #Fh, n+1 quartets seront copiés dans le registre de travail à partir de l'adresse désignée par le pointeur utilisé. On peut donc transférer de #1h à #10h quartets. Par exemple, considérons `C=DAT1 (2+1)`. « C » indique que les données seront transférées vers le registre C, « DAT1 » signifie que les données transférées sont en provenance de l'adresse désignée par D1 alors que « (2+1) » signale que #3h quartets seront transférés (n vaut ici #2h).

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=DAT0 *(n+1)*	15A*n*	*n*+1 quartets	non	16+*n*+1
A=DAT1 *(n+1)*	15B*n*	*n*+1 quartets	non	16+*n*+1
C=DAT0 *(n+1)*	15E*n*	*n*+1 quartets	non	16+*n*+1
C=DAT1 *(n+1)*	15F*n*	*n*+1 quartets	non	16+*n*+1

■ Ecriture en mémoire du contenu du registre A ou du registre C

Nous avons vu comment lire des quartets en mémoire pour les placer dans un champ du registre A ou du registre C. Inversement, il est possible d'écrire en mémoire le contenu des registres A et C. Les mnémoniques utilisés sont de la forme `DAT0=A` *ch*. A gauche du signe « = » est indiquée la zone en mémoire où sera copié le registre A ou le registre C. DAT0 représente la portion de mémoire commençant à l'adresse contenue dans D0, de même DAT1 représente la portion de mémoire commençant à

l'adresse contenue dans le pointeur D1. A droite du signe « = » est précisé le registre dont le contenu sera copié. Il ne peut s'agir que du registre A ou du registre C. Le champ *ch* du registre concerné par la copie est précisé juste après le nom du registre. Ainsi, `DAT0=A  A` correspond à la copie du champ A du registre A à partir de l'adresse contenue dans le registre D0.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
DAT0=A *ch*	140	A	non	17
	148	B		14
	150*a*	P, WP, XS, X, S, M, B, W		16+*n*
DAT1=A *ch*	141	A	non	17
	149	B		14
	151*a*	P, WP, XS, X, S, M, B, W		16+*n*
DAT0=C *ch*	144	A	non	17
	14C	B		14
	154*a*	P, WP, XS, X, S, M, B, W		16+*n*
DAT1=C *ch*	145	A	non	17
	14D	B		14
	155*a*	P, WP, XS, X, S, M, B, W		16+*n*

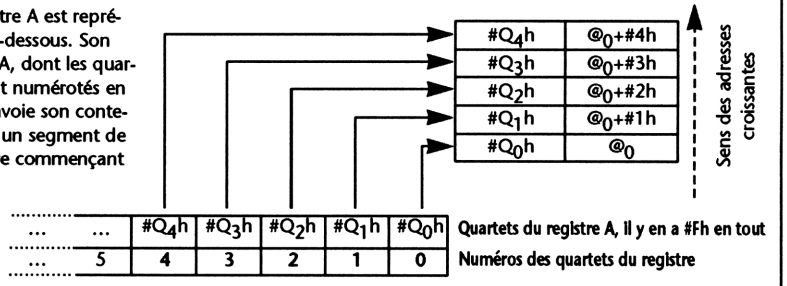
Le paramètre *n* dont dépend le nombre de cycles nécessaires à l'exécution est le nombre de quartets constituant le champ auquel est appliquée l'instruction.

Ici aussi l'existence d'un code spécifique et simplifié pour les opérations concernant le champ B se justifie par la facilité qu'il procure pour les manipulations de chaînes de caractères.

Exemple

Exécution de `DAT0=A  A`

Le registre A est représenté ci-dessous. Son champ A, dont les quartets sont numérotés en gras, envoie son contenu vers un segment de mémoire commençant en $@_0$.



$@_0$ est l'adresse contenue dans le pointeur D0 utilisé par l'instruction en cours d'exécution. Les contenus des quartets n°0, n°1, n°2, n°3 et n°4 (on se limite ici au champ A) sont copiés dans la mémoire vive à partir de l'adresse $@_0$ (puis dans le sens des adresses croissantes). #Q_0h, #Q_1h, #Q_2h, #Q_3h et #Q_4h sont les cinq quartets du registre A (le champ A du registre A) qui sont transférés vers la mémoire vive. Il s'agit ici d'une opération d'écriture en mémoire.

■ Ecriture en mémoire d'un nombre déterminé de quartets en provenance de A ou de C

Quatre instructions nous permettent d'écrire de 1 à 16 quartets en mémoire, et ce, en une seule opération. La copie des quartets commence à partir du premier quartet du registre de travail considéré (quartet n°#0h de A ou de C). Le nombre de quartets copiés est défini à partir du quartet variable *n* qui varie entre #0h et #Fh. Le nombre de quartets effectivement copiés est égal à *n*+1. Ainsi si nous souhaitons copier trois quartets, il faut affecter à *n* la valeur #2h. L'écriture en mémoire est réalisée à partir de l'adresse placée dans l'un des registres pointeurs, D0 ou D1 selon votre choix.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
DAT0=A *(n+1)*	158*n*	*n*+1 quartets	non	15+*n*+1
DAT1=A *(n+1)*	159*n*	*n*+1 quartets	non	15+*n*+1
DAT0=C *(n+1)*	15C*n*	*n*+1 quartets	non	15+*n*+1
DAT1=C *(n+1)*	15D*n*	*n*+1 quartets	non	15+*n*+1

Dans le mnémonique, le « 0 » ou le « 1 » indique que l'on utilise l'adresse contenue dans D0 ou celle contenue dans D1. A droite du signe « = » se trouve le nom du registre dont *n*+1 quartets vont être copiés en mémoire à partir de l'adresse contenue dans le registre pointeur choisi (D0 ou D1 selon votre choix).

Par exemple, si nous souhaitons copier les six premiers quartets du registre C à l'adresse que désigne le registre pointeur D1, le quartet variable *n* vaudra #5h. Le mnémonique de l'instruction utilisée est ici `DAT1=C (5+1)` que nous coderons `15D5`.

L'élément variable *n* dont dépend le nombre de cycles nécessaires à l'exécution correspond au nombre de quartets copiés moins un.

16

Instructions : sauts et tests

1 Les sauts

Chaque instruction d'un programme écrit en assembleur se trouve à une adresse précise en mémoire. Un programme (sans ses éventuels sous-programmes) se trouve « en un seul morceau » en mémoire. Admettons que la première instruction du programme comporte 5 quartets et qu'elle se trouve à l'adresse $@_0$, alors l'instruction suivante se trouve à l'adresse $@_0$+#5h. Par suite, si un programme est constitué de #*n*h quartets et si sa première instruction se trouve à l'adresse $@_0$, alors le dernier quartet de la dernière instruction du programme se trouve à l'adresse $@_0$+#(*n*-1)h.

Dans le cas général, les instructions constituant un programme sont exécutées « en séquence », autrement dit, les unes après les autres, on parle alors d'exécution séquentielle. Un « saut » permet de bousculer cet ordre et de faire en sorte que l'exécution reprenne à partir de n'importe quel point du programme. Il s'agit alors d'un « branchement » de l'exécution. Une instruction de saut détourne donc l'exécution vers une instruction autre que celle qui la suit. L'instruction exécutée à la suite d'un saut est appelée « point de branchement ».

Le PC (« Program Counter ») est un registre très particulier qui contient à chaque instant l'adresse de la prochaine instruction à exécuter. En modifiant le contenu du PC, on modifie l'ordre d'exécution des instructions. D0 et D1 sont des pointeurs utilisés, par exemple, pour l'échange de quartets entre la mémoire et les registres A et C. PC est aussi un pointeur, mais

d'un type totalement différent puisque l'adresse qu'il désigne en mémoire est celle de la prochaine instruction à traiter.

Il existe trois grandes catégories de sauts en assembleur sur HP-48 :

- les sauts relatifs permettent de sauter un nombre de quartets déterminés à partir de l'instruction de saut. Selon le cas il est ainsi possible de reprendre l'exécution *n* quartets avant ou après l'instruction de saut relatif. On se déplace alors à l'intérieur du programme sans avoir à tenir compte de l'adresse en mémoire à laquelle le programme a été placé. Retenez qu'un saut relatif s'effectue en précisant un décalage d'adresses (offset) et non pas une adresse,
- les sauts absolus permettent de diriger l'exécution vers une adresse précise en mémoire. Il suffit donc d'indiquer une adresse (codée sur 5 quartets) pour que l'exécution reprennent à partir de cette adresse. Attention ! Il faut que la mémoire (morte, vive, entrées/sorties, etc.) à laquelle vous faites appel soit visible,
- les sauts indirects consistent en un saut absolu vers une adresse en mémoire. A cette adresse sera lue une autre adresse. C'est vers cette seconde adresse que sera « branchée » l'exécution.

Lors d'un saut relatif l'exécution reprend après l'application d'un décalage d'adresses au contenu du registre PC. Ce décalage peut être positif, on s'éloigne alors vers la fin du programme. Inversement, si l'on souhaite revenir vers le début du programme, le décalage d'adresse doit être négatif puisque l'adresse à partir de laquelle l'exécution doit reprendre est inférieure à celle de l'instruction commandant le saut relatif. Le codage d'un nombre positif en hexadécimal ne pose pas de problème particulier... Quand on connaît l'hexadécimal ! Par contre, le codage d'une valeur négative ne peut pas s'obtenir par simple ajout du signe « - » avant la valeur hexadécimale. Nous sommes ici obligés de recourir au complément à 2 pour coder la valeur du décalage négatif. Pour obtenir ce complément à 2, on code normalement en hexadécimal la valeur absolue de la valeur négative, puis pour obtenir le complément à 2, il suffit de calculer à l'aide de la HP-48 `#0h - #`*`valeur_absolue`*`h`. On ne conservera sur la droite du résultat obtenu que le nombre de quartets qui nous intéressent en ignorant tous les #Fh inutiles se trouvant sur la gauche du résultat utile.

Lors de la rédaction du programme à l'aide de mnémoniques, il est utile de représenter chaque point de branchement à l'aide d'une étiquette (ou label). On place ces labels entre deux mnémoniques, les points de branchement qu'ils représentent provoquent, en cas de saut, l'exécution des instructions placées immédiatement après le label. Notez que les labels ne sont qu'une facilité que se donne le programmeur qui devra compter les

quartets séparant l'instruction de saut et le label afin d'ordonner un saut relatif. Cela dit, les assembleurs (à ne pas confondre avec les programmes écrit en langage assembleur), génèrent un code hexadécimal (ou même un code exécutable) directement à partir des mnémoniques. Les assembleurs autorisent l'emplois de labels. C'est là l'une des principales facilités qu'ils offrent au programmeur en langage assembleur qui n'a ainsi plus à transformer ses mnémoniques en codes ni à compter les quartets pour ordonner un saut relatif.

Vous trouverez la mention `label` comme paramètre de certains mnémoniques. Pendant la rédaction de votre programme, placez des labels aux points de branchement de votre programme. En effet, le paramètre `label` correspond simplement au point de branchement vers lequel l'instruction de saut branche l'exécution. Un assembleur se chargera de traiter vos labels en calculant les décalages d'adresses nécessaires, cela dit, si vous codez vous-mêmes vos programmes – ce qui est d'ailleurs conseillé aux débutants – il vous faudra calculer vous-mêmes les décalages d'adresses lorsque vous transformerez vos mnémoniques associés à des labels en des codes hexadécimaux. Lors d'un saut on pourra faire suivre le mnémonique du nom du label ou du décalage d'adresses (ou de l'adresse pour les sauts absolus). Par exemple `GOC nom_du_label` et `GOC pq` (où les quartets #*p*h et #*q*h codent le décalage) sont équivalents et correspondent tous les deux au code hexadécimal `4qp` (remarquez l'inversion des quartets codant le décalage). A vous de choisir la notation qui vous convient le mieux. Quoi qu'il en soit, une présentation commode de votre programme consiste à placer dans une colonne vos mnémoniques et, en marge de ceux-ci, les labels représentant les points de branchements. Au fur et à mesure du codage, vous remplirez une colonne « codes » et une colonne « adresses ». Cette dernière contient les adresses des différentes instructions du programme, la première instruction ayant l'adresse #0h, on peut ainsi calculer les décalages d'adresses.

Adresses	Labels	Mnémoniques	Codes
#112h		**GOC Label_1**	4qp
#115h		*Mnémonique...*	*XXX*
#118h		*Mnémonique...*	*XX*
#11Ah		*Mnémonique...*	*XXX*
	Label_1		
#11Dh		*Mnémonique...*	**XXXX*

Dans l'exemple ci-dessus `GOC Label_1` branche l'exécution sur le point de branchement correspondant à `Label_1`. Autrement dit, après le saut, l'exécution reprendra à partir du quartet représenté par « * » (les « *X* » sont des quartets d'instructions quelconques). `4qp` est le codage de

GOC pq qui correspond à un saut relatif de #*pq*h quartets. Pour connaître le nombre de quartets à sauter, on calcule $@_{label}$-$@_{saut}$-1 où $@_{label}$ est l'adresse du point de branchement (le quartet noté « * » dans notre exemple) et $@_{saut}$ l'adresse de l'instruction de saut. Ce calcul est valable pour un saut vers la fin programme (le décalage d'adresses est positif) et pour les instructions dont le code hors codage du décalage n'occupe qu'un quartet. C'est le cas par exemple de GOC, GONC, GOTO et GOSUB. Pour des instructions de saut relatif telles que GOLONG et GOSUBL dont le code hors codage du décalage occupe deux quartets, on utilisera la formule $@_{label}$-$@_{saut}$-2 pour connaître le nombre de quartets à sauter.

Toujours dans le cadre de notre exemple, l'instruction de saut se trouve à l'adresse #112h et le quartet de branchement représenté par « * » se trouve à l'adresse #11Dh. La longueur du code de GOC hors codage du décalage est de 1 quartet. On utilise donc ici la formule $@_{label}$-$@_{saut}$-1 pour calculer la valeur du décalage. On calcule #11Dh-#112h-1. On obtient #Ah ou #0Ah sur deux quartets. GOC label_1 sera codé 4A0 après inversion du champ codant le décalage.

Répétons que les adresses de chaque instruction sont relatives, on considère que le premier quartet de la première instruction du programme se trouve en #0h puis on affecte les adresses des instructions suivantes en se contentant de compter les quartets. Ces adresses affectées arbitrairement par le programmeur ne sont pas du tout représentatives des adresses réelles en mémoire.

Considérons maintenant un saut relatif vers le début du programme. Le décalage d'adresse sera donc négatif. Le code de l'instruction de saut (hors codage du décalage) utilisée pour cette exemple ne comporte qu'un quartet. On applique donc la formule ($@_{saut}$+1)-$@_{label}$ pour connaître le nombre de quartets à sauter. Notez qu'on utilise la formule ($@_{saut}$+2)-$@_{label}$ avec les instructions dont le code hors codage du décalage comporte 2 quartets. On cherche ensuite le complément à 2 du décalage. Pour cela on soustrait à #0h le décalage calculé. On supprime ensuite tous les #Fh inutiles sur la gauche du résultat pour ne conserver que le nombre de quartets qui nous intéressent (les deux quartets de droite pour GOC et

Adresses	Labels	Mnémoniques	Codes
#2D2h	Label_1	*Mnémonique...*	*XXXXX*
#2D7h		*Mnémonique...*	**XXX*
#2DBh		*Mnémonique...*	*XXXXX*
#2E0h		*Mnémonique...*	*XX*
#2E2h		GOC Label_1	4qp

GONC). Ces quelques quartets qui nous intéressent devront bien entendu être inversés avant d'être incorporés dans le code. Dans le cadre de notre exemple, on a $@_{label}$=#2D7h et $@_{saut}$=#2E2h. On applique la formule ($@_{saut}$+1)-$@_{label}$. On obtient #Ch après avoir confié le calcul à la HP-48. Passons maintenant à la complémentation à 2. Si #Ch se trouve au niveau 1 de la pile, on saisie #0h, on effectue un SWAP pour que #0h se place au niveau 2 en renvoyant #Ch au niveau 1. Après une pression sur la touche [-], on obtient #FFFFFFFFFFFFFFF4h. Seuls les deux derniers quartets nous intéressent. Après inversion du champ codant le décalage, `GOC label_1` (c'est-à-dire `GOC F4`) sera codé par `44F`.

Retenez qu'un saut relatif correspond à un décalage d'adresses. Pour connaître la valeur de ce décalage, on compte les quartets à partir de la fin de l'instruction de saut hors codage du décalage et le premier quartet de l'instruction constituant le point de branchement. Pour le calcul de la valeur d'un décalage on pourra utiliser la formule $@_{label}$-($@_{saut}$+*n*) dans le cas d'un saut vers la fin du programme (décalage positif) ou la formule $@_{saut}$+*n*-$@_{label}$ dans le cas d'un saut vers le début du programme (décalage négatif). Ici *n* vaut 1 ou 2 et correspond au nombre de quartets constituant le code de l'instruction de saut hors codage du décalage. Dans le cas d'un décalage négatif, le complément à 2 s'obtient à l'aide la HP-48 en calculant #0h-*valeur_du_décalage* et en ne retenant sur la droite du résultat que le nombre de quartets attendus. Bien entendu les quartets codant le décalage devront être inversés.

1.1
Les sauts relatifs, absolus et indirects

■ Saut relatif conditionnel limité à #80h quartets

Il existe deux instructions de saut relatif conditionnel limité à #80h quartets. L'une, GONC, provoque un saut si le bit de retenue (« carry bit ») vaut zéro. L'autre, GOC, provoque un saut si le bit de retenue vaut un. Cela dit, aucune de ces deux instructions ne modifie le bit de retenue dont elles ne font qu'exploiter la valeur.

Mnémoniques	Codes hexadécimaux	Conditions	Retenue	Cycles
GONC ***label***	5*qp*	saut si Carry = 0	oui	3 ou 10
GOC ***label***	4*qp*	saut si Carry = 1	oui	3 ou 10

Le saut vers le point de branchement *label* est un saut relatif codé sur deux quartets #*pq*h. L'emploi du complément à 2 pour les valeurs négatives permet de remonter jusqu'à 128 quartets (#*pq*h=#80h) avant l'instruction de saut ou de s'éloigner de 127 quartets (#*pq*h=#7Fh) après l'instruction de saut.

Si le saut peut avoir lieu (la retenue a la valeur convenable), 10 cycles sont nécessaires à l'exécution de GOC ou de GONC. Si la condition n'est pas remplie, le saut n'a pas lieu et le traitement de GOC ou de GONC occupe le processeur pendant 3 cycles seulement.

■ Saut relatif inconditionnel limité à #800h quartets

GOTO ordonne un saut relatif inconditionnel. Le codage sur trois quartets du décalage d'adresses et l'emploi du complément à 2 pour les valeurs négatives du décalage permet un déplacement au sein du programme compris entre -2048 quartets (#*pqr*h=800h) et +2047 quartets (#*pqr*h=#7FFh). Le bit de retenue (« carry bit ») n'est ni considéré ni modifié par GOTO.

Mnémoniques	Codes hexadécimaux	Condition	Retenue	Cycles
Goto *label*	6*rqp*	aucune	non	11

■ Saut relatif inconditionnel limité à #8000h quartets

GOLONG ordonne un saut relatif inconditionnel. Le codage sur quatre quartets du décalage d'adresses et l'emploi du complément à 2 pour les valeurs négatives du décalage permet un déplacement au sein du programme compris entre -32768 quartets (#*pqrs*h=8000h) et +32767 quartets (#*pqrs*h=#7FFFh). Le bit de retenue (« carry bit ») n'est ni considéré ni modifié par GOLONG.

Mnémoniques	Codes hexadécimaux	Condition	Retenue	Cycles
Golong *label*	8C*srqp*	aucune	non	14

■ Saut absolu inconditionnel vers une adresse codée sur 5 quartets

GOVLNG provoque un saut absolu. Il suffit donc d'indiquer une adresse #*pqrst*h, codée sur 5 quartets pour que l'exécution reprenne à partir de cette adresse.

Mnémoniques	Codes hexadécimaux	Condition	Retenue	Cycles
Govlng *label*	8D*tsrqp*	aucune	non	14

■ Saut absolu inconditionnel vers une adresse définie par le champ A du registre A ou C

PC=A (respectivement, PC=C) copie dans le registre PC le contenu du champ A du registre A (respectivement, le champ A du registre C). Ces champs A contiennent 5 quartets *#pqrsth*, l'exécution reprend donc à partir de l'adresse *#pqrsth*. Le bit de retenue (« carry bit ») n'est ni considéré ni modifié par ces instructions.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
PC=A	81B2	champ A du registre A	non	19
PC=C	81B3	champ A du registre C	non	19

■ Saut absolu inconditionnel vers une adresse définie par le champ A du registre A ou C avec sauvegarde du PC

APCEX (respectivement, CPCEX) copie dans le registre PC le contenu du champ A du registre A (respectivement, le champ A du registre C). Le champ A du registre A ou C contient 5 quartets *#pqrsth*, l'exécution reprend donc à partir de l'adresse *#pqrsth*. Le registre PC effectue une copie de son ancien contenu dans le champ A du registre dont il vient de prendre la valeur. Le registre PC et le champ A du registre A ou C échangent donc leurs valeurs. Le bit de retenue (« carry bit ») n'est ni considéré ni modifié par ces instructions.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
APCEX	81B6	champ A du registre A	non	19
CPCEX	81B7	champ A du registre C	non	19

■ Saut indirect et inconditionnel

PC=(A) (respectivement, PC=(C)) entraîne la lecture du champ A du registre A (respectivement, champ A du registre C). Ce champ contient cinq quartets *#pqrsth*. On lit cinq quartets à partir de l'adresse *#pqrsth*. Admettons que les cinq quartets lus à partir de l'adresse *#pqrsth* soient *#abcdeh*. Alors, l'exécution reprend à partir de l'adresse *#edcbah*.

Notez l'inversion : #*abcde*h est lu mais c'est à partir de #*edcba*h que reprend l'exécution.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
PC=(A)	808C	champ A du registre A	non	23
PC=(C)	808E	champ A du registre C	non	23

■ Sauvegarde du registre PC (« Program counter»)

A=PC et C=PC ne provoquent pas de saut mais seront parfois employées avant un saut afin d'effectuer une sauvegarde du registre PC. A=PC (respectivement C=PC) copie dans le champ A du registre A (respectivement, registre C) les cinq quartets contenus dans le registre PC.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
A=PC	81B4	champ A du registre A	non	11
C=PC	81B5	champ A du registre C	non	11

1.2
Les appels de sous-programmes

Un appel de sous-programme se décompose en trois étapes :

- la mémorisation de l'adresse de retour qui est placée au sommet de la pile des retours (RSTK),
- la reprise de l'exécution du programme à partir du début du sous-programme (c'est un point de branchement),
- la rencontre d'une instruction de retour provoque un saut vers l'instruction de retour dont l'adresse est alors supprimée de la pile RSTK.

Par rapport à un saut classique, un appel de sous-programme provoque la mémorisation d'une adresse de retour. Cette adresse de retour est l'adresse de l'instruction qui suit l'instruction d'appel du sous-programme.

Lors de l'appel d'un sous-programme, l'adresse de retour est « empilée » dans la pile des retours (appelée RSTK pour « return stack »). La pile RSTK comporte plusieurs niveaux, il est ainsi possible d'appeler depuis un sous-

programme un « sous-sous-programme ». Le nombre de niveaux d'imbrication des sous-programmes est limité par la capacité de la pile RSTK. dont deux niveaux devront impérativement demeurer libres pour permettre le traitement d'une interruption.

Il existe différentes instructions de retour de sous-programme. Elles peuvent être conditionnelles, modifier le bit de retenue, autoriser les interruptions ou armer le bit XM.

Un appel de sous-programme comporte, outre la sauvegarde d'une adresse de retour, un saut relatif ou absolu. Le calcul du décalage pour un saut relatif diffère de celui déjà vu pour les sauts relatifs simples (sans appel de sous-programme). Un saut absolu se résume à l'indication d'une adresse (sur 5 quartets) à partir de laquelle l'exécution doit reprendre.

Pour calculer la valeur du décalage d'adresses utilisé pour un saut relatif lors de l'appel d'un sous-programme, on applique la formule $@_{label}$-$@_{retour}$. $@_{label}$ est l'adresse du premier quartet du point de branchement, c'est donc l'adresse du premier quartet de la première instruction du sous-programme. $@_{retour}$ est l'adresse de l'instruction de retour, c'est donc l'adresse du premier quartet de l'instruction qui suit immédiatement l'instruction d'appel du sous-programme. Si le décalage est négatif (retour vers le début du programme), on utilise le complément à 2.

Afin de faciliter le développement de vos programmes vous pouvez marquer les débuts de sous-programmes à l'aide de labels. Votre code source comportera donc des GOSUB *label* ou des GOSBVL *label* qui ne sont que des représentations commodes de GOSUB *pqr* ou de GOSBVL *pqrst* où *p*, *q*, *r*, *s* et *t* sont des quartets composant une adresse ou un décalage d'adresses selon le cas.

■ Appel d'un sous-programme par saut relatif limité à #800h quartets

GOSUB ordonne un appel de sous-programme par saut relatif inconditionnel. Le codage sur trois quartets du décalage d'adresses et l'emploi du complément à 2 pour les valeurs négatives du décalage permet, au sein du programme, un déplacement compris entre -2048 quartets (#*pqr*h=800h) et +2047 quartets (#*pqr*h=#7FFh). Le bit de retenue (« carry bit ») n'est ni considéré ni modifié par GOSUB. Un sous-programme est donc exécuté à l'adresse de *label*. L'adresse de *label* est sépa-

Mnémonique	Code hexadécimal	Champ concerné	Retenue	Cycles
GOSUB *label*	7*rqp*	1 niveau de RSTK	non	12

rée de l'adresse de GOSUB (GOSUB étant codée 7*rqp*) par un décalage de #*pqr*h quartets. Ce décalage est codé en complément à 2 s'il est négatif.

■ Appel d'un sous-programme par saut relatif limité à #8000h quartets

GOSUBL (« Gosub long ») ordonne un appel de sous-programme par saut relatif inconditionnel. Le codage sur quatre quartets du décalage d'adresses et l'emploi du complément à 2 pour les valeurs négatives du décalage permet, au sein du programme, un déplacement compris entre -32768 quartets (#*pqrs*h=8000h) et +32767 quartets (#*pqrs*h=#7FFFh). Le bit de retenue (« carry bit ») n'est ni considéré ni modifié par GOSUBL.

Mnémonique	Code hexadécimal	Champ concerné	Retenue	Cycles
GOSUBL *label*	8E*srqp*	1 niveau de RSTK	non	15

Un sous-programme est donc exécuté à l'adresse de *label*. L'adresse de *label* est séparée de l'adresse de GOSUBL (GOSUBL étant codé 8E*srqp*) par un décalage de #*pqrs*h quartets. Ce décalage est codé en complément à 2 s'il est négatif.

■ Appel d'un sous-programme par saut absolu

GOSBVL permet l'appel d'un sous-programme à une adresse codée sur 5 quartets, il s'agit donc ici d'un branchement absolu. GOSBVL est particulièrement utile pour l'appel de routines en mémoire morte. Pour l'appel d'un sous-programme à l'adresse #*pqrst*h, GOSBVL sera codée 8F*tsrqp*.

Mnémonique	Code hexadécimal	Champ concerné	Retenue	Cycles
GOSBVL *label*	8F*tsrqp*	1 niveau de RSTK	non	15

■ Retour inconditionnel de sous-programme

Lorsqu'une adresse de retour inconditionnel est rencontrée, un branchement est effectué à l'adresse de retour qui se trouve au sommet de la pile RSTK, cette adresse est ensuite supprimée de la pile RSTK. Le programme poursuit ensuite son exécution à partir de l'instruction de retour, c'est-à-dire l'instruction qui suit celle ayant provoqué l'exécution du sous-programme.

La plus simple des instructions de retour de sous-programme est RTN (« return ») qui se contente de brancher l'exécution sur l'instruction de

retour. En plus de ce simple retour, RTNSC, RTNCC, RTI et RTNSXM effectue une manipulation annexe. RTNSC (« return and set carry bit ») met à un le bit de retenue. RTNCC (« return and clear carry bit ») met à zéro le bit de retenue. RTI autorise les interruptions. RTNSXM (« return and set external module missing bit ») met à un le bit XM.

Mnémoniques	Codes hexadécimaux	Bit concerné	Retenue	Cycles
RTN	01	aucun	non	9
RTNSC	02	la retenue passe à 1	oui	11
RTNCC	03	la retenue est mise à 0	oui	11
RTI	0F	aucun	non	11
RTNSXM	00	le bit XM passe à 1	non	11

■ Retour conditionnel de sous-programme en fonction de la retenue

RTNC provoque un retour de sous-programme à condition que le bit de retenue vaille 1. Inversement, RTNNC provoque un retour de sous-programme si le bit de retenue vaut 0. RTNC et RTNNC ne modifient pas la valeur du bit de retenue. Ne confondez pas RTNNC et RTNCC !

Mnémoniques	Codes hexadécimaux	Condition	Retenue	Cycles
RTNC	400	Bit de retenue à 1	oui	3 ou 10
RTNNC	500	Bit de retenue à 0	oui	3 ou 10

Si la condition (valeur de la retenue) n'est pas remplie, 3 cycles sont nécessaires à l'exécution de ces instructions. Si la condition est remplie, le retour est possible et 10 cycles sont alors nécessaires.

2
Les tests

Lors d'un test, on effectue une comparaison, par exemple A>B. Si A est effectivement strictement supérieur à B, alors le test est vrai, sinon, le test est faux. Si le test est vrai, on interprète les deux quartets #*pq*h qui suivent le test. Si #*pq*h = #00h alors un test vrai provoque un RTNYES, autrement dit, un retour de sous-programme (ce retour de sous-programme

est conditionnel puisque soumis au résultat du test). Si les deux quartets #*pq*h sont différents de #00h alors on effectue un saut relatif de #*pq*h quartets si le test est vrai. Les deux quartets #*pq*h correspondent alors à une instruction GOYES *pq*, autrement dit « se décaler, si le test est vrai, d'un nombre de quartets égal à #*pq*h ».

Un décalage d'adresses est calculé à l'aide de la formule $@_{label}$-$@_{GOYES}$ si $@_{label}$>$@_{GOYES}$. Si le décalage est négatif ($@_{GOYES}$>$@_{label}$), il faut calculer *décalage*=$@_{GOYES}$-$@_{label}$ puis #0h-*décalage* et ne retenir que les deux quartets de droite si on effectue ce calcul à l'aide de la HP-48. Les deux quartets devront bien entendu être inversés lors du codage.

Retenez que les instructions RTNYES et GOYES *pq* sont implicites après un test qui est toujours de la forme :

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

Après le test, on trouve deux quartets #*pq*h. #*pq*h = #00h correspond à RTNYES. #*pq*h non nul représente l'instruction GOYES *pq*.

Premier exemple : codons la phrase « *provoquer un retour de sous-programme si le champ A du registre D est nul* ». Traduite en mnémoniques, la phrase devient :

```
?D=0          A
RTNYES
```

?D=0 A correspond à « *le champ A du registre D est-il nul ?* » alors que RTNYES signifie « *si oui, provoquer un retour de sous-programme* ». Passons maintenant au codage en hexadécimal :

```
8AB
00
```

Etant placé immédiatement après un test (8AB), 00 représente RTNYES et non pas RTNSXM comme dans le cas général. Ici les deux quartets #*pq*h qui suivent le test sont tels que #*pq*h = #00h.

Second exemple : codons la phrase « *poursuivre l'exécution à partir du label_1 si le champ A du registre D est nul* ». Traduite en mnémoniques, la phrase devient :

```
?D=0          A
GOYES label_1
```

Considérons que l'adresse de l'instruction GOYES soit #EF1h et que l'adresse de label_1 soit #EC5h. Rappelons qu'il s'agit ici d'un adressage relatif propre au programme : on se contente d'affecter l'adresse #0h au premier quartet de la première instruction du programme puis on numérote les autres quartets. Dans le cadre de cet exemple on a $@_{GOYES} > @_{label\_1}$, le décalage d'adresse est donc négatif.

On pose $@_{GOYES} - @_{label\_1}$ = #EF1h-#EC5h, on obtient #2Ch. Pour obtenir le complément à 2, on calcule 0-#2Ch. On trouve, à l'aide de la HP-48, le résultat #FFFFFFFFFFFFFD4h dont on ne retiendra que les deux derniers quartets qu'il faudra ensuite inverser. On peut maintenant écrire :

```
?D=0          A
GOYES D4
```

`?D=0 A` correspond à « *le champ A du registre D est-il nul ?* » alors que `GOYES D4` signifie « *si oui, revenir #2Ch quartets en arrière* ». Passons maintenant au codage en hexadécimal :

```
8AB
4D
```

Etant placé immédiatement après un test (`8AB`), `4D` représente `GOYES D4`. Ici les deux quartets #*pq*h qui suivent le test sont tels que #*pq*h = #D4h. Ils se retrouvent sous la forme *qp* après le codage en hexadécimal qui s'accompagne de l'inversion des champs de paramètres.

2.1
Comparaisons de registres

Un champ d'un registre va être comparé à un champ d'un autre registre ou à une constante (zéro). Les champs comparés peuvent être des registres de travail entiers (champs W). Si le test est vrai, les deux quartets qui suivent le test seront considérés. S'ils valent #00h, il s'agit d'un `RTNYES`, s'ils valent #*pq*h (#*pq*h ≠ #00h), il s'agit d'un `GOYES` *pq*. Répétons que `GOYES` *label* et `GOYES` *pq* sont équivalents si l'instruction `GOYES` et le point de branchement défini par *label* sont séparés par *pq* quartets (codage en complément à 2 si le décalage est négatif).

On utilise ici aussi *a* et *b* pour désigner les quartets variables des codes hexadécimaux. Reportez-vous au chapitre *Introduction au jeu d'instruc-*

tions. Toutes les instructions de test à partir de registre(s) occupent le processeur pendant 13+*n* cycles si le test est vrai (les deux quartets GOYES ou RTNYES qui suivent le test sont alors considérés). Le processeur est occupé pendant 6+*n* cycles si le test est faux. Ici, *n* est le nombre de quartets constituant le champ auquel est appliquée l'instruction.

■ Un registre est-il nul ?

Chacun des quatre registres de travail (A, B, C et D) peut être comparé à une constante (ici, zéro). Le test sera « vrai » si le champ ou le registre comparé à zéro est effectivement nul. Dans ce cas, les deux quartets *pq* seront considérés.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
?A=0 *ch* *pq*	8A8 *q p*	A	oui	6+*n* ou 13+*n*
	9a8 *q p*	P, WP, XS, X, S, M, B, W		
?B=0 *ch* *pq*	8A9 *q p*	A	oui	6+*n* ou 13+*n*
	9a9 *q p*	P, WP, XS, X, S, M, B, W		
?C=0 *ch* *pq*	8AA *q p*	A	oui	6+*n* ou 13+*n*
	9aA *q p*	P, WP, XS, X, S, M, B, W		
?D=0 *ch* *pq*	8AB *q p*	A	oui	6+*n* ou 13+*n*
	9aB *q p*	P, WP, XS, X, S, M, B, W		

Les deux quartets *pq* sont les compléments obligatoires du test puisqu'ils déterminent ce qui devra être fait si le test est vrai. Les deux quartets *pq* représentent, en fonction de leur valeur, RTNYES ou GOYES *pq*.

Si #*pq*h = #00h, *pq* correspond à RTNYES (retour de sous-programme si le test est vrai). Si #*pq*h ≠ #00h, *pq* représente GOYES *pq* (saut relatif de *pq* quartets si le test est vrai).

Le bit de retenue passe à 1 si le test est vrai. Il est mis à zéro si le test est faux.

Dans les tableaux les mnémoniques sont formulés à l'aide d'une notation du type **?A=0 *ch* *pq***, en fait, vous emploierez dans vos listings la notation :

```
?A=0    ch                ou                ?A=0    ch
GOYES pq                                    RTNYES
```

en fonction du rôle de l'octet *pq*. Les instructions GOYES et RTNYES sont donc implicites dans les colonnes de mnémoniques puisqu'elles sont intimement liées à la valeur (et donc au rôle) de *pq*.

■ Un registre est-il non nul ?

Ici encore, l'un des quatre registres de travail (A, B, C et D) va être comparé à une constante (ici, zéro) mais le test sera « vrai » si le champ ou le registre comparé à zéro n'est pas nul.

Les deux quartets *pq* sont considérés si le test est vrai.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
?A≠0 *ch pq*	8AC *q p*	A	oui	6+*n* ou 13+*n*
	9aC *q p*	P, WP, XS, X, S, M, B, W		
?B≠0 *ch pq*	8AD *q p*	A	oui	6+*n* ou 13+*n*
	9aD *q p*	P, WP, XS, X, S, M, B, W		
?C≠0 *ch pq*	8AE *q p*	A	oui	6+*n* ou 13+*n*
	9aE *q p*	P, WP, XS, X, S, M, B, W		
?D≠0 *ch pq*	8AF *q p*	A	oui	6+*n* ou 13+*n*
	9aF *q p*	P, WP, XS, X, S, M, B, W		

Les deux quartets *pq* sont les compléments obligatoires du test puisqu'ils déterminent ce qui devra être fait si le test est vrai. Les deux quartets *pq* représentent, en fonction de leur valeur, `RTNYES` ou `GOYES` *pq*. Pour plus de détails, voir le texte d'introduction aux tests et le paragraphe *Un registre est-il nul ?*

Le bit de retenue passe à 1 si le test est vrai. Il est mis à zéro si le test est faux.

■ Deux registres sont-ils égaux ?

Ici, le test est vrai si les deux champs de registres comparés sont égaux (ils ont le même contenu). Remarquez que ce test ne peut s'effectuer qu'entre les couples de registres (A,B), (B,C), (A,C) et (C,D). Les deux quartets *pq* sont considérés si les champs comparés sont égaux. Les deux quartets *pq* sont les compléments obligatoires du test puisqu'ils détermi-

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
?A=B *ch pq*	8A0 *q p*	A	oui	6+*n* ou 13+*n*
	9a0 *q p*	P, WP, XS, X, S, M, B, W		
?B=C *ch pq*	8A1 *q p*	A	oui	6+*n* ou 13+*n*
	9a1 *q p*	P, WP, XS, X, S, M, B, W		
?A=C *ch pq*	8A2 *q p*	A	oui	6+*n* ou 13+*n*
	9a2 *q p*	P, WP, XS, X, S, M, B, W		
?C=D *ch pq*	8A3 *q p*	A	oui	6+*n* ou 13+*n*
	9a3 *q p*	P, WP, XS, X, S, M, B, W		

nent ce qui devra être fait si le test est vrai. Les deux quartets *pq* représentent, en fonction de leur valeur, RTNYES ou GOYES *pq*. Pour plus de détails, voir le texte d'introduction aux tests et le paragraphe *un registre est-il nul ?*

Le bit de retenue passe à 1 si le test est vrai. Il est mis à zéro si le test est faux.

■ Deux registres sont-ils différents ?

Ici, le test est vrai si les deux champs de registres comparés sont différents (ils n'ont pas le même contenu). Ce test n'est possible qu'entre les couples de registres (A,B), (B,C), (A,C) et (C,D). Les deux quartets *pq* sont considérés si les champs comparés sont inégaux. Les deux quartets *pq* sont les compléments obligatoires du test puisqu'ils déterminent ce qui devra être fait si le test est vrai. Les deux quartets *pq* représentent, en fonction de leur valeur, RTNYES ou GOYES *pq*. Pour plus de détails, voir le texte d'introduction aux tests et le paragraphe *Un registre est-il nul ?*

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
?A≠B ch pq	8A4 *q p*	A	oui	6+*n* ou 13+*n*
	9*a*4 *q p*	P, WP, XS, X, S, M, B, W		
?B≠C ch pq	8A5 *q p*	A	oui	6+*n* ou 13+*n*
	9*a*5 *q p*	P, WP, XS, X, S, M, B, W		
?A≠C ch pq	8A6 *q p*	A	oui	6+*n* ou 13+*n*
	9*a*6 *q p*	P, WP, XS, X, S, M, B, W		
?C≠D ch pq	8A7 *q p*	A	oui	6+*n* ou 13+*n*
	9*a*7 *q p*	P, WP, XS, X, S, M, B, W		

Le bit de retenue passe à 1 si le test est vrai, il est mis à zéro sinon.

■ Tester l'infériorité ou la supériorité strictes

Ici, le test est vrai si l'un des champs est effectivement strictement supérieur (ou inférieur, selon le cas) au champ du registre auquel il est comparé. Ce test n'est possible qu'entre les couples de registres (A,B), (B,C), (A,C) et (C,D). Les deux quartets *pq* sont les compléments obligatoires du test puisqu'ils déterminent ce qui devra être fait si le test est vrai. Les deux quartets *pq* représentent, en fonction de leur valeur, RTNYES ou GOYES *pq*. Remarquez que ?A<B *ch* peut aussi être noté ?B>A *ch*. Pour plus de détails, voir le texte d'introduction aux tests et le paragraphe *Un registre est-il nul ?*

Le bit de retenue passe à 1 si le test est vrai. Il est mis à zéro si le test est faux.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
?A>B *ch pq*	8B0 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*0 *q p*	P, WP, XS, X, S, M, B, W		
?B>C *ch pq*	8B1 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*1 *q p*	P, WP, XS, X, S, M, B, W		
?C>A *ch pq*	8B2 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*2 *q p*	P, WP, XS, X, S, M, B, W		
?D>C *ch pq*	8B3 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*3 *q p*	P, WP, XS, X, S, M, B, W		
?A<B *ch pq*	8B4 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*4 *q p*	P, WP, XS, X, S, M, B, W		
?B<C *ch pq*	8B5 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*5 *q p*	P, WP, XS, X, S, M, B, W		
?C<A *ch pq*	8B6 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*6 *q p*	P, WP, XS, X, S, M, B, W		
?D<C *ch pq*	8B7 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*7 *q p*	P, WP, XS, X, S, M, B, W		

■ Tester l'infériorité ou la supériorité larges

Ici, le test est vrai si l'un des champs est effectivement supérieur ou égal (ou inférieur ou égal, selon le cas) au champ du registre auquel il est comparé. Ce test n'est possible qu'entre les couples de registres (A,B), (B,C),

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
?A≥B *ch pq*	8B8 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*8 *q p*	P, WP, XS, X, S, M, B, W		
?B≥C *ch pq*	8B9 *q p*	A	oui	6+*n* ou 13+*n*
	9*b*9 *q p*	P, WP, XS, X, S, M, B, W		
?C≥A *ch pq*	8BA *q p*	A	oui	6+*n* ou 13+*n*
	9*b*A *q p*	P, WP, XS, X, S, M, B, W		
?D≥C *ch pq*	8BB *q p*	A	oui	6+*n* ou 13+*n*
	9*b*B *q p*	P, WP, XS, X, S, M, B, W		
?A≤B *ch pq*	8BC *q p*	A	oui	6+*n* ou 13+*n*
	9*b*C *q p*	P, WP, XS, X, S, M, B, W		
?B≤C *ch pq*	8BD *q p*	A	oui	6+*n* ou 13+*n*
	9*b*D *q p*	P, WP, XS, X, S, M, B, W		
?C≤A *ch pq*	8BE *q p*	A	oui	6+*n* ou 13+*n*
	9*b*E *q p*	P, WP, XS, X, S, M, B, W		
?D≤C *ch pq*	8BF *q p*	A	oui	6+*n* ou 13+*n*
	9*b*F *q p*	P, WP, XS, X, S, M, B, W		

(A,C) et (C,D). Les deux quartets *pq* sont les compléments obligatoires du test puisqu'ils déterminent ce qui devra être fait si le test est vrai. Les deux quartets *pq* représentent, en fonction de leur valeur, `RTNYES` ou `GOYES` *pq*. Remarquez que `?A≤B` *ch* peut aussi être noté `?B≥A` *ch*. Pour plus de détails, voir le texte d'introduction aux tests et le paragraphe *Un registre est-il nul ?*

Le bit de retenue passe à 1 si le test est vrai. Il est mis à zéro si le test est faux.

■ Tester l'état d'un bit

Les instructions du type `?ABIT=0` *n* permettent de tester l'état d'un bit. Par exemple, nous voulons savoir si le premier bit du second quartet du registre C vaut 1. On utilisera ici le mnémonique `?CBIT=1 4` que l'on code `808B4`. Le quartet variable *n* varie entre #0h et #Fh. Il peut donc désigner chacun des quatre bits des quatre premiers quartets du registre A (`?ABIT=...`) ou du registre C (`?CBIT=...`), *n*=4 désigne donc le premier bit du second quartet. Le test sera vrai si le bit est armé (`...=1`) ou si le bit est désarmé (`...=0`).

Ces quatre instructions gagnent à être utilisées en complément de `ABIT=...` ou de `CBIT=...` qui affectent la valeur de votre choix (limité à 0 ou à 1 !) à l'un des bits de l'un des quatre premiers quartets du registre A ou du registre C.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
`?ABIT=0 n pq`	`8086`*n* *q p*	4 premiers quartets de A	oui	12 ou 21
`?ABIT=1 n pq`	`8087`*n* *q p*	4 premiers quartets de A	oui	12 ou 21
`?CBIT=0 n pq`	`808A`*n* *q p*	4 premiers quartets de C	oui	12 ou 21
`?CBIT=1 n pq`	`808B`*n* *q p*	4 premiers quartets de C	oui	12 ou 21

Le bit de retenue passe à 1 si le test est vrai. Il est mis à zéro si le test est faux.

L'exécution de l'une de ces quatre instructions requiert 21 cycles si le test est vrai et 12 cycles si ce n'est pas le cas.

Les deux quartets *pq* qui suivent ces instructions sont employés comme ceux qui suivent les instructions de test entre registres. Autrement dit, *pq* représentent un `RTNYES` si #*pq*h = #00h ou un `GOYES` *pq* si l'octet #*pq*h est non nul.

Par exemple la phrase « provoquer un retour de sous-programme si le troisième bit du premier quartet du registre C est nul » correspond aux mnémoniques :

```
?CBIT=0     2
RTNYES
```

`?CBIT` indique qu'il s'agit d'un test concernant un bit du registre C, `=0` précise que l'on vérifie que ce bit est bien nul. Les bits des quatre premiers quartets étant numérotés de #0h à #Fh, le troisième bit du premier quartet porte le numéro 2. `RTNYES` indique que si le test est vrai on provoque un retour de sous-programme.

Finalement, on obtient le code :

```
808A2
00
```

Ici encore, l'octet #*pq*h (il vaut ici #00h) qui détermine l'action en cas de test vrai peut être considéré comme une partie du code du test même s'il s'agit, en fait, d'une instruction de saut conditionnel ou de retour conditionnel.

2.2
Gérer le résultat du test

■ Exploiter la retenue

L'état du bit de retenue (« carry bit ») est représentatif du résultat du test (vrai ou faux). Le bit de retenue vaut 1 si le test est vrai, il vaut 0 si la condition définie par le test n'est pas vérifiée. La valeur de la retenue pourra être exploitée par les instructions de saut conditionnel utilisant le bit de retenue. `GOC` et `GONC` sont des instructions de saut relatif contrôlées par la valeur du bit de retenue.

■ Exploiter les deux quartets qui complètent les tests

Un test sur un registre est codé sur 5 quartets alors qu'un test sur l'un des bits de l'un des quatre premiers quartets du registre A ou du registre C est

codé sur 7 quartets. Dans les deux cas, les deux derniers quartets #*pq*h du codage contrôlent la suite de l'exécution du programme si le test est vrai. A travers ces deux quartets, un test peut provoquer un retour de sous-programme ou un saut relatif limité à #80h quartets. Un test « vrai » entraîne donc toujours un branchement de l'exécution.

Ce branchement s'effectue à l'adresse de retour se trouvant au sommet de la pile RSTK si #*pq*h = #00h. #pqh représente ici `RTNYES`. Si #*pq*h ≠ #00h le branchement de l'exécution s'effectue à l'instruction atteinte après un saut relatif de *pq* quartets.

17

Instructions : mode, registres spéciaux, interruptions, etc.

Nous savons depuis longtemps que les registres de travail A, B, C et D sont ceux qui autorisent le plus d'opérations. Nous avons ensuite constaté qu'il existe pour les registres A et C des instructions indisponibles pour B et D. Pour la première fois nous allons être confrontés à des instructions utilisables uniquement avec le registre C. Celui-ci est donc le registre roi, le seul à qui tout soit permis dans les limites de l'assembleur SATURN.

1

Les registres spéciaux

■ Manipuler l'unique quartet du registre P

Le registre P ne contient qu'un quartet. Celui-ci voit donc son contenu varier entre #0h et #Fh. Le contenu du registre P définit le champ P des registres de travail A, B, C et D. Par exemple, si l'unique quartet du registre P vaut #2h, le champ P d'un registre de travail correspond à son quartet n°#2h. Le champ WP s'étendant du quartet n°#0h jusqu'au quartet n°#*P*h d'un registre de travail, le champ WP d'un registre de travail correspond aux quartets n°#0h, n°#1h et n°#2h si le registre P contient #2h.

Le registre P, outre les deux champs qu'il est chargé de définir, peut être utilisé comme compteur ou pour tout autre usage astucieux exploitant les nombreux traitements possibles à partir de P. En effet, il est possible d'af-

fecter une valeur initiale à P, de le décrémenter, de l'incrémenter, de tester son contenu et d'effectuer des échanges entre le contenu de P et celui de C.

Le paramètre *n* est ici un quartet variable compris entre #0h et #Fh. Les instructions de traitement de P sont les suivantes :

- `P=` *n* affecte au registre P la valeur de *n*,
- `P=P+1` incrémente (augmente par pas de 1) le contenu du registre P. L'opération s'effectue en hexadécimal indépendamment du mode courant,
- `P=P-1` décrémente (diminue par pas de 1) le contenu du registre. L'opération s'effectue en hexadécimal indépendamment du mode courant,
- `?P≠`*n* *pq* est un test qui est vrai si le contenu de P est différent de *n*. Les deux quartets *pq* ont le même rôle que les quartets *pq* placés à la suite d'un test classique. #*pq*h = #00h correspond à `RTNYES` (retour de sous-programme si P≠*n*). De même, #*pq*h ≠ #00h correspond à `GOYES` *pq* et entraîne un saut relatif sur #*pq*h quartets si P≠*n*,
- `?P=`*n* *pq* est un test qui est vrai si le contenu de P est égal à *n*. Les deux quartets *pq* ont le même rôle que les quartets *pq* placés à la suite d'un test classique. #*pq*h = #00h correspond à `RTNYES` (retour de sous-programme si P=*n*). Par ailleurs, si #*pq*h ≠ #00h, *pq* correspond à `GOYES` *pq* et entraîne un saut relatif sur #*pq*h quartets si P=*n*,
- `C=P` *n* affecte au quartet n°#*n*h du registre C la valeur du registre P,
- `P=C` *n* affecte au registre P la valeur du quartet n°#*n*h du registre C,
- `C=C+P+1` (aussi notée `C+P+1`) ajoute le contenu du registre P plus 1 au champ A du registre C,
- `CPEX` *n* (aussi notée `PCEX` *n*) échange le contenu du quartet n°#*n*h du registre C avec le contenu du registre P.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
P= n	2*n*	registre P	non	2
P=P+1	0C	registre P	oui	3
P=P-1	0D	registre P	oui	3
?P≠ n pq	88*n* *qp*	registre P	oui	6 ou 13
?P= n pq	89*n* *qp*	registre P	oui	6 ou 13
C=P n	80C*n*	registre P et quartet #*n*h de C	non	6
P=C n	80D*n*	registre P et quartet #*n*h de C	non	6
C+P+1	809	champ A du registre C	oui	8
C=C+P+1	809	champ A du registre C	oui	8
CPEX n	80F*n*	registre P et quartet #*n*h de C	non	6

Le bit de retenue passe à un si le test (`?P≠n` ou `?P=n`, selon le cas) est vrai. Si le test est vrai et que le branchement est possible, l'exécution dure 13 cycles, sinon, 6 cycles seulement sont nécessaires.

Par exemple, la phrase « *provoquer un retour de sous-programme si le registre P contient 10 (#Ah)* » correspond aux mnémoniques :

```
?P=      A
RTNYES
```

Si la condition (P=#Ah) est remplie, `RTNYES` est exécutée. Le code correspondant est :

```
89A
00
```

■ Manipuler la pile des retours (RSTK)

RSTK est la pile des adresses de retour de sous-programme (voir les appels de sous-programmes) au sommet de laquelle on trouve la prochaine adresse de retour à utiliser lorsqu'une instruction de retour (`RTN`, `RTNYES`, etc.) sera rencontrée. Vous pouvez ajouter une adresse au sommet de la pile RSTK ou supprimer l'adresse se trouvant au sommet en la copiant dans le champ A du registre C. Les instructions :

- `C=RSTK` copie dans le champ A du registre C l'adresse se trouvant au sommet de la pile RSTK, cette adresse libère le niveau qu'elle occupe dans la pile RSTK,
- `RSTK=C` ajoute au sommet de la pile RSTK le contenu du champ A du registre C.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
RSTK=C	06	champ A du registre C	non	8
C=RSTK	07	champ A du registre C	non	8

■ Le registre des entrées (IN) et le registre des sorties (OUT)

Les entrées et sorties passent par les registres IN et OUT. Il existe quatre instructions permettant de leur affecter un contenu ou d'exploiter la valeur qu'ils contiennent. On remarquera l'emploi assez exceptionnel du champ X capable d'accueillir trois quartets ce qui correspond à la capacité du registre OUT. Le registre C permet les échanges avec IN et OUT alors que le registre A se contente de pouvoir recevoir une copie de IN.

Les instructions :

- `A=IN` provoque la copie des quatre quartets du registre IN sur les quartets n°#0h à n°#3h du registre A,
- `C=IN` provoque la copie des quatre quartets du registre IN sur les quartets n°#0h à n°#3h du registre C,
- `OUT=C` copie le champ X du registre C dans le registre OUT, cette opération ne concerne donc que trois quartets,
- `OUT=CS` copie le quartet n°#0h sur le quartet de poids faible du registre OUT.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
OUT=CS	800	quartet n°#0h de C	non	4
OUT=C	801	champ X du registre de C	non	6
A=IN	802	4 premiers quartets du registre A	non	7
C=IN	803	4 premiers quartets du registre C	non	7

■ Les bits de statut (ST)

Nous disposons en RPL classique de plusieurs dizaines de drapeaux (flags) aidant à paramétrer la HP-48. Le registre ST (« Status bits ») a une capacité de quatre quartets (16 bits). Les bits de poids fort n°#Ch à n°#Fh sont réservés à la HP-48. Par contre, les douze bits de poids faible n°#0h à n°#Bh sont utilisables par le programmeur. Chacun de ces douze bits peut être comparé à un flag RPL.

Dans les instructions suivantes, le paramètre *n* correspond à un quartet variable :

- `?ST=0` *n pq* est un test qui est vrai si le bit n°#*n*h du registre ST vaut 0. Les deux quartets *pq* ont le même rôle que les quartets *pq* placés à la suite d'un test classique. #*pq*h = #00h correspond à `RTNYES` (retour de sous-programme si le bit testé vaut 0). De même, #*pq*h ≠ #00h correspond à `GOYES` *pq* et entraîne un saut relatif sur #*pq*h quartets si le test est vrai,
- `?ST=1` *n pq* est un test qui est vrai si le bit n°#*n*h du registre ST vaut 1. Les deux quartets *pq* ont le même rôle que les quartets *pq* placés à la suite d'un test classique. #*pq*h = #00h correspond à `RTNYES` (retour de sous-programme si le bit testé vaut 0). De même,#*pq*h ≠ #00h correspond à `GOYES` *pq* et entraîne un saut relatif sur #*pq*h quartets si le test est vrai,
- `CLRST` met à zéro les bits n°#0h à n°#Bh du registre ST,
- `ST=C` copie les trois quartets du champ X du registre C dans les bits n°#0h à n°#Bh du registre ST,

• `C=ST` copie dans le champ X du registre C les bits n°#0h à n°#Bh du registre ST,
• `CSTEX` (aussi notée `STCEX`) échange le contenu des 12 bits du champ X du registre C avec les bits n°#0h à n°#Bh du registre ST,
• `ST=1` *n* met à 1 le bit n°#nh du registre ST,
• `ST=0` *n* met à 0 le bit n°#nh du registre ST,

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
CLRST	08	3 premiers quartets de ST	non	6
C=ST	09	3 premiers quartets de ST & champ X de C	non	6
ST=C	0A	3 premiers quartets de ST & champ X de C	non	6
CSTEX	0B	3 premiers quartets de ST & champ X de C	non	6
ST=0 *n*	84*n*	bit n°#*n*h du registre ST	non	4
ST=1 *n*	85*n*	bit n°#*n*h du registre ST	non	4
?ST=0 *n pq*	86*n* *qp*	bit n°#*n*h du registre ST	oui	7 ou 14
?ST=1 *n pq*	87*n* *qp*	bit n°#*n*h du registre ST	oui	7 ou 14

Les instructions `?ST=0` *n* et `?ST=1` *n* mettent le bit de retenue à 1 si le test est vrai. Le bit de retenue vaut 0 dans le cas contraire. Ces instructions s'exécutent en 14 cycles si le test est vrai, en 7 cycles sinon.

Par exemple, la phrase « *sauter vers le label* `label_1` *distant de #2Ah quartets si le bit n°#Bh du registre ST est nul* » correspond aux mnémoniques :

```
?ST=0     B
GOYES label_1
```

`?ST=0 B` signifie que l'on vérifie que le bit n°#Bh du registre ST est bien nul. `GOYES label_1` indique que si la condition est remplie, l'exécution reprend à partir du point de branchement représenté par `label_1`. Rappelons qu'un label n'est qu'une facilité, il faut quoi qu'il arrive considérer des décalages d'adresses, on peut donc écrire :

```
?ST=0     B
GOYES 2A
```

Après traduction en codes hexadécimaux, on obtient :

```
86B
A2
```

Comme toujours, les champs sont inversés. Le décalage de #2Ah quartets vers le label `label_1` devient donc `A2`.

■ Les bits de statut matériel (HS, « hardware status bits »)

Nous disposons d'une dizaine d'instructions pour affecter des valeurs aux bits du registre HS (« hardware status bits ») ou pour tester la valeur de l'un de ces bits. Les bits utiles du registre HS sont :

- SR (« service request »),
- SB (« sticky bit »),
- MP (« module pulled »),
- XM (« external module missing »).

Dans les instructions qui suivent, *n* représente un quartet variable :

- `CLRHST` (« clear hardware status bits ») met à zéro SR, SB, MP et XM,
- `XM=0`, `SB=0`, `SR=0` et `MP=0` désarment respectivement XM, SB, SR et MP,
- `HS=0` *n* désarme de un à quatre bits de statut matériel selon la formule $n = v_{XM}+2\times v_{SB}+4\times v_{SR}+8\times v_{MP}$ où v_{XM}, v_{SB}, v_{SR} et v_{MP} valent 1 si les bits que ces variables représentent (respectivement XM, SB, SR et MP) doivent être désarmés,
- `?XM=0` *pq*, `?SB=0` *pq*, `?SR=0` *pq* et `?MP=0` *pq* testent la nullité des bits XM, SB, SR et MP (respectivement). Les deux quartets *pq* ont le même rôle que les quartets *pq* placés à la suite d'un test classique. #*pq*h = #00h correspond à `RTNYES` (retour de sous-programme si le bit testé vaut 0). De même, #*pq*h ≠ #00h correspond à `GOYES` *pq* et entraîne un saut relatif sur #*pq*h quartets si le test est vrai,
- `?HS=0` *n* *pq* teste la nullité de un ou plusieurs bits HS (XM, SB, SR et MP). Le test concerne les bits HS définis par *n* selon la formule $n = v_{XM}+2\times v_{SB}+4\times v_{SR}+8\times v_{MP}$ où v_{XM}, v_{SB}, v_{SR} et v_{MP} valent 1 si les bits que ces variables représentent (respectivement XM, SB, SR et MP) doivent être testés. Les deux quartets *pq* ont le même rôle que les quartets *pq* placés à la suite d'un test classique. #*pq*h = #00h correspond à `RTNYES` (retour de sous-programme si le bit testé vaut 0). De même, #*pq*h ≠ #00h correspond à `GOYES` *pq* et entraîne un saut relatif sur #*pq*h quartets si le test est vrai.

Les instructions de test (`?XM=0` *pq*, `?SB=0` *pq*, `?SR=0` *pq*, `?HS=0` *n* *pq*) s'exécutent en 13 cycles si le test est vrai (un saut ou un retour de sous-programme est alors provoqué) ou en 6 cycles si ce n'est pas le cas. Les instructions de test mettent la retenue à 1 si le test est vrai, à 0 sinon.

Par exemple, codons la phrase « *effectuer un retour de sous-programme si les bits XM et MP sont nuls* ». Nous allons utiliser l'instruction `?HS=0` *n* *pq*. La valeur de *n* est définie par $n = v_{XM}+2\times v_{SB}+4\times v_{SR}+8\times v_{MP}$ où v_{XM} et v_{MP}

valent 1 puisque nous testons la nullité du bit XM et du bit MP. On calcule donc n à partir de la formule $n = 1+2\times0+4\times0+8\times1$ (notez que v_{SB} et v_{SR} valent 0). On a donc $n = 9$, c'est-à-dire n = #9h (ici, le passage à l'hexadécimal ne change rien). A ce stade nous avons les mnémoniques :

```
?HS=0      9
RTNYES
```

Un retour de sous-programme en cas de test réussi (RTNYES) est codé en affectant #00h aux quartets #*pq*h. Le codage hexadécimal de notre phrase est :

```
839
00
```

Comme d'habitude, le champ *pq* est inversé, mais bien entendu, dans le cas présent 00 inversé reste 00...

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
CLRHST	82F	les 4 bits HS	non	3
XM=0	821	bit XM	non	3
SB=0	822	bit SB	non	3
SR=0	824	bit SR	non	3
MP=0	828	bit MP	non	3
HS=0 n	82*n*	un ou plusieurs bits HS	non	3
?XM=0	831	bit XM	oui	6 ou 13
?SB=0	832	bit SB	oui	6 ou 13
?SR=0	834	bit SR	oui	6 ou 13
?MP=0	838	bit MP	oui	6 ou 13
?HS=0 n	83*n*	un ou plusieurs bits HS	oui	6 ou 13

2

Modes, interruptions, bus, etc.

■ Mode de numération hexadécimale ou décimale

Le SATURN peut travailler avec deux modes de numération : l'hexadécimal (base 16) ou le décimal (base 10). Rappelons que l'hexadécimal n'est

qu'une représentation commode du binaire naturel dont les bits sont groupés par quatre pour constituer des quartets représentés par les chiffres de 0 à 9 et les lettres A à Z. Le mode décimal fait appel au DCB (ou BCD en anglais) qui n'est autre que le « décimal codé binaire ». En DCB, chaque chiffre de 0 à 9 est codé sur 4 bits. Les représentations sur un quartet des chiffres hexadécimaux A à F sont donc proscrites en DCB.

`SETDEC` place le SATURN en mode décimal (DCB) alors que `SETHEX` active le mode hexadécimal.

Mnémoniques	Codes hexadécimaux	Remarques	Retenue	Cycles
SETHEX	04	représentation du binaire	non	3
SETDEC	05	DCB	non	3

Remarquons qu'il n'existe pas d'instruction permettant de connaître directement le mode de numération employé. Une méthode simple pour connaître le mode courant consiste à charger #9h dans un quartet puis à l'incrémenter (d'abord `LAHEX 9` puis `A=A+1 P`). Si la retenue passe à 1 après l'incrémentation, alors le mode décimal est le mode courant, sinon, le mode hexadécimal est le mode courant.

■ Les interruptions

Les interruptions relatives au clavier sont « masquables ». `INTOFF` interdit au SATURN de tenir compte des pressions sur le clavier. Au contraire `INTON` autorise à nouveau les interruptions relatives au clavier. `RSI` initialise les interruptions en provoquant une nouvelle interruption si l'un des bits du registre IN (INPUT) est armé. L'interruption que provoque `RSI` ne pourra être traitée que lors de l'exécution de la prochaine instruction `RTI` si le SATURN est déjà en train de traiter une interruption.

Mnémoniques	Codes hexadécimaux	Remarques	Retenue	Cycles
INTOFF	808F	Interruptions interdites	non	5
INTON	8080	Interruptions autorisées	non	5
RSI	80810	Interruptions initialisées	non	6

■ Instructions relatives au bus

Le bus sert à l'échange de données entre le microprocesseur SATURN et les différents modules qui composent la mémoire. Les commandes de bus servent surtout à configurer ou à déconfigurer un module de mémoire afin de le rendre visible ou non.

Les instructions BUSCB, BUSCC et BUSCD émettent sur le bus des données dont l'effet est mal connu. On se demande même si ces instructions sont d'une réelle utilité.

SHUTDN place la HP-48 en mode basse consommation (seuls quelques circuits essentiels restent alimentés). Cet état de veille est maintenu jusqu'à ce qu'un événement particulier (passage par zéro de l'horloge, pression sur la touche [ON], etc.) place la machine en mode de fonctionnement normal.

UNCNFG déconfigure le module de mémoire dont l'adresse est placée dans le champ A du registre C. Inversement, CONFIG place le contenu du champ A du registre C dans le registre de configuration du premier module non configuré rencontré (selon l'ordre de la « daisy chain »).

RESET fait passer tous les modules de mémoire configurables (toutes les mémoires sauf la ROM) par un état déconfiguré.

C=ID copie dans le champ A du registre C l'identificateur du module de mémoire courant. Plusieurs cas sont possibles :

- si aucun module de mémoire n'est configuré, le champ A du registre C contient #00000h,
- si le module courant a déjà été configuré (sa taille est déterminée), le champ B du registre C contient :
  - #F2h s'il s'agit du port carte n°2,
  - #F4h s'il s'agit de la mémoire vive interne,
  - #F6h s'il s'agit du gestionnaire de partitions,
  - #F8h s'il s'agit du port carte n°1,

  en annulant les deux quartets de poids faible du registre C (C=0 B), on obtient dans le champ A du registre C la précédente adresse utilisée pour le module de mémoire courant,
- si le module courant est la RAM des entrées/sorties, le champ A du registre C contient la dernière adresse de configuration à laquelle est ajouté #00019h. En général, on obtient donc #00119h,
- si le module de mémoire courant est complètement déconfiguré, ni son adresse de base ni sa taille ne sont définies. On trouve alors dans le champ B du registre C un octet permettant d'identifier le module de mémoire courant :
  - #01h s'il s'agit du port carte n°2,
  - #03h s'il s'agit de la mémoire vive interne,
  - #05h s'il s'agit du gestionnaire de partitions,
  - #07h s'il s'agit du port carte n°1.

SREQ (« service request ») n'est pas utilisé par la HP-48.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
RESET	80A		non	6
CONFIG	805	champ A du registre C	non	11
UNCNFG	804	champ A du registre C	non	12
C=ID	806	champ A du registre C	non	11
SHUTDN	807		non	5
BUSCB	8083		non	10
BUSCC	80B		non	9
BUSCD	808D		non	10

■ Instructions sans effet

Il existe trois instructions sans effet. Loin d'être inutiles elles permettent, entre autres, des temporisations ou la réservation de quartets à l'intérieur du programme pour permettre des ajouts futurs sans qu'il soit nécessaire de reconsidérer tout l'adressage relatif (pour les sauts en particulier). Il existe des instructions sans effet codées sur 3, 4 ou 5 quartets.

Mnémoniques	Codes hexadécimaux	Champ(s) concerné(s)	Retenue	Cycles
NOP3	420	aucun	non	?
NOP4	6300	aucun	non	14
NOP5	64000	aucun	non	14

Attention ! Contrairement à ce que certains débutants ont tendance à croire des « `000...` » à l'intérieur d'un programme sont loin d'être sans effet. Par exemple, l'octet « `00` » peut représenter, selon le cas, l'instruction `RTNYES` (après un test) ou l'instruction `RTNSXM` (dans un sous-programme).

18
Accès à la mémoire masquée

1
La mémoire masquée

A chaque adresse en mémoire correspond un quartet. Les adresses étant codées sur 5 quartets (20 bits), il est possible de désigner 2^{20} quartets différents, c'est-à-dire 512 ko. La mémoire morte a une capacité de 512 ko et utilise donc toutes les adresses disponibles. Une même adresse doit donc pouvoir être utilisée par différents modules de mémoire.

Comme nous l'avons vu, il existe pour chaque module de mémoire un niveau de priorité. Les modules de mémoire sont, par ordre de priorité décroissante :

- la mémoire vive des entrées/sorties (RAM I/O),
- la mémoire vive interne (32 ko ou 128 ko),
- le gestionnaire de partitions,
- la carte en port 1,
- la carte en port 2,
- la mémoire morte.

Avec 5 quartets, l'espace adressable s'étend de #00000h à #FFFFFh. Une adresse en mémoire correspond au quartet se trouvant à ladite adresse dans la mémoire la plus prioritaire. Par exemple, l'adresse #00101h existe en mémoire morte, mais si vous tentez d'accéder au quartet d'adresse #00101h, vous accéderez en fait à la RAM I/O où cette adresse existe aussi car la RAM I/O a un niveau de priorité supérieur à celui de la mémoire morte.

Tenter d'accéder à une adresse existant dans plusieurs modules de mémoire donne accès à ladite adresse dans le module le plus prioritaire parmi ceux où existe cette adresse.

Reportez-vous au chapitre *Plages d'adresses des différentes mémoires* où des tableaux mettent en évidence le masquage de certaines zones de mémoire par des modules plus prioritaires utilisant les mêmes adresses.

2
Utiliser CONFIG et UNCNFG

2.1
Taille et adresse de départ d'un module

Chaque module de mémoire (RAM, RAM I/O, ROM, cartes, etc.) utilise une plage d'adresses. Remarquez que les adresses utilisées par un module ne commencent pas forcément à #00000h. Par exemple, la RAM I/O utilise les adresses #00100h à #0013Fh.

La configuration d'un module de mémoire consiste à lui affecter :

- une taille,
- une adresse de départ.

Ces informations définissent la plage d'adresses utilisées par le module.

Il existe une configuration standard de la mémoire (voir le chapitre : *Plages d'adresses des différentes mémoires*), mais il est possible de reconfigurer la mémoire, et ce, afin de rendre « visibles » des portions de mémoire masquées en configuration standard.

Définir une nouvelle configuration de la mémoire consiste à fixer une nouvelle adresse de départ pour le contenu d'un module. Une fois « déplacé » ce module ne masque plus un second module moins prioritaire, ce dernier est alors accessible. Pour changer la configuration d'un module de mémoire, on commence par le déconfigurer puis on le reconfigure avec une nouvelle adresse de départ et une nouvelle taille.

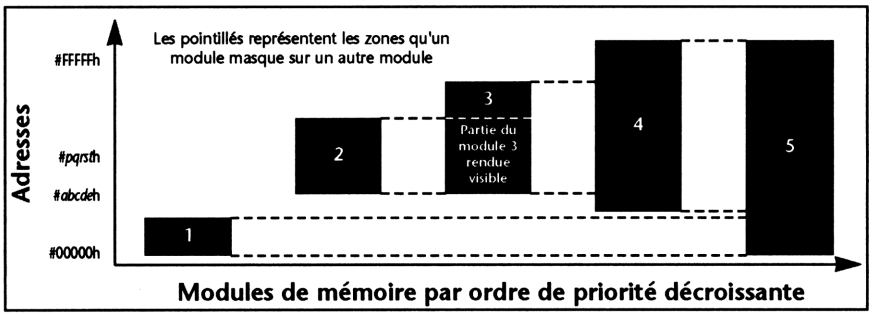
2.2
Déconfigurer un module de mémoire

Le mnémonique de l'instruction de déconfiguration d'un module de mémoire est UNCNFG. On donne en argument à UNCNFG l'adresse de départ du module à déconfigurer. On procède de la façon suivante :

- placez dans le champ A du registre C l'adresse de départ du module de mémoire à déconfigurer. Par exemple, si l'adresse du module à déconfigurer est #*abcde*h, utilisez LCHEX *abcde*,
- exécutez UNCNFG.

Les différents modules de mémoire seront passés en revue par ordre de priorité décroissante (de la RAM I/O à la ROM), le premier module rencontré dont l'adresse de départ est #*abcde*h sera déconfiguré.

CCD 28



Le schéma ci-dessus illustre le phénomène de déconfiguration d'un module de mémoire afin de rendre accessible un autre module. En demandant une déconfiguration de module de mémoire nous n'avons précisé qu'une adresse. Dans le cadre de notre exemple l'adresse #*abcde*h est l'adresse de départ (adresse de base) des modules 2 et 3. Le module 2 ayant un niveau de priorité plus élevé que le module 3, c'est le module 2 qui fait l'objet de la déconfiguration. Le module 2 masquait une partie du module 3. Après la déconfiguration, l'ensemble du module 3 est accessible. Admettons que l'adresse #*pqrst*h existe avant la déconfiguration dans les modules 2 et 3. Lorsque l'on spécifie l'adresse #pqrsth, on obtient :

- avant la déconfiguration du module 2, le quartet d'adresse #*pqrst*h du module 2,
- après le déconfiguration du module 2, le quartet d'adresse #*pqrst*h du module 3 qui était jusqu'ici masqué par le quartet de même adresse du module 2 car le module 2 a un niveau de priorité supérieur à celui du module 3.

On peut aussi envisager le cas d'une double déconfiguration si des cartes sont présentes dans les deux ports (HP-48 GX). Cette double déconfiguration est indispensable pour accéder à la mémoire morte masquée par les cartes des ports 1 et 2 (HP-48 GX). Elle consiste en deux emplois successifs de l'instruction UNCNFG sans changer l'adresse de départ du module à déconfigurer. Pour déconfigurer les deux cartes :

- placez dans le champ A du registre C l'adresse #C0000h qui est l'adresse de départ (adresse de base) des modules de mémoire que constituent les cartes en ports 1 et 2. Utilisez pour cela l'instruction LCHEX prenant #C0000h pour argument,
- exécutez une première fois UNCNFG. Par ordre de priorité décroissante, le premier module de mémoire rencontré configuré à l'adresse #C0000h sera déconfiguré. La carte en port 1 est donc déconfigurée,
- exécutez une seconde fois UNCNFG. L'instruction va à nouveau tenter de déconfigurer un module de mémoire ayant pour adresse de base #C0000h mais cette fois-ci, c'est la carte en port 2 qui est détectée à cette adresse. La carte en port 2 est ainsi déconfigurée. Ni la carte en port 1 ni la carte en port 2 ne masquent plus les adresses en ROM commençant à #C0000h.

2.3
Configurer un module de mémoire

Configurer un module de mémoire revient à lui affecter :

- une adresse de départ (adresse de base) à partir de laquelle tous les quartets de ce module sont numérotés séquentiellement,
- une taille. Cette taille peut être différente de la taille physique du module. Il est ainsi possible de configurer un module de mémoire avec une taille inférieure à sa taille réelle. Dans ce cas, l'espace accessible du module se limite à la taille déclarée. Au contraire, vous pouvez aussi donner au module une taille supérieure à sa taille physique. L'espace défini par la taille déclarée du module contiendra autant de copies de l'étendue physique du module que la taille déclarée peut en accueillir.

Avec ces deux informations on dispose en fait :

- de l'adresse du premier quartet du module (adresse de base),
- de l'adresse du dernier quartet accessible du module (adresse de base augmentée de la taille déclarée du module).

L'instruction utilisée pour configurer un module de mémoire est CONFIG. Celle-ci configure le premier module de mémoire déconfiguré rencontré lorsque sont passés en revue les différents modules dans le sens des niveaux de priorité décroissants. Pour utiliser CONFIG :

- placez dans le champ A du registre C la taille du module de mémoire. Attention ! La taille est ici codée de façon particulière (voir ci dessous),
- exécutez CONFIG,
- placez dans le champ A du registre C l'adresse (sur 5 quartets) à laquelle doit être configuré le module,
- exécutez CONFIG. On a ainsi reconfiguré avec une nouvelle taille et une nouvelle adresse le module non configuré ayant le plus haut niveau de priorité.

Le codage de la taille du module est assez particulier et fait appel à la complémentation. Admettons que l'on souhaite affecter la taille #*a*h au module. On pose : #100000h-#*b*h=#*a*h où #*a*h est la taille à affecter au module et #*b*h, l'argument à donner à CONFIG pour que la taille #*a*h soit affectée au module à configurer. On peut écrire #100000h-#*a*h=#*b*h dont on déduit la valeur de #*b*h.

Pour affecter 8 ko (16384 quartets) à un module on calcule :

#100000h-#*a*h=#*b*h avec #*a*h=#4000h, d'où :
#100000h-#4000h=#*b*h
#*b*h=#FC000h

Rappelons que #4000h = 16384_{10}. Donc, pour affecter à un module une taille de 8 ko (16384 quartets), il faut, à l'aide de LCHEX, charger dans le champ A du registre C la valeur #FC000h avant d'exécuter CONFIG. Notez que la taille configurée doit être un multiple de 2 ko.

La taille d'un module de mémoire ainsi que son adresse doivent impérativement être codées sur 5 quartets. Ajoutez des #0h sur la gauche si cela est nécessaire.

Notez que la ROM (mémoire morte) est le seul des six modules de mémoire qui ne peut pas être déconfiguré ou reconfiguré.

La démarche la plus classique consiste à déconfigurer une mémoire puis à la reconfigurer immédiatement à une autre adresse, et ce, afin de rendre visible un module précédemment masqué par le module reconfiguré. On fait appel à UNCNFG (une fois) puis à CONFIG (deux fois, la première pour la taille, la seconde pour l'adresse).

Considérons le cas d'une HP-48GX dont l'importante RAM interne (128 ko) masque de grandes plages d'adresses des modules moins prioritaires. Il est ici utile d'affecter à la RAM une taille inférieure à sa taille physique. Ainsi, pour transformer notre HP-48GX en HP-48G (qui pose moins de problèmes de ROM masquée) nous procéderons ainsi :

- on déconfigure la RAM interne. Pour cela, on place, à l'aide de LCHEX (ou LC) dans le champ A du registre C l'adresse de base actuelle de la RAM qui est #80000h,
- on exécute UNCNFG qui déconfigure la RAM,
- on charge dans le champ A du registre C la nouvelle taille de la RAM. 32 ko correspondent à 65536 quartets, c'est-à-dire #10000h quartets. Appliquons la formule #100000h-#*a*h=#*b*h. Celle-ci nous indique qu'il faut charger #F0000h dans le champ A du registre C. #F0000h est donc l'argument de LCHEX (ou LC),
- on exécute CONFIG pour affecter cette nouvelle taille au module déconfiguré (ici, la RAM interne de la HP-48 GX),
- on charge dans le champ A du registre C l'adresse de base à laquelle doit êre reconfiguré le module. Nous ne souhaitons pas déplacer la RAM (mais nous pourrions tout à fait le faire), on charge donc #80000h dans le champ A du registre C à l'aide de LCHEX. Il est indispensable de préciser la nouvelle adresse même si celle-ci est identique à l'adresse d'origine, en effet, depuis l'exécution de UNCNFG, le module n'a plus d'adresse de base déclarée,
- on exécute CONFIG. Ce n'est qu'après l'exécution de ce second CONFIG, chargé de fixer l'adresse de départ, qu'est effectivement reconfiguré le module. Cela explique qu'il nous ait fallu préciser l'adresse de base lors de la reconfiguration alors que cette adresse reste finalement inchangée.

La mémoire vive de la HP-48 GX est ainsi limitée à 32 ko mais rend visible des zones masquées des mémoires moins prioritaires (ROM, ports 1 et 2).

Pour rendre à la mémoire vive interne ses attributs initiaux :

- on déconfigure la RAM. Elle se trouve à l'adresse #80000h qui est sa place normale car nous n'avons pas jugé utile de la déplacer à l'occasion de l'exemple précédent. On place donc #80000h dans le champ A du registre C à l'aide de LCHEX (ou LC) puis on exécute UNCNFG,
- on rend à la RAM sa taille normale de 128 ko. 128 ko correspondent à #40000h quartets. Pour obtenir la valeur à placer dans C(A) on calcule #100000h-#40000h. On obtient #C0000h qui est la valeur à placer dans le champ A du registre C. On exécute CONFIG pour affecter cette espace en mémoire à la RAM,

• on indique l'adresse de configuration en plaçant #80000h dans le champ A du registre C puis en exécutant `CONFIG`,
• la RAM de la HP-48 GX est maintenant reconfigurée avec ses attributs initiaux (capacité physique et adressable de 128 ko avec une adresse de départ en #80000h).

2.4
Routines d'accès à la ROM masquée

Les routines sont des programmes en mémoire morte qu'il suffit d'appeler en exécutant l'instruction d'appel de sous-programme `GOSBVL`. Pour exécuter une routine, l'instruction `GOSBVL` et une adresse en mémoire morte non masquée suffisent.

Quelques routines utiles permettent d'accéder aux plages d'adresses masquées de la ROM et des ports :

• la routine en #726A1h donne accès à la ROM masquée par la RAM de la HP-48 G ou par la première tranche de 32 ko de la RAM de la HP-48 GX. Les adresses #80000h à #8FFFFh de la ROM sont ainsi accessibles,
• la routine en #72386h donne accès à la ROM masquée par les 96 derniers ko de la RAM interne de la HP-48 GX. Les adresses #90000h à #BFFFFh de la ROM sont ainsi accessibles,
• la routine en #70B38h donne accès aux adresses #C0000h à #FFFFFh.

3
Les pointeurs étendus

Les pointeurs étendus sont des moyens simples d'accéder à une donnée se trouvant dans une mémoire masquée, en particulier, la ROM masquée. Un pointeur étendu est un objet dont le prologue est 02BAA (AAB20 après inversion des champs pour le codage, voir le chapitre sur les objets).

Un pointeur étendu consiste en l'association de deux adresses :

- la première est l'adresse d'un objet à atteindre. Cet objet se trouve dans une mémoire masquée (la ROM par exemple),
- la seconde adresse est celle d'une routine (ou celle d'un programme développé par vos soins) chargée de rendre visible la zone où se trouve l'objet à atteindre.

Un pointeur étendu comporte donc une adresse d'un objet à atteindre et le moyen d'accès à cette adresse.

Par exemple, créons un pointeur étendu donnant accès à l'adresse #80000h en ROM masquée :

- #02BAAh est le prologue d'un pointeur étendu,
- #80000h est l'adresse à atteindre en ROM masquée,
- #726A1h est l'adresse en ROM non masquée de la routine rendant visible les adresses #80000h à #8FFFFh en ROM masquée.

Après inversion de ces trois champs, on obtient :

```
"AAB20000081A627"
```

Il suffit de placer cette chaîne de codes hexadécimaux (entre " et ") au niveau 1 de la pile et d'exécuter le programme ASS pour créer un objet de type pointeur étendu que vous pourrez ensuite sauvegarder dans une variable en lui associant un nom (entre ' et ') et en appuyant sur [STO]. L'évaluation du pointeur étendu sera ensuite un moyen d'accès à l'adresse normalement masquée que vous avez choisie.

Détail amusant, vous pouvez remplacer les 5 derniers quartets (chargés de coder l'adresse de la routine d'accès à une zone masquée) par #00000h. La HP-48 considère alors que l'adresse de l'objet recherché se trouve dans la mémoire visible en configuration normale. Dans ce cas, l'utilisation d'un pointeur étendu présente un intérêt non négligeable puisqu'il est possible, en évaluant une variable (le pointeur étendu), d'accéder à un point de la ROM non masquée sans avoir à mémoriser une adresse (le pointeur étendu s'en charge). Ainsi, pour créer un pointeur étendu donnant accès à l'adresse #*abcde*h en ROM non masquée, il suffit de soumettre à ASS la chaîne suivante :

```
"AAB20edcba00000"
```

Retenez qu'un pointeur étendu peut nous éviter les fastidieuses opérations de déconfiguration et reconfiguration de la mémoire.

2

Références

Cette seconde partie est consacrée aux nombreuses informations dont il vous faut disposer pour développer vos programmes en langage assembleur.

Les chapitres 19 et 20 présentent des tables faisant correspondre les mnémoniques avec les codes qui leurs sont associés. Il est indispensable de maîtriser les chapitres 13 à 17 de la première partie pour utiliser les chapitres 19 et 20. Le chapitre 19 permet, lors du codage de vos programmes, d'obtenir le code hexadécimal correspondant à un mnémonique donné. Inversement, le chapitre 20 vous permet de retrouver un mnémonique à partir de son code hexadécimal ce qui est utile si vous analysez un programme dont seul le code hexadécimal vous a été communiqué.

Le chapitre 21 présente les masques IN et OUT des touches du clavier. L'utilisation de ces masques est décrite au chapitre 1.

Le chapitre 22 présente la structure des différents objets reconnus par la HP-48. Vous pourrez ainsi, grâce au programme ASS vu dès le chapitre 1, élaborer vos propres objets dont les quartets seront codés par des caractères hexadécimaux au sein d'une chaîne de caractères.

Les chapitres 23 et 24 décrivent la structure de la mémoire vive des entrées/sorties (RAM I/O) et de la mémoire vive du système (RAM système). Connaissant ces mémoires particulières, vous pourrez consulter ou modifier certains paramètres de fonctionnement de la HP-48 à l'aide d'instructions telles que `DAT0=C` (écriture) ou `C=DAT0` (lecture).

Le chapitre 25 présente une sélection de routines utiles. Il s'agit simplement de sous-programmes en ROM qui pourront être appelés à l'aide de l'instruction `GOSBVL`.

Le chapitre 26 est consacré aux codes d'erreur WSLOG.

19

Les instructions classées par mnémoniques

Vous trouverez dans ce chapitre la liste des instructions du langage assembleur classées selon l'ordre alphabétique des mnémoniques. Les tableaux sont divisés en quatre colonnes. La première contient les mnémoniques, un trait en pointillés sépare la partie principale du mnémonique de sa partie variable (qui peut, par exemple, être un nom de champ). La deuxième colonne contient les codes hexadécimaux relatifs aux mnémoniques. Les troisième et quatrième colonnes fournissent des informations complémentaires à propos de l'activité de l'instruction. Ainsi, un « • » dans la colonne « **R** » signale que l'instruction modifie le bit de retenue (carry bit), de même, un « • » dans la colonne « **B** » indique que l'instruction voit son action modifiée en fonction de la base de calcul courante du SATURN. Rappelons que ce dernier peut travailler en décimal ou en hexadécimal.

Les parties variables des mnémoniques sont notées en italique. Ainsi, ***n*** correspond à un nombre hexadécimal compris entre #0h et #Fh. Les adresses (relatives ou non) sont notées ***pq***, ***pqr***, ***pqrs***, ***pqrst*** selon le nombre de quartets utilisés. Chaque lettre ***p***, ***q***, ***r***, ***s*** ou ***t*** représente un nombre hexadécimal (un quartet). L'inversion des champs étant requise, ***pq*** dans un mnémonique deviendra ***qp*** dans le code hexadécimal, de même, ***pqr*** deviendra ***rqp***, etc. Nous avons placé dans les tableaux tous les codes portés à notre connaissance, vous constaterez qu'il existe parfois deux codes hexadécimaux correspondant à un même mnémonique. Par contre, pour ne pas doubler la longueur de la liste, les mnémoniques synonymes n'ont pas été mentionnés. Par exemple, vous trouverez **?B>C** dans la liste, mais vous n'y trouverez pas **?C<B**. De même **CAEX** se trouve dans la liste, mais pas **ACEX**, etc. **LAHEX** et **LCHEX** doivent respectivement être recherchés à **LA** et **LC**. **RTNYES** et **GOYES** sont absentes de cette liste puisqu'elles sont liées aux instructions de test (**?...**). Notez toutefois qu'un décalage d'adresses tel que **#pqh = #00h** correspond à **RTNYES**.

Mnémoniques		Codes hexa	R	B
?A<B	A	8B4*qp*	•	
?A<B	B	9E4*qp*	•	
?A<B	M	9D4*qp*	•	
?A<B	P	984*qp*	•	
?A<B	S	9C4*qp*	•	
?A<B	W	9F4*qp*	•	
?A<B	WP	994*qp*	•	
?A<B	X	9B4*qp*	•	
?A<B	XS	9A4*qp*	•	
?A=0	A	8A8*qp*	•	
?A=0	B	968*qp*	•	
?A=0	M	958*qp*	•	
?A=0	P	908*qp*	•	
?A=0	S	948*qp*	•	
?A=0	W	978*qp*	•	
?A=0	WP	918*qp*	•	
?A=0	X	938*qp*	•	
?A=0	XS	928*qp*	•	
?A=B	A	8A0*qp*	•	
?A=B	B	960*qp*	•	
?A=B	M	950*qp*	•	
?A=B	P	900*qp*	•	
?A=B	S	940*qp*	•	
?A=B	W	970*qp*	•	
?A=B	WP	910*qp*	•	
?A=B	X	930*qp*	•	
?A=B	XS	920*qp*	•	
?A>B	A	8B0*qp*	•	
?A>B	B	9E0*qp*	•	
?A>B	M	9D0*qp*	•	
?A>B	P	980*qp*	•	
?A>B	S	9C0*qp*	•	
?A>B	W	9F0*qp*	•	
?A>B	WP	990*qp*	•	
?A>B	X	9B0*qp*	•	
?A>B	XS	9A0*qp*	•	
?ABIT=0	*n*	8086*nqp*	•	
?ABIT=1	*n*	8087*nqp*	•	
?A≠0	A	8AC*qp*	•	
?A≠0	B	96C*qp*	•	
?A≠0	M	95C*qp*	•	
?A≠0	P	90C*qp*	•	
?A≠0	S	94C*qp*	•	

Mnémoniques		Codes hexa	R	B
?A≠0	W	97C*qp*	•	
?A≠0	WP	91C*qp*	•	
?A≠0	X	93C*qp*	•	
?A≠0	XS	92C*qp*	•	
?A≠B	A	8A4*qp*	•	
?A≠B	B	964*qp*	•	
?A≠B	M	954*qp*	•	
?A≠B	P	904*qp*	•	
?A≠B	S	944*qp*	•	
?A≠B	W	974*qp*	•	
?A≠B	WP	914*qp*	•	
?A≠B	X	934*qp*	•	
?A≠B	XS	924*qp*	•	
?A≤B	A	8BC*qp*	•	
?A≤B	B	9EC*qp*	•	
?A≤B	M	9DC*qp*	•	
?A≤B	P	98C*qp*	•	
?A≤B	S	9CC*qp*	•	
?A≤B	W	9FC*qp*	•	
?A≤B	WP	99C*qp*	•	
?A≤B	X	9BC*qp*	•	
?A≤B	XS	9AC*qp*	•	
?A≥B	A	8B8*qp*	•	
?A≥B	B	9E8*qp*	•	
?A≥B	M	9D8*qp*	•	
?A≥B	P	988*qp*	•	
?A≥B	S	9C8*qp*	•	
?A≥B	W	9F8*qp*	•	
?A≥B	WP	998*qp*	•	
?A≥B	X	9B8*qp*	•	
?A≥B	XS	9A8*qp*	•	
?B<C	A	8B5*qp*	•	
?B<C	B	9E5*qp*	•	
?B<C	M	9D5*qp*	•	
?B<C	P	985*qp*	•	
?B<C	S	9C5*qp*	•	
?B<C	W	9F5*qp*	•	
?B<C	WP	995*qp*	•	
?B<C	X	9B5*qp*	•	
?B<C	XS	9A5*qp*	•	
?B=0	A	8A9*qp*	•	
?B=0	B	969*qp*	•	
?B=0	M	959*qp*	•	

Mnémoniques		Codes hexa	R	B
?B=0	P	909*qp*	•	
?B=0	S	949*qp*	•	
?B=0	W	979*qp*	•	
?B=0	WP	919*qp*	•	
?B=0	X	939*qp*	•	
?B=0	XS	929*qp*	•	
?B=C	A	8A1*qp*	•	
?B=C	B	961*qp*	•	
?B=C	M	951*qp*	•	
?B=C	P	901*qp*	•	
?B=C	S	941*qp*	•	
?B=C	W	971*qp*	•	
?B=C	WP	911*qp*	•	
?B=C	X	931*qp*	•	
?B=C	XS	921*qp*	•	
?B>C	A	8B1*qp*	•	
?B>C	B	9E1*qp*	•	
?B>C	M	9D1*qp*	•	
?B>C	P	981*qp*	•	
?B>C	S	9C1*qp*	•	
?B>C	W	9F1*qp*	•	
?B>C	WP	991*qp*	•	
?B>C	X	9B1*qp*	•	
?B>C	XS	9A1*qp*	•	
?B≠0	A	8AD*qp*	•	
?B≠0	B	96D*qp*	•	
?B≠0	M	95D*qp*	•	
?B≠0	P	90D*qp*	•	
?B≠0	S	94D*qp*	•	
?B≠0	W	97D*qp*	•	
?B≠0	WP	91D*qp*	•	
?B≠0	X	93D*qp*	•	
?B≠0	XS	92D*qp*	•	
?B≠C	A	8A5*qp*	•	
?B≠C	B	965*qp*	•	
?B≠C	M	955*qp*	•	
?B≠C	P	905*qp*	•	
?B≠C	S	945*qp*	•	
?B≠C	W	975*qp*	•	
?B≠C	WP	915*qp*	•	
?B≠C	X	935*qp*	•	
?B≠C	XS	925*qp*	•	
?B≤C	A	8BD*qp*	•	

Mnémoniques		Codes hexa	R	B
?B≤C	B	9ED*qp*	•	
?B≤C	M	9DD*qp*	•	
?B≤C	P	98D*qp*	•	
?B≤C	S	9CD*qp*	•	
?B≤C	W	9FD*qp*	•	
?B≤C	WP	99D*qp*	•	
?B≤C	X	9BD*qp*	•	
?B≤C	XS	9AD*qp*	•	
?B≥C	A	8B9*qp*	•	
?B≥C	B	9E9*qp*	•	
?B≥C	M	9D9*qp*	•	
?B≥C	P	989*qp*	•	
?B≥C	S	9C9*qp*	•	
?B≥C	W	9F9*qp*	•	
?B≥C	WP	999*qp*	•	
?B≥C	X	9B9*qp*	•	
?B≥C	XS	9A9*qp*	•	
?C<A	A	8B6*qp*	•	
?C<A	B	9E6*qp*	•	
?C<A	M	9D6*qp*	•	
?C<A	P	986*qp*	•	
?C<A	S	9C6*qp*	•	
?C<A	W	9F6*qp*	•	
?C<A	WP	996*qp*	•	
?C<A	X	9B6*qp*	•	
?C<A	XS	9A6*qp*	•	
?C=0	A	8AA*qp*	•	
?C=0	B	96A*qp*	•	
?C=0	M	95A*qp*	•	
?C=0	P	90A*qp*	•	
?C=0	S	94A*qp*	•	
?C=0	W	97A*qp*	•	
?C=0	WP	91A*qp*	•	
?C=0	X	93A*qp*	•	
?C=0	XS	92A*qp*	•	
?C=A	A	8A2*qp*	•	
?C=A	B	962*qp*	•	
?C=A	M	952*qp*	•	
?C=A	P	902*qp*	•	
?C=A	S	942*qp*	•	
?C=A	W	972*qp*	•	
?C=A	WP	912*qp*	•	
?C=A	X	932*qp*	•	

Mnémoniques		Codes hexa	R	B
?C=A	XS	922*qp*	•	
?C>A	A	8B2*qp*	•	
?C>A	B	9E2*qp*	•	
?C>A	M	9D2*qp*	•	
?C>A	P	982*qp*	•	
?C>A	S	9C2*qp*	•	
?C>A	W	9F2*qp*	•	
?C>A	WP	992*qp*	•	
?C>A	X	9B2*qp*	•	
?C>A	XS	9A2*qp*	•	
?CBIT=0	*n*	808A*nqp*	•	
?CBIT=1	*n*	808B*nqp*	•	
?C≠0	A	8AE*qp*	•	
?C≠0	B	96E*qp*	•	
?C≠0	M	95E*qp*	•	
?C≠0	P	90E*qp*	•	
?C≠0	S	94E*qp*	•	
?C≠0	W	97E*qp*	•	
?C≠0	WP	91E*qp*	•	
?C≠0	X	93E*qp*	•	
?C≠0	XS	92E*qp*	•	
?C≠A	A	8A6*qp*	•	
?C≠A	B	966*qp*	•	
?C≠A	M	956*qp*	•	
?C≠A	P	906*qp*	•	
?C≠A	S	946*qp*	•	
?C≠A	W	976*qp*	•	
?C≠A	WP	916*qp*	•	
?C≠A	X	936*qp*	•	
?C≠A	XS	926*qp*	•	
?C≤A	A	8BE*qp*	•	
?C≤A	B	9EE*qp*	•	
?C≤A	M	9DE*qp*	•	
?C≤A	P	98E*qp*	•	
?C≤A	S	9CE*qp*	•	
?C≤A	W	9FE*qp*	•	
?C≤A	WP	99E*qp*	•	
?C≤A	X	9BE*qp*	•	
?C≤A	XS	9AE*qp*	•	
?C≥A	A	8BA*qp*	•	
?C≥A	B	9EA*qp*	•	
?C≥A	M	9DA*qp*	•	
?C≥A	P	98A*qp*	•	

Mnémoniques		Codes hexa	R	B
?C≥A	S	9CA*qp*	•	
?C≥A	W	9FA*qp*	•	
?C≥A	WP	99A*qp*	•	
?C≥A	X	9BA*qp*	•	
?C≥A	XS	9AA*qp*	•	
?D<C	A	8B7*qp*	•	
?D<C	B	9E7*qp*	•	
?D<C	M	9D7*qp*	•	
?D<C	P	987*qp*	•	
?D<C	S	9C7*qp*	•	
?D<C	W	9F7*qp*	•	
?D<C	WP	997*qp*	•	
?D<C	X	9B7*qp*	•	
?D<C	XS	9A7*qp*	•	
?D=0	A	8AB*qp*	•	
?D=0	B	96B*qp*	•	
?D=0	M	95B*qp*	•	
?D=0	P	90B*qp*	•	
?D=0	S	94B*qp*	•	
?D=0	W	97B*qp*	•	
?D=0	WP	91B*qp*	•	
?D=0	X	93B*qp*	•	
?D=0	XS	92B*qp*	•	
?D=C	A	8A3*qp*	•	
?D=C	B	963*qp*	•	
?D=C	M	953*qp*	•	
?D=C	P	903*qp*	•	
?D=C	S	943*qp*	•	
?D=C	W	973*qp*	•	
?D=C	WP	913*qp*	•	
?D=C	X	933*qp*	•	
?D=C	XS	923*qp*	•	
?D>C	A	8B3*qp*	•	
?D>C	B	9E3*qp*	•	
?D>C	M	9D3*qp*	•	
?D>C	P	983*qp*	•	
?D>C	S	9C3*qp*	•	
?D>C	W	9F3*qp*	•	
?D>C	WP	993*qp*	•	
?D>C	X	9B3*qp*	•	
?D>C	XS	9A3*qp*	•	
?D≠0	A	8AF*qp*	•	
?D≠0	B	96F*qp*	•	

Mnémoniques		Codes hexa	R	B
?D≠0	M	95F*qp*	•	
?D≠0	P	90F*qp*	•	
?D≠0	S	94F*qp*	•	
?D≠0	W	97F*qp*	•	
?D≠0	WP	91F*qp*	•	
?D≠0	X	93F*qp*	•	
?D≠0	XS	92F*qp*	•	
?D≠C	A	8A7*qp*	•	
?D≠C	B	967*qp*	•	
?D≠C	M	957*qp*	•	
?D≠C	P	907*qp*	•	
?D≠C	S	947*qp*	•	
?D≠C	W	977*qp*	•	
?D≠C	WP	917*qp*	•	
?D≠C	X	937*qp*	•	
?D≠C	XS	927*qp*	•	
?D≤C	A	8BF*qp*	•	
?D≤C	B	9EF*qp*	•	
?D≤C	M	9DF*qp*	•	
?D≤C	P	98F*qp*	•	
?D≤C	S	9CF*qp*	•	
?D≤C	W	9FF*qp*	•	
?D≤C	WP	99F*qp*	•	
?D≤C	X	9BF*qp*	•	
?D≤C	XS	9AF*qp*	•	
?D≥C	A	8BB*qp*	•	
?D≥C	B	9EB*qp*	•	
?D≥C	M	9DB*qp*	•	
?D≥C	P	98B*qp*	•	
?D≥C	S	9CB*qp*	•	
?D≥C	W	9FB*qp*	•	
?D≥C	WP	99B*qp*	•	
?D≥C	X	9BB*qp*	•	
?D≥C	XS	9AB*qp*	•	
?HS=0	*n*	83*nqp*	•	
?MP=0		838*qp*	•	
?P=	*n*	89*nqp*	•	
?P≠	*n*	88*nqp*	•	
?SB=0		832*qp*	•	
?SR=0		834*qp*	•	
?ST=0	*n*	86*nqp*	•	
?ST=1	*n*	87*nqp*	•	
?XM=0		831*qp*	•	

Mnémoniques		Codes hexa	R	B
A=-A	A	F8	•	•
A=-A	B	BE8	•	•
A=-A	M	BD8	•	•
A=-A	P	B88	•	•
A=-A	S	BC8	•	•
A=-A	W	BF8	•	•
A=-A	WP	B98	•	•
A=-A	X	BB8	•	•
A=-A	XS	BA8	•	•
A=-A-1	A	FC	•	•
A=-A-1	B	BEC	•	•
A=-A-1	M	BDC	•	•
A=-A-1	P	B8C	•	•
A=-A-1	S	BCC	•	•
A=-A-1	W	BFC	•	•
A=-A-1	WP	B9C	•	•
A=-A-1	X	BBC	•	•
A=-A-1	XS	BAC	•	•
A=0	A	D0		
A=0	B	AE0		
A=0	M	AD0		
A=0	P	A80		
A=0	S	AC0		
A=0	W	AF0		
A=0	WP	A90		
A=0	X	AB0		
A=0	XS	AA0		
A=A!B	A	0EF8		
A=A!B	B	0E68		
A=A!B	M	0E58		
A=A!B	P	0E08		
A=A!B	S	0E48		
A=A!B	W	0E78		
A=A!B	WP	0E18		
A=A!B	X	0E38		
A=A!B	XS	0E28		
A=A&B	A	0EF0		
A=A&B	B	0E60		
A=A&B	M	0E50		
A=A&B	P	0E00		
A=A&B	S	0E40		
A=A&B	W	0E70		
A=A&B	WP	0E10		

Mnémoniques		Codes hexa	R	B
A=A&B	X	0E30		
A=A&B	XS	0E20		
A=A+ *(n+1)*	A	818F0*n*		
A=A+ *(n+1)*	B	81860*n*		
A=A+ *(n+1)*	M	81850*n*		
A=A+ *(n+1)*	P	81800*n*		
A=A+ *(n+1)*	S	81840*n*		
A=A+ *(n+1)*	W	81870*n*		
A=A+ *(n+1)*	WP	81810*n*		
A=A+ *(n+1)*	X	81830*n*		
A=A+ *(n+1)*	XS	81820*n*		
A=A+1	A	E4	•	•
A=A+1	B	B64	•	•
A=A+1	M	B54	•	•
A=A+1	P	B04	•	•
A=A+1	S	B44	•	•
A=A+1	W	B74	•	•
A=A+1	WP	B14	•	•
A=A+1	X	B34	•	•
A=A+1	XS	B24	•	•
A=A+A	A	C4	•	•
A=A+A	B	A64	•	•
A=A+A	M	A54	•	•
A=A+A	P	A04	•	•
A=A+A	S	A44	•	•
A=A+A	W	A74	•	•
A=A+A	WP	A14	•	•
A=A+A	X	A34	•	•
A=A+A	XS	A24	•	•
A=A+B	A	C0	•	•
A=A+B	B	A60	•	•
A=A+B	M	A50	•	•
A=A+B	P	A00	•	•
A=A+B	S	A40	•	•
A=A+B	W	A70	•	•
A=A+B	WP	A10	•	•
A=A+B	X	A30	•	•
A=A+B	XS	A20	•	•
A=A- *(n+1)*	A	818F8*n*	•	
A=A- *(n+1)*	B	81868*n*	•	
A=A- *(n+1)*	M	81858*n*	•	
A=A- *(n+1)*	P	81808*n*	•	
A=A- *(n+1)*	S	81848*n*	•	

Mnémoniques		Codes hexa	R	B
A=A- *(n+1)*	W	81878*n*	•	
A=A- *(n+1)*	WP	81818*n*	•	
A=A- *(n+1)*	X	81838*n*	•	
A=A- *(n+1)*	XS	81828*n*	•	
A=A-1	A	CC	•	•
A=A-1	B	A6C	•	•
A=A-1	M	A5C	•	•
A=A-1	P	A0C	•	•
A=A-1	S	A4C	•	•
A=A-1	W	A7C	•	•
A=A-1	WP	A1C	•	•
A=A-1	X	A3C	•	•
A=A-1	XS	A2C	•	•
A=A-B	A	E0	•	•
A=A-B	B	B60	•	•
A=A-B	M	B50	•	•
A=A-B	P	B00	•	•
A=A-B	S	B40	•	•
A=A-B	W	B70	•	•
A=A-B	WP	B10	•	•
A=A-B	X	B30	•	•
A=A-B	XS	B20	•	•
A=A-C	A	EA	•	•
A=A-C	B	B6A	•	•
A=A-C	M	B5A	•	•
A=A-C	P	B0A	•	•
A=A-C	S	B4A	•	•
A=A-C	W	B7A	•	•
A=A-C	WP	B1A	•	•
A=A-C	X	B3A	•	•
A=A-C	XS	B2A	•	•
A=B	A	D4		
A=B	B	AE4		
A=B	M	AD4		
A=B	P	A84		
A=B	S	AC4		
A=B	W	AF4		
A=B	WP	A94		
A=B	X	AB4		
A=B	XS	AA4		
A=B-A	A	EC	•	•
A=B-A	B	B6C	•	•
A=B-A	M	B5C	•	•

Mnémoniques		Codes hexa	R	B
A=B-A	P	B0C	•	•
A=B-A	S	B4C	•	•
A=B-A	W	B7C	•	•
A=B-A	WP	B1C	•	•
A=B-A	X	B3C	•	•
A=B-A	XS	B2C	•	•
A=C	A	DA	•	
A=C	B	AEA	•	
A=C	M	ADA	•	
A=C	P	A8A	•	
A=C	S	ACA	•	
A=C	W	AFA	•	
A=C	WP	A9A	•	
A=C	X	ABA	•	
A=C	XS	AAA	•	
A=C!A	A	0EFE		
A=C!A	B	0E6E		
A=C!A	M	0E5E		
A=C!A	P	0E0E		
A=C!A	S	0E4E		
A=C!A	W	0E7E		
A=C!A	WP	0E1E		
A=C!A	X	0E3E		
A=C!A	XS	0E2E		
A=C&A	A	0EF6		
A=C&A	B	0E66		
A=C&A	M	0E56		
A=C&A	P	0E06		
A=C&A	S	0E46		
A=C&A	W	0E76		
A=C&A	WP	0E16		
A=C&A	X	0E36		
A=C&A	XS	0E26		
A=C+A	A	CA	•	•
A=C+A	B	A6A	•	•
A=C+A	M	A5A	•	•
A=C+A	P	A0A	•	•
A=C+A	S	A4A	•	•
A=C+A	W	A7A	•	•
A=C+A	WP	A1A	•	•
A=C+A	X	A3A	•	•
A=C+A	XS	A2A	•	•
A=DAT0	*(n+1)*	15A*n*		

Mnémoniques		Codes hexa	R	B
A=DAT0	A	142		
A=DAT0	B	1526		
A=DAT0	B	14A		
A=DAT0	M	1525		
A=DAT0	P	1520		
A=DAT0	S	1524		
A=DAT0	W	1527		
A=DAT0	WP	1521		
A=DAT0	X	1523		
A=DAT0	XS	1522		
A=DAT1	A	143		
A=DAT1	B	1536		
A=DAT1	B	14B		
A=DAT1	M	1535		
A=DAT1	P	1530		
A=DAT1	S	1534		
A=DAT1	W	1537		
A=DAT1	WP	1531		
A=DAT1	X	1533		
A=DAT1	XS	1532		
A=IN		802		
A=PC		81B4		
A=R0		110		
A=R0.F	A	81AF10		
A=R0.F	B	81A610		
A=R0.F	M	81A510		
A=R0.F	P	81A010		
A=R0.F	S	81A410		
A=R0.F	W	81A710		
A=R0.F	WP	81A110		
A=R0.F	X	81A310		
A=R0.F	XS	81A210		
A=R1		111		
A=R1.F	A	81AF11		
A=R1.F	B	81A611		
A=R1.F	M	81A511		
A=R1.F	P	81A011		
A=R1.F	S	81A411		
A=R1.F	W	81A711		
A=R1.F	WP	81A111		
A=R1.F	X	81A311		
A=R1.F	XS	81A211		
A=R2		112		

Mnémoniques		Codes hexa	R	B
A=R2.F	A	81AF12		
A=R2.F	B	81A612		
A=R2.F	M	81A512		
A=R2.F	P	81A012		
A=R2.F	S	81A412		
A=R2.F	W	81A712		
A=R2.F	WP	81A112		
A=R2.F	X	81A312		
A=R2.F	XS	81A212		
A=R3		113		
A=R3.F	A	81AF13		
A=R3.F	B	81A613		
A=R3.F	M	81A513		
A=R3.F	P	81A013		
A=R3.F	S	81A413		
A=R3.F	W	81A713		
A=R3.F	WP	81A113		
A=R3.F	X	81A313		
A=R3.F	XS	81A213		
A=R4	A	81AF14		
A=R4	B	81A614		
A=R4	M	81A514		
A=R4	P	81A014		
A=R4	S	81A414		
A=R4	W	81A714		
A=R4	WP	81A114		
A=R4	X	81A314		
A=R4	XS	81A214		
A=R4		114		
ABEX	A	DC		
ABEX	B	AEC		
ABEX	M	ADC		
ABEX	P	A8C		
ABEX	S	ACC		
ABEX	W	AFC		
ABEX	WP	A9C		
ABEX	X	ABC		
ABEX	XS	AAC		
ABIT=0	*n*	8084*n*		
ABIT=1	*n*	8085*n*		
AD0EX		132		
AD0XS		13A		
AD1EX		133		

Mnémoniques		Codes hexa	R	B
AD1XS		13B		
APCEX		81B6		
AR0EX		120		
AR0EX.F	A	81AF20		
AR0EX.F	B	81A620		
AR0EX.F	M	81A520		
AR0EX.F	P	81A020		
AR0EX.F	S	81A420		
AR0EX.F	W	81A720		
AR0EX.F	WP	81A120		
AR0EX.F	X	81A320		
AR0EX.F	XS	81A220		
AR1EX		121		
AR1EX.F	A	81AF21		
AR1EX.F	B	81A621		
AR1EX.F	M	81A521		
AR1EX.F	P	81A021		
AR1EX.F	S	81A421		
AR1EX.F	W	81A721		
AR1EX.F	WP	81A121		
AR1EX.F	X	81A321		
AR1EX.F	XS	81A221		
AR2EX		122		
AR2EX.F	A	81AF22		
AR2EX.F	B	81A622		
AR2EX.F	M	81A522		
AR2EX.F	P	81A022		
AR2EX.F	S	81A422		
AR2EX.F	W	81A722		
AR2EX.F	WP	81A122		
AR2EX.F	X	81A322		
AR2EX.F	XS	81A222		
AR3EX		123		
AR3EX.F	A	81AF23		
AR3EX.F	B	81A623		
AR3EX.F	M	81A523		
AR3EX.F	P	81A023		
AR3EX.F	S	81A423		
AR3EX.F	W	81A723		
AR3EX.F	WP	81A123		
AR3EX.F	X	81A323		
AR3EX.F	XS	81A223		
AR4EX		124		

Mnémoniques		Codes hexa	R	B
AR4EX.F	A	81AF24		
AR4EX.F	B	81A624		
AR4EX.F	M	81A524		
AR4EX.F	P	81A024		
AR4EX.F	S	81A424		
AR4EX.F	W	81A724		
AR4EX.F	WP	81A124		
AR4EX.F	X	81A324		
AR4EX.F	XS	81A224		
ASL	A	F0		
ASL	B	BE0		
ASL	M	BD0		
ASL	P	B80		
ASL	S	BC0		
ASL	W	BF0		
ASL	WP	B90		
ASL	X	BB0		
ASL	XS	BA0		
ASLC		810		
ASR	A	F4		
ASR	B	BE4		
ASR	M	BD4		
ASR	P	B84		
ASR	S	BC4		
ASR	W	BF4		
ASR	WP	B94		
ASR	X	BB4		
ASR	XS	BA4		
ASRB		81C		
ASRB.F	A	819F0		
ASRB.F	B	81960		
ASRB.F	M	81950		
ASRB.F	P	81900		
ASRB.F	S	81940		
ASRB.F	W	81970		
ASRB.F	WP	81910		
ASRB.F	X	81930		
ASRB.F	XS	81920		
ASRC		814		
B=-B	A	F9	•	•
B=-B	B	BE9	•	•
B=-B	M	BD9	•	•
B=-B	P	B89	•	•

Mnémoniques		Codes hexa	R	B
B=-B	S	BC9	•	•
B=-B	W	BF9	•	•
B=-B	WP	B99	•	•
B=-B	X	BB9	•	•
B=-B	XS	BA9	•	•
B=-B-1	A	FD	•	•
B=-B-1	B	BED	•	•
B=-B-1	M	BDD	•	•
B=-B-1	P	B8D	•	•
B=-B-1	S	BCD	•	•
B=-B-1	W	BFD	•	•
B=-B-1	WP	B9D	•	•
B=-B-1	X	BBD	•	•
B=-B-1	XS	BAD	•	•
B=0	A	D1		
B=0	B	AE1		
B=0	M	AD1		
B=0	P	A81		
B=0	S	AC1		
B=0	W	AF1		
B=0	WP	A91		
B=0	X	AB1		
B=0	XS	AA1		
B=A	A	D8		
B=A	B	AE8		
B=A	M	AD8		
B=A	P	A88		
B=A	S	AC8		
B=A	W	AF8		
B=A	WP	A98		
B=A	X	AB8		
B=A	XS	AA8		
B=A!B	A	0EFC		
B=A!B	B	0E6C		
B=A!B	M	0E5C		
B=A!B	P	0E0C		
B=A!B	S	0E4C		
B=A!B	W	0E7C		
B=A!B	WP	0E1C		
B=A!B	X	0E3C		
B=A!B	XS	0E2C		
B=A&B	A	0EF4		
B=A&B	B	0E64		

Mnémoniques		Codes hexa	R	B
B=A&B	M	0E54		
B=A&B	P	0E04		
B=A&B	S	0E44		
B=A&B	W	0E74		
B=A&B	WP	0E14		
B=A&B	X	0E34		
B=A&B	XS	0E24		
B=A+B	A	C8	•	•
B=A+B	B	A68	•	•
B=A+B	M	A58	•	•
B=A+B	P	A08	•	•
B=A+B	S	A48	•	•
B=A+B	W	A78	•	•
B=A+B	WP	A18	•	•
B=A+B	X	A38	•	•
B=A+B	XS	A28	•	•
B=B!C	A	0EF9		
B=B!C	B	0E69		
B=B!C	M	0E59		
B=B!C	P	0E09		
B=B!C	S	0E49		
B=B!C	W	0E79		
B=B!C	WP	0E19		
B=B!C	X	0E39		
B=B!C	XS	0E29		
B=B&C	A	0EF1		
B=B&C	B	0E61		
B=B&C	M	0E51		
B=B&C	P	0E01		
B=B&C	S	0E41		
B=B&C	W	0E71		
B=B&C	WP	0E11		
B=B&C	X	0E31		
B=B&C	XS	0E21		
B=B+ *(n+1)*	A	818F1*n*	•	
B=B+ *(n+1)*	B	81861*n*	•	
B=B+ *(n+1)*	M	81851*n*	•	
B=B+ *(n+1)*	P	81801*n*	•	
B=B+ *(n+1)*	S	81841*n*	•	
B=B+ *(n+1)*	W	81871*n*	•	
B=B+ *(n+1)*	WP	81811*n*	•	
B=B+ *(n+1)*	X	81831*n*	•	
B=B+ *(n+1)*	XS	81821*n*	•	

Mnémoniques		Codes hexa	R	B
B=B+1	A	E5	•	•
B=B+1	B	B65	•	•
B=B+1	M	B55	•	•
B=B+1	P	B05	•	•
B=B+1	S	B45	•	•
B=B+1	W	B75	•	•
B=B+1	WP	B15	•	•
B=B+1	X	B35	•	•
B=B+1	XS	B25	•	•
B=B+B	A	C5	•	•
B=B+B	B	A65	•	•
B=B+B	M	A55	•	•
B=B+B	P	A05	•	•
B=B+B	S	A45	•	•
B=B+B	W	A75	•	•
B=B+B	WP	A15	•	•
B=B+B	X	A35	•	•
B=B+B	XS	A25	•	•
B=B+C	A	C1	•	•
B=B+C	B	A61	•	•
B=B+C	M	A51	•	•
B=B+C	P	A01	•	•
B=B+C	S	A41	•	•
B=B+C	W	A71	•	•
B=B+C	WP	A11	•	•
B=B+C	X	A31	•	•
B=B+C	XS	A21	•	•
B=B- *(n+1)*	A	818F9*n*	•	
B=B- *(n+1)*	B	81869*n*	•	
B=B- *(n+1)*	M	81859*n*	•	
B=B- *(n+1)*	P	81809*n*	•	
B=B- *(n+1)*	S	81849*n*	•	
B=B- *(n+1)*	W	81879*n*	•	
B=B- *(n+1)*	WP	81819*n*	•	
B=B- *(n+1)*	X	81839*n*	•	
B=B- *(n+1)*	XS	81829*n*	•	
B=B-1	A	CD	•	•
B=B-1	B	A6D	•	•
B=B-1	M	A5D	•	•
B=B-1	P	A0D	•	•
B=B-1	S	A4D	•	•
B=B-1	W	A7D	•	•
B=B-1	WP	A1D	•	•

Mnémoniques		Codes hexa	R	B
B=B-1	X	A3D	•	•
B=B-1	XS	A2D	•	•
B=B-A	A	E8	•	•
B=B-A	B	B68	•	•
B=B-A	M	B58	•	•
B=B-A	P	B08	•	•
B=B-A	S	B48	•	•
B=B-A	W	B78	•	•
B=B-A	WP	B18	•	•
B=B-A	X	B38	•	•
B=B-A	XS	B28	•	•
B=B-C	A	E1	•	•
B=B-C	B	B61	•	•
B=B-C	M	B51	•	•
B=B-C	P	B01	•	•
B=B-C	S	B41	•	•
B=B-C	W	B71	•	•
B=B-C	WP	B11	•	•
B=B-C	X	B31	•	•
B=B-C	XS	B21	•	•
B=C	A	D5		
B=C	B	AE5		
B=C	M	AD5		
B=C	P	A85		
B=C	S	AC5		
B=C	W	AF5		
B=C	WP	A95		
B=C	X	AB5		
B=C	XS	AA5		
B=C-B	A	ED	•	•
B=C-B	B	B6D	•	•
B=C-B	M	B5D	•	•
B=C-B	P	B0D	•	•
B=C-B	S	B4D	•	•
B=C-B	W	B7D	•	•
B=C-B	WP	B1D	•	•
B=C-B	X	B3D	•	•
B=C-B	XS	B2D	•	•
BCEX	A	DD		
BCEX	B	AED		
BCEX	M	ADD		
BCEX	P	A8D		
BCEX	S	ACD		

Mnémoniques		Codes hexa	R	B
BCEX	W	AFD		
BCEX	WP	A9D		
BCEX	X	ABD		
BCEX	XS	AAD		
BSL	A	F1		
BSL	B	BE1		
BSL	M	BD1		
BSL	P	B81		
BSL	S	BC1		
BSL	W	BF1		
BSL	WP	B91		
BSL	X	BB1		
BSL	XS	BA1		
BSLC		811		
BSR	A	F5		
BSR	B	BE5		
BSR	M	BD5		
BSR	P	B85		
BSR	S	BC5		
BSR	W	BF5		
BSR	WP	B95		
BSR	X	BB5		
BSR	XS	BA5		
BSRB		81D		
BSRB.F	A	819F1		
BSRB.F	B	81961		
BSRB.F	M	81951		
BSRB.F	P	81901		
BSRB.F	S	81941		
BSRB.F	W	81971		
BSRB.F	WP	81911		
BSRB.F	X	81931		
BSRB.F	XS	81921		
BSRC		815		
BUSCB		8083		
BUSCC		80B		
BUSCD		808D		
C+P+1		809	•	
C=-C	A	FA	•	•
C=-C	B	BEA	•	•
C=-C	M	BDA	•	•
C=-C	P	B8A	•	•
C=-C	S	BCA	•	•

Mnémoniques		Codes hexa	R	B
C=-C	W	BFA	•	•
C=-C	WP	B9A	•	•
C=-C	X	BBA	•	•
C=-C	XS	BAA	•	•
C=-C-1	A	FE	•	•
C=-C-1	B	BEE	•	•
C=-C-1	M	BDE	•	•
C=-C-1	P	B8E	•	•
C=-C-1	S	BCE	•	•
C=-C-1	W	BFE	•	•
C=-C-1	WP	B9E	•	•
C=-C-1	X	BBE	•	•
C=-C-1	XS	BAE	•	•
C=0	A	D2		
C=0	B	AE2		
C=0	M	AD2		
C=0	P	A82		
C=0	S	AC2		
C=0	W	AF2		
C=0	WP	A92		
C=0	X	AB2		
C=0	XS	AA2		
C=A	A	D6		
C=A	B	AE6		
C=A	M	AD6		
C=A	P	A86		
C=A	S	AC6		
C=A	W	AF6		
C=A	WP	A96		
C=A	X	AB6		
C=A	XS	AA6		
C=A-C	A	EE	•	•
C=A-C	B	B6E	•	•
C=A-C	M	B5E	•	•
C=A-C	P	B0E	•	•
C=A-C	S	B4E	•	•
C=A-C	W	B7E	•	•
C=A-C	WP	B1E	•	•
C=A-C	X	B3E	•	•
C=A-C	XS	B2E	•	•
C=B	A	D9		
C=B	B	AE9		
C=B	M	AD9		

Mnémoniques		Codes hexa	R	B
C=B	P	A89		
C=B	S	AC9		
C=B	W	AF9		
C=B	WP	A99		
C=B	X	AB9		
C=B	XS	AA9		
C=B!C	A	0EFD		
C=B!C	B	0E6D		
C=B!C	M	0E5D		
C=B!C	P	0E0D		
C=B!C	S	0E4D		
C=B!C	W	0E7D		
C=B!C	WP	0E1D		
C=B!C	X	0E3D		
C=B!C	XS	0E2D		
C=B&C	A	0EF5		
C=B&C	B	0E65		
C=B&C	M	0E55		
C=B&C	P	0E05		
C=B&C	S	0E45		
C=B&C	W	0E75		
C=B&C	WP	0E15		
C=B&C	X	0E35		
C=B&C	XS	0E25		
C=B+C	A	C9	•	•
C=B+C	B	A69	•	•
C=B+C	M	A59	•	•
C=B+C	P	A09	•	•
C=B+C	S	A49	•	•
C=B+C	W	A79	•	•
C=B+C	WP	A19	•	•
C=B+C	X	A39	•	•
C=B+C	XS	A29	•	•
C=C!A	A	0EFA		
C=C!A	B	0E6A		
C=C!A	M	0E5A		
C=C!A	P	0E0A		
C=C!A	S	0E4A		
C=C!A	W	0E7A		
C=C!A	WP	0E1A		
C=C!A	X	0E3A		
C=C!A	XS	0E2A		
C=C&A	A	0EF2		

Mnémoniques		Codes hexa	R	B
C=C&A	B	0E62		
C=C&A	M	0E52		
C=C&A	P	0E02		
C=C&A	S	0E42		
C=C&A	W	0E72		
C=C&A	WP	0E12		
C=C&A	X	0E32		
C=C&A	XS	0E22		
C=C+ *(n+1)*	A	818F2*n*	•	
C=C+ *(n+1)*	B	81862*n*	•	
C=C+ *(n+1)*	M	81852*n*	•	
C=C+ *(n+1)*	P	81802*n*	•	
C=C+ *(n+1)*	S	81842*n*	•	
C=C+ *(n+1)*	W	81872*n*	•	
C=C+ *(n+1)*	WP	81812*n*	•	
C=C+ *(n+1)*	X	81832*n*	•	
C=C+ *(n+1)*	XS	81822*n*	•	
C=C+1	A	E6	•	•
C=C+1	B	B66	•	•
C=C+1	M	B56	•	•
C=C+1	P	B06	•	•
C=C+1	S	B46	•	•
C=C+1	W	B76	•	•
C=C+1	WP	B16	•	•
C=C+1	X	B36	•	•
C=C+1	XS	B26	•	•
C=C+A	A	C2	•	•
C=C+A	B	A62	•	•
C=C+A	M	A52	•	•
C=C+A	P	A02	•	•
C=C+A	S	A42	•	•
C=C+A	W	A72	•	•
C=C+A	WP	A12	•	•
C=C+A	X	A32	•	•
C=C+A	XS	A22	•	•
C=C+C	A	C6	•	•
C=C+C	B	A66	•	•
C=C+C	M	A56	•	•
C=C+C	P	A06	•	•
C=C+C	S	A46	•	•
C=C+C	W	A76	•	•
C=C+C	WP	A16	•	•
C=C+C	X	A36	•	•

Mnémoniques		Codes hexa	R	B
C=C+C	XS	A26	•	•
C=C- *(n+1)*	A	818FA*n*	•	
C=C- *(n+1)*	B	8186A*n*	•	
C=C- *(n+1)*	M	8185A*n*	•	
C=C- *(n+1)*	P	8180A*n*	•	
C=C- *(n+1)*	S	8184A*n*	•	
C=C- *(n+1)*	W	8187A*n*	•	
C=C- *(n+1)*	WP	8181A*n*	•	
C=C- *(n+1)*	X	8183A*n*	•	
C=C- *(n+1)*	XS	8182A*n*	•	
C=C-1	A	CE	•	•
C=C-1	B	A6E	•	•
C=C-1	M	A5E	•	•
C=C-1	P	A0E	•	•
C=C-1	S	A4E	•	•
C=C-1	W	A7E	•	•
C=C-1	WP	A1E	•	•
C=C-1	X	A3E	•	•
C=C-1	XS	A2E	•	•
C=C-A	A	E2	•	•
C=C-A	B	B62	•	•
C=C-A	M	B52	•	•
C=C-A	P	B02	•	•
C=C-A	S	B42	•	•
C=C-A	W	B72	•	•
C=C-A	WP	B12	•	•
C=C-A	X	B32	•	•
C=C-A	XS	B22	•	•
C=C-B	A	E9	•	•
C=C-B	B	B69	•	•
C=C-B	M	B59	•	•
C=C-B	P	B09	•	•
C=C-B	S	B49	•	•
C=C-B	W	B79	•	•
C=C-B	WP	B19	•	•
C=C-B	X	B39	•	•
C=C-B	XS	B29	•	•
C=C-D	A	EB	•	•
C=C-D	B	B6B	•	•
C=C-D	M	B5B	•	•
C=C-D	P	B0B	•	•
C=C-D	S	B4B	•	•
C=C-D	W	B7B	•	•

Mnémoniques		Codes hexa	R	B
C=C-D	WP	B1B	•	•
C=C-D	X	B3B	•	•
C=C-D	XS	B2B	•	•
C=D	A	DB		
C=D	B	AEB		
C=D	M	ADB		
C=D	P	A8B		
C=D	S	ACB		
C=D	W	AFB		
C=D	WP	A9B		
C=D	X	ABB		
C=D	XS	AAB		
C=D!C	A	0EFF		
C=D!C	B	0E6F		
C=D!C	M	0E5F		
C=D!C	P	0E0F		
C=D!C	S	0E4F		
C=D!C	W	0E7F		
C=D!C	WP	0E1F		
C=D!C	X	0E3F		
C=D!C	XS	0E2F		
C=D&C	A	0EF7		
C=D&C	B	0E67		
C=D&C	M	0E57		
C=D&C	P	0E07		
C=D&C	S	0E47		
C=D&C	W	0E77		
C=D&C	WP	0E17		
C=D&C	X	0E37		
C=D&C	XS	0E27		
C=D+C	A	CB	•	•
C=D+C	B	A6B	•	•
C=D+C	M	A5B	•	•
C=D+C	P	A0B	•	•
C=D+C	S	A4B	•	•
C=D+C	W	A7B	•	•
C=D+C	WP	A1B	•	•
C=D+C	X	A3B	•	•
C=D+C	XS	A2B	•	•
C=DAT0	*(n+1)*	15E*n*		
C=DAT0	A	146		
C=DAT0	B	1566		
C=DAT0	B	14E		

Mnémoniques		Codes hexa	R	B
C=DAT0	M	1565		
C=DAT0	P	1560		
C=DAT0	S	1564		
C=DAT0	W	1567		
C=DAT0	WP	1561		
C=DAT0	X	1563		
C=DAT0	XS	1562		
C=DAT1	*(n+1)*	15F*n*		
C=DAT1	A	147		
C=DAT1	B	14F		
C=DAT1	B	1576		
C=DAT1	M	1575		
C=DAT1	P	1570		
C=DAT1	S	1574		
C=DAT1	W	1577		
C=DAT1	WP	1571		
C=DAT1	X	1573		
C=DAT1	XS	1572		
C=ID		806		
C=IN		803		
C=P	*n*	80C*n*		
C=PC		81B5		
C=R0		118		
C=R0.F	A	81AF18		
C=R0.F	B	81A618		
C=R0.F	M	81A518		
C=R0.F	P	81A018		
C=R0.F	S	81A418		
C=R0.F	W	81A718		
C=R0.F	WP	81A118		
C=R0.F	X	81A318		
C=R0.F	XS	81A218		
C=R1		119		
C=R1.F	A	81AF19		
C=R1.F	B	81A619		
C=R1.F	M	81A519		
C=R1.F	P	81A019		
C=R1.F	S	81A419		
C=R1.F	W	81A719		
C=R1.F	WP	81A119		
C=R1.F	X	81A319		
C=R1.F	XS	81A219		
C=R2		11A		

Mnémoniques		Codes hexa	R	B
C=R2.F	A	81AF1A		
C=R2.F	B	81A61A		
C=R2.F	M	81A51A		
C=R2.F	P	81A01A		
C=R2.F	S	81A41A		
C=R2.F	W	81A71A		
C=R2.F	WP	81A11A		
C=R2.F	X	81A31A		
C=R2.F	XS	81A21A		
C=R3		11B		
C=R3.F	A	81AF1B		
C=R3.F	B	81A61B		
C=R3.F	M	81A51B		
C=R3.F	P	81A01B		
C=R3.F	S	81A41B		
C=R3.F	W	81A71B		
C=R3.F	WP	81A11B		
C=R3.F	X	81A31B		
C=R3.F	XS	81A21B		
C=R4		11C		
C=R4.F	A	81AF1C		
C=R4.F	B	81A61C		
C=R4.F	M	81A51C		
C=R4.F	P	81A01C		
C=R4.F	S	81A41C		
C=R4.F	W	81A71C		
C=R4.F	WP	81A11C		
C=R4.F	X	81A31C		
C=R4.F	XS	81A21C		
C=RSTK		07		
C=ST		09		
CAEX	A	DE		
CAEX	B	AEE		
CAEX	M	ADE		
CAEX	P	A8E		
CAEX	S	ACE		
CAEX	W	AFE		
CAEX	WP	A9E		
CAEX	X	ABE		
CAEX	XS	AAE		
CBIT=0	*n*	8088*n*		
CBIT=1	*n*	8089*n*		
CD0EX		136		

Mnémoniques		Codes hexa	R	B
CD0XS		13E		
CD1EX		137		
CD1XS		13F		
CLRHST		82F		
CLRST		08		
CONFIG		805		
CPCEX		81B7		
CPEX	*n*	80F*n*		
CR0EX		128		
CR0EX.F	A	81AF28		
CR0EX.F	B	81A628		
CR0EX.F	M	81A528		
CR0EX.F	P	81A028		
CR0EX.F	S	81A428		
CR0EX.F	W	81A728		
CR0EX.F	WP	81A128		
CR0EX.F	X	81A328		
CR0EX.F	XS	81A228		
CR1EX		129		
CR1EX.F	A	81AF29		
CR1EX.F	B	81A629		
CR1EX.F	M	81A529		
CR1EX.F	P	81A029		
CR1EX.F	S	81A429		
CR1EX.F	W	81A729		
CR1EX.F	WP	81A129		
CR1EX.F	X	81A329		
CR1EX.F	XS	81A229		
CR2EX		12A		
CR2EX.F	A	81AF2A		
CR2EX.F	B	81A62A		
CR2EX.F	M	81A52A		
CR2EX.F	P	81A02A		
CR2EX.F	S	81A42A		
CR2EX.F	W	81A72A		
CR2EX.F	WP	81A12A		
CR2EX.F	X	81A32A		
CR2EX.F	XS	81A22A		
CR3EX		12B		
CR3EX.F	A	81AF2B		
CR3EX.F	B	81A62B		
CR3EX.F	M	81A52B		
CR3EX.F	P	81A02B		

Mnémoniques		Codes hexa	R	B
CR3EX.F	S	81A42B		
CR3EX.F	W	81A72B		
CR3EX.F	WP	81A12B		
CR3EX.F	X	81A32B		
CR3EX.F	XS	81A22B		
CR4EX		12C		
CR4EX.F	A	81AF2C		
CR4EX.F	B	81A62C		
CR4EX.F	M	81A52C		
CR4EX.F	P	81A02C		
CR4EX.F	S	81A42C		
CR4EX.F	W	81A72C		
CR4EX.F	WP	81A12C		
CR4EX.F	X	81A32C		
CR4EX.F	XS	81A22C		
CSL	A	F2		
CSL	B	BE2		
CSL	M	BD2		
CSL	P	B82		
CSL	S	BC2		
CSL	W	BF2		
CSL	WP	B92		
CSL	X	BB2		
CSL	XS	BA2		
CSLC		812		
CSR	A	F6		
CSR	B	BE6		
CSR	M	BD6		
CSR	P	B86		
CSR	S	BC6		
CSR	W	BF6		
CSR	WP	B96		
CSR	X	BB6		
CSR	XS	BA6		
CSRB		81E		
CSRB.F	A	819F2		
CSRB.F	B	81962		
CSRB.F	M	81952		
CSRB.F	P	81902		
CSRB.F	S	81942		
CSRB.F	W	81972		
CSRB.F	WP	81912		
CSRB.F	X	81932		

Mnémoniques		Codes hexa	R	B
CSRB.F	XS	81922		
CSRC		816		
CSTEX		0B		
D0=(2)	*pq*	19*qp*		
D0=(4)	*pqrs*	1A*srqp*		
D0=(5)	*pqrst*	1B*tsrqp*		
D0=A		130		
D0=AS		138		
D0=C		134		
D0=CS		13C		
D0=D0+	*(n+1)*	16*n*	•	
D0=D0-	*(n+1)*	18*n*	•	
D1=(2)	*pq*	1D*qp*		
D1=(4)	*pqrs*	1E*srqp*		
D1=(5)	*pqrst*	1F*tsrqp*		
D1=A		131		
D1=AS		139		
D1=C		135		
D1=CS		13D		
D1=D1+	*(n+1)*	17*n*	•	
D1=D1-	*(n+1)*	1C*n*	•	
D=-D	A	FB	•	•
D=-D	B	BEB	•	•
D=-D	M	BDB	•	•
D=-D	P	B8B	•	•
D=-D	S	BCB	•	•
D=-D	W	BFB	•	•
D=-D	WP	B9B	•	•
D=-D	X	BBB	•	•
D=-D	XS	BAB	•	•
D=-D-1	A	FF	•	•
D=-D-1	B	BEF	•	•
D=-D-1	M	BDF	•	•
D=-D-1	P	B8F	•	•
D=-D-1	S	BCF	•	•
D=-D-1	W	BFF	•	•
D=-D-1	WP	B9F	•	•
D=-D-1	X	BBF	•	•
D=-D-1	XS	BAF	•	•
D=0	A	D3		
D=0	B	AE3		
D=0	M	AD3		
D=0	P	A83		

Mnémoniques		Codes hexa	R	B
D=0	S	AC3		
D=0	W	AF3		
D=0	WP	A93		
D=0	X	AB3		
D=0	XS	AA3		
D=C	A	D7		
D=C	B	AE7		
D=C	M	AD7		
D=C	P	A87		
D=C	S	AC7		
D=C	W	AF7		
D=C	WP	A97		
D=C	X	AB7		
D=C	XS	AA7		
D=C-D	A	EF	•	•
D=C-D	B	B6F	•	•
D=C-D	M	B5F	•	•
D=C-D	P	B0F	•	•
D=C-D	S	B4F	•	•
D=C-D	W	B7F	•	•
D=C-D	WP	B1F	•	•
D=C-D	X	B3F	•	•
D=C-D	XS	B2F	•	•
D=D!C	A	0EFB		
D=D!C	B	0E6B		
D=D!C	M	0E5B		
D=D!C	P	0E0B		
D=D!C	S	0E4B		
D=D!C	W	0E7B		
D=D!C	WP	0E1B		
D=D!C	X	0E3B		
D=D!C	XS	0E2B		
D=D&C	A	0EF3		
D=D&C	B	0E63		
D=D&C	M	0E53		
D=D&C	P	0E03		
D=D&C	S	0E43		
D=D&C	W	0E73		
D=D&C	WP	0E13		
D=D&C	X	0E33		
D=D&C	XS	0E23		
D=D+ *(n+1)*	A	818F3*n*	•	
D=D+ *(n+1)*	B	81863*n*	•	

Mnémoniques		Codes hexa	R	B
D=D+ *(n+1)*	M	81853*n*	•	
D=D+ *(n+1)*	P	81803*n*	•	
D=D+ *(n+1)*	S	81843*n*	•	
D=D+ *(n+1)*	W	81873*n*	•	
D=D+ *(n+1)*	WP	81813*n*	•	
D=D+ *(n+1)*	X	81833*n*	•	
D=D+ *(n+1)*	XS	81823*n*	•	
D=D+1	A	E7	•	•
D=D+1	B	B67	•	•
D=D+1	M	B57	•	•
D=D+1	P	B07	•	•
D=D+1	S	B47	•	•
D=D+1	W	B77	•	•
D=D+1	WP	B17	•	•
D=D+1	X	B37	•	•
D=D+1	XS	B27	•	•
D=D+C	A	C3	•	•
D=D+C	B	A63	•	•
D=D+C	M	A53	•	•
D=D+C	P	A03	•	•
D=D+C	S	A43	•	•
D=D+C	W	A73	•	•
D=D+C	WP	A13	•	•
D=D+C	X	A33	•	•
D=D+C	XS	A23	•	•
D=D+D	A	C7	•	•
D=D+D	B	A67	•	•
D=D+D	M	A57	•	•
D=D+D	P	A07	•	•
D=D+D	S	A47	•	•
D=D+D	W	A77	•	•
D=D+D	WP	A17	•	•
D=D+D	X	A37	•	•
D=D+D	XS	A27	•	•
D=D- *(n+1)*	A	818FB*n*	•	
D=D- *(n+1)*	B	8186B*n*	•	
D=D- *(n+1)*	M	8185B*n*	•	
D=D- *(n+1)*	P	8180B*n*	•	
D=D- *(n+1)*	S	8184B*n*	•	
D=D- *(n+1)*	W	8187B*n*	•	
D=D- *(n+1)*	WP	8181B*n*	•	
D=D- *(n+1)*	X	8183B*n*	•	
D=D- *(n+1)*	XS	8182B*n*	•	

Mnémoniques		Codes hexa	R	B
D=D-1	A	CF	•	•
D=D-1	B	A6F	•	•
D=D-1	M	A5F	•	•
D=D-1	P	A0F	•	•
D=D-1	S	A4F	•	•
D=D-1	W	A7F	•	•
D=D-1	WP	A1F	•	•
D=D-1	X	A3F	•	•
D=D-1	XS	A2F	•	•
D=D-C	A	E3	•	•
D=D-C	B	B63	•	•
D=D-C	M	B53	•	•
D=D-C	P	B03	•	•
D=D-C	S	B43	•	•
D=D-C	W	B73	•	•
D=D-C	WP	B13	•	•
D=D-C	X	B33	•	•
D=D-C	XS	B23	•	•
DAT0=A	*(n+1)*	158*n*		
DAT0=A	A	140		
DAT0=A	B	148		
DAT0=A	B	1506		
DAT0=A	M	1505		
DAT0=A	P	1500		
DAT0=A	S	1504		
DAT0=A	W	1507		
DAT0=A	WP	1501		
DAT0=A	X	1503		
DAT0=A	XS	1502		
DAT0=C	*(n+1)*	15C*n*		
DAT0=C	A	144		
DAT0=C	B	1546		
DAT0=C	B	14C		
DAT0=C	M	1545		
DAT0=C	P	1540		
DAT0=C	S	1544		
DAT0=C	W	1547		
DAT0=C	WP	1541		
DAT0=C	X	1543		
DAT0=C	XS	1542		
DAT1=A	*(n+1)*	159*n*		
DAT1=A	A	141		
DAT1=A	B	149		

Mnémoniques		Codes hexa	R	B
DAT1=A	B	1516		
DAT1=A	M	1515		
DAT1=A	P	1510		
DAT1=A	S	1514		
DAT1=A	W	1517		
DAT1=A	WP	1511		
DAT1=A	X	1513		
DAT1=A	XS	1512		
DAT1=C	*(n+1)*	15D*n*		
DAT1=C	A	145		
DAT1=C	B	1556		
DAT1=C	B	14D		
DAT1=C	M	1555		
DAT1=C	P	1550		
DAT1=C	S	1554		
DAT1=C	W	1557		
DAT1=C	WP	1551		
DAT1=C	X	1553		
DAT1=C	XS	1552		
DCEX	A	DF		
DCEX	B	AEF		
DCEX	M	ADF		
DCEX	P	A8F		
DCEX	S	ACF		
DCEX	W	AFF		
DCEX	WP	A9F		
DCEX	X	ABF		
DCEX	XS	AAF		
DSL	A	F3		
DSL	B	BE3		
DSL	M	BD3		
DSL	P	B83		
DSL	S	BC3		
DSL	W	BF3		
DSL	WP	B93		
DSL	X	BB3		
DSL	XS	BA3		
DSLC		813		
DSR	A	F7		
DSR	B	BE7		
DSR	M	BD7		
DSR	P	B87		
DSR	S	BC7		

Mnémoniques		Codes hexa	R	B
DSR	W	BF7		
DSR	WP	B97		
DSR	X	BB7		
DSR	XS	BA7		
DSRB		81F		
DSRB.F	A	819F3		
DSRB.F	B	81963		
DSRB.F	M	81953		
DSRB.F	P	81903		
DSRB.F	S	81943		
DSRB.F	W	81973		
DSRB.F	WP	81913		
DSRB.F	X	81933		
DSRB.F	XS	81923		
DSRC		817		
GOC	*label*	4*qp*		
GOLONG	*label*	8C*srqp*		
GONC	*label*	5*qp*		
GOSBVL	*label*	8F*tsrqp*		
GOSUB	*label*	7*rqp*		
GOSUBL	*label*	8E*srqp*		
GOTO	*label*	6*rqp*		
GOVLNG	*label*	8D*tsrqp*		
HS=0	*n*	82*n*		
INTOFF		808F		
INTON		8080		
LA *(n+1)*	hn..h0	8082*nh0..hn*		
LC *(n+1)*	hn..h0	3*nh0..hn*		
MP=0		828		
NOP3		420		
NOP4		6300		
NOP5		64000		
OUT=C		801		
OUT=CS		800		
P=	*n*	2*n*		
P=C	*n*	80D*n*		
P=P+1		0C	•	
P=P-1		0D	•	
PC=(A)		808C		
PC=(C)		808E		
PC=A		81B2		
PC=C		81B3		
R0=A		100		

Mnémoniques		Codes hexa	R	B
R0=A.F	A	81AF00		
R0=A.F	B	81A600		
R0=A.F	M	81A500		
R0=A.F	P	81A000		
R0=A.F	S	81A400		
R0=A.F	W	81A700		
R0=A.F	WP	81A100		
R0=A.F	X	81A300		
R0=A.F	XS	81A200		
R0=C		108		
R0=C.F	A	81AF08		
R0=C.F	B	81A608		
R0=C.F	M	81A508		
R0=C.F	P	81A008		
R0=C.F	S	81A408		
R0=C.F	W	81A708		
R0=C.F	WP	81A108		
R0=C.F	X	81A308		
R0=C.F	XS	81A208		
R1=A		101		
R1=A.F	A	81AF01		
R1=A.F	B	81A601		
R1=A.F	M	81A501		
R1=A.F	P	81A001		
R1=A.F	S	81A401		
R1=A.F	W	81A701		
R1=A.F	WP	81A101		
R1=A.F	X	81A301		
R1=A.F	XS	81A201		
R1=C		109		
R1=C.F	A	81AF09		
R1=C.F	B	81A609		
R1=C.F	M	81A509		
R1=C.F	P	81A009		
R1=C.F	S	81A409		
R1=C.F	W	81A709		
R1=C.F	WP	81A109		
R1=C.F	X	81A309		
R1=C.F	XS	81A209		
R2=A		102		
R2=A.F	A	81AF02		
R2=A.F	B	81A602		
R2=A.F	M	81A502		

Mnémoniques		Codes hexa	R	B
R2=A.F	P	81A002		
R2=A.F	S	81A402		
R2=A.F	W	81A702		
R2=A.F	WP	81A102		
R2=A.F	X	81A302		
R2=A.F	XS	81A202		
R2=C		10A		
R2=C.F	A	81AF0A		
R2=C.F	B	81A60A		
R2=C.F	M	81A50A		
R2=C.F	P	81A00A		
R2=C.F	S	81A40A		
R2=C.F	W	81A70A		
R2=C.F	WP	81A10A		
R2=C.F	X	81A30A		
R2=C.F	XS	81A20A		
R3=A		103		
R3=A.F	A	81AF03		
R3=A.F	B	81A603		
R3=A.F	M	81A503		
R3=A.F	P	81A003		
R3=A.F	S	81A403		
R3=A.F	W	81A703		
R3=A.F	WP	81A103		
R3=A.F	X	81A303		
R3=A.F	XS	81A203		
R3=C		10B		
R3=C.F	A	81AF0B		
R3=C.F	B	81A60B		
R3=C.F	M	81A50B		
R3=C.F	P	81A00B		
R3=C.F	S	81A40B		
R3=C.F	W	81A70B		
R3=C.F	WP	81A10B		
R3=C.F	X	81A30B		
R3=C.F	XS	81A20B		
R4=A		104		
R4=A.F	A	81AF04		
R4=A.F	B	81A604		
R4=A.F	M	81A504		
R4=A.F	P	81A004		
R4=A.F	S	81A404		
R4=A.F	W	81A704		

Mnémoniques		Codes hexa	R	B
R4=A.F	WP	81A104		
R4=A.F	X	81A304		
R4=A.F	XS	81A204		
R4=C		10C		
R4=C.F	A	81AF0C		
R4=C.F	B	81A60C		
R4=C.F	M	81A50C		
R4=C.F	P	81A00C		
R4=C.F	S	81A40C		
R4=C.F	W	81A70C		
R4=C.F	WP	81A10C		
R4=C.F	X	81A30C		
R4=C.F	XS	81A20C		
RESET		80A		
RSI		80810		
RSTK=C		06		
RTI		0F		
RTN		01		
RTNC		400		
RTNCC		03	•	
RTNNC		500		
RTNSC		02	•	
RTNSXM		00		
SB=0		822		
SETDEC		05		
SETHEX		04		
SHUTDN		807		
SR=0		824		
SREQ?		80E		
ST=0	*n*	84*n*		
ST=1	*n*	85*n*		
ST=C		0A		
UNCNFG		804		
XM=0		821		

20

Les instructions classées par codes

La liste d'instructions du chapitre précédent est utilisée lors de la création d'un programme pour transformer un programme présenté sous la forme d'une suite de mnémoniques en une chaîne de codes hexadécimaux à fournir en argument au programme ASS qui en fera un programme exécutable (objet « Code »). Au contraire, la liste présentée dans ce chapitre permet une recherche en partant d'un code hexadécimal pour obtenir le mnémonique correspondant. Cette liste permet donc l'analyse d'une chaîne de codes hexadécimaux.

Vous trouverez d'abord dans la liste les codes constitués uniquement de chiffres décimaux (0 à 9) puis les codes constitués de chiffres hexadécimaux (0 à 9 et A à F). Attention ! Les codes sont classés par ordre alphabétique et non pas par ordre des nombres hexadécimaux croissants.

Lorsqu'une instruction permet de sélectionner un champ, seul un quartet du code hexadécimal de l'instruction varie en fonction dudit champ. Généralement, le quartet variable vaut #Fh pour le registre A, pour les autres champs, on utilise selon le cas, l'une de ces deux séries :

Champ	Série A	Série B
P	#0h	#8h
WP	#1h	#9h
XS	#2h	#Ah
X	#3h	#Bh
S	#4h	#Ch
M	#5h	#Dh
B	#6h	#Eh
W	#7h	#Fh

Lisez impérativement les recommandations au début du chapitre précédent ! Avez-vous lu les chapitres 13 à 17 ? Non, alors c'est le moment !

Codes hexa	Mnémoniques		R	B
00	RTNSXM			
01	RTN			
02	RTNSC		•	
03	RTNCC		•	
04	SETHEX			
05	SETDEC			
06	RSTK=C			
07	C=RSTK			
08	CLRST			
09	C=ST			
100	R0=A			
101	R1=A			
102	R2=A			
103	R3=A			
104	R4=A			
108	R0=C			
109	R1=C			
110	A=R0			
111	A=R1			
112	A=R2			
113	A=R3			
114	A=R4			
118	C=R0			
119	C=R1			
120	AR0EX			
121	AR1EX			
122	AR2EX			
123	AR3EX			
124	AR4EX			
128	CR0EX			
129	CR1EX			
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			
140	DAT0=A	A		
141	DAT1=A	A		

Codes hexa	Mnémoniques		R	B
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		
400	RTNC			
420	NOP3			
500	RTNNC			
800	OUT=CS			
801	OUT=C			
802	A=IN			
803	C=IN			
804	UNCNFG			
805	CONFIG			
806	C=ID			
807	SHUTDN			
809	C+P+1		•	
810	ASLC			
811	BSLC			
812	CSLC			
813	DSLC			
814	ASRC			
815	BSRC			
816	CSRC			
817	DSRC			
821	XM=0			
822	SB=0			
824	SR=0			
828	MP=0			
1500	DAT0=A	P		
1501	DAT0=A	WP		
1502	DAT0=A	XS		
1503	DAT0=A	X		
1504	DAT0=A	S		
1505	DAT0=A	M		
1506	DAT0=A	B		
1507	DAT0=A	W		
1510	DAT1=A	P		
1511	DAT1=A	WP		
1512	DAT1=A	XS		

Codes hexa	Mnémoniques		R	B
1513	DAT1=A	X		
1514	DAT1=A	S		
1515	DAT1=A	M		
1516	DAT1=A	B		
1517	DAT1=A	W		
1520	A=DAT0	P		
1521	A=DAT0	WP		
1522	A=DAT0	XS		
1523	A=DAT0	X		
1524	A=DAT0	S		
1525	A=DAT0	M		
1526	A=DAT0	B		
1527	A=DAT0	W		
1530	A=DAT1	P		
1531	A=DAT1	WP		
1532	A=DAT1	XS		
1533	A=DAT1	X		
1534	A=DAT1	S		
1535	A=DAT1	M		
1536	A=DAT1	B		
1537	A=DAT1	W		
1540	DAT0=C	P		
1541	DAT0=C	WP		
1542	DAT0=C	XS		
1543	DAT0=C	X		
1544	DAT0=C	S		
1545	DAT0=C	M		
1546	DAT0=C	B		
1547	DAT0=C	W		
1550	DAT1=C	P		
1551	DAT1=C	WP		
1552	DAT1=C	XS		
1553	DAT1=C	X		
1554	DAT1=C	S		
1555	DAT1=C	M		
1556	DAT1=C	B		
1557	DAT1=C	W		
1560	C=DAT0	P		
1561	C=DAT0	WP		
1562	C=DAT0	XS		
1563	C=DAT0	X		
1564	C=DAT0	S		
1565	C=DAT0	M		

Codes hexa	Mnémoniques		R	B
1566	C=DAT0	B		
1567	C=DAT0	W		
1570	C=DAT1	P		
1571	C=DAT1	WP		
1572	C=DAT1	XS		
1573	C=DAT1	X		
1574	C=DAT1	S		
1575	C=DAT1	M		
1576	C=DAT1	B		
1577	C=DAT1	W		
6300	NOP4			
8080	INTON			
8083	BUSCB			
64000	NOP5			
80810	RSI			
81900	ASRB.F	P		
81901	BSRB.F	P		
81902	CSRB.F	P		
81903	DSRB.F	P		
81910	ASRB.F	WP		
81911	BSRB.F	WP		
81912	CSRB.F	WP		
81913	DSRB.F	WP		
81920	ASRB.F	XS		
81921	BSRB.F	XS		
81922	CSRB.F	XS		
81923	DSRB.F	XS		
81930	ASRB.F	X		
81931	BSRB.F	X		
81932	CSRB.F	X		
81933	DSRB.F	X		
81940	ASRB.F	S		
81941	BSRB.F	S		
81942	CSRB.F	S		
81943	DSRB.F	S		
81950	ASRB.F	M		
81951	BSRB.F	M		
81952	CSRB.F	M		
81953	DSRB.F	M		
81960	ASRB.F	B		
81961	BSRB.F	B		
81962	CSRB.F	B		
81963	DSRB.F	B		

Codes hexa	Mnémoniques		R	B
81970	ASRB.F	W		
81971	BSRB.F	W		
81972	CSRB.F	W		
81973	DSRB.F	W		
0A	ST=C			
0B	CSTEX			
0C	P=P+1		•	
0D	P=P-1		•	
0E00	A=A&B	P		
0E01	B=B&C	P		
0E02	C=C&A	P		
0E03	D=D&C	P		
0E04	B=A&B	P		
0E05	C=B&C	P		
0E06	A=C&A	P		
0E07	C=D&C	P		
0E08	A=A!B	P		
0E09	B=B!C	P		
0E0A	C=C!A	P		
0E0B	D=D!C	P		
0E0C	B=A!B	P		
0E0D	C=B!C	P		
0E0E	A=C!A	P		
0E0F	C=D!C	P		
0E10	A=A&B	WP		
0E11	B=B&C	WP		
0E12	C=C&A	WP		
0E13	D=D&C	WP		
0E14	B=A&B	WP		
0E15	C=B&C	WP		
0E16	A=C&A	WP		
0E17	C=D&C	WP		
0E18	A=A!B	WP		
0E19	B=B!C	WP		
0E1A	C=C!A	WP		
0E1B	D=D!C	WP		
0E1C	B=A!B	WP		
0E1D	C=B!C	WP		
0E1E	A=C!A	WP		
0E1F	C=D!C	WP		
0E20	A=A&B	XS		
0E21	B=B&C	XS		
0E22	C=C&A	XS		

Codes hexa	Mnémoniques		R	B
0E23	D=D&C	XS		
0E24	B=A&B	XS		
0E25	C=B&C	XS		
0E26	A=C&A	XS		
0E27	C=D&C	XS		
0E28	A=A!B	XS		
0E29	B=B!C	XS		
0E2A	C=C!A	XS		
0E2B	D=D!C	XS		
0E2C	B=A!B	XS		
0E2D	C=B!C	XS		
0E2E	A=C!A	XS		
0E2F	C=D!C	XS		
0E30	A=A&B	X		
0E31	B=B&C	X		
0E32	C=C&A	X		
0E33	D=D&C	X		
0E34	B=A&B	X		
0E35	C=B&C	X		
0E36	A=C&A	X		
0E37	C=D&C	X		
0E38	A=A!B	X		
0E39	B=B!C	X		
0E3A	C=C!A	X		
0E3B	D=D!C	X		
0E3C	B=A!B	X		
0E3D	C=B!C	X		
0E3E	A=C!A	X		
0E3F	C=D!C	X		
0E40	A=A&B	S		
0E41	B=B&C	S		
0E42	C=C&A	S		
0E43	D=D&C	S		
0E44	B=A&B	S		
0E45	C=B&C	S		
0E46	A=C&A	S		
0E47	C=D&C	S		
0E48	A=A!B	S		
0E49	B=B!C	S		
0E4A	C=C!A	S		
0E4B	D=D!C	S		
0E4C	B=A!B	S		
0E4D	C=B!C	S		

Codes hexa	Mnémoniques		R	B
0E4E	A=C!A	S		
0E4F	C=D!C	S		
0E50	A=A&B	M		
0E51	B=B&C	M		
0E52	C=C&A	M		
0E53	D=D&C	M		
0E54	B=A&B	M		
0E55	C=B&C	M		
0E56	A=C&A	M		
0E57	C=D&C	M		
0E58	A=A!B	M		
0E59	B=B!C	M		
0E5A	C=C!A	M		
0E5B	D=D!C	M		
0E5C	B=A!B	M		
0E5D	C=B!C	M		
0E5E	A=C!A	M		
0E5F	C=D!C	M		
0E60	A=A&B	B		
0E61	B=B&C	B		
0E62	C=C&A	B		
0E63	D=D&C	B		
0E64	B=A&B	B		
0E65	C=B&C	B		
0E66	A=C&A	B		
0E67	C=D&C	B		
0E68	A=A!B	B		
0E69	B=B!C	B		
0E6A	C=C!A	B		
0E6B	D=D!C	B		
0E6C	B=A!B	B		
0E6D	C=B!C	B		
0E6E	A=C!A	B		
0E6F	C=D!C	B		
0E70	A=A&B	W		
0E71	B=B&C	W		
0E72	C=C&A	W		
0E73	D=D&C	W		
0E74	B=A&B	W		
0E75	C=B&C	W		
0E76	A=C&A	W		
0E77	C=D&C	W		
0E78	A=A!B	W		

Codes hexa	Mnémoniques		R	B
0E79	B=B!C	W		
0E7A	C=C!A	W		
0E7B	D=D!C	W		
0E7C	B=A!B	W		
0E7D	C=B!C	W		
0E7E	A=C!A	W		
0E7F	C=D!C	W		
0EF0	A=A&B	A		
0EF1	B=B&C	A		
0EF2	C=C&A	A		
0EF3	D=D&C	A		
0EF4	B=A&B	A		
0EF5	C=B&C	A		
0EF6	A=C&A	A		
0EF7	C=D&C	A		
0EF8	A=A!B	A		
0EF9	B=B!C	A		
0EFA	C=C!A	A		
0EFB	D=D!C	A		
0EFC	B=A!B	A		
0EFD	C=B!C	A		
0EFE	A=C!A	A		
0EFF	C=D!C	A		
0F	RTI			
10A	R2=C			
10B	R3=C			
10C	R4=C			
11A	C=R2			
11B	C=R3			
11C	C=R4			
12A	CR2EX			
12B	CR3EX			
12C	CR4EX			
13A	AD0XS			
13B	AD1XS			
13C	D0=CS			
13D	D1=CS			
13E	CD0XS			
13F	CD1XS			
14A	A=DAT0	B		
14B	A=DAT1	B		
14C	DAT0=C	B		
14D	DAT1=C	B		

Codes hexa	Mnémoniques		R	B
14E	C=DAT0	B		
14F	C=DAT1	B		
158*n*	DAT0=A	*(n+1)*		
159*n*	DAT1=A	*(n+1)*		
15A*n*	A=DAT0	*(n+1)*		
15C*n*	DAT0=C	*(n+1)*		
15D*n*	DAT1=C	*(n+1)*		
15E*n*	C=DAT0	*(n+1)*		
15F*n*	C=DAT1	*(n+1)*		
16*n*	D0=D0+	*(n+1)*	•	
17*n*	D1=D1+	*(n+1)*	•	
18*n*	D0=D0-	*(n+1)*	•	
19*qp*	D0=(2)	*pq*		
1A*srqp*	D0=(4)	*pqrs*		
1Bt*srqp*	D0=(5)	*pqrst*		
1C*n*	D1=D1-	*(n+1)*	•	
1D*qp*	D1=(2)	*pq*		
1E*srqp*	D1=(4)	*pqrs*		
1Ft*srqp*	D1=(5)	*pqrst*		
2*n*	P=	*n*		
3*nh0..hn*	LC*(n+1)*	*hn..h0*		
4*qp*	GOC	*label*		
5*qp*	GONC	*label*		
6*rqp*	GOTO	*label*		
7*rqp*	GOSUB	*label*		
8082*nh0..hn*	LA*(n+1)*	*hn..h0*		
8084*n*	ABIT=0	*n*		
8085*n*	ABIT=1	*n*		
8086*nqp*	?ABIT=0	*n*	•	
8087*nqp*	?ABIT=1	*n*	•	
8088*n*	CBIT=0	*n*		
8089*n*	CBIT=1	*n*		
808A*nqp*	?CBIT=0	*n*	•	
808B*nqp*	?CBIT=1	*n*	•	
808C	PC=(A)			
808D	BUSCD			
808E	PC=(C)			
808F	INTOFF			
80A	RESET			
80B	BUSCC			
80C*n*	C=P	*n*		
80D*n*	P=C	*n*		
80E	SREQ?			

Codes hexa	Mnémoniques		R	B
80F*n*	CPEX	*n*		
81800*n*	A=A+ *(n+1)*	P		
81801*n*	B=B+ *(n+1)*	P	•	
81802*n*	C=C+ *(n+1)*	P	•	
81803*n*	D=D+ *(n+1)*	P	•	
81808*n*	A=A- *(n+1)*	P	•	
81809*n*	B=B- *(n+1)*	P	•	
8180A*n*	C=C- *(n+1)*	P	•	
8180B*n*	D=D- *(n+1)*	P	•	
81810*n*	A=A+ *(n+1)*	WP		
81811*n*	B=B+ *(n+1)*	WP	•	
81812*n*	C=C+ *(n+1)*	WP	•	
81813*n*	D=D+ *(n+1)*	WP	•	
81818*n*	A=A- *(n+1)*	WP	•	
81819*n*	B=B- *(n+1)*	WP	•	
8181A*n*	C=C- *(n+1)*	WP	•	
8181B*n*	D=D- *(n+1)*	WP	•	
81820*n*	A=A+ *(n+1)*	XS		
81821*n*	B=B+ *(n+1)*	XS	•	
81822*n*	C=C+ *(n+1)*	XS	•	
81823*n*	D=D+ *(n+1)*	XS	•	
81828*n*	A=A- *(n+1)*	XS	•	
81829*n*	B=B- *(n+1)*	XS	•	
8182A*n*	C=C- *(n+1)*	XS	•	
8182B*n*	D=D- *(n+1)*	XS	•	
81830*n*	A=A+ *(n+1)*	X		
81831*n*	B=B+ *(n+1)*	X	•	
81832*n*	C=C+ *(n+1)*	X	•	
81833*n*	D=D+ *(n+1)*	X	•	
81838*n*	A=A- *(n+1)*	X	•	
81839*n*	B=B- *(n+1)*	X	•	
8183A*n*	C=C- *(n+1)*	X	•	
8183B*n*	D=D- *(n+1)*	X	•	
81840*n*	A=A+ *(n+1)*	S		
81841*n*	B=B+ *(n+1)*	S	•	
81842*n*	C=C+ *(n+1)*	S	•	
81843*n*	D=D+ *(n+1)*	S	•	
81848*n*	A=A- *(n+1)*	S	•	
81849*n*	B=B- *(n+1)*	S	•	
8184A*n*	C=C- *(n+1)*	S	•	
8184B*n*	D=D- *(n+1)*	S	•	
81850*n*	A=A+ *(n+1)*	M		
81851*n*	B=B+ *(n+1)*	M	•	

Codes hexa	Mnémoniques		R	B
81852*n*	C=C+ *(n+1)*	M	•	
81853*n*	D=D+ *(n+1)*	M	•	
81858*n*	A=A- *(n+1)*	M	•	
81859*n*	B=B- *(n+1)*	M	•	
8185A*n*	C=C- *(n+1)*	M	•	
8185B*n*	D=D- *(n+1)*	M	•	
81860*n*	A=A+ *(n+1)*	B		
81861*n*	B=B+ *(n+1)*	B	•	
81862*n*	C=C+ *(n+1)*	B	•	
81863*n*	D=D+ *(n+1)*	B	•	
81868*n*	A=A- *(n+1)*	B	•	
81869*n*	B=B- *(n+1)*	B	•	
8186A*n*	C=C- *(n+1)*	B	•	
8186B*n*	D=D- *(n+1)*	B	•	
81870*n*	A=A+ *(n+1)*	W		
81871*n*	B=B+ *(n+1)*	W	•	
81872*n*	C=C+ *(n+1)*	W	•	
81873*n*	D=D+ *(n+1)*	W	•	
81878*n*	A=A- *(n+1)*	W	•	
81879*n*	B=B- *(n+1)*	W	•	
8187A*n*	C=C- *(n+1)*	W	•	
8187B*n*	D=D- *(n+1)*	W	•	
818F0*n*	A=A+ *(n+1)*	A		
818F1*n*	B=B+ *(n+1)*	A	•	
818F2*n*	C=C+ *(n+1)*	A	•	
818F3*n*	D=D+ *(n+1)*	A	•	
818F8*n*	A=A- *(n+1)*	A	•	
818F9*n*	B=B- *(n+1)*	A	•	
818FA*n*	C=C- *(n+1)*	A	•	
818FB*n*	D=D- *(n+1)*	A	•	
819F0	ASRB.F	A		
819F1	BSRB.F	A		
819F2	CSRB.F	A		
819F3	DSRB.F	A		
81A000	R0=A.F	P		
81A001	R1=A.F	P		
81A002	R2=A.F	P		
81A003	R3=A.F	P		
81A004	R4=A.F	P		
81A008	R0=C.F	P		
81A009	R1=C.F	P		
81A00A	R2=C.F	P		
81A00B	R3=C.F	P		

Codes hexa	Mnémoniques		R	B
81A00C	R4=C.F	P		
81A010	A=R0.F	P		
81A011	A=R1.F	P		
81A012	A=R2.F	P		
81A013	A=R3.F	P		
81A014	A=R4	P		
81A018	C=R0.F	P		
81A019	C=R1.F	P		
81A01A	C=R2.F	P		
81A01B	C=R3.F	P		
81A01C	C=R4.F	P		
81A020	AR0EX.F	P		
81A021	AR1EX.F	P		
81A022	AR2EX.F	P		
81A023	AR3EX.F	P		
81A024	AR4EX.F	P		
81A028	CR0EX.F	P		
81A029	CR1EX.F	P		
81A02A	CR2EX.F	P		
81A02B	CR3EX.F	P		
81A02C	CR4EX.F	P		
81A100	R0=A.F	WP		
81A101	R1=A.F	WP		
81A102	R2=A.F	WP		
81A103	R3=A.F	WP		
81A104	R4=A.F	WP		
81A108	R0=C.F	WP		
81A109	R1=C.F	WP		
81A10A	R2=C.F	WP		
81A10B	R3=C.F	WP		
81A10C	R4=C.F	WP		
81A110	A=R0.F	WP		
81A111	A=R1.F	WP		
81A112	A=R2.F	WP		
81A113	A=R3.F	WP		
81A114	A=R4	WP		
81A118	C=R0.F	WP		
81A119	C=R1.F	WP		
81A11A	C=R2.F	WP		
81A11B	C=R3.F	WP		
81A11C	C=R4.F	WP		
81A120	AR0EX.F	WP		
81A121	AR1EX.F	WP		

Codes hexa	Mnémoniques		R	B
81A122	AR2EX.F	WP		
81A123	AR3EX.F	WP		
81A124	AR4EX.F	WP		
81A128	CR0EX.F	WP		
81A129	CR1EX.F	WP		
81A12A	CR2EX.F	WP		
81A12B	CR3EX.F	WP		
81A12C	CR4EX.F	WP		
81A200	R0=A.F	XS		
81A201	R1=A.F	XS		
81A202	R2=A.F	XS		
81A203	R3=A.F	XS		
81A204	R4=A.F	XS		
81A208	R0=C.F	XS		
81A209	R1=C.F	XS		
81A20A	R2=C.F	XS		
81A20B	R3=C.F	XS		
81A20C	R4=C.F	XS		
81A210	A=R0.F	XS		
81A211	A=R1.F	XS		
81A212	A=R2.F	XS		
81A213	A=R3.F	XS		
81A214	A=R4	XS		
81A218	C=R0.F	XS		
81A219	C=R1.F	XS		
81A21A	C=R2.F	XS		
81A21B	C=R3.F	XS		
81A21C	C=R4.F	XS		
81A220	AR0EX.F	XS		
81A221	AR1EX.F	XS		
81A222	AR2EX.F	XS		
81A223	AR3EX.F	XS		
81A224	AR4EX.F	XS		
81A228	CR0EX.F	XS		
81A229	CR1EX.F	XS		
81A22A	CR2EX.F	XS		
81A22B	CR3EX.F	XS		
81A22C	CR4EX.F	XS		
81A300	R0=A.F	X		
81A301	R1=A.F	X		
81A302	R2=A.F	X		
81A303	R3=A.F	X		
81A304	R4=A.F	X		

Codes hexa	Mnémoniques		R	B
81A308	R0=C.F	X		
81A309	R1=C.F	X		
81A30A	R2=C.F	X		
81A30B	R3=C.F	X		
81A30C	R4=C.F	X		
81A310	A=R0.F	X		
81A311	A=R1.F	X		
81A312	A=R2.F	X		
81A313	A=R3.F	X		
81A314	A=R4	X		
81A318	C=R0.F	X		
81A319	C=R1.F	X		
81A31A	C=R2.F	X		
81A31B	C=R3.F	X		
81A31C	C=R4.F	X		
81A320	AR0EX.F	X		
81A321	AR1EX.F	X		
81A322	AR2EX.F	X		
81A323	AR3EX.F	X		
81A324	AR4EX.F	X		
81A328	CR0EX.F	X		
81A329	CR1EX.F	X		
81A32A	CR2EX.F	X		
81A32B	CR3EX.F	X		
81A32C	CR4EX.F	X		
81A400	R0=A.F	S		
81A401	R1=A.F	S		
81A402	R2=A.F	S		
81A403	R3=A.F	S		
81A404	R4=A.F	S		
81A408	R0=C.F	S		
81A409	R1=C.F	S		
81A40A	R2=C.F	S		
81A40B	R3=C.F	S		
81A40C	R4=C.F	S		
81A410	A=R0.F	S		
81A411	A=R1.F	S		
81A412	A=R2.F	S		
81A413	A=R3.F	S		
81A414	A=R4	S		
81A418	C=R0.F	S		
81A419	C=R1.F	S		
81A41A	C=R2.F	S		

Codes hexa	Mnémoniques		R	B
81A41B	C=R3.F	S		
81A41C	C=R4.F	S		
81A420	AR0EX.F	S		
81A421	AR1EX.F	S		
81A422	AR2EX.F	S		
81A423	AR3EX.F	S		
81A424	AR4EX.F	S		
81A428	CR0EX.F	S		
81A429	CR1EX.F	S		
81A42A	CR2EX.F	S		
81A42B	CR3EX.F	S		
81A42C	CR4EX.F	S		
81A500	R0=A.F	M		
81A501	R1=A.F	M		
81A502	R2=A.F	M		
81A503	R3=A.F	M		
81A504	R4=A.F	M		
81A508	R0=C.F	M		
81A509	R1=C.F	M		
81A50A	R2=C.F	M		
81A50B	R3=C.F	M		
81A50C	R4=C.F	M		
81A510	A=R0.F	M		
81A511	A=R1.F	M		
81A512	A=R2.F	M		
81A513	A=R3.F	M		
81A514	A=R4	M		
81A518	C=R0.F	M		
81A519	C=R1.F	M		
81A51A	C=R2.F	M		
81A51B	C=R3.F	M		
81A51C	C=R4.F	M		
81A520	AR0EX.F	M		
81A521	AR1EX.F	M		
81A522	AR2EX.F	M		
81A523	AR3EX.F	M		
81A524	AR4EX.F	M		
81A528	CR0EX.F	M		
81A529	CR1EX.F	M		
81A52A	CR2EX.F	M		
81A52B	CR3EX.F	M		
81A52C	CR4EX.F	M		
81A600	R0=A.F	B		

Codes hexa	Mnémoniques		R	B
81A601	R1=A.F	B		
81A602	R2=A.F	B		
81A603	R3=A.F	B		
81A604	R4=A.F	B		
81A608	R0=C.F	B		
81A609	R1=C.F	B		
81A60A	R2=C.F	B		
81A60B	R3=C.F	B		
81A60C	R4=C.F	B		
81A610	A=R0.F	B		
81A611	A=R1.F	B		
81A612	A=R2.F	B		
81A613	A=R3.F	B		
81A614	A=R4	B		
81A618	C=R0.F	B		
81A619	C=R1.F	B		
81A61A	C=R2.F	B		
81A61B	C=R3.F	B		
81A61C	C=R4.F	B		
81A620	AR0EX.F	B		
81A621	AR1EX.F	B		
81A622	AR2EX.F	B		
81A623	AR3EX.F	B		
81A624	AR4EX.F	B		
81A628	CR0EX.F	B		
81A629	CR1EX.F	B		
81A62A	CR2EX.F	B		
81A62B	CR3EX.F	B		
81A62C	CR4EX.F	B		
81A700	R0=A.F	W		
81A701	R1=A.F	W		
81A702	R2=A.F	W		
81A703	R3=A.F	W		
81A704	R4=A.F	W		
81A708	R0=C.F	W		
81A709	R1=C.F	W		
81A70A	R2=C.F	W		
81A70B	R3=C.F	W		
81A70C	R4=C.F	W		
81A710	A=R0.F	W		
81A711	A=R1.F	W		
81A712	A=R2.F	W		
81A713	A=R3.F	W		

Codes hexa	Mnémoniques		R	B
81A714	A=R4	W		
81A718	C=R0.F	W		
81A719	C=R1.F	W		
81A71A	C=R2.F	W		
81A71B	C=R3.F	W		
81A71C	C=R4.F	W		
81A720	AR0EX.F	W		
81A721	AR1EX.F	W		
81A722	AR2EX.F	W		
81A723	AR3EX.F	W		
81A724	AR4EX.F	W		
81A728	CR0EX.F	W		
81A729	CR1EX.F	W		
81A72A	CR2EX.F	W		
81A72B	CR3EX.F	W		
81A72C	CR4EX.F	W		
81AF00	R0=A.F	A		
81AF01	R1=A.F	A		
81AF02	R2=A.F	A		
81AF03	R3=A.F	A		
81AF04	R4=A.F	A		
81AF08	R0=C.F	A		
81AF09	R1=C.F	A		
81AF0A	R2=C.F	A		
81AF0B	R3=C.F	A		
81AF0C	R4=C.F	A		
81AF10	A=R0.F	A		
81AF11	A=R1.F	A		
81AF12	A=R2.F	A		
81AF13	A=R3.F	A		
81AF14	A=R4	A		
81AF18	C=R0.F	A		
81AF19	C=R1.F	A		
81AF1A	C=R2.F	A		
81AF1B	C=R3.F	A		
81AF1C	C=R4.F	A		
81AF20	AR0EX.F	A		
81AF21	AR1EX.F	A		
81AF22	AR2EX.F	A		
81AF23	AR3EX.F	A		
81AF24	AR4EX.F	A		
81AF28	CR0EX.F	A		
81AF29	CR1EX.F	A		

Codes hexa	Mnémoniques		R	B
81AF2A	CR2EX.F	A		
81AF2B	CR3EX.F	A		
81AF2C	CR4EX.F	A		
81B2	PC=A			
81B3	PC=C			
81B4	A=PC			
81B5	C=PC			
81B6	APCEX			
81B7	CPCEX			
81C	ASRB			
81D	BSRB			
81E	CSRB			
81F	DSRB			
82F	CLRHST			
82*n*	HS=0	*n*		
831*qp*	?XM=0		•	
832*qp*	?SB=0		•	
834*qp*	?SR=0		•	
838*qp*	?MP=0		•	
83*nqp*	?HS=0	*n*	•	
84*n*	ST=0	*n*		
85*n*	ST=1	*n*		
86*nqp*	?ST=0	*n*	•	
87*nqp*	?ST=1	*n*	•	
88*nqp*	?P≠	*n*	•	
89*nqp*	?P=	*n*	•	
8A0*qp*	?A=B	A	•	
8A1*qp*	?B=C	A	•	
8A2*qp*	?C=A	A	•	
8A3*qp*	?D=C	A	•	
8A4*qp*	?A≠B	A	•	
8A5*qp*	?B≠C	A	•	
8A6*qp*	?C≠A	A	•	
8A7*qp*	?D≠C	A	•	
8A8*qp*	?A=0	A	•	
8A9*qp*	?B=0	A	•	
8AA*qp*	?C=0	A	•	
8AB*qp*	?D=0	A	•	
8AC*qp*	?A≠0	A	•	
8AD*qp*	?B≠0	A	•	
8AE*qp*	?C≠0	A	•	
8AF*qp*	?D≠0	A	•	
8B0*qp*	?A>B	A	•	

Codes hexa	Mnémoniques		R	B
8B1*qp*	?B>C	A	•	
8B2*qp*	?C>A	A	•	
8B3*qp*	?D>C	A	•	
8B4*qp*	?A<B	A	•	
8B5*qp*	?B<C	A	•	
8B6*qp*	?C<A	A	•	
8B7*qp*	?D<C	A	•	
8B8*qp*	?A≥B	A	•	
8B9*qp*	?B≥C	A	•	
8BA*qp*	?C≥A	A	•	
8BB*qp*	?D≥C	A	•	
8BC*qp*	?A≤B	A	•	
8BD*qp*	?B≤C	A	•	
8BE*qp*	?C≤A	A	•	
8BF*qp*	?D≤C	A	•	
8C*srqp*	GOLONG	*label*		
8D*tsrqp*	GOVLNG	*label*		
8E*srqp*	GOSUBL	*label*		
8F*tsrqp*	GOSBVL	*label*		
900*qp*	?A=B	P	•	
901*qp*	?B=C	P	•	
902*qp*	?C=A	P	•	
903*qp*	?D=C	P	•	
904*qp*	?A≠B	P	•	
905*qp*	?B≠C	P	•	
906*qp*	?C≠A	P	•	
907*qp*	?D≠C	P	•	
908*qp*	?A=0	P	•	
909*qp*	?B=0	P	•	
90A*qp*	?C=0	P	•	
90B*qp*	?D=0	P	•	
90C*qp*	?A≠0	P	•	
90D*qp*	?B≠0	P	•	
90E*qp*	?C≠0	P	•	
90F*qp*	?D≠0	P	•	
910*qp*	?A=B	WP	•	
911*qp*	?B=C	WP	•	
912*qp*	?C=A	WP	•	
913*qp*	?D=C	WP	•	
914*qp*	?A≠B	WP	•	
915*qp*	?B≠C	WP	•	
916*qp*	?C≠A	WP	•	
917*qp*	?D≠C	WP	•	

Codes hexa	Mnémoniques		R	B
918*qp*	?A=0	WP	•	
919*qp*	?B=0	WP	•	
91A*qp*	?C=0	WP	•	
91B*qp*	?D=0	WP	•	
91C*qp*	?A≠0	WP	•	
91D*qp*	?B≠0	WP	•	
91E*qp*	?C≠0	WP	•	
91F*qp*	?D≠0	WP	•	
920*qp*	?A=B	XS	•	
921*qp*	?B=C	XS	•	
922*qp*	?C=A	XS	•	
923*qp*	?D=C	XS	•	
924*qp*	?A≠B	XS	•	
925*qp*	?B≠C	XS	•	
926*qp*	?C≠A	XS	•	
927*qp*	?D≠C	XS	•	
928*qp*	?A=0	XS	•	
929*qp*	?B=0	XS	•	
92A*qp*	?C=0	XS	•	
92B*qp*	?D=0	XS	•	
92C*qp*	?A≠0	XS	•	
92D*qp*	?B≠0	XS	•	
92E*qp*	?C≠0	XS	•	
92F*qp*	?D≠0	XS	•	
930*qp*	?A=B	X	•	
931*qp*	?B=C	X	•	
932*qp*	?C=A	X	•	
933*qp*	?D=C	X	•	
934*qp*	?A≠B	X	•	
935*qp*	?B≠C	X	•	
936*qp*	?C≠A	X	•	
937*qp*	?D≠C	X	•	
938*qp*	?A=0	X	•	
939*qp*	?B=0	X	•	
93A*qp*	?C=0	X	•	
93B*qp*	?D=0	X	•	
93C*qp*	?A≠0	X	•	
93D*qp*	?B≠0	X	•	
93E*qp*	?C≠0	X	•	
93F*qp*	?D≠0	X	•	
940*qp*	?A=B	S	•	
941*qp*	?B=C	S	•	
942*qp*	?C=A	S	•	

Codes hexa	Mnémoniques		R	B
943*qp*	?D=C	S	•	
944*qp*	?A≠B	S	•	
945*qp*	?B≠C	S	•	
946*qp*	?C≠A	S	•	
947*qp*	?D≠C	S	•	
948*qp*	?A=0	S	•	
949*qp*	?B=0	S	•	
94A*qp*	?C=0	S	•	
94B*qp*	?D=0	S	•	
94C*qp*	?A≠0	S	•	
94D*qp*	?B≠0	S	•	
94E*qp*	?C≠0	S	•	
94F*qp*	?D≠0	S	•	
950*qp*	?A=B	M	•	
951*qp*	?B=C	M	•	
952*qp*	?C=A	M	•	
953*qp*	?D=C	M	•	
954*qp*	?A≠B	M	•	
955*qp*	?B≠C	M	•	
956*qp*	?C≠A	M	•	
957*qp*	?D≠C	M	•	
958*qp*	?A=0	M	•	
959*qp*	?B=0	M	•	
95A*qp*	?C=0	M	•	
95B*qp*	?D=0	M	•	
95C*qp*	?A≠0	M	•	
95D*qp*	?B≠0	M	•	
95E*qp*	?C≠0	M	•	
95F*qp*	?D≠0	M	•	
960*qp*	?A=B	B	•	
961*qp*	?B=C	B	•	
962*qp*	?C=A	B	•	
963*qp*	?D=C	B	•	
964*qp*	?A≠B	B	•	
965*qp*	?B≠C	B	•	
966*qp*	?C≠A	B	•	
967*qp*	?D≠C	B	•	
968*qp*	?A=0	B	•	
969*qp*	?B=0	B	•	
96A*qp*	?C=0	B	•	
96B*qp*	?D=0	B	•	
96C*qp*	?A≠0	B	•	
96D*qp*	?B≠0	B	•	

Codes hexa	Mnémoniques		R	B
96E*qp*	?C≠0	B	•	
96F*qp*	?D≠0	B	•	
970*qp*	?A=B	W	•	
971*qp*	?B=C	W	•	
972*qp*	?C=A	W	•	
973*qp*	?D=C	W	•	
974*qp*	?A≠B	W	•	
975*qp*	?B≠C	W	•	
976*qp*	?C≠A	W	•	
977*qp*	?D≠C	W	•	
978*qp*	?A=0	W	•	
979*qp*	?B=0	W	•	
97A*qp*	?C=0	W	•	
97B*qp*	?D=0	W	•	
97C*qp*	?A≠0	W	•	
97D*qp*	?B≠0	W	•	
97E*qp*	?C≠0	W	•	
97F*qp*	?D≠0	W	•	
980*qp*	?A>B	P	•	
981*qp*	?B>C	P	•	
982*qp*	?C>A	P	•	
983*qp*	?D>C	P	•	
984*qp*	?A<B	P	•	
985*qp*	?B<C	P	•	
986*qp*	?C<A	P	•	
987*qp*	?D<C	P	•	
988*qp*	?A≥B	P	•	
989*qp*	?B≥C	P	•	
98A*qp*	?C≥A	P	•	
98B*qp*	?D≥C	P	•	
98C*qp*	?A≤B	P	•	
98D*qp*	?B≤C	P	•	
98E*qp*	?C≤A	P	•	
98F*qp*	?D≤C	P	•	
990*qp*	?A>B	WP	•	
991*qp*	?B>C	WP	•	
992*qp*	?C>A	WP	•	
993*qp*	?D>C	WP	•	
994*qp*	?A<B	WP	•	
995*qp*	?B<C	WP	•	
996*qp*	?C<A	WP	•	
997*qp*	?D<C	WP	•	
998*qp*	?A≥B	WP	•	

Codes hexa	Mnémoniques		R	B
999*qp*	?B≥C	WP	•	
99A*qp*	?C≥A	WP	•	
99B*qp*	?D≥C	WP	•	
99C*qp*	?A≤B	WP	•	
99D*qp*	?B≤C	WP	•	
99E*qp*	?C≤A	WP	•	
99F*qp*	?D≤C	WP	•	
9A0*qp*	?A>B	XS	•	
9A1*qp*	?B>C	XS	•	
9A2*qp*	?C>A	XS	•	
9A3*qp*	?D>C	XS	•	
9A4*qp*	?A<B	XS	•	
9A5*qp*	?B<C	XS	•	
9A6*qp*	?C<A	XS	•	
9A7*qp*	?D<C	XS	•	
9A8*qp*	?A≥B	XS	•	
9A9*qp*	?B≥C	XS	•	
9AA*qp*	?C≥A	XS	•	
9AB*qp*	?D≥C	XS	•	
9AC*qp*	?A≤B	XS	•	
9AD*qp*	?B≤C	XS	•	
9AE*qp*	?C≤A	XS	•	
9AF*qp*	?D≤C	XS	•	
9B0*qp*	?A>B	X	•	
9B1*qp*	?B>C	X	•	
9B2*qp*	?C>A	X	•	
9B3*qp*	?D>C	X	•	
9B4*qp*	?A<B	X	•	
9B5*qp*	?B<C	X	•	
9B6*qp*	?C<A	X	•	
9B7*qp*	?D<C	X	•	
9B8*qp*	?A≥B	X	•	
9B9*qp*	?B≥C	X	•	
9BA*qp*	?C≥A	X	•	
9BB*qp*	?D≥C	X	•	
9BC*qp*	?A≤B	X	•	
9BD*qp*	?B≤C	X	•	
9BE*qp*	?C≤A	X	•	
9BF*qp*	?D≤C	X	•	
9C0*qp*	?A>B	S	•	
9C1*qp*	?B>C	S	•	
9C2*qp*	?C>A	S	•	
9C3*qp*	?D>C	S	•	

Codes hexa	Mnémoniques		R	B
9C4*qp*	?A<B	S	•	
9C5*qp*	?B<C	S	•	
9C6*qp*	?C<A	S	•	
9C7*qp*	?D<C	S	•	
9C8*qp*	?A≥B	S	•	
9C9*qp*	?B≥C	S	•	
9CA*qp*	?C≥A	S	•	
9CB*qp*	?D≥C	S	•	
9CC*qp*	?A≤B	S	•	
9CD*qp*	?B≤C	S	•	
9CE*qp*	?C≤A	S	•	
9CF*qp*	?D≤C	S	•	
9D0*qp*	?A>B	M	•	
9D1*qp*	?B>C	M	•	
9D2*qp*	?C>A	M	•	
9D3*qp*	?D>C	M	•	
9D4*qp*	?A<B	M	•	
9D5*qp*	?B<C	M	•	
9D6*qp*	?C<A	M	•	
9D7*qp*	?D<C	M	•	
9D8*qp*	?A≥B	M	•	
9D9*qp*	?B≥C	M	•	
9DA*qp*	?C≥A	M	•	
9DB*qp*	?D≥C	M	•	
9DC*qp*	?A≤B	M	•	
9DD*qp*	?B≤C	M	•	
9DE*qp*	?C≤A	M	•	
9DF*qp*	?D≤C	M	•	
9E0*qp*	?A>B	B	•	
9E1*qp*	?B>C	B	•	
9E2*qp*	?C>A	B	•	
9E3*qp*	?D>C	B	•	
9E4*qp*	?A<B	B	•	
9E5*qp*	?B<C	B	•	
9E6*qp*	?C<A	B	•	
9E7*qp*	?D<C	B	•	
9E8*qp*	?A≥B	B	•	
9E9*qp*	?B≥C	B	•	
9EA*qp*	?C≥A	B	•	
9EB*qp*	?D≥C	B	•	
9EC*qp*	?A≤B	B	•	
9ED*qp*	?B≤C	B	•	
9EE*qp*	?C≤A	B	•	

Codes hexa	Mnémoniques		R	B
9EF*qp*	?D≤C	B	•	
9F0*qp*	?A>B	W	•	
9F1*qp*	?B>C	W	•	
9F2*qp*	?C>A	W	•	
9F3*qp*	?D>C	W	•	
9F4*qp*	?A<B	W	•	
9F5*qp*	?B<C	W	•	
9F6*qp*	?C<A	W	•	
9F7*qp*	?D<C	W	•	
9F8*qp*	?A≥B	W	•	
9F9*qp*	?B≥C	W	•	
9FA*qp*	?C≥A	W	•	
9FB*qp*	?D≥C	W	•	
9FC*qp*	?A≤B	W	•	
9FD*qp*	?B≤C	W	•	
9FE*qp*	?C≤A	W	•	
9FF*qp*	?D≤C	W	•	
A00	A=A+B	P	•	•
A01	B=B+C	P	•	•
A02	C=C+A	P	•	•
A03	D=D+C	P	•	•
A04	A=A+A	P	•	•
A05	B=B+B	P	•	•
A06	C=C+C	P	•	•
A07	D=D+D	P	•	•
A08	B=A+B	P	•	•
A09	C=B+C	P	•	•
A0A	A=C+A	P	•	•
A0B	C=D+C	P	•	•
A0C	A=A-1	P	•	•
A0D	B=B-1	P	•	•
A0E	C=C-1	P	•	•
A0F	D=D-1	P	•	•
A10	A=A+B	WP	•	•
A11	B=B+C	WP	•	•
A12	C=C+A	WP	•	•
A13	D=D+C	WP	•	•
A14	A=A+A	WP	•	•
A15	B=B+B	WP	•	•
A16	C=C+C	WP	•	•
A17	D=D+D	WP	•	•
A18	B=A+B	WP	•	•
A19	C=B+C	WP	•	•

Codes hexa	Mnémoniques		R	B
A1A	A=C+A	WP	•	•
A1B	C=D+C	WP	•	•
A1C	A=A-1	WP	•	•
A1D	B=B-1	WP	•	•
A1E	C=C-1	WP	•	•
A1F	D=D-1	WP	•	•
A20	A=A+B	XS	•	•
A21	B=B+C	XS	•	•
A22	C=C+A	XS	•	•
A23	D=D+C	XS	•	•
A24	A=A+A	XS	•	•
A25	B=B+B	XS	•	•
A26	C=C+C	XS	•	•
A27	D=D+D	XS	•	•
A28	B=A+B	XS	•	•
A29	C=B+C	XS	•	•
A2A	A=C+A	XS	•	•
A2B	C=D+C	XS	•	•
A2C	A=A-1	XS	•	•
A2D	B=B-1	XS	•	•
A2E	C=C-1	XS	•	•
A2F	D=D-1	XS	•	•
A30	A=A+B	X	•	•
A31	B=B+C	X	•	•
A32	C=C+A	X	•	•
A33	D=D+C	X	•	•
A34	A=A+A	X	•	•
A35	B=B+B	X	•	•
A36	C=C+C	X	•	•
A37	D=D+D	X	•	•
A38	B=A+B	X	•	•
A39	C=B+C	X	•	•
A3A	A=C+A	X	•	•
A3B	C=D+C	X	•	•
A3C	A=A-1	X	•	•
A3D	B=B-1	X	•	•
A3E	C=C-1	X	•	•
A3F	D=D-1	X	•	•
A40	A=A+B	S	•	•
A41	B=B+C	S	•	•
A42	C=C+A	S	•	•
A43	D=D+C	S	•	•
A44	A=A+A	S	•	•

Codes hexa	Mnémoniques		R	B
A45	B=B+B	S	•	•
A46	C=C+C	S	•	•
A47	D=D+D	S	•	•
A48	B=A+B	S	•	•
A49	C=B+C	S	•	•
A4A	A=C+A	S	•	•
A4B	C=D+C	S	•	•
A4C	A=A-1	S	•	•
A4D	B=B-1	S	•	•
A4E	C=C-1	S	•	•
A4F	D=D-1	S	•	•
A50	A=A+B	M	•	•
A51	B=B+C	M	•	•
A52	C=C+A	M	•	•
A53	D=D+C	M	•	•
A54	A=A+A	M	•	•
A55	B=B+B	M	•	•
A56	C=C+C	M	•	•
A57	D=D+D	M	•	•
A58	B=A+B	M	•	•
A59	C=B+C	M	•	•
A5A	A=C+A	M	•	•
A5B	C=D+C	M	•	•
A5C	A=A-1	M	•	•
A5D	B=B-1	M	•	•
A5E	C=C-1	M	•	•
A5F	D=D-1	M	•	•
A60	A=A+B	B	•	•
A61	B=B+C	B	•	•
A62	C=C+A	B	•	•
A63	D=D+C	B	•	•
A64	A=A+A	B	•	•
A65	B=B+B	B	•	•
A66	C=C+C	B	•	•
A67	D=D+D	B	•	•
A68	B=A+B	B	•	•
A69	C=B+C	B	•	•
A6A	A=C+A	B	•	•
A6B	C=D+C	B	•	•
A6C	A=A-1	B	•	•
A6D	B=B-1	B	•	•
A6E	C=C-1	B	•	•
A6F	D=D-1	B	•	•

Codes hexa	Mnémoniques		R	B
A70	A=A+B	W	•	•
A71	B=B+C	W	•	•
A72	C=C+A	W	•	•
A73	D=D+C	W	•	•
A74	A=A+A	W	•	•
A75	B=B+B	W	•	•
A76	C=C+C	W	•	•
A77	D=D+D	W	•	•
A78	B=A+B	W	•	•
A79	C=B+C	W	•	•
A7A	A=C+A	W	•	•
A7B	C=D+C	W	•	•
A7C	A=A-1	W	•	•
A7D	B=B-1	W	•	•
A7E	C=C-1	W	•	•
A7F	D=D-1	W	•	•
A80	A=0	P		
A81	B=0	P		
A82	C=0	P		
A83	D=0	P		
A84	A=B	P		
A85	B=C	P		
A86	C=A	P		
A87	D=C	P		
A88	B=A	P		
A89	C=B	P		
A8A	A=C	P	•	
A8B	C=D	P		
A8C	ABEX	P		
A8D	BCEX	P		
A8E	CAEX	P		
A8F	DCEX	P		
A90	A=0	WP		
A91	B=0	WP		
A92	C=0	WP		
A93	D=0	WP		
A94	A=B	WP		
A95	B=C	WP		
A96	C=A	WP		
A97	D=C	WP		
A98	B=A	WP		
A99	C=B	WP		
A9A	A=C	WP	•	

Codes hexa	Mnémoniques		R	B
A9B	C=D	WP		
A9C	ABEX	WP		
A9D	BCEX	WP		
A9E	CAEX	WP		
A9F	DCEX	WP		
AA0	A=0	XS		
AA1	B=0	XS		
AA2	C=0	XS		
AA3	D=0	XS		
AA4	A=B	XS		
AA5	B=C	XS		
AA6	C=A	XS		
AA7	D=C	XS		
AA8	B=A	XS		
AA9	C=B	XS		
AAA	A=C	XS	•	
AAB	C=D	XS		
AAC	ABEX	XS		
AAD	BCEX	XS		
AAE	CAEX	XS		
AAF	DCEX	XS		
AB0	A=0	X		
AB1	B=0	X		
AB2	C=0	X		
AB3	D=0	X		
AB4	A=B	X		
AB5	B=C	X		
AB6	C=A	X		
AB7	D=C	X		
AB8	B=A	X		
AB9	C=B	X		
ABA	A=C	X	•	
ABB	C=D	X		
ABC	ABEX	X		
ABD	BCEX	X		
ABE	CAEX	X		
ABF	DCEX	X		
AC0	A=0	S		
AC1	B=0	S		
AC2	C=0	S		
AC3	D=0	S		
AC4	A=B	S		
AC5	B=C	S		

Codes hexa	Mnémoniques		R	B
AC6	C=A	S		
AC7	D=C	S		
AC8	B=A	S		
AC9	C=B	S		
ACA	A=C	S	•	
ACB	C=D	S		
ACC	ABEX	S		
ACD	BCEX	S		
ACE	CAEX	S		
ACF	DCEX	S		
AD0	A=0	M		
AD1	B=0	M		
AD2	C=0	M		
AD3	D=0	M		
AD4	A=B	M		
AD5	B=C	M		
AD6	C=A	M		
AD7	D=C	M		
AD8	B=A	M		
AD9	C=B	M		
ADA	A=C	M	•	
ADB	C=D	M		
ADC	ABEX	M		
ADD	BCEX	M		
ADE	CAEX	M		
ADF	DCEX	M		
AE0	A=0	B		
AE1	B=0	B		
AE2	C=0	B		
AE3	D=0	B		
AE4	A=B	B		
AE5	B=C	B		
AE6	C=A	B		
AE7	D=C	B		
AE8	B=A	B		
AE9	C=B	B		
AEA	A=C	B	•	
AEB	C=D	B		
AEC	ABEX	B		
AED	BCEX	B		
AEE	CAEX	B		
AEF	DCEX	B		
AF0	A=0	W		

Codes hexa	Mnémoniques		R	B
AF1	B=0	W		
AF2	C=0	W		
AF3	D=0	W		
AF4	A=B	W		
AF5	B=C	W		
AF6	C=A	W		
AF7	D=C	W		
AF8	B=A	W		
AF9	C=B	W		
AFA	A=C	W	•	
AFB	C=D	W		
AFC	ABEX	W		
AFD	BCEX	W		
AFE	CAEX	W		
AFF	DCEX	W		
B00	A=A-B	P	•	•
B01	B=B-C	P	•	•
B02	C=C-A	P	•	•
B03	D=D-C	P	•	•
B04	A=A+1	P	•	•
B05	B=B+1	P	•	•
B06	C=C+1	P	•	•
B07	D=D+1	P	•	•
B08	B=B-A	P	•	•
B09	C=C-B	P	•	•
B0A	A=A-C	P	•	•
B0B	C=C-D	P	•	•
B0C	A=B-A	P	•	•
B0D	B=C-B	P	•	•
B0E	C=A-C	P	•	•
B0F	D=C-D	P	•	•
B10	A=A-B	WP	•	•
B11	B=B-C	WP	•	•
B12	C=C-A	WP	•	•
B13	D=D-C	WP	•	•
B14	A=A+1	WP	•	•
B15	B=B+1	WP	•	•
B16	C=C+1	WP	•	•
B17	D=D+1	WP	•	•
B18	B=B-A	WP	•	•
B19	C=C-B	WP	•	•
B1A	A=A-C	WP	•	•
B1B	C=C-D	WP	•	•

Codes hexa	Mnémoniques		R	B
B1C	A=B-A	WP	•	•
B1D	B=C-B	WP	•	•
B1E	C=A-C	WP	•	•
B1F	D=C-D	WP	•	•
B20	A=A-B	XS	•	•
B21	B=B-C	XS	•	•
B22	C=C-A	XS	•	•
B23	D=D-C	XS	•	•
B24	A=A+1	XS	•	•
B25	B=B+1	XS	•	•
B26	C=C+1	XS	•	•
B27	D=D+1	XS	•	•
B28	B=B-A	XS	•	•
B29	C=C-B	XS	•	•
B2A	A=A-C	XS	•	•
B2B	C=C-D	XS	•	•
B2C	A=B-A	XS	•	•
B2D	B=C-B	XS	•	•
B2E	C=A-C	XS	•	•
B2F	D=C-D	XS	•	•
B30	A=A-B	X	•	•
B31	B=B-C	X	•	•
B32	C=C-A	X	•	•
B33	D=D-C	X	•	•
B34	A=A+1	X	•	•
B35	B=B+1	X	•	•
B36	C=C+1	X	•	•
B37	D=D+1	X	•	•
B38	B=B-A	X	•	•
B39	C=C-B	X	•	•
B3A	A=A-C	X	•	•
B3B	C=C-D	X	•	•
B3C	A=B-A	X	•	•
B3D	B=C-B	X	•	•
B3E	C=A-C	X	•	•
B3F	D=C-D	X	•	•
B40	A=A-B	S	•	•
B41	B=B-C	S	•	•
B42	C=C-A	S	•	•
B43	D=D-C	S	•	•
B44	A=A+1	S	•	•
B45	B=B+1	S	•	•
B46	C=C+1	S	•	•

Codes hexa	Mnémoniques		R	B
B47	D=D+1	S	•	•
B48	B=B-A	S	•	•
B49	C=C-B	S	•	•
B4A	A=A-C	S	•	•
B4B	C=C-D	S	•	•
B4C	A=B-A	S	•	•
B4D	B=C-B	S	•	•
B4E	C=A-C	S	•	•
B4F	D=C-D	S	•	•
B50	A=A-B	M	•	•
B51	B=B-C	M	•	•
B52	C=C-A	M	•	•
B53	D=D-C	M	•	•
B54	A=A+1	M	•	•
B55	B=B+1	M	•	•
B56	C=C+1	M	•	•
B57	D=D+1	M	•	•
B58	B=B-A	M	•	•
B59	C=C-B	M	•	•
B5A	A=A-C	M	•	•
B5B	C=C-D	M	•	•
B5C	A=B-A	M	•	•
B5D	B=C-B	M	•	•
B5E	C=A-C	M	•	•
B5F	D=C-D	M	•	•
B60	A=A-B	B	•	•
B61	B=B-C	B	•	•
B62	C=C-A	B	•	•
B63	D=D-C	B	•	•
B64	A=A+1	B	•	•
B65	B=B+1	B	•	•
B66	C=C+1	B	•	•
B67	D=D+1	B	•	•
B68	B=B-A	B	•	•
B69	C=C-B	B	•	•
B6A	A=A-C	B	•	•
B6B	C=C-D	B	•	•
B6C	A=B-A	B	•	•
B6D	B=C-B	B	•	•
B6E	C=A-C	B	•	•
B6F	D=C-D	B	•	•
B70	A=A-B	W	•	•
B71	B=B-C	W	•	•

Codes hexa	Mnémoniques		R	B
B72	C=C-A	W	•	•
B73	D=D-C	W	•	•
B74	A=A+1	W	•	•
B75	B=B+1	W	•	•
B76	C=C+1	W	•	•
B77	D=D+1	W	•	•
B78	B=B-A	W	•	•
B79	C=C-B	W	•	•
B7A	A=A-C	W	•	•
B7B	C=C-D	W	•	•
B7C	A=B-A	W	•	•
B7D	B=C-B	W	•	•
B7E	C=A-C	W	•	•
B7F	D=C-D	W	•	•
B80	ASL	P		
B81	BSL	P		
B82	CSL	P		
B83	DSL	P		
B84	ASR	P		
B85	BSR	P		
B86	CSR	P		
B87	DSR	P		
B88	A=-A	P	•	•
B89	B=-B	P	•	•
B8A	C=-C	P	•	•
B8B	D=-D	P	•	•
B8C	A=-A-1	P	•	•
B8D	B=-B-1	P	•	•
B8E	C=-C-1	P	•	•
B8F	D=-D-1	P	•	•
B90	ASL	WP		
B91	BSL	WP		
B92	CSL	WP		
B93	DSL	WP		
B94	ASR	WP		
B95	BSR	WP		
B96	CSR	WP		
B97	DSR	WP		
B98	A=-A	WP	•	•
B99	B=-B	WP	•	•
B9A	C=-C	WP	•	•
B9B	D=-D	WP	•	•
B9C	A=-A-1	WP	•	•

Codes hexa	Mnémoniques		R	B
B9D	B=-B-1	WP	•	•
B9E	C=-C-1	WP	•	•
B9F	D=-D-1	WP	•	•
BA0	ASL	XS		
BA1	BSL	XS		
BA2	CSL	XS		
BA3	DSL	XS		
BA4	ASR	XS		
BA5	BSR	XS		
BA6	CSR	XS		
BA7	DSR	XS		
BA8	A=-A	XS	•	•
BA9	B=-B	XS	•	•
BAA	C=-C	XS	•	•
BAB	D=-D	XS	•	•
BAC	A=-A-1	XS	•	•
BAD	B=-B-1	XS	•	•
BAE	C=-C-1	XS	•	•
BAF	D=-D-1	XS	•	•
BB0	ASL	X		
BB1	BSL	X		
BB2	CSL	X		
BB3	DSL	X		
BB4	ASR	X		
BB5	BSR	X		
BB6	CSR	X		
BB7	DSR	X		
BB8	A=-A	X	•	•
BB9	B=-B	X	•	•
BBA	C=-C	X	•	•
BBB	D=-D	X	•	•
BBC	A=-A-1	X	•	•
BBD	B=-B-1	X	•	•
BBE	C=-C-1	X	•	•
BBF	D=-D-1	X	•	•
BC0	ASL	S		
BC1	BSL	S		
BC2	CSL	S		
BC3	DSL	S		
BC4	ASR	S		
BC5	BSR	S		
BC6	CSR	S		
BC7	DSR	S		

Codes hexa	Mnémoniques		R	B
BC8	A=-A	S	•	•
BC9	B=-B	S	•	•
BCA	C=-C	S	•	•
BCB	D=-D	S	•	•
BCC	A=-A-1	S	•	•
BCD	B=-B-1	S	•	•
BCE	C=-C-1	S	•	•
BCF	D=-D-1	S	•	•
BD0	ASL	M		
BD1	BSL	M		
BD2	CSL	M		
BD3	DSL	M		
BD4	ASR	M		
BD5	BSR	M		
BD6	CSR	M		
BD7	DSR	M		
BD8	A=-A	M	•	•
BD9	B=-B	M	•	•
BDA	C=-C	M	•	•
BDB	D=-D	M	•	•
BDC	A=-A-1	M	•	•
BDD	B=-B-1	M	•	•
BDE	C=-C-1	M	•	•
BDF	D=-D-1	M	•	•
BE0	ASL	B		
BE1	BSL	B		
BE2	CSL	B		
BE3	DSL	B		
BE4	ASR	B		
BE5	BSR	B		
BE6	CSR	B		
BE7	DSR	B		
BE8	A=-A	B	•	•
BE9	B=-B	B	•	•
BEA	C=-C	B	•	•
BEB	D=-D	B	•	•
BEC	A=-A-1	B	•	•
BED	B=-B-1	B	•	•
BEE	C=-C-1	B	•	•
BEF	D=-D-1	B	•	•
BF0	ASL	W		
BF1	BSL	W		
BF2	CSL	W		

Codes hexa	Mnémoniques		R	B
BF3	DSL	W		
BF4	ASR	W		
BF5	BSR	W		
BF6	CSR	W		
BF7	DSR	W		
BF8	A=-A	W	•	•
BF9	B=-B	W	•	•
BFA	C=-C	W	•	•
BFB	D=-D	W	•	•
BFC	A=-A-1	W	•	•
BFD	B=-B-1	W	•	•
BFE	C=-C-1	W	•	•
BFF	D=-D-1	W	•	•
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=A+B	A	•	•
C9	C=B+C	A	•	•
CA	A=C+A	A	•	•
CB	C=D+C	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		

Codes hexa	Mnémoniques		R	B
DE	CAEX	A		
DF	DCEX	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	•	•

21

Les masques de clavier

1 Les registres IN et OUT

Nous disposons de deux registres d'entrées/sorties :

- le registre OUT (ou OUTPUT) a une capacité de 3 quartets (12 bits). Il permet de commander le vibreur sonore (buzzer) ou d'envoyer vers le clavier le masque OUT d'une touche,
- le registre IN (ou INPUT) a une capacité de 4 quartets (16 bits). Il permet de récupérer un masque en sortie de clavier.

Vous ne pouvez qu'écrire dans OUT, vous ne pouvez que lire IN.

Pour savoir si une touche du clavier est enfoncée, il suffit de :

- placer dans le registre OUT le masque OUT de la touche à tester. Si le masque OUT de la touche est #*abc*h, il suffit de charger le registre C avec #*abc*h (`LCHEX abc`) puis de copier C dans OUT (`OUT=C`),
- récupérer dans le registre C les signaux en sortie du clavier que contient le registre IN (`C=IN`, ou mieux, `GOSBVL 01160`),
- on charge le masque IN #*defg*h de la touche dans le registre A (`LAHEX 0defg`),
- un produit logique entre les signaux en sortie de clavier (en provenance de IN, stockés dans C) et le masque (dans A) fournit un résultat nul si la touche testée n'est pas pressée (`C=C&A A` puis `?C=0 A`... le test est vrai si la touche n'est pas pressée).

2
Masques des touches

Pour chaque touche du clavier vous sont présentés : le masque OUT, le masque IN et le code buffer (mémoire tampon de clavier).

A OUT : 002 IN : 0010 Buffer : 01	**B** OUT : 100 IN : 0010 Buffer : 02	**C** OUT : 100 IN : 0008 Buffer : 03	**D** OUT : 100 IN : 0004 Buffer : 04	**E** OUT : 100 IN : 0002 Buffer : 05	**F** OUT : 100 IN : 0001 Buffer : 06
MTH OUT : 004 IN : 0010 Buffer : 07	**PRG** OUT : 080 IN : 0010 Buffer : 08	**CST** OUT : 080 IN : 0008 Buffer : 09	**VAR** OUT : 080 IN : 0004 Buffer : 0A	**▲** OUT : 080 IN : 0002 Buffer : 0B	**NXT** OUT : 080 IN : 0001 Buffer : 0C
' OUT : 001 IN : 0010 Buffer : 0D	**STO** OUT : 040 IN : 0010 Buffer : 0E	**EVAL** OUT : 040 IN : 0008 Buffer : 0F	**◀** OUT : 040 IN : 0004 Buffer : 10	**▼** OUT : 040 IN : 0002 Buffer : 11	**▶** OUT : 040 IN : 0001 Buffer : 12
SIN OUT : 008 IN : 0010 Buffer : 13	**COS** OUT : 020 IN : 0010 Buffer : 14	**TAN** OUT : 020 IN : 0008 Buffer : 15	**$\sqrt{x}$** OUT : 020 IN : 0004 Buffer : 16	**y^x** OUT : 020 IN : 0002 Buffer : 17	**$^1/_x$** OUT : 020 IN : 0001 Buffer : 18
ENTER OUT : 010 IN : 0010 Buffer : 19	**+/-** OUT : 010 IN : 0008 Buffer : 1A	**EEX** OUT : 010 IN : 0004 Buffer : 1B	**DEL** OUT : 010 IN : 0002 Buffer : 1C	**⌫** OUT : 010 IN : 0001 Buffer : 1D	
α OUT : 008 IN : 0020 Buffer : 80	**7** OUT : 008 IN : 0008 Buffer : 1F	**8** OUT : 008 IN : 0004 Buffer : 20	**9** OUT : 008 IN : 0002 Buffer : 21	**÷** OUT : 008 IN : 0001 Buffer : 22	
↰ OUT : 004 IN : 0020 Buffer : 40	**4** OUT : 004 IN : 0008 Buffer : 24	**5** OUT : 004 IN : 0004 Buffer : 25	**6** OUT : 004 IN : 0002 Buffer : 26	**×** OUT : 004 IN : 0001 Buffer : 27	
↱ OUT : 002 IN : 0020 Buffer : C0	**1** OUT : 002 IN : 0008 Buffer : 29	**2** OUT : 002 IN : 0004 Buffer : 2A	**3** OUT : 002 IN : 0002 Buffer : 2B	**-** OUT : 002 IN : 0001 Buffer : 2C	
ON OUT : 400 IN : 8000 Buffer : 2D	**0** OUT : 001 IN : 0008 Buffer : 2E	**.** OUT : 001 IN : 0004 Buffer : 2F	**SPC** OUT : 001 IN : 0002 Buffer : 30	**+** OUT : 001 IN : 0001 Buffer : 31	

22

Les objets en mémoire

1 Introduction

Aux yeux de la HP-48, toutes les entités stockées en mémoire sont des objets. Ainsi, un nombre réel ou un code exécutable (programme) est placé en mémoire sous forme d'objet. Il existe différents types de données (des nombres, des chaînes de caractères, des programmes, etc.), il y aura donc en mémoire plusieurs types d'objets.

L'objet est donc une sorte de « capsule » dans laquelle on place une donnée à stocker en mémoire. L'objet va être rangé à une adresse en mémoire. Cette adresse correspond à celle du premier quartet de l'objet, les autres quartets de l'objet étant placés aux adresses suivantes (écriture séquentielle de l'objet en mémoire).

Le prologue d'un objet est un nombre hexadécimal à 5 chiffres occupant les 5 premiers quartets de l'objet. Le prologue définit le type de l'objet. Notez qu'il existe 28 types d'objets reconnus par la HP-48 dont une partie seulement est utilisable via le RPL classique.

Nous présenterons l'implantation en mémoire de chaque objet. Nous noterons « @ » l'adresse de l'objet qui correspond à l'adresse du premier quartet du prologue de l'objet considéré. Un objet pouvant être décomposé en champs, chaque champ aura une adresse notée sous la forme « @ + #*n*h » où #*n*h est un nombre hexadécimal (#*n*h = n_{16}).

Prenez garde ! La HP-48 inverse le contenu de chaque champ d'un objet. Etudiez attentivement l'implantation générique de l'objet que nous présentons sous forme de tableau ainsi que les exemples proposés afin de bien comprendre le mécanisme d'inversion des quartets d'un objet.

Un objet doit être saisi sous la forme d'une chaîne de caractères ne contenant que des chiffres hexadécimaux. C'est à l'aide d'un programme tel que ASS (présenté au premier chapitre) que l'on convertit une chaîne de caractères en l'objet qu'elle code.

La commande TYPE renvoie un type pour chaque objet placé au niveau 1 de la pile. Le tableau ci-dessous présente chaque objet avec son prologue et son type.

Nom de l'objet	Prologue	Type
Binaire système (system binary)	02911	20
Nombre réel (real)	02933	0
Nombre réel double précision (long real)	02955	21
Nombre complexe (complex)	02977	1
Nombre complexe double précision (long complex)	0299D	22
Caractère (character)	029BF	24
Tableau (array)	029E8	3 / 4
Tableau indexé (linked array)	02A0A	23
Chaîne de caractères (string)	02A2C	2
Entier (binary integer)	02A4E	10
Liste (list)	02A74	5
Répertoire (directory)	02A96	15
Expression algébrique (algebraic)	02AB8	9
Objet avec unité (unit)	02ADA	13
Objet « taggué » (tagged)	02AFC	12
Objet graphique (graphic)	02B1E	11
Bibliothèque (library)	02B40	16
Sauvegarde (backup)	02B62	17
Données de bibliothèque (library data)	02B88	26
Pointeur étendu (extended pointer)	02BAA	27
Objet réservé 1 (reserved 1)	02BCC	27
Objet réservé 2 (reserved 2)	02BEE	27
Objet réservé 3 (reserved 3)	02C10	27
Programme RPL (program)	02D9D	8
Code exécutable (code)	02DCC	25
Nom global (global name)	02E48	6
Nom local (local name)	02E6D	7
Accès à une bibliothèque (XLIB name)	02E92	14

Notations utilisées :

- @ est l'adresse de l'objet (adresse de son premier quartet),
- #*n*h est un nombre hexadécimal (en base 16).

Les tailles de chaque champ sont exprimées en nombre de quartets. Le quartet étant, rappelons-le, l'unité d'information au sein de la HP-48.

Pour chaque objet un tableau standardisé récapitule ce que vous devez savoir de l'objet étudié :

Nom de l'objet (*nom de l'objet dans la documentation américaine*)	
Prologue : 02911	**Taille totale** : 10 quartets
Type : 20	**Nombre de champs** : 2
Classe : donnée	**Prologue** : pas de prologue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02911	@	5 quartets
Codage du nombre	@ + #5h	5 quartets
Fin de l'objet	*@ + #Ah*	

Les rubriques du tableau contiennent les informations suivantes :

- le nom de l'objet suivi de l'appellation américaine placée entre parenthèses,
- le prologue constitué de 5 chiffres hexadécimaux,
- la taille totale de l'objet exprimée en quartets,
- le type renvoyé par la fonction TYPE,
- le nombre de champs qui constituent l'objet,
- la classe de l'objet,
- l'éventuel épilogue marquant la fin de l'objet,
- un tableau d'implantation en mémoire donnant l'adresse et la taille de chaque champ (@ étant l'adresse du prologue *et* de l'objet).

Attention ! La taille totale annoncée pour les objets de longueur variable est la plus petite taille possible de l'objet considéré.

Nous avons traduit le mot « library » par « bibliothèque » et non par « librairie » comme c'est le cas dans d'autres ouvrages traitant de l'assembleur HP-48. Nous respectons ainsi la véritable traduction de « library ».

2
Binaire système

Binaire système (*system binary*)	
Prologue : 02911	**Taille totale** : 10 quartets
Type : 20	**Nombre de champs** : 2
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02911	@	5 quartets
Codage du nombre	@ + #5h	5 quartets
Fin de l'objet	*@ + #Ah*	

L'objet entier binaire système est réservé au système et correspond à un nombre entier codé sur 5 quartets (20 bits). Il est utilisé directement par la HP-48 et ne doit pas être confondu avec un autre entier binaire (dont le prologue est 02A4E) destiné à l'utilisateur.

Les 5 quartets de « données utiles » représentent un nombre exprimé dans la base de numération active (base 2, 8, 10 ou 16). L'entier binaire système pourra, par exemple, représenter un prologue ou une adresse, ou encore, servir au transfert de paramètres entre deux programmes.

■ Quelques exemples

En mémoire, les dix quartets #1192009876h représentent un objet de type binaire système dont la valeur est #67890h (à adapter à la base de numération active). Le SATURN inverse le contenu de chaque champ, le prologue #02911h est donc devenu #11920h alors que la donnée #67890h a été codée #09876h. Pour coder l'entier binaire valant #A2B3Ch, il suffit de juxtaposer le prologue « inversé » (#11920h) à la donnée inversée (#C3B2Ah). On obtient en mémoire les dix quartets 11920C3B2A codant l'entier binaire valant #A2B3Ch.

3

Nombre réel

Nombre réel (*real*)	
Prologue : 02933	**Taille totale** : 21 quartets
Type : 0	**Nombre de champs** : 4
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02933	@	5 quartets
Exposant (EEX)	@ + #5h	3 quartets
Mantisse (MANT)	@ + #8h	12 quartets
Signe (SIGN)	@ + #14h	1 quartet
Fin de l'objet	*@ + #15h*	

Cet objet est le nombre réel classique que manipule l'utilisateur. La donnée utile est divisée en trois champs que sont la mantisse, son signe et son exposant. L'exposant est en fait celui d'un facteur 10 associé à la mantisse. Le signe est codé sur un quartet qui vaut #0h si le nombre est positif ou #9h si le nombre est négatif. La mantisse comporte 12 chiffres représentés directement en base 10 par codage en DCB. L'exposant est codé sur trois chiffres en DCB. Si l'exposant est positif, ses 3 quartets DCB représentent directement l'exposant en base 10. Par contre, si l'exposant est négatif, on le remplace par « 1000 moins la valeur absolue de l'exposant ». Les exposants du facteur 10 sont compris entre -499 et 499.

■ Exemple

Quel objet représente en mémoire le réel $3{,}14159 \times 10^{-3}$? Respectant l'ordre des champs, juxtaposons le prologue inversé #33620h, l'exposant inversé #799h (l'exposant du facteur 10 étant négatif, on effectue « 1000 moins la valeur absolue de l'exposant » ce qui donne 997), la mantisse inversée #000000951413h (codage sur 12 quartets) et le signe #0h. On obtient en mémoire les 21 quartets #336207990000009514130h.

4

Nombre réel double précision

Nombre réel double précision (*long real ou extended real*)	
Prologue : 02955	**Taille totale** : 26 quartets
Type : 21	**Nombre de champs** : 4
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02955	@	5 quartets
Exposant (EEX)	@ + #5h	5 quartets
Mantisse (MANT)	@ + #Ah	15 quartets
Signe (SIGN)	@ + #19h	1 quartet
Fin de l'objet	*@ + #1Ah*	

Contrairement au nombre réel dont il vient d'être question, le nombre réel dit « double précision » n'est normalement pas accessible à l'utilisateur et n'est utilisé que pour les résultats intermédiaires d'un calcul en cours. Le codage de cet objet est très proche de celui du nombre réel classique, la seule différence réside en un plus grand nombre de quartets affectés à chaque champ. La donnée utile est divisée en trois champs que sont la mantisse, son signe et l'exposant du facteur 10. Le signe est codé sur un quartet qui vaut #0h si le nombre est positif, ou #9h si le nombre est négatif. La mantisse est codée sur 15 quartets DCB. L'exposant du facteur 10 est codé sur 5 quartets DCB. Si l'exposant est positif, ses 5 quartets DCB représentent directement l'exposant en base 10 (jusqu'à +49999). Par contre, si l'exposant est négatif, on le remplace par « 100000 moins la valeur absolue de l'exposant », le plus petit exposant du facteur 10 est donc -49999. Bien entendu, chaque champ est inversé.

■ Exemple

Reprenant le principe de codage des réels, -456×10^{-19999} sera représenté par les 26 quartets #55920 10008 000000000000654 9h.

5
Nombre complexe

Nombre complexe (*complex*)	
Prologue : 02977	**Taille totale** : 37 quartets
Type : 1	**Nombre de champs** : 7
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02977	@	5 quartets
Exposant (réel)	@ + #5h	3 quartets
Mantisse (réel)	@ + #8h	12 quartets
Signe (réel)	@ + #14h	1 quartet
Exposant (imaginaire)	@ + #15h	3 quartets
Mantisse (imaginaire)	@ + #18h	12 quartets
Signe (imaginaire)	@ + #24h	1 quartet
Fin de l'objet	*@ + #25h*	

Cet objet correspond au nombre complexe manipulable par l'utilisateur. La représentation d'un nombre complexe n'est autre que la juxtaposition de deux nombres réels. Le premier représente la partie réelle du complexe alors que le second représente la partie imaginaire du complexe. Comme toujours, le prologue occupe 5 quartets, il est ici suivi de deux nombres réels codés sur 16 quartets chacun.

■ Exemple

Pour représenter le complexe 1000 0000 0000 + i.2000 0000 0000 on utilise l'objet #77920 110 000000000001 0 110 000000000002 0h où chaque groupe de quartets correspond respectivement au prologue, à l'exposant de la partie réelle (exposant du facteur 10), à la mantisse de la partie réelle (DCB), au signe de la mantisse réelle, à l'exposant de la partie imaginaire (exposant du facteur 10), à la mantisse de la partie imaginaire (DCB) et au signe de la mantisse imaginaire.

6

Nombre complexe double précision

Nombre complexe double précision (*long complex/extended complex*)	
Prologue : 0299D	**Taille totale** : 47 quartets
Type : 22	**Nombre de champs** : 7
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 0299D	@	5 quartets
Exposant (réel)	@ + #5h	5 quartets
Mantisse (réel)	@ + #Ah	15 quartets
Signe (réel)	@ + #19h	1 quartet
Exposant (imaginaire)	@ + #1Ah	5 quartets
Mantisse (imaginaire)	@ + #1Fh	15 quartets
Signe (imaginaire)	@ + #2Eh	1 quartet
Fin de l'objet	*@ + #2Fh*	

La représentation du nombre complexe destiné à l'utilisateur était fondé sur la juxtaposition de deux réels. La représentation de ce nombre complexe « double précision » destiné aux calculs intermédiaires effectués par la HP-48 est fondée sur la juxtaposition de deux nombres réels double précision.

■ Exemple

Nous coderons -10 + i.20 sous la forme d'un objet de type nombre complexe double précision. On obtient les 47 quartets suivants : #0299D 10000 100000000000000 9 10000 200000000000000 0h. Bien entendu, le DCB est utilisé et tous les champs sont inversés. Notez que les mantisses des deux nombres réels double précision comportent une partie réelle limitée à 1 chiffre d'où l'emploi de facteurs valant 10^1. Par exemple, pour représenter une mantisse égale à 457 896 125 548 124, il faut lui associer un exposant 10^{14}.

7
Caractère

Caractère (*character*)	
Prologue : 029BF	**Taille totale** : 7 quartets
Type : 24	**Nombre de champs** : 2
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 029BF	@	5 quartets
Code ASCII	@ + #5h	2 quartets
Fin de l'objet	*@ + #7h*	

Cet objet est l'un des plus simples : le prologue est plus long que les données utiles ! Ne confondez pas l'objet caractère (caractère isolé) avec l'objet chaîne de caractères.

L'objet caractère est accessible à l'utilisateur. Il s'agit tout simplement de l'un des caractères que peut afficher la HP-48. La HP-48 dispose d'un code ASCII étendu comprenant quelques caractères particuliers. Vous pouvez visualiser l'ensemble des caractères affichables par la HP-48 en utilisant la commande CHARS (HP-48 G/GX).

La table ASCII de la HP-48 comporte 256 caractères comme tout ASCII étendu ; notez qu'il y a 127 symboles - codables sur 7 bits - dans la table ASCII de base. Chaque caractère est donc codé par deux quartets, c'est-à-dire un octet. On a 16 × 16 possibilités, donc 256 caractères.

■ Exemple

Pour coder un caractère, il suffit de se reporter à la table ASCII (voir le chapitre consacré à l'ASCII) et de considérer le code hexadécimal associé au caractère. Par exemple, 'Z' a pour code #5Ah. Il est donc codé en mémoire par l'objet #FB920 A5h (n'oubliez pas d'inverser chaque champ).

8

Tableau

Tableau (*array*)	
Prologue : 029E8	**Taille totale** : au moins 27 quartets
Type : 3/4	**Nombre de champs** : au moins 6
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 029E8	@	5 quartets
Longueur hors prologue	@ + #5h	5 quartets
Type des objets	@ + #Ah	5 quartets
Nombre de dimensions	@ + #Fh	5 quartets
Dimension 1 (D_1)	@ + #14h	5 quartets
...	...	...
Dimension n (D_n)	@ + N_D×5 + #14h	5 quartets
Contenu objet 1	@ + N_D×5 + #19h	T_o quartets
...	...	...
Contenu objet D_1x...xD_n	@ + T_t - T_o + #5h	T_o quartets
Fin de l'objet	*@ + T_t + #5h*	

NB :
N_D représente le nombre total de dimensions du tableau,
T_t est la taille totale hors prologue de l'objet tableau,
T_o est la taille de chaque objet du tableau.

L'objet tableau (array) contient un ensemble d'objets qui sont tous du même type. L'objet tableau ayant pour dimension N_D=1 est un vecteur alors que l'objet tableau ayant pour dimension N_D=2 est une matrice. On pourra choisir un nombre de dimensions N_D supérieur à 2 puisque 5 quartets sont réservés au codage de N_D. Les vecteurs et les matrices sont habituellement utilisés pour accueillir des réels ou des complexes. Vous pourrez généraliser leur emploi en plaçant des objets d'un type quelconque dans un objet de type tableau. Remarquez qu'au-delà de la dimension 2 (matrice) des problèmes de gestion de l'objet tableau contrarient une exploitation intéressante de celui-ci.

Comme nous allons le voir, une liste importante de déclarations est associée à l'objet tableau. Le prologue de cet objet est 029E8, ce prologue, comme tous les champs de tous les objets, devra être inversé pour être exploitable. La taille totale T_t de cet objet varie en fonction de la taille et du nombre des objets placés dans le tableau. Il est donc indispensable de préciser la taille de l'objet tableau. Connaissant l'adresse @ du premier quartet de l'objet tableau et la taille de celui-ci, la HP-48 en déduit l'adresse du dernier quartet de l'objet.

On place juste après le prologue la taille totale hors prologue de l'objet tableau. Pour obtenir la taille totale hors prologue T_t de l'objet, il suffit de retirer les 5 quartets du prologue à la taille totale. Après la taille totale hors prologue vient le type des objets contenus dans le tableau. Ces objets sont tous de même type. Le type des objets est codé sur 5 quartets et correspond tout simplement au prologue des objets contenus. Vient ensuite le nombre de dimensions du tableau. Ce nombre est codé sur 5 quartets et peut, par exemple, valoir #00001h pour un vecteur ou #00002h pour une matrice. Il faut ensuite préciser le nombre d'objets par dimension. Pour chaque dimension, on codera le nombre d'éléments sur 5 quartets (par exemple, dans le cas d'une matrice, on précisera le nombre de colonnes et de rangées).

Tous les champs « déclaratifs » étant remplis, on code ensuite chaque objet du tableau. Ce codage dépend bien sûr du type qui a été déclaré pour les objets. Remarquez que vous avez précisé le prologue des objets du tableau dans la partie déclarative. Il vous suffit donc maintenant de placer les uns derrière les autres chaque objet du tableau (sans prologue).

Remarquez que la taille d'un objet tableau est entièrement définie par le type des objets contenus, le nombre de dimensions du tableau ainsi que le nombre d'éléments par dimension. Un tableau contenant des éléments « vides », valant #000...000h occupe donc le même espace en mémoire qu'un tableau rempli de données intéressantes.

■ Exemple

Codons le vecteur [A B C] contenant des caractères. Chaque champ étant inversé, on obtient le prologue #8E920h, la longueur totale #A1000h, le nombre de dimensions #10000h, le nombre d'éléments de la première (et unique) dimension #30000h, puis le codage des trois caractères #41h, #42h et #43h.

Finalement, on obtient les 31 quartets : #8E920 A1000 10000 30000 41 42 43h.

9

Tableau indexé

Tableau indexé (*linked array*)	
Prologue : 02A0A	**Taille totale** : au moins 32 quartets
Type : 23	**Nombre de champs** : au moins 7
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02A0A	@	5 quartets
Longueur hors prologue	@ + #5h	5 quartets
Type des objets	@ + #Ah	5 quartets
Nombre de dimensions	@ + #Fh	5 quartets
Dimension 1 (D_1)	@ + #14h	5 quartets
...	...	...
Dimension n (D_n)	@ + N_D×5 + #14h	5 quartets
Pointeur 1er objet	@+(N_D+1)×5+#14h	5 quartets
...		...
Pointeur objet D_1x...xD_n	@+(N_D+D_1x...xD_n)x5+#14h	5 quartets
Contenu premier objet	@+(N_D+D_1x...xD_n)x5+#19h	T_o quartets
...	...	...
Contenu dernier objet	@ + T_t - T_o + #5h	T_o quartets
Fin de l'objet	*@ + T_t + #5h*	

NB :
D_1 à D_n sont les nombres d'éléments pour chaque dimension,
N_D représente le nombre total de dimensions du tableau,
T_t est la taille totale hors prologue de l'objet tableau indexé,
T_o est la taille de chaque objet du tableau indexé.

Le tableau indexé est un tableau particulier contenant des pointeurs (ou offsets). Un pointeur est un moyen d'accès à un objet mémorisé. Le tableau indexé est à la fois commode et économique pour les tableaux comportant de nombreux objets identiques (un pointeur occupe 5 quartets alors qu'un réel placé dans un tableau en occupe 16) ou pour stocker des tables de correspondance de code.

10
Chaîne de caractères

Chaîne de caractères (*string ou character string*)	
Prologue : 02A2C	**Taille totale** : au moins 12 quartets
Type : 2	**Nombre de champs** : au moins 3
Classe : donnée	**Exécution indirecte** : DOCSTR

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02A2C	@	5 quartets
Longueur hors prologue $L_{t\text{-}p}$	@ + #5h	5 quartets
Caractère 1	@ + #Ah	2 quartets
...	...	...
Caractère n	@ + #A+(n×5)h	2 quartets
Fin de l'objet	*@ + #A+5+(n×5)h*	

Les chaînes de caractères servent à la représentation de messages. Elles consistent en une juxtaposition de codes ASCII sur 2 quartets, chaque code représentant un caractère. Une chaîne de caractères est définie par son prologue, sa longueur totale hors prologue suivie des codes de tous les caractères de la chaîne.

Un vecteur (objet tableau à une dimension) contenant des caractères peut être assimilé à une chaîne facilitant la recherche d'un n-ième caractère au sein de la chaîne.

■ Exemple

La chaîne de caractères "HP-48" sera codée de la façon suivante : le prologue #02A2Ch, la longueur totale hors prologue #00014h puis les codes ASCII des caractères #48h, #50h, #2Dh, #34h et #38h. En inversant chaque champ puis en les juxtaposant, on obtient en mémoire l'objet #C2A20 41000 84 05 D2 43 83h.

11
Entier

Entier utilisateur ou chaîne hexadécimale (*Binary integer/hex string*)	
Prologue : 02A4E	**Taille totale** : au moins 11 quartets
Type : 10	**Nombre de champs** : 3
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02A4E	@	5 quartets
Longueur hors prologue	@ + #5h	5 quartets
Chiffre 1	@ + #Ah	1 quartet
...	...	...
Chiffre n	@ + #5 + (n×5)h	1 quartet
Fin de l'objet	*@ + #5 + 5 + (n×5)h*	

Cet objet correspond à l'entier binaire auquel a normalement accès l'utilisateur. Il est représenté par des chiffres précédés par # et suivis d'une lettre minuscule déterminant la base de numération dans laquelle est exprimée l'entier binaire.

Attention ! La base active (voir le menu BASE accessible par [MTH]) s'applique aux objets de type entier binaire au détriment de la base utilisée lors de leur saisie. La conversion est automatique.

Notez qu'il faut déclarer un quartet par chiffre et que les problèmes d'exposant de facteur 10 n'interviennent pas ici. Le nombre de chiffres hexadécimaux (codage sur un quartet pour chacun) est illimité, on constitue ainsi des chaînes hexadécimales d'une longueur adaptée aux besoins.

■ Exemple

L'entier binaire valant #1Ah est codé par les 12 quartets #E4A20 70000 A1h représentant le prologue, la longueur hors prologue et #1Ah.

12
Liste

Liste (*list*)	
Prologue : 02A74	**Taille totale** : au moins 10 quartets
Type : 5	**Nombre de champs** : au moins 2
Classe : donnée	**Epilogue** : 0312B

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02A74	@	5 quartets
Objet 1	@ + #5h	*selon l'objet*
...	...	...
Objet n	@ + #(n×5)h	*selon l'objet*
Epilogue = 0312B	@ + #5 + (n×5)h	5 quartets
Fin de l'objet	*@ + #A + (n×5)h*	

Cet objet contient une liste d'objets de types différents et quelconques. La liste sert donc à encapsuler une série d'objets divers. Cet objet est borné par un prologue et par un épilogue. Entre le prologue et l'épilogue on trouve des objets complets avec leurs propres prologues. Il est possible de remplacer un objet de la liste par son adresse. Cette facilité est très intéressante dans le cadre de l'utilisation de programmes en mémoire morte.

Notez qu'une liste n'occupant que 10 quartets (pour les deux champs que sont le prologue et l'épilogue) est une liste vide.

■ Quelques exemples

Soit une liste { #1h A "HP" } comportant un entier binaire, un caractère et une chaîne de caractères. On se contente de coder ces trois objets déjà connus et de les encadrer avec un prologue et un épilogue. On obtient #47A20 E4A20 60000 1 FB920 14 C2A20 90000 84 05h où #E4A20 60000 1h code l'entier binaire, #FB920 14h code le caractère et #C2A20 90000 84 05h code la chaîne des caractères.

13
Répertoire

On peut distinguer deux types de répertoires : le répertoire-racine (HOME) et les autres répertoires. L'emboîtement des répertoires détermine le classement des variables de l'utilisateur. Le contenu de HOME constitue le menu principal accessible par la touche [VAR]. Les contenus des autres répertoires (sous-répertoires de HOME) sont accessibles en sélectionnant des articles du menu présentant le contenu de HOME.

Des objets particuliers, du type « bibliothèque » (library) peuvent être attachés à un répertoire. Un nombre quelconque de bibliothèques peut être attaché au répertoire racine alors que seule une unique bibliothèque pourra être attachée à un sous-répertoire. Cette différence explique les différences entre la structure du répertoire racine et celle d'un sous-répertoire.

13.1
Le répertoire racine

Répertoire racine (*Directory HOME*)	
Prologue : 02A96	**Taille totale** : au moins 57 quartets
Type : 15	**Nombre de champs** : au moins 13
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02A96	@	5 quartets
N_B	@ + #5h	3 quartets
N_{B1}	@ + #8h	3 quartets
$@_{HT1}$	@ + Bh	5 quartets
$@_{M1}$	@ + 10h	5 quartets

→

Implantation en mémoire (suite) :

Champ	Adresse	Taille du champ
N_{B2}	@ + #13h	3 quartets
$@_{HT2}$	@ + 18h	5 quartets
$@_{M2}$	@ + 1Dh	5 quartets
...	...	...
N_{Bn}	@+#8h+(n-1)×#Dh	3 quartets
$@_{HTn}$	@+#Bh+(n-1)×#Dh	5 quartets
$@_{Mn}$	@+#10h+(n-1)×#Dh	5 quartets
Décalage sur dernier objet	$@_A$	5 quartets
00000	$@_A$ + #5h	5 quartets
N_{no1}	$@_B$	2 quartets
C_{n1} n°1	$@_B$ + #2h	2 quartets
...	...	...
C_{n1} n°N_{no1}	$@_B$ + #(N_{no1}×2)h	2 quartets
N_{no1}	$@_B$ + #(N_{no1}×2+2)h	2 quartets
Objet n°1	$@_B$ + #(N_{no1}×2+4)h	selon objet 1
Taille zone objet 1 ($@_C$-$@_B$)	$@_C$	5 quartets
N_{no2}	$@_C$ + #2h	2 quartets
C_{n2} n°1	$@_C$ + #4h	2 quartets
...	...	...
C_{n2} n°N_{no2}	$@_c$ + #(N_{no2}×2+2)h	2 quartets
N_{no2}	$@_c$ + #(N_{no2}×2+4)h	2 quartets
Objet n°2	$@_c$ + #(N_{no2}×2+6)h	selon objet 2
Taille zone objet 2 ($@_d$-$@_c$)	$@_d$	5 quartets
...	...	...
N_{non}	$@_e$	2 quartets
C_{nn} n°1	$@_e$ + #2h	2 quartets
...	...	2 quartets
C_{nn} n°N_{non}	$@_e$ + #(N_{non}×2)h	2 quartets
N_{non}	$@_e$ + #(N_{non}×2+2)h	2 quartets
Objet n°n	$@_e$ + #(N_{non}×2+4)h	selon objet n
fin de l'objet	$@_f$	

NB :
N_B est le nombre de bibliothèques attachées au répertoire,
N_{B1}, N_{B2}... N_{Bn} sont les numéros des bibliothèques,
$@_{HT1}$, $@_{HT2}$... $@_{HTn}$ sont les adresses des « hash tables » associées aux bibliothèques de numéros N_{B1}, N_{B2}... N_{Bn},
$@_{M1}$, $@_{M2}$... $@_{Mn}$ sont les adresses des listes de messages associées aux bibliothèques de numéros N_{B1}, N_{B2}... N_{Bn},
Le décalage sur le dernier objet vaut $@_e$-$@_a$.
N_{no1}, N_{no2}... N_{non} sont les nombres de caractères des noms des objets 1 à n,
C_{n1} n°1 à C_{n1} n°N_{no1}, C_{n2} n°1 à C_{n2} n°N_{no2}... C_{nn} n°1 à C_{nn} n°N_{non} sont les caractères (pris un par un) des noms des objets 1 à n.

L'objet correspondant au répertoire racine (HOME) contient des informations relatives aux bibliothèques qui lui sont attachées ainsi que des objets. Ces derniers sont, en fait, des variables de l'utilisateur.

Le répertoire racine commence par le prologue des objets de type répertoire (02A96). On trouve ensuite le nombre (codé sur 3 quartets) de bibliothèques qui lui sont attachées, puis des informations sur chacune de ces bibliothèques. Pour chaque bibliothèque, trois champs présentent :

- le numéro de la bibliothèque,
- l'adresse de la « hash table » de la bibliothèque,
- l'adresse de la liste des messages de la bibliothèque.

Deux objets de type pointeur étendu (« extended pointer ») remplacent ces deux dernières adresses si la « hash table » et la liste des messages de la bibliothèque se trouvent dans des zones de mémoire inaccessibles directement (problèmes de ports notamment). Notez que l'adresse des messages de la bibliothèque vaut #00000h lorsque la liste de messages est vide. Reportez-vous à la description de l'objet de type bibliothèque.

Remarquez qu'un numéro de bibliothèque est loin d'être quelconque. Le tableau ci-dessous résume les plages de numéros utilisées. Chaque numéro est codé sur 3 quartets et est compris entre #000h et #7FFh.

Type de bibliothèque	Plage de numéros
Bibliothèques Hewlett-Packard en ROM	#000h à #100h
Bibliothèques Hewlett-Packard en RAM	#101h à #200h
Bibliothèques hors Hewlett-Packard	#201h à #300h
Bibliothèques de l'utilisateur	#301h à #6FFh
Plage réservée à la HP-48	#700h à #7FFh

Trois bibliothèques sont attachées en permanence au répertoire racine. Leurs numéros sont #700h, #002h et #0ABh. Comme vous pouvez le constater en vous référant au tableau ci-dessus, la première se trouve dans la plage que se réserve la HP-48 alors que les deux suivantes sont placées en ROM.

13.2
Les sous-répertoires

Un sous-répertoire a une structure plus simple que celle du répertoire racine. En effet, un sous-répertoire ne peut accueillir qu'une seule biblio-

thèque. Si aucune bibliothèque n'est attachée à un sous-répertoire, on trouve, dans la structure de l'objet répertoire correspondant, le numéro de la bibliothèque #7FFh.

(sous-)Répertoire (*(sub-)directory*)	
Prologue : 02A96	**Taille totale** : au moins 20 quartets
Type : 15	**Nombre de champs** : au moins 15
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02A96	@	5 quartets
N_B	@ + #5h	3 quartets
Décalage sur dernier objet	$@_A$	5 quartets
00000	$@_A$ + #5h	5 quartets
N_{no1}	$@_B$	2 quartets
C_{n1} n°1	$@_B$ + #2h	2 quartets
...	...	...
C_{n1} n°N_{no1}	$@_B$ + #(N_{no1}×2)h	2 quartets
N_{no1}	$@_B$ + #(N_{no1}×2+2)h	2 quartets
Objet n°1	$@_B$ + #(N_{no1}×2+4)h	selon objet 1
Taille zone objet 1 ($@_C$-$@_B$)	$@_C$	5 quartets
...	...	...
N_{non}	$@_e$	2 quartets
C_{nn} n°1	$@_e$ + #2h	2 quartets
...	...	2 quartets
C_{nn} n°N_{non}	$@_e$ + #(N_{non}×2)h	2 quartets
N_{non}	$@_e$ + #(N_{non}×2+2)h	2 quartets
Objet n°n	$@_e$ + #(N_{non}×2+4)h	selon objet n
fin de l'objet	$@_f$	

Dans le tableau ci-dessus, nous nous sommes contentés de mettre en évidence les différences entre la structure d'un sous-répertoire et celle, déjà vue, du répertoire racine.

Remarquez qu'un répertoire comporte un champ contenant un décalage, qui, ajouté à l'adresse dudit décalage, permet de localiser le dernier objet du répertoire. Le premier objet est précédé de 5 quartets nuls, l'espace réservé à chaque objet (sauf le dernier) est signalé dans un champ contenant une donnée de type $@_2$-$@_1$. Remarquez la répétition du nombre de caractères des noms des objets (utile lors d'un parcours du répertoire de la fin vers le début ?).

14
Expression algébrique

Expression algébrique (*algebraic ou symbolic*)	
Prologue : 02AB8	**Taille totale** : au moins 10 quartets
Type : 9	**Nombre de champs** : au moins 2
Classe : donnée	**Epilogue** : 0312B

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02AB8	@	5 quartets
Objet n°1	@ + #5h	selon objet n°1
...	...	...
Objet n°n	...	selon objet n°n
Epilogue = 0312B	...	5 quartets
Fin de l'objet	selon les tailles des objets	

Cet objet correspond à l'expression algébrique couramment manipulée en RPL. Il est délimité par un prologue (au début) et par un épilogue (à la fin). Cet objet contient lui-même de nombreux objets constituant l'expression algébrique codée en notation polonaise inversée même si la saisie et la restitution de l'objet dans la pile utilisent la notation algébrique classique. Remarquez que la notation polonaise inversée (RPN), ou notation postfixe, n'utilise aucune parenthèse. Notez que les opérations intervenant dans l'expression algébrique sont codées par l'adresse (sur 5 quartets) du « programme » placé en ROM qui correspond à l'opération.

■ Exemple

Codons l'expression algébrique '3×A'. En RPN, '3×A' devient « 3 A × ». L'objet commence par le prologue 8BA20, suivi du premier objet (un réel) codé par 33920 000 000000000003 0, suivi du deuxième objet (un nom global) codé par 84E20 10 16, suivi du troisième objet (adresse #1ADEEh en ROM) codé par EEDA1. Une fois le prologue ajouté, on obtient l'objet #8BA20 33920 000 000000000003 0 84E20 10 16 EEDA1 B2130h.

15
Objet avec unité

Objet avec unité (*unit*)	
Prologue : 02ADA	**Taille totale** : selon l'objet et l'unité
Type : 13	**Nombre de champs** : au moins 4
Classe : donnée	**Epilogue** : 0312B

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02ADA	@	5 quartets
Objet	@ + #5h	selon l'objet
Unité associée à l'objet	selon l'objet	selon l'unité
Epilogue = 0312B	selon l'objet et l'unité	selon l'objet et l'unité
Fin de l'objet	*selon l'objet et l'unité*	

L'objet muni d'une unité est délimité par un prologue (au début) et par un épilogue (à la fin). Il contient une suite d'objets dont le premier est l'objet à munir d'une unité (un réel par exemple) et les suivants sont le codage de l'unité proprement dite à l'aide de chaînes de caractères et d'opérations spécifiques de type produit codées par leurs adresses en ROM (la multiplication est à l'adresse #10B86h en ROM, la division en #10B68h et l'exponentiation en #10B72h).

■ Exemple

Codons 10 $m.s^{-1}$, c'est-à-dire 10 m/s. L'objet commence par le prologue ADA20, suivi de l'objet réel codant 10, c'est-à-dire 33920 100 000000000001 0. On code ensuite l'unité « m/s » qui devient « m s / » en notation polonaise inversée (notation postfixe). « m » et « s » sont des chaînes de caractères codées respectivement par C2A20 70000 D6 et par C2A20 70000 37. On code la division par 86B01 puis la multiplication (de l'unité par l'objet à munir de l'unité) par 68B01. Une fois l'épilogue B2130 ajouté, on obtient l'objet #ADA20 33920 100 000000000001 0 C2A20 70000 D6 C2A20 70000 37 86B01 68B01 B2130h sur 65 quartets.

16
Objet « taggué »

Objet « taggué » ou objet avec étiquette (*Tagged*)	
Prologue : 02AFC	**Taille totale** : selon tag et objet
Type : 12	**Nombre de champs** : au moins 4
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02AFC	@	5 quartets
Longueur L de l'étiquette	@ + #5h	2 quartets
Caractère n°1 de l'étiquette	@ + #7h	2 quartets
...	...	...
Caractère n°n de l'étiquette	@ + #(7+(n-1)×2)h	2 quartets
Objet taggué	@ + #(7+(n)×2)h	selon l'objet taggué
Fin de l'objet	*Selon l'objet taggué*	

Un objet « taggué » est un objet muni d'une étiquette. Celle-ci est une chaîne d'une longueur maximale de 256 caractères. Pour coder un objet taggué, il suffit de coder l'objet normalement et de placer avant celui-ci de quoi le transformer en objet avec étiquette. Autrement dit, on ajoute avant l'objet un prologue spécifique (02AFC), le codage sur 2 quartets de la longueur de l'étiquette, puis le codage sur 2 quartets de chaque caractère de l'étiquette.

■ Exemple

On veut obtenir « CHAR : A », autrement dit, ajouter l'étiquette (tag) « CHAR » à l'objet de type caractère codant « A ». L'objet taggué est constitué d'un prologue CFA20, suivi de 40 qui est la longueur de l'étiquette. On place ensuite les codes hexadécimaux 34 84 14 25 de chaque caractère de l'étiquette. Enfin, on code l'objet de type caractère FB920 14. On obtient un objet taggué codé sur 22 quartets par #CFA20 40 34 84 14 25 FB920 14h.

17
Objet graphique

Objet graphique (*Graphic ou GROB*)	
Prologue : 02B1E	**Taille totale** : au moins 22 quartets
Type : 11	**Nombre de champs** : au moins 5
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02B1E	@	5 quartets
Taille hors prologue	@ + #5h	5 quartets
Nombre de lignes	@ + #Ah	5 quartets
Nombre de colonnes	@ + #Fh	5 quartets
Pixels en haut à gauche	@ + #14h	codage par octet
...	...	...
Pixels en bas à droite	selon GROB	codage par octet
Fin de l'objet	selon GROB	

Une image est une juxtaposition de pixels (« picture element » ou points de l'écran). L'objet est codé par un prologue (02B1E) suivi de la taille hors prologue de l'objet, du nombre de lignes (nombre de pixels verticaux), du nombre de colonnes (nombre de pixels horizontaux), puis de la description pixel par pixel de l'image en partant de l'angle supérieur gauche de l'image jusqu'à l'angle inférieur droit (on parcourt tous les pixels un à un de haut en bas et de gauche à droite). A chaque pixel correspond un bit (un bit à 1 égale un pixel allumé). Le codage s'effectue par octets. Chaque octet représente huit points placés horizontalement et se divise en deux quartets lus à l'envers. Même si l'image ne comporte pas un nombre horizontal de points multiple de huit, nous sommes obligés de coder les pixels « inexistants ». Par exemple, codons l'objet graphique ❑■❑❑❑❑ qui comporte 6 pixels dont 1 est « allumé ». Traduite en bits, l'image devient 0100 00, qui devient l'octet 0100 0000 dont les deux derniers chiffres ne sont pas significatifs. Chaque quartet étant lu à l'envers, $0100\ 0000_2$ devient $0010\ 0000_2$ puis 20_{16} après une conversion en hexadécimal. On obtient l'objet #02B1E 11000 10000 60000 20h sur 27 quartets.

18
Bibliothèque

Bibliothèque (*library*)	
Prologue : 02B40	**Taille totale** : variable
Type : 16	**Nombre de champs** : variable
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02B40	@	5 quartets
L_{thp}	@ + #5h	5 quartets
N_c	@ + #Ah	2 quartets
C_1	@ + #Ch	2 quartets
C_2	@ + #Eh	2 quartets
...	...	...
C_n	@ + #N_c×2+Ah	2 quartets
N_c	@ + #N_c×2+Ch	2 quartets
N_L	@ + #N_c×2+Eh	3 quartets
D_{HT}	$@_A$	5 quartets
D_{MA}	$@_B$	5 quartets
D_{LT}	$@_C$	5 quartets
D_{CO}	$@_D$	5 quartets
Hash table	$@_{HT}$	variable
Message table/array	$@_{MA}$	variable
Link table	$@_{LT}$	variable
$XLIB_1$ (type)	$@_{ObXLIB1}$ -#*7(ou 9)*h	1 ou 3 quartets
N_B de $XLIB_1$	$@_{ObXLIB1}$ -#6h	3 quartets
N_{com} de $XLIB_1$	$@_{ObXLIB1}$ -#3h	3 quartets
$XLIB_1$ (objets)	$@_{ObXLIB1}$	variable
$XLIB_2$ (type)	$@_{ObXLIB2}$ -#*7(ou 9)*h	1 ou 3 quartets
N_B de $XLIB_2$	$@_{ObXLIB2}$ -#6h	3 quartets
N_{com} de $XLIB_2$	$@_{ObXLIB2}$ -#3h	3 quartets
$XLIB_2$ (objets)	$@_{ObXLIB2}$	variable
...	...	...
$XLIB_n$ (type)	$@_{ObXLIBn}$ -#*7(ou 9)*h	1 ou 3 quartets
N_B de $XLIB_n$	$@_{ObXLIBn}$ -#6h	3 quartets
N_{com} de $XLIB_n$	$@_{ObXLIBn}$ -#3h	3 quartets

Implantation en mémoire (suite) :

Champ	Adresse	Taille du champ
$XLIB_n$ (objets)	$@_{ObXLIBn}$	variable
Objet masqué 1	$@_{OM1}$	variable
Objet masqué 2	$@_{OM2}$	variable
...	...	...
Objet masqué *n*	$@_{OMn}$	variable
CO	$@_{CO}$	variable
CRC	@ + L_{thp} + #1h	4 quartets
Fin de l'objet	*@ + L_{thp} + #5h*	

NB :
L_{thp} est la longueur totale hors prologue - et hors CRC - de l'objet bibliothèque,
N_c est le nombre de caractères du nom de la bibliothèque,
C_1, C_2... C_n sont les N_c caractères du nom de la bibliothèque,
N_L est le numéro de la bibliothèque,
D_{HT} est le décalage d'adresse (ou offset) $@_{HT}$-$@_A$ pour atteindre la table « hash table »,
D_{MA} est le décalage d'adresse (ou offset) $@_{MA}$-$@_B$ pour atteindre la table « message table » (ou « message array »),
D_{LT} est le décalage d'adresse (ou offset) $@_{LT}$-$@_C$ pour atteindre la table « link table »,
D_{CO} est le décalage d'adresse $@_{CO}$-$@_D$ pour atteindre CO qui correspond à un objet de configuration (config object),
CO est l'objet de configuration,
CRC est le code de redondance cyclique (CRC) correspondant au checksum de l'objet bibliothèque.

18.1
L'objet bibliothèque

La structure d'un objet de type bibliothèque est à la fois compliquée et variable. On trouve dans cet objet quatre autres objets avec leurs structures propres. Ces objets sont la « hash table », la « message table », la « link table » et l'objet de configuration. La bibliothèque contient les décalages d'adresses (aussi appelés « offset ») permettant de localiser chacun de ces quatre objets au sein de l'objet bibliothèque.

Une bibliothèque peut ne pas avoir de nom. Dans ce cas, elle ne figure pas dans le menu *library* et les champs placés aux adresses @+#Ah à @+#N_c×2+Ch sont remplacés par un unique champ N_c valant #00h en @+#Ah. Une bibliothèque qui n'a pas de nom, n'a donc qu'un seul champ N_c au lieu de deux.

L'objet bibliothèque contient entre autres :

- la hash table (ou table des liens),
- la message table (ou message array, ou table des messages),
- la link table (ou table des liens),
- l'objet de configuration (config object).

On localise ces quatre objets dans l'objet bibliothèque à l'aide de décalages (ou offsets) utilisés pour des sauts relatifs vers lesdits objets. Un décalage valant #00000h indique que l'objet auquel il devrait donner accès n'existe pas.

Mis à part les quatre objets dont il vient d'être question, une bibliothèque contient deux grandes catégories d'objets : les visibles (XLIB) et les masqués. Pour chaque objet visible de la bibliothèque, on précise :

- le type (de 1 à 3 quartets),
- le numéro de bibliothèque (3 quartets),
- le numéro de commande (3 quartets).

Comme son nom le laisse deviner, le champ « type » d'un objet visible de la bibliothèque est représentatif de la nature dudit objet visible.

L'objet de configuration (config object) est évalué à la suite d'un arrêt du système.

Le code de redondance cyclique (CRC), ou checksum, est calculé à partir d'un processus décrit avec l'objet sauvegarde auquel vous devrez vous reporter. Le code de redondance cyclique est calculé à partir de tous les quartets en partant du prologue (non compris) de l'objet bibliothèque jusqu'au CRC (non compris).

18.2

La table des noms ou « hash table »

La hash table est un moyen - ici assez primitif - utilisé pour accélérer l'accès aux fonctions d'une bibliothèque. Il s'agit en fait de classer les fonc-

tions par catégories. La catégorie d'une fonction (ou commande) est déterminée par le nombre de caractères constituant son nom.

On crée ainsi 16 catégories pour les fonctions dont le nom comporte 1, 2, 3... 16 ou plus caractères (toutes les fonctions dont les noms comportent 16 caractères ou plus sont rangées dans la même catégorie).

Table des hash-code (*hash table*)

Prologue : 02A4E	**Taille totale** : variable
Type : néant	**Nombre de champs** : variable
Classe : sous-objet	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02A4E	$@_{HT}$	5 quartets
L_{thp}	$@_{HT}$ + #5h	5 quartets
D_{c1}	$@_{DC1}$ (= $@_{HT}$+#Ah)	5 quartets
D_{c2}	$@_{DC2}$ (= $@_{DC1}$+#5h)	5 quartets
D_{c3}	$@_{DC3}$ (= $@_{DC2}$+#5h)	5 quartets
D_{c4}	$@_{DC4}$ (= $@_{DC3}$+#5h)	5 quartets
...	...	...
D_{c16}	$@_{DC16}$ (= $@_{DC15}$+#5h)	5 quartets
L_{LN}	$@_{HT}$ + #5Ah	5 quartets
Nb_{nc1}	$@_{Nbnc1}$	2 quartets
C_{N1} n° 1	$@_{Nbnc1}$ +#2h	2 quartets
C_{N1} n° 2	$@_{Nbnc1}$ +#4h	2 quartets
...	...	...
C_{N1} n° Nb_{nc1}	$@_{Nbnc1}$ +#Nb_{nc1}h	2 quartets
N_{c1}	$@_{Nbnc1}$ +#Nb_{nc1}+2h	2 quartets
Nb_{nc2}	$@_{Nbnc2}$	2 quartets
C_{N2} n° 1	$@_{Nbnc2}$ +#2h	2 quartets
C_{N2} n° 2	$@_{Nbnc2}$ +#4h	2 quartets
...	...	...
C_{N2} n° Nb_{nc2}	$@_{Nbnc2}$ +#Nb_{nc2}h	2 quartets
N_{c2}	$@_{Nbnc2}$ +#Nb_{nc2}+2h	2 quartets
...	...	...
Nb_{ncn}	$@_{Nbncn}$	2 quartets
C_{Nn} n° 1	$@_{Nbncn}$ +#2h	2 quartets
C_{Nn} n° 2	$@_{Nbncn}$ +#4h	2 quartets
...	...	...
C_{Nn} n° Nb_{ncn}	$@_{Nbncn}$ +#Nb_{ncn}h	2 quartets
N_{cn}	$@_{Nbncn}$ +#Nb_{ncn}+2h	2 quartets

Implantation en mémoire (suite) :

Champ	Adresse	Taille du champ
D_{nc1}	$@_{nc1}$	5 quartets
D_{nc2}	$@_{nc2}$	5 quartets
D_{nc3}	$@_{nc3}$	5 quartets
...	...	...
D_{ncn}	$@_{ncn}$	5 quartets
Fin de l'objet	*@ + L_{thp} #5h*	

NB :

L_{thp} est la longueur totale hors prologue de l'objet hash table,

D_{c1}, D_{c2}, D_{c3}... D_{c16} sont les décalages d'adresses à appliquer pour accéder aux catégories 1 à 16 qui correspondent au nombre de caractères des noms des commandes. Par exemple, on a $D_{C1}=@N_{bnc1}-@_{DC1}$, comme on décale vers une adresse supérieure, le décalage est obtenu par soustraction de l'adresse $@_{DC1}$ (de D_{c1}) à l'adresse à atteindre,

L_{LN} est la longueur de la liste des noms,

Nb_{nc1}, Nb_{nc2}, Nb_{nc3}... Nb_{ncn} sont les nombres de caractères des noms des commandes,

C_{N1} n°1 à C_{N1} n°Nb_{nc1}, C_{N2} n°1 à C_{N2} n°Nb_{nc2}... C_{Nn} n°1 à C_{Nn} n°Nb_{ncn} sont les caractères - codés en ASCII - de chaque nom de commande,

N_{C1}, N_{C2}, N_{C3}... N_{CN} sont les numéros des commandes,

D_{nc1}, D_{nc2}, D_{nc3}... D_{ncN} sont les décalages d'adresses (offsets) à appliquer pour accéder aux nom des commandes. Par exemple, on a $D_{nc1} = @_{nc1}-@Nb_{nc1}$. On décale ici vers une adresse inférieure. La valeur du décalage recherché est donc obtenue en ôtant l'adresse à atteindre (par exemple $@Nb_{nc1}$) à l'adresse du décalage (par exemple $@_{nc1}$).

On remarque qu'une fonction sans nom n'est pas représentée dans la hash table (elle l'est dans la link table). Les noms sont codés par caractères pas caractères, chaque caractère étant codé en ASCII.

A chaque commande (ou fonction) de la bibliothèque correspond un nom et un numéro de commande au sein de la bibliothèque. Ce sont ces deux dernières informations qu'est chargée d'ordonner la hash table. Comme son nom l'indique, celle-ci est une table. Les noms des commandes de la bibliothèque y sont classés par ordre de nombre de caractères croissant. Chaque nom classé dans la table est suivi du numéro de la commande au sein de la bibliothèque.

On trouve dans la hash table deux séries de champs correspondant à des décalages d'adresses (offsets). La première série de décalage permet d'accéder au différentes catégories de noms (il y a toujours 16 catégories de noms). La seconde, placée à la fin de l'objet hash table, donne accès à chaque nom de commande.

Remarquez que le nom d'un objet d'une bibliothèque n'est pas forcément représenté dans la hash table. Dans ce cas, l'objet est dit caché et n'est normalement pas accessible. Cette catégorie d'objets masqués ne peut pas être appelée par la fonction XLIB *nom_de_l'_objet*.

18.3
La table des messages ou « message table »

La table des messages peut prendre deux formes en fonction du type de la bibliothèque. La table des messages peut être assimilée à un objet de type tableau (à une dimension) ou à un objet de type tableau indexé (à une dimension).

Si la table des messages prend la forme d'un tableau (prologue 029E8), on y trouve le nombre de messages suivi de chaque message commençant par l'annonce de sa longueur (en nombre de caractères) suivi du code ASCII de chaque caractère. Si la table des messages prend la forme d'un tableau indexé (prologue 02A0A), on y trouve le nombre de messages, la liste des décalages donnant accès à chaque message puis chaque message constitué de sa longueur (en nombre de caractères) suivi de chaque caractère du message codé en ASCII.

18.4
La table des liens ou « link table »

La table des liens est un entier binaire codant la liste des adresses des objets de la bibliothèque. On peut couper la table des liens en segments de 5 quartets. Le premier code le prologue (02A4E) suivi de la longueur totale hors prologue de la table des liens. On trouve ensuite les adresses de chaque objet. La table des liens n'est donc qu'une table d'adresses, chacune d'elles donnant accès à un objet de la bibliothèque considérée.

19
Sauvegarde

Sauvegarde (*backup*)	
Prologue : 02B62	**Taille totale** : variable
Type : 17	**Nombre de champs** : variable
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02B62	@	5 quartets
L_{thp}	@ + #5h	5 quartets
Nb_{cno}	@ + #Ah	2 quartets
C_{no} n°1	@ + #A+2h	2 quartets
C_{no} n°2	@ + #A+2×2h	2 quartets
...	...	...
C_{no} n°Nb_{cno}	@ + #A+2×Nb_{cno}h	2 quartets
Nb_{cno}	@ + #A+2×Nb_{cno}+2h	2 quartets
OS	$@_{OS}$	selon OS
02911 (prologue)	$@_{OS}$ + L_{OS}	5 quartets
0	$@_{OS}$ + L_{OS} + #1h	1 quartet
CRC	@ + L_{thp}	4 quartets
Fin de l'objet	@ + L_{thp} + #5h	

NB :
L_{thp} est la longueur totale hors prologue et hors CRC (qui correspond checksum) de l'objet sauvegarde,
Nb_{cno} est le nombre de caractères constituant le nom de l'objet,
C_{no} est un caractère du nom de l'objet,
OS est l'objet sauvegardé,
L_{OS} est la longueur de l'objet sauvegardé.

L'objet de sauvegarde permet d'encapsuler plusieurs objets. En pratique, on y place un unique objet précédé par son nom et suivi par sa somme de contrôle (checksum). L'objet sauvegarde permet de retrouver un objet stocké sur le port de votre choix. Il s'agit du port 0 dans le cas d'utilisation le plus simple. Remarquez que deux prologues se trouvent au sein de l'ob-

jet sauvegarde. Le second est celui d'un entier binaire système (prologue 02911) et se trouve juste avant le checksum de l'objet. Le checksum, ou somme de contrôle est un code de redondance cyclique (CRC). Ce valeur de contrôle est codée sur 4 quartets et est précédée du quartet #0h (voir l'implantation en mémoire). Le checksum est calculé très rapidement au moyen d'un circuit spécialisé (logique câblée). Le checksum est le reste d'une division de polynômes ; le diviseur est le polynôme $x^{16}+x^{12}+x^5+1$ alors que le dividende est un polynôme dont les coefficients sont les bits de l'objet dont on calcule le checksum.

20
Données de bibliothèque

Données de bibliothèque (*Library data*)	
Prologue : 02B88	**Taille totale** : variable
Type : 26	**Nombre de champs** : variable
Classe : donnée	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02B88	@	5 quartets
L_{thp}	@ + #5h	5 quartets
D_b	@ + #Ah	L_{Db}
Fin de l'objet	*@ + #A+L_{Db}h*	

NB :
L_{thp} est la longueur totale hors prologue de l'objet,
D_b représente les données de la bibliothèque,
L_{Db} est la longueur de D_b.

Mis à part un prologue spécifique (02B88) et la longueur totale hors prologue, le contenu de cet objet n'est déterminé que par le programmeur dans le cadre de la réalisation de bibliothèque.

21
Pointeur étendu

Pointeur étendu (*extended pointer*)	
Prologue : 02BAA	**Taille totale** : 15 quartets
Type : 27	**Nombre de champs** : 3
Classe : N.C.	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02BAA	@	5 quartets
adresse de l'objet	@ + #5h	5 quartets
adresse programme	@ + #Ah	5 quartets
Fin de l'objet	*@ + #Fh*	

Certaines zones des la mémoire étant masquées, il faut, pour accéder à un objet se trouvant dans une mémoire masquée, préciser:

- l'adresse de l'objet,
- l'adresse d'un programme donnant accès à la zone de mémoire où se trouve l'objet recherché.

Des routines spécialisées (en ROM) rendent visibles des portions masquées de la mémoire morte. Vous pouvez aussi reconfigurer la mémoire.

22
Objets réservés

Les objets dont les prologues sont #02BCCh, #02BEEh et #02C10h ne sont pas utilisés actuellement. Il est vraisemblable qu'ils soient réservés à

des objets qu'utiliseront de futures versions de la HP-48. Le programmeur curieux pourra essayer d'exploiter ces prologues...

Objets réservés n°1, n°2 et n°3 (reserved #1, #2 et #3)	
Prologues : voir ci-après	**Taille totale** : N.C.
Type : 27	**Nombre de champs** : au moins 3
Classe : N.C.	**Epilogue** : N.C.

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue	@	5 quartets
L_{thp}	@ + #5h	5 quartets
Données...	@ + #Ah	selon les données
Fin de l'objet	*selon les données !*	

Le tableau ci-dessous rappelle les prologue attribués aux objets réservés :

Objets	Prologues
Objet réservé 1	#02BCCh
Objet réservé 2	#02BEEh
Objet réservé 3	#02C10h

23
Programme RPL

Bien que ce livre soit consacré à l'assembleur, la description des objets traités par la HP-48 ne peut nous faire oublier qu'un programme écrit en RPL est un objet comme les autres à ceci près qu'il est exécutable (il entre dans la classe des procédures).

La structure de l'objet programme RPL est très proche de celle de la liste. Un épilogue et un prologue encadrent une suite d'objets. Ceux-ci peuvent être remplacés par leur adresse en mémoire. C'est notamment le cas des

fonctions de la HP-48. Par exemple, l'adresse #1AB67h désigne la fonction d'addition.

Les guillemets « et » encadrent les objets (des fonctions et leurs arguments) d'un programme RPL. Ces guillemets peuvent aussi être utilisées (toujours par paires) pour délimiter des procédures indépendantes au sein d'un programme RPL. Les symboles « se trouvent respectivement aux adresses #2361Eh et #23639h.

Programme RPL (*program*)	
Prologue : 02D9D	**Taille totale** : au moins 20 quartets
Type : 8	**Nombre de champs** : au moins 4
Classe : donnée	**Epilogue** : 0312B

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02D9D	@	5 quartets
«	@ + #5h	5 quartets
objets divers...	@ + #Ah	5 quartets
...	...	...
»	$@_f$ - #Ah	5 quartets
Epilogue = 0312B	$@_f$ - #5h	5 quartets
Fin de l'objet	$@_f$	

Un programme RPL se compose d'une foule d'objet : des fonctions qui seront appelées par leur adresse en mémoire morte et les arguments de celles-ci (des nombres, les listes, des chaînes de caractères ,etc.).

24
Code exécutable

L'objet code exécutable sert au stockage des codes utilisés pour commander la HP-48 via le langage machine.

Code exécutable (*code*)	
Prologue : 02DCC	**Taille totale** : au moins 10 quartets
Type : 25	**Nombre de champs** : 2 ou 3
Classe : procédure	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02DCC	@	5 quartets
L_{thp}	@ + #5h	5 quartets
codes...	@ + #Ah	variable
...	...	
Fin de l'objet	@ + L_{thp} + #5h	

25
Nom global

Nom global (*global name*)	
Prologue : 02E48	**Taille totale** : au moins 7 quartets
Type : 6	**Nombre de champs** : au moins 2
Classe : identificateur	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02E48	@	5 quartets
N_c	@ + #5h	2 quartets
Caractère n°1	@ + #5+2×1h	2 quartets
Caractère n°2	@ + #5+2×2h	2 quartets
Caractère n°3	@ + #5+2×3h	2 quartets
Caractère n°4	@ + #5+2×4h	2 quartets

Implantation en mémoire (suite) :

Champ	Adresse	Taille du champ
Caractère n°5	@ + #5+2×5	5 quartets
...	...	...
Caractère n°N_c	@ + #5+2×N_ch	2 quartets
Fin de l'objet	@ + *#7+2×N_ch*	

NB :
N_c est le nombre de caractères (codés en ASCII) constituant le nom global.

Un nom global est placé entre des délimiteurs ' et '. Un exemple de nom global est 'HP-48'. Les caractères du nom sont codés en ASCII.

26
Nom local

Nom local (*local name*)	
Prologue : 02E6D	**Taille totale** : variable
Type : 7	**Nombre de champs** : variable
Classe : identificateur	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02E48	@	5 quartets
Nombre de caractères N_c	@ + #5h	2 quartets
Caractère n°1	@ + #5+2×1h	2 quartets
Caractère n°2	@ + #5+2×2h	2 quartets
...	...	...
Caractère n°N_c	@ + #5+2×N_ch	2 quartets
Fin de l'objet	@ + *#7+2×N_ch*	

Seul le prologue distingue le nom local du nom global. Un nom local n'est accessible que dans un environnement particulier (contrairement au nom global).

27
Accès à une bibliothèque

Accès à une bibliothèque (*XLIB name*)	
Prologue : 02E92	**Taille totale** : 11 quartets
Type : 14	**Nombre de champs** : 3
Classe : identificateur	**Epilogue** : pas d'épilogue

Implantation en mémoire :

Champ	Adresse	Taille du champ
Prologue = 02E92	@	5 quartets
N_B	@ + #5h	3 quartets
N_{com}	@ + #8h	3 quartets
Fin de l'objet	*@ + #Bh*	

NB :
N_B est le numéro de la bibliothèque,
N_{com} est le numéro de la commande.

C'est grâce à « *XLIB nom_de_commande* » que l'on exploite une commande placée dans une bibliothèque (library). Un objet XLIB name se contente de contenir un numéro de bibliothèque et un numéro de commande, il ne s'agit donc que d'un moyen d'accès.

L'évaluation d'une commande d'une bibliothèque peut être obtenue à l'aide de la commande LIBEVAL. Pour cela, il suffit - à travers la pile RPL classique - de donner en argument à LIBEVAL un entier système (de type #BBBCCCh) dont les trois premiers chiffres (#BBBh) correspondent au

numéro de la bibliothèque alors que les trois suivants (#CCCh) correspondent au numéro de la commande.

■ **Exemples**

Le commande D→R porte le numéro de commande #070h dans la bibliothèque numéro #002h. La commande D→R peut donc être représentée par les 11 quartets #29E20 200 070h où #29E20h est le prologue inversé et où #200h et #070h sont les numéros inversés de la bibliothèque et de la commande.

#29E20 200 D30h représente la commande #03Dh de la bibliothèque #002h. Reportez-vous aux informations données en annexe et constatez qu'il s'agit de la commande ABS.

28
Primitives RPL

La mémoire morte contient des petites portions de programmes qui correspondent aux instructions RPL ou aux primitives de celles-ci. En effet, une instruction peut être décomposée en une succession d'instructions plus élémentaires. Une instructions RPL est donc un « assemblage » de primitives RPL.

On peut exécuter une instruction RPL (ou une primitive d'une instruction RPL) en exécutant le programme auquel elle correspond. Il suffit pour cela de « brancher » l'exécution du programme sur l'adresse du « programme » en mémoire morte. La liste des adresse en mémoire morte des instructions RPL est présentée en annexe.

On appelle programmation en « external » ou en « RPL système » l'utilisation des adresses en mémoire morte des instructions RPL (ou leurs primitives).

En RPL classique, il suffit, au sein d'un programme délimités par « et » de placer une adresse en ROM de la forme #NNNNNh suivie de SYSEVAL pour exécuter un mini-programme en ROM. En fait, on gagne à créer un objet programme RPL particulier. Celui-ci commence par le prologue

D9D20 qui est suivi d'adresses de minis programmes en ROM. Ces adresses sont codés sur 5 quartets et que certaines vous présentées à la fin de cet ouvrage. Notez que ce type de programmation fourni des programmes nettement plus rapides que les programmes RPL sans pour autant atteindre la rapidité d'exécution de l'assembleur !

29

Remarques importantes

■ **Ordre de présentation des objets**

Note : Les objets vont été présentés en respectant l'ordre des prologues croissants de #02911h à #02E92h. Les primitives RPL peuvent constituer des programmes dotés du prologue #02D9Dh. Etant donné qu'il s'agit là d'un cas très particulier, nous avons préféré relégué celui-ci à la fin de ca chapitre.

■ **Appelations**

Nous avons tenté de regrouper toutes les appellations rencontrées dans la littérature HP, qu'elle soit française ou américaine.

■ **La « double précision »**

Nous avons appelé certains objets « nombre réel double précision » et « nombre complexe double précision ». Cette appellation est un peu abusive puisqu'il ne s'agit pas d'un doublage du nombre de bits. Ces noms auraient aussi pu être appelés « nombre réel étendu » (ou « réel long ») et « nombre complexe étendu » (« complexe long »).

■ **La délimiteurs d'objets**

Les délimiteurs des objets sont pas codés. C'est le prologue qui détermine les délimiteurs affichés lors de l'édition. Cela dit les délimiteurs « et » d'un programme RPL doivent être codés.

■ Taille fixe ou variable des objets

Certains objets ont une taille fixe (le réel par exemple). Le prologue détermine donc la longueur de l'objet, et par conséquent, l'adresse de son dernier quartet. Dans le cas d'un objet de taille variable, la longueur et l'adresse de fin de l'objet peuvent être indiquées à la HP-48 de deux façons différentes : soit un épilogue borne l'objet, soit la longueur totale hors prologue de l'objet (souvent notée L_{thp}) est placée juste après le prologue en tête de l'objet.

■ Codage de la mantisse

Attention ! Dans le cas des réels, complexes, réels double précision et complexes double précision, la mantisse est de la forme 1,234567890 et non 1234567890. La mantisse a donc une partie entière limitée à un chiffre. Ainsi, pour représenter 12,34567890, il faudra que l'exposant du facteur 10 (appelé couramment « exposant ») vaille 1... Car il faut multiplier la mantisse par 10^1=10...

Ainsi, pour faire passer tous les chiffres de la mantisse (nombres réels et complexes) devant la virgule, il faudra que l'exposant vaille 11 (en fait, 10^{11}). De même pour les réels double précision et complexes double précisions, l'exposant du facteur 10 devra valoir 14 pour que tous les chiffres de la mantisse passent devant la virgule.

■ Codage... Attention !

Pour créer un objet, il suffit de saisir dans une chaîne de caractères (délimitées par " et ") chaque champ de l'objet à créer. Tous les champs devront bien entendu être inversés, ainsi #1A2h deviendra #2A1h. La chaîne de caractères étant au premier niveau de la pile exécutez le programme ASS (voir premier chapitre) ou le programme GASS, couramment employés par les programmeurs en assembleur. Il ne vous reste plus ensuite qu'à sauver votre objet dans une variable globale à l'aide de [STO].

Sauvegardez le contenu de votre mémoire avant de créer manuellement vos propres objets... Les « Memory lost » sont moins rares que vous le pensez !

23

Structure de la mémoire vive des entrées/sorties

1 Vue d'ensemble

La mémoire de la HP-48 est constituée de six modules que sont, par ordre de priorité décroissante : la mémoire vive des entrées/sorties, la mémoire vive interne, le gestionnaire de partitions, la carte mémoire du port 1, la carte mémoire du port 2 et la mémoire morte.

La capacité de la mémoire vive des entrées/sorties est extrêmement modeste : 32 octets (64 quartets). Cette capacité est à comparer aux 32768 octets de la mémoire vive interne des HP-48 S, G et SX, ou aux 524288 octets de la mémoire morte des HP-48 G et GX.

La mémoire vive des entrées/sorties, aussi appelée RAM I/O (« Random Access Memory for Input/Output ») est le module de mémoire ayant le niveau de priorité le plus élevé. La plage d'adresses utilisées par la RAM I/O (adresses #00100h à #0013Fh) masque donc les zones des autres modules de mémoire utilisant les mêmes adresses.

La mémoire vive des entrées/sorties est utilisée pour l'échange de données avec les périphériques (port série RS 232, port infrarouge, écran, etc.). La lecture et l'écriture en RAM I/O sont possibles à l'aide des instructions standard sans reconfiguration de la mémoire. Une grande partie des 256 bits de la RAM I/O joue un rôle intéressant. Notez que, au sein d'un même quartet de cette mémoire, deux bits peuvent jouer des rôles très différents. Nous décrirons donc cette mémoire presque bit par bit.

2
Adresses utiles de la RAM I/O

Pour chaque adresse de la mémoire vive des entrées/sorties (RAM I/O), les bits concernés sont annoncés. A chaque adresse correspond un quartet. Le bit n°0 du quartet est le bit de poids faible, le bit n°3 étant le bit de poids fort. La RAM I/O est présentée dans le sens des adresses croissantes.

#00100h	Bits n°0 à 2	Décalage à gauche de l'écran
On code sur ces trois bits une valeur limitée à #7h qui correspond au nombre de pixels de la marge sur la gauche de l'écran. Notez que seule la partie haute est concernée (les menus ne sont pas décalés à gauche).		

#00100h	Bit n°3	Allumage ou extinction de l'écran
Ce bit vaut 1 lorsque l'écran est allumé, et 0 lorsque l'écran est éteint. Bien que la valeur normale du bit n°3 en #100h soit 1, le faire passer à 0 permet d'augmenter de 10 % la vitesse d'exécution d'un programme.		

#00101h	Bits n°0 à 3	Contraste de l'écran
Le contraste peut prendre 32 valeurs (#20h) codées sur 5 bits (voir aussi le quartet #00102h). Modifier ces 5 bits revient à modifier la tension de référence de l'alimentation de l'écran, et donc, le contraste.		

#00102h	Bit n°0	Contraste de l'écran (suite)
Parmi les 5 bits codant le contraste de l'écran, celui qui a le plus fort poids se trouve #00102h (bit n°0 de ce quartet). Ce bit complète les quatre bits du quartet se trouvant en #00101h.		

#00103h	Bit n°3	Rafraîchissement de l'écran
Le bit n°3 du quartet se trouvant en #00103h contrôle le balayage de l'écran, autrement dit, le renouvellement de l'image affichée (c'est un « rafraîchissement » de l'écran). Le contenu de l'écran est renouvelé ligne par ligne. Le bit n°3 en #00103h vaut normalement 0 et dans ce cas le rafraîchissement a lieu normalement. Si ce bit passe à 1, le rafraîchissement est bloqué et l'affichage (limité à une ligne de pixels) deviendra anarchique. Le blocage du rafraîchissement peut provoquer une détérioration irrémédiable de l'affichage. C'est ce qui a fait dire à certains qu'il est possible de détruire sa HP-48 en programmant (mal) en assembleur.		

#00104h à #00107h	Bits n°0 à 3	CRC (checksum)

La commande RPL BYTES renvoie le nombre de quartets consacrés à un objet mais aussi la somme de contrôle (CRC ou checksum) permettant de vérifier l'intégrité de l'objet. Cette somme de contrôle correspond à un codage sur 4 quartets (4 chiffres hexadécimaux). Le calculateur de CRC est un circuit particulier connecté directement sur le bus. Vous pouvez provoquer un calcul de CRC puis lire la somme de contrôle recherchée en consultant les quartets #00104h, #00105h, #00106h et #00107h, pour cela, procédez de la façon suivante :

- interdisez les interruptions masquables (à l'aide de `INTOFF`),
- placez #0h dans chacun des quatre quartets cités (adresses #00104h à #00107h),
- procédez à une lecture des quartets de l'objet dont la somme de contrôle est recherchée,
- lisez la somme de contrôle sur 4 quartets à partir de #00104h,
- autorisez à nouveau les interruptions (à l'aide de `INTON`).

#00108h	Bits n°0 à 3	Etat des piles

Le quartet en #00108h contient le résultat d'un test des piles (voir aussi en #00109h). A l'issue d'un test de l'alimentation, les bits en #00108h passent à 1 dans les conditions suivantes :

- le bit 0 vaut 1 si les piles de la HP-48 sont très faibles (VLB),
- le bit 1 vaut 1 si les piles de la HP-48 sont faibles (LB),
- le bit 2 vaut 1 si l'alimentation de la carte mémoire se trouvant dans le support de carte n°1 est faible (LBP1),
- le bit 3 vaut 1 si l'alimentation de la carte mémoire se trouvant dans le support de carte n°2 est faible (LBP2).

#00109h	Bits n°0 à 3	Test des piles

On provoque un test de l'alimentation en donnant au bit n°3 en #00109h la valeur 1 (on écrit 1100_2 en #00109h). Le résultat des tests est placé en #00108h. En fin de test, placez 0100_2 en #00109h.

#00109h	Bit n°0	Initialisation (reset)

Ce bit, déjà vu dans le cadre du test des piles, vaut 1 si une initialisation (« reset ») a eu lieu. Il s'agit ici d'un reset provoqué en insérant un trombone déplié dans l'orifice prévu à cet effet sous l'un des patins.

#0010Bh	Bits n°0 à 3	Allumage des indicateurs

De l'état de ces bits dépend l'allumage des indicateurs en haut de l'écran. Le bit 0 allume l'indicateur ↰ quand il vaut 1, de même, le bit 1 allume ↱, le bit 2 allume α et le bit 3 allume l'indicateur de sonnerie.

#0010Ch	Bits n°0, 1 & 3	Allumage des indicateurs
Ces bits contrôlent l'allumage des indicateurs. Le bit 0 allume X lorsqu'il vaut 1 et le bit 1 allume l'indicateur de transmission. Le bit 3 en #10Ch autorise l'allumage des indicateurs ou les éteint tous quand il vaut 0.		

#0010Dh	Bits n°0 à 2	Vitesse sur le port série
La vitesse des échanges de données au moyen du port série se mesure en bits par seconde (bps ou $bit.s^{-1}$) et non en bauds comme cela est pourtant écrit dans de nombreux ouvrages sérieux ! Les bits 0 à 2 en #10Dh déterminent la vitesse de transmission. Ci-dessous, *a* représente le bit n°2, *b*, le bit n°1 et *c*, le bit n°0. On a : • si #*abc*h = #000h, la vitesse de transmission en 1200 bps, • si #*abc*h = #010h, la vitesse de transmission en 2400 bps, • si #*abc*h = #100h, la vitesse de transmission en 4800 bps, • si #*abc*h = #110h, la vitesse de transmission en 9600 bps.		

#0010Eh	Bits n°0 à 3	Contrôle de la détection des cartes
Voici un quartet dangereux ! Son contenu détermine la détection des données sur les ports 1 et 2. Modifier ce quartet bloque la machine, le bit n°0 correspond à EHINTR (« Enable Hard Interruptions »).		

#0010Fh	Bits n°0 à 3	Informations sur les ports
Chacun des quatre bits de ce quartet fournit une information relative au contenu d'un port (port 1 ou 2) et aux possibilités d'écriture sur celui-ci. En particulier : • le bit n°0 (P1) vaut 1 s'il y a une carte dans le port 1, • le bit n°1 (P2) vaut 1 s'il y a une carte dans le port 2 • le bit n°2 (WR1) vaut 1 s'il est possible d'écrire sur la carte se trouvant dans le port 1, • le bit n°3 (WR2) vaut 1 s'il est possible d'écrire sur la carte se trouvant dans le port 2.		

#00110h	Bits n°0 à 3	Transmission série
Chacun des quatre bits de ce quartet vaut 1 dans les conditions suivantes : • le bit n°0, si les interruptions sont permises pendant le transfert, • le bit n°1, si une interruption peut avoir lieu lorsque la mémoire tampon de réception est saturée, • le bit n°2, si une interruption peut avoir lieu lorsque la mémoire tampon d'émission est vide, • le bit n°3, si le port série (câble) est « ouvert ».		

#00111h	Bits n°0 à 2	Réception série

Les bits 0 à 2 du quartet se trouvant en #00111h sont représentatifs des conditions de réception au moyen du port série :

- le bit 0 vaut 1 s'il y a un octet non traité dans la mémoire tampon de réception. Ce bit passe automatiquement à 1 dès qu'une réception d'octet a lieu, il repasse à 0 quand la donnée reçue est traitée,
- le bit 1 vaut 1 si une réception est en cours,
- le bit 2 vaut 1 si une erreur de transmission a été détectée. La remise à zéro de ce bit dépend du contenu de #00113h.

#00112h	Bits n°0, 1 et 3	Emission série

Les bits 0, 1et 3 du quartet se trouvant en #00112h sont représentatifs des conditions d'émission au moyen du port série :

- le bit 0 vaut 1 s'il y a un octet non encore émis dans la mémoire tampon d'émission. Ce bit passe automatiquement à 1 dès qu'une donnée est placée dans la mémoire tampon d'émission, il repasse à 0 quand la donnée a été émise,
- le bit 1 vaut 1 si une émission est en cours,
- en mettant le bit 3 à 1, la ligne de communication est mise en état « espace ».

#00113h	Bits n°0 à 3	Erreurs de transmission

En donnant au quartet #00113h une valeur autre que 0000_2, on désarme le bit n°2 en #00111h (ce bit indique une erreur lors d'une réception via le port série). Il s'agit d'un CRER (« clear error »).

#00114h à #00115h	Bits n°0 à 3	Tampon de réception série

Un octet reçu se place sur les quartets RBR (« receive buffer register») des adresses #00114h et #00115h. Il suffit de lire en #00114h pour recevoir un caractère. Cette lecture désarme le bit n°0 en #00111h.

#00116h à #00117h	Bits n°0 à 3	Tampon d'émission série

Un octet à émettre est à copier sur les quartets TBR (« transmitter buffer register») des adresses #00116h et #00117h. Il suffit d'écrire en #00116h pour émettre un caractère (à condition que le bit n°0 en #00112h vaille 0).

#00118h à #00119h	Bits n°0 à 3	Identité de l'interruption

Le gestionnaire d'interruptions utilise l'octet en #00118h car ce dernier identifie les interruptions. Notons que le bit de poids fort du quartet en #00119h détecte les interruptions (masquables) liées au clavier.

#0011Ah	Bits n°0 à 3	Informations réception infrarouge

L'état des bits du quartet situé en #0011Ah apporte différentes informations relatives à la réception au moyen de la diode infrarouge :

- le bit 0 (IRIMP) vaut 1 pour signaler qu'une réception via l'infrarouge a provoqué une interruption,
- le bit 1 vaut 1 si les interruptions en faveur de l'infrarouge sont autorisées,
- le bit 2 (MODE) vaut 1 si les octets « tampons » en #00114h et en #00116h sont utilisés (il vaut 0 en cas de transmission par câble),
- le bit 3 (IRRECV) vaut 1 si une réception infrarouge est en cours.

#0011Bh	Bits ?	Horloge de transmission IR

Une horloge d'une fréquence de 16 Hz fonctionne pendant les transmissions infrarouge. Cette horloge est liée au quartet situé à l'adresse #0011Bh de la RAM I/O.

#0011Ch	Bits n°0 à 3	Informations émission infrarouge

L'état des bits du quartet situé en #0011Ch apporte différentes informations relatives à l'émission au moyen de la diode infrarouge :

- le bit 0 (IBUFF) vaut 1 pour signaler que le tampon infrarouge (voir en #0011Dh) est saturé. Il peut s'agir d'une saturation du canal de données,
- le bit 1 (IR BUSY) vaut 1 si une transmission est en cours,
- le bit 2 vaut 1 si les interruptions « tampon vide » sont autorisées,
- le bit 3 (IRSEND) vaut 1 lorsque la transmission est en cours (mise en service de la diode infrarouge émettrice).

#0011Dh	Bit n°0	Bit à émettre par infrarouge

Le bit n°0 (IRE, « IR Event ») contient le bit à émettre au moyen de la diode infrarouge émettrice. Selon certaines sources, le quartet entier peut être utilisé comme mémoire tampon d'émission IR (buffer IR).

#0011Eh	Bits n°0 à 3	Sauvegarde de D1

Lors du traitement d'une interruption, le gestionnaire d'interruptions sauvegarde le quartet n°#0h (quartet de poids faible) du registre pointeur D1 dans le quartet d'adresse #0011Eh.

#0011Fh	Bits n°0 à 3	Adresse de la RAM

Ce quartet détermine l'adresse de base de la RAM interne. Il vaut #8h si la RAM commence en #80000h, #Ch si elle commence en #C0000h (#7h pour une RAM en #70000h sur S/SX). Utilité : accès à la ROM masquée.

#00120h à #00124h	Bits n°0 à 3	Adresse de l'écran

Le contenu de l'écran est sauvegardé sous la forme d'un objet graphique (GROB) à une adresse enregistrée sur 5 quartets à partir de #00120h. La lecture de #00120h (sur 5 quartets) ne fournit pas de résultat exploitable. Vous devez donc vous contenter d'écrire en #00120h afin de fixer l'adresse de la « bitmap » écran. Cependant, il est possible de sauvegarder l'adresse de la bitmap écran (en #8068Dh sur G/GX). Le quartet de poids faible de l'adresse n'est pas considéré. Vous devrez donc ajuster votre image en utilisant la marge à gauche qui vaudra 0 à 3 si le GROB se trouve à une adresse paire, entre 4 et 7, s'il s'agit d'une adresse impaire.

#00125h à #00127h	Bits n°0 à 3	Marge à droite de l'écran

Contrairement à la marge à gauche de l'écran (voir en #00100h) qui est exprimée en pixels, la marge à droite est exprimée en quartets et est codée sur trois quartets. Le bit n°3 en #00127h détermine le signe de la marge à droite (elle peut être négative). Le bit n°3 en #00125h n'est pas considéré. Vous ne pouvez qu'écrire en #00125h à #00127h, la lecture à ces adresses ne fournit pas de résultat exploitable, de plus, la marge à droite doit être paire. Le défilement rapide d'un GROB s'obtient en se contentant de modifier l'adresse du GROB (adresse de l'écran) et les marges à gauche et à droite.

#00128h à #00129h	Bits n°0 à 3	Balayage et hauteur menu

Les 4 bits en #00128h et les bits 0 et 1 en #00129h contrôlent :

- en lecture, le numéro de la ligne de l'écran en cours de rafraîchissement (balayage vertical),
- en écriture, la hauteur de la barre des menus.

L'affichage est divisé en deux parties : d'une part, la barre des menus, d'autre part, la partie supérieure de l'écran (représentation de la pile, etc.). En écrivant 111111_2 (#3Fh=63) en #00128h on fait disparaître la barre des menus. Les bits n°2 et 3 en #00128h ne sont pas paramétrables par le programmeur et ne doivent pas être modifiés même après récupération de la hauteur du menu depuis la sauvegarde. Pour modifier la position du menu, on lira en #8069Ah (adresse de la sauvegarde de la position du menu sur G/GX) l'actuelle position du menu. La nouvelle position du menu devra être copiée dans #00128h et dans #8069Ah (sur G/GX). On obtient une information de type différent quand, au lieu d'écrire, on lit 6 bits à partir de #00128h. Il s'agit du numéro de la ligne de l'écran en cours de rafraîchissement (VSYNC). Ce numéro a pour valeur maximale le nombre de lignes de la mémoire « bitmap écran » et est limité à 111111_2=#3Fh=63. VSYNC est décrémenté par pas de 1 à chaque fois qu'une ligne est rafraîchie. VSYNC vaut donc 0 lorsque la dernière ligne de l'écran est en cours de redéfinition. Le rafraîchissement d'une ligne de l'écran dure 1/4096 ème de seconde.

#0012Eh à #0012Fh	Bits n°0 à 3	Horloges (contrôle)

En #12Eh, le bit 1 vaut 1 si le passage par zéro de l'horloge 1 (timer 1) doit déclencher une interruption. Le bit 2 vaut 1 si le passage par zéro provoque la sortie du mode « basse consommation » (voir SHUTDN). Le bit 3 sert au gestionnaire d'interruptions, il vaut 1 si l'horloge 1 demande une modification. De même, on trouve en #12Fh quatre bits de contrôle de l'horloge 2 (timer 2). Le bit 0 en #12Fh provoque l'interruptions des oscillations de l'horloge 2 s'il vaut 0, la machine est alors arrêtée. Les bits 1, 2 et 3 en #12Fh fonctionnent comme leurs homologues en #12Eh mais travaillent avec l'horloge 2. Voir aussi en #00137h.

#00130h à #00134h	Bits n°0 à 3	Adresse de la bitmap menu

L'écran est divisé en deux (menus et partie supérieure). La mémoire écran des menus (bitmap menu) peut être écrite en #00130h sur 5 quartets. Cette adresse est sauvegardée en #80695h (HP-48 G/GX).

#00137h à #0013Fh	Bits n°0 à 3	Horloges 1 et 2

L'horloge 1 occupe 1 quartet en #00137h et sert à la scrutation du clavier, en particulier, cette horloge est chargée de détecter deux pressions successives sur une même touche. L'horloge 2 contient une « heure » codée sur 8 quartets en $1/2^{13}$ seconde. La valeur stockée décroît de #FFFFFFFFh à #00000000h. Quelques remarques s'imposent :

- le passage par #00000000h de cette horloge déclenche une interruption,
- cette horloge n'est pas celle utilisée pour la synchronisation du bus mais elle est néanmoins utilisée pour cadencer certains échanges de données,
- l'étendue des valeurs normalement utilisées correspond au nombre de $1/2^{13}$ de seconde ($1/2^{13}$=1/8192) qu'il y a dans une heure. L'horloge 2 décroît donc en général de #1C20000h à #0h,
- si l'heure est affichée, l'horloge travaille à partir de secondes et non plus de $1/2^{13}$ de seconde,
- si l'heure n'est pas affichée et qu'une sonnerie programmée doit être déclenchée durant l'heure en cours, la valeur stockée dans horloge 2 correspond au nombre de $1/2^{13}$ de seconde qui nous séparent encore de l'instant de déclenchement de la sonnerie.

24

La mémoire vive réservée au système

Une partie de la mémoire vive est réservée au système qui y place des informations relatives au fonctionnement de la HP-48. Cette mémoire vive réservée est une portion de la mémoire vive normale (plage d'adresse débutant en #80000h sur HP-48 G/GX et en #70000h sur HP-48 S/SX). La mémoire vive réservée au système ne constitue donc pas un module de mémoire indépendant et se résume à quelques quartets « réservés » au sein de la mémoire vive dont se sert l'utilisateur. Nous commencerons par dresser la liste des champs de la RAM système (dans l'ordre des adresses croissantes) puis, dans la seconde partie de ce chapitre et pour certains champs, nous apporterons quelques précisions complémentaires.

1 Liste des champs

Les tableaux ont la structure suivante :

- la première colonne contient des adresses en RAM interne valables sur HP-48 G/GX,
- la deuxième colonne contient des adresses en RAM interne valables sur HP-48 S/SX,
- la troisième contient les tailles des champs en quartets,
- la quatrième contient le nom du champ suivi d'un commentaire ou du symbole « • » qui vous renvoie à la deuxième partie de ce chapitre pour de plus amples informations.

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#80000h	#70000h	5	**Vérification de l'intégrité de la RAM** •
#80005h	-	5	**Taille de la mémoire vive interne** •
#8000Ah	-	5	**Taille de la mémoire vive interne (bis)**
#8000Fh	-	1	**Fin de la RAM** •
-	#70009h	1	**Inhibition de l'arrêt système** Lorsque le bit n°3 de ce quartet vaut 0, l'arrêt-système ([ON]-[C]) et le test ([ON]-[D]) sont inhibés
#80010h	#7000Ah	18	**Dernier démarrage à chaud (1)** •
#80022h	#7001Ch	18	**Avant-dernier démarrage à chaud (2)** •
#80034h	#7002Eh	18	**Avant-avant-dernier démarrage... (3)** •
#80046h	#70040h	18	**Avant-avant-avant-dernier... (4)** •
#80058h	#70052h	13	**Décalage d'horloge (date et heure)** Ce décalage correspond au nombre de 1/8192e de seconde depuis la date initiale (1/1/00)
#80065h	#7005Fh	4	**CRC du décalage d'horloge**
#80069h	#70070h	13	**Heure de l'extinction automatique** Ces 13 quartets sont mis à zéro pendant l'exécution d'un programme utilisateur.
#80076h	-	1	**Détection de l'extinction automatique** Ce quartet est mis à zéro si la machine s'éteint automatiquement
#80077h	-	1	**Drapeau « piles faibles »** Ce quartet vaut zéro en cas de passage en mode « coma »
#80078h	#70072h	13	**Date de début du test automatique** Date et heure sont précisées (décalage)
#80085h	#7007Fh	13	**Date d'échec du test automatique** Date et heure sont précisées (décalage)
#80092h	#7008Ch	44	**Tampon pour les messages du test** Mémoire tampon (buffer) pour les messages du test automatique
#800BEh	#700B8h	22	**Tampon pour « Stop watch »** Mémoire tampon (buffer)
#800D4h	#700CEh	13	**Horloge pour « Stop watch »**
#800E1h	#700DBh	1	**Configuration des ports** •
#800E2h	#700DCh	4	**CRC de la carte en port 1** Somme de contrôle classique codée sur 4 quartets
#800E6h	-	1	**Sauvegarde d'inhibition de touches** Sauvegarde l'inhibition de [ON]-[?]

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#800E7h	-	1	**Sauvegarde du drapeau ST(15)** Sauvegarde de l'état du bit 15 du registre ST. Ce bit d'état autorise ou interdit les interruptions.
#800E8h	#700E7h	3	**Etat de la RAM (avant reconfiguration)** Sauvegarde de l'état de la RAM avant une opération de reconfiguration de la mémoire (accès à une zone masquée)
#800EBh	#700EAh	10	**Sauvegarde pour reconfiguration** Sauvegarde utilisée pendant un accès à une zone masquée de la mémoire
#800F5h	#700F1h	256	**Boîte aux lettres (mailbox)** Mémoire tampon (boîte aux lettres) servant pour les transferts entre une zone masquée de la mémoire et la mémoire vive réservée aux objets temporaires
#8030Eh	#701F1h	11	**Nom de XLIB (GraphPrtHook)**
#80319h	#701FCh	512	**Mémoire tampon circulaire port série** Tampon (buffer) capable de contenir 256 octets
#80519h	#703FCh	2	**Nombre de caractères dans le tampon** Concerne le tampon en #80319h
#8051Bh	#703FEh	1	**Drapeau « tampon saturé »** Le bit n°3 de ce quartet vaut 1 si le tampon en #80319h est saturé
#8051Ch	#703FFh	2	**Pointeur du tampon port série** •
#8051Eh	#70401h	1	**Paramétrage du port série**
#8051Fh	#70402h	1	**Paramétrage du port série (bis)**
#80520h	#70403h	1	**Mode courant de contrôle de parité** •
#80521h	#70404h	2	**Attente avant erreur de transmission** •
#80523h	#70406h	1	**Sauvegarde de #00110h (RAM I/O)**
#80524h	#70407h	1	**Configuration des ports 1 et 2** •
#8052Bh	#70428h	4	**CRC de la configuration détaillée** CRC des 22 quartets commençant en #7042Ch (S/SX) ou #8052Fh (G/GX)
#8052Fh	#7042Ch	1	**Détails de configuration du port 1** •
#80530h	#7042Dh	1 ou 5	**Taille de la carte en port 1** • Occupe 1 quartet sur SX et 5 sur GX
-	#70432h	5	**Adresse de début de la carte du port 1**
#80531h	#70437h	1	**Détails de configuration du port 2** •
#80532h	#70438h	1 ou 5	**Taille de la carte en port 2** • Occupe 1 quartet sur SX et 5 sur GX

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#80533h	-	1	**Nombre de partitions du port 2**
-	#7043Dh	5	**Adresse de début de la carte en port 2**
#80535h	-	1	**Taille des cartes •**
#80536h	#7044Dh	5	**Prochain objet (port 0) •**
#8053Bh	#70452h	5	**Prochain objet (port 1) •**
#80540h	#70457h	5	**Prochain objet (port 2) •**
#80545h	-	5	**Prochain objet (port 3) •**
#8054Ah	-	5	**Prochain objet (port 4) •**
#8054Fh	-	5	**Prochain objet (port 5) •**
#80554h	-	5	**Prochain objet (port 6) •**
#80559h	-	5	**Prochain objet (port 7) •**
#8055Eh	-	5	**Prochain objet (port 8) •**
#80563h	-	5	**Prochain objet (port 9) •**
#80568h	-	5	**Prochain objet (port 10) •**
#8056Dh	-	5	**Prochain objet (port 11) •**
#80572h	-	5	**Prochain objet (port 12) •**
#80577h	-	5	**Prochain objet (port 13) •**
#8057Ch	-	5	**Prochain objet (port 14) •**
#80581h	-	5	**Prochain objet (port 15) •**
#80586h	-	5	**Prochain objet (port 16) •**
#8058Bh	-	5	**Prochain objet (port 17) •**
#80590h	-	5	**Prochain objet (port 18) •**
#80595h	-	5	**Prochain objet (port 19) •**
#8059Ah	-	5	**Prochain objet (port 20) •**
#8059Fh	-	5	**Prochain objet (port 21) •**
#805A4h	-	5	**Prochain objet (port 22) •**
#805A9h	-	5	**Prochain objet (port 23) •**
#805AEh	-	5	**Prochain objet (port 24) •**
#805B3h	-	5	**Prochain objet (port 25) •**
#805B8h	-	5	**Prochain objet (port 26) •**
#805BDh	-	5	**Prochain objet (port 27) •**
#805C2h	-	5	**Prochain objet (port 28) •**
#805C7h	-	5	**Prochain objet (port 29) •**
#805CCh	-	5	**Prochain objet (port 30) •**
#805D1h	-	5	**Prochain objet (port 31) •**
#805D6h	-	5	**Prochain objet (port 32) •**
#805DBh	-	5	**Prochain objet (port 33) •**
#805DCh	#7045Dh	4	**Sauvegarde des bits 0, 1, 2 et 3 de D1** Cette sauvegarde et les 11 suivantes sont utilisées pendant le traitement d'une interruption
#805E0h	#70461h	11	**Sauvegarde des bits 5 à 15 de C**

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#805EBh	#7046Ch	5	**Sauvegarde de P, de SB et de la retenue**
#805F0h	#70471h	5	**Sauvegarde du champ A de C**
#805F5h	#70476h	16	**Sauvegarde du registre A**
#80605h	#70486h	3	**Sauvegarde des bits ST n°0 à 11**
#80608h	#70489h	16	**Sauvegarde du registre B**
#80618h	#70499h	16	**Sauvegarde du registre D**
#80628h	#704A9h	16	**Sauvegarde du registre R0**
#80638h	#704B9h	5	**Sauvegarde du registre PC**
#8063Dh	#704BEh	5	**Sauvegarde du registre pointeur D0**
#80642h	#704C3h	3	**Sauvegarde « OR » pour OUT et IN**
#80645h	#704C6h	5	**Adresse de l'écran (début)** Adresse de la bitmap à afficher lors du rafraîchissement de l'écran
#8064Ah	#704CBh	5	**Adresse de l'écran (fin)** Adresse du dernier quartet à afficher lors du rafraîchissement de l'écran
#8064Fh	#704D0h	3	**Sauvegarde (interruptions)** Sauvegarde de l'historique du gestionnaire d'interruptions
#80652h	#704D3h	2	**Contrôle sauvegarde (interruptions)** Quartets de contrôle de la sauvegarde de l'historique du gestionnaire d'interruptions
#80655h	#704D6h	3	**Fréquence de l'horloge** Fréquence exprimée en unités de 16 Hz
#8065Ah	-	1	**Interdiction des séquences [ON]-[...] •**
#8065Bh	#704DCh	1	**Désactivation du tampon du clavier •**
#8065Ch	#704DDh	13	**Gestion des pressions simultanées** Cette table (13 quartets) sert à traiter les pressions simultanées sur plusieurs touches
#80669h	#704EAh	1	**Première touche à traiter •**
#8066Ah	#704EBh	1	**Stockage de la prochaine touche •**
#8066Bh	#704ECh	32	**Mémoire tampon de clavier •**
#8068Bh	#7050Ch	2	**Dernière touche traitée** Connaître la dernière touche traitée, et donc, ôtée du tampon de clavier, permet de détecter les appuis successifs sur une même touche
#8068Dh	#7050Eh	5	**Adresse de la bitmap écran** Sauvegarde de l'adresse du GROB écran qui se trouve normalement en #120h

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#80692h	#70513h	3	**Marge à droite de l'écran** Sauvegarde du contenu de #125h
#80695h	#70516h	5	**Adresse de la bitmap menu** Sauvegarde de l'adresse du GROB écran qui se trouve normalement en #130h
#8069Ah	#7051Bh	2	**Suivi du rafraîchissement de l'écran** •
#8069Ch	-	5	**Sauvegarde pile interne (niveau 0)** Cette adresse ainsi que les 5 suivantes correspondent chacune à la sauvegarde d'un niveau de la pile interne. Celle-ci est modifiée par une copie de l'écran sur l'imprimante (IR) ou via le port série
#806A1h	-	5	**Sauvegarde pile interne (niveau 1)**
#806A6h	-	5	**Sauvegarde pile interne (niveau 2)**
#806ABh	-	5	**Sauvegarde pile interne (niveau 3)**
#806B0h	-	5	**Sauvegarde pile interne (niveau 4)**
#806B5h	-	5	**Sauvegarde pile interne (niveau 5)**
#806BAh	-	5	**Sauvegarde du champ A de R2**
#806BFh	-	1	**Sauvegarde du champ S de R2**
#806C0h	-	5	**Sauvegarde du champ A de R1**
#806C5h	#70546h	1	**Sauvegarde du bit 3 du quartet #100h** Cette sauvegarde ainsi que les 3 suivantes servent lorsque l'on change de GROB courant (changement de l'objet graphique à afficher). Il s'agit d'une sauvegarde de la RAM I/O
#806C6h	#70547h	2	**Sauvegarde de #128h**
#806C8h	#70549h	3	**Sauvegarde de #125h**
#806CBh	#7054Ch	5	**Sauvegarde des bits 0, 1 et 2 de #100h**
#806D0h	#70551h	5	**Adresse du GROB menu** Codage sur 5 quartets de l'adresse de l'objet graphique chargé de l'affichage des menus. Ce GROB a 131 × 8 pour dimensions
#806D5h	#70556h	5	**Adresse du GROB pile** Codage sur 5 quartets de l'adresse de l'objet graphique chargé de l'affichage de la pile. Ce GROB a 131 × 56 pour dimensions
#806DAh	#7055Bh	5	**Adresse du GROB courant (à afficher)** Il peut s'agir du GROB décrivant la pile ou du GROB PICT (image graphique)

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#806DFh	#70560h	5	**Adresse du GROB inactif** Il existe deux objets graphiques, STACK pour la pile et PICT pour l'image graphique. On stocke ici l'adresse de l'objet graphique actuellement non affiché
#806E4h	#70565h	5	**Adresse de l'objet graphique PICT** Ce GROB est affiché à l'aide de GRAPH sur S/SX ou de PICTURE sur G/GX
#806E9h	#7056Ah	5	**Adresse des objets temporaires** •
#806EEh	#7056Fh	5	**Adresse des retours RPL système** Adresse de la pile des retours RPL système à ne pas confondre avec RSTK
#806F3h	#70574h	5	**Adresse du début de la mémoire vive** Il s'agit de l'adresse du début de la mémoire vive. Cette adresse désigne le premier quartet se trouvant juste après la dernière adresse mémorisée dans la pile des retours RPL système
#806F8h	#70579h	5	**Adresse de la fin de la mémoire vive** Cette adresse correspond au dernier niveau inséré dans la pile utilisateur. Cette adresse est donc celle du contenu du niveau 1 de la pile
#806FDh	#7057Eh	5	**Adresse de la ligne de commande** Il s'agit de l'adresse du premier caractère en cours d'édition dans la ligne de commande. Les caractères sont stockés sous la forme de leurs codes ASCII
#80702h	#70583h	5	**Adresse de l'environnement temporaire** •
#8070Ch	#7058Dh	5	**Adresse de la table des menus** •
#80711h	#70592h	5	**Adresse du répertoire racine (HOME)**
#80716h	#70597h	5	**Adresse du premier objet du port 0**
#8071Bh	#7059Ch	5	**Adresse du répertoire courant**
#80720h	#705A1h	5	**Adresse du répertoire isolé** Le répertoire dont l'adresse est ici copiée se trouve isolé du reste de l'arborescence. Cela a des conséquences au niveau de la « visibilité » des variables
#80725h	#705A6h	5	**Adresse de la redéfinition des touches** Il s'agit de l'adresse de la liste des touches redéfinies par l'utilisateur

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#8072Ah	#705ABh	5	**Adresse des sonneries** Il s'agit de l'adresse de la liste des sonneries programmées par l'utilisateur
#8072Fh	#705B0h	5	**Sauvegarde du registre pointeur D0**
#80739h	#705BAh	5	**Adresse de l'argument 1** Ce champ et les quatre suivants stockent les adresses des arguments de la dernière instruction exécutée. Ces adresses correspondent à des adresses de niveaux de la pile (niveaux 1 à 5 pour les arguments 1 à 5)
#8073Eh	#705BFh	5	**Adresse de l'argument 2**
#80743h	#705C4h	5	**Adresse de l'argument 3**
#80748h	#705C9h	5	**Adresse de l'argument 4**
#8074Dh	#705CEh	5	**Adresse de l'argument 5**
#80752h	-	5	**Adresse d'un entier hexa long** Il s'agit d'un entier hexadécimal de 184 chiffres servant - selon certaines sources - à la compilation de la ligne de commande
#80757h	-	5	**Adresse d'un entier binaire** Cet entier binaire est utilisé par l'application Equation Writer
#8075Ch	#705DDh	5	**Adresse dernière ligne de commande** La HP-48 mémorise les 4 dernières lignes de commande utilisées. Les adresses où sont stockées ces lignes de commandes sont codées par ce champ et par les 3 suivants
#80761h	#705E2h	5	**Adresse ligne de commande (2)**
#80766h	#705E7h	5	**Adresse ligne de commande (3)**
#8076Bh	#705FCh	5	**Adresse ligne de commande (4)**
#80775h	#705F6h	5	**Adresse de la dernière erreur Kermit** Il s'agit de l'adresse du texte commentant l'erreur
#8077Ah	-	5	**Adresse utilisée par Equation Writer**
#8077Fh	#70600h	5	**Adresse du dernier message d'erreur** Il s'agit uniquement des messages d'erreur définis par l'utilisateur
#80784h	#70605h	5	**Adresse objet à traiter avant une touche** Adresse de l'objet chargé de mettre à jour l'affichage avant de traiter une pression sur une touche

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#80789h	#7060Ah	5	**Adresse redéfinition des touches** Adresse d'un objet où sont mémorisées les touches redéfinies par l'environnement (Equation Writer, MatrixWriter...)
#8078Eh	#7060Fh	5	**Adresse d'un objet de sortie** L'évaluation de l'objet de sortie signale qu'il faut - ou non - sortir de l'environnement courant (Equation, Matrix, etc.)
#80793h	#70614h	5	**Adresse en cas d'erreur** Dans un environnement particulier (Matrix, Equation, etc.), ce champ contient l'adresse de l'objet qui devra être évalué si le traitement d'une pression sur une touche conduit à une erreur
#8079Dh	#7061Eh	5	**Adresse du menu courant**
#807A2h	#70623h	5	**Adresse du précédent menu**
#807ACh	#7062Dh	5	**Adresse pour le défilement des menus** Ce champ contient l'adresse d'un objet à évaluer en cas d'utilisation de NXT ou de PREV (défilement d'un menu)
#807B6h	#70637h	5	**Traitement normal d'une touche** Ce champ contient l'adresse d'une routine qui s'exécute lorsqu'une touche du clavier est pressée sans appui préalable sur [↰] ni sur [↱]
#807BBh	#7063Ch	5	**Traitement d'une touche après [↰]** Ce champ contient l'adresse d'une routine qui s'exécute lorsqu'une touche du clavier est pressée à la suite d'une pression sur [↰]
#807C0h	#70641h	5	**Traitement d'une touche après [↱]** Ce champ contient l'adresse d'une routine qui s'exécute lorsqu'une touche du clavier est pressée à la suite d'une pression sur [↱]
#807C5h	#70646h	5	**Adresse du traitement (Re-)View** Champ contenant l'adresse de l'objet à évaluer après lorsque View (G/GX) ou Review (S/SX) sont sollicitées
#807CAh	-	5	**Adresse du précédent répertoire racine** Adresse où est stocké le précédent répertoire « maître »

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#807CFh	#70650h	5	**Adresse routine si changement de menu** Ce champ contient l'adresse du programme à exécuter à chaque changement de menu
#807D4h	#70655h	5	**Adresse routine si sortie de menu** Ce champ contient l'adresse du programme à exécuter en cas de sortie d'un menu
#807D9h	#7065Ah	5	**Adresse d'instruction de bibliothèque** Ce champ contient l'adresse de la dernière instruction de bibliothèque exécutée
#807E8h	#70669h	5	**Adresse du dernier quartet de la RAM**
#807EDh	#7066Eh	5	**Mémoire vive disponible** Ce champ contient le nombre de quartets libres divisé par 5
#807F2h	#70673h	5	**Numéro de la prochaine erreur**
#807F7h	#70679h	5	**Nombre de pressions sur [ON]**
#807FCh	#7067Eh	5	**Adresse de la boucle d'attente RPL** Ce champ contient l'adresse de la routine chargée de guetter en permanence vos sollicitations
#80801h	#70683h	1	**Quartet drapeau interne n°1 •**
#80802h	#70684h	1	**Quartet drapeau interne n°2 •**
#80803h	#70685h	1	**Quartet drapeau interne n°3 •**
#80804h	#70686h	1	**Quartet drapeau interne n°4 •**
#80805h	#70687h	1	**Quartet drapeau interne n°5 •**
#80806h	#70688h	1	**Quartet drapeau interne n°6 •**
#80807h	#70689h	1	**Quartet drapeau interne n°7 •**
#80808h	#7068Ah	1	**Quartet drapeau interne n°8 •**
#80809h	#7068Bh	1	**Quartet drapeau interne n°9 •**
#8080Ah	#7068Ch	1	**Quartet drapeau interne n°10 •**
#8080Bh	#7068Dh	1	**Quartet drapeau interne n°11 •**
#8080Ch	#7068Eh	1	**Quartet drapeau interne n°12 •**
#8080Dh	#7068Fh	1	**Quartet drapeau interne n°13**
#8080Eh	#70690h	1	**Quartet drapeau interne n°14**
#8080Fh	#70691h	1	**Quartet drapeau interne n°15**
#80810h	#70692h	1	**Quartet drapeau interne n°16**
#80811h	#70693h	1	**Quartet drapeau interne n°17**
#80812h	#70694h	1	**Quartet drapeau interne n°18**
#80813h	#70695h	1	**Quartet drapeau interne n°19**
#80814h	#70696h	1	**Quartet drapeau interne n°20**

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#80815h	-	2	**Nombre d'environnements emboîtés** Il s'agit ici d'environnements personnalisés
#80817h	-	1	**Nombre de lignes pour le niveau 1** Nombre de lignes de texte utilisées pour visualiser l'objet du niveau 1 de la pile
#80818h	-	5	**Niveau visualisé de la pile** Ce niveau est édité dans la ligne de commande
#8081Dh	#7069Fh	5	**Taille de la pile RPL en quartets** Espace consacré en mémoire à la pile
#80822h	#706A4h	15	**Réel du générateur aléatoire** Corps du réel sur lequel se fonde le générateur de nombres aléatoires
#80831h	#706B3h	1	**Sauvegarde drapeau affichage horloge** Ce drapeau permet de savoir si l'horloge doit ou non être affichée
#80832h	#706B4h	1	**Drapeau sonnerie à déclencher** Ce quartet vaut #Fh si une sonnerie doit être déclenchée, il vaut #0h sinon
#80833h	#706B5h	1	**Drapeau sonnerie non validée** Ce quartet vaut #Fh si une sonnerie a été déclenchée puis n'a pas été confirmée par l'utilisateur. Il vaut #0h sinon
#80834h	#706B6h	1	**Drapeau sonnerie à répéter** Ce quartet vaut #Fh après saisie de [ON]-[4]. Il vaut #0h si la sonnerie est à répéter
#80835h	#706B7h	1	**Drapeau d'inhibition des sonneries** Ce quartet interdit les sonneries quand il vaut #Fh
#80836h	#706B8h	1	**Numéro de la prochaine sonnerie**
#8083Bh	-	1	**Numéro de la dernière sonnerie** Numéro de la dernière sonnerie non validée lors de son déclenchement
#80840h	#706C2h	1	**Sonneries à venir et état des piles** •
#80841h	#706C3h	2	**Etat des témoins** •
#80843h	#706C5h	16	**Drapeaux (flags) du système** •
#80853h	#706D5h	16	**Drapeaux (flags) de l'utilisateur** •
#80863h	-	2	**Compteur de « profondeur »** Utilisé pour la décompilation
#80865h	-	5	**Décalage vers le 1er caractère visible** Caractère visible de la ligne de commande

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#8086Ah	#706ECh	5	**Nombre de CR en ligne de commande** Nombre de retours à la ligne dans la ligne de commande en cours (nombre de lignes moins 1)
#80874h	-	5	**Première ligne de la ligne de commande** Numéro de la première ligne affichée en ligne de commande
#8087Ah	-	1	**Numéro de sauvegarde disponible** Numéro du premier numéro de niveau de sauvegarde libre pour la sauvegarde de la ligne de commande. Voir en #8075Ch, #80761h, #80766h et #8076Bh pour la sauvegarde des quatre dernières lignes de commande
#8087Bh	-	1	**Nombre d'éléments sauvegardés** Nombre d'éléments dont la sauvegarde a été provoquée par l'exécution de la dernière ligne de commande
#8087Ch	-	1	**Drapeau sauvegarde d'éléments** Ce quartet est différent de #0h si l'exécution de la dernière de commande s'est accompagnée de la sauvegarde d'un nombre d'éléments supérieur à 3
#8087Dh	#706FFh	5	**Numéro du dernier message d'erreur**
#80882h	#70704h	5	**Position du point d'insertion** Position du curseur (représentant le point d'insertion des nouveaux caractères) dans la ligne de commande
#80887h	#70709h	5	**Position verticale du point d'insertion** Numéro de la ligne où se trouve le point d'insertion (curseur)
#8088Ch	#7070Eh	5	**Position horizontale du point d'insertion** Numéro de la colonne où se trouve le point d'insertion (curseur)
#8088Eh	#70710h	1	**Clignotement du curseur** Le curseur clignote (à un instant donné, seul le curseur ou le caractère qui se trouve au-dessous de lui est affiché). Le curseur est affiché si ce quartet vaut #0h, sinon, c'est le caractère qui se trouve au-dessous qui est affiché
#8088Fh	#70711h	2	**Code ASCII du caractère sous le curseur**

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#80891h	#70713h	40	**GROB du caractère sous le curseur** Objet graphique (matrice 6×10) représentant le caractère placé au-dessous du curseur
#808B9h	-	5	**Abscisse du curseur** Position du curseur (point d'insertion) sur l'écran exprimée en pixels
#808BEh	-	5	**Ordonnée du curseur** Position du curseur (point d'insertion) sur l'écran exprimée en pixels
#808C3h	-	2	**Numéro du menu courant**
#808C5h	-	5	**Premier article visible du menu courant** Ce numéro varie par pas de 6
#808CAh	-	2	**Numéro du précédent menu**
#808CCh	-	1	**Compteur utilisé lors des interruptions**
#808CDh	-	1	**Compteur d'espaces** Compte les espaces dans la ligne de commande
#808CEh	-	5	**Sauvegarde d'un retour RPL** Pendant l'opération de suppression des variables devenues inutiles (Garbage collection), on sauvegarde sur ce quartet un niveau de la pile des retours RPL
#808D3h	-	5	**Adresse des objets temporaires** Sauvegarde de l'adresse déjà précisée en #80702h
#808D8h	-	10	**Coordonnées du curseur PICT**
#808E2h	-	1	**Sauvegarde d'un quartet de ST**
#808E3h	-	10	**Sauvegarde d'un niveau de RSTK**
#808EDh	-	1	**Mode Kermit courant**
#808EEh	-	2	**Tampon de décompilation** Cet octet détermine la taille du buffer (mémoire tampon) utilisé pour la décompilation des objets
#808F0h	-	1	**Hauteur (corps) des caractères (EW)** Ce champ et les cinq suivants concernent l'application Equation Writer (EW)
#808F1h	-	1	**Largeur (chasse) des caractères (EW)**
#808F2h	-	3	**Compteur de types de caractères (EW)**
#808F5h	-	3	**Nombre de niveaux logiques (EW)**
#808F8h	-	3	**Longueur de l'objet édité (EW)**
#808FBh	-	1	**Type (arité) du dernier objet édité (EW)**

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#808FCh	-	3	**Développements à gauche (EW)** Nombre de niveaux de développements à gauche. Ce champ et les onze suivants contiennent des paramètres pour l'application Equation Writer (EW)
#808FFh	-	3	**Développements à droite (EW)**
#80902h	-	3	**Niveaux de parenthèses (EW)**
#80905h	-	7	**Paramètre (pile des données ?) (EW)**
#8090Ch	-	5	**Paramètre (liste des touches ?) (EW)**
#80911h	#70793h	1	**Mode « alpha-lock » (EW)** Ce quartet est différent de #0h si le clavier est verrouillé en mode alphabétique et si la ligne de commande est en service
#80912h	#70794h	1	**Mode PRG actif** Ce quartet est différent de #0h si le mode de saisie PRG est actif et si une saisie sous ce mode a débuté
#80913h	-	1	**Paramètre (EW)**
#80914h	#70796h	2	**Nombre de colonnes (Matrix Writer)** Champ aussi utilisé par Equation Writer
#80916h	#70798h	1	**Mode d'introduction des matrices** Champ utilisé par Matrix writer et par Equation Writer
#80917h	-	2	**Compteur de parenthèses (EW)**
#80919h	-	2	**Compteur d'étendue (EW)**
#8091Bh	#7079Dh	1	**Drapeau affichage horloge** L'horloge est affichée si ce quartet est différent de #0h
#8091Ch	#7079Eh	2	**Délai de détection d'erreur (I/O)**
#8091Eh	#707A0h	2	**Délai entre l'impression de deux lignes**
#80920h	#707A2h	2	**Nombre de colonnes à imprimer** Exprimé en nombre de pixels
#80922h	#707A4h	2	**Nombre de lignes à imprimer** Exprimé en nombre de pixels
#80924h	#707A6h	3	**Paramètres d'impression •**
#80947h	#707C9h	5	**Numéro du premier article du menu** Détermine l'article de droite du menu courant
#8094Ch	-	5	**Numéro de l'équation éditée** Si une liste d'équations est utilisée
#80951h	-	11	**Routine de traitement du clavier** Il s'agit d'un nom XLIB

Adresse HP-48 G/GX	Adresse HP-48 S/SX	Taille du champ	Nom du champ
#8095Ch	-	11	**Routine liée à la touche [A]** Ce champ et les 5 suivants contiennent des noms XLIB
#80967h	-	11	**Routine liée à la touche [B]**
#80972h	-	11	**Routine liée à la touche [C]**
#8097Dh	-	11	**Routine liée à la touche [D]**
#80988h	-	11	**Routine liée à la touche [E]**
#80993h	-	11	**Routine liée à la touche [F]**
#8099EH	-	5	**Premier article du menu précédent** Il s'agit du numéro du premier article du menu précédent
#809A3h	#707D9h	3	**Nombre de bibliothèques attachées •**

Faites bien la distinction entre une **donnée mémorisée** dans la RAM système (par exemple, le contenu d'un registre) et une **adresse mémorisée** dans la RAM système. La RAM système stocke simultanément des données (de courtes sauvegardes en général) et des adresses de données qui peuvent se trouver n'importe où en mémoire, et donc, hors de la zone habituellement consacrée à la RAM système.

Remarquez qu'il existe en RAM système (mémoire vive réservée au système) de nombreuses copies de champs de la RAM I/O (mémoire vive des entrées/sorties). Vous trouverez donc en RAM système de nombreux champs intitulés « sauvegarde de #*abc*h » où #*abc*h est compris entre #100h et #13Fh ce qui correspond à la plage d'adresses réservées à la RAM I/O.

Si vous modifiez un champ de la RAM I/O, n'oubliez pas d'apporter les mêmes modifications à son éventuelle sauvegarde en RAM système !

2

Description de certains champs

Reprenant l'ordre de la liste (adresses croissantes), nous allons maintenant commenter les champs suivis d'un « • » dans la liste.

Les adresses des champs sont données pour les HP-48 G/GX à gauche de la barre « / », et pour les HP-48 S/SX à droite de la barre « / ». Les adresses remplacées par « - » n'existent pas pour l'une des familles de HP-48 ou ne nous ont pas été confirmées.

■ #80000h/#70000h • Vérification de l'intégrité de la mémoire vive

Si ces quartets ne correspondent pas à #A5C3Fh, cela signifie que la mémoire vive a été altérée. Le gestionnaire d'interruptions prend alors en charge une vérification du répertoire racine (HOME), le CRC des cartes présentes, etc.

■ #80005h/- • Taille de la mémoire vive interne

La taille est codée en complément à #100000h. Ainsi, les 5 quartets en #80005h valent #C0000h si la mémoire vive a une capacité de 128 ko (cas de la HP-48 GX), ou #F0000h si la mémoire vive a une capacité de 32 ko (cas de la HP-48 G).

■ #8000Fh/- • Fin de la mémoire vive

Ce quartet correspond au quartet de poids fort de l'adresse de la fin de la mémoire vive interne. Ce quartet vaut donc #9h si la RAM a une capacité de 32 ko et #Ch si la RAM a une capacité de 128 ko.

■ #80010h à #80046h/#7000Ah à #70040h • Quatre derniers redémarrages à chaud

A ces quatre adresses sont mémorisés les quatre derniers redémarrages à chaud. Un redémarrage à chaud est stocké sous la forme suivante :

- le premier quartet code le type de l'erreur (voir la table des erreurs WSLOG au chapitre « Codes WSLOG »),
- les 13 quartets suivants codent la date et l'heure de l'erreur en 1/8192e de seconde depuis la date initiale,
- les 4 derniers quartets codent le CRC des 14 premiers quartets.

■ #800E1h/#700DBh • Configuration des ports

Chacun des 4 bits de ce quartet apporte une information sur les cartes présentes en ports 1 et 2.

Bits	Rôles des bits du quartet en #800E1h/#700DBh
0	ce bit vaut 1 si une carte se trouve dans le port 1
1	ce bit vaut 1 si une carte se trouve dans le port 2
2	ce bit vaut 1 si l'écriture est autorisée sur la carte en port 1
3	ce bit vaut 1 si l'écriture est autorisée sur la carte en port 2

■ #8051Ch/#703FFh • Pointeur du tampon port série

En #80319h/#701FCh se trouve une mémoire tampon pour le port série (Buffer RS 232). Ce buffer peut contenir 256 caractères (512 quartets). En #80519h/#703FCh est codé sur 2 quartets l'espace utilisé dans la mémoire tampon. Le nombre de caractères contenus dans cet espace correspond au nombre de caractères non traités restant dans la mémoire tampon. Rappelons qu'un caractère ASCII est codé sur 1 octet, et donc, sur deux quartets. En #8051Ch/#703FFh est codé sur deux quartets un pointeur désignant la position dans le buffer du prochain caractère du buffer RS 232 qui sera traité. Notez le fonctionnement « circulaire » de cette mémoire tampon qui boucle sur elle-même pendant les opérations de lecture/écriture (on revient à #00h après avoir utilisé #FFh).

■ #80520h/#70403h • Mode courant de contrôle de parité

Ce quartet détermine le mode de contrôle de parité courant.

Valeur du quartet	Type de contrôle	Commentaires
#0h	None (aucun)	Pas de contrôle de parité
#8h	Even (paire)	Réception seulement
#9h	Odd (impaire)	Réception seulement
#Ah	Space	Réception seulement
#Bh	Mark	Réception seulement
#Ch	Even (paire)	Emission et réception
#Dh	Odd (impaire)	Emission et réception
#Eh	Space	Emission et réception
#Fh	Mark	Emission et réception

■ #80521h/#70404h • Attente avant erreur de transmission

Ces deux quartets codent le délai après lequel SRECV et XMIT signaleront une erreur de transmission si la tentative de communication n'a pas abouti (erreur « time out »). La valeur stockée correspond au nombre de dixièmes de secondes avant de signaler un « time out » (délai écoulé).

■ #80524h/#70407h • Configuration des ports 1 et 2

Sur HP-48 GX, la structure des données est la suivante :

- statut du port 1 (1 quartet),
- taille de la carte en port 1 (1 quartet),
- statut du port 2 (1 quartet),
- taille de la carte en port 2 (1 quartet),
- nombre de partitions sur le port 2 (2 quartets),
- quartet de configuration.

Les quartets déterminant la taille des cartes (ou d'une partition), valent #0h si le port est vide, 1 si le port contient une carte 32 ko et 2 si le port contient une carte (*n*×) 128 ko. Décomposition des quartets de statut :

- le bit 1 vaut 1 si la carte est fusionnée,
- le bit 2 vaut 1 si la carte est accessible en écriture,
- le bit 3 vaut 1 si la carte est accessible en lecture.

■ #8052Fh/#7042Ch • Détails de configuration du port 1

Trois des bits de ce quartet apportent des précisions sur l'état du port 1 :

- le bit 1 vaut 1 si la carte est fusionnée avec la RAM interne,
- le bit 2 vaut 1 si la carte est accessible en écriture,
- le bit 3 vaut 1 si la carte est accessible en lecture.

■ #80530h/#7042Dh • Taille de la carte en port 1

Sur HP-48SX, ce quartet contient la taille de la carte en port 1 exprimée en complément à #100000h. Sur HP-48 GX, ce quartet vaut #0h, #1h ou #2h s'il est vide, contient respectivement une carte de 32 ko ou 128 ko.

■ #80531h/#70437h • Détails de configuration du port 2

Trois des bits de ce quartet apportent des précisions sur l'état du port 2 :

- le bit 1 vaut 1 si la carte est fusionnée avec la RAM interne,
- le bit 2 vaut 1 si la carte est accessible en écriture,
- le bit 3 vaut 1 si la carte est accessible en lecture.

■ #80532h/#70438h • Taille de la carte en port 2

Sur HP-48SX, ce quartet contient la taille de la carte en port 2 exprimée en complément à #100000h. Sur HP-48 GX, ce quartet contient une puis-

sance de 2 augmentée de 7 car les capacités des cartes mémoires croissent comme les puissances de 2 (128 ko, 256 ko, 512 ko, 1 Mo, 2 Mo et 4 Mo).

■ #80535h/- • Taille des cartes

Ce quartet précise le contenu des ports destinés aux cartes (HP-48 GX). Dans ce tableau, *n* est un entier compris entre 0 et 5.

Valeur du quartet	Port carte 1	Port carte 2
#1h	32 ko	pas de carte
#2h	128 ko	pas de carte
#4h	pas de carte	32 ko
#5h	32 ko	32 ko
#6h	128 ko	32 ko
#8h	pas de carte	$2^n \times 128$ ko
#9h	32 ko	$2^n \times 128$ ko
#Ah	128 ko	$2^n \times 128$ ko
#Fh	pas de carte	pas de carte

■ #80536h à #805DBh/#7044Dh à #70457h • Prochains objets de chacun des ports

Pour chacun des ports, ces séries de 5 quartets contiennent l'adresse où sera placé le prochain objet enregistré dans l'un des ports.

■ #8065Ah/- • Interdiction des séquences de touches [ON]-[...]

Lorsque ce quartet vaut #0h, les combinaisons de touches provoquant des arrêts du système ou des interruptions particulières comme [ON]-[C], [ON]-[D], [ON]-[E] et [ON]-[SPC] sont sans effet. La HP-48 se charge d'affecter régulièrement une valeur non nulle à ce quartet.

■ #8065Bh/#704DCh • Désactivation du tampon de clavier

Lorsque ce quartet a une valeur différente de #0h, les codes des touches pressées ne sont pas copiés dans le buffer de clavier (ces codes sont présentés dans le chapitre « Les masques des touches »). Cependant, vous pouvez continuer à exploiter les frappes au clavier à l'aide des manipulations habituelles des registres OUT et IN.

■ #80669h/#704EAh • Première touche à traiter

Ce quartet, dont la valeur est notée $C_{à\_traiter}$, contient le numéro du caractère à traiter parmi ceux placés dans la mémoire tampon (buffer) de clavier en #8066Bh/#704ECh. Selon le type de machine, deux formules permettent d'obtenir l'adresse du prochain caractère à traiter :

$$\#8066Bh + 2 \times C_{à\_traiter} \text{ sur HP-48 G/GX}$$
$$\#704ECh + 2 \times C_{à\_traiter} \text{ sur HP-48 S/SX}$$

Le facteur 2 est lié au fait que 2 quartets soient nécessaires au codage sur 1 octet d'un code de touche.

■ #8066Ah/#704EBh • Stockage de la prochaine touche

Ce quartet, dont la valeur est notée $C_{à\_saisir}$, contient le numéro du prochain caractère saisi au sein de la mémoire tampon (buffer) de clavier en #8066Bh/#704ECh. Selon le type de machine, deux formules permettent d'obtenir l'adresse où sera inséré le prochain caractère :

$$\#8066Bh + 2 \times C_{à\_saisir} \text{ sur HP-48 G/GX}$$
$$\#704ECh + 2 \times C_{à\_saisir} \text{ sur HP-48 S/SX}$$

On multiplie par 2 car un code de touche est codé sur 2 quartets.

■ #8066Bh/#704ECh • Mémoire tampon de clavier (buffer clavier)

Ces 32 quartets permettent le stockage de 16 pressions sur le clavier (voir les codes buffer de chaque touche au chapitre « Les masques des touches »). Ce buffer est circulaire, en effet, après avoir inséré un code en position #Fh, il introduit le suivant en #0h.

■ #8069Ah/#7051Bh • Suivi du rafraîchissement de l'écran

Ces quartets correspondent à une sauvegarde de l'adresse #128h de la RAM I/O (décompte des lignes « rafraîchies » de l'écran).

■ #806E9h/#7056Ah • Adresse des objets temporaires

Ces 5 quartets contiennent l'adresse de la zone où sont stockés les objets temporaires. Ces objets sont ceux créés à partir de la ligne de la com-

mande. Ils peuvent être maintenus en mémoire après leur suppression de la pile et ne sont détruits de façon certaine que lorsqu'ils sont devenus inutiles (un drapeau indique qu'un objet temporaire est - ou non - encore en activité) et à l'occasion d'une récupération de tous les espaces disponibles en mémoire. Cette opération, la « garbage collection » (aussi appelée, « ramasse-miettes », « récupération de mémoire » ou « glaneur de cellules ») consiste à rendre à nouveau disponibles les espaces occupés par des objets devenus inutiles.

La zone des objets temporaires commence par 5 quartets nuls. Chaque objet temporaire est ensuite codé à l'aide de trois champs :

- un drapeau indique que l'objet temporaire est utilisé ou non, (dans ce dernier cas il sera supprimé lors de la prochaine « garbage collection »),
- la taille de l'objet temporaire,
- l'objet temporaire lui-même.

■ #80702h/#70583h • Adresse de l'environnement temporaire

L'adresse codée sur ces 5 quartets correspond à la zone se trouvant à la suite de celle mémorisant le contenu de la ligne de commande. L'environnement temporaire s'y trouve. Il correspond à la sauvegarde de la dernière pile utilisateur (pile récupérable) et à des variables locales.

■ #8070Ch/#7058Dh • Adresse de la table des menus

Ces 5 quartets codent l'adresse où se trouve la table des menus. Celle-ci contient trois types de champs :

- #07Ch semble être une en-tête,
- une série de 7 adresses (alors qu'il n'y a que 6 articles par menu) pointant sur les objets graphiques représentant les différents articles du menu,
- une série d'adresses qui sont celles des objets à évaluer lorsque l'un des articles est sélectionné.

■ #80801h à #80814h/#70683h à #70696h • Drapeaux internes

Une série de drapeaux internes paramètrent la calculatrice. Nous allons décrire certains de ces drapeaux. Le quartet #80801h/#70683h :

- le bit 0 signale qu'une portion de l'écran est figée,
- si l'alimentation est extrêmement faible, le bit 1 signale que les objets de la pile doivent être représentés par leurs types,

• le bit 2 signale que la partie inférieure de l'écran affiche la ligne de commande,
• le bit 3 est armé si les sonneries programmées doivent être inhibées pendant qu'une pression sur le clavier est attendue.

Le quartet #80802h/#70684h :
• si l'heure est affichée, le bit 0 sert au gestionnaire d'interruptions,
• le bit 1 vaut 1 si le mode serveur est le mode courant,
• le bit 2 vaut 1 en cas de comparaison du menu courant avec le précédent menu,
• le bit 3 vaut 1 si le menu courant pose un problème.

Le quartet #80803h/#70685h :
• le bit 0 vaut 1 si la dernière pile utilisateur doit être enregistrée automatiquement,
• le bit 1 vaut 0 si le curseur intervient sur la ligne de commande en mode « écrasement ». Il vaut 1 si le curseur est en mode « insertion »,
• le bit 2 vaut 1 si ALG est le mode de saisie courant,

Le quartet #80804/#70686h :
• les bits 0, 1 et 2 signalent chacun qu'une zone particulière de l'écran est figée,
• le bit 3 vaut 1 si les objets graphiques représentant les articles de chaque menu doivent être mis à jour à chaque changement de menu.

Le quartet #80805h/#70687h :
• le bit 0 vaut 1 si la dernière ligne de commande doit être sauvegardée,
• le bit 1 vaut 1 quand l'interruption relative à l'horloge 1 (voir la description de la mémoire des entrées/sorties) est active,
• le bit 2 vaut 1 lorsque les caractères alphabétiques saisis le sont en mode « caractères minuscules »,
• le bit 3 vaut 1 en cas d'inhibition de la décompilation des objets en ligne de commande (cette décompilation a pourtant toujours lieu au sein de la pile).

Le quartet #80806h/#70688h :
• le bit 0 si le mode 1USER est le mode courant,
• le bit 1 vaut 1 si l'environnement courant peut être masqué,
• le bit 2 vaut 1 si l'environnement temporaire a été altéré,
• le bit 3 vaut 1 si l'une des zones de l'écran est figée.

Chacun des bits du quartet #80807h/#70689h indique que l'une des 4 zones de l'écran est utilisable normalement. Chacun des bits du quartet #80808h/#7068Ah indique que l'une des 4 zones de l'écran n'a pas été modifiée. Chacun des bits du quartet #80809h/#7068Bh indique que l'une des 4 zones de l'écran n'est pas utilisable normalement ou qu'un problème y a été détecté.

Le quartet #8080Ah/#7068Ch :

- le bit 0 vaut 1 si la ligne de commande est active et non vide,
- le bit 1 vaut 1 s'il est possible d'utiliser « normalement » les touches dont les fonctions restent inchangées lors de l'utilisation de l'un des environnements particuliers (Equation Writer, Matrix Writer, etc.),
- le bit 2 vaut 1 si le mode USR tolère l'utilisation de touches dont la fonction n'est pas modifiée par l'utilisation d'un environnement particulier,
- le bit 3 vaut 1 si un environnement particulier tel que Equation Writer ou Matrix Writer est utilisé.

Le quartet #8080Bh/#7068Dh :

- le bit 0 vaut 1 si, lors de l'utilisation de Equation Writer, l'utilisation de parenthèses est implicite pour l'exponentiation, la division et la racine carrée,
- le bit 1 sert à la décompilation des objets en ligne de commande,
- le bit 2 vaut 1 en cas de création d'une ligne de commande,
- le bit 3 vaut 1 si les touches du clavier ont leurs fonctions usuelles.

Le bit 0 du quartet #8080Ch/#7068Eh affiche ou non l'heure, et ce, sans tenir compte des paramètres introduits par l'utilisateur. En effet, le système masque l'horloge lorsqu'il a besoin de l'emplacement de celle-ci sur l'écran pour afficher des données.

■ #80840h/#706C2h • Sonnerie à venir et état des piles

Les bits de ce quartet sont représentatifs de l'état des piles et annoncent une sonnerie à venir :

- le bit 0 vaut 1 si une sonnerie a été programmée,
- le bit 1 vaut 1 si l'alimentation principale de la HP-48 est trop faible pour assurer à court terme un fonctionnement normal,
- le bit 2 vaut 1 si la pile de la carte en port 1 est faible,
- le bit 3 vaut 1 si la pile de la carte en port 2 est faible.

■ #80841h/#706C3h • Etat des témoins

L'état des 8 bits des 2 quartets de ce champ détermine l'allumage ou l'extinction de l'un des huit témoins (affichables en haut de l'écran).

Bits d'adresse #80841h/#706C3h	Témoin allumé si bit armé	Bits d'adresse #80842h/#706C4h	Témoin allumé si bit armé
0	« Radians »	0	↰
1	Entrées/sorties	1	↱
2	Halt !	2	α
3	Sonnerie échue	3	Sablier

■ #80843h/#706C5h • Drapeaux (flags) du système

Chaque bit de ces 16 quartets (le premier est en #80843h/#706C5h) correspond à un drapeau du système. Pour chacun de ces quartets, le tableau ci-après résume le rôle de chaque bit. Le bit 0 est celui de plus faible poids alors que le bit 3 est le bit de plus fort poids.

Adresse quartet HP-48 G/GX	Adresse quartet HP-48 S/SX	Quartet			
		Bit 3	Bit 2	Bit 1	Bit 0
#80843h	#706C5h	Inutilisé (4)	Résultats numériques (3)	Constante symbolique (2)	Solution principale (1)
#80844h	#706C6h	Taille des mots codant les entiers binaires (5 à 10)			
#80845h	#706C7h	Base de numération pour les entiers binaires (11 et 12)			
#80846h	#706C8h	Coordination du système (15 et 16)		Inutilisé (14)	Inutilisé (13)
#80847h	#706C9h	Exception si dépassement (20)	Mode nombres complexes (19)	Angle trigonométrique (17 et 18)	
#80848h	#706CAh	Dépassement positif (24)	Dépassement négatif (23)	Exception en cas de résultat infini (22)	Exception si dépassement (21)
#80849h	#706CBh	Inutilisé (28)	Inutilisé (27)	Indicateur de résultat infini (26)	Indicateur de dépassement (25)
#8084Ah	#706CCh	Nature du curseur en environnement graphique (32)	Remplissage des courbes (31)	Tracé continu ou discontinu des fonctions (30)	Inutilisé (29)
#8084Bh	#706CDh	Ecrasement ou non par les données introduites (36)	Format des données (35)	Périphérique d'impression (34)	Périphérique pour les entrées/sorties (33)
#8084Ch	#706CEh	Affichage de l'horloge (40)	Messages pendant les entrées/sorties (39)	Retour à la ligne pendant les entrées/sorties (38)	Double interligne lors des impressions (37)
#8084Dh	#706CFh	Sonneries programmées autorisées (44)	Répétition autorisée des sonneries programmées (43)	Format d'affichage de la date (42)	Format d'affichage de l'heure (41)
#8084Eh	#706D0h	Nombre de chiffres décimaux (45 à 48)			
#8084Fh	#706D1h	Affichage sur une seule ligne (52)	Signe de fraction (51)	Format des données (49 et 50)	
#80850h	#706D2h	Bip d'erreur (56)	Mémorisation du dernier argument (55)	Inutilisé (54)	Priorité (53)
#80851h	#706D3h	Verrouillage alphabétique « Alpha-lock » (60)	Affichage rapide du catalogue (59)	Affichage par le système de messages longs (58)	Sonnerie programmée (57)
#80852h	#706D4h	Enveloppe d'index (64)	ENTER « vectoriel » (63)	Mode utilisateur (62)	Verrouillage du mode utilisateur (61)

Le nombre entre parenthèses est le numéro du drapeau.

Ne confondez pas :

- les drapeaux du système (#80843h/#706C5h),
- les drapeaux internes (voir en #80801h/#70683h),
- les drapeaux de l'utilisateur (voir en #80853h/#70684h) qui sont accessibles via le menu MODES sur HP-48 G/GX (flags RPL),
- les drapeaux constitués par chaque bit du registre ST.

■ #80853h/#706D5h • Drapeaux (flags) de l'utilisateur

Chacun des drapeaux utilisateur (flag RPL) est représenté par l'état d'un bit de l'un de ces 16 quartets.

Adresse quartet HP-48 G/GX	Adresse quartet HP-48 S/SX	Quartet			
		Bit 3	Bit 2	Bit 1	Bit 0
#80853h	#706D5h	Flag 4	Flag 3	Flag 2	Flag 1
#80854h	#706D6h	Flag 8	Flag 7	Flag 6	Flag 5
#80855h	#706D7h	Flag 12	Flag 11	Flag 10	Flag 9
#80856h	#706D8h	Flag 16	Flag 15	Flag 14	Flag 13
#80857h	#706D9h	Flag 20	Flag 19	Flag 18	Flag 17
#80858h	#706DAh	Flag 24	Flag 23	Flag 22	Flag 21
#80859h	#706DBh	Flag 28	Flag 27	Flag 26	Flag 25
#8085Ah	#706DCh	Flag 32	Flag 31	Flag 30	Flag 29
#8085Bh	#706DDh	Flag 36	Flag 35	Flag 34	Flag 33
#8085Ch	#706DEh	Flag 40	Flag 39	Flag 38	Flag 37
#8085Dh	#706DFh	Flag 44	Flag 43	Flag 42	Flag 41
#8085Eh	#706E0h	Flag 48	Flag 47	Flag 46	Flag 45
#8085Fh	#706E1h	Flag 52	Flag 51	Flag 50	Flag 49
#80860h	#706E2h	Flag 56	Flag 55	Flag 54	Flag 53
#80861h	#706E3h	Flag 60	Flag 59	Flag 58	Flag 57
#80862h	#706E4h	-	Flag 63	Flag 62	Flag 61

■ #80924h/#707A6h • Paramètres d'impression

Sur 12 bits sont codés des paramètres de l'impression en cours :

- le bit 0 de #80924h/#707A6h vaut 1 si le dernier caractère imprimé n'était pas graphique,
- le bit 1 de #80924h/#707A6h vaut 1 si le dernier caractère était un « Esc » (échap.),

- le bit 2 de #80924h/#707A6h vaut 1 en cas d'impression en largeur double,
- le bit 3 de #80924h/#707A6h vaut 1 si une impression graphique (avec des codes graphiques) est en cours,
- le bit 0 de #80925h/#707A6h vaut 1 si le dernier caractère était un LF (passage à la ligne suivante),
- le bit 2 de #80925h/#707A6h vaut 1 si une ligne à imprimer est attendue,
- le bit 3 de #80925h/#707A6h vaut 1 si l'alimentation est trop faible (risque d'erreur de transmission),
- le bit 0 de #80926h/#707A7h vaut 1 si le double interligne doit être utilisé après le prochain retour à la ligne,
- le bit 1 de #80926h/#707A7h vaut 1 si le double interligne cesse d'être utilisé après le prochain retour à la ligne.

■ #809A3h/#707D9h • Nombre de bibliothèques attachées

On trouve sur 3 quartets en #809A3h/#707D9h le nombre de bibliothèques attachées. Chaque bibliothèque est ensuite décrite sur 16 quartets par ordre croissant de numéro de bibliothèque. Ce champ de 16 quartets à la structure suivante :

- 3 quartets pour le numéro d'identification de la bibliothèque,
- 5 quartets désignent l'adresse du champ où se trouve le numéro de la bibliothèque (au sein de la bibliothèque),
- 5 quartets désignent l'adresse de la routine donnant accès à la bibliothèque. Il peut s'agir d'une routine chargée de rendre temporairement visible une zone normalement masquée de la mémoire,
- 3 quartets sont à l'usage de la HP-48.

25

Quelques routines en mémoire morte

1 Introduction aux routines

Les routines peuvent être comparées à des (sous-) programmes placés en ROM (ROM non masquée par les autres mémoires de préférence !). Il ne nous a pas semblé utile dans le présent ouvrage de vous présenter de façon exhaustive l'intégralité des points d'entrée en ROM. C'est donc une sélection de routines utilisables par la plupart d'entre vous qui vous est présentée. Ces routines sont classées par domaine d'utilisation. Chaque routine vous est présentée dans un petit tableau. Vous y trouverez l'adresse de la routine (pour HP-48 S/SX et G/GX), l'instruction d'appel, l'instruction de retour (les routines sont des sous-programmes !) ainsi que le nombre de niveaux de la pile RSTK qui seront temporairement utilisés par la routine. Les champs des registres sont ici notés ***registre(champ)***. Ainsi C(A) représente le champ A du registre C. De même, C(0→4) représente les cinq premiers quartets du champ C.

Description succincte de la routine		
Adresse HP-48 S/SX : #...h	**Adresse HP-48 G/GX** : #...h	
Appel : ...	**RSTK** : ...	**Retour** : ...
Remarques : *La rubrique « remarque » vous apporte des informations complémentaires sur l'activité de la routine, en particulier, les contenus des registres utilisés ou modifiés par ladite routine vous sont précisés. Lorsque seuls certains champs des registres utilisés sont affectés par la routine, ceux-ci vous sont précisés dans cette rubrique.*		

2
Manipulation des registres

Sauvegarde de D0, D1, B(A) et D(A).		
Adresse HP-48 S/SX : #0679Bh		**Adresse HP-48 G/GX** : #0679Bh
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : Dans le cadre d'une utilisation sur HP-48 S/SX, le registre D0 contiendra #7066Eh à l'issue de l'appel de la routine. Cette routine sauvegarde les registres D0, D1, B(A) et D(A) aux adresses #705B0h (D0), #70579h (D1), #70574h (B(A)) et #7066Eh (D(A)). Ces adresses sont valables pour les HP-48 S/SX mais la routine fonctionne aussi sur G/GX.		

Récupération de D0, D1, B(A), C(A) et D(A).		
Adresse HP-48 S/SX : #067D2h		**Adresse HP-48 G/GX** : #067D2h
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : Cette routine complète la précédente. On remarquera que son action ne concerne que des groupes de 5 quartets (des adresses). Sur HP-48 S/SX, les contenus des adresses #705B0h, #70579h, #70574h, #705B0h et #7066Eh seront respectivement placés dans D0, D1, B(A), C(A) et D(A). Cette routine est aussi utilisable sur HP-48 G/GX.		

Sauvegarde de A et B dans R0 et R1 (et routines assimilées)		
Adresse HP-48 S/SX : #2BE61h		**Adresse HP-48 G/GX** : #2BE61h
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : Les contenus des registres de calcul à A et C (qui demeurent inchangés) sont intégralement copiés dans les registres de sauvegarde R0 et R1. De même on trouve en #2BE6Fh, #2BE7Dh et #2BE8Bh des routines sauvegardant respectivement A et B dans R2 et R3 (#2BE6Fh), C et D dans R0 et R1 (#2BE7Dh), C et D dans R2 et R3 (#2BE8Bh).		

Récupération de A et B depuis R0 et R1 (et routines assimilées)		
Adresse HP-48 S/SX : #2BEB5h		**Adresse HP-48 G/GX** : #2BEB5h
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : A et C prennent pour valeurs les contenus des registres de sauvegarde R0 et R1. On trouve aussi en #2BEC0h, #2BECBh et #2BED6h des routines récupérant respectivement A et B depuis R2 et R3 (#2BEC0h), C et D depuis R0 et R1 (#2BECBh), C et D depuis R2 et R3 (#2BED6h).		

Copie dans D0 de l'adresse de la fin de la mémoire disponible		
Adresse HP-48 S/SX : #6384Eh	**Adresse HP-48 G/GX** : #6384Eh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : une fois cette routine exécutée, le registre D0 pointe sur l'adresse de la fin de la mémoire disponible. Remarquez que le champ A du registre A contiendra aussi l'adresse de la fin de la mémoire disponible. On aura donc D0 = A(A)...		

Copie dans D1 de l'adresse de la fin de la mémoire disponible		
Adresse HP-48 S/SX : #6385Dh	**Adresse HP-48 G/GX** : #6385Dh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : cette routine est très proche de la précédente. Une fois exécutée, le registre D1 pointe sur l'adresse de la fin de la mémoire disponible. Remarquez que le champ A du registre C contiendra aussi l'adresse de la fin de la mémoire disponible. On aura donc ici D1 = C(A).		

3

La gestion du clavier

Initialisation des quartets « key state »		
Adresse HP-48 S/SX : #00D3Bh	**Adresse HP-48 G/GX** : #00D3Bh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : Cette routine met à zéro les quartets « key state », ces quartets sont au nombre de treize. Après l'appel de cette routine, le registre C est mis à zéro et le registre D1 pointe en #704DDh (HP-48 S/SX dont la RAM interne commence en #70000h) ou en #8065Ch (HP-48 G/GX dont la RAM interne commence en #80000).		

Initialisation de la mémoire tampon du clavier (buffer de clavier)		
Adresse HP-48 S/SX : #00D57h	**Adresse HP-48 G/GX** : #00D57h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : la mémoire tampon du clavier (buffer de clavier) est mise à zéro. Dans le cas des HP-48 S/SX dont la RAM interne commence en #70000h, le registre D1 pointe en #704EAh après l'appel de la routine.		

Ajout d'une touche dans la mémoire tampon du clavier (buffer)		
Adresse HP-48 S/SX : #00D9Fh		**Adresse HP-48 G/GX** : #00D9Fh
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : C(B) doit contenir le code de la touche avant l'appel de la routine. Après l'appel de la routine, A(A) est une copie de C(A) avant l'appel de la routine. D1 pointe sur l'adresse où est stocké le code de la touche. Tout ceci n'étant valable que si le buffer n'est pas plein. S'il l'est, le retour s'effectue grâce à RTNYES.		

Détection d'une pression sur une touche		
Adresse HP-48 S/SX : #00E0Bh		**Adresse HP-48 G/GX** : #00E0Bh
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNYES
Remarques : si une touche est pressée, C(A) contient, après exécution de la routine, la valeur d'entrée de la touche. Si aucune touche n'est pressée, le retour s'effectue par RTN et C(A) vaut #00000h. Dans les deux cas, le masque de sortie pour la détection d'une pression sur une touche du clavier est mis à #1FFh.		

Instruction C = IN.		
Adresse HP-48 S/SX : #01160h		**Adresse HP-48 G/GX** : #01160h
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : cette routine peut se substituer à l'instruction C = IN. Après son appel, les 5 quartets de poids faible du registre C (quartets 0 à 4), contiennent la valeur du registre IN.		

Instruction OUT = C puis C = IN.		
Adresse HP-48 S/SX : #01EECh		**Adresse HP-48 G/GX** : #01EECh
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : avant l'appel de la routine, les quartets 0, 1 et 2 du registre C doivent contenir le masque de sortie du clavier. Après l'appel de la routine , les quartets 0, 1, 2 et 3 du registre C contiennent la valeur du registre IN.		

Suppression du code de la première touche à traiter		
Adresse HP-48 S/SX : #04840h		**Adresse HP-48 G/GX** : #04840h
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : Le code de la première touche à traiter est supprimé de la mémoire tampon du clavier. C(B) contient le code de le première touche de la mémoire tampon. Si toutes les touches de la mémoire tampon du clavier ont déjà été traitées, le retour s'effectue avec RTNYES et C(A) contient #00000h.		

4

Ecran et graphisme

Activation de l'écran		
Adresse HP-48 S/SX : #01B8Fh	**Adresse HP-48 G/GX** : #01B8Fh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : cette routine active l'écran (elle « l'allume »). Après l'utilisation de cette routine le registre D0 pointe en #100h. Les quartets 0, 1 et 2 placés à l'adresse #100h sont utilisés pour le codage de la marge sur la gauche de l'écran.		

Désactivation de l'écran		
Adresse HP-48 S/SX : #01BBDh	**Adresse HP-48 G/GX** : #01BBDh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : cette routine désactive l'écran (elle « l'éteint »). Après l'utilisation de cette routine le registre D0 pointe en #100h.		

Mise à jour des adresses utilisées pour l'affichage		
Adresse HP-48 S/SX : #01C7Fh	**Adresse HP-48 G/GX** : #01C7Fh	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : cette routine est à utiliser après un déplacement de la RAM interne pour mettre à jour les adresses (leurs contenus) nécessaires pour l'affichage. A(W), B(W), C(W), D(A), ST3, D0 et D1 sont modifiés par cette routine.		

Copie des contenus des adresses #7050Eh, #70513h et #7051Bh		
Adresse HP-48 S/SX : #01D53h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : les contenus des adresses #7050Eh, #70513h et #7051Bh sont respectivement copiés en #120h, #125h et #128h. Avant l'appel de la routine, C(S) doit contenir la marge à gauche. Après l'exécution de cette routine, l'écran est éteint et le clavier est désactivé.		

Création d'un objet graphique (GROB) vierge		
Adresse HP-48 S/SX : #115B3h	**Adresse HP-48 G/GX** : #115B3h	
Appel : GOSBVL	**RSTK** : 2	**Retour** : RTNCC
Remarques : Avant l'appel de la routine, R0(A) doit contenir le nombre de ligne du GROB alors que R1(A) contient son nombre de colonnes. Après l'appel de la routine, B(A) contient la longueur totale hors prologue, C(A) contient l'adresse de la fin du GROB alors que R3(A) contient le nombre de quartets nécessaires pour coder une ligne du GROB.		

Recherche de l'adresse d'un pixel		
Adresse HP-48 S/SX : #11C58h	**Adresse HP-48 G/GX** : #11C58h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTN
Remarques : avant l'appel de la routine, R2(A) contient le numéro de la ligne du pixel dans le GROB, R4(A), le numéro de la colonne, et D1, l'adresse du GROB. Après l'appel de la routine A(A) contient l'adresse du quartet dont l'un des bits définis le pixel dont le numéro de ligne et le numéro de colonne ont été précisés.		

Masquage de la pile, le menu horizontal étant maintenu		
Adresse HP-48 S/SX : #126A3h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : L'objet graphique correspondant à la pile est masqué. Le clavier demeure actif alors que les menus horizontaux restent affichés. Après l'appel de la routine D1 pointe sur l'adresse définissant la hauteur du menu.		

Bascule entre deux écrans		
Adresse HP-48 S/SX : #132C5h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : avec cette routine l'objet graphique non visible est affiché et devient donc le GROB courant. Les registres A(W), B(W), C(W), D(A), ST3, D0 et D1 sont modifiés par cette routine.		

Coordonnées du premier pixel		
Adresse HP-48 S/SX : #137DCh	**Adresse HP-48 G/GX** : #137DCh	
Appel : GOSBVL	**RSTK** : 2	**Retour** : RTNCC
Remarques : après l'appel de la routine C(W), A(W), B(W) et R0 contiennent respectivement l'abscisse du point supérieur gauche de l'écran, l'ordonnée de ce point, le nombre de colonnes du GROB et le nombre de quartets codant une ligne. Ces coordonnées dépendent du GROB courant et sont nulles si les dimensions du GROB sont 131 × 64.		

Etat d'un pixel		
Adresse HP-48 S/SX : #138F2h		**Adresse HP-48 G/GX** : #138F2h
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTN
Remarques : avant l'appel de la routine, A(A) contient le numéro de la colonne du pixel (en hexadécimal), R0(A), le numéro de la ligne et ST1, 0 s'il s'agit du GROB graphique et 1 s'il s'agit du GROB de la pile. Après l'appel de la routine, effectuer `A = A & B P` (P doit ici correspondre au quartet 0). Si A(P) est nul, le pixel est éteint, sinon, le pixel est allumé.		

Blocage de l'affichage		
Adresse HP-48 S/SX : #53B31h		**Adresse HP-48 G/GX** : N.C.
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTN
Remarques : comme la fonction FREEZE, cette routine fige l'affichage. Avant l'appel de cette routine, le champ X de C doit contenir #0028Ah.		

5
Opérations à partir des répertoires

Décalage d'adresse vers le dernier objet d'un répertoire		
Adresse HP-48 S/SX : #0839Bh		**Adresse HP-48 G/GX** : #0839Bh
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : avant l'appel de la routine, on place dans A(A) l'adresse d'un objet appartenant à un répertoire non caché autre que HOME. Après l'appel de la routine, D0 pointe sur le décalage vers le dernier objet de ce même répertoire. Cas particuliers : l'objet recherché est le répertoire HOME, le répertoire caché ou un objet du répertoire HOME, etc.		

Décalage d'adresse vers le dernier objet d'un répertoire		
Adresse HP-48 S/SX : #083D1h		**Adresse HP-48 G/GX** : #083D1h
Appel : GOSBVL	**RSTK** : 2	**Retour** : RTNCC
Remarques : cette routine est très proche de la précédente. La différence réside dans le fait que la présente routine fournisse (après appel) dans A(A) l'adresse du répertoire contenant l'objet.		

Saut du nom d'un objet pour pointer sur le contenu de celui-ci		
Adresse HP-48 S/SX : #08400h	**Adresse HP-48 G/GX** : #08400h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : Avant l'appel de la routine, D0 doit contenir l'adresse du nom de l'objet moins 2 quartets, c'est-à-dire l'adresse du nombre de caractères du nom de l'objet. Après l'appel de la routine, D0 contient l'adresse de l'objet.		

Saut vers l'adresse du début du nom d'un objet		
Adresse HP-48 S/SX : #0841Eh	**Adresse HP-48 G/GX** : #0841Eh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : cette routine est à opposer à la précédente. Avant l'appel de la routine, D0 doit contenir l'adresse de l'objet. Après l'appel de celle-ci, C(A) contient l'adresse du nom de l'objet. S'il s'agit d'un objet portant un « nom vide », C(A) vaut #00000h après l'appel de la routine.		

Décalage d'adresse vers le dernier objet d'un répertoire		
Adresse HP-48 S/SX : #08503h	**Adresse HP-48 G/GX** : #08503h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNSC
Remarques : A(A) contenant l'adresse d'un répertoire, cette routine fournit dans D0 (après appel) l'adresse du décalage vers le dernier objet du répertoire.		

6

Applications mathématiques

Vérification que A(A) est bien divisible par #32d		
Adresse HP-48 S/SX : #0D485h	**Adresse HP-48 G/GX** : #0D485h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNYES ou RTN
Remarques : le nombre dont la divisibilité par 32_{10} est à tester doit être placé dans A(A). SB vaut 1 après appel de la routine si le nombre est divisible par 32_{10}, le retour s'effectue alors par RTNYES. Si SB vaut 0, le retour s'effectue par RTN. Si A(B) = #00h, la divisibilité par 32_{10} n'est testée que sur les quartets 2, 3 et 4 du champ A du registre A.		

Conversion du décimal vers l'hexadécimal du registre C		
Adresse HP-48 S/SX : #0D62Fh	**Adresse HP-48 G/GX** : #0D62Fh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : C(W) doit contenir une valeur décimale, après l'appel de la routine C(W) mais aussi A(W) et B(W) contiennent la valeur initiale de C(W) convertie en hexadécimal. Cette routine ne peut pas renvoyer comme résultat de la conversion une valeur supérieure à #2386F26FC0FFFFh.		

Conversion de l'hexadécimal vers le décimal du registre A		
Adresse HP-48 S/SX : #0DB91h	**Adresse HP-48 G/GX** : #0DB91h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : A(W) doit contenir une valeur hexadécimale. Après l'appel de la routine, A(W), mais aussi B(W) et C(W) contiendront le nombre converti en décimal. La HP-48 est placée en mode « décimal » après l'utilisation de cette routine.		

Multiplication de A(A) par C(A)		
Adresse HP-48 S/SX : #11642h	**Adresse HP-48 G/GX** : #11642h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : A(A) et C(A) doivent contenir les deux valeurs (sur 5 quartets) à multiplier. Si le produit désiré est codable sur 5 quartets, B(A) contient ce produit et C(A) contient #FFFFFh après appel de la routine. En cas de dépassement, le retour s'effectue par RTNC. Dans les deux cas A(A) contient (après appel) la plus grande des deux valeurs à multiplier.		

Etat d'un bit appartenant à un quartet d'adresse connue		
Adresse HP-48 S/SX : #1395Bh	**Adresse HP-48 G/GX** : #1395Bh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : avant l'appel de la routine, A(P), C(P) et D0 contiennent respectivement le numéro du bit à tester, le nombre 3 et l'adresse du quartet contenant le bit à tester. Après appel, A(P) contient le quartet testé et B(P), une valeur (1, 2, 4 ou 8). Après avoir exécuter `A = A & C P` A(P) contient la valeur du bit (il ne peut s'agir que de 0 ou de 1 !).		

Multiplication de C(W) par A(W)		
Adresse HP-48 S/SX : #53EE4h	**Adresse HP-48 G/GX** : #53EE4h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : placer dans A(W) et C(W) les deux valeurs à multiplier. Après l'appel de la routine A(W), C(W) et D(W) contiendront le produit calculé. Notez que le cas d'un débordement n'est pas pris en compte par cette routine.		

Conversion de l'hexadécimal vers le décimal des quartets 0 à 5 de A		
Adresse HP-48 S/SX : #657DFh	**Adresse HP-48 G/GX** : #657DFh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : les quartets 0 à 5 de A contiennent la valeur à convertir avant l'appel de la routine. Après l'appel de la routine, les quartets 0 à 5 de A valent #000000h, C(A) vaut #09999h et la valeur convertie est placée dans B(W). La capacité de ce champ (en fait, l'intégralité du registre B), interdit tout dépassement de capacité (débordement).		

Division de A par C		
Adresse HP-48 S/SX : #65807h	**Adresse HP-48 G/GX** : #65807h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : A(W) et C(W) doivent contenir respectivement le dividende et le diviseur. Après l'appel de la routine, A(W) et C(W) contiennent le quotient alors que B(W) contient le reste de la division. Notez que les nombres sont des entiers positifs. On trouve en #0DB91h une routine capable de diviser des nombres décimaux.		

7
Gestion de la mémoire

Vérification du bon fonctionnement d'un groupe de cinq quartets		
Adresse HP-48 S/SX : #01E2Ch	**Adresse HP-48 G/GX** : #01E2Ch	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : C(A) doit contenir l'adresse des 5 quartets à tester. Après l'appel de la routine, A(A) contient #00000h si tous les bits fonctionnent. Si ce n'est pas le cas, le retour s'effectue par RTNSC, A(A) ne vaut pas #00000h et D0 ne pointe pas sur le prochain groupe de 5 quartets comme il le fait si le test est normal.		

Configuration du pilote de l'écran		
Adresse HP-48 S/SX : #02248h	**Adresse HP-48 G/GX** : #02241h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : après l'appel de cette routine, C(A) contient 01000.		

Configuration du module courant en #70000h (HP-48 S/SX)		
Adresse HP-48 S/SX : #02256h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : après l'appel de la routine, C(A) contient #F0000h et D1, #70005h. Un module ne peut être configuré que s'il est déconfiguré. Notez qu'une mise à jour de l'affichage est nécessaire à l'issue de l'appel de cette procédure (on peut pour cela faire appel à la routine se trouvant en #01C7Fh).		

Déplacement de la totalité de la RAM des HP-48 S/SX		
Adresse HP-48 S/SX : #0228Eh	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : après l'appel de cette routine, C(A) contient #F0000h et D11, #F0005h. La RAM est déplacée dans sa totalité et est configurée à l'adresse #F0000h. L'utilisation d'une routine spécialisée, telle que celle disponible en #01C7Fh est indispensable pour reconfigurer l'affichage convenablement.		

Mise à jour des pointeurs pour une reconfiguration de la RAM		
Adresse HP-48 S/SX : #0229Fh	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : une mémoire ne peut être configurée que si elle ne l'est pas déjà. Avant l'appel de la routine, A(A) doit contenir l'adresse de configuration alors que C(A) contient *#800h × (#200h - TR)* où TR est la taille de la RAM à configurer. TR est exprimé en kilo-octets et en hexadécimal. Après l'appel de la routine, C(A) contient l'adresse de configuration.		

Configuration de 32 ko en #C0000h		
Adresse HP-48 S/SX : #022D9h	**Adresse HP-48 G/GX** : #022D9h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : après l'appel de cette routine, C(A) contient #F0000h et D1 pointe en #C0005h. Dans le cas des HP-48 G/GX, la RAM interne doit être déconfigurée avant d'exécuter cette routine.		

Exécution d'une routine placée en ROM cachée		
Adresse HP-48 S/SX : #026BFh	**Adresse HP-48 G/GX** : #026CAh	
Appel : GOSBVL	**RSTK** : 2	**Retour** : RTN
Remarques : avant l'appel de cette routine, placez dans C(A) l'adresse en ROM cachée de la routine à exécuter. A, B, C, D, ST3, D0 et D1 sont utilisés par cette routine qui prend par ailleurs en charge une mise à jour de l'affichage.		

Taille d'un objet dont on fournit l'adresse		
Adresse HP-48 S/SX : #05927h	**Adresse HP-48 G/GX** : #05927h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTN
Remarques : avant l'appel de la routine, A(A) et R1(A) doivent contenir l'adresse de l'objet. Après l'appel de la routine, A(A) et R0(A) contiennent la longueur de l'objet (nombre de quartets utilisés pour le coder intégralement). D0 contient l'adresse d'un éventuel épilogue de l'objet. S'il n'y a pas d'épilogue, D0 contient l'adresse de l'objet + #5h.		

Code de redondance cyclique (CRC) d'une portion de mémoire		
Adresse HP-48 S/SX : #0597Eh	**Adresse HP-48 G/GX** : #0597Eh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTI
Remarques : A(A) et C(A) doivent contenir respectivement la longueur de la zone et l'adresse de celle-ci. Après l'exécution de la routine, A(A) contient le CRC recherché et D0 pointe sur l'adresse de la fin de la zone dont le CRC devait être calculé. Notez que les interruptions sont inhibées pendant l'exécution de cette routine.		

Copie d'une portion de mémoire vers une adresse supérieure		
Adresse HP-48 S/SX : #066B9h	**Adresse HP-48 G/GX** : #066B9h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : C(A) doit contenir la longueur de la portion de mémoire à copier, D0 contient l'adresse de « l'original » alors que D1 doit contenir l'adresse de la « copie ». Il faut ici que l'adresse en D1 soit supérieure à l'adresse D0 et, pour éviter les superpositions, que D0 + C(A) < D1. Après l'appel, C(A) vaut #FFFFFh si la zone à copier a une longueur nulle.		

Copie d'une portion de mémoire vers une adresse inférieure		
Adresse HP-48 S/SX : #0670Ch	**Adresse HP-48 G/GX** : #0670Ch	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : C(A) contient la longueur du segment de mémoire à copier, D0 contient l'adresse de « l'original » alors que D1 contient l'adresse de la « copie ». Il faut ici que l'adresse en D1 soit inférieure à l'adresse D0 et, pour que les zones ne soient pas superposées, que D1 + C(A) < D0. Après l'appel, C(A) vaut #FFFFFh si la zone à copier a une longueur nulle.		

Remplissage par des zéros d'une zone de mémoire		
Adresse HP-48 S/SX : #0675Ch	**Adresse HP-48 G/GX** : #0675Ch	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : C(A) et D1 contiennent respectivement la longueur de la zone à initialiser et l'adresse de cette zone. La portion de mémoire ainsi définie sera remplie de zéros. Après l'appel de la routine, C(A) et D1 contiendront l'adresse de fin de la zone à initialiser alors que A(W) sera mis à zéro.		

Calcul de la mémoire disponible		
Adresse HP-48 S/SX : #069F7h	**Adresse HP-48 G/GX** : #069F7h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : sur HP-48 S/SX, le contenu de #7066Eh est calculé. Sur HP-48 G/GX, c'est le contenu de #807EDh qui sera recalculé. Après appel de cette routine, C(A) contient le nouveau contenu de #7066Eh ou de #807EDh selon le type de machine.		

Vérification : deux zones en mémoire sont-elles identiques ?		
Adresse HP-48 S/SX : #0787Dh	**Adresse HP-48 G/GX** : #07848h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : C(X) contient la longueur des zones à comparer (C(XS) devant être nul), D0 et D1 contiennent les adresses des zones à comparer. Après appel de la routine D0 contient l'adresse de fin de la première zone si les deux zones sont identiques. Si ce n'est pas le cas, D0 pointe à 16 quartets près sur la première adresse où diffèrent les deux zones.		

Sélection d'une partition du port 2 (partition 2:n avec n≥0)		
Adresse HP-48 S/SX : N.C.	**Adresse HP-48 G/GX** : #09CFAh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : placer préalablement dans C(B) le numéro de la partition à sélectionner + 1. Après appel, D0 contient l'adresse de la partition, c'est-à-dire *#7F040h + 2×(numéro de la partition - 1)*. Dans le cas où le numéro de la partition est nul, le retour s'effectue grâce à RTNYES. Cette routine n'effectue pas de déconfiguration d'une zone de mémoire.		

Libération d'une zone de mémoire réservée aux objets temporaires		
Adresse HP-48 S/SX : #16671h	**Adresse HP-48 G/GX** : #16671h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : placer dans R0(A) l'adresse de l'objet et dans D0 l'adresse de fin de cette objet. Cette routine rend à nouveau disponible la portion de mémoire réservée aux objets temporaires, et ce, à partir de l'adresse de fin (ici placée dans D0) de l'un des objets y résidant.		

Sélection d'une partition en port 2		
Adresse HP-48 S/SX : N.C.	**Adresse HP-48 G/GX** : #70902h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNYES
Remarques : D0 doit contenir l'adresse de la partition. Si une carte se trouve dans le port 1, son contenu est déconfiguré, un étage de RSTK est utilisé, C(A) et D0 contiennent respectivement #C0000h et #800E6h et le retour doit alors s'effectuer avec RTN.		

Annulation de la sélection d'une partition du port 2		
Adresse HP-48 S/SX : N.C.	**Adresse HP-48 G/GX** : #70927h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : il n'est pas nécessaire d'affecter des valeurs particulières aux registres avant leur appel. Si une carte se trouve en port 1, celle-ci est configurée en #C0000h et le retour s'effectue avec RTN. S'il n'y a pas de carte en port 1, le retour s'effectue avec RTNC et C(A) contient #F000Fh.		

La ROM cachée (entre #C0000h et #F0000h) est rendue visible		
Adresse HP-48 S/SX : N.C.	**Adresse HP-48 G/GX** : #70B66h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNC
Remarques : aucune valeur particulière ne doit être affectée aux registres avant l'appel de la routine. Il existe deux cas particuliers. Si une carte est détectée dans l'un des ports, D0 vaut #00128h (après l'appel) et le retour s'effectue avec RTNCC. Si le contenu initial de #80535h est nul, le retour s'effectue avec RTNYES. Voir aussi la routine suivante.		

La ROM cachée (entre #C0000h et #F0000h) est à nouveau masquée		
Adresse HP-48 S/SX : N.C.	**Adresse HP-48 G/GX** : #70BB7h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNC
Remarques : aucune valeur particulière ne doit être affectée aux registres avant l'appel de la routine. Il existe deux cas particuliers. Si une carte est détectée dans l'un des ports, D0 vaut #8065Ah (après l'appel) et le retour s'effectue avec RTN. Si le contenu initial de #80535h est nul, le retour s'effectue avec RTNYES. Cette routine est à associer à la précédente.		

8

Horloge, coma, interruptions, etc.

Extinction du calculateur ou passage en mode « coma »		
Adresse HP-48 S/SX : #002FFh	**Adresse HP-48 G/GX** : #002FFh	
Appel : GOSBVL	**RSTK** : 3	**Retour** : RTNCC
Remarques : avant d'appeler la routine, réglez ST0 à 0 pour éteindre le calculateur, ou à 1 pour le placer en mode « coma ». Les registres A (quartet n°0), C(A), D(B), D0 et D1 sont utilisés par cette routine. Une fois la HP-48 placée en mode « coma » (ou éteinte), l'exécution du programme reprend normalement dès qu'une pression sur [ON] est détectée.		

Autorisation de traiter les interruptions		
Adresse HP-48 S/SX : #010E5h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTI
Remarques : cette routine est à comparer à l'instruction INTON. Il n'est pas nécessaire d'affecter des valeurs particulières aux registres avant d'exécuter cette routine. Les drapeaux ST14 et ST15 sont modifiés par cette routine.		

Inhibition des interruptions		
Adresse HP-48 S/SX : #01115h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : les valeurs initiales des registres sont ici quelconques. Après l'exécution de la routine, C(A) contient la valeur initiale de D0 et ST15 est mis à 0 pour rendre inopérant le gestionnaire d'interruptions. La dernière instruction exécutée par cette routine est INTOFF.		

Vérification de la non-corruption du décalage de l'horloge		
Adresse HP-48 S/SX : #011FCh	**Adresse HP-48 G/GX** : #011FCh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : les valeurs initiales des registres sont quelconques. Après appel, D1 vaut #00138h. B(W) contient un décalage de date en 1/8192e de seconde à partir de la date de début de série (1/1/1989 pour les S/SX). C(W) contient la date. Si la date est invalide le retour s'effectue par RTNSC ou RTNYES (date antérieure au 1/1/1989 ou postérieure au 31/12/2088).		

Enregistrement du décalage d'horloge et de son CRC		
Adresse HP-48 S/SX : #016E9h	**Adresse HP-48 G/GX** : #016E9h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : les 13 premiers quartets de A contient le décalage de l'horloge avant l'appel de la routine. Après l'appel de la routine, D0 contient l'adresse du calculateur de CRC et D1 l'adresse où se trouve le décalage de l'horloge (#70052h sur S/SX et #80052h sur G/GX). Le CRC est enregistrée 13 quartets après le décalage d'horloge (en #7005Fh et #8005Fh).		

Message fourni par WSLOG		
Adresse HP-48 S/SX : #026FDh	**Adresse HP-48 G/GX** : #0207Dh	
Appel : GOSBVL	**RSTK** : 2	**Retour** : RTN
Remarques : placer dans P le numéro de l'erreur avant d'appeler la routine. Si le décalage d'horloge n'est pas altéré, A(W) contient la date valide. S'il est altéré , A(W) est mis à zéro. Dans les deux cas, D0 contient l'adresse du CRC du premier message de WSLOG. Cette routine mémorise le type d'erreur, la date et le CRC du premier message relatif à WSLOG.		

Vérification de l'alimentation en vue d'entrées/sorties		
Adresse HP-48 S/SX : #325AAh	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : ST7 doit valoir 0 avant d'appeler la routine. Après appel et si les piles sont normales, le quartet 0 de B vaut #2h. Si les piles sont trop faibles pour envisager l'utilisation du port série ou du port infrarouge, B(A) vaut #00C14h (ce qui correspond au message d'erreur « piles faibles ») et le retour s'effectue RTNSC.		

9

Transmission de données

Un octet reçu a-t-il été traité ?		
Adresse HP-48 S/SX : #310CAh	**Adresse HP-48 G/GX** : #310CAh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN ou RTNYES
Remarques : avant l'appel, les valeurs des registres peuvent être quelconques. Après l'appel, le quartet n°0 du registre A contient une copie du quartet d'adresse #00112h. Le retour s'effectue par RTN (le bit 0 en #112h est 0) ou par RTNYES (le bit 0 en #112h est 1). Un caractère a été reçu mais n'a pas été traité si le bit n°0 en #00112h est un 1.		

Association d'un bit de parité à un caractère codé en ASCII		
Adresse HP-48 S/SX : #3113Dh	**Adresse HP-48 G/GX** : #3113Dh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : avant l'appel, A(B) doit contenir le code ASCII du caractère auquel doit être associé son bit de parité. Après l'appel, si le mode de parité est « none », A(B) reste inchangé. Si le mode est « odd », « even », « mark », ou « space », A(B) contient le caractère ASCII associé à son bit de parité.		

Vérification de l'ouverture du port d'entrées/sorties		
Adresse HP-48 S/SX : #311B3h	**Adresse HP-48 G/GX** : #311B3h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : le contenu des registres est quelconque avant l'appel. Après l'appel, le troisième bit en #00110h vaut 1 si le port d'entrées/sorties est ouvert. Si le port est fermé, on trouve en C le code du message d'erreur « port fermé ». Si le port est fermé, le retour s'effectue grâce à RTNSC. Dans le deux cas, D0 vaut, après appel, #00110h.		

Fermeture du port d'entrées/sorties		
Adresse HP-48 S/SX : #315F9h	**Adresse HP-48 G/GX** : #315F9h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : le port d'entrées/sorties est ouvert quand le quartet en #00110h vaut #Bh. Il est fermé quand ce quartet vaut #0h. Cette routine modifie donc le quartet d'adresse #00110h. Le port ne peut pas être fermé si une transmission est en cours. Après appel, D0 et C(A) valent respectivement #0010h et #00000h.		

Abandon du mode « serveur » pour les entrées/sorties		
Adresse HP-48 S/SX : #53B0Ah	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTN
Remarques : il faut placer #284h dans C(X) avant d'appeler cette routine. Après l'appel de la routine, D0 pointe sur l'adresse #70684h ce qui correspond - sur HP-48 S/SX - à l'adresse du quartet codant le mode courant. A(A) contient une copie de D0 et le cinquième quartet de C vaut #Dh après l'appel de cette routine.		

Transfert bit à bit par infrarouge du registre A		
Adresse HP-48 S/SX : #65906h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : placer dans le registre A les quartets à transmettre et dans C(B) le nombre de bits à transmettre moins 1 (il s'agit du nombre de bits du registre A qui sont à transmettre). Pendant la transmission, le registre A est décalé bit par bit jusqu'à ce qu'il soit transmis dans son ensemble. D0 pointe après l'appel en #0011Dh.		

10
Bibliothèques et ports

Information sur la présence des cartes et les possibilités d'écrire		
Adresse HP-48 S/SX : #00735h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : après l'appel le quartet C(XS), c'est-à-dire 4 bits, résume les possibilités de lecture ou d'écriture sur les cartes. Le bit n°0 (bit de poids 2^0) de C(XS) vaut 1 s'il y a une carte en port 1. Le bit n°1 de C(XS) vaut 1 s'il y a une carte en port 2. Le bit n°2 de C(XS) vaut 1 si l'écriture est autorisée en port 1. Le bit n°3 vaut 1 si l'écriture est autorisée en port 2.		

Affectation à une bibliothèque d'un numéro inférieure à #7FFh		
Adresse HP-48 S/SX : #07BF0h	**Adresse HP-48 G/GX** : #07C0Bh	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTN
Remarques : placer avant l'appel le numéro de bibliothèque dans A(A). Après l'appel, A(A) contient le nouveau numéro de la bibliothèque, il s'agit d'un numéro normalisé inférieur à #7FFh. C(A) contient #007FFh.		

Recherche de l'adresse où se trouve les données d'une bibliothèque		
Adresse HP-48 S/SX : #0843Ah	**Adresse HP-48 G/GX** : #0843Ah	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : avant l'appel, placer dans A(X) le numéro de la bibliothèque. Après l'appel, D0 contient l'adresse du numéro de la bibliothèque. Si la bibliothèque de numéro recherché n'existe pas, c'est le numéro de la bibliothèque immédiatement suivant qui se trouve dans D0 (retour par RTNYES).		

Adresse de la hash-table ou message array d'une bibliothèque		
Adresse HP-48 S/SX : #08471h	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNC
Remarques : avant l'appel, A(X) contient le numéro de la bibliothèque concerné, ST(11) vaut 0 si on recherche une hash-table ou 1 si on recherche une message array. Après appel, A(A) contient l'adresse recherchée. Le retour s'effectue par RTNCC si la bibliothèque ne se trouve pas dans le répertoire HOME. Pour cela il faut que la bibliothèque existe.		

Adresse de la link-table ou du config object d'une bibliothèque		
Adresse HP-48 S/SX : #084DFh	**Adresse HP-48 G/GX** : N.C.	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : avant l'appel, A(X) contient le numéro de la bibliothèque concerné, ST(11) vaut 0 si on recherche une link-table ou 1 si on recherche un config object. Après appel, A(A) contient l'adresse recherchée. Le retour s'effectue par RTNYES si la bibliothèque n'existe pas, dans ce cas, A(A) vaut #00000h.		

CRC des données des ports 1 et 2		
Adresse HP-48 S/SX : #09B73h	**Adresse HP-48 G/GX** : #09B73h	
Appel : GOSBVL	**RSTK** : 0	**Retour** : RTNCC
Remarques : il n'est pas nécessaire d'affecter des valeurs particulières aux registres avant l'appel de cette routine. Après l'appel de la routine, les quatre premiers quartets du registre C contiennent le CRC. Attention ! Il s'agit ici d'un calcul particulier du CRC car le calculateur de CRC est ici initialisé à #FFFFh.		

Vérification du CRC d'une bibliothèque		
Adresse HP-48 S/SX : #0A01Ch	**Adresse HP-48 G/GX** : #0A01h	
Appel : GOSBVL	**RSTK** : 1	**Retour** : RTNCC
Remarques : avant l'appel, D0 contient l'adresse de la bibliothèque. Si le CRC est valide, A(A) contient #00000h. Si le CRC n'est pas valide, A(A) contient la différence entre les deux CRC, le retour s'effectue alors par RTNYES.		

26

Codes WSLOG

Le tableau ci-après résume les significations des 16 codes d'erreurs associés à WSLOG.

Codes	Erreurs WSLOG
0	Mise en route de la HP-48 alors que celle-ci se trouve en mode « coma ». L'horloge voit son fonctionnement interrompu alors que la consommation d'énergie est réduite au minimum.
1	Les piles sont trop faibles pour assurer un fonctionnement normal. Une vérification de VLB a lieu.
2	Un problème matériel a eu lieu durant un échange de données via le port infrarouge ou le port série (câble). Il est possible que l'UART soit saturé (UART est le sigle de « universal asynchronous receiver-transmitter ».
3	Il s'agit d'un redémarrage de la HP-48 depuis l'adresse #00000h.
4	Le décalage d'horloge est anormal, en effet, l'horloge affiche une date et une heure qui sont calculées par rapport à la date de début de la construction de la série des HP-48. Il s'agit du 1/1/1989 pour les HP-48 S/SX.
5	Les CRC (codes de redondance cyclique) des ports ont été modifiés et le contenu de la carte a été modifié.

Codes	Erreurs WSLOG
6	Code inutilisé.
7	La mémoire vive a peut-être été altérée.
8	Les périphériques, ou au moins l'un d'eux, ont une configuration anormale.
9	Il peut y avoir un « memory lost » (perte d'informations en mémoire). La récupération des informations peut ne pas être possible. La liste des sonneries programmées peut être altérée.
A	Code inutilisé.
B	Une carte se trouve dans l'un des ports (ou en a été retirée prématurément). Il est impossible d'établir un dialogue avec elle. Il peut s'agir d'un problème de connexion.
C	Initialisation (reset).
D	Il n'est pas possible de traiter une erreur, le programme nécessaire à ce traitement étant inaccessible.
E	Le CRC (code de redondance cyclique) des cartes placées dans les ports 1 et 2 n'est pas valide. La table de configuration n'est pas valide non plus.
F	La carte en port 1 ou 2 a été ôtée prématurément.

3

Listings commentés

Le chapitre 27 vous explique comment doivent être saisis les différents objets associés aux jeux présentés dans les chapitres 28 à 35.

Les chapitres 28 à 31 présentent des listings commentés de jeux pour les HP-48 S et SX

Les chapitres 32 à 35 proposent ces mêmes listings adaptés aux HP-48 G et GX. Toutefois, les programmes des chapitres 34 et 35 ne sont pas utilisables sur HP-48 GX.

Le chapitre 36 décrit la structure des différents programmes, et ce, afin de vous aider à les analyser.

Les versions originales des jeux ALPHA, STAR, XALK et SETH sont dues à Mathieu Guilmineau qui en a assuré le développement.

27

Présentation des jeux

1 Les jeux

Vous trouverez dans cette troisième partie les listings commentés de programmes à vocation ludique. Contrairement à la programmation en RPL, la programmation en assembleur tient compte des spécificités de chaque version de la HP-48. Ainsi, il n'est parfois pas suffisant de faire la distinction entre les deux familles S/SX et G/GX.

Nous vous proposons donc successivement :

- le jeu ALPHA pour la HP-48 S et la HP-48 SX,
- le jeu STAR pour la HP-48 S et la HP-48 SX,
- le jeu XALK pour la HP-48 S et la HP-48 SX,
- le jeu SETH pour la HP-48 S et la HP-48 SX,
- le jeu ALPHA pour la HP-48 G et la HP-48 GX,
- le jeu STAR pour la HP-48 G et la HP-48 GX,
- le jeu XALK pour la HP-48 G,
- le jeu SETH pour la HP-48 G.

Il s'agit donc de quatre jeux très différents les uns des autres. SETH est un classique jeu de course automobile. ALPHA, STAR, XALK et SETH vous demanderons de réfléchir (très) vite et bien (de préférence !). Pour une fois, les utilisateurs de HP-48 GX ne seront pas les plus favorisés puisque leurs superbes machines ne permettent pas une exécution normale des jeux XALK et SETH.

2
Exploitation des listings

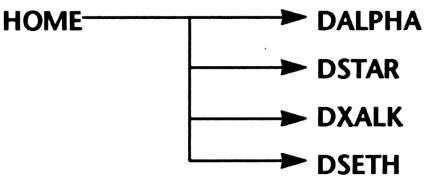
2.1
Création des répertoires nécessaires

L'utilisation de la pile, la création de répertoire, la navigation entre les répertoires et les autres opérations élémentaires à partir des variables globales doivent être connues. Ces manipulations de base ne seront pas décrites dans cet ouvrage. Si vous ne vous sentez pas sûr de vous (il n'y a pas de honte à ne pas savoir !), reportez vous au livre « HP-48 : permis de conduire » (DUNOD) qui vous permettra d'apprendre facilement toutes les manipulations élémentaires de votre HP-48.

Dans le répertoire racine (autrement dit, le répertoire HOME), créez :

- un répertoire DALPHA,
- un répertoire DSTAR,
- un répertoire DXALK,
- un répertoire DSETH.

Vous pourrez alors organiser de façon efficace le classement des différents objets. Tous les objets relatifs au jeu ALPHA devront être placés dans le répertoire DALPHA. De même, les objets relatifs à STAR seront placés dans DSTAR, les objets relatifs à XALK iront dans DXALK et ceux qui dépendent de SETH seront rangés dans DSETH. L'arborescence de votre répertoire HOME vous est présentée ci-dessous.



Seuls les utilisateurs très expérimentés et très sûrs d'eux pourront se permettre de modifier les noms des variables dans lesquelles vous devrez placer les objets associés aux différents jeux. A chaque objet correspond un fichier de même nom sur la disquette jointe.

2.2
Manipuler les variables

Chaque objet est stocké dans une variable globale par association d'un nom à un objet en appuyant sur la touche [STO]. En pratique :

- placez-vous dans le répertoire où devra être placée la variable,
- placez l'objet à mémoriser au niveau 1 de la pile,
- saisir le nom entre apostrophes (par exemple, 'nom'),
- appuyer sur [STO],
- le nom de la variable apparaît dans la barre de menu.

N'oubliez pas d'appuyer sur [VAR] pour afficher les variables du répertoire courant. Le chemin courant est affiché en haut à gauche de l'écran. La touche [NXT] permet de faire défiler les variables du répertoire courant.

Le chemin courant est la liste des répertoires ouverts pour accéder aux variable actuellement disponibles (affichées en appuyant sur [VAR]). Le répertoire courant est celui dont les variables sont affichées en appuyant sur [VAR]. Le répertoire courant est donc le dernier du chemin courant.

Une pression sur une touche blanche placée sous l'un des articles de la barre de menu appelle la variable correspondante. S'il s'agit d'une donnée « inerte », par exemple une chaîne de caractères, la chaîne de caractères sera envoyée dans la pile. Par contre, s'il s'agit d'un programme (RPL ou code exécutable), ce programme sera exécuté. Pour envoyer dans la pile le nom d'une variable, appuyez sur ['] puis sur la touche correspondant à l'article du menu relatif à la variable choisi.

Pour modifier une variable (autrement dit, pour l'éditer), il suffit d'appuyer sur [↱] puis sur la touche blanche relative à la variable choisie et enfin sur la touche [↓]. Une fois vos modifications effectuées, appuyez sur [ENTER] pour les valider et envoyer l'objet modifié au niveau 1 de la pile. Attention ! La variable reste associée à la précédente version de l'objet. Il est donc indispensable d'effectuer une nouvelle opération d'enregistrement. Pour cela, appuyez sur ['] puis sur la touche blanche relative à la variable (le nom de la variable se place au niveau 1 de la pile) puis appuyez sur [STO]. L'ancienne version sera « écrasée » car on ne peut avoir deux variables portant le même nom dans un même répertoire.

Nous allons utiliser deux catégories de variables : les variables de développement et les variables utilisées pendant l'exécution du jeu. Les premières servent à générer les secondes...

2.3
Générer les objets CODE avec ASS

La création des codes exécutables avec ASS est pour nous l'occasion de réviser quelques bases relatives à la saisie des objets. Messieurs les initiés, ce livre doit pouvoir être lu par ceux qui disposent de leur HP-48 depuis peu de temps, veuillez donc pardonner un afflux de détails qui vous sembleront peut être inutiles...

Le programme écrit en assembleur sera saisi sous la forme d'une chaîne de caractères ne contenant que les caractères 0 à 9 et A à F. Il faudra placer cette chaîne au niveau 1 de la pile et exécuter le programme ASS pour générer le code exécutable. Ci-dessous, le listing de ASS :

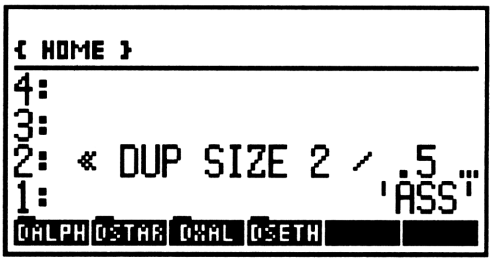
```
« DUP SIZE 2 / .5
+ IP "GROB 1 "
SWAP + " "
+ SWAP + OBJ→
#4017h SYSEVAL
#56B6h SYSEVAL
DROP NEWOB »
```

Ce programme doit être enregistré sous le nom 'ASS' et gagne à être placé dans le répertoire HOME. Il se trouve sur la disquette sous le nom ASS. Rappelons que pour saisir les instructions d'un programme RPL tel que ASS, vous pouvez, soit sélectionner les instructions dans les sous-menus accessibles par la touche [PRG] (et autres), soit saisir caractère par caractère les instructions.

Votre machine doit se trouver en mode hexadécimal ! Pour cela, choisir l'article *hex* dans le sous-menu *base* accessible grâce à la touche [MTH].

Une fois validé par une pression sur [ENTER], ASS se trouve au niveau 1 de la pile.

On introduit ensuite 'ASS' [ENTER] qui se place à son tour au niveau 1 et refoule l'objet programme RPL au niveau 2.



Il ne reste ensuite qu'à appuyer sur [STO] pour associer le nom 'ASS' au programme RPL, on crée ainsi une variable qui se placera dans le répertoire courant.

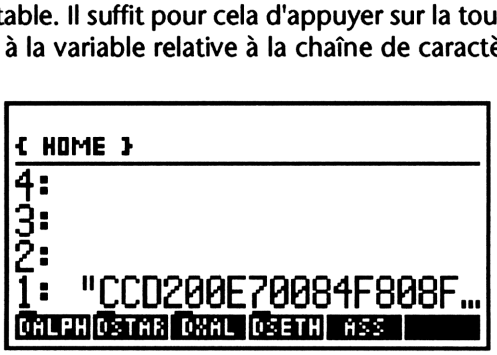
■ Le rôle du programme ASS

Le programme ASS transforme une chaîne de caractères en un code exécutable. La chaîne de caractères n'est pas quelconque. Elle ne comporte que des chiffres (0 à 9) et des lettres majuscules (A à F) qui sont les représentations directes de nombres hexadécimaux. Cette chaîne ne comporte aucun espace et est placée entre " et " qui sont les délimiteurs normaux d'une chaîne de caractères.

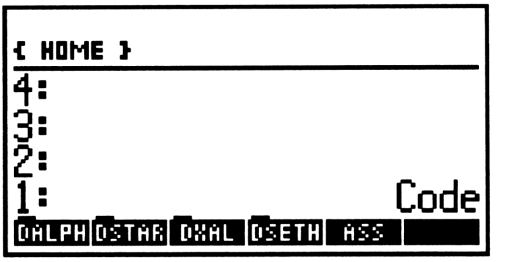
A partir de la chaîne de caractères (chaîne de codes hexadécimaux), le programme ASS génère un code exécutable qui s'affiche sous la forme de **code** au niveau 1 de la pile.

■ Un exemple d'utilisation du programme ASS

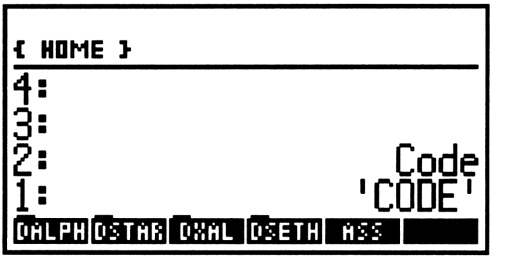
Commencez par placer dans la pile la chaîne de caractères à transformer en code exécutable. Il suffit pour cela d'appuyer sur la touche de sélection correspondant à la variable relative à la chaîne de caractères.



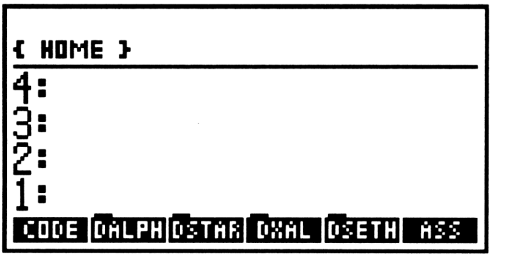
CCD 20 Choisissez ensuite ASS dans le menu. La chaîne est remplacée dans la pile par **Code** qui représente un code exécutable que vous ne pouvez pas éditer directement.



Saisissez 'CODE' qui est le nom de la variable qui contiendra le code exécutable. Appuyez sur [STO] pour lier le nom 'CODE' à l'objet code exécutable et créer une variable nommée CODE.



La nouvelle variable apparaît dans le menu affiché après une pression sur [VAR].



Dans nos différents exemples, les codes exécutables sont à placer dans des variables nommées 'CODE'. Le nom du jeu, par exemple, STAR, est généralement réservé au lanceur qui est un programme écrit en RPL chargé d'empêcher un lancement intempestif du code exécutable (le jeu en assembleur). Le lanceur sert aussi dans certains cas à l'introduction de variable dans la pile.

2.4
Les objets

Tenez compte des quelques recommandations énoncées ci-dessous. L'informatique est une science exacte, un symbole inadapté peut donc être responsable du non-fonctionnement d'un programme.

■ Les délimiteurs

Les types des différents objets que vous devrez manipuler dans le cadre des exemples des chapitres suivants sont :

- programme en RPL,
- chaîne de caractères,
- code exécutable (programme en langage machine),
- nom,
- objet graphique (GROB).

Les chaînes de codes hexadécimaux sont du type « chaîne de caractères » et sont destinées à être traitées par le programme ASS pour créer un objet de type code exécutable.

Les objets de type nom servent juste à l'enregistrement d'un objet sous la forme d'une variable portant le nom spécifié par l'objet de type nom.

Les objets graphiques (GROB) ont la structure suivante :

- GROB,
- un espace suivi d'un entier,
- à nouveau, un espace suivi d'un entier,
- un nouvel espace suivi d'une chaîne d'entiers ininterrompue.

Mis à part ces trois espaces, aucun espace ni retour à la ligne ne doit être introduit dans un GROB.

Les délimiteurs que vous utiliserez sont les suivants :

Objet	Délimiteurs
Nom	' *et* '
Programme RPL	« *et* »
Chaîne de caractères	" *et* "
Entier système (ici, hexadécimal)	# *et* h

■ Attention aux espaces !

Vous pouvez placer dans un programme RPL autant d'espaces consécutifs que vous le souhaitez, cependant il suffit d'introduire un unique espace là où celui-ci est indispensable dans le rôle de séparateur. Dans le cadre d'un programme RPL, la même remarque s'applique aux sauts de ligne. L'élimination des espaces et sauts de lignes inutiles est automatique dès que le programme est validé par une pression sur [ENTER]. Par contre, une chaîne de codes hexadécimaux ne doit comporter aucun espace ni saut de ligne. De même un objet graphique (GROB) ne doit comporter aucun saut de ligne et aucun espace si on excepte les trois espaces utilisés comme séparateurs au début du GROB.

■ Les objets à compiler

Les chaînes de codes hexadécimaux sont du type chaîne de caractères. Elles doivent impérativement être placées entre " et ". Elles constituent les arguments du programme ASS.

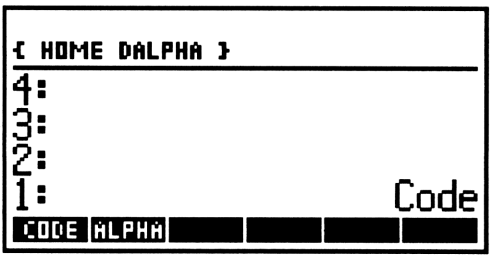
■ Les objets graphiques

Les objets graphiques (GROB) sont à saisir sans espace ni saut de ligne. Seuls trois espaces sont utilisés comme séparateurs au début du GROB, les autres espaces devront être ignorés. Prenons un exemple :

```
GROB 1 35 7555557322222774471177446447555744471174477
11755774422227557557755744 7
```

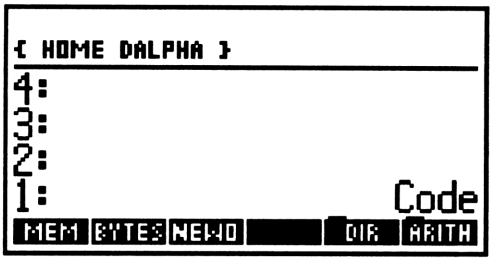
■ Longueur et CRC

La commande BYTES nous fournit un moyen simple et fiable pour vérifier qu'un objet a été saisi correctement. Commencez par placer l'objet au niveau 1 de la pile (en appuyant d'abord sur [↱] pour ne pas évaluer l'ob-

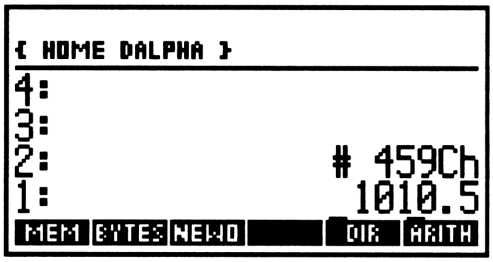


jet) puis sur la touche blanche correspondant à la variable stockant l'objet à vérifier. Dans le cadre de notre exemple, nous avons saisi [↱] [A].

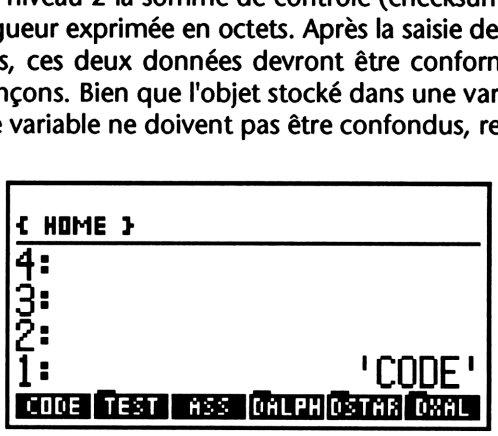
Attention ! Il ne faut pas confondre l'objet en lui-même et le nom de la variable qui l'accueille ! Dans le cadre de notre exemple, [↱] [A] renvoie l'objet dans la pile (`Code` s'affiche au niveau 1 de la pile), alors que ['] [A] renvoie le nom de la variable contenant l'objet qui est ici un code exécutable (`'CODE'` s'affiche au niveau 1 de la pile).

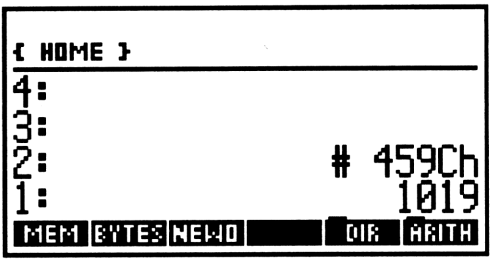


Saisissez [↰] [VAR] pour afficher le menu où se trouve la commande BYTES. Appuyez sur [B] pour activer BYTES qui prend en argument l'objet placé au niveau 1.



On retrouve au niveau 2 la somme de contrôle (checksum ou CRC) et au niveau 1 la longueur exprimée en octets. Après la saisie des exemples que nous proposons, ces deux données devront être conformes aux valeurs que nous annonçons. Bien que l'objet stocké dans une variable globale et le nom de cette variable ne doivent pas être confondus, remarquez qu'on





obtient la même somme de contrôle (CRC ou checksum) en donnant le nom de la variable à tester en argument à la commande BYTES. Cependant, la longueur annoncée est différente.

3
Présentation des listings

Adresse	Codes hexadécimaux	Mnémoniques		Renvois
	CCD20			❶
#000h	01000			❷
	143	A=DAT1	A	❸
	324A2	C=(3)	#2A4	
#00Dh	932...	?A=C...	X...	❹

❶ Prologue de l'objet de type code exécutable.
❷ Longueur hors prologue du code exécutable (non significative dans le cadre de cet exemple...).
❸ Lecture du niveau 1 de la pile.
❹ Test de la valeur au niveau 1 de la pile.

Afin de faciliter la lecture des programmes, nous avons choisi de les présenter dans des tableaux. La première colonne contient des adresses. L'adresse #000h correspond à l'adresse du premier des cinq quartets codant la longueur hors prologue du code exécutable. Il s'agit donc d'un adressage relatif (cela dit, le compteur d'adresses est parfois remis à zéro en plein milieu du code, en particulier, au début d'une zone où sont

codées des routines). La deuxième colonne contient les codes hexadécimaux qui, une fois juxtaposés, constituent la chaîne de codes hexadécimaux à traiter avec ASS pour obtenir le code exécutable proprement dit. Dans la troisième colonne, vous trouverez les mnémoniques correspondant aux codes hexadécimaux de la deuxième colonne. La dernière colonne vous renvoie au remarque du bas de la page à l'aide de « pastilles » ❶, ❷, ❸, etc. Précisons maintenant quelques points essentiels.

■ Les procédures

Au sein du code, une procédure est une portion de code remplissant une tâche précise. Les procédures sont notées **`$nom_de_la_procédure`**, on aura ainsi la procédure **`$SCRASH`** et la procédure **`$VOITURE`** au sein du code de SETH version S.

■ Les points de branchements

C'est à partir d'un « point de branchement » que reprendra l'exécution du programme à l'issue d'un saut ayant branché l'exécution sur ledit point de branchement. Les points de branchements sont notés **`%numéro`**, on aura ainsi **`%1`**, **`%2`**, **`%3`**, etc.

Attention ! Les points de branchements sont propres à une procédure. La numérotation des points de branchements reprend donc à **`%1`** à chaque nouvelle procédure.

■ « Pointer ! »

L'expression « pointer sur une adresse » signifie que le registre dont il est question contient une adresse. On dit donc que ledit registre pointe vers une adresse (cette adresse peut correspondre à une procédure, à un point de branchement, etc.).

■ Les sprites

Un sprite est un objet graphique se déplaçant ou non sur l'écran mais qui peut apparaître ou disparaître de l'écran sans porter atteinte au reste du contenu de l'écran. Le codage des sprites peut être incorporé au code ou placé dans un objet graphique (GROB) indépendant du code.

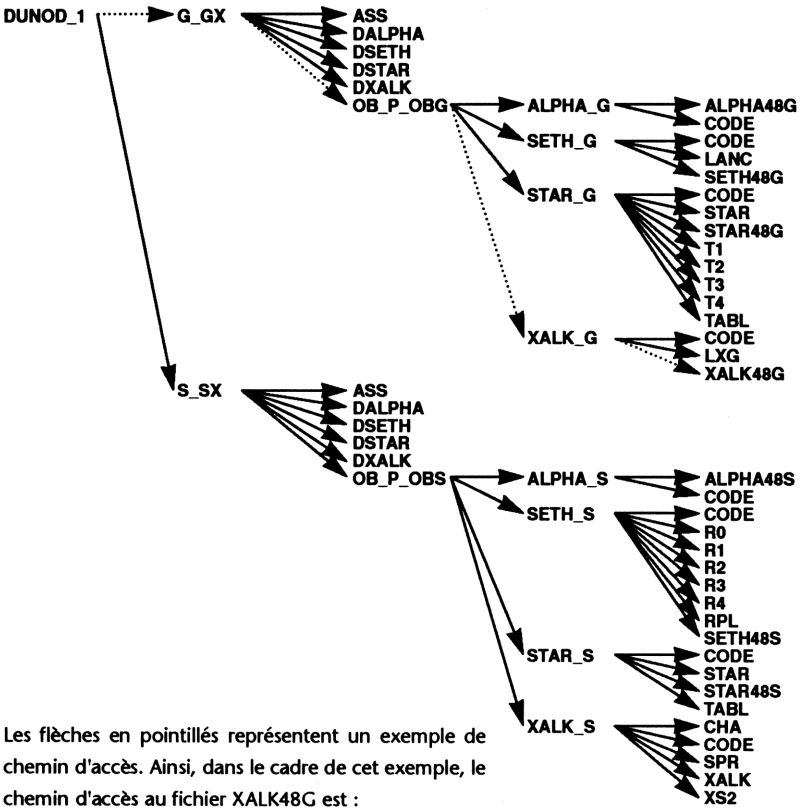
■ Attention ! Copie interdite !

Rappelons que les programmes présentés dans ce livre ainsi que le contenu du répertoires DUNOD_1 ne sont pas du domaine public, leur diffusion sans l'accord de *DUNOD Editeur* est donc rigoureusement interdite. En effet, leur utilisation impose l'achat du présent livre.

4

Utiliser la disquette jointe

Le schéma ci-dessous présente l'accès aux fichiers relatifs aux jeux proposés dans cette partie de l'ouvrage.



Les flèches en pointillés représentent un exemple de chemin d'accès. Ainsi, dans le cadre de cet exemple, le chemin d'accès au fichier XALK48G est :
DUNOD_1\G_GX\OB_P_OBG\XALK_G\XALK48G.

Selon le type de votre machine, choisissez les fichiers du répertoire G_GX ou ceux du répertoire S_SX. Comme nous l'avons dit au début de ce chapitre, certains programmes du répertoire G_GX ne fonctionnent correctement que sur HP-48G.

Le programme ASS est accessible par DUNOD_1\G_GX\ASS ou par DUNOD_1\S_SX\ASS. Les deux copies présentes de ASS sont absolument identiques.

Pour chaque jeu, vous pouvez choisir entre l'importation de toutes les variables en une seule opération et l'importation de chaque variable une à une. Les fichiers DALPHA, DSETH, DSTAR et DXALK correspondent à des répertoires HP-48 contenant toutes les variables relatives à un jeu. Dans OB_P_OBG et dans OB_P_OBS vous trouverez chaque objet de chaque jeu dans un fichier particulier. Les objets relatifs à un jeu donné sont regroupés dans un fichier portant le nom du jeu choisi.

Parmi toutes les variables, il faut distinguer les variables de service et les variables utilisées pendant l'exécution du programme. La chaîne de codes hexadécimaux à traiter avec ASS est une variable de service alors que le code exécutable (toujours nommé 'CODE') est une variable utilisée pendant l'exécution du programme.

Tous les fichiers du répertoire DUNOD doivent être transférés en mode binaire.

28

Alpha pour HP-48 S et SX

1

Le jeu

1.1

Présentation

ALPHA est celui de nos jeux dont la mise en œuvre est la plus simple. En effet, il suffit pour l'utiliser d'évaluer l'objet code exécutable.

ALPHA allie réflexion et réactions rapides. Vous jouez avec une sphère couverte de 256 figures (triangles, sabliers, papillons, carrés, etc.). Seules neuf pièces sont visibles simultanément : une pièce centrale et les huit qui l'entourent. La pièce centrale peut agir sur les autres pour créer un carré.

ALPHA est un intéressant jeu de réflexion « contre la montre », son but est la reconstitution - en moins de deux heures ! - de 400 carrés en juxtaposant deux triangles (orientés convenablement) ou en associant un sablier à un « nœud papillon ». La juxtaposition de deux pièces provoque la libération d'une case, rendant ainsi visible une seconde couche de pièces juxtaposables ! Seul un carré (que vous aurez reconstitué...) peut migrer vers cette seconde couche de pièces et permettre, par échange des positions des figures, la manipulation des figures de la seconde couche...

1.2
Liste des objets

Trois objets sont associés à la mise en œuvre de ce jeu :

- 'ALPHA48S' est la chaîne de codes hexadécimaux permettant de générer 'CODE',
- 'CODE' est le code exécutable généré en traitant la chaîne 'ALPHA48S' à l'aide de 'ASS',
- 'ASS' est le programme RPL permettant de générer l'objet 'CODE' à partir de 'ALPHA48S'.

En fait, seul 'CODE' est nécessaire à l'exécution du programme. C'est en évaluant 'CODE' que l'on exécute le programme.

2
Les objets

2.1
'ALPHA48S' et 'ASS' génèrent 'CODE'

Ci-dessous le code hexadécimal à saisir dans une chaîne de caractères, c'est-à-dire entre les délimiteurs " et ". Cette chaîne de caractères doit être enregistrée dans une variable nommée 'ALPHA48S'.

Cette chaîne ne doit contenir que les chiffres 0 à 9 et les lettres A à F. Attention ! Il ne faut surtout pas saisir les espaces et les sauts de ligne qui ne sont là que pour faciliter la lecture.

```
CCD20 0E700 84F80 8F082 01BC0 100D2
15C01 9E215 C0185 CE15C 0D1D3 81823
31F00 20716 F14AC AD680 84280 85314
9170B 6555E 03400 00214 51E2D 6014F
808B3 50850 1EE05 01431 01D23 1B8CA
```

```
10011 1130A F0317 81507 16FA6 E55F1
10130 33F02 006A6 913D3 44A50 081B4
CA842 D215F 0808B 35085 28088 3808A
273F2 C6CA1 311C7 30E14 31583 17380
8B350 1C716 F16F1 61A0E 5FD4D 48438
08B0F 00631 83CA0 78538 08A18 031C3
CA131 30114 31583 17386 3501C 716F1
6F161 B0651 E8625 413A3 33420 EA138
31212 515A5 A74B9 C81C1 58516 F16F1
61A6E 52E20 13A32 340EA 13813 A338F
10EA1 3807B 66DA3 02906 B2818 68F81
86823 1FD96 2C1D6 06316 913AC A1380
7DAD6 6BBE1 1033B 850CA 138DB DA302
76437 CA232 E008F CEE10 93A50 5AE74
92328 008FC EE108 08A49 21B00 100D2
15C03 20048 FCEE1 0808A FFE30 815C0
32400 8FCEE 10808 A4806 AE400 32040
8FCEE 10808 B0F08 08A2B 08405 50850
D032E 008FC EE108 08A17 0E459 1808A
370CC 4D080 8B260 685F3 22008 FCEE1
090A8 08186 8F328 008FC EE109 0A808
1860F 968BC 84434 00207 AE913 4A621
3514B 30C90 68085 45123 04906 D1854
14A80 84330 59026 067CE 6FD01 4E808
43808 83906 01854 30590 2C25E DCA30
29022 13049 02A03 0D906 4C05C F0456
06AC3 1321B 83100 14E13 08168 08828
08831 4A808 73E03 04A46 45030 C15C0
30586 4C0E6 A4644 0E614 B8086 37080
89315 D0860 33307 06D0D 22880 F2801
AE681 862B5 9F818 60156 E0720 A0E54
D4C31 4A14F C4C6A 8E81C 81E15 8015D
0874C 18084 33059 02F31 4B808 43902
23137 135D5 860B4 D0223 08801 D2801
B665C F8186 0758E 20482 86032 D0D22
880F2 801AE 6B665 CF818 60259 E206C
CB1BB 31001 E0030 14A80 86080 8415D
38718 38511 4B05A 0C041 59054 21331
31808 60903 0515D 01708 08623 D6F42
11034 6D100 CA130 1D001 4B301 71201
801E2 03014 B3017 E0018 01E40 3014B
D2102 06D23 1D481 B4CAD 281A0 1ACAC
AF2CA 13130 814B1 4816F 16F16 1171A
0E5AE 13A33 4310E A1381 12F40 7A0E5
```

```
8A01E 11212 12121 21212 E180C 08080
80808 080C1 E1121 20180 40201 0F3E1
12020 2C102 0212E 10181 41211 1F301
0101F 31010 F1020 20212 E1E11 21010
F1121 212E1 F3020 20180 40404 040E1
12121 2E112 1212E 1E112 1212E 30202
12E11 00030 00700 0F000 7100D 200D7
005B0 0D610 DF305 B50D6 B0DFF 130C3
FFF7F FF3E1 06CFF 586B5 0D650 EF504
B5086 500F5 00A50 04700 87000 70006
00040 00000 00000 00000 00000 00000
00000 0FFF7 5145B FF650 05700 75005
50055 00530 06700 7D085 B1C67 36756
353D5 67B67 FFF7B DD66B 63C08 18DD0
0B600 6300C 10850 45084 032FF 78FCE
E1093 E1F32 FF78F CEE10 93A80 D281B
31113 17813 01527 86080 A74B2 4BFC1
50716 FA6E5 3E340 00308 60803 30001
CE5DF 48A
```

Cette chaîne est à placer dans une variable nommée 'ALPHA48S'. Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à 8D0Ah,
- au niveau 1, une longueur égale à 2022 octets.

La chaîne de codes hexadécimaux 'ALPHA48S' est à placer au niveau 1 de la pile. Activez le programme 'ASS', vous obtenez alors au niveau 1 de la pile un objet code exécutable à sauvegarder dans une variable 'CODE'. Activez la commande BYTES en donnant pour argument à cette dernière une copie du code exécutable. Vous devez obtenir :

- au niveau 2, une somme de contrôle égale à 1BDAh,
- au niveau 1, une longueur égale à 1010,5 octets.

2.2

Le code hexadécimal commenté

Les pages qui suivent vous présentent commande par commande le programme ALPHA pour HP-48 S et SX. Rappelons que la première colonne comporte des adresses utilisées pour les sauts relatifs. La dernière colonne renvoie à la remarque de bas de page.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	CCD20			❶
#000h	0E700			❷
	84F	ST=0	15	❸
	808F	INTOFF		
	08	CLRST		
	20	P=	0	
	1BC0100	D0=(5)	#0010C	❹
	D2	C=0	A	
	15C0	DAT0=C	1	
	19E2	D0=(2)	#2E	❺
	15C0	DAT0=C	1	
	185	D0=D0-	#6	
	CE	C=C-1	A	❻
	15C0	DAT0=C	1	
	D1	B=0	A	❼
	D3	D=0	A	
	818233	D=D+#4	XS	❽
	1F00207	D1=(5)	#70200	
	16F	D0=D0+	#10	
#042h	14A	A=DAT0	B	❾
	CA	A=A+C	A	
	D6	C=A	A	❿
	80842	ABIT=0	2	
	80853	ABIT=1	3	
	149	DAT1=A	B	
	170	D1=D1+	#1	

❶ CCD20 est le prologue d'un objet de type code exécutable.
❷ 0E700 correspond au codage sur 5 quartets de la longueur (hors prologue) du code qui contient #7E0h quartets. C'est à partir de cette adresse que l'on balise le code avec des adresses utilisées pour les sauts relatifs.
❸ Inhibition des interruptions (la touche [ON] est donc désactivée).
❹ Suppression du petit sablier (témoin de processus en cours).
❺ Encore moins d'interruptions !
❻ Suppression des menus horizontaux en bas de l'écran.
❼ Le registre B sert de compteur.
❽ Le registre D compte le nombre de carrés à créer : encore 400 ! On pointe ensuite sur le début du tableau, puis sur l'horloge. On lit celle-ci.
❾ Etiquette %1 en #042h vers lequel branche Gonc %1.
❿ Encore plus de hasard ! Que des triangles (bit 2)... sur la première couche (bit 3), on mémorise la case... Puis on passe à la case suivante...

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	B65	B=B+1	B	
	55E	GONC	%1	❶
	3400002	C=(5)	#20000	
	145	DAT1=C	A	❷
	1E2D60	D1=(4)	#06D2	
	14F	C=DAT1	B	
	808B3	?CBIT=1	3	❸
	50	GOYES	%2	
	850	ST=1	0	❹
#07Ch	1EE050	D1=(4)	#050E	❺
	143	A=DAT1	A	❻
	101	R1=A		
	D2	C=0	A	
	31B8	C=(2)	#8B	❼
	CA	A=A+C	A	
	100	R0=A		
#093h	111	A=R1		❽
	130	D0=A		
	AF0	A=0	W	
	3178	C=(2)	#87	
$0				
#0A0	1507	DAT0=A	W	❾
	16F	D0=D0+	#10	❿
	A6E	C=C-1	B	
	55F	GONC	%1	

❶ Boucle (256 fois) vers #042h.
❷ Mémorisation du chronomètre.
❸ On test le flag (drapeau) signal sonore (beep). Attention ! Il s'agit du flag RPL et non d'un drapeau interne du SATURN. Selon le résultat du test, on saute ou non vers #07Ch.
❹ Si le flag beep est actif, on a ST0=1.
❺ L'adresse #07C constitue l'étiquette de destination %2.
❻ Lecture de la bitmap écran dont le début est dans le registre R1.
❼ On pointe sur le troisième quartet de la quatrième ligne. Le premier sprite est placé dans R0.
❽ La procédure $0 débute en #093h. On efface l'écran (D0 désigne le début de l'écran).
❾ L'adresse #0A0h est le point de branchement %1 de la procédure $0.
❿ On efface tout l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#0ADh	110	A=R0		❶
	130	D0=A		
	33F020	C=(4)	#020F	
#0B9h	06	RSTK=C		❷
	A69	C=C+B	B	
	13D	D1=CS		❸
	344A500	C=(5)	#005A4	
	81B4	A=PC		
#0CCh	CA	A=A+C	A	
	842	ST=0	2	
	D2	C=0	A	
	15F0	C=DAT1	1	❹
	808B3	?CBIT=1	3	
	50	GOYES	%3	
	852	ST=1	2	❺
#0E1h	80883	CBIT=0	3	❻
	808A2	?CBIT=0	2	
	73	GOYES	%6	❼
#0EDh	F2	CSL	A	
	C6	C=C+C	A	
	CA	A=A+C	A	
	131	D1=A		❽
	1C7	D1=D1-	#8	
	30E	C=(1)	#E	❾
#0FCh	143	A=DAT1	A	❿

❶ En #0ADh commence la procédure de lecture des cases. D0 pointe sur la première case puis on désigne le premier sprite dans le tableau.
❷ L'adresse #0B9h est le point de branchement %2 de la procédure. Cette adresse correspond aussi à la sauvegarde du compteur.
❸ On calcule l'emplacement dans le tableau. On pointe ensuite sur la procédure $SPRITE.
❹ Lecture de la case à afficher.
❺ S'agit-il de la première couche ?
❻ L'adresse #0E1h est le point de branchement %3 de la procédure.
❼ S'agit-il d'un triangle ? Si oui, on saute vers le point de branchement %6 (en #122h). Le traitement des figures autres que les triangles suit.
❽ On pointe sur les données relatives au sprite
❾ Compteur.
❿ L'adresse #0FCh est le point de branchement %4 de la procédure.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1583	DAT0=A	4	❶
	173	D1=D1+	#4	❷
	808B3	?CBIT=1	3	
	50	GOYES	%5	❸
	1C7	D1=D1-	#8	
#110h	16F	D0=D0+	#10	❹
	16F	D0=D0+	#10	
	161	D0=D0+	#2	
	A0E	C=C-1	P	❺
	5FD	GONC	%4	
	4D4	GOC	%11	
#122H	843	ST=0	3	❻
	808B0	?CBIT=1	0	
	F0	GOYES	%7	
	06	RSTK=C		
	3183	C=(2)	#38	❼
	CA	A=A+C	A	
	07	C=RSTK		
	853	ST=1	3	
#139h	808A1	?CBIT=0	1	❽
	80	GOYES	%8	
	31C3	C=(2)	#3C	
	CA	A=A+C	A	
#146h	131	D1=A		❾
	301	C=(1)	#1	
#14Ch	143	A=DAT1	A	❿
	1583	DAT0=A	4	
	173	D1=D1+	#4	
	863	?ST=0	3	

❶ Affichage d'une ligne du sprite.
❷ Passage à la donnée suivante.
❸ Décompactage en fonction du compteur.
❹ Point de branchement %5 de la procédure.
❺ Boucle sur 15 lignes.
❻ Point de branchement %6 de la procédure.
❼ Réajustement en fonction du triangle.
❽ Point de branchement %7 de la procédure.
❾ Point de branchement %8 de la procédure.
❿ Point de branchement %9 de la procédure.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	50	GOYES	%10	
	1C7	D1=D1-	#8	
#15Eh	16F	D0=D0+	#10	❶
	16F	D0=D0+	#10	
	161	D0=D0+	#2	
	B06	C=C+1	P	❷
	51E	GONC	%9	
#16Dh	862	?ST=0	2	❸
	54	GOYES	%13	
	13A	AD0xs		
	333420	C=(4)	#0243	❹
	EA	A=A-C	A	
	138	D0=AS		
	3121	C=(2)	#12	❺
	25	P=	5	
#186h	15A5	A=DAT0	6	❻
	A74	A=A+A	W	
	B9C	A=-A-1	WP	❼
	81C	ASRB		
	1585	DAT0=A	6	❽
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❾
	161	D0=D0+	#2	
	A6E	C=C-1	B	
	52E	GONC	%12	
	20	P=	0	
#1A8h	13A	AD0xs		
	32340	C=(3)	#043	❿

❶ Passage à la ligne suivante.
❷ Boucle sur 15 lignes.
❸ Le label %11 est en #16D (début de la procédure d'inversion vidéo).
❹ On repointe en haut puis on retire 2 lignes et 1 quartet.
❺ Compteur.
❻ Le label %12 est en #186h. Lecture de l'écran.
❼ Inversion vidéo sur 23 bits.
❽ Ecriture écran.
❾ Passage à la ligne inférieure puis boucle sur 19 lignes (2+15+2).
❿ Et on repointe comme si de rien n'était !

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	EA	A=A-C	A	
	138	D0=AS		
#1B5h	13A	AD0xs		❶
	338F10	C=(4)	#01F8	❷
	EA	A=A-C	A	
	138	D0=AS		
	07	C=RSTK		
	B66	C=C+1	B	❸
	DA	A=C	A	
	302	C=(1)	#2	
	906	?A≠C	P	❹
	B2	GOYES	%14	
	81868F	A=A-#10	B	❺
	818682	A=A-#3	B	
	31FD	C=(2)	#DF	
	962	?A=C	B	❻
	C1	GOYES	$1	
	D6	C=A	A	
	06	RSTK=C		
	3169	C=(2)	#96	❼
	13A	AD0xs		❽
	CA	A=A+C	A	
	138	D0=AS		
	07	C=RSTK		
	DA	A=C	A	
#1FBh	D6	C=A	A	❾
	6BBE	GOTO	%2	
#201h	110	A=R0		❿
	33B850	C=(4)	#058B	

❶ Point de branchement %13 de la procédure.
❷ On pointe sur le sprite suivant (en restant sur la même ligne).
❸ On cherche le suivant dans le tableau.
❹ S'agit-il de la fin de la ligne affichée ?
❺ Passage à la ligne suivante dans le tableau.
❻ Hors de l'écran.
❼ On ajoute #0296h.
❽ On pointe ainsi sur la ligne suivante.
❾ Passage au sprite suivant.
❿ Procédure $1 relative aux triangles restants.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	CA	A=A+C	A	
	138	D0=AS		
	DB	C=D	A	❶
	DA	A=C	A	
	302	C=(1)	#2	
	7643	GOSUB	$AFF	
#21Ah	7CA2	GOSUB	$CHRONO	❷
#21Eh	32E00	C=(3)	#00E	
	8FCEE10	GOSBVL	#01EEC	❸
	93A	?C=0	X	
	50	GOYES	$3	
	5AE	GONC	$2	
#232h	7492	GOSUB	$CHRONO	❹
#236h	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	
	808A4	?CBIT=0	4	❺
	92	GOYES	%2	
	1B00100	D0=(5)	#00100	
	D2	C=0	A	❻
	15C0	DAT0=C	1	
#256h	32004	C=(3)	#400	❼
	8FCEE10	GOSBVL	#01EEC	
	808AF	?CBIT=0	15	
	FE	GOYES	%1	
	308	C=(1)	#8	❽
	15C0	DAT0=C	1	
#270h	32400	C=(3)	#004	❾
	8FCEE10	GOSBVL	#01EEC	

❶ On charge dans le registre A le nombre de triangles restants. On affiche ce nombre (sur trois chiffres).
❷ Début de la procédure $2 (attente de relâchement).
❸ Attente du relâchement des touches.
❹ Début de la procédure $3 (test des touches).
❺ Test : pression sur [SIN] ? So oui, on provoque une « pause ».
❻ Extinction de l'écran.
❼ Point de branchement %1. Attente d'une pression sur [ON].
❽ Allumage de l'écran.
❾ Point de branchement %2 de la procédure.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	808A4	?CBIT=0	4	❶
	80	GOYES	%3	
	6AE4	GOTO	$PERDU	
	00	RTNSXM		
#289h	32040	C=(3)	#040	❷
	8FCEE10	GOSBVL	#01EEC	
	808B0	?CBIT=1	0	❸
	F0	GOYES	%4	
	808A2	?CBIT=0	2	
	B0	GOYES	%5	
	840	ST=0	0	
	550	GONC	%5	
#2A9h	850	ST=1	0	❹
#2ACh	D0	A=0	A	❺
	32E00	C=(3)	#00E	
	8FCEE10	GOSBVL	#01EEC	❻
	808A1	?CBIT=0	1	
	70	GOYES	%6	
	E4	A=A+1	A	
	591	GONC	%9	
#2C6h	808A3	?CBIT=0	3	❼
	70	GOYES	%7	
	CC	A=A-1	A	
	4D0	GOC	%9	
#2D2h	808B2	?CBIT=1	2	❽
	60	GOYES	%9	
#2D9h	685F	GOTO	$3	❾
#2DDh	32200	C=(3)	#002	❿
	8FCEE10	GOSBVL	#01EEC	

❶ Test d'une pression sur [MTH], si oui, fin du jeu.
❷ Point de branchement %3.
❸ Test d'une pression sur [←] (flag beep off) ou sur [→] (flag beep on).
❹ Point de branchement %4.
❺ Point de branchement %5.
❻ Test du pavé numérique. Déplacement à droite ?
❼ Point de branchement %6. Déplacement à gauche ?
❽ Point de branchement %7. Il ne s'agit pas d'un déplacement latéral.
❾ Point de branchement %8. Avait-on appuyé sur une touche ?
❿ Point de branchement %9.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	90A	?C=0	P	❶
	80	GOYES	%10	
	81868F	A=A-#10	B	
#2F4h	32800	C=(3)	#008	❷
	8FCEE10	GOSBVL	#01EEC	
	90A	?C=0	P	❸
	80	GOYES	%11	
	81860F	A=A+#10	B	
#30Bh	968	?A=0	B	❹
	BC	GOYES	%8	
#310h	844	ST=0	4	❺
	3400207	C=(5)	#70200	
	AE9	C=B	B	
	134	D0=C		❻
	A62	C=C+A	B	
	135	D1=C		
#326h	14B	A=DAT1	B	❼
	30C	C=(1)	#C	
	906	?A≠C	P	
	80	GOYES	%1	
	854	ST=1	4	
	512	GONC	%2	
#337h	304	C=(1)	#4	❽
	906	?A≠C	P	❾
	D1	GOYES	%3	
	854	ST=1	4	❿
	14A	A=DAT0	B	
	80843	ABIT=0	3	

❶ S'agit-il d'un déplacement vers le bas ?
❷ Point de branchement %10.
❸ S'agit-il d'un déplacement vers le haut ?
❹ Point de branchement %11. Annulation si pression sur %5.
❺ Procédure $4. Pas de déplacement et carré (assemblage).
❻ Case centrale dans D0, case de destination dans D1.
❼ Case arrivée dans le registre A.
❽ Point de branchement %1 (procédure d'initialisation des pointeurs).
❾ Pièce blanche ?
❿ Déplacement, départ dans A sans tenir compte de la couche.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	305	C=(1)	#5	
	902	?A=C	P	❶
	60	GOYES	%2	
	67CE	GOTO	$2	
#356h	6FD0	GOTO	%14	❷
#35Ah	14E	C=DAT0	B	❸
	80843	ABIT=0	3	
	80883	CBIT=0	3	❹
	906	?A≠C	P	
	01	GOYES	%4	❺
	854	ST=1	4	
	305	C=(1)	#5	
	902	?A=C	P	❻
	C2	GOYES	%6	
	5ED	GONC	%2	
#37Ah	CA	A=A+C	A	❼
	302	C=(1)	#2	
	902	?A=C	P	
	21	GOYES	%5	
	304	C=(1)	#4	
	902	?A=C	P	
	A0	GOYES	%5	
	30D	C=(1)	#D	
	906	?A≠C	P	
	4C	GOYES	%2	❽
#394h	05	SETDEC		
	CF	D=D-1	A	❾
	04	SETHEX		

❶ S'agit-il d'un carré ? Non ? Alors le coup est interdit.
❷ Point de branchement %2. S'il s'agit d'un carré, on se déplace.
❸ Point de branchement %3 (réaction). Départ vers C.
❹ On ne tient pas compte de la couche.
❺ Test de l'identité départ/arrivée ? Déplacement.
❻ Y a-t-il deux carrés ? Si oui, assemblage. Si non, déplacement.
❼ Point de branchement %4 (assemblage). Y a-t-il deux figures complémentaires ?
❽ Point de branchement %5.
❾ Un carré apparaît si les deux figures sont complémentaires.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	560	GONC	%5	
	6AC3	GOTO	$GAGNE	
#3A1h	132	AD0EX		❶
	1B83100	D0=(5)	#00138	
	14E	C=DAT0	B	❷
	130	D0=A		
	816	CSRC		❸
	80882	CBIT=0	2	
	80883	CBIT=0	3	
	14A	A=DAT0	B	
	80873	?ABIT=1	3	❹
	E0	GOYES	%7	
	304	C=(1)	#4	❺
	A46	C=C+C	S	
	450	GOC	%7	
	30C	C=(1)	#C	
#3D4h	15C0	DAT0=C	1	❻
	305	C=(1)	#5	
	864	?ST=0	4	
	C0	GOYES	%8	
	E6	C=C+1	A	
	A46	C=C+C	S	❼
	440	GOC	%8	
	E6	C=C+1	A	
#3EAh	14B	A=DAT1	B	❽
	80863	?ABIT=0	3	
	70	GOYES	%9	❾
	80893	CBIT=1	3	
#3F9h	15D0	DAT1=C	1	❿
	860	?ST=0	0	

❶ Point de branchement %6.
❷ Lecture de l'horloge.
❸ Sauvegarde d'un flag aléatoire. On veut un triangle de la 2e couche.
❹ Etait-on déjà sur la 2e couche ?
❺ Si oui, on mémorise un blanc ou un noir selon l'horloge.
❻ Point de branchement %7. Modification de la case de départ.
❼ Dans ce cas, deux figures (nœud papillon/sablier) prises aléatoirement.
❽ Point de branchement %8. Lecture de la case de destination.
❾ Conservation de la couche.
❿ Point de branchement %9. Modification de la case de destination.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	33	GOYES	%13	❶
	307	C=(1)	#7	
#405h	06	RSTK=C		❷
	D0	A=0	A	❸
	D2	C=0	A	
	28	P=	8	
#40Dh	80F2	CPex	2	❹
	801	OUT=C		
	AE6	C=A	B	
#417h	81862B	C=C+#C	B	❺
	59F	GONC	%12	
	818601	A=A+#2	B	
	56E	GONC	%11	
	07	C=RSTK		
	20	P=	0	
	A0E	C=C-1	P	❻
	54D	GONC	%10	
#433h	4C3	GOC	%15	❼
#436h	14A	A=DAT0	B	❽
	14F	C=DAT1	B	❾
	C4	A=A+A	A	
	C6	C=C+C	A	
	A8E	ACex	P	
	81C	ASRB		
	81E	CSRB		
	1580	DAT0=A	1	
	15D0	DAT1=C	1	
	874	?ST=1	4	❿
	C1	GOYES	%15	

❶ Le son est-il autorisé ? 7 « tics »...
❷ Point de branchement %10
❸ Copie de la durée dans A.
❹ Point de branchement %11. Bip !
❺ Point de branchement %12. Temporisation de la fréquence.
❻ « Tic » suivant.
❼ Point de branchement %13. Modification pointeur case centrale
❽ Point de branchement %14.
❾ Inversion case de départ/d'arrivée avec conservation de la couche.
❿ Est-ce un déplacement ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	80843	ABIT=0	3	
	305	C=(1)	#5	
	902	?A=C	P	❶
	F3	GOYES	%18	
	14B	A=DAT1	B	
	80843	ABIT=0	3	
	902	?A=C	P	
	23	GOYES	%18	
#470h	137	CD1ex		❷
	135	D1=C		
	D5	B=C	A	
	860	?ST=0	0	❸
	B4	GOYES	%21	
	D0	A=0	A	
	22	P=	2	
#481h	308	C=(1)	#8	❹
	801	OUT=C		
	D2	C=0	A	
	801	OUT=C		
#48Ch	B66	C=C+1	B	❺
	5CF	GONC	%17	
	818607	A=A+#8	B	❻
	58E	GONC	%16	
	20	P=	0	
	482	GOC	%21	❼
#4A0h	860	?ST=0	0	❽
	32	GOYES	%21	❾
	D0	A=0	A	❿
	D2	C=0	A	

❶ S'il s'agit d'un carré, inversion uniquement.
❷ Point de branchement %15. B désigne la nouvelle case centrale.
❸ Test : le son est-il autorisé ?
❹ Point de branchement %16. Bip avec volume faible.
❺ Point de branchement %17. Temporisation fréquence.
❻ Boucle durée.
❼ Affichage du zoom.
❽ Point de branchement %18.
❾ Test : le son est-il autorisé ?
❿ A contient la durée.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	28	P=	8	
#4ABh	80F2	CPex	2	❶
	801	OUT=C		
	AE6	C=A	B	
#4B5h	B66	C=C+1	B	❷
	5CF	GONC	%20	
	818602	A=A+#3	B	❸
	59E	GONC	%19	
	20	P=	0	
#4C6h	6CCB	GOTO	$0	❹
#4CAh				❺
#000h	1BB3100	D0=(5)	#0013B	
	1E0030	D1=(4)	#0300	
	14A	A=DAT0	B	❻
	80860	?ABIT=0	0	
	80	GOYES	%1	❼
	841	ST=0	1	
	5D3	GONC	%4	
#01Dh	871	?ST=1	1	❽
	83	GOYES	%4	
	851	ST=1	1	
#025h	14B	A=DAT1	B	❾
	05	SETDEC		
	A0C	A=A-1	P	❿
	04	SETHEX		
	1590	DAT1=A	1	
	542	GONC	%4	
	133	AD1ex		

❶ Point de branchement %19. Bip.
❷ Point de branchement %20. Temporisation fréquence.
❸ Boucle durée.
❹ Point de branchement %21. Affichage du zoom.
❺ Début de la procédure $CHRONO en #4CAh.
❻ Lecture de l'horloge (1/2 seconde).
❼ Test de la parité (1/2 seconde). Attente si imparité.
❽ Point de branchement %1. S'agit-il d'un changement.
❾ Point de branchement %2. Modification du chronomètre.
❿ Chiffre par chiffre. Fin si absence de retenue.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	131	D1=A		
	80860	?ABIT=0	0	❶
	90	GOYES	%3	
	305	C=(1)	#5	❷
	15D0	DAT1=C	1	
#04Ah	170	D1=D1+	#1	❸
	80862	?ABIT=0	2	
	3D	GOYES	%2	
	6F42	GOTO	$PERDU	
#058h	110	A=R0		❹
	346D100	C=(5)	#001D6	
	CA	A=A+C	A	❺
	130	D0=A		
	1D00	D1=(2)	#00	❻
	14B	A=DAT1	B	
	301	C=(1)	#1	
	7120	GOSUB	$AFF	
#075h	180	D0=D0-	#1	
	1E2030	D1=(4)	#0302	❼
	14B	A=DAT1	B	
	301	C=(1)	#1	
	7E00	GOSUB	$AFF	❽
#088h	180	D0=D0-	#1	
	1E4030	D1=(4)	#0304	❾
	14B	A=DAT1	B	
	D2	C=0	A	
#096h	102	R2=A		❿
	06	RSTK=C		
	D2	C=0	A	

❶ Est-ce une dizaine ?
❷ On fixe le maximum à 59 (et non à 99).
❸ Point de branchement %3. Passage au suivant.
❹ Point de branchement %4.
❺ D0 pointe sur l'écran.
❻ On pointe sur les secondes. Affichage du nombre de secondes.
❼ On pointe sur les minutes.
❽ On affiche le nombre de minutes (sur deux chiffres).
❾ On pointe sur les heures (affichage sur un chiffre)
❿ Début de la procédure $AFF. Point de branchement %1 et sauvegarde.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	31D4	C=(2)	#4D	
	81B4	A=PC		
#0A5h	CA	A=A+C	A	❶
	D2	C=0	A	
	81A01A	C=R2	P	
	CA	A=A+C	A	
	CA	A=A+C	A	❷
	F2	CSL	A	
	CA	A=A+C	A	
	131	D1=A		
	308	C=(1)	#8	❸
#0BDh	14B	A=DAT1	B	❹
	148	DAT0=A	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❺
	161	D0=D0+	#2	
	171	D1=D1+	#2	❻
	A0E	C=C-1	P	
	5AE	GONC	%2	❼
	13A	AD0xs		
	334310	C=(4)	#0134	❽
	EA	A=A-C	A	
	138	D0=AS		
	112	A=R2		
	F4	ASR	A	❾
	07	C=RSTK		
	A0E	C=C-1	P	❿
	58A	GONC	%1	
	01	RTN		

❶ On pointe sur les données relatives aux chiffres.
❷ Calcul de A = A + (C×#18h).
❸ Compteur.
❹ Point de branchement %2. Transfert de données vers l'écran.
❺ Passage à la ligne du dessous.
❻ Passage à la donnée suivante.
❼ Boucle sur 9 lignes.
❽ On passe au chiffre suivant sur l'écran.
❾ Chiffre suivant.
❿ Boucle sur les chiffres.

Adresses	Codes hexadécimaux	Mnémoniques	Renvois
#0F2h	E1121	*Données graphiques*	❶
	21212	*Données graphiques*	
	12121	*Données graphiques*	
	2E180	*Données graphiques*	
	C0808	*Données graphiques*	
	08080	*Données graphiques*	
	8080C	*Données graphiques*	
	1E112	*Données graphiques*	
	12018	*Données graphiques*	
	04020	*Données graphiques*	
	10F3E	*Données graphiques*	
	11202	*Données graphiques*	
	02C10	*Données graphiques*	
	20212	*Données graphiques*	
	E1018	*Données graphiques*	
	14121	*Données graphiques*	
	11F30	*Données graphiques*	
	10101	*Données graphiques*	
	F3101	*Données graphiques*	
	0F102	*Données graphiques*	
	02021	*Données graphiques*	
	2E1E1	*Données graphiques*	
	12101	*Données graphiques*	
	0F112	*Données graphiques*	
	1212E	*Données graphiques*	
	1F302	*Données graphiques*	
	02018	*Données graphiques*	
	04040	*Données graphiques*	
	4040E	*Données graphiques*	
	11212	*Données graphiques*	
	12E11	*Données graphiques*	
	21212	*Données graphiques*	
	E1E11	*Données graphiques*	
	21212	*Données graphiques*	
	E3020	*Données graphiques*	
	212E1	*Données graphiques*	
#1A6h	10003	*Données graphiques*	❷

❶ Données graphiques relatives aux chiffres (de #0F2h à #1A5h).
❷ Données graphiques relatives aux sprites (de #1A6h à #29Dh).

Adresses	Codes hexadécimaux	Mnémoniques	Renvois
	00070	Données graphiques	
	00F00	Données graphiques	
	07100	Données graphiques	
	D200D	Données graphiques	
	7005B	Données graphiques	
	00D61	Données graphiques	
	0DF30	Données graphiques	
	5B50D	Données graphiques	
	6B0DF	Données graphiques	
	F130C	Données graphiques	
	3FFF7	Données graphiques	
	FFF3E	Données graphiques	
	106CF	Données graphiques	
	F586B	Données graphiques	
	50D65	Données graphiques	
	0EF50	Données graphiques	
	4B508	Données graphiques	
	6500F	Données graphiques	
	500A5	Données graphiques	
	00470	Données graphiques	
	08700	Données graphiques	
	07000	Données graphiques	
	60004	Données graphiques	
#21E	00000	Données graphiques	❶
	00000	Données graphiques	
	00000	Données graphiques	
	00000	Données graphiques	
	00000	Données graphiques	
	00000	Données graphiques	
	00FFF	Données graphiques	
	75145	Données graphiques	
	BFF65	Données graphiques	
	00570	Données graphiques	
	07500	Données graphiques	
	55005	Données graphiques	
	50053	Données graphiques	
	00670	Données graphiques	
	07D08	Données graphiques	

❶ Données graphiques pour blanc et carrés.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
CCD 20	5B1C6	***Données graphiques***		
	73675	***Données graphiques***		
	6353D	***Données graphiques***		
	567B6	***Données graphiques***		
	7FFF7	***Données graphiques***		
	BDD66	***Données graphiques***		
	B63C0	***Données graphiques***		
	818DD	***Données graphiques***		
	00B60	***Données graphiques***		
	06300	***Données graphiques***		
	C10	***Données graphiques***		
#29Eh	850	ST=1	0	❶
	450	GOC	$FIN	
#294h	840	ST=0	0	❷
#2A7h	32FF7	C=(3)	#7FF	❸
	8FCEE10	GOSBVL	#01EEC	❹
	93E	?C≠0	X	
	1F	GOYES	%1	
#2B8h	32FF7	C=(3)	#7FF	❺
	8FCEE10	GOSBVL	#01EEC	❻
	93A	?C=0	X	
	80	GOYES	%3	
	D2	C=0	A	
	81B3	PC=C		❼
#2CFh	111	A=R1		❽
	3178	C=(2)	#87	❾
	130	D0=A		❿

❶ Procédure $GAGNE. L'écran est effacé.
❷ Prccédure $PERDU. Inversion vidéo.
❸ Procédure $Fin. Point de branchement %1.
❹ Attente du relâchement des touches.
❺ Point de branchement %2.
❻ Attente d'une pression sur une touche.
❼ Reset dès qu'un appui est détecté.
❽ Point de branchement %3.
❾ #880h quartets.
❿ D0 désigne le début de l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#2D9h	1527	A=DAT0	W	❶
	860	?ST=0	0	
	80	GOYES	%5	❷
	A74	A=A+A	W	❸
	B24	A=A+1	XS	
#2E8h	BFC	A=-A-1	W	❹
#2EBh	1507	DAT0=A	W	❺
	16F	D0=D0+	#10	❻
	A6E	C=C-1	B	
	53E	GONC	%4	❼
	3400030	C=(5)	#03000	
	860	?ST=0	0	
	80	GOYES	%7	❽
	330001	C=(4)	#1000	
#30Ah	CE	C=C-1	A	❾
	5DF	GONC	%7	
	48A	GOC	%2	❿
#312h				

❶ Point de branchement %4. Lecture écran.
❷ Que fait-on ?
❸ Si le joueur a gagné, on efface progressivement.
❹ Point de branchement %5. Si le joueur a perdu, inversion vidéo.
❺ Point de branchement %6.
❻ Passage aux quartets suivants.
❼ Boucle sur #880h quartets.
❽ Si le joueur a gagné, la temporisation est plus courte.
❾ Point de branchement %7. Temporisation.
❿ Retour vers le test du clavier.

3

Exécuter le programme

La méthode la plus académique pour exécuter le programme consiste à sauvegarder l'objet code exécutable dans une variable 'CODE'. Cela dit,

dans le cas du jeu ALPHA, il nous semble intéressant de ne mémoriser que l'objet chaîne de caractères ('ALPHA48S'). On évite ainsi la multiplication des variables et la mobilisation en permanence de la mémoire affectée à la sauvegarde de l'objet code exécutable.

Pour exécuter le programme, on pourra donc procéder de la façon suivante :

- la chaîne de codes hexadécimaux étant enregistrée dans la variable 'ALPHA48S', on place ladite chaîne au niveau 1 de la pile,
- au niveau 1 de la pile apparaît **"CCD200E70084F808F...**
- lancez le programme 'ASS' qui va prendre la chaîne hexadécimale comme argument,
- au niveau 1 de la pile **"CCD200E70084F808F...** est remplacé par **Code** *,
- Il suffit ensuite d'appuyer sur la touche [EVAL] pour lancer le jeu.

* A ce stade, il est aussi possible de saisir 'CODE' et d'appuyer sur [STO] pour mémoriser le code exécutable dans la variable 'CODE' qu'il vous suffira ensuite d'évaluer pour exécuter le jeu.

4

Comment jouer ?

ALPHA est un jeu d'association de pièces. Il faut assembler deux pièces complémentaires pour créer un carré. Créez 400 carrés en moins de deux heures et vous aurez gagné ! Vous jouez sur une sphère couverte de 256 cases. Seules neuf cases sont visibles à un instant donné. La case centrale est la case active et pourra interagir avec les huit cases qui l'entourent. Pour faire réagir le contenu de la case centrale avec le contenu de l'une des huit cases qui l'entourent, il suffit d'appuyer sur les touches [1], [2], [3], [4], [6], [7], [8] et [9] du pavé numérique. Ainsi, pour faire réagir le contenu de la case centrale avec celui de la case se trouvant en bas à droite, il suffira d'appuyer sur la touche [3].

Retenez que c'est toujours la case centrale qui réagit avec celles qui l'entourent. Les figures utilisées sont les suivantes : triangles (il en existe

quatre différents), le nœud papillon, le sablier, le carré, la pièce blanche, et la pièce noire. Les triangles, le nœud papillon et le sablier sont les pièces dites « mineures ». Les différentes réactions possibles entre les figures sont les suivantes :

- l'assemblage de deux triangles complémentaires, dans ce cas, un carré se forme,
- l'assemblage d'un nœud papillon avec un sablier, dans ce cas, un carré se forme,
- la réaction entre deux carrés provoque la création de pièces mineures,
- la réaction entre une pièce mineure et un carré provoque un échange des positions,
- dans tous les autres cas, la case de destination choisie devient la case centrale (on se déplace sur la sphère).

A chaque fois qu'une case vide apparaît (à la suite, par exemple, de la création d'un carré), elle rend visible la seconde couche de figures (en inversion vidéo). Les cases de la première couche ont un fond blanc alors que celles de la seconde couche ont un fond noir.

Seul le carré peut passer de la première couche (fond blanc) à la seconde couche (fond noir) par simple inversion des positions. Ce sont donc vos carrés qu'il faudra exploiter pour faire « monter » sur la première couche ou « descendre » vers la seconde couche les pièces de votre choix (en échangeant la position d'un carré de l'une des couches avec la position d'une pièce mineure de l'autre couche).

En résumé, voici les fonctions de chacune des touches utilisées pendant une partie :

- [SIN] provoque une pause,
- [ON] sert à reprendre une partie interrompue en appuyant sur [SIN],
- [MTH] [MTH] (deux pressions) met fin au jeu. En fait, vous pouvez aussi appuyer sur [MTH] puis sur une touche quelconque,
- [1] fait réagir la figure au centre de l'écran avec celle qui se trouve en bas à gauche,
- [2] fait réagir la figure au centre de l'écran avec celle qui se trouve en bas au centre,
- [3] fait réagir la figure au centre de l'écran avec celle qui se trouve en bas à droite,
- [4] fait réagir la figure au centre de l'écran avec celle qui se trouve à gauche,
- [6] fait réagir la figure au centre de l'écran avec celle qui se trouve à droite,

• [7] fait réagir la figure au centre de l'écran avec celle qui se trouve en haut à gauche,
• [8] fait réagir la figure au centre de l'écran avec celle qui se trouve en haut au centre,
• [9] fait réagir la figure au centre de l'écran avec celle qui se trouve en haut à droite.

Plus de 100 carrés en moins de 20 minutes... C'est possible

5

Retrouver les fichiers sur la disquette

Les chemins d'accès aux différents fichiers sont :
• pour 'ASS', DUNOD_1\S_SX\ASS,
• pour 'ALPHA48S', DUNOD_1\S_SX\OB_P_OBS\ALPHA_S\ALPHA48S,
• pour 'CODE', DUNOD_1\S_SX\OB_P_OBS\ALPHA_S\CODE.

29

Star pour HP-48 S et SX

1

Le jeu

1.1

Présentation

Les adeptes du jeu sur Macintosh retrouveront avec STAR les joies des Tinies™, célèbre jeu de l'ordinateur à la pomme. En effet, on retrouve avec STAR des principes de jeu assez proches de ceux qui ont fait le succès des Tinies™ : vous déplacez un sprite qui ne peut s'arrêter qu'en rencontrant une paroi. Pour passer au tableau suivant, vous devrez « gober » toutes les pastilles rencontrées... Comme nous l'avons dit, vous êtes obligé de vous déplacer en vous heurtant aux murs, heureusement, vous pouvez déplacer un carré, sorte de « petit mur mobile », pour mieux contrôler vos déplacements. Bien entendu, ce dernier est soumis aux mêmes règles de déplacements que votre sprite gobeur !

Bref, un cocktail Pac-Man™/Tinies™ boulimique et rebondissant !

Contrairement à ALPHA qui brille par la simplicité de sa mise en œuvre, STAR nécessite un programme lanceur écrit en RPL. Ce programme, au demeurant très simple, est chargé de la mise en place d'un objet graphique dans la pile.

1.2
Liste des objets

Neuf objets sont associés à la mise en œuvre de ce jeu :

- 'STAR48S' est la chaîne de codes hexadécimaux qui devra être traitée avec le programme ASS pour générer le code exécutable 'CODE',
- 'CODE' est le code exécutable généré à partir de 'STAR48S',
- 'ASS' est le programme permettant de générer 'CODE' à partir de 'STAR48S',
- 'STAR' est le programme lanceur (écrit en RPL),
- 'T1', 'T2', 'T3' et 'T4' sont des variables intermédiaires qui facilitent la saisie de 'TABL',
- 'TABL' est un objet graphique (GROB).

'STAR48S', 'ASS', 'T1', 'T2', 'T3' et 'T4' sont des variables utilisées pour le développement, l'édition et l'introduction de STAR mais ne sont absolument pas indispensables à l'exécution du programme.

Seules les variables 'STAR' (lanceur RPL), 'CODE' (code exécutable) et 'TABL' (objet graphique) sont indispensables à l'exécution du programme.

2
Les objets

2.1
Le lanceur RPL

Le listing est un modèle de simplicité :

```
« TABL -6 CODE »
```

Cet objet est à enregistrer dans une variable nommée 'STAR'.

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #D011h,
- au niveau 1, une longueur égale à 27,5 octets.

2.2
Les objets graphiques

L'objet graphique 'TABL' est assez long. Nous avons donc préféré le segmenter en quatre chaînes de caractères 'T1', 'T2', 'T3' et 'T4' qu'il suffira de concaténer (une concaténation est une juxtaposition de chaînes de caractères).

■ La chaîne de caractères 'T1'

Cet objet est à saisir sans espace ni saut de ligne et à enregistrer dans une variable nommée 'T1'.

```
"B 0F0 A01 FFFF 500B 7008 1C08 30C8 100C 1308
           1038 FFFF 0000 8000 0100 0200 0010
           8021 4402 2800 0000
 9 A01 2B4 B8A4 1008 0011 500A 2020 1058 9821
           2008 4084 0000 2080 8880 0280 0200
           0200 6000 0000 0000
 F A01 8F1 FFFF 5008 D0C8 1088 1608 320A 702B
           1038 FFFF 0000 8000 0000 0140 2802
           4421 8290 0804 0000
 F C96 A01 FFFF 5008 1028 551D BA8A 554D 1028
           9088 FFFF 0000 0000 0000 AAA2 0050
           AAA2 0000 0000 0000
 8 0F0 0B5 EFF7 1988 5029 190C 3049 111C 545A
           5288 EFF7 0000 0000 0980 0000 0084
           0400 0004 2400 0000"
```

Note : chaque groupe de trois lignes correspond à un tableau de jeu.

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #0930h,
- au niveau 1, une longueur égale à 400 octets.

■ La chaîne de caractères 'T2'

Cet objet est à saisir sans espace ni saut de ligne et à enregistrer dans une variable nommée 'T2'.

```
"9 AB3 486 FFF0 1101 B002 1004 1008 320C 9008
           1309 FFFF 0000 A010 0010 0010 0010
           0010 0010 8010 0000
 7 886 696 FFFF 1008 D4E6 AA44 CE42 AA46 1008
           5088 FFFF 0000 0000 2200 0022 0000
           4400 0000 2040 0000
 A CA5 895 EFFF 30C8 1819 14D8 591C 900D 5188
           100C FFF7 0000 0001 0280 0201 8000
           4100 0805 0000 0000
 D A01 8C3 EFF2 1105 2088 9815 1029 2818 400A
           8008 0FF7 0000 0082 0405 0120 2490
           0020 0005 0000 0000
 C 0C4 4E1 EFF0 100F 9838 2428 9028 A42A C934
           A208 9CF7 0000 0040 2500 0212 2400
           0805 0002 0000 0000"
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #B068h,
- au niveau 1, une longueur égale à 400 octets.

■ La chaîne de caractères 'T3'

Cet objet est à saisir sans espace ni saut de ligne et à enregistrer dans une variable nommée 'T3'.

```
"E A01 CA5 FFFF 1018 3009 160C 7828 168D 1048
           32CD EFF6 0000 0000 81C0 2040 0614
           0000 2094 0002 0000
 A 0F0 EB4 E91C 37EB 1008 513B 992A 513B 1008
           1008 FFFF 0000 4010 8060 8800 0480
           0000 0020 0400 0000
 A 2F0 486 EFF7 3009 D02A 5008 5008 100A 500C
           5188 EFF7 0000 8000 0110 0800 2002
           0400 2A00 0400 0000
 C 4E1 EC2 FFFF 500A 9109 1008 1208 1008 9009
           500A FFFF 0000 A005 0002 0100 0400
           0010 0002 A005 0000
```

```
C 0D2 0D3 EFE7 1018 142A 9888 5015 9408 2004
          1008 EFF7 0000 4020 4080 4600 0400
          4004 0800 2004 0000"
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #F491h,
- au niveau 1, une longueur égale à 400 octets.

■ La chaîne de caractères 'T4'

Cet objet est à saisir sans espace ni saut de ligne et à enregistrer dans une variable nommée 'T4'.

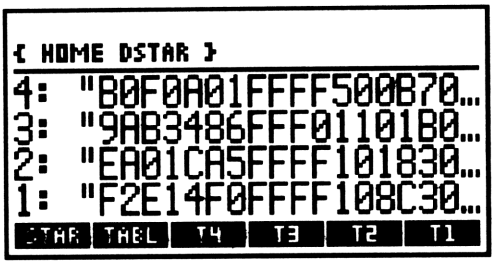
```
"F 2E1 4F0 FFFF 108C 3048 30CD 1A0C 1108 1F0C
          102D FFFF 0000 2050 00A4 0200 0410
          8206 8001 0002 0000
 F 801 A96 FFFF 9009 542A 1A58 9009 1188 5A5A
          1018 FFFF 0000 0002 0A50 8420 0102
          0241 0400 0840 0000
 F 886 201 FFFF 1108 1249 11A8 9018 198A 1229
          902A FFFF 0000 0012 01A0 8040 4180
          0201 0882 0004 0000
 B E96 0B5 FFFF D068 904A 1C09 101A 1248 5C2B
          D049 FFFF 0000 2004 4000 8012 0800
          0400 0240 0022 0000
 C 2D3 CB3 FFFF 1048 3C2C 528A 1998 514A 343C
          1208 FFFF 0000 00A0 0242 0020 2000
          0400 4240 0500 0000 0"
```

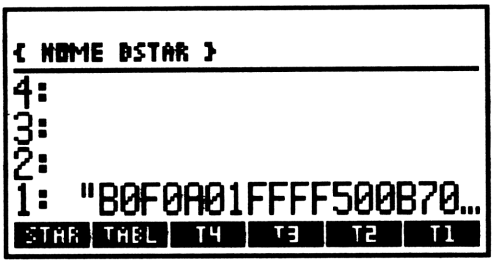
Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #7E85h,
- au niveau 1, une longueur égale à 401 octets.

■ L'objet graphique 'TABL'

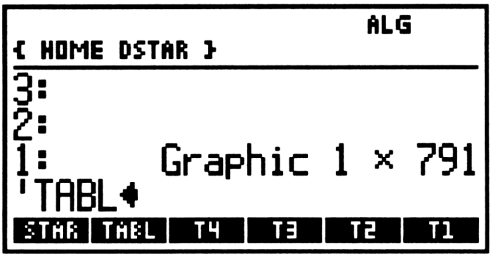
L'objet graphique que nous allons maintenant élaborer est à enregistrer dans une variable nommée 'TABL'. Placez-vous dans le répertoire DSTAR où vous devez avoir placé les variables 'T1', 'T2', 'T3' et 'T4'. Placez dans la pile les variables T1, T2, T3 et T4 en respectant cet ordre ! Appuyez ensuite trois fois sur la touche [+], vous concaténerez ainsi ces chaînes.





Pour créer la variable TABL :

- passez en environnement d'édition en appuyant sur [↓],
- supprimez les guillemets aux deux extrémités de la chaîne,
- insérez **GROB 1 791** au début de la chaîne,
- appuyez sur [ENTER],
- saisissez 'TABL',
- appuyez sur [STO].



Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez, au niveau 2, une somme de contrôle égale à #A263h, et au niveau 1, une longueur égale à 801 octets.

2.3
Le code hexadécimal

La chaîne à saisir

Cet objet est à enregistrer dans une variable nommée 'STAR48S'. La chaîne de codes hexadécimaux est à saisir entre les guillemets " et ", sans espace ni saut de ligne.

```
CCD20 52400 84F80 8F082 01433 4FE3A
28A2C 01421 64808 C1741 43104 1F2D6
0714F 808B3 50851 1BB01 00314 814CD
219E2 15C01 EE050 14710 81853 0F15C
0114D 8D231 41C1D 910B1 35118 134AF
03178 15071 6FA6E 55F14 FD717 0118D
01533 CA101 172D0 1533C A1021 72134
30806 13713 5D5AC 2147F 2C65A 40634
BE200 81B4C A1313 07061 43146 0E3E1
50316 F16F1 61172 07A0E 5ED13 23201
1EA13 00716 1B465 CA132 D231E CCA13
0D913 51730 7A0E4 60697 F1181 34308
06AC2 147F2 C65B1 06302 79121 32D23
1CCEA 13007 161B4 65BD1 32D23 1ECCA
13017 307A0 E58B1 37D51 11130 30175
D1112 130D2 79C18 401BB 0100D 215C0
87090 30415 C0320 048FC EE108 08AFB
011BD 5618E D0320 408FC EE108 08A46
081B2 808A0 60E4E 4808A 260CC CC8AC
06808 A1A0D 231EE CA320 808FC EE108
08A1A 0D231 EEEA8 AC138 08A06 28608
08405 50850 DA330 006CE 5DF31 51962
476C3 F1198 60501 1ADE0 6CAD2 3198C
A1301 4A30A 90660 611F8 08707 F8086
1C487 0BE86 17231 04DA3 20088 01630
0D280 1D6A6 E5CFA 6C53E A0F57 1D913
514B9 0C606 36D6C 6D111 86050 112D2
3144C A1303 03061 4A311 80E66 14816
F16F1 6107A 0E51E 07061 32CAD 231CC
EA130 86080 10255 0101D 28704 0E673
```

```
10320 09A3E 5CF07 DA640 FD0A8 A1370
634A4 000F0 C281B 4CA13 13050 614B1
4E0E6 E1481 6F16F 16117 107A0 E5FD0
71350 1EF09 213C1 D013A 19415 21EF0
0000C 3C3C3 C3000 00000 81C2C 38100
00000 08142 4281
```

Placez dans la pile une copie de l'objet 'STAR48S', activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #C3Dh,
- au niveau 1, une longueur égale à 1069 octets.

Pour générer l'objet 'CODE' à partir de 'STAR48S', procédez de la façon suivante :

- placez la chaîne de codes hexadécimaux (variable 'STAR48S') au niveau 1 de la pile, **"CCD205240084F808F…** s'affiche,
- exécuter le programme ASS,
- au niveau 1 de la pile doit apparaître **Code**,
- si ce n'est pas encore fait, placez-vous dans le répertoire DSTAR,
- saisissez le nom **'CODE'**,
- appuyez sur [STO].

Placez dans la pile une copie de l'objet 'CODE', activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #C39Fh,
- au niveau 1, une longueur égale à 533 octets.

Codes commentés

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	CCD20			❶
#000h	52400			
	84F	ST=0	15	
	808F	INTOFF		❷
	08	CLRST		
	20	P=	0	

❶ Prologue puis, en #000h, longueur totale hors prologue.
❷ Inhibition des interruptions.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	143	A=DAT1	A	❶
	34FE3A2	C=(5)	#2A3EF	
	8A2	?A=C	A	❷
	C0	GOYES	%1	
	142	A=DAT0	A	❸
	164	DO=DO+	#5	
	808C	PC=(A)		
%1				
#029h	174	D1=D1+	#5	❹
	143	A=DAT1	A	
	104	R4=A		
	1F2D607	D1=(5)	#706D2	
	14F	C=DAT1	B	❺
	808B3	?CBIT=1	3	
	50	GOYES	%2	
	851	ST=1	1	❻
%2				
#046h	1BB0100	D0=(5)	#0010B	
	3148	C=(2)	#84	❼
	14C	DAT0=C	B	
	D2	C=0	A	
	19E2	D0=(2)	#2E	❽
	15C0	DAT0=C	1	
	1EE050	D1=(4)	#050E	
	147	C=DAT1	A	❾
	108	R0=C		
	185	D0=D0-	#6	
	30F	C=(1)	#F	❿
	15C0	DAT0=C	1	

❶ Lecture du niveau 1 de la pile RPL.
❷ Le niveau 1 de la pile contient-il -6 ?
❸ Si ce n'est pas le cas, on rend la main au RPL.
❹ Point de branchement %1. R4 désigne le début du GROB tableau.
❺ On teste le flag beep RPL.
❻ Si flag beep actif on a ST1=1.
❼ Point de branchement %2 en #046h. Supprime le sablier, allume α.
❽ Moins d'interruption.
❾ Lecture de la bitmap écran. R0 désigne le début de l'écran.
❿ Suppression des menus.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$0				❶
#074h	114	A=R4		
	D8	B=A	A	
	D2	C=0	A	
	3141	C=(2)	#14	❷
	C1	B=B+C	A	
$1				
#081h	D9	C=B	A	❸
	10B	R3=C		
	135	D1=C		❹
	118	C=R0		
	134	D0=C		❺
	AF0	A=0	W	
	3178	C=(2)	#87	❻
%1				
#096h	1507	DAT0=A	W	❼
	16F	D0=D0+	#10	❽
	A6E	C=C-1	B	
	55F	GONC	%1	
	14F	C=DAT1	B	
	D7	D=C	A	❾
	170	D1=D1+	#1	
	118	C=R0		
	D0	A=0	A	
	1533	A=DAT1	X	
	CA	A=A+C	A	
	101	R1=A		❿
	172	D1=D1+	#3	
	D0	A=0	A	

❶ Début de la procédure $0 (tableau n°1).
❷ On pointe sur le contenu du GROB TABL.
❸ Procédure $1 (affichage tableau). Sauvegarde adresse début tableau.
❹ D1 désigne le début du tableau.
❺ D0 désigne le début de l'écran.
❻ #880h quartets.
❼ Point de branchement %1.
❽ Efface tout l'écran.
❾ Lecture du nombre de pastilles à gober.
❿ R1 vaut position du gobeur + R0.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1533	A=DAT1	X	
	CA	A=A+C	A	❶
	102	R2=A		
	172	D1=D1+	#3	
#0CAh	134	D0=C		❷
	308	C=(1)	#8	
%2				
#0D0h	06	RSTK=C		❸
	137	CD1ex		
	135	D1=C		
	D5	B=C	A	❹
	AC2	C=0	S	
	147	C=DAT1	A	
	F2	CSL	A	❺
%3				❻
#0E2h	C6	C=C+C	A	
	5A4	GONC	%5	❼
	06	RSTK=C		
	34BE200	C=(5)	#002EB	
	81B4	A=PC		
#0F4h	CA	A=A+C	A	
	131	D1=A		❽
	307	C=(1)	#7	
%4				
#0FCh	06	RSTK=C		❾
	143	A=DAT1	A	
	146	C=DAT0	A	
	0E3E	A=A!C	X	❿
	1503	DAT0=A	X	

❶ R2 = position du carré + R0
❷ Affichage des bords. On pointe sur le début de l'écran.
❸ Point de branchement %2. Sauvegarde du compteur.
❹ Sauvegarde de D1 dans B.
❺ On pointe sur la première colonne et la première ligne.
❻ Point de branchement %3.
❼ Test : est-ce un bord ? Sauvegarde de la ligne.
❽ On pointe sur les données graphiques relatives à un obstacle.
❾ On affiche l'obstacle en D0.
❿ On procède par « ou » logique avec l'affichage.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	16F	D0=D0+	#10	❶
	16F	D0=D0+	#10	
	161	D0=D0+	#2	
	172	D1=D1+	#3	
	07	C=RSTK		
	A0E	C=C-1	P	❷
	5ED	GONC	%4	
	132	AD0ex		
	32011	C=(3)	#110	❸
	EA	A=A-C	A	
	130	D0=A		
	07	C=RSTK		❹
%5				❺
#12Fh	161	D0=D0+	#2	
	B46	C=C+1	S	❻
	5CA	GONC	%3	
	132	AD0ex		
	D2	C=0	A	
	31EC	C=(2)	#CE	
	CA	A=A+C	A	❼
	130	D0=A		
	D9	C=B	A	
	135	D1=C		❽
	173	D1=D1+	#4	
	07	C=RSTK		
	A0E	C=C-1	P	❾
	460	GOC	%6	
	697F	GOTO	%2	
%6				❿
#15Ah	118	C=R0		

❶ Passage à la ligne suivante.
❷ Boucle sur 8 lignes.
❸ Retour au début de la colonne.
❹ Restauration de la ligne.
❺ Point de branchement %5.
❻ Passage à la colonne suivante sur la même ligne.
❼ Passage à la ligne suivante.
❽ Passage à la ligne suivante dans le tableau.
❾ Boucle sur 9 lignes.
❿ Point de branchement %6.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	134	D0=C		❶
	308	C=(1)	#8	
%7				
#163h	06	RSTK=C		❷
	AC2	C=0	S	
	147	C=DAT1	A	
	F2	CSL	A	❸
%8				❹
#16Dh	C6	C=C+C	A	
	5B1	GONC	%9	
	06	RSTK=C		
	302	C=(1)	#2	
	7912	GOSUB	$6	❺
#17Bh	132	ADOex		
	D2	C=0	A	
	31CC	C=(2)	#CC	
	EA	A=A-C	A	❻
	130	D0=A		
	07	C=RSTK		❼
%9				❽
#18Bh	161	D0=D0+	#2	
	B46	C=C+1	S	
	5BD	GONC	%8	
	132	AD0ex		
	D2	C=0	A	
	31EC	C=(2)	#CE	❾
	CA	A=A+C	A	
	130	D0=A		
	173	D1=D1+	#4	❿
	07	C=RSTK		

❶ On pointe sur le début de l'écran.
❷ Point de branchement %7. Sauvegarde du compteur.
❸ On pointe sur le première colonne.
❹ Point de branchement %8. Test : est-ce un bord ? Sauve la ligne.
❺ Affichage d'une pastille à gober en D0.
❻ Retour au début de la colonne.
❼ Restauration de la ligne.
❽ Point de branchement %9. Passage à la colonne suivante.
❾ Passage à la ligne suivante (1 ligne jeu = 7 lignes écran).
❿ Passage à la ligne suivante dans le tableau.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	A0E	C=C-1	P	❶
	58B	GONC	%7	
	137	CD1ex		❷
	D5	B=C	A	
#1B2h	111	A=R1		❸
	130	D0=A		
	301	C=(1)	#1	❹
	75D1	GOSUB	$6	
#1BFh	112	A=R2		❺
	130	D0=A		
	D2	C=0	A	❻
	79C1	GOSUB	$6	
#1CBh	840	ST=0	0	❼
$2				❽
#1CEh	1BB0100	D0=(5)	#0010B	
	D2	C=0	A	
	15C0	DAT0=C	1	❾
	870	?ST=1	0	
	90	GOYES	%1	
	304	C=(1)	#4	
	15C0	DAT0=C	1	
%1				
#1E7h	32004	C=(3)	#400	❿
	8FCEE10	GOSBVL	#01EEC	
	808AF	?CBIT=0	15	
	B0	GOYES	%2	
	11B	C=R3		
	D5	B=C	A	

❶ Boucle sur 9 lignes.
❷ Sauvegarde du début du tableau suivant.
❸ On calcule l'emplacement sur l'écran.
❹ Affiche le gobeur.
❺ Calcul emplacement sur l'écran.
❻ Affiche un carré.
❼ La pièce mobile courante est le gobeur (et non le carré).
❽ Procédure $2 : test des touches.
❾ On supprime le témoin α (sauf si on déplace le gobeur).
❿ Point de branchement %1. Test pression sur [ON], reprise au début.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	618E	GOTO	$1	
%2				❶
#203h	D0	A=0	A	
	32040	C=(3)	#040	
	8FCEE10	GOSBVL	#01EEC	❷
	808A4	?CBIT=0	4	
	60	GOYES	%3	
	81B2	PC=A		
%3				❸
#21Ch	808A0	?CBIT=0	0	
	60	GOYES	%4	❹
	E4	A=A+1	A	
	E4	A=A+1	A	
%4				❺
#227h	808A2	?CBIT=0	2	
	60	GOYES	%5	❻
	CC	A=A-1	A	
	CC	A=A-1	A	
%5				❼
#232h	8AC	?A≠0	A	
	06	GOYES	$3	
	808A1	?CBIT=0	1	
	A0	GOYES	%6	❽
	D2	C=0	A	
	31EE	C=(2)	#EE	
	CA	A=A+C	A	
%6				❾
#246h	32080	C=(3)	#080	
	8FCEE10	GOSBVL	#01EEC	
	808A1	?CBIT=0	1	❿

❶ Point de branchement %2.
❷ Test : pression sur la touche [STO] ? Si oui, RESET système.
❸ Point de branchement %3.
❹ Test : pression sur la touche [→] ?
❺ Point de branchement %4.
❻ Test : pression sur la touche [←] ?
❼ Point de branchement %5.
❽ Test : pression sur la touche [↓] ?
❾ Point de branchement %6.
❿ Test : pression sur la touche [↑] ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	A0	GOYES	%7	
	D2	C=0	A	
	31EE	C=(2)	#EE	
	EA	A=A-C	A	
%7				❶
#261h	8AC	?A≠0	A	
	13	GOYES	$3	❷
	808A0	?CBIT=0	1	
	62	GOYES	%11	
	860	?ST=0	1	
	80	GOYES	%8	
	840	ST=0	1	
	550	GONC	%9	
%8				❸
#278h	850	ST=1	1	
%9				❹
#27Bh	DA	A=C	A	
	330006	C=(4)	#6000	
%10				❺
#283h	CE	C=C-1	A	
	5DF	GONC	%10	❻
	3151	C=(2)	#15	
	962	?A=C	B	❼
	47	GOYES	$4	
%11				❽
#291h	6C3F	GOTO	$2	
$3				❾
#295h	119	C=R1		
	860	?ST=0	0	❿

❶ Point de branchement %7.
❷ Test : un déplacement a-t-il été demandé ? Test : pression sur [NXT] ?
❸ Point de branchement %8.
❹ Point de branchement %9.
❺ Point de branchement %10.
❻ Temporisation.
❼ Mode « tricheur » demandé ? ([PRG]-[VAR]-[NXT]).
❽ Point de branchement %11.
❾ Procédure $3 : test collision.
❿ On pointe sur le gobeur ou sur le carré.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	50	GOYES	%1	
	11A	C=R2		
%1				❶
#2A0h	DE	ACex	A	
	06	RSTK=C		
	CA	A=A+C	A	
	D2	C=0	A	
	3198	C=(2)	#89	❷
	CA	A=A+C	A	
	130	D0=A		
	14A	A=DAT0	B	❸
	30A	C=(1)	#A	
	906	?A≠C	P	❹
	60	GOYES	%3	
%2				❺
#2BBh	611F	GOTO	$2	
%3				❻
#2C0h	80870	?ABIT=1	0	
	7F	GOYES	%2	❼
	80861	?ABIT=0	1	
	C4	GOYES	$5	
	870	?ST=1	0	
	BE	GOYES	%2	❽
	861	?ST=0	1	
	72	GOYES	%6	❾
	3104	C=(2)	#40	
	DA	A=C	A	
%4				❿
#2DEh	32008	C=(3)	#800	
	801	OUT=C		

❶ Point de branchement %1. Ajout et sauvegarde du décalage.
❷ Descente de 4 lignes pour identification.
❸ Lecture de l'écran (case d'arrivée du déplacement).
❹ Est-ce un bord ?
❺ Point de branchement %2. Fin du déplacement.
❻ Point de branchement %3.
❼ Test : est-ce le gobeur ou le carré ? Test : est-ce une pastille ?
❽ Arrêt si le carré est utilisé.
❾ Le son est-il autorisé ? Durée et fréquence du son.
❿ Point de branchement %4. Bip avec volume moyen.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	6300	NOP4		
	D2	C=0	A	
	801	OUT=C		
	D6	C=A	A	
%5				❶
#2F1h	A6E	C=C-1	B	
	5CF	GONC	%5	
	A6C	A=A-1	B	❷
	53E	GONC	%4	
%6				❸
#2FDh	A0F	D=D-1	P	
	571	GONC	$5	
$4				❹
#303h	D9	C=B	A	
	135	D1=C		❺
	14B	A=DAT1	B	
	90C	?A≠0	P	❻
	60	GOYES	%7	
#310h	636D	GOTO	$0	
%7				❼
#314h	6C6D	GOTO	$1	
$5				❽
#318h	111	A=R1		
	860	?ST=0	0	❾
	50	GOYES	%1	
	112	A=R2		
%1				❿
#323h	D2	C=0	A	

❶ Point de branchement %5. Temporisation fréquence.
❷ Boucle durée, fréquence du son plus élevée.
❸ Point de branchement %6. Une pastille supprimée.
❹ Procédure $4 : mode « tricheur ».
❺ On pointe sur la tableau suivant.
❻ Y a-t-il des pastilles à gober dans le nouveau tableau ?
❼ Point de branchement %7. Si pas de pastille, tableau suivant.
❽ Procédure $5. déplacement.
❾ On pointe sur le gobeur ou sur le carré.
❿ Point de branchement %1. Ajout d'un décalage.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	3144	C=(2)	#44	
	CA	A=A+C	A	
	130	D0=A		
	303	C=(1)	#3	❶
%2				❷
#331h	06	RSTK=C		
	14A	A=DAT0	B	
	3118	C=(2)	#81	❸
	0E66	A=A&C	B	
	148	DAT0=A	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❹
	161	D0=D0+	#2	
	07	C=RSTK		
	A0E	C=C-1	P	❺
	51E	GONC	%2	
	07	C=RSTK		
	06	RSTK=C		
	132	AD0ex		
	CA	A=A+C	A	
	D2	C=0	A	❻
	31CC	C=(2)	#CC	
	EA	A=A-C	A	
	130	D0=A		
	860	?ST=0	0	
	80	GOYES	%3	
	102	R2=A		❼
	550	GONC	%4	
%3				❽
#371h	101	R1=A		

❶ Quatre lignes.
❷ Point de branchement %2. Sauvegarde du compteur.
❸ On ne conserve que les extrémités.
❹ Passage à la ligne du dessous.
❺ Boucle sur 4 lignes.
❻ On pointe sur la case d'arrivée.
❼ Modification gobeur ou carré.
❽ Point de branchement %3.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%4				❶
#374h	D2	C=0	A	
	870	?ST=1	0	
	40	GOYES	%5	❷
	E6	C=C+1	A	
%5				❸
#37Dh	7310	GOSUB	$6	
	32009	C=(3)	#900	
%6				❹
#386h	A3E	C=C-1	X	
	5CF	GONC	%6	
	07	C=RSTK		
	DA	A=C	A	
#390h	640F	GOTO	$3	❺
$6				❻
#394h	D0	A=0	A	
	A8A	A=C	P	❼
	137	CD1ex		
	06	RSTK=C		❽
	34A4000	C=(5)	#0004A	
	F0	ASL	A	
	C2	C=C+A	A	
	81B4	A=PC		❾
#3ADh	CA	A=A+C	A	
	131	D1=A		
	305	C=(1)	#5	
%1				❿
#3B5h	06	RSTK=C		
	14B	A=DAT1	B	

❶ Point de branchement %4.
❷ Affichage du gobeur ou du carré.
❸ Point de branchement %5.
❹ Point de branchement %6. Temporisation.
❺ On poursuit le déplacement.
❻ Procédure $6 d'affichage des sprites.
❼ Lecture du sprite à afficher.
❽ Sauvegarde de D1.
❾ D1 désigne la description du sprite.
❿ Point de branchement %1.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	14E	C=DAT0	B	❶
	0E6E	A=A!C	B	
	148	DAT0=A	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❷
	161	D0=D0+	#2	
	171	D1=D1+	#2	
	07	C=RSTK		
	A0E	C=C-1	P	❸
	5FD	GONC	%1	
	07	C=RSTK		
	135	D1=C		❹
	01	RTN		
$BORD				❺
#3DFh	EF092	*Données graphiques*		
	13C1D	*Données graphiques*		
	013A1	*Données graphiques*		
	94152	*Données graphiques*		
	1EF0	*Données graphiques*		
$SPRITES				❻
#3F7h	0000C	*Données graphiques*		
	3C3C3	*Données graphiques*		
	C3000	*Données graphiques*		
	00000	*Données graphiques*		
	81C2C	*Données graphiques*		
	38100	*Données graphiques*		
	00000	*Données graphiques*		
	08142	*Données graphiques*		
	4281	*Données graphiques*		
#423h				❼

❶ Transfert vers l'écran par « ou » logique.
❷ Passage à la ligne du dessous.
❸ Boucle sur 6 lignes.
❹ Restauration de D1.
❺ Procédure $BORD. Données graphiques décrivant les obstacles.
❻ Procédure $SPRITES. Données graphiques décrivant les pièces mobiles.
❼ Fin du code en #423h.

3

Exécuter le programme

Pour exécuter le programme, vous devez réunir dans le répertoire DSTAR (spécialement créé), les trois objets suivants :

- 'STAR', le lanceur RPL,
- 'TABL', l'objet graphique,
- 'CODE', le code exécutable

Les objets de développement tels que la chaîne de codes hexadécimaux et les variables 'T1' à 'T4' peuvent rester dans le répertoire DSTAR mais ne seront plus utilisées.

Pour jouer, il suffit ensuite de lancer l'exécution du programme RPL 'STAR'. Vous pouvez activer ou non le son, pour cela activez ou désactivez le flag bip RPL.

4

Comment jouer ?

Le but du jeu est "l'absorption" par le gobeur (le rond plein) de toutes les pastilles (les ronds vides). Le carré permet de se sortir des situations difficiles en offrant des « surfaces de rebondissement mobiles ». Votre gobeur et le carré ne peuvent se déplacer que d'une paroi vers une autre paroi. Votre gobeur avale tout sur son passage. Pour jouer, utilisez les touches suivantes :

- les quatre flèches [←], [↑], [→] et [↓] permettent de déplacer votre gobeur ou le carré,
- la touche [NXT] permet de passer de la manipulation du gobeur à celle du carré (et inversement). Lorsque le témoin α est allumé, cela signifie que vous déplacez le gobeur (ce témoin est éteint quand vous manipulez le carré),
- une pression sur [ON] permet de reprendre à partir du début du tableau,
- une pression sur [STO] permet de quitter le jeu.

Les tricheurs et les joueurs un peu perdus seront heureux d'apprendre qu'en appuyant simultanément sur les touches [PRG]-[VAR]-[NXT] ils pourront passer directement au tableau suivant...

5
Créer vos propres tableaux

Bien que ce jeu soit livré avec 20 tableaux prêts à l'emploi, il vous est possible d'ajouter vos propres tableaux (leur nombre n'est pas limité). Rappelons qu'un quartet est codé par un nombre hexadécimal (de 0 à F). Chaque tableau est codé de la façon suivante :

- un quartet codant le nombre de pastilles à gober (ce qui limite à #Fh le nombre de pastilles par tableau !),
- trois quartets codant la position initiale du gobeur,
- trois quartets codant la position initiale du carré,
- neuf fois quatre quartets codant les positions des murs,
- neuf fois quatre quartets codant les positions des pastilles.

Pour coder la position initiale du gobeur ou du carré, utilisez la formule :

$$(\textit{colonne} - 1) \times 2 + (\textit{ligne} - 1) \times \#EEh$$

Ainsi, pour coder la sixième colonne de la troisième ligne, il suffit de calculer $(6\text{-}1) \times 2 + (3\ \text{-}1) \times \#EEh = 5 \times 2 + 2 \times \#EEh = \#1E6h$. Une fois les quartets inversés, on obtient #6E1h.

Un quartet est une association de quatre bits, chacun de ces quatre bits a pour poids 8, 4, 2 et 1 (respectivement). Quatre quartets sont consacrés au codage d'une ligne. Seize emplacements peuvent donc être codés avec quatre quartets.

Quartet 4	Quartet 3	Quartet 2	Quartet 1
8 4 2 1	*8 4 2 1*	*8 4 2 1*	*8 4 2 1*
■	■ ■		■ ■ ■
8	6	0	B (B_{16}=11_{10})

Ce tableau illustre le codage d'une ligne (avec six obstacles). Les quatre quartets #860Bh devront être inversés, on obtient ainsi #B068h. Les lignes sont à coder de haut en bas. On code d'abord neuf lignes d'obstacles puis neuf lignes de pastilles à gober. Le nombre de pastilles placées sur le tableau doit correspondre au nombre annoncé (voir le premier quartet de codage du tableau).

N'oubliez surtout pas d'ajouter un 0 après le dernier tableau ! Ajustez la taille du GROB à chaque ajout de tableau. Ainsi, si votre objet graphique a pour dimensions GROB 1 396 et que vous lui ajoutez un tableau, ajoutez 40 à 396 (en fait, il faudrait ajouter 39,5...). On obtient GROB 1 436.

6

Retrouver les fichiers sur la disquette

Les chemins d'accès aux objets utilisés sont les suivants :

- 'CODE' : DUNOD_1\S_SX\OB_P_OBS\STAR_S\CODE,
- 'STAR' : DUNOD_1\S_SX\OB_P_OBS\STAR_S\STAR,
- 'STAR48S' : DUNOD_1\S_SX\OB_P_OBS\STAR_S\STAR48S,
- 'TABL' : DUNOD_1\S_SX\OB_P_OBS\STAR_S\TABL,
- 'ASS' : DUNOD_1\S_SX\ASS,
- 'T1' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\T1,
- 'T2' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\T2,
- 'T3' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\T3,
- 'T4' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\T4.

30
Xalk pour HP-48 S et SX

1
Le jeu

1.1
Présentation

Xalk est un jeu de la célèbre famille des Tetris™ qui comblera tous les amateurs de jeux d'assemblage. Comme pour tous les jeux de ce type, des pièces tombent du ciel (cela dit, dans le cas de XALK, elles arrivent par la gauche...), il vous faut les assembler convenablement car les lignes de pièces identiques disparaissent. La place libérée permet d'accueillir de nouvelles pièces... Vous perdez quand un empilement de pièces atteint la largeur de l'écran. Trois pièces identiques disparaissent lorsqu'elles sont alignées horizontalement ou verticalement. Pour augmenter l'intérêt du jeu, des coups spéciaux (cascade, combinaison, etc.) imposent l'élaboration d'une véritable stratégie. Selon le niveau choisi, de 5 à 8 figures différentes vous seront proposées, la difficulté du jeu dépend directement de ce nombre de figures.

Pour la petite histoire, Xalk est une adaptation de Klax™ sur Amstrad CPC. Klax™ est lui-même un parent éloigné de Tetris™... La généalogie des jeux informatiques existe !

1.2
Liste des objets

Six objets sont associés à la mise en œuvre de ce jeu :

- 'XS2' est la chaîne de codes hexadécimaux qui devra être traitée à l'aide du programme ASS afin d'obtenir l'objet 'CODE',
- 'CODE' est le programme assembleur proprement dit, il est obtenu en traitant 'XS2' à l'aide de ASS,
- 'XALK' est le programme lanceur RPL,
- 'SPR' est un objet graphique (sprites),
- 'CHA' est un objet graphique (chiffres).

'XS2' et 'ASS' ne sont pas nécessaires à l'exécution du programme. Seules les variables 'XALK' (lanceur RPL), 'CODE' (code exécutable), 'SPR' (objet graphique) et 'CHA' (objet graphique) sont nécessaires pour jouer.

2
Les objets

2.1
Le lanceur RPL

Cet objet est à enregistrer dans une variable nommée 'XALK'. Saisissez-le entre « et » qui sont les délimiteurs d'un programme RPL.

```
« DO CLEAR
"XALK 48S/SX"
":NIVEAU 1→4:"
INPUT OBJ→ DTAG
  UNTIL { 1 2 3 4 }
OVER POS
  END #0h DUP DUP
DUP2 PDIM 2 →LIST
```

```
PVIEW PICT { #0h
#37h } "SCORE:" 3
→GROB REPL + 3 +
CHA SPR CODE
 »
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #BF18h,
- au niveau 1, une longueur égale à 191 octets.

2.2
Les objets graphiques

Attention ! Pendant la saisie des objets graphiques (**GROB...**) seuls les trois premiers espaces sont à saisir !

■ Le codage des chiffres

Cet objet est à enregistrer dans une variable nommée 'CHA'. Cet objet est à saisir sans délimiteur ni saut de ligne.

```
GROB 1 90 E0111 11111 11111 1E040 60404 04040
          4040E 0E011 11804 02010 10F1E 01101
          01E00 10111 E080C 0A090 F1808 08080
          F1101 0F001 01011 1E0C0 20101 0F011
          1111E 0F101 01804 02020 2020E 01111
          11E01 11111 E0E01 11111 E1010 18060
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #7F25h,
- au niveau 1, une longueur égale à 100 octets.

■ Le codage des sprites

Cet objet est à enregistrer dans une variable nommée 'SPR'. Cet objet est à saisir sans délimiteur ni saut de ligne.

```
GROB 1 80 F7141 41414 14141 414F7 F755B 655B6
          55B65 5B6F7 F7F7F 7F7F7 F7F7F 7F7F7
          F794D 514A2 A214D 594F7 F741E 38063
          6380E 341F7 F7D59 43677 77369 4D5F7
          F7360 0C122 22C10 036F7 E3B65 55555
          55555 5B6E3
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #1D49h,
- au niveau 1, une longueur égale à 90 octets.

2.3
Le code hexadécimal

a
La chaîne à saisir

Cette chaîne de codes hexadécimaux est à enregistrer dans une variable nommée 'XS2'. Saisissez la chaîne sans espace ni saut de lignes mais entre deux délimiteurs " et ".

```
CCD20 02600 84F80 8F082 01471 08174
14710 91741 47135 17914 F1F50 20714
D75D4 AF219 C0175 15C01 55719 E21CC
15C01 45177 326C6 1451E 2D601 4F808
B3508 561E6 02014 FF614 D1C01 4F7F7
4D1D3 7234E 5E57A 24840 84484 51711
471C1 06324 008FC EE108 08A2B 21B00
100D2 15C03 22008 FCEE1 0808A 2FE30
815C0 D2808 B5808 415C2 87172 85130
4901C 1E57D 63CD1 4A96C C07C8 3E573
A3328 008FC EE108 08B08 08425 11872
C0850 85265 80808 B2468 08B58 08435
B2873 62853 D28A1 C1CD7 803E5 14A96
CC077 23CD7 E3307 CE404 6E1FD 1D228
80F28 66508 01D4C C5DFB 6559E 20328
008FC EE108 08A2F ED206 00E77 9A2CF
14A96 CD178 C2E77 FD2DB B064B 0870B
```

```
D67AE 7704D 29032 91C0A F215F 0818F
A2711 322D2 302D5 301D7 F7308 866A0
801D2 801D4 CC5DF CF56E 20D23 04D51
DD1D3 E7742 216F1 6F161 14E14 D170B
0756E CD5DD D2304 D5D3E 7D4F0 E0339
520E2 13D30 E98BE 314BE 71709 08DE1
4F906 5EE71 7014F 9068D 7AE1C F74E1
CF7ED 1D231 F47DF 2CD54 AD3E7 33A32
0CB13 D14B9 08361 7E14F 90607 17E14
F1CE1 CE1CE 906D4 14F90 6E11C E14F9
0631D 2304D 57B71 CD757 1D2D5 7D61E
57761 E5716 1D231 F4708 2B075 88423
14F90 61FD2 E65EC 1CE1C E14F9 06CD1
CE14F 9061D D2302 5DA87 46068 B0D23
04D5D 230FD 776D0 14A90 CD318 114A9
08238 55309 14A16 11481 81D01 4816F
16F16 1A0E5 2ED23 1F179 E1CF3 01907
1BCD5 5A865 90845 644E1 E5020 AF215
F0818 FA205 C6C67 7011C 41433 20518
BAA18 18F86 1411C 70514 3E414 104D1
34BA3 00D77 42016 01ED0 20305 74D01
691E0 02030 274C0 6BFB1 BB550 7146C
BCBD4 C4E2C 4C4E2 F0C2C 4C213 416E1
64017 FCFD0 30914 816F1 6F161 A0E50
F0172 BFD21 5F0C6 C6110 CAC6C 6CA13
117E1 74309 14B14 816F1 6F161 171A0
E5AE1 E6020 011B8 31001 4A808 43986
5F171 15900 11770 51537 A7A15 17040
11537 D7CE8 14A0E 59FAF 606D1 A8511
1C5C0 81DF1 C0131 17E17 43080 614BD
B808B 0B0D2 15435 60C4C 414E0 E6A14
C16F1 6F161 17107 A0E59 C3413 10013
2EA13 0DB80 8A050 16007 812A0 F5280
1854D ADD06 2880F 28665 08010 6D6CE
5DF07 CD56E 2007D 5D2B3 65205 9F01
```

Placez dans la pile une copie de la chaîne de caractères stockée dans 'XS2', activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #9DF1h,
- au niveau 1, une longueur égale à 1564 octets.

Pour générer l'objet 'CODE' à partir de la chaîne 'XS2' :

- placez la chaîne de codes hexadécimaux (variable 'XS2') au niveau 1 de la pile, `"CCD200260084F808F...` s'affiche,
- exécutez le programme ASS,

- au niveau 1 de la pile doit apparaître **Code**,
- si ce n'est pas encore fait, placez-vous dans le répertoire DXALK,
- saisissez le nom **'CODE'**,
- appuyez sur la touche [STO],
- appuyez sur la touche [VAR], le nom **CODE** doit apparaître dans la liste des variables.

Rappelons que pour placer 'CODE' dans la pile sans l'évaluer, vous devez appuyer sur [↱] avant de choisir l'article CODE du menu. Vérifiez l'intégrité de la variable 'CODE' en plaçant au niveau 1 de la pile une copie de cet objet (**Code** doit apparaître au niveau 1), activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #4FC2h,
- au niveau 1, une longueur égale à 786,5 octets.

Codes commentés

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$$				❶
	CCD20			❷
#000h	02600			
	84F	ST=0	15	❸
	808F	INTOFF		
	08	CLRST		❹
	20	P=	0	
	147	C=DAT1	A	
	108	R0=C		❺
	174	D1=D1+	#5	
	147	C=DAT1	A	
	109	R1=C		❻
	174	D1=D1+	#5	

❶ Début de la procédure d'initialisation.
❷ Prologue du code exécutable puis longueur totale hors prologue.
❸ Inhibition des interruptions.
❹ ST1/2/3=0. Boucles champ P, C=(x).
❺ R0 contient l'adresse du GROB sprites figures.
❻ R1 contient l'adresse du GROB sprites chiffres.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	147	C=DAT1	A	
	135	D1=C		❶
	179	D1=D1+	#A	
	14F	C=DAT1	B	
	1F50207	D1=(5)	#70205	
	14D	DAT1=C	B	❷
	75D4	GOSUB	$NEXT	
#03Ch	AF2	C=0	W	
	19C0	D0=(2)	#0C	
	175	D1=D1+	#6	
	15C0	DAT0=C	1	❸
	1557	DAT1=C	W	
	19E2	D0=(2)	#2E	❹
	1CC	D1=D1-	#D	
	15C0	DAT0=C	1	
	145	DAT1=C	A	❺
	177	D1=D1+	#8	
	326C6	C=(3)	#6C6	
	145	DAT1=C	A	❻
	1E2D60	D1=(4)	#06D2	
	14F	C=DAT1	B	
	808B3	?CBIT=1	3	❼
	50	GOYES	$0	
	856	ST=1	6	❽
$0				❾
#07Ah	1E6020	D1=(4)	#0206	
	14F	C=DAT1	B	❿
	F6	CSR	A	
	14D	DAT1=C	B	

❶ Lecture du niveau 1 de la pile RPL (c'est un nombre hexadécimal).
❷ On stocke le niveau puis la nature de la prochaine figure.
❸ Suppression du sablier (témoin). Initialisation du score à 0.
❹ On pointe en #0012Eh.
❺ Vitesse (affichage) en #70200h.
❻ Vitesse (compteur) en #70208h.
❼ Test du flag BEEP RPL.
❽ Si le flag BEEP est armé, on a ST6=1.
❾ Procédure $0 (traitement de la figure à venir).
❿ La figure « à venir » devient la figure courante.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1C0	D1=D1-	#1	
	14F	C=DAT1	B	❶
	7F74	GOSUB	$NEXT	
#092h	D1	B=0	A	
	D3	D=0	A	❷
	7234	GOSUB	$SPR1	
#09Ah	E5	B=B+1	A	
	E5	B=B+1	A	❸
	7A24	GOSUB	$SPR1	
$1				❹
#0A2h	840	ST=0	0	❺
	844	ST=0	4	
	845	ST=0	5	
	171	D1=D1+	#2	
	147	C=DAT1	A	❻
	1C1	D1=D1-	#2	
%1				❼
#0B4h	06	RSTK=C		
	32400	C=(3)	#004	
	8FCEE10	GOSBVL	#01EEC	❽
	808A2	?CBIT=0	2	
	B2	GOYES	%3	
	1B00100	D0=(5)	#00100	
	D2	C=0	A	
	15C0	DAT0=C	1	❾
%2				❿
	32200	C=(3)	#002	
	8FCEE10	GOSBVL	#01EEC	
	808A2	?CBIT=0	2	

❶ Lecture du niveau et stockage du suivant.
❷ Affichage de la figure à venir (en haut, à gauche).
❸ Affichage de la figure courante (à mi-hauteur, à gauche).
❹ Procédure $1 (test des touches).
❺ Initialisation flags (pression sur [÷], triplets, cascades).
❻ Charge la vitesse (compteur).
❼ Point de branchement %1 (sauvegarde de la temporisation).
❽ [←] et [5]. Teste une pression sur [5] (déclenchement d'une pause).
❾ Extinction de l'écran.
❿ Point de branchement %2. Attente d'une pression sur [2].

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	FE	GOYES	%2	
	308	C=(1)	#8	
	15C0	DAT0=C	1	❶
	D2	C=0	A	
%3				❷
	808B5	?CBIT=1	5	
	80	GOYES	%4	❸
	841	ST=0	1	
	5C2	GONC	%5	
%4				❹
	871	?ST=1	1	
	72	GOYES	%5	
	851	ST=1	1	❺
	304	C=(1)	#4	
	901	?B=C	P	
	C1	GOYES	%5	❻
	E5	B=B+1	A	
	7D63	GOSUB	$SPRCALC	
#115h	CD	B=B-1	A	
	14A	A=DAT0	B	
	96C	?A≠0	B	❼
	C0	GOYES	%5	
	7C83	GOSUB	$SPRO	
#123h	E5	B=B+1	A	
	73A3	GOSUB	$SPR1	
%5				❽
#129h	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	❾
	808B0	?CBIT=1	0	
	80	GOYES	%6	❿

❶ Rallume l'écran.
❷ Point de branchement %3.
❸ Teste une pression sur [↰] puis réactive [↰].
❹ Point de branchement %4. Test : [↰] déjà pressée ?
❺ Désactive la touche [↰].
❻ La figure est-elle sur le bord inférieur ?
❼ Test : case vide au-dessous ? Efface la figure et la replace à droite.
❽ Point de branchement %5.
❾ [α], [8] et [÷].
❿ Test : pression sur [÷] ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	842	ST=0	2	
	511	GONC	%7	
%6				❶
	872	?ST=1	2	
	C0	GOYES	%7	❷
	850	ST=1	0	❸
	852	ST=1	2	
#14Dh	6580	GOTO	$3	
%7				❹
	808B2	?CBIT=1	2	
	46	GOYES	$2b	
	808B5	?CBIT=1	5	
	80	GOYES	%8	❺
	843	ST=0	3	
	5B2	GONC	%9	
%8				❻
	873	?ST=1	3	
	62	GOYES	%9	❼
	853	ST=1	3	
	D2	C=0	A	
	8A1	?B=C	A	
	C1	GOYES	%9	❽
	CD	B=B-1	A	
	7803	GOSUB	$SPRCALC	
#17Ah	E5	B=B+1	A	
	14A	A=DAT0	B	
	96C	?A≠0	B	❾
	C0	GOYES	%9	
	7723	GOSUB	$SPRO	❿
#188h	CD	B=B-1	A	

❶ Point de branchement %6.
❷ Test : [÷] déjà pressée ?
❸ Inhibition des tests du clavier. Désactivation de [÷].
❹ Point de branchement %7. Test : pression sur [8] ?
❺ Test : pression sur [α] ?
❻ Point de branchement %8.
❼ Test : [α] déjà pressée ? Désactivation de [α].
❽ Test : la figure a-t-elle atteint le bord supérieur ?
❾ Test : case vide au-dessus ?
❿ On efface la figure courante puis on l'affiche à nouveau au-dessus.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	7E33	GOSUB	$SPR1	
%9				❶
#18Eh	07	C=RSTK		
	CE	C=C-1	A	
	404	GOC	$3	
	6E1F	GOTO	%1	
$2				❷
%a				❸
#199h	D1	B=0	A	
	D2	C=0	A	
	28	P=	8	
%1				❹
	80F2	CPex	2	
	866	?ST=0	6	
	50	GOYES	%2	❺
	801	OUT=C		
%2				❻
	D4	A=B	A	
%3				❼
	CC	A=A-1	A	
	5DF	GONC	%3	❽
	B65	B=B+1	B	
	59E	GONC	%1	❾
	20	P=	0	
%b				❿
#1BAh	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	
	808A2	?CBIT=0	2	
	FE	GOYES	%b	

❶ Point de branchement %9. Temporisation.
❷ Procédure $2 (fin du jeu).
❸ Point de branchement %a.
❹ Point de branchement %1. Alterne OUT #800 et OUT#000...
❺ ...sauf si flag BEEP RPL désactivé.
❻ Point de branchement %2.
❼ Point de branchement %3.
❽ Fréquence.
❾ Durée.
❿ Point de branchement %b. Attente d'une pression sur [8].

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D2	C=0	A	
	06	RSTK=C		
	00	RTNSXM		❶
$3				❷
#1D3h	E7	D=D+1	A	
	79A2	GOSUB	$SPRCALC	
#1D9h	CF	D=D-1	A	
	14A	A=DAT0	B	
	96C	?A≠0	B	❸
	D1	GOYES	$4	
	78C2	GOSUB	$SPRO	❹
#1E7h	E7	D=D+1	A	
	7FD2	GOSUB	$SPR1	
#1EDh	DB	C=D	A	
	B06	C=C+1	P	❺
	4B0	GOC	$4	
	870	?ST=1	0	
	BD	GOYES	$3	❻
	67AE	GOTO	$1	❼
$4				❽
#1FEh	7704	GOSUB	$TEMPO	
	D2	C=0	A	
	903	?C=D	P	
	29	GOYES	$2a	❾
	1C0	D1=D1-	#1	
	AF2	C=0	W	
	15F0	C=DAT1	1	❿
	818FA2	C=C-3	A	

❶ Reset système.
❷ Procédure $3 (décalage vers la droite de la figure).
❸ Test : y a-t-il déjà un cube sur la droite ?
❹ Affichage de la figure courante un cran à droite.
❺ Test : D=15 ? Est-on sur l'extrémité droite de l'écran.
❻ On teste les touches...
❼ ...sauf si la touche [÷] est pressée.
❽ Procédure $4 (pose du cube).
❾ Test : une figure est-elle bloquée à l'extrémité gauche ? Oui, alors fin.
❿ Lecture du niveau.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	7113	GOSUB	$SCORE	❶
#21Dh	22	P=	2	
	D2	C=0	A	
	302	C=(1)	#2	
	D5	B=C	A	❷
	301	C=(1)	#1	
	D7	D=C	A	
	F7	DSR	A	
%1				❸
	308	C=(1)	#8	
	866	?ST=0	6	
	A0	GOYES	%2	❹
	801	OUT=C		❺
	D2	C=0	A	
	801	OUT=C		
%2				❻
	D4	A=B	A	
%3				❼
	CC	A=A-1	A	
	5DF	GONC	%3	
	CF	D=D-1	A	
	56E	GONC	%1	❽
	20	P=	0	
$5				❾
#24Bh	D2	C=0	A	
	304	C=(1)	#4	
	D5	B=C	A	❿
	1DD1	D1=(2)	#1D	

❶ Modification du score.
❷ Fréquence copiée dans B.
❸ Point de branchement %1. Durée dans D.
❹ Emission d'un son sauf si le flag BEEP RPL est désactivé.
❺ Emission d'un son avec volume faible.
❻ Point de branchement %2.
❼ Point de branchement %3. Fréquence.
❽ Durée.
❾ Procédure $5 (création du tableau de traitement des cascades).
❿ Création du tableau en #7021D.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%1				❶
	D3	D=0	A	
	E7	D=D+1	A	
%2				❷
	7422	GOSUB	$SPRCALC	
#25Eh	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❸
	161	D0=D0+	#2	
	14E	C=DAT0	B	
	14D	DAT1=C	B	❹
	170	D1=D1+	#1	
	B07	D=D+1	P	
	56E	GONC	%2	❺
	CD	B=B-1	A	
	5DD	GONC	%1	❻
$6				❼
#27Bh	D2	C=0	A	
	304	C=(1)	#4	
	D5	B=C	A	
%1				❽
	D3	D=0	A	
	E7	D=D+1	A	
	D4	A=B	A	
	F0	ASL	A	
	E0	A=A-B	A	❾
339520	C=(4)	#0259		
	E2	C=C-A	A	
	13D	D1=CS		❿

❶ Point de branchement %1.
❷ Point de branchement %2. On pointe sur le sprite.
❸ On descend d'une ligne pour identification.
❹ Transfert écran/tableau. Donnée suivante.
❺ Boucle colonne.
❻ Boucle ligne.
❼ Procédure $6 (recherche horizontale).
❽ Point de branchement %1. Première colonne.
❾ A = B × 15.
❿ On pointe sur le tableau créé en $5.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%2				❶
	30E	C=(1)	#E	
	98B	?C≤D	P	❷
	E3	GOYES	%3	
	14B	A=DAT1	A	
	E7	D=D+1	A	
	170	D1=D1+	#1	❸
	908	?A=0	P	
	DE	GOYES	%2	❹
	14F	C=DAT1	B	
	906	?A≠C	P	
	5E	GOYES	%2	❺
	E7	D=D+1	A	
	170	D1=D1+	#1	
	14F	C=DAT1	B	❻
	906	?A≠C	P	
	8D	GOYES	%2	❼
	7AE1	GOSUB	$SPRO	
#				
2C5h	CF	D=D-1	A	
	74E1	GOSUB	$SPRO	
#2CBh	CF	D=D-1	A	
	7ED1	GOSUB	$SPRO	❽
#2D1h	D2	C=0	A	
	31F4	C=(2)	#4F	
	7DF2	GOSUB	$BEEP	❾
%3				❿
#2DBh	CD	B=B-1	A	
	54A	GONC	%1	

❶ Point de branchement %2.
❷ On s'arrête deux figures avant la fin de la ligne.
❸ On pointe sur la figure suivante.
❹ On recherche la figure suivante (recherche vers la gauche).
❺ ...identité ?
❻ Lecture de la figure à gauche de la figure courante.
❼ ...identité ?
❽ On efface le triplet.
❾ Son et temporisation.
❿ Point de branchement %10. Boucle ligne.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$7				❶
#2E0h	D3	D=0	A	
	E7	D=D+1	A	❷
%1				❸
	33A320	C=(4)	#023A	
	CB	C=C+D	A	
	13D	D1=CS		❹
	14B	A=DAT1	B	
	908	?A=0	P	
	36	GOYES	%4	❺
	17E	D1=D1+	#F	
	14F	C=DAT1	B	❻
	906	?A≠C	P	
	07	GOYES	%6	❼
	17E	D1=D1+	#F	
	14F	C=DAT1	B	❽
	1CE	D1=D1-	#F	
	1CE	D1=D1-	#F	
	1CE	D1=D1-	#F	❾
	906	?A≠C	P	
	D4	GOYES	%5	❿
	14F	C=DAT1	B	
	906	?A≠C	P	
	E1	GOYES	%2	
	1CE	D1=D1-	#F	
	14F	C=DAT1	B	
	906	?A≠C	P	
	31	GOYES	%2	
	D2	C=0	A	
	304	C=(1)	#4	

❶ Procédure $7 (recherche verticale d'un triplet).
❷ On pointe sur la première colonne.
❸ Point de branchement %1.
❹ On pointe sur le tableau créé en $5 (lecture de B2).
❺ Test : cette position est-elle occupée ?
❻ Lecture de la figure B1 (au-dessus de B2).
❼ Test B1 = B2 ?
❽ Lecture de B0.
❾ Préparation de B3.
❿ Tests : B0 = B2 ? B2 = B3 ? B2 = B4 ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D5	B=C	A	❶
	7B71	GOSUB	$SPRO	❷
#334h	CD	B=B-1	A	
	7571	GOSUB	$SPRO	❸
%2				❹
#33Ah	D2	C=0	A	
%3				❺
	D5	B=C	A	
	7D61	GOSUB	$SPRO	
#342h	E5	B=B+1	A	
	7761	GOSUB	$SPRO	
#348h	E5	B=B+1	A	
	7161	GOSUB	$SPRO	
#34Eh	D2	C=0	A	
	31F4	C=(2)	#4F	
	7082	GOSUB	$BEEP	❻
%4				❼
#358h	B07	D=D+1	P	
	588	GONC	%1	
	423	GOC	$8	
%5				❽
	14F	C=DAT1	B	
	906	?A≠C	P	
	1F	GOYES	%4	❾
	D2	C=0	A	
	E6	C=C+1	A	
	5EC	GONC	%3	
%6				❿
	1CE	D1=D1-	#F	
	1CE	D1=D1-	#F	

❶ Recherche d'un coup spécial : toute la ligne avec les mêmes figures ?
❷ Efface B4.
❸ Efface B3.
❹ Point de branchement %2. Efface B0, B1 et B2.
❺ Point de branchement %3. Efface le triplet pointé par B.
❻ Bip et temporisation.
❼ Point de branchement %4. Boucle sur colonne.
❽ Point de branchement %5.
❾ Test : B2 = B3 ? On efface B1, B2 et B3.
❿ Point de branchement %6.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	14F	C=DAT1	B	
	906	?A≠C	P	
	CD	GOYES	%4	❶
	1CE	D1=D1-	#F	
	14F	C=DAT1	B	
	906	?A≠C	P	
	1D	GOYES	%4	❷
	D2	C=0	A	
	302	C=(1)	#2	
	5DA	GONC	%3	❸
$8				❹
#391h	874	?ST=1	4	
	60	GOYES	%1	❺
#396h	68B0	GOTO	$10	
%1				❻
	D2	C=0	A	
	304	C=(1)	#4	
	D5	B=C	A	
%2				❼
	D2	C=0	A	
	30F	C=(1)	#F	❽
	D7	D=C	A	
%3				❾
	76D0	GOSUB	$SPRCALC	
#3AC	14A	A=DAT0	B	
	90C	?A≠0	P	
	D3	GOYES	%5	❿
	181	D0=D0-	#2	
	14A	A=DAT0	B	

❶ Test : B2 = B3 ?
❷ Test : B2 = B4 ?
❸ Efface B2, B3 et B4.
❹ Procédure $8 (recherche du coup spécial cascade).
❺ Test : y a-t-il des triplets ?
❻ Point de branchement %1.
❼ Point de branchement %2.
❽ La recherche commence par la colonne de droite.
❾ Point de branchement %3. Lecture des figures.
❿ Recherche d'une case vide.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	908	?A=0	P	❶
	23	GOYES	%5	
	855	ST=1	5	❷
	309	C=(1)	#9	
%4				❸
	14A	A=DAT0	B	
	161	D0=D0+	#2	
	148	DAT0=A	B	❹
	181	D0=D0-	#2	
	D0	A=0	A	
	148	DAT0=A	B	❺
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	❻
	A0E	C=C-1	P	
	52E	GONC	%4	
	D2	C=0	A	
	31F1	C=(2)	#1F	
	79E1	GOSUB	$BEEP	❼
%5				❽
#3EFh	CF	D=D-1	A	
	301	C=(1)	#1	
	907	?C≠D	P	
	1B	GOYES	%3	❾
	CD	B=B-1	A	
	55A	GONC	%2	
$9				❿
#3FEh	865	?ST=0	5	
	90	GOYES	%1	

❶ Recherche d'un cube à la gauche de la case vide.
❷ Flag cascade.
❸ Point de branchement %4.
❹ Transfert de la figure.
❺ On efface la figure (position précédente).
❻ Passage à la ligne du dessous.
❼ Son plus court et plus aigu.
❽ Point de branchement %5.
❾ Boucle colonne puis boucle ligne.
❿ Procédure $9 (score et vitesse).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	845	ST=0	5	❶
	644E	GOTO	$5	❷
%1				❸
	1E5020	D1=(4)	#0205	
	AF2	C=0	W	
	15F0	C=DAT1	1	❹
	818FA2	C=C-3	A	
	05	SETDEC		
	C6	C=C+C	A	
	C6	C=C+C	A	
	7701	GOSUB	$SCORE	❺
#427h	1C4	D1=D1-	#5	
	143	A=DAT1	A	
	32051	C=(3)	#150	❻
	8BA	?A≤C	A	
	A1	GOYES	$10	
	818F86	A=A-7	A	
	141	DAT1=A	A	❼
	1C7	D1=D1-	#8	
	05	SETDEC		
	143	A=DAT1	A	
	E4	A=A+1	A	
	141	DAT1=A	A	❽
	04	SETHEX		
$10				❾
#44Fh	D1	B=1	A	
	34BA300	C=(5)	#003AB	
	D7	D=C	A	
	7420	GOSUB	$SPRCALC	❿

❶ Si cascade (coup spécial)...
❷ ...on recommence le traitement.
❸ Point de branchement %1.
❹ Lecture du niveau.
❺ Score = niveau × 5 (voir procédure $4.)
❻ La vitesse peut-elle encore être modifiée ?
❼ Modification de la vitesse (compteur).
❽ modification de la vitesse (affichage).
❾ Procédure $10 (affichage)
❿ Adresse sur écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#45Eh	160	D0=D0+	#1	
	1ED020	D1=(4)	#020D	❶
	305	C=(1)	#5	
	74D0	GOSUB	$AFF	
#46Eh	169	D0=D0+	#A	
	1E0020	D1=(4)	#0200	❷
	302	C=(1)	#2	
	74C0	GOSUB	$AFF	❸
#47Eh	6BFB	GOTO	$0	❹
#482h				
$SPRCALC				❺
#000h	1BB5507	D0=(5)	#7055B	
	146	C=DAT0	A	❻
	CB	C=C+D	A	
	CB	C=C+D	A	❼
	D4	A=B	A	
	C4	A=A+A	A	
	E2	C=C-A	A	
	C4	A=A+A	A	
	C4	A=A+A	A	
	E2	C=C-A	A	❽
	F0	ASL	A	
	C2	C=C+A	A	
	C4	A=A+A	A	
	C2	C=C+A	A	❾
	134	D0=C		
	16E	D0=D0+	#F	
	164	D0=D0+	#5	❿
	01	RTN		

❶ Données. Nombre de quartets (6). Affichage du score.
❷ Données. Nombre de quartets (3).
❸ Affichage de la vitesse.
❹ Retour au début du traitement.
❺ Procédure $SPRCALC (en #482h).
❻ Début de l'écran.
❼ + #2h × D puis - #2h × B.
❽ - #8h × B puis + #80h × B.
❾ + #100h × B.
❿ + #14h (pour le contenu) = D0...

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$SPRO				❶
#02Dh	7FCF	GOSUB	$SPRCALC	❷
	D0	A=0	A	
	309	C=(1)	#9	❸
%1				❹
	148	DAT0=A	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	❺
	A0E	C=C-1	P	
	50F	GONC	%1	
	01	RTN		
$SPR1				❻
#04Ah	72BF	GOSUB	$SPRCALC	❼
	D2	C=0	A	
	15F0	C=DAT1	1	
	C6	C=C+C	A	
	C6	C=C+C	A	
	110	A=R0		
	CA	A=A+C	A	
	C6	C=C+C	A	
	C6	C=C+C	A	❽
	CA	A=A+C	A	
	131	D1=A		
	17E	D1=D1+	#F	❾
	174	D1=D1+	#5	
	309	C=(1)	#9	
%1				❿
	14B	A=DAT1	B	

❶ Procédure $SPR0.
❷ D0 correspond à (B,D).
❸ 10 lignes.
❹ Point de branchement %1. On efface l'écran.
❺ Passage à la ligne inférieure puis boucle sur ligne.
❻ Procédure $SPR1.
❼ D0 correspond à [B,D] (numéro du sprite à chercher dans la table).
❽ D1=R0+C×#14h (longueur d'un sprite dans la table des sprites).
❾ Correction de R0 (qui pointe sur le contenu du GROB).
❿ Point de branchement %1.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	148	DAT0=A	B	❶
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	❷
	171	D1=D1+	#2	
	A0E	C=C-1	P	
	5AE	GONC	%1	❸
	1E6020	D1=(4)	#0206	
	01	RTN		
$NEXT				❹
#08Fh	1B83100	D0=(5)	#00138	
%1				❺
	14A	A=DAT0	B	
	80843	ABIT=0	3	❻
	986	?A>C	P	
	5F	GOYES	%1	
	171	D1=D1+	#2	
	1590	DAT1=A	1	
	01	RTN		
$SCORE				❼
#0ACh	177	D1=D1+	#8	❽
	05	SETDEC		
	1537	A=DAT1	W	
	A7A	A=A+C	W	
	1517	DAT1=A	W	
	04	SETHEX		
	01	RTN		
$AFF				❾
#0C0h	1537	A=DAT1	W	❿
	D7	D=C	A	

❶ Transfert entre la table des sprites et l'écran.
❷ Passage à la ligne du dessous, donnée suivante.
❸ Boucle sur ligne. Restauration de D1 sur figure courante.
❹ Procédure $NEXT.
❺ On pointe sur l'horloge (on capture ainsi un nombre aléatoire).
❻ Point de branchement %1. Accroissement du hasard.
❼ Procédure $SCORE.
❽ On pointe sur le score que l'on modifie.
❾ Procédure $AFF.
❿ Lecture du chiffre à afficher.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	CE	C=C-1	A	
%1				❶
	814	ASRC		
	A0E	C=C-1	P	❷
	59F	GONC	%1	
	AF6	C=A	W	
%2				❸
	06	RSTK=C		
	D1	B=0	A	
	A85	B=C	P	❹
	111	A=R1		
	C5	B=B+B	A	
	C0	A=A+B	A	❺
	81D	BSRB		
	F1	BSL	A	
	C0	A=A+B	A	❻
	131	D1=A		
	17E	D1=D1+	#F	
	174	D1=D1+	#5	❼
	308	C=(1)	#8	
%3				❽
	06	RSTK=C		
	14B	A=DAT1	B	
	DB	C=D	A	
	808B0	?CBIT=1	0	
	B0	GOYES	%4	❾
	D2	C=0	A	
	1543	DAT0=C	X	❿
	560	GONC	%5	

❶ Point de branchement %1.
❷ C contient les quartets à afficher.
❸ Point de branchement %2.
❹ Numéro du sprite chiffre à afficher. Début de la table des sprites.
❺ × #2h.
❻ × #10h.
❼ D1 désigne le sprite chiffre (dans la table des sprites chiffres).
❽ Point de branchement %3. Lecture des données sprite chiffre.
❾ Test de la parité du compteur.
❿ Si le compteur est pair, on vide trois quartets.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%4				❶
	C4	A=A+A	A	
	C4	A=A+A	A	
%5				❷
	14E	C=DAT0	B	
	0E6A	C=C!A	B	❸
	14C	DAT0=C	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	❹
	171	D1=D1+	#2	
	07	C=RSTK		
	A0E	C=C-1	P	
	59C	GONC	%3	❺
	3413100	C=(5)	#00131	
	132	AD0ex		
	EA	A=A-C	A	
	130	D0=A		❻
	DB	C=D	A	
	808A0	?CBIT=0	0	
	50	GOYES	%6	❼
	160	D0=D0+	#1	
%6				❽
	07	C=RSTK		
	812	CSLC		❾
	A0F	D=D-1	P	
	528	GONC	%2	❿
	01	RTN		

❶ Point de branchement %4. Décalage du chiffre (si compteur impair).
❷ Point de branchement %5.
❸ Calcul d'un OU logique avec l'affichage.
❹ On passe à la ligne du dessous. Donnée suivante.
❺ Boucle sur 9 lignes.
❻ On passe au chiffre suivant sur l'écran.
❼ Rectification du compteur de quartets (selon la parité).
❽ Point de branchement %6.
❾ Quartet suivant.
❿ Boucle sur les quartets.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$BEEP				❶
#156h	854	ST=1	4	
	DA	A=C	A	❷
	DD	BCex	A	
	06	RSTK=C		❸
	28	P=	8	
%1				❹
	80F2	CPex	2	
	866	?ST=0	6	
	50	GOYES	%2	
	801	OUT=C		❺
%2				❻
	06	RSTK=C		
	D6	C=A	A	
%3				❼
	CE	C=C-1	A	
	5DF	GONC	%3	
	07	C=RSTK		
	CD	B=B-1	A	
	56E	GONC	%1	
	20	P=	0	
	07	C=RSTK		
	D5	B=C	A	❽
$TEMPO				❾
#183h	D2	C=0	A	
%1				❿
	B36	C=C+1	X	
	520	NOP3		
	59F	GONC	%1	
	01	RTN		

❶ Procédure $BEEP. Flag triplet.
❷ Fréquence copiée dans A.
❸ Sauvegarde de B. Durée placée dans B.
❹ Point de branchement %1. Aucun son si le flag BEEP RPL est désactivé.
❺ Emission d'un bip (volume normal).
❻ Point de branchement %2.
❼ Point de branchement %3. Fréquence puis durée du bip.
❽ Restauration de B (colonne courante).
❾ Procédure $TEMPO (temporisation).
❿ Point de branchement %1.

Affectations des registres

Registres	Affectations
P	Toujours zéro
A, C	Compteur (C) avec D0 et D1
B	Ligne (0 à 4) logique de l'écran de jeu.
D	Colonne (0 à 15) logique de l'écran de jeu.
D0	Ecran ou RAM des entrées/sorties
D1	Données.

Affectations des drapeaux

Drapeaux	Affectations
ST0	En cas de pression sur [÷], interdit le test des touches
ST1	Interdit une nouvelle pression sur [←]
ST2	Interdit une nouvelle pression sur [÷]
ST3	Interdit une nouvelle pression sur [α]
ST4	Détection des triplets
ST5	Détection des cascades
ST6	Flag BEEP RPL
ST15	Inhibition des interruptions

zones utilisées en RAM

Adresses de départ	Affectations de la RAM RS232C
R0	Table des sprites (figures cubiques)
R1	Table des sprites (chiffres)
#70200h	Vitesse (affichage) variant entre 0 et 200
#70205h	Niveau +3 (4, 5, 6 ou 7) d'où nombre de figures
#70206h	Figure courante (en cours de chute)
#70207h	Figure suivante
#70208h	Vitesse (compteur)
#7020Dh	Score
#7021Dh	Tableau créé par les cascades (taille 15×5)

Les zones utilisées en RAM sont celles normalement réservées à l'interface série RS 232. Il s'agit des adresses comprises entre #701FCh et #703FBh.

3
Exécuter le programme

Si ce n'est pas encore fait, créez un répertoire DXALK et placez-y les objets suivants :

- 'XALK', le lanceur en RPL,
- 'CODE', le programme proprement dit,
- 'SPR', l'objet graphique codant les sprites,
- 'CHA', l'objet graphique codant les chiffres.

La chaîne des codes hexadécimaux 'XS2' et le programme 'ASS' ne sont absolument pas nécessaires à l'exécution du programme.

Pour lancer le jeu, il suffit d'évaluer XALK (le lanceur RPL). Tous les objets indispensables que nous avons cités devront bien entendu se trouver dans le répertoire DXALK.

4
Comment jouer ?

Des figures arrivent de la gauche de l'écran, à vous de les empiler correctement sur 5 colonnes. Vous aurez perdu lorsqu'une colonne aura atteint toute la largeur de l'écran. Le lanceur RPL vous demande le niveau du jeu, de 1 à 4. Au niveau 4 vous devrez faire face à 8 figures différentes pour 5 colonnes !

Sachez que trois figures identiques disparaissent lorsqu'elles sont alignées horizontalement ou verticalement (les diagonales ne sont pas reconnues).

Il existe plusieurs coups spéciaux :

- une cascade s'obtient en éliminant plusieurs triplets à l'aide d'une unique figure. En fait, une figure provoque l'élimination d'un triplet. Cette élimination provoque la chute vers la droite des figures se trouvant à gauche du triplet éliminé. C'est ainsi qu'une seule figure permet d'éliminer plusieurs triplets,
- une combinaison est l'élimination de plusieurs triplets à l'aide d'une unique figure sans avoir pour autant recours à la cascade. Autrement dit, les figures ne tombent pas. Une combinaison peut par exemple s'obtenir en élimant un triplet horizontal et un triplet vertical à l'aide d'une unique figure,
- cinq figures peuvent disparaître simultanément, à vous de savoir constituer une colonne ou une ligne de cinq figures identiques. Notez qu'une ligne ou une colonne de 4 figures ne permet l'élimination que de 3 figures... C'est 3 ou 5 !

A l'écran apparaissent la prochaine figure (en haut à gauche), le score (en bas à gauche) et la vitesse de chute (en bas à droite).

Au niveau 1, chaque figure posée rapporte 1 point, chaque triplet éliminé rapporte 5 points. Au niveau 4, une figure posée et un triplet éliminé rapportent respectivement 4 et 20 points. Les touches à utiliser pendant le jeu sont les suivantes :

- [α] provoque un déplacement vers le haut,
- [↰] provoque un déplacement vers le bas,
- [÷] lâche le carré dans la colonne,
- [5] interrompt temporairement la partie,
- [2] permet de reprendre la partie,
- [8] sert à quitter le jeu.

Appuyer sur [8] est le seul moyen normal de quitter le jeu. Le son est fonction du flag beep RPL (qui n'est pas modifiable pendant le jeu).

5

Retrouver les fichiers sur la disquette

Les répertoires dans lesquels se trouvent les fichiers associés à ce jeu sont : DUNOD_1\S_SX\OB_P_OBS et DUNOD_1\S_SX.

Les chemins d'accès aux différents fichiers relatifs aux variables associées à ce jeu sont :

- 'XS2' : DUNOD_1\S_SX\OB_P_OBS\XALK_S\XS2,
- 'CODE' : DUNOD_1\S_SX\OB_P_OBS\XALK_S\CODE,
- 'XALK' : DUNOD_1\S_SX\OB_P_OBS\XALK_S\XALK,
- 'CHA' : DUNOD_1\S_SX\OB_P_OBS\XALK_S\CHA,
- 'SPR' : DUNOD_1\S_SX\OB_P_OBS\XALK_S\SPR,
- 'ASS' : DUNOD_1\S_SX\ASS.

31

Seth pour HP-48 S et SX

1

Le jeu

1.1

Présentation

SETH est un jeu de course automobile qui vous rappellera les classiques du genre sur Sinclair Spectrum et Oric Atmos. Une quinzaine de tours d'une piste longue de 6 km auxquels s'ajoute un final de 10 km... Bref 100 km d'une course endiablée... Les 10 voitures à votre disposition ne seront pas de trop ! Des concurrents cherchant la collision, des flaques d'huile et une piste sinueuse sont au programme des réjouissances !

1.2

Liste des objets

En plus du code exécutable, SETH exige un lanceur en RPL et 5 objets graphiques (GROB).

Neuf objets sont associés à la mise en œuvre de la version pour HP-48 S et SX de SETH :

- 'SETH48S' est la chaîne de codes hexadécimaux qui devra être traitée avec le programme ASS pour générer le code exécutable 'CODE',
- 'CODE' est le code exécutable générer avec ASS à partir de 'SETH48S',
- 'ASS' est le programme permettant de générer 'CODE' à partir de 'SETH48S',
- 'RPL' est le programme lanceur écrit en RPL,
- 'R0' est l'objet graphique codant le score,
- 'R1' est l'objet graphique codant les chiffres,
- 'R2' est l'objet graphique codant les sprites objets,
- 'R3' est l'objet graphique codant le sprite « ON »,
- 'R4' est l'objet graphique codant la coupe affichée en cas de victoire.

'SETH48S' et 'ASS' ne sont absolument pas nécessaires pour exécuter le programme. Par contre, 'RPL' (le programme lanceur en RPL), 'CODE' (le code exécutable), 'R0', 'R1', 'R2', 'R3' et 'R4' (les objets graphiques) doivent absolument se trouver dans le répertoire DSETH (à créer).

2
Les objets

2.1
Le lanceur RPL

Le lanceur est un programme écrit en RPL. Il est à enregistrer dans une variable portant le nom 'RPL'. Il doit être saisi entre « et » qui sont les délimiteurs d'un objet de type programme RPL.

```
« CLEAR #90h #0h
PDIM PICT { #18h
#18h } { #0h #0h
} PVIEW
```

```
"SETH 48 S/SX" 3
→GROB REPL R0 R1 R2
R3 R4 -1 WAIT
  IF 16.6 ==
  THEN -7
  ELSE -9
  END ERASE { #0h
#0h } PVIEW CODE
»
```

Placez dans la pile une copie de l'objet 'RPL', activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #E7EFh,
- au niveau 1, une longueur égale à 236,5 octets.

2.2
Les objets graphiques

Attention ! Les objets graphiques (**GROB...**) sont à saisir sans délimiteurs. Seuls leurs trois premiers espaces sont à saisir, les autres espaces sont à ignorer et ne sont là que pour améliorer la lisibilité. Les sauts de ligne doivent aussi être ignorés.

■ Les scores

Cet objet est à enregistrer dans une variable nommée 'R0'.

```
GROB 1 112 77737 07777 77005 50730 73005 27707
           37115 51255 55550 05725 20520 05211
           25251 15512 55555 50057 25205 20052
           11252 57153 70555 55500 35052 75200
           52370 52541 55125 55555 00552 52052
           00521 42525 41551 25555 55005 52520
           52002 21425 25777 57077 77770 05507
           70770 02277 0777
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #E606h,
- au niveau 1, une longueur égale à 122 octets.

■ Les chiffres

Cet objet est à enregistrer dans une variable nommée 'R1'.

```
GROB 1 35 75555 57322 22277 44711 77446 44755
          57444 71174 47711 75577 44222 27557
          55775 57447
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #433Eh,
- au niveau 1, une longueur égale à 45 octets.

■ Les objets

Cet objet est à enregistrer dans une variable nommée 'R2'.

```
GROB 1 104 00000 00000 00000 0FFBA 5DBA5 DBA5D
           FFF00 0F000 F000F 000F0 F0F0F 0F0F0
           F0F0F 0FE64 BFBF6 4FEF0 707E2 4FFFF
           247E7 05076 21EDB 76236 60E7F E9BFF
           F93FF DE7DB E7BDF FE742 E7998 D9FF7
           E3C7C FEE6C 64FC9 9BA9A B490F 0C324
           DBDBD BDB24 C3814 26699 99664 281
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #1E9Eh,
- au niveau 1, une longueur égale à 114 octets.

■ Représentation de « ON »

Cet objet est à enregistrer dans une variable nommée 'R3'. Il correspond au dessin de la touche [ON] en bas de l'écran au début du jeu.

```
GROB 1 26 FFF71 00493 15543 55435 54555 45554
          95549 55415 93151 004FF F7
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #DB6Bh,
- au niveau 1, une longueur égale à 36 octets.

■ **La coupe du vainqueur**

```
GROB 1 336 FFFFF FFFFF F7100 00000 00041 0008F
           F0000 41000 70070 00410 0E000 83004
           10810 000C0 04106 00000 03041 01000
           00040 41800 00000 80414 00000 00014
           14000 00000 14120 00000 00241 20000
           00002 41100 00000 04411 00000 00044
           11000 00000 44110 00000 00441 30000
           00006 41300 00000 06417 00000 00074
           16000 F7000 341A0 0C081 08241 21020
           02042 41460 10040 31414 81100 4C014
           180E2 00A38 04101 07007 04041 0108F
           F0040 41020 00000 20410 40000 00104
           10810 000C0 04100 20000 20041 00C00
           08100 41000 70070 00410 00CFF 10004
           10004 00100 04100 08080 00041 00080
           80000 41000 80800 00410 00014 00004
           10000 14000 04100 081C0 00041 000E1
           C3000 41000 F1C70 00410 00F1C 70004
           1008F 3EF00 04100 8FFFF 00041 008FF
           FF000 41000 00000 00499 E210D 5D144
           9A261 05550 4499E E1055 D1449 A2A14
           55500 499E2 1CDDD 14410 00000 00004
           FFFFF FFFFF F7
```

Cet objet est à enregistrer dans une variable nommée 'R4'. Placez dans la pile une copie de l'objet, activez BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #6D39h,
- au niveau 1, une longueur égale à 346 octets.

Si vous savez par avance que vous n'aurez pas droit à la superbe coupe du vainqueur, contentez-vous de stocker dans 'R4' l'objet suivant :

```
GROB 1 1
```

2.3
Le code hexadécimal

Le code de la version pour HP-48 G comprend les objets graphiques. Ce n'est pas le cas du code destiné au HP-48 S/SX.

a

La chaîne à saisir

Cet objet est à enregistrer dans une variable nommée 'SETH48S'. Saisissez la chaîne de codes hexadécimaux présentée ci-dessous entre " et ". Ne saisissez ni les espaces ni les sauts de ligne.

```
"CCD20 09D00 14381 48143 24A29 32C01
42164 808C8 4F808 F0881 08103 14096
65085 920D2 3141D 52417 481B5 143C0
C9134 80C01 5C010 00D51 E2013 11BC0
10015 C019E 215C0 1B2D6 0714E 808B3
50852 1AE05 01421 3030D 15371 50716
F17F8 08B05 0163A 0E54E 16F16 F1631
36108 AF235 10010 11E00 20869 62171
14332 99993 2E017 B14B3 03982 C0D28
1B315 57840 8411E F0203 20671 553AF
232A0 DAFA3 21101 73151 7179C E56F3
32222 1D991 4534F B8008 1B4C2 06135
31011 4D853 1E312 07387 1D311 4B302
90211 34008 00CE5 DF40E 07135 31341
4D74B 67D46 86223 D2303 22D73 04D53
08801 D2801 D4CC5 DF818 F9F57 ECF5D
D2034 78300 110CA 13014 A1E21 20159
08EE8 901B4 31008 43146 C6550 85330
7D511 0130D 23168 15278 1C158 E16EC
E5FE7 C1632 0408F CEE10 808A0 50852
808A2 50842 32800 8FCEE 10808 A280D
281B3 808A0 921B0 0100D 215C0 32004
8FCEE 10808 AFFE3 0815C 09091 31DD0
D015B 08618 0818F 05340 0800F 0C4C4
E2CE5 DFCD6 54F17 53111 17915 B91C9
15991 79A6E 5BE1D 2114B 7C271 B8310
01DDB AF215 571C9 14B14 E90AF 0E480
8A060 CCCC1 79149 1C714 BE480 8A160
CCCC1 79149 1C114 FEA30 49826 A3059
86E91 73137 135D7 D1A88 1DD0D 215F0
81B46 34000 00900 07000 9A00B 0009C
F0A00 09B00 7C009 A00BF 009C0 07AB0
9CF07 9B09A BFE6C 6C6CA 13114 70614
A9047 1DB80 86440 C9135 30615 D014A
96471 DB808 6440C 91353 0815D 0CDCD
E71ED B2014 F1711 4BEA3 05982 A214A
```

```
80872 02808 7391D 215E0 8B59F CB135
30515 D0E5E 5CF81 507D5 30314 A8084
28084 3A0C4 70F55 7F14A 80873 74310
A9E6E 31564 9C59F D2812 C3816 DB135
D915D 030F9 05813 0715D 01709 45501
C115D 01E01 2014B A6C14 95053 18614
D1D1C 35222 22215 D51D5 014B3 16196
29205 C4D6C 4C204 1C414 B9662 11D1C
35333 33315 D5110 D2312 2CA32 0E7C2
D7D81 30306 06D27 81307 A0E52 F3402
1001D E014B A0C49 0C1E3 55FD4 130D0
15B01 DDB14 FEA34 1C207 A62D5 14B17
114FB 62171 133A6 ADC13 11331 31311
C9E2D 09E08 014F5 50301 7A921 70136
1348B 33D84 43280 08FCE E1080 8A501
84385 48605 08533 24008 FCEE1 0808A
55187 49184 38548 70508 53864 60759
21E01 2014B 31069 E6711 7614B 30290
2D0E6 90260 6AE08 41D81 D0014 B05B6
40414 95606 FC630 3901C 2D231 F5D5D
72880 F2862 50801 D4CC5 DFCF5 AE206
B9017 414B0 5E404 14984 01771 4BE41
59030 4902A 030E9 06341 70D01 5B0D6
E4159 0110F 2F2CA F6C6C A1307 2A111
0C413 6EA34 4E600 CA130 78818 6262D
230CD 5311C D7288 0F280 1D4CC 5DFCF
5FE20 1D701 5B5AF 23070 5A7A0 41595
34511 00110 EA130 74301 6C1E0 02030
17420 1641E 50203 01741 016B1 E2020
30274 006D3 AD715 F5AF5 111D2 A89EA
F281E CA139 30614 B1580 16F16 F1631
70A0E 59E13 2D231 DFEA1 30BF5 A0F5B
B0111 0345E 400C2 06134 11313 130C1
43140 16F16 F1631 73A0E 5AE32 0048F
CEE10 808AF FE071 34D03 0C140 16F16
F163A 0E50F 01340 63001 10CA1 30304
DA137 06D2A 86F21 12CA1 31317 014B1
4816F 16F16 3171C E5BE0 71350 1D231
11D73 2E11D 5D273 BF136 E9134 CF5FE
01345 62001 10863 70325 A4CA1 3014A
77A23 20631 10CA1 30D27 17F1D 31313
214BE 48636 0CCCC 14917 1808B 05017
5A6E5 ED1DE 01103 40210 0D587 3B032
0D7CA F914F 06DDC AA0D5 AF131 EA130
```

```
07D73 1F8D5 3142E 1A0F5 AF153 71507
873A1 18F1C FA6D5 9E17F 13713 45E01
6F17F A6D51 DD21F E0207 15F0D 71763
11106 DB863 70306 EB14B 1C1B6 AA644
8214B B6280 8B7B1 133D8 A6A13 11731
4FD41 31550 30179 6E17B 13234 E1100
EA130 07CE5 F9345 83001 10CA1 3014A
30290 69387 08085 05508 40862 32D23
1F6D5 D2288 0F280 1D4CC 5DFCD 5FE20
41630 49068 08515 BC307 906F4 1D201
4705E 60415 53D23 08063 0BD5D 72880
F2862 50801 D4CC5 DFCF5 AE80F 2CE5D
F07CE 5FC6D 8D30C 906D5 1D701 5B5AF
23104 05A7A 04159 51771 4BB04 15904
38862 7C310 FD531 51D73 20088 01630
0D280 1D4CC 5DFCF 55E4B 930F9 06397
98D75 8D862 00D1D 2B253 0FD72 880F2
801D4 CC5DF 81861 F5BEC D80F2 801D4
CC5DF 81869 F5BEC F5DC2 00180 87000
80861 00302 90200 11034 06300 CA130
30671 BC31D 7D511 01301 527BF C1507
16F31 0AA6E 5CFA6 D54ED 0D1D2 D3CC2
8B654 51B67 56FD7 D6DFC 7C75A 0D8A6
C4218 0F286 25080 162DF D5D28 01CD5
DF31F 1D531 CF110 13015 A7A74 15871
67A6E 5EEA6 E5CFA 6D58D 1E202 01430
5A3C0 41513 4D111 03410 100EA 130E6
7B1B8 C584F 34B40 00862 7481B 4CA13
13200 3A3E5 CF143 908B2 17214 F171D
7D228 80F28 01D8A 3D5CF A6F5D E2046
C721B D281B 31010 F1010 A1010 51010
F7D00 A5E00 55E00 A1010 51010 AE010
5101F F07D0 067D0 037D0 03F11 027D0
03DA0 037D0 09011 01301 14131 3183D
53200 9A3E5 CF15B B158B 16F16 F1631
7BA6D 5DD34 38000 611F
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #B7FEh,
- au niveau 1, une longueur égale à 3474 octets.

Nous allons maintenant générer un code exécutable à partir de la chaîne de codes hexadécimaux sauvegardée dans 'SETH48S'.

Pour générer 'CODE' (le code exécutable) à partir de 'SETH48S', procédez de la façon suivante :

- placez la chaîne de codes hexadécimaux (variable 'SETH48S') au niveau 1 de la pile "CCD2009D001438148... s'affiche,
- exécuter le programme ASS,
- au niveau 1 de la pile doit apparaître Code,
- si ce n'est pas encore fait, placez-vous dans le répertoire DSETH (à créer s'il n'existe pas déjà),
- saisissez le nom 'CODE',
- appuyez sur la touche [STO].

Placez dans la pile une copie de l'objet code exécutable (variable 'CODE'), activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #2C2Bh,
- au niveau 1, une longueur égale à 1738,5 octets.

Codes commentés

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$$				❶
	CCD20			
#000h	09D00			
	143	A=DAT1	A	❷
	814	ASRC		
	814	ASRC		
	324A2	C=(3)	#2A4	
	932	?A=C	X	❸
	C0	GOYES	%1	
	142	A=DAT0	A	❹
	164	D0=D0+	#5	
	808C	PC=(A)		
%1				❺
#022h	84F	ST=0	15	

❶ Procédure $$ (initialisation du jeu).
❷ Lecture du premier étage de la pile RPL.
❸ Test : est-ce -9 ou -7 ?
❹ Si le test est négatif, on retourne vers le RPL.
❺ Point de branchement %1. Inhibition des interruptions.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	808F	INTOFF		
	08	CLRST		❶
	810	ASLC		
	810	ASLC		
	3140	C=(2)	#04	❷
	966	?A≠C	B	
	50	GOYES	%2	
	859	ST=1	9	❸
%2				❹
#03Dh	20	P=	0	
	D2	C=0	A	❺
	3141	C=(2)	#14	
	D5	B=C	A	
	24	P=	4	
%3				❻
#049h	174	D1=D1+	#5	
	81B5	C=PC		
	143	A=DAT1	A	❼
	C0	A=A+B	A	
	C9	C=C+B	A	
	134	D0=C		
	80C0	C=P	0	
	15C0	DAT0=C	1	❽
	100	R0=A		
	0D	P=P-1		
	51E	GONC	%3	❾
	20	P=	0	
	131	D1=A		❿
	1BC0100	D0=(5)	#0010C	
	15C0	DAT0=C	1	

❶ ST2=0.
❷ Test : a-t-on -7 ?
❸ Si oui, ST9=1 en attendant.
❹ Point de branchement %2.
❺ Chargement de la constante B. Décalage indicatif/contenu (GROB).
❻ Point de branchement %3.
❼ A pointe sur le contenu du GROB (étage n de la pile RPL).
❽ Change R0=A en Rn=A (avec 0≤n≤4).
❾ De R4 à R0.
❿ Sauvegarde R0. Suppression du témoin « sablier ».

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	19E2	D0=(2)	#2E	❶
	15C0	DAT0=C	1	
	1B2D607	D0=(5)	0706D2	❷
	14E	C=DAT0	B	
	808B3	?CBIT=1	3	❸
	50	GOYES	%4	
	852	ST=1	2	
%4				❹
#096h	1AE050	D0=(4)	#050E	
	142	A=DAT0	A	
	130	D0=A		
	30D	C=(1)	#D	
%5				❺
#0A5h	1537	A=DAT1	W	
	1507	DAT0=A	W	
	16F	D0=D0+	#10	
	17F	D1=D1+	#10	❻
	808B0	?CBIT=1	0	
	50	GOYES	%6	
	163	D0=D0+	#4	
%6				❼
#0BDh	A0E	C=C-1	P	
	54E	GONC	%5	❽
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	163	D0=D0+	#4	
	136	CD0ex		❾
	108	R0=C		
	AF2	C=0	W	
	35100101	C=(6)	#101001	❿

❶ Toujours moins d'interruptions !
❷ Lecture du flag BEEP RPL.
❸ Si le son est autorisé, alors, ST2=1.
❹ Point de branchement %4. D0 désigne la bitmap écran.
❺ Point de branchement %5. Affiche 16 quartets du sprite score.
❻ Donnée suivante.
❼ Point de branchement %6.
❽ Boucle les demi-lignes. Saute une ligne.
❾ R0 pointe sur le début de la neuvième ligne.
❿ Champ de vision initial (KM=01).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1E0020	D1=(4)	#0200	❶
	869	?ST=0	9	
	62	GOYES	%7	❷
	171	D1=D1+	#2	
	143	A=DAT1	A	
	32999	C=(3)	#999	❸
	932	?A=C	X	
	E0	GOYES	%7	
	17B	D1=D1+	#C	
	14B	A=DAT1	B	
	303	C=(1)	#3	❹
	982	?A<C	P	
	C0	GOYES	$0	
%7				❺
#106h	D2	C=0	A	
	81B3	PC=C		
%8				❻
#10Ch	1557	DAT1=C	W	
$0				❼
#110h	840	ST=0	0	❽
	841	ST=0	1	
	1EF020	D1=(4)	#020F	
	32067	C=(3)	#760	❾
	1553	DAT1=C	X	
	AF2	C=0	W	
	32A0D	C=(3)	#D0A	❿
	AFA	A=C	W	
	32110	C=(3)	#011	
	173	D1=D1+	#4	

❶ Initialisation : VIES = 005, TOUR = 01, SCORE = 000000...
❷ Test : veut-on reprendre une partie en cours ?
❸ Etait-on « mort » ?
❹ Test : le champ de vision a-t-il été détérioré par une transmission IR ?
❺ Point de branchement %7.
❻ Point de branchement %8.
❼ Procédure $0 (gestion du départ).
❽ Désactivation de l'inverseur et de l'accélérateur.
❾ Compteur kilomètres = #76h. Compteur essence = #0h.
❿ Première case : coordonnées 2. Dernière case : coordonnées → 5.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%1				❶
#138h	1517	DAT1=A	W	
	179	D1=D1+	#A	
	CE	C=C-1	A	❷
	56F	GONC	%1	
	332222	C=(4)	#2222	❸
	1D99	D1=(2)	#99	
	145	DAT1=C	A	
#151h	34FB800	C=(4)	#008BF	
	81B4	A=PC		❹
#15Ch	C2	C=C+A	A	
	06	RSTK=C		
	135	D1=C		❺
	3101	C=(2)	#10	
	14D	DAT1=C	B	
	853	ST=1	3	❻
	1E3120	D1=(4)	#0213	❼
%2				❽
#173h	7387	GOSUB	$APPUI2	
#177h	1D31	D1=(2)	#13	
	14B	A=DAT1	B	
	302	C=(1)	#2	❾
	902	?A=C	P	
	11	GOYES	%4	
	3400800	C=(5)	#00800	
%3				❿
#18Dh	CE	C=C-1	A	
	5DF	GONC	%3	
	40E	GOC	%2	

❶ Point de branchement %1.
❷ Passage à la colonne suivante.
❸ Création de la ligne de départ.
❹ On pointe sur la procédure $APPUI.
❺ Court-circuit, affichage de la voiture...
❻ Descente...
❼ On pointe sur les données.
❽ Point de branchement %2.
❾ La route est-elle centrée.
❿ Point de branchement %3. Temporisation.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%4				❶
#195h	07	C=RSTK		
	135	D1=C		
	3134	C=(2)	#43	
	14D	DAT1=C	B	
	74B6	GOSUB	$VOITURE	❷
#1A5h	7D46	GOSUB	$ON	❸
#1A9h	862	?ST=0	2	
	23	GOYES	$1	❹
	D2	C=0	A	
	303	C=(1)	#3	
	22	P=	2	
	D7	D=C	A	
%7				❺
#1B7h	304	C=(1)	#4	
	D5	B=C	A	
%8				❻
#1BCh	308	C=(1)	#8	
	801	OUT=C		
	D2	C=0	A	
	801	OUT=C		
	D4	A=B	A	❼
%9				❽
#1C9h	CC	A=A-1	A	
	5DF	GONC	%9	
	818F9F	B=B-#10	A	❾
	57E	GONC	%8	
	CF	D=D-1	A	❿
	5DD	GONC	%7	

❶ Point de branchement %4. Suppression du court-circuit...
❷ Affichage de la voiture.
❸ Attente d'une pression sur la touche [ON].
❹ Le son est-il autorisé.
❺ Point de branchement %7. Fréquence copiée dans B.
❻ Point de branchement %8. Bip avec volume faible.
❼ Temporisation fréquence.
❽ Point de branchement %9.
❾ Boucle durée (plus aigu...).
❿ Son produit 5 fois.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	20	P=	0	
$1				❶
#1DEh	3478300	C=(5)	#00387	
	110	A=R0		
	CA	A=A+C	A	
	130	D0=A		❷
	14A	A=DAT0	B	
	1E2120	D1=(4)	#0212	
	1590	DAT1=A	1	❸
	8EE890	GOSUBL	#SCRASH	
#200h	1B43100	D0=(5)	#00134	
	843	ST=0	3	
	146	C=DAT0	A	❹
	C6	C=C+C	A	
	550	GONC	%1	
	853	ST=1	3	
#215h	307	C=(1)	#7	❺
	D5	B=C	A	
%1				❻
#21Ah	110	A=R0		
	130	D0=A		
	D2	C=0	A	
	3168	C=(2)	#86	❼
%2				❽
#226h	1527	A=DAT0	W	
	81C	ASRB		❾
	158E	DAT0=A	15	
	16E	D0=D0+	#F	
	CE	C=C-1	A	❿

❶ Procédure $1 (scrolling).
❷ Test : collision sur la droite ?
❸ Mémorisation provisoire.
❹ Calcul du flag ST3 (au hasard) selon #00138 (horloge).
❺ Huit décalages.
❻ Point de branchement %1. On pointe en bas à droite.
❼ Compteur de boucle.
❽ Point de branchement %2.
❾ Scrolling de 15 quartets (d'un bit).
❿ Quartets suivants.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	5FE	GONC	%2	
	7C16	GOSUB	$VOITURE	❶
#23Dh	32040	C=(3)	#040	
	8FCEE10	GOSBVL	#01EEC	❷
	808A0	?CBIT=0	0	
	50	GOYES	%3	
	852	ST=1	2	❸
%3				❹
#253h	808A2	?CBIT=0	2	
	50	GOYES	%4	
	842	ST=0	2	
%4				❺
#25Dh	32800	C=(3)	#800	
	8FCEE10	GOSBVL	#01EEC	
	808A2	?CBIT=0	2	❻
	80	GOYES	%5	
	D2	C=0	A	
	81B3	PC=C		
%5				❼
#276h	808A0	?CBIT=0	0	
	92	GOYES	%7	❽
	1B00100	D0=(5)	#00100	
	D2	C=0	A	
	15C0	DAT0=C	1	
%6				❾
#28Ah	32004	C=(3)	#400	
	8FCEE10	GOSBVL	#01EEC	
	808AF	?CBIT=0	15	❿
	FE	GOYES	%6	

❶ Affichage de la voiture.
❷ Test : pression sur [→] ?
❸ Flag son activé.
❹ Point de branchement %3. Test : pression sur [←] ?
❺ Point de branchement %4.
❻ Test : pression sur [8] ? Reset système.
❼ Point de branchement %5.
❽ Test : pression sur [÷] ? Extinction de l'écran.
❾ Point de branchement %6.
❿ Attente d'une pression sur [ON].

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	308	C=(1)	#8	❶
	15C0	DAT0=C	1	
%7				❷
#2A4h	909	?B=0	P	
	13	GOYES	%10	❸
	1DD0	D1=(2)	#0D	
	D0	A=0	A	
	15B0	A=DAT1	1	❹
	861	?ST=0	1	
	80	GOYES	%8	
	818F05	A=A+#6	A	
%8				❺
#2BEh	3400800	C=(4)	#00800	
	F0	ASL	A	❻
	C4	A=A+A	A	
	C4	A=A+A	A	
	E2	C=C-A	A	
%9				❼
#2CDh	CE	C=C-1	A	
	5DF	GONC	%9	
	CD	B=B-1	A	❽
	654F	GOTO	%1	
%10				❾
#2D8h	175	D1=D1+	#6	
	3111	C=(2)	#11	
%11				❿
#2DFh	179	D1=D1+	#4	
	15B9	A=DAT1	10	
	1C9	D1=D1-	#A	

❶ Rallumage de l'écran.
❷ Point de branchement %7.
❸ Pas de temporisation pour le huitième décalage.
❹ Lecture du numéro du tour en cours. Test : accélérateur ?
❺ Point de branchement %8.
❻ Calcul du compteur en fonction du numéro du tour ?
❼ Point de branchement %9.
❽ Décalage suivant.
❾ Point de branchement %10. On pointe sur le début du tableau.
❿ Point de branchement %11. Lecture de l'une des 18 colonnes.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1599	DAT1=A	10	❶
	179	D1=D1+	#A	
	A6E	C=C-1	B	❷
	5BE	GONC	%11	
	1D21	D1=(2)	#12	
	14B	A=DAT1	B	❸
	7C27	GOSUB	$COLLISION	
$2				❹
#301h	1B83100	D0=(5)	#00138	❺
	1DDB	D1=(2)	#BD	
	AF2	C=0	W	
	1557	DAT1=C	W	
	1C9	D1=D1-	#A	
	14B	A=DAT1	B	❻
	14E	C=DAT0	B	
	90A	?C=0	P	
	F0	GOYES	%1	❼
	E4	A=A+1	A	
	808A0	?CBIT=0	0	
	60	GOYES	%1	
	CC	A=A-1	A	
	CC	A=A-1	A	
%1				❽
#32Eh	179	D1=D1+	#A	
	149	DAT1=A	B	
	1C7	D1=D1-	#8	
	14B	A=DAT1	B	❾
	E4	A=A+1	A	
	808A1	?CBIT=0	1	❿

❶ Transfert d'une colonne (du tableau de jeu).
❷ Boucle sur 18 colonnes.
❸ Test : collision sur la position de la voiture ?
❹ Procédure $2 (création de la 18ème colonne du tableau de jeu).
❺ D0 pointe sur l'horloge. On vide la colonne.
❻ Lecture de l'ancienne position relative à R0.
❼ Tendance à monter. Probabilité = 1/2.
❽ Point de branchement %1. On stocke la nouvelle 1ère position.
❾ Lecture de l'ancienne 2ème position.
❿ Probabilité = 1/2.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	60	GOYES	%2	
	CC	A=A-1	A	
	CC	A=A-1	A	
%2				❶
#347h	179	D1=D1+	#A	
	149	DAT1=A	B	
#34Dh	1C1	D1=D1-	#2	
	14F	C=DAT1	B	
	EA	A=A-C	A	❷
	304	C=(1)	#4	
	982	?A<C	P	
	6A	GOYES	$2	❸
	305	C=(1)	#5	
	986	?A>C	P	❹
	E9	GOYES	$2	
#365h	173	D1=D1+	#4	❺
	137	CD1ex		
	135	D1=C		
	D7	D=C	A	❻
	D1	B=0	A	❼
	A88	B=A	P	
#375h	1DD0	D1=(2)	#0D	❽
	D2	C=0	A	❾
	15F0	C=DAT1	1	
	81B4	A=PC		❿
	6340	GONC	%3	

❶ Point de branchement %2. On stocke la nouvelle deuxième position.
❷ Lecture de la largeur.
❸ Si la largeur est hors norme...
❹ ...on recalcule les positions.
❺ On pointe sur la première case du contenu.
❻ On place le pointeur dans D.
❼ On place la largeur dans B (nombre de cases moins 1).
❽ Début de la lecture du tableau par PC.
❾ Lecture du numéro du tour.
❿ Calcule PC.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#387h	00009	*Données du tableau*		❶
	00070	*Données du tableau*		
	009A0	*Données du tableau*		
	0B000	*Données du tableau*		
	9CF0A	*Données du tableau*		
	0009B	*Données du tableau*		
	007C0	*Données du tableau*		
	09A00	*Données du tableau*		
	BF009	*Données du tableau*		
	C007A	*Données du tableau*		
	B09CF	*Données du tableau*		
	079B0	*Données du tableau*		
	9ABF	*Données du tableau*		
%3				❷
#3C7h	E6	C=C+1	A	
	C6	C=C+C	A	
	C6	C=C+C	A	
	CA	A=A+C	A	
	131	D1=A		
	147	C=DAT1	A	❸
	06	RSTK=C		
#3D7h	14A	A=DAT0	B	
	904	?A≠B	P	❹
	71	GOYES	%5	
	DB	C=D	A	❺
	80864	?ABIT=0	4	
	40	GOYES	%4	❻
	C9	C=C+B	A	
%4				❼
#3EAh	135	D1=C		
	306	C=(1)	#6	
	15D0	DAT1=C	1	

❶ Début du tableau en #387h.
❷ Point de branchement %3. Calcul de la position dans le tableau en fonction du numéro du tour en cours.
❸ Sauvegarde dans la pile des types de sprite à afficher.
❹ Probabilité = 1/16.
❺ Selon l'horloge (aléatoire) ...soit en haut (début du contenu),
❻ ...soit en bas (fin du contenu).
❼ Point de branchement %4 (mémorisation de la voiture détruite).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%5				❶
#3F4h	14A	A=DAT0	B	
	964	?A≠0	B	❷
	71	GOYES	%7	
	DB	C=D	A	❸
	80864	?ABIT=0	4	
	40	GOYES	%6	❹
	C9	C=C+B	A	
%6				❺
#407h	135	D1=C		
	308	C=(1)	#8	❻
	15D0	DAT1=C	1	
%7				❼
#411h	CD	B=B-1	A	
	CD	B=B-1	A	
	E7	D=D+1	A	
	1EDB20	D1=(4)	#02BD	
	14F	C=DAT1	A	❽
	171	D1=D1+	#1	
	14B	A=DAT1	A	
	EA	A=A-C	A	
	305	C=(1)	#5	❾
	982	?A<C	B	
	A2	GOYES	%9	
	14A	A=DAT0	B	
	80872	?ABIT=1	2	❿
	02	GOYES	%9	
	80873	?ABIT=1	3	
	91	GOYES	%9	
	D2	C=0	A	

❶ Point de branchement %5. Traitement des vies.
❷ Probabilité = 1/256.
❸ Selon l'horloge... ...soit en haut (début contenu).
❹ ...soit en bas (fin contenu).
❺ Point de branchement %6.
❻ On stocke vie dans contenu.
❼ Point de branchement %7. Traitement voiture adverse.
❽ Calcul de la largeur (différence des positions).
❾ Largeur minimale = 4 positions.
❿ Pseudo hasard.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%8				❶
#443h	15E0	C=DAT0	1	
	8B5	?B<C	A	
	9F	GOYES	%8	
	CB	C=C+D	A	❷
	135	D1=C		
	305	C=(1)	#5	❸
	15D0	DAT1=C	1	
%9				❹
#458h	E5	B=B+1	A	
	E5	B=B+1	A	❺
	CF	D=D-1	A	
	815	BSRC		❻
	07	C=RSTK		
	D5	B=C	A	❼
	303	C=(1)	#3	
%10				❽
#468h	14A	A=DAT0	B	
	80842	ABIT=0	2	
	80843	ABIT=0	3	
%11				❾
#475h	A0C	A=A-1A	P	
	470	GOC	%12	
	F5	BSR	A	
	57F	GONC	%11	
%12				❿
#480h	14A	A=DAT0	B	
	80873	?ABIT=1	3	
	74	GOYES	%15	

❶ Point de branchement %8. Largeur de la piste comme maximum.
❷ Calcul de la place dans le contenu.
❸ Mémorisation de la voiture adverse dans contenu.
❹ Point de branchement %9. Traitement des autres sprites.
❺ On se trouve à nouveau au bord de la route.
❻ Largeur dans B (S).
❼ Les sprites à afficher sont dans B (A).
❽ Point de branchement %10. Nombre aléatoire entre 0 et 3.
❾ Point de branchement %11. Copie du sprite dans P.
❿ Point de branchement %12. Diminution de la fréquence d'apparition.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	310A	C=(2)	#A0	❶
	9E6	?A>C	B	
	E3	GOYES	%15	
%13				❷
#493h	1564	C=DAT0	S	
	9C5	?B<C	S	
	9F	GOYES	%13	
	D2	C=0	A	❸
	812	CSLC		
	C3	D=D+C	A	
	816	CSRC		
	DB	C=D	A	
	135	D1=C		
	D9	C=B	A	❹
	15D0	DAT1=C	1	
	30F	C=(1)	#F	❺
	905	?B≠C	P	
	81	GOYES	%15	
	307	C=(1)	#7	❻
	15D0	DAT1=C	1	
	170	D1=D1+	#1	❼
	945	?B≠C	S	
	50	GOYES	%14	
	1C1	D1=D1-	#2	
%14				❽
#4CBh	15D0	DAT1=C	1	❾
%15				❿
#4CFh	1E0120	D1=(4)	#0210	
	14B	A=DAT1	B	

❶ Diminution de la fréquence d'apparition.
❷ Point de branchement %13. Test : largeur convenable ?
❸ Calcul de la place dans le contenu.
❹ Mémorisation du sprite.
❺ Test : s'agit-il d'un double obstacle ?
❻ ...dans l'affirmative on mémorise un obstacle simple.
❼ Calcul de l'emplacement du second obstacle simple.
❽ Point de branchement %14.
❾ Mémorisation du second obstacle simple.
❿ Point de branchement %15. Gestion fin de kilomètre ou fin de tour.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	A6C	A=A-1	B	❶
	149	DAT1=A	B	
	505	GONC	$3	❷
	3186	C=(2)	#68	
	14D	DAT1=C	B	❸
	1D1C	D1=(2)	#C1	
	35222222	C=(8)	#222222	
	15D5	DAT1=C	6	❹
	1D50	D1=(2)	#05	
	14B	A=DAT1	B	❺
	3161	C=(2)	#16	
	962	?A=C	B	❻
	92	GOYES	$3	
	05	SETDEC		
	C4	A=A+A	A	❼
	D6	C=C	A	
	C4	A=A+A	A	
	C2	C=C+A	A	
	04	SETHEX		
	1C4	D1=D1-	#5	❽
	14B	A=DAT1	B	
	966	?A≠C	B	
	21	GOYES	$3	
	1D1C	D1=(2)	#C1	
	35333333	C=(6)	#333333	❾
	15D5	DAT1=C	6	
$3				❿
#52Fh	110	A=R0		
	D2	C=0	A	

❶ Décrémentation du compteur kilométrique.
❷ Test : fin d'un kilomètre ?
❸ Initialisation du compteur de kilomètres.
❹ Dessin de la ligne de fin de kilomètre.
❺ Lecture du numéro du tour.
❻ Il n'y a pas de fin de tour s'il s'agit du 16ème tour.
❼ Calcul tour × #6h.
❽ Test : tour × #6h = numéro du kilomètre ?
❾ ...dans l'affirmative, on dessine la ligne de fin de tour.
❿ Procédure $3. Affichage de 18è colonne.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	3122	C=(2)	#22	❸
	CA	A=A+C	A	
	320E7	C=(3)	#7E0	
	C2	C=C+A	A	❷
	D7	D=C	A	
	D8	B=A	A	
	130	D0=A		❸
	306	C=(1)	#1	
%1				❹
#54Bh	06	RSTK=C		
	D2	C=0	A	
	7813	GOSUB	$SPR	❺
#553h	07	C=RSTK		
	A0E	C=C-1	P	
	52F	GONC	%1	
	3402100	C=(5)	#00120	❻
	1DE0	D1=(2)	#0E	
	14B	A=DAT1	B	
%2				❼
#569h	A0C	A=A-1	P	
	490	GOC	%3	
	C1	B=B+C	A	❽
	E3	D=D-C	A	
	55F	GONC	%2	
%3				❾
#576h	D4	A=B	A	
	130	D0=A		
#57Bh	D0	A=0	A	❿
	15B0	A=DAT1	1	

❶ B pointe sur la 1ère case de la 18ème colonne du tableau de jeu.
❷ D pointe sur la dernière case de la 18ème colonne.
❸ On pointe maintenant en haut de l'écran.
❹ Point de branchement %1.
❺ Affiche un « blanc » à 7 reprises (sur toute la colonne).
❻ Décalage entre deux cases d'une colonne de l'écran.
❼ Point de branchement %2.
❽ Case du dessous puis case du dessus...
❾ Point de branchement %3. Initialisation de D0.
❿ Lecture du champ de vision.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1DDB	D1=(2)	#BD	
	14F	C=DAT1	B	❶
	EA	A=A-C	A	
	341C207	C=(5)	#702C1	
	A62	C=C+A	B	❷
	D5	B=C	A	
	14B	A=DAT1	B	❸
	171	D1=D1+	#2	
	14F	C=DAT1	B	❹
	B62	C=C-A	B	
	171	D1=D1+	#2	
	133	ADlex		
	A6A	A=A+C	B	❺
	DC	ABex	A	
	131	D1=A		
%4				❻
#5B0h	133	ADlex		
	131	D1=A		
	311C	C=(2)	#C1	
	9E2	?A<C	B	
	D0	GOYES	%5	❼
	9E0	?A>B	B	
	80	GOYES	%5	
	14F	C=DAT1	B	
	550	GONC	%6	
%5				❽
#5CAh	301	C=(1)	#1	
%6				❾
#5CDh	7A92	GOSUB	$SPR	

❶ Lecture de la position de la première case.
❷ B pointe sur la première case (du champ de vision...).
❸ A nouveau, lecture de la première case.
❹ Lecture de la position de la dernière case et calcul de la largeur.
❺ On pointe sur la fin du contenu. B = dernière case du contenu.
A = 1ère case du champ de vision.
❻ Point de branchement %4. Case courante puis 1ère case de la colonne.
❼ Tests : est-on déjà dans le contenu ? Est-on encore dans le contenu ?
❽ Point de branchement %5. Lecture des bords.
❾ Point de branchement %6. Affichage des sprites.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#5D1h	170	D1=D1+	#1	❶
	136	CD0ex		
	134	D0=C		
	8B3	?C<D	A	❷
	3D	GOYES	%4	
$4				❸
#5DFh	844	ST=0	4	
	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	
	808A5	?CBIT=0	5	❹
	01	GOYES	%1	
	843	ST=0	3	❺
	854	ST=1	4	
	860	?ST=0	0	
	50	GOYES	%1	❻
	853	ST=1	3	
%1				❼
#603h	32400	C=(3)	#004	
	8FCEE10	GOSBVL	#01EEC	
	808A5	?CBIT=0	5	❽
	51	GOYES	%2	
	874	?ST=1	4	
	91	GOYES	$5	
	843	ST=0	3	❾
	854	ST=1	4	
	870	?ST=1	0	
	50	GOYES	%2	❿
	853	ST=1	3	

❶ Passage à la case suivante dans le tableau.
❷ Test : est-on encore dans le champ de vision.
❸ Procédure $4 : test des touches.
❹ Test : pression sur [α] ?
❺ Déplacement vers le haut (touche pressée).
❻ Si inverseur, déplacement vers le bas.
❼ Point de branchement %1.
❽ Test : pression sur [←] ? Avait-on appuyé sur [α] ?
❾ Déplacement vers le haut (touche pressée).
❿ En l'absence d'inverseur, déplacement vers le bas.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%2				❶
#629h	864	?ST=0	4	
	60	GOYES	$5	
	7592	GOSUB	$APPUI	❷
$5				❸
#632h	1E0120	D1=(4)	#0210	❹
	14B	A=DAT1	B	
	3106	C=(2)	#60	
	9E6	?A>C	B	
	71	GOYES	%1	❺
	176	D1=D1+	#7	
	14B	A=DAT1	B	
	302	C=(1)	#2	
	902	?A=C	P	
	D0	GOYES	%2	❻
	E6	C=C+1	A	
	902	?A=C	P	❼
	60	GOYES	%2	
%1				❽
#658h	6AE0	GOTO	$6	
%2				❾
#65Ch	841	ST=0	1	
	D8	B=A	A	
	1D00	D1=(2)	#00	
	14B	A=DAT1	B	
	05	SETDEC		
	B64	A=A+1	B	❿
	04	SETHEX		
	149	DAT1=A	B	

❶ Point de branchement %2. Test : pression sur [α] ou sur [←] ?
❷ Déplacement. Test : collision ?
❸ Procédure $5 (kilomètre suivant, tour suivant).
❹ Lecture du compteur de kilomètres.
❺ Test : est-ce une ligne délimitant un kilomètre ?
❻ Test : est-ce une ligne de fin de kilomètre ?
❼ Test : est-ce une ligne de fin de tour ?
❽ Point de branchement %1.
❾ Point de branchement %2. Inhibition de l'accélérateur.
❿ Modification du numéro du kilomètre.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	560	GONC	%3	❶
#676h	6FC6	GOTO	$GAGNE	
%3				❷
#67Ah	303	C=(1)	#3	
	901	?B=C	P	❸
	C2	GOYES	%7	
	D2	C=0	A	
	31F5	C=(2)	#5F	❹
	D5	B=C	A	
	D7	D=C	A	
	28	P=	8	
%4				❺
#68Eh	80F2	CPex	2	
	862	?ST=0	2	❻
	50	GOYES	%5	
	801	OUT=C		
%5				❼
#69Ah	D4	A=B	A	
%6				❽
#69Ch	CC	A=A-1	A	
	5DF	GONC	%6	
	CF	D=D-1	A	❾
	5AE	GONC	%4	
	20	P=	0	
	6B90	GOTO $6		
%7				❿
#6ACh	174	D1=D1+	#5	
	14B	A=DAT1	B	
	05	SETDEC		

❶ Test : fin du jeu ?
❷ Point de branchement %3.
❸ Test : est-ce une ligne de fin de tour ?
❹ Fréquence dans B, durée dans D.
❺ Point de branchement %4.
❻ Bip autorisé ? Bip !
❼ Point de branchement %5.
❽ Point de branchement %6. Temporisation fréquence.
❾ Boucle durée.
❿ Point de branchement %7. Début de la gestion du tour suivant.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	E4	A=A+1	A	❶
	04	SETHEX		
	149	DAT1=A	B	
	840	ST=0	0	❷
	177	D1=D1+	#8	
	14B	A=DAT1	B	❸
	E4	A=A+1	A	
	1590	DAT1=A	1	
	304	C=(1)	#4	
	902	?A=C	P	
	A0	GOYES	%8	
	30E	C=(1)	#E	
	906	?A≠C	P	❹
	34	GOYES	%9	❺
%8				❻
#6DAh	170	D1=D1+	#1	
	D0	A=0	A	
	15B0	A=DAT1	1	
	D6	C=A	A	
	E4	A=A+1	A	❼
	1590	DAT1=A	1	
	110	A=R0		
	F2	CSL	A	❽
	F2	CSL	A	
	CA	A=A+C	A	
	F6	CSR	A	
	C6	C=C+C	A	
	CA	A=A+C	A	
	130	D0=A		❾
	72A1	GOSUB	$LIGNE	❿

❶ Incrémentation du numéro du tour (décimal).
❷ Suppression de l'inverseur.
❸ Incrémentation du numéro du tour (hexadécimal).
❹ Réduction du champ de vision.
❺ A partir du tour #4h et #Dh...
❻ Point de branchement %6.
❼ Modification du champ de vision.
❽ Champ de vision × #120h.
❾ On pointe sur la ligne à effacer.
❿ Branchement sur la procédure $LIGNE.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#701h	110	A=R0		
	C4	A=A+A	A	
	136	CD0ex		
	EA	A=A-C	A	
	344E600	C=(5)	#006E4	
	CA	A=A+C	A	
	130	D0=A		❶
	7881	GOSUB	$LIGNE	
%9				❷
#71Bh	862	?ST=0	2	
	62	GOYES	$6	❸
	D2	C=0	A	
	30C	C=(1)	#C	❹
	D5	B=C	A	
	311C	C=(2)	#C1	❺
	D7	D=C	A	
	28	P=	8	
%10				❻
#72Fh	80F2	CPex	2	
	801	OUT=C		
	D4	A=B	A	
%11				❼
#738h	CC	A=A-1	A	
	5DF	GONC	%11	
	CF	D=D-1	A	
	5FE	GONC	%10	❽
	20	P=	0	
$6				❾
#744h	1D70	D1=(2)	#07	

❶ On efface aussi la ligne symétrique de la première ligne à effacer par rapport à la ligne centrale.
❷ Point de branchement %9.
❸ Test : bip autorisé ?
❹ B contient la fréquence du bip.
❺ D contient la durée du bip.
❻ Point de branchement %10. Bip !
❼ Point de branchement %11. Temporisation pour la fréquence.
❽ Boucle sur la durée.
❾ Procédure $6 (affichage des données).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	15B5	A=DAT1	6	❶
	AF2	C=0	W	
	307	C=(1)	#7	
	05	SETDEC		
	A7A	A=A+C	W	❷
	04	SETHEX		
	1595	DAT1=A	6	
	3451100	C=(5)	#00115	
	110	A=R0		❸
	EA	A=A-C	A	
	130	D0=A		
	7430	GOSUB	$AFF	❹
#770h	16C	D0=D0+	#D	
	1E0020	D1=(4)	#0200	
	301	C=(1)	#1	
	7420	GOSUB	$AFF	❺
#780h	164	D0=D0+	#5	
	1E5020	D1=(4)	#0205	
	301	C=(1)	#1	
	7410	GOSUB	$AFF	❻
#790h	16B	D0=D0+	#C	
	1E2020	D1=(4)	#0202	
	302	C=(1)	#2	
	7400	GOSUB	$AFF	❼
#7A0h	6D3A	GOTO	$1	
#7A4h				
$AFF				❽
#000h	D7	D=C	A	❾
	15F5	C=DAT1	6	❿

❶ Lecture du score.
❷ Ajout de 7 au score.
❸ Calcul de l'adresse des unités sur l'écran.
❹ Affichage du score (6 quartets).
❺ Affichage du score (2 quartets).
❻ Affichage du nombre tours (2 quartets).
❼ Affichage du nombre de vies (2 quartets).
❽ Procédure $AFF placée en #7A4h (remise à #000h des adresses).
❾ D contient le nombre de quartets.
❿ B contient les quartets eux-mêmes.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	AF5	B=C	W	
%1				❶
#009h	111	A=R1		
	D2	C=0	A	
	A89	C=B	P	
	EA	A=A-C	A	
	F2	CSL	A	
	81E	CSRB		
	CA	A=A+C	A	
	139	D1=AS		❷
	306	C=(1)	#6	❸
%2				❹
#020h	14B	A=DAT1	B	
	1580	DAT0=A	1	
	16F	D0=D0+	#10	❺
	16F	D0=D0+	#10	
	163	D0=D0+	#4	
	170	D1=D1+	#1	❻
	A0E	C=C-1	P	❼
	59E	GONC	%2	
	132	AD0ex		❽
	D2	C=0	A	
	31DF	C=(2)	#FD	
	EA	A=A-C	A	
	130	D0=A		
	BF5	BSR	W	❾
	A0F	D=D-1	P	
	5BB	GONC	%1	❿
	01	RTN		

❶ Point de branchement %1. A = R1+B×#7h.
❷ Cette ligne est imposée par l'instruction CSRB (2 lignes au-dessus).
❸ 7 lignes.
❹ Point de branchement %2. Transfert des données vers l'écran.
❺ Passage à la ligne inférieure.
❻ Donnée suivante.
❼ Boucle sur 7 lignes.
❽ On pointe sur le chiffre précédent.
❾ Quartet précédent.
❿ Boucle sur les lignes.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$ON				❶
#052h	110	A=R0		
	345E400	C=(5)	#004E5	
	C2	C=C+A	A	
	06	RSTK=C		❷
	134	D0=C		
	113	A=R3		
	131	D1=A		
	30C	C=(1)	#C	
%1				❸
#06Ch	143	A=DAT1	A	
	140	DAT0=A	A	❹
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❺
	163	D0=D0+	#4	
	173	D1=D1+	#4	❻
	A0E	C=C-1	P	
	5AE	GONC	%1	
%2				❼
#084h	32004	C=(3)	#400	
	8FCEE10	GOSBVL	#01EEC	
	808AF	?CBIT=0	15	❽
	FE	GOYES	%2	
	07	C=RSTK		
	134	D0=C		❾
	D0	A=0	A	
	30C	C=(1)	#C	
%3				❿
#0A1h	140	DAT0=A	A	

❶ Procédure $ON. Position en bas à droite.
❷ Copie de D0 en vue de l'effacement du sprite.
❸ 13 lignes...
❹ Transfert des données vers l'écran.
❺ Passage à la ligne du dessous.
❻ Données suivantes et boucle sur 13 lignes.
❼ Point de branchement %2.
❽ Attente d'une pression sur [ON].
❾ Initialisation de D0.
❿ Point de branchement %3.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❶
	163	D0=D0+	#4	
	A0E	C=C-1	P	
	50F	GONC	%3	❷
	01	RTN		
$VOITURE			❸	
#0B5h	3406300	C=(5)	#00360	
	110	A=R0		
	CA	A=A+C	A	❹
	130	D0=A		
	304	C=(1)	#4	
$SPR				❺
#0C7h	DA	A=C	A	
	137	CD1ex		❻
	06	RSTK=C		
	D2	C=0	A	
	A86	C=A	P	
	F2	CSL	A	❼
	112	A=R2		
	CA	A=A+C	A	
	131	D1=A		
	3170	C=(2)	#07	❽
%1				❾
#0E1h	14B	A=DAT1	B	
	148	DAT0=A	B	
	16F	D0=D0+	#10	❿
	16F	D0=D0+	#10	

❶ Passage à la ligne du dessous.
❷ Boucle sur 13 lignes.
❸ Procédure $VOITURE.
❹ On pointe sur l'écran.
❺ Procédure $SPR.
❻ D1 ne sera pas altéré par cette routine.
❼ D1 = R2+#10h×numéro (un sprite est représenté par 16 quartets).
❽ 8 lignes.
❾ Point de branchement %1. Transfert des données vers l'écran.
❿ Passage à la ligne du dessous.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	163	D0=D0+	#4	
	171	D1=D1+	#2	❶
	CE	C=C-1	A	
	5BE	GONC	%1	
	07	C=RSTK		❷
	135	D1=C		
	01	RTN		
$LIGNE				❸
#0FFh	D2	C=0	A	
	3111	C=(2)	#11	
	D7	D=C	A	
	32E11	C=(3)	#11E	❹
	D5	B=C	A	
%1				❺
#10Eh	D2	C=0	A	
	73BF	GOSUB	$SPR	❻
#114h	136	CD0ex		
	E9	C=C-B	A	
	134	D0=C		
	CF	D=D-1	A	
	5FE	GONC	%1	
	01	RTN		
$APPUI				❼
#123h	3456200	C=(5)	#00265	❽
	110	A=R0		
	863	?ST=0	3	❾
	70	GOYES	%1	
	325A4	C=(3)	#4A5	❿

❶ Donnée suivante. Boucle 8 lignes.
❷ Restauration de D1.
❸ Procédure $LIGNE (sur 18 colonnes...).
❹ Décalage vers la case suivante de la même ligne.
❺ Point de branchement %1.
❻ Affiche un blanc et passe à la case suivante.
❼ Procédure $APPUI (test collision).
❽ Case supérieure sur l'écran...
❾ ...sauf si on descend !
❿ Dans ce second cas, on passe à la case inférieure.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%1				❶
#137h	CA	A=A+C	A	
	130	D0=A		
	14A	A=DAT0	B	
	77A2	GOSUB	$SCRASH	❷
#143h	32063	C=(3)	#360	
	110	A=R0		
	CA	A=A+C	A	❸
	130	D0=A		
	D2	C=0	A	
	717F	GOSUB	$SPR	
$APPUI2				
#156h	1D31	D1=(2)	#13	❹
	3132	C=(2)	#23	
%2				❺
#15Eh	14B	A=DAT1	B	
	E4	A=A+1	A	
	863	?ST=0	3	
	60	GOYES	%3	❻
	CC	A=A-1	A	
	CC	A=A-1	A	
%3				❼
#16Ch	149	DAT1=A	B	
	171	D1=D1+	#2	
	808B0	?CBIT=1	0	❽
	50	GOYES	%4	
	175	D1=D1+	#6	❾
%4				❿
#17Ch	A6E	C=C-1	B	

❶ Point de branchement %1.
❷ Test : collision ? On efface la voiture...
❸ Procédure $APPUI2.
❹ On pointe au début du tableau.
❺ Point de branchement %2. Lecture de la position.
❻ Réajustement en cas de descente.
❼ Point de branchement %3. Sauvegarde de la nouvelle position.
❽ Test de la parité du compteur.
❾ Réajustement colonne suivante.
❿ Point de branchement %4.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	5ED	GONC	%2	
#182h	1DE0	D1=(2)	#0E	
	110	A=R0		
	3402100	C=(5)	#00120	
	D5	B=C	A	
	873	?ST=1	3	
	B0	GOYES	%5	❶
	320D7	C=(3)	#7D0	
	CA	A=A+C	A	
	F9	B=-B	A	
%5				❷
#1A0h	14F	C=DAT1	B	
	06	RSTK=C		❸
	DD	BCex		❹
%6				❺
#1A7h	CA	A=A+C	A	
	A0D	B=B-1	P	❻
	5AF	GONC	%6	
	131	D1=A		❼
	EA	A=A-C	A	
	130	D0=A		❽
	07	C=RSTK		
	D7	D=C	A	
	31F8	C=(2)	#8F	
	D5	B=C	A	❾
	3142	C=(2)	#24	
%7				❿
#1C5h	E1	B=B-C	A	
	A0F	D=D-1	P	

❶ En cas de montée, le scrolling commence en fin d'écran.
❷ Point de branchement %5. Lecture du champ de vision.
❸ Sauvegarde du champ de vision.
❹ + ou - #120h dans C.
❺ Point de branchement %6.
❻ Modification du début du scrolling selon le champ de vision.
❼ D1 = D0 + ou - #120h.
❽ D0 = R0 (+#7D0h) + ou - #120h× champ de vision...
❾ Compteur dans B.
❿ Point de branchement %7. Compteur = #6Bh - #24h ¥ champ de vision.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	5AF	GONC	%7	
%8				❶
#1CDh	1537	A=DAT1	W	
	1507	DAT0=A	W	
	873	?ST=1	3	❷
	A1	GOYES	%9	
	18F	D0=D0+	#10	
	1CF	D1=D1+	#10	
	A6D	B=B-1	B	
	59E	GONC	%8	
	17F	D1=D1+	#10	
	137	CD1ex		❸
	134	D0=C		
	5E0	GONC	%10	
%9				❹
#1F2h	16F	D0=D0-	#10	
	17F	D1=D1-	#10	
	A6D	B=B-1	B	
	51D	GONC	%8	❺
%10				❻
#1FEh	D2	C=0	A	
	1FE0207	D1=(5)	#7020E	
	15F0	C=DAT1	1	❼
	D7	D=C	A	
	176	D1=D1+	#7	❽
	3111	C=(2)	#11	❾
%11				❿
#214h	06	RSTK=C		

❶ Point de branchement %8. Transfert inter-écran...
❷ Test : descente ? Quartets suivants
❸ D0 pointe sur les quartets à afficher.
❹ Point de branchement %9. Quartets suivants.
❺ D0 est déjà pointé...
❻ Point de branchement %10. Initialisation des données.
❼ Champ de vision dans D
❽ On pointe sur la position de la dernière case.
❾ C contient le compteur.
❿ Point de branchement %11.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	DB	C=D	A	❶
	863	?ST=0	3	
	70	GOYES	%12	❷
	306	C=(1)	#6	
	EB	C=C-D	A	
%12				❸
#222h	14B	A=DAT1	B	
	1C1	D1=D1-	#2	
	B6A	A=A-C	B	❹
	A64	A=A+A	B	
	482	GOC	%14	❺
	14B	A=DAT1	B	
	B62	C=C-A	B	
	808B7	?CBIT=1	7	
	B1	GOYES	%14	
%13				❻
#23Eh	133	AD1ex		
	D8	B=A	A	
	A6A	A=A+C	B	
	131	D1=A		
	173	D1=D1+	#4	
	14F	C=DAT1	B	❼
	D4	A=B	A	
	131	D1=A		❽
	550	GONC	%15	
%14				❾
#257h	301	C=(1)	#1	
%15				❿
#25Ah	796E	GOSUB	$SPR	
#25Eh	17B	D1=D1+	#C	

❶ C = champ de vision...
❷ ...mais si une descente est en cours C vaut 6 moins le champ de vision.
❸ Point de branchement %12.
❹ Test : position de la dernière case < C ?
❺ Lecture de la position de la 1ère case. Test : position > C ?
❻ Point de branchement %13. Sauvegarde de D1.
❼ Lecture du sprite à afficher.
❽ Restauration de D1.
❾ Point de branchement %14. Lecture des bords.
❿ Point de branchement %15. Affichage du sprite. Colonne suivante.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	132	AD0ex		
	34E1100	C=(5)	#0011E	
	EA	A=A-C	A	❶
	130	D0=A		
	07	C=RSTK		
	CE	C=C-1	A	❷
	5F9	GONC	%11	
#277h	3458300	C=(5)	#00385	
	110	A=R0		❸
	CA	A=A+C	A	
	130	D0=A		
	14A	A=DAT0	B	
$COLLISION				❹
#289h	302	C=(1)	#2	❺
	906	?A≠C	P	
	93	GOYES	%6	
	870	?ST=1	0	❻
	80	GOYES	%1	
	850	ST=1	0	
	550	GONC	%2	
%1				❼
#29Ch	840	ST=0	0	
%2				❽
#29Fh	862	?ST=0	2	
	32	GOYES	%5	
	D2	C=0	A	
	31F6	C=(2)	#6F	
	D5	B=C	A	❾

❶ On pointe sur la case suivante de la même ligne.
❷ Boucle sur 18 colonnes.
❸ Lecture du quartet correspondant à la position de la voiture.
❹ Procédure $COLLISION (test collision).
❺ Test : inverseur ?
❻ Inversion du flag ST0.
❼ Point de branchement %1.
❽ Point de branchement %2. Test : le bip est-il autorisé ? Si ce n'est pas le cas, on affiche la voiture.
❾ B contient la fréquence...

CCD 20

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D2	C=0	A	
	28	P=	8	
%3				❶
#2B0h	80F2	CPex	2	
	801	OUT=C		
	D4	A=B	A	
%4				❷
#2B9h	CC	A=A-1	A	
	5DF	GONC	%4	
	CD	B=B-1	A	❸
	5FE	GONC	%3	
	20	P=	0	
%5				❹
#2C5h	416	GOC	%14	
%6				❺
#2C8h	304	C=(1)	#4	
	906	?A≠C	P	
	80	GOYES	%7	
	851	ST=1	1	❻
	5BC	GONC	%2	❼
%7				❽
#2D6h	307	C=(1)	#7	
	906	?A≠C	P	
	F4	GOYES	%15	
%8				❾
#2DEh	1D20	D1=(2)	#02	
	147	C=DAT1	A	
	05	SETDEC		❿

❶ Point de branchement %3. Bip !
❷ Point de branchement %4. Temporisation fréquence.
❸ Boucle durée (de plus en plus aigu et rapide).
❹ Point de branchement %5.
❺ Point de branchement %6. Test : accélérateur ?
❻ Activation du flag accélérateur.
❼ Bip modulé croissant.
❽ Point de branchement %7. Test : vie ?
❾ Point de branchement %8.
❿ Préparation à l'ajout d'une vie.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	E6	C=C+1	A	
	04	SETHEX		
	1553	DAT1=C	X	
	D2	C=0	A	❶
	308	C=(1)	#8	
%9				❷
#2F4h	06	RSTK=C		
	30B	C=(1)	#B	
	D5	B=C	A	❸
	D7	D=C	A	
	28	P=	8	
%10				❹
#2FFh	80F2	CPex	2	
	862	?ST=0	2	
	50	GOYES	%11	
	801	OUT=C		❺
%11				❻
#30Bh	D4	A=B	A	
%12				❼
#30Dh	CC	A=A-1	A	
	5DF	GONC	%12	
	CF	D=D-1	A	
	5AE	GONC	%10	
	80F2	CPex	2	❽
%13				❾
#31Bh	CE	C=C-1	A	
	5DF	GONC	%13	
	07	C=RSTK		
	CE	C=C-1	A	❿
	5FC	GONC	%9	

❶ 9 « tics ! ».
❷ Point de branchement %9.
❸ B contient la fréquence et D la durée.
❹ Point de branchement %10. Test : bip autorisé ?
❺ Bip !
❻ Point de branchement %11.
❼ Point de branchement %12. Temporisation fréquence. Boucle durée.
❽ P = 0, c = #800h.
❾ Point de branchement %13. Temporisation entre deux « tics ! ».
❿ « Tic » suivant...

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%14				❶
#327h	6D8D	GOTO	$VOITURE	
%15				❷
#32Bh	30C	C=(1)	#C	
	906	?A≠C	P	
	D5	GOYES	%18	
	1D70	D1=(2)	#07	
	15B5	A=DAT1	6	❸
	AF2	C=0	W	
	3104	C=(2)	#40	
	05	SETDEC		
	A7A	A=A+C	W	
	04	SETHEX		
	1595	DAT1=A	6	
	177	D1=D1+	#8	
	14B	A=DAT1	B	❹
	B04	A=A+1	P	
	1590	DAT1=A	1	❺
	438	GOC	%8	
	862	?ST=0	2	❻
	7C	GOYES	%14	
	310F	C=(2)	#F0	
	D5	B=C	A	❼
	3151	C=(2)	#15	
	D7	D=C	A	❽
%16				❾
#36Eh	32008	C=(3)	#800	
	801	OUT=C		❿
	6300	NOP3		

❶ Point de branchement %14. Affichage de la voiture.
❷ Point de branchement %15. Test : essence ?
❸ Modification du score (+40).
❹ +1 au compteur essence.
❺ Test : 16 incrémentation successives ? Si oui, ajout d'une vie.
❻ Test : bip autorisé ?
❼ B contient la fréquence.
❽ D contient la durée.
❾ Point de branchement %16.
❿ Bip ! Avec un volume moyen.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D2	C=0	A	
	801	OUT=C		
	D4	A=B	A	
%17				❶
#381h	CC	A=A-1	A	
	5DF	GONC	%17	
	CF	D=D-1	A	
	55E	GONC	%16	❷
	4B9	GOC	%14	
%18				❸
#38Eh	30F	C=(1)	#F	
	906	?A≠C	P	
	39	GOYES	%14	
	798D	GOSUB	$APPUI	
#39Ah	758D	GOSUB	$APPUI	
#39Eh	862	?ST=0	2	❹
	00	RTNYES		
	D1	B=0	A	
	D2	C=0	A	
	B25	B=B+1	XS	❺
	30F	C=(1)	#F	
	D7	D=C	A	❻
	28	P=	8	
%19				❼
#3B1h	80F2	CPex	2	
	801	OUT=C		
	D4	A=B	A	
%20				❽
#3BAh	CC	A=A-1	A	
	5DF	GONC	%20	
	81861F	B=B+#10	B	

❶ Point de branchement %17. Temporisation fréquence.
❷ Boucle durée.
❸ Point de branchement %18. Traitement s'il s'agit d'une flaque d'huile.
❹ Test : bip autorisé ?
❺ Fréquence dans B;
❻ 16 fois la boucle.
❼ Point de branchement %19. Bip !
❽ Point de branchement %20. Temporisation fréquence.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	5BE	GONC	%19	❶
	CD	B=B-1	A	❷
%21				❸
#3CAh	80F2	CPex	2	
	801	OUT=C		❹
	D4	A=B	A	
%22				❺
#3D3h	CC	A=A-1	A	
	5DF	GONC	%22	
	81869F	B=B-#10	B	❻
	5BE	GONC	%21	
	CF	D=D-1	A	❼
	5DC	GONC	%19	
	20	P=	0	
	01	RTN		
$SCRASH				❽
#3EAh	80870	?ABIT=1	0	
	00	RTNYES		
	80861	?ABIT=0	1	
	00	RTNYES		
	302	C=(1)	#2	
	902	?A=C	P	❾
	00	RTNYES		
	110	A=R0		
	3406300	C=(5)	#00360	
	CA	A=A+C	A	
	130	D0=A		❿
	306	C=(1)	#6	
	71BC	GOSUB	$SPR	

❶ Boucle durée... plus grave !
❷ Réajustement pour l'effet.
❸ Point de branchement %21.
❹ Bip !
❺ Point de branchement %22. Temporisation fréquence.
❻ Boucle durée... plus aiguë...
❼ ...16 fois !
❽ Procédure $SCRASH.
❾ Seuls sont acceptés #6h, #Ah ou #Eh.
❿ Affichage de la voiture détruite.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#416h	31D7	C=(2)	#7D	❶
	D5	B=C	A	
	110	A=R0		
	130	D0=A		
%1				❷
#422h	1527	A=DAT0	W	
	BFC	A=-A-1	W	❸
	1507	DAT0=A	W	
	16F	D0=D0+	#10	❹
	310A	C=(2)	#A0	
%2				❺
	A6E	C=C-1	B	
	5CF	GONC	%2	
	A6D	B=B-1	B	
	54E	GONC	%1	❻
	D0	A=0	A	
	D1	B=0	A	❼
	D2	C=0	A	
	D3	D=0	A	
	CC	A=A-1	A	
	28	P=	8	
%3				❽
#44Ch	B65	B=B+1	B	
	451	GOC	%4	
	B67	D=D+1	B	
	56F	GONC	%5	
	D7	D=C	A	❾
	D6	C=A	A	
	DF	CDex	A	
	C7	D=D+D	A	❿

❶ Tout l'écran !
❷ Point de branchement %1.
❸ Inversion vidéo.
❹ 16 quartets à la fois.
❺ Point de branchement %2.
❻ Quartets suivants. Durée.
❼ Fréquence 1 puis fréquence 2.
❽ Point de branchement %3. Temporisation fréquences 1 puis 2.
❾ Les instructions **D7**, **D6** puis **DF** correspondent à **D=A A.**
❿ D = A × #4h.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	C7	D=D+D	A	
	5A0	GONC	%6	
%4				❶
#465h	D8	B=A	A	
	A6C	A=A-1	B	
	421	GOC	%7	
%5				❷
#46Dh	80F2	CPex	2	
	862	?ST=0	2	
	50	GOYES	%6	
	801	OUT=C		❸
%6				❹
#479h	62DF	GOTO	%3	
%7				❺
#47Dh	D5	B=C	A	
	D2	C=0	A	
	801	OUT=C		
%8				❻
#484h	CD	B=B-1	A	
	5DF	GONC	%8	
	31F1	C=(2)	#1F	❼
	D5	B=C	A	
%9				❽
#48Fh	31CF	C=(2)	#FC	
	110	A=R0		
	130	D0=A		
%10				❾
#499h	15A7	A=DAT0	8	
	A74	A=A+A	W	❿
	1587	DAT0=A	8	

❶ Point de branchement %4. B=A puis boucle durée
❷ Point de branchement %5.
❸ Bip commun aux deux fréquences.
❹ Point de branchement %6.
❺ Point de branchement %7. OUT=C (un peu de ménage...).
❻ Point de branchement %8.
❼ 32 répétitions (8 quartets × 4 bits).
❽ Point de branchement %9. On pointe au début de l'écran.
❾ Point de branchement %10.
❿ On escamote 1 bit.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	167	D0=D0+	#8	
	A6E	C=C-1	B	❶
	5EE	GONC	%10	
%11				❷
#4ADh	A6E	C=C-1	B	
	5CF	GONC	%11	
	A6D	B=B-1	B	
	58D	GONC	%8	
	1E2020	D1=(4)	#0202	
	143	A=DAT1	A	
	05	SETDEC		
	A3C	A=A-1	X	❸
	04	SETHEX		
	1513	DAT1=A	X	
	4D1	GOC	%12	❹
	110	A=R0		
	3410100	C=(5)	#00101	
	EA	A=A-C	A	
	130	D0=A		❺
	E6	C=C+1	A	
	7B1B	GOSUB	$AFF	
#4E5h	8C584F	GOLONG	$0	
#4EBh	34B4000	C=(5)	#0004B	❻
$SOUND				❼
#4F2h	862	?ST=0	2	❽
	74	GOYES	%5	
	81B4	A=PC		
#4FBh	CA	A=A+C	A	❾
	131	D1=A		

❶ Quartets suivants.
❷ Point de branchement %11. Temporisation.
❸ Suppression d'une vie.
❹ Test : perdu ? Vous avez perdu si vous n'avez plus de vie. En ce cas, c'est la fin du jeu...
❺ Affichage du nombre de vie(s).
❻ Décalage PC/Données.
❼ Procédure $SOUND.
❽ Test : bip autorisé ?
❾ On pointe sur les données.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%1				❶
#500h	32003	C=(3)	#300	
%2				❷
#505h	A3E	C=C-1	X	
	5CF	GONC	%2	
	143	A=DAT1	A	❸
	908	?A=0	P	
	B2	GOYES	%5	❹
	172	D1=D1+	#3	
	14F	C=DAT1	B	❺
	171	D1=D1+	#2	
	D7	D=C	A	
	D2	C=0	A	
	28	P=	8	
%3				❻
#522h	80F2	CPex	2	
	801	OUT=C		
	D8	B=A	A	
%4				❼
#52Bh	A3D	B=B-1	X	
	5CF	GONC	%4	
	A6F	D=D-1	B	❽
	5DE	GONC	%3	
	20	P=	0	
	46C	GOC	%1	
%5				❾
#53Ch	721B	GOSUB	$ON	
#540h	D2	C=0	A	
	81B3	PC=C		❿

❶ Point de branchement %1.
❷ Point de branchement %2. Temporisation entre deux fréquences.
❸ Lecture de la fréquence.
❹ Test : fin ?
❺ Lecture de la durée.
❻ Point de branchement %3. Bip !
❼ Point de branchement %4. Temporisation fréquence.
❽ Boucle durée.
❾ Point de branchement %5. Attente d'une pression sur [ON].
❿ Reset système.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	$DONNEES			❶
	#546h	1010F	*Codage des données :*	❷
		1010A	*3 quartets pour la*	
		10105	*fréquence, puis*	
		1010F	*2 quartets pour la*	
		7D00A	*durée suivis d'un 0*	
		5E005	*final.*	
		5E00A		
		10105		
		1010A		
		E0105		
		101FF		
		0		
	#57Eh	7D006	*idem.*	❸
		7D003		
		7D003		
		F1102		
		7D003		
		DA003		
		7D009		
		0		
	$GAGNE			❹
	#5A2h	110	A=R0	❺
		130	D0=A	
		114	A=R4	❻
		131	D1=A	
		3183	C=(2) #38	❼
		D5	B=C A	
	%1			❽
	#5B4h	32009	C=(3) #900	

❶ Procédure $DONNEES.
❷ Données en cas d'échec (perdu !)...
❸ Données en cas de victoire (gagné !)...
❹ Procédure $GAGNE.
❺ On pointe sur l'écran.
❻ On pointe sur la coupe de victoire (GROB R4).
❼ Nombre de lignes.
❽ Temporisation.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%2				❶
#5B9h	A3E	C=C-1	X	
	5CF	GONC	%2	
	15BB	A=DAT1	12	❷
	158B	DAT0=A	12	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❸
	163	D0=D0+	#4	
	17B	D1=D1+	#C	❹
A6D	B=B-1	B		
	5DD	GONC	%1	
#5D9h	3438000	C=(5)	#00083	❺
#5E0h	611F	GOTO	$SOUND	
#5E4h				

❶ Point de branchement %2.
❷ Transfert des données vers l'écran.
❸ Passage à la ligne du dessous.
❹ Passage à la donnée suivante.
❺ Décalage PC-données.

3

Exécuter le programme

Pour exécuter le programme, vous devez réunir dans le répertoire DSETH (spécialement créé), les objets suivants :

- 'RPL', le programme lanceur en RPL,
- 'CODE', le code exécutable,
- 'R0', 'R1', 'R2', 'R3' et 'R4', les 5 objets graphiques.

Les objets de développement tels que la chaîne de codes hexadécimaux ('SETH48S') et le programme ASS peuvent rester dans le répertoire DSETH mais ils ne seront pas utilisés pendant l'exécution du programme. Pour gagner de la mémoire, vous pouvez donc supprimer 'SETH48S' et 'ASS'.

Pour jouer, il suffit d'évaluer le programme 'RPL' qui est le lanceur écrit en langage RPL. Le message « SETH HP-48S/SX » étant affiché, appuyez ensuite sur [ENTER] pour lancer le jeu. Vous pouvez aussi appuyer simultanément sur [α]-[↱]-[F] pour reprendre une partie en cours (ne pas utiliser cette possibilité si vous venez d'effectuer une transmission par infrarouge). Une pression sur [ON] fera ensuite démarrer votre voiture...

4

Comment jouer ?

Vous disposez de 10 voitures pour effectuer 15 tours de 6 km, un final de 10 km vous permettra ensuite de prouver que vous êtes vraiment le meilleur !

Bien entendu, le jeu se charge de faire accélérer votre voiture au fil des tours... Le seul moyen de ralentir reste l'accident (toujours brutal) ou la victoire finale !

Pour jouer, utilisez les touches suivantes :

- [α] pour monter (vers la gauche par rapport au déplacement),
- [↰] pour descendre (vers la gauche par rapport au déplacement),
- [8] pour quitter le jeu,
- [÷] pour interrompre temporairement une partie,
- [ON] pour reprendre la partie interrompue avec [÷],
- [←] et [→] servent à activer ou à désactiver le son.

Attention ! Le seul moyen de quitter le jeu consiste à appuyer sur la touche [8].

En début de partie, le son, ou l'absence de son, est fonction du flag beep RPL. Cependant, les touches [←] et [→] permettent d'activer ou de désactiver le son en cours de partie.

Quelques astuces :

- anticipez vos déplacements,
- passez sur 16 pompes à essence et gagner ainsi une vie (en fait, une voiture !) supplémentaire,

- utilisez bien les flaques d'huiles en les prenant par le côté,
- maintenez enfoncée la touche [α] ou la touche [↰] pour vous déplacer, cette technique est généralement plus efficace que des pressions successives sur les touches [α] et [↰].

5
Retrouver les fichiers sur la disquette

Les chemins d'accès aux différents fichiers relatifs aux variables associées à ce jeu sont :

- 'SETH48S', DUNOD_1\S_SX\OB_P_OBS\SETH_S\SETH48S,
- 'CODE', DUNOD_1\S_SX\OB_P_OBS\SETH_S\CODE,
- 'RPL', DUNOD_1\S_SX\OB_P_OBS\SETH_S\RPL,
- 'R0', DUNOD_1\S_SX\OB_P_OBS\SETH_S\R0,
- 'R1', DUNOD_1\S_SX\OB_P_OBS\SETH_S\R1,
- 'R2', DUNOD_1\S_SX\OB_P_OBS\SETH_S\R2,
- 'R3', DUNOD_1\S_SX\OB_P_OBS\SETH_S\R3,
- 'R4', DUNOD_1\S_SX\OB_P_OBS\SETH_S\R4,
- 'ASS', DUNOD_1\S_SX\ASS.

32

Alpha pour HP-48 G et GX

1
Le jeu

1.1
Présentation

ALPHA pour HP-48 G/GX est le seul de nos jeux s'exécutant en utilisant un unique objet qu'est son code exécutable. ALPHA est un jeu de réflexion contre la montre. Le but est la reconstitution de 400 carrés à partir de formes complémentaires. Ainsi, deux triangles complémentaires ou un sablier associé à un nœud papillon reconstituent un carré. Seules neuf cases de la surface de jeu qui en comporte 256 sont visibles.

Vous jouez en associant le contenu de la case centrale au contenu de l'une des huit cases qui l'entourent. Lorsqu'une l'association est impossible, la case centrale se déplace sur la surface de jeu pour devenir la case avec laquelle vous avez tenté d'associer l'ancienne case centrale. Par contre, lorsqu'une tentative d'association aboutit, un carré se forme et une case est libérée. Une seconde couche de case est ainsi rendue visible. Il vous faudra ensuite associer les pièces de la première couche à celle de la deuxième couche. Pour cela, vous ne pouvez qu'utiliser un carré préalablement formé, et ce, pour qu'il échange sa position avec l'une des pièces à associer puisque les associations doivent se faire sur un même niveau.

1.2
Liste des objets

Trois objets sont associés à la mise en œuvre de ce jeu :

- 'ALPHA48G' est la chaîne de codes hexadécimaux permettant de générer 'CODE',
- 'CODE' est le code exécutable généré en traitant la chaîne 'ALPHA48G' à l'aide de 'ASS',
- 'ASS' est le programme RPL permettant de générer l'objet 'CODE' à partir de 'ALPHA48G'.

En fait, seul 'CODE' est nécessaire à l'exécution du programme. C'est en évaluant 'CODE' que l'on exécute le programme.

2
Les objets

2.1
'ALPHA48G' et 'ASS' génèrent 'CODE'

Ci-dessous le code hexadécimal à saisir dans une chaîne de caractères, autrement dit, entre les délimiteurs " et ". Cette chaîne de caractères doit être enregistrée dans une variable nommée 'ALPHA48G'.

Cette chaîne ne doit contenir que les chiffres 0 à 9 et les lettres A à F. Attention ! Il ne faut surtout pas saisir les espaces et les sauts de ligne qui ne sont là que pour faciliter la lecture.

```
CCD20 0E700 84F80 8F082 01BC0 100D2
15C01 9E215 C0185 CE15C 0D1D3 81823
31F00 40816 F14AC AD680 84280 85314
9170B 6555E 34000 02145 1E058 014F8
08B35 08501 ED860 14310 1D231 B8CA1
```

```
00111 130AF 03178 15071 6FA6E 55F11
01303 3F040 06A69 13D34 5A500 81B4C
A842D 215F0 808B3 50852 80883 808A2
73F2C 6CA13 11C73 0E143 15831 73808
B3501 C716F 16F16 1A0E5 FD4D4 84380
8B0F0 06318 3CA07 85380 8A180 31C3C
A1313 01143 15831 73863 501C7 16F16
F161B 0651E 86254 13A33 3420E A1383
12125 15A5A 74B9C 81C15 8516F 16F16
1A6E5 2E201 3A323 40EA1 3813A 338F1
0EA13 807B6 6DA30 2906C 28186 8F818
68231 FD962 D1D60 63269 213AC A1380
7DAD6 6ABE1 1033B 850CA 138DB DA302
76437 CA232 E008F CEE10 93A50 5AE74
92328 008FC EE108 08A49 21B00 100D2
15C03 20048 FCEE1 0808A FFE30 815C0
32400 8FCEE 10808 A4806 AE400 32040
8FCEE 10808 B0F08 08A2B 08405 50850
D032E 008FC EE108 08A17 0E459 1808A
370CC 4D080 8B260 685F3 22008 FCEE1
090A8 08186 8F328 008FC EE109 0A808
1860F 968BC 84434 00408 AE913 4A621
3514B 30C90 68085 45123 04906 D1854
14A80 84330 59026 067CE 6FD01 4E808
43808 83906 01854 30590 2C25E DCA30
29022 13049 02A03 0D906 4C05C F0456
06AC3 1321B 83100 14E13 08168 08828
08831 4A808 73E03 04A46 45030 C15C0
30586 4C0E6 A4644 0E614 B8086 37080
89315 D0860 33307 06D0D 22880 F2801
AE681 86255 9F818 60156 E0720 A0E54
D4C31 4A14F C4C6A 8E81C 81E15 8015D
0874C 18084 33059 02F31 4B808 43902
23137 135D5 860B4 D0223 08801 D2801
B665C F8186 0458E 20482 86032 D0D22
880F2 801AE 6B665 CF818 60159 E206B
CB1BB 31001 E0050 14A80 86080 8415D
38718 38511 4B05A 0C041 59054 21331
31808 60903 0515D 01708 08623 D6F42
11034 6D100 CA130 1D001 4B301 71201
801E2 05014 B3017 E0018 01E40 5014B
D2102 06D23 1D481 B4CAD 281A0 1ACAC
AF2CA 13130 814B1 4816F 16F16 1171A
0E5AE 13A33 4310E A1381 12F40 7A0E5
```

```
8A01E 11212 12121 21212 E180C 08080
80808 080C1 E1121 20180 40201 0F3E1
12020 2C102 0212E 10181 41211 1F301
0101F 31010 F1020 20212 E1E11 21010
F1121 212E1 F3020 20180 40404 040E1
12121 2E112 1212E 1E112 1212E 30202
12E11 00030 00700 0F000 7100D 200D7
005B0 0D610 DF305 B50D6 B0DFF 130C3
FFF7F FF3E1 06CFF 586B5 0D650 EF504
B5086 500F5 00A50 04700 87000 70006
00040 00000 00000 00000 00000 00000
00000 0FFF7 5145B FF650 05700 75005
50055 00530 06700 7D085 B1C67 36756
353D5 67B67 FFF7B DD66B 63C08 18DD0
0B600 6300C 10850 45084 032FF 78FCE
E1093 E1F32 FF78F CEE10 93A80 D281B
31113 17813 01527 86080 A74B2 4BFC1
50716 FA6E5 3E340 00608 60803 30002
CE5DF 48A
```

Cette chaîne est à placer dans une variable nommée 'ALPHA48G'. Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à # 5C1Bh,
- au niveau 1, une longueur égale à 2023 octets.

La chaîne de codes hexadécimaux 'ALPHA48G' est à placer au niveau 1 de la pile. Activez le programme 'ASS', vous obtenez alors au niveau 1 de la pile un objet code exécutable à sauvegarder dans une variable 'CODE'. Activez la commande BYTES en donnant pour argument à cette dernière une copie du code exécutable. Vous devez obtenir :

- au niveau 2, une somme de contrôle égale à # 459Ch,
- au niveau 1, une longueur égale à 1010,5 octets.

2.2

Le code hexadécimal commenté

Les pages qui suivent vous présentent commande par commande le programme ALPHA pour HP-48 G et GX. Rappelons que la première colonne comporte des adresses utilisées pour les sauts relatifs. La dernière colonne renvoie à la remarque de bas de page.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
CCD 20	$$			❶
#000h	CCD20			
	0E700			
	84F	ST=0	15	❷
	808F	INTOFF		
	08	CLRST		
	20	P=	0	
	1BC0100	D0=(5)	#0010C	❸
	D2	C=0	A	
	15C0	DAT0=C	1	
	19E2	D0=(2)	#2E	
	15C0	DAT0=C	1	
	185	D0=D0-	#6	
	CE	C=C-1	A	❹
	15C0	DAT0=C	1	
	D1	B=0	A	
	D3	D=0	A	
	818233	D=D+#4	XS	❺
	1F00408	D1=(5)	#80400	❻
	16F	D0=D0+	#10	
%1				❼
#042h	14A	A=DAT0	B	
	CA	A=A+C	A	
	D6	C=A	A	❽
	80842	ABIT=0	2	
	80853	ABIT=1	3	
	149	DAT1=A	B	❾
	170	D1=D1+	#1	
	B65	B=B+1	B	
	55E	GONC	%1	❿

❶ Procédure $$ (Initialisation).
❷ Inhibition des interruptions. ST0=0 (pour le son).
❸ Suppresion du témoin « sablier ». Encore moins d'interruption.
❹ Suppresion des menus en bas de l'écran. B compteur, puis case 0.
❺ D compte les carrés (encore 400 !).
❻ On pointe sur le début du tableau puis sur l'horloge.
❼ Point de branchement %1. Lecture de l'horloge.
❽ Plus de hasard, que des triangles (bit 2), sur la 1ère couche (bit 3).
❾ Stocke la case en mémoire. Case suivante
❿ Boucle sur 256 cases (tableau 16×16 cases).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	3400002	C=(5)	#20000	
	145	DAT1=C	A	❶
	1E0580	D1=(4)	#0850	
	14F	C=DAT1	B	
	808B3	?CBIT=1	3	❷
	50	GOYES	%2	
	850	ST=1	0	
%2				❸
#07Ch	1ED860	D1=(4)	#068D	
	143	A=DAT1	A	❹
	101	R1=A		
	D2	C=0	A	
	31B8	C=(2)	#8B	❺
	CA	A=A+C	A	
	100	R0=A		
$0				❻
#093h	111	A=R1		
	130	D0=A		❼
	AF0	A=0	W	
	3178	C=(2)	#87	
%1				❽
#0A0	1507	DAT0=A	W	
	16F	D0=D0+	#10	
	A6E	C=C-1	B	
	55F	GONC	%1	
#0ADh	110	A=R0		❾
	130	D0=A		
	33F040	C=(4)	#040F	❿

❶ Mémorisation du chronomètre.
❷ Test : flag bip RPL ? Si le flag est actif, on a ST0 = 1 (flag son).
❸ Point de branchement %2.
❹ Lecture de l'adresse de la bitmap écran (début de l'écran dans R1).
❺ On pointe sur 3ème quartet de la 4ème ligne (1er sprite dans R0).
❻ Procédure $0 (affichage du zoom).
❼ D0 pointe sur le début de l'écran. On efface #880h quartets.
❽ Point de branchement %1. On efface tout l'écran.
❾ D0 pointe sur la première case de l'écran.
❿ Premier sprite dans le tableau.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	%2			❶
#0B9h	06	RSTK=C		
	A69	C=C+B	B	
	13D	D1=CS		
	345A500	C=(5)	#005A5	
	81B4	A=PC		❷
#0CCh	CA	A=A+C	A	
	842	ST=0	2	❸
	D2	C=0	A	
	15F0	C=DAT1	1	
	808B3	?CBIT=1	3	❹
	50	GOYES	%3	
	852	ST=1	2	❺
%3				❻
#0E1h	80883	CBIT=0	3	
	808A2	?CBIT=0	2	
	73	GOYES	%6	❼
#0EDh	F2	CSL	A	
	C6	C=C+C	A	❽
	CA	A=A+C	A	
	131	D1=A		
	1C7	D1=D1-	#8	
	30E	C=(1)	#E	
%4				❾
#0FCh	143	A=DAT1	A	
	1583	DAT0=A	4	
	173	D1=D1+	#4	
	808B3	?CBIT=1	3	
	50	GOYES	%5	❿

❶ Point de branchement %2. Sauvegarde du compteur.
❷ On pointe sur $SPRITE après calcul de la position dans le tableau.
❸ Flag ST2 (détermine la couche, inversion vidéo si c'est la 2ème couche).
❹ Lecture de la case à afficher.
❺ Test : première couche ?
❻ Point de branchement %3. Annulation du bit déterminant la couche.
❼ Test : est-ce un triangle ? En fait, on teste le bit 2.
❽ Numéro du sprite × #20h. On pointe sur les données sprite.
❾ Point de branchement %4. Affichage d'une ligne du sprite.
❿ Décompactage du sprite selon le compteur.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1C7	D1=D1-	#8	
%5				❶
#110h	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	
	A0E	C=C-1	P	❷
	5FD	GONC	%4	
	4D4	GOC	%11	
%6				❸
#122h	843	ST=0	3	
	808B0	?CBIT=1	0	
	F0	GOYES	%7	
	06	RSTK=C		
	3183	C=(2)	#38	❹
	CA	A=A+C	A	❺
	07	C=RSTK		
	853	ST=1	3	
%7				❻
#139h	808A1	?CBIT=0	1	
	80	GOYES	%8	
	31C3	C=(2)	#3C	
	CA	A=A+C	A	
%8				❼
#146h	131	D1=A		
	301	C=(1)	#1	❽
%9				❾
#14Ch	143	A=DAT1	A	
	1583	DAT0=A	4	❿
	173	D1=D1+	#4	

❶ Point de branchement %5. Ligne suivante.
❷ Boucle sur 15 lignes.
❸ Point de branchement %6. Incrémentation par défaut.
❹ Réajustement selon le triangle.
❺ Décompactage des données (ST3 vaut 0 ou 1 selon le triangle).
❻ Point de branchement %7. On pointe sur les données convenables.
❼ Point de branchement %8.
❽ Compteur de lignes.
❾ Point de branchement %9. Lecture des données.
❿ Affichage d'une ligne du sprite puis donnée suivante.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	863	?ST=0=	3	
	50	GOYES	%10	
	1C7	D1=D1-	#8	
%10				❶
#15Eh	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	
	B06	C=C+1	P	❷
	51E	GONC	%9	
%11				❸
#16Dh	862	?ST=0	2	
	54	GOYES	%13	
	13A	AD0xs		❹
	333420	C=(4)	#0243	
	EA	A=A-C	A	
	138	D0=AS		
	3121	C=(2)	#12	❺
	25	P=	5	
%12				❻
#186h	15A5	A=DAT0	6	
	A74	A=A+A	W	
	B9C	A=-A-1	WP	❼
	81C	ASRB		
	1585	DAT0=A	6	❽
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❾
	161	D0=D0+	#2	
	A6E	C=C-1	B	❿
	52E	GONC	%12	

❶ Point de branchement %10. Passage à la ligne suivante.
❷ Boucle sur 15 lignes.
❸ Point de branchement %11. Test : est-ce la deuxième couche ?
❹ On pointe en haut du sprite à l'écran (moins deux lignes et un quartet).
❺ Compteur de lignes.
❻ Point de branchement %12. Lecture de l'écran.
❼ Inversion vidéo sur 23 bits.
❽ Ecriture sur l'écran.
❾ Passage à la ligne du dessous.
❿ Boucle sur 19 lignes (2 + 15 + 2).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	20	P=	0	
	13A	AD0xs		
	32340	C=(3)	#043	❶
	EA	A=A-C	A	
	138	D0=AS		
%13				❷
#1B5h	13A	AD0xs		
	338F10	C=(4)	#01F8	❸
	EA	A=A-C	A	
	138	D0=AS		
	07	C=RSTK		
	B66	C=C+1	B	❹
	DA	A=C	A	
	302	C=(1)	#2	
	906	?A≠C	P	❺
	C2	GOYES	%14	
	81868F	A=A-#10	B	❻
	818682	A=A-#3	B	
	31FD	C=(2)	#DF	
	962	?A=C	B	❼
	D1	GOYES	$1	❽
	D6	C=A	A	
	06	RSTK=C		
	32692	C=(3)	#296	❾
	13A	AD0xs		
	CA	A=A+C	A	
	138	D0=AS		
	07	C=RSTK		
	DA	A=C	A	

❶ On pointe en fin du sprite (en phase avec la routine de 1ère couche).
❷ Point de branchement %13.
❸ On pointe sur le sprite suivant de la même ligne.
❹ Sprite suivant dans le tableau.
❺ Test : fin de ligne en vue ? Il y a 16 lignes de 16 colonnes.
❻ Si le test est positif, on passe à la ligne suivante dans le tableau.
❼ Test : est-on hors de l'écran ?
❽ Si le test est positif, l'affichage est achevé.
❾ Si le test est négatif, on ajoute #0296h pour pointer sur la ligne suivante à l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%14				❶
#1FCh	D6	C=A	A	
	6ABE	GOTO	%2	
$1				❷
#202h	110	A=R0		
	33B850	C=(4)	#058B	
	CA	A=A+C	A	❸
	138	D0=AS		
	DB	C=D	A	❹
	DA	A=C	A	
	302	C=(1)	#2	
	7643	GOSUB	$AFF	❺
$2				❻
#21Bh	7CA2	GOSUB	$CHRONO	
#21Fh	32E00	C=(3)	#00E	
	8FCEE10	GOSBVL	#01EEC	❼
	93A	?C=0	X	
	50	GOYES	$3	
	5AE	GONC	$2	
$3				❽
#233h	7492	GOSUB	$CHRONO	❾
#237h	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	
	808A4	?CBIT=0	4	❿
	92	GOYES	%2	
	1B00100	D0=(5)	#00100	
	D2	C=0	A	

❶ Point de branchement %14. On passe au sprite suivant.
❷ Procédure $1 (triangles suivants).
❸ D0 pointe sur l'écran.
❹ Chargement dans A du nombre de triangles restants.
❺ Affichage du nombre de triangles restants.
❻ Procédure $2 (attente du relâchement). Affichage du chronomètre.
❼ attente du relâchement des touches, [MTH], [SIN], [1], [2], [3], etc.
❽ Procédure $3 (test des touches).
❾ Affichage du chronomètre.
❿ Test : pression sur [SIN] ? Si oui, extinction de l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	15C0	DAT0=C	1	
%1				❶
#257h	32004	C=(3)	#400	
	8FCEE10	GOSBVL	#01EEC	
	808AF	?CBIT=0	15	
	FE	GOYES	%1	❷
	308	C=(1)	#8	
	15C0	DAT0=C	1	
%2				❸
#271h	32400	C=(3)	#004	
	8FCEE10	GOSBVL	#01EEC	
	808A4	?CBIT=0	4	❹
	80	GOYES	%3	
	6AE4	GOTO	$PERDU	
	00	RTNSXM		
%3				❺
#28Ah	32040	C=(3)	#040	
	8FCEE10	GOSBVL	#01EEC	
	808B0	?CBIT=1	0	❻
	F0	GOYES	%4	
	808A2	?CBIT=0	2	❼
	B0	GOYES	%5	
	840	ST=0	0	❽
	550	GONC	%5	
%4				❾
#2AAh	850	ST=1	0	
%5				❿
#2ADh	D0	A=0	A	
	32E00	C=(3)	#00E	

❶ Point de branchement %1. Attente d'une pression sur [ON].
❷ Allumage de l'écran dès qu'une pression sur [ON] est détectée.
❸ Point de branchement %2.
❹ Test : pression sur [MTH] ? Si oui, fin du jeu.
❺ Point de branchement %3.
❻ Test : pression sur [→] ?
❼ Test : pression sur [←] ?
❽ Désactivation du flag bip si une pression sur [→] ou sur [←] est détectée.
❾ Point de branchement %4. Si pression sur [→], activation du flag bip.
❿ Point de branchement %5.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	8FCEE10	GOSBVL	#01EEC	❶
	808A1	?CBIT=0	1	
	70	GOYES	%6	❷
	E4	A=A+1	A	
	591	GONC	%9	
%6				❸
#2C7h	808A3	?CBIT=0	3	
	70	GOYES	%7	
	CC	A=A-1	A	
	4D0	GOC	%9	
%7				❹
#2D3h	808B2	?CBIT=1	2	
	60	GOYES	%9	
%8				❺
#2DAh	685F	GOTO	$3	
%9				❻
#2DEh	32200	C=(3)	#002	
	8FCEE10	GOSBVL	#01EEC	
	90A	?C=0	P	❼
	80	GOYES	%10	
	81868F	A=A-#10	B	
%10				❽
#2F5h	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	
	90A	?C=0	P	❾
	80	GOYES	%11	
	81860F	A=A+#10	B	
%11				❿
#30Ch	968	?A=0	B	
	BC	GOYES	%8	

❶ On teste le pavé numérique.
❷ A droite ?
❸ Point de branchement %6. A gauche ?
❹ Point de branchement %7. Pas de déplacement latéral ?
❺ Point de branchement %8. Une pression était-elle détectée ?
❻ Point de branchement %9.
❼ En bas ?
❽ Point de branchement %10.
❾ En haut ?
❿ Point de branchement %11. Annulation en cas de pression sur [5].

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
CCD 20				
$4				❶
#311h	844	ST=0	4	❷
	3400408	C=(5)	#80400	
	AE9	C=B	B	
	134	D0=C		❸
	A62	C=C+A	B	
	135	D1=C		
#327h	14B	A=DAT1	B	❹
	30C	C=(1)	#C	
	906	?A≠C	P	
	80	GOYES	%1	
	854	ST=1	4	❺
	512	GONC	%2	
%1				❻
#338h	304	C=(1)	#4	
	906	?A≠C	P	
	D1	GOYES	%3	
	854	ST=1	4	❼
	14A	A=DAT0	B	❽
	80843	ABIT=0	3	
	305	C=(1)	#5	
	902	?A=C	P	❾
	60	GOYES	%2	
	67CE	GOTO	$2	
%2				❿
#357h	6FD0	GOTO	%14	

❶ Procédure $4 (gestion du coup).
❷ Pas de déplacement et carré au centre en cas d'assemblage.
❸ D0 contient la case courante et D1, la case courante.
❹ Contenu de la case d'arrivée. Test : est-ce un blanc (bit 3 = 1) ?
❺ Le flag ST4 indique le déplacement (pour %14).
❻ Point de branchement %1. Est-ce un noir ?
❼ ST4 indique le déplacement.
❽ Contenu de la case de départ (sans tenir compte de la couche).
❾ La case de départ est-elle un carré ? Si non, coup interdit avec case noire.
❿ Point de branchement %2. Déplacement si le test est positif.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	%3			❶
	#35Bh	14E	C=DAT0	B	
		80843	ABIT=0	3	
		80883	CBIT=0	3	
		906	?A≠C	P	
		01	GOYES	%4	❷
		854	ST=1	4	
		305	C=(1)	#5	
		902	?A=C	P	❸
		C2	GOYES	%6	
		5ED	GONC	%2	
	%4				❹
	#37Bh	CA	A=A+C	A	
		302	C=(1)	#2	
		902	?A=C	P	❺
		21	GOYES	%5	
		304	C=(1)	#4	
		902	?A=C	P	
		A0	GOYES	%5	❻
		30D	C=(1)	#D	
		906	?A≠C	P	
		4C	GOYES	%2	
	%5				❼
	#395h	05	SETDEC		
		CF	D=D-1	A	❽
		04	SETHEX		
		560	GONC	%5	
		6AC3	GOTO	$GAGNE	❾
	%6				❿
	#3A2h	132	AD0EX		

❶ Point de branchement %3. Contenu de la case de départ.
❷ Test : identité départ/arrivée ? Oui, alors déplacement.
❸ Assemblage s'il y a deux carrés, déplacement dans le cas contraire.
❹ Point de branchement %4.
❺ Test : s'agit-il d'un nœud papillon et d'un sablier complémentaires ?
❻ Test : s'agit-il de deux triangles complémentaires.
❼ Point de branchement %5.
❽ Dans les deux cas un carré apparaît (jusqu'au 400ème).
❾ Fin du jeu (et victoire) si c'est le 400ème carré.
❿ Point de branchement %6.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1B83100	D0=(5)	#00138	
	14E	C=DAT0	B	❶
	130	D0=A		
	816	CSRC		❷
	80882	CBIT=0	2	❸
	80883	CBIT=0	3	
	14A	A=DAT0	B	
	80873	?ABIT=1	3	❹
	E0	GOYES	%7	
	304	C=(1)	#4	❺
	A46	C=C+C	S	
	450	GOC	%7	
	30C	C=(1)	#C	
%7				❻
#3D5h	15C0	DAT0=C	1	
	305	C=(1)	#5	❼
	864	?ST=0	4	
	C0	GOYES	%8	
	E6	C=C+1	A	
	A46	C=C+C	S	❽
	440	GOC	%8	
	E6	C=C+1	A	
%8				❾
#3EBh	14B	A=DAT1	B	
	80863	?ABIT=0	3	
	70	GOYES	%9	
	80893	CBIT=1	3	
%9				❿
#3FAh	15D0	DAT1=C	1	
	860	?ST=0	0	

❶ Lecture de l'horloge.
❷ Sauvegarde d'un flag aléatoire (selon l'horloge).
❸ On veut un triangle sur la deuxième couche.
❹ Test : était-on déjà sur la deuxième couche ?
❺ Si oui, on stocke un blanc ou un noir selon l'horloge (flag aléatoire).
❻ Point de branchement %7. Modification de la case de départ.
❼ La case d'arrivée est un carré (sauf si assemblage de deux carrés).
❽ Sinon, un nœud papillon ou un sablier pris aléatoirement.
❾ Point de branchement %8. Lecture de la case d'arrivée.
❿ Point de branchement %9. Modification de la case d'arrivée.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	33	GOYES	%13	❶
	307	C=(1)	#7	
%10				❷
#406h	06	RSTK=C		
	D0	A=0	A	
	D2	C=0	A	
	28	P=	8	
%11				❸
#40Eh	80F2	CPex	2	
	801	OUT=C		
	AE6	C=A	B	
%12				❹
#418h	818625	C=C+#6	B	
	59F	GONC	%12	
	818601	A=A+#2	B	❺
	56E	GONC	%11	
	07	C=RSTK		
	20	P=	0	
	A0E	C=C-1	P	❻
	54D	GONC	%10	
%13				❼
#434h	4C3	GOC	%15	
%14				❽
#437h	14A	A=DAT0	B	
	14F	C=DAT1	B	❾
	C4	A=A+A	A	
	C6	C=C+C	A	❿
	A8E	ACex	P	
	81C	ASRB		

❶ Test : le son est-il autorisé ? Sept « tics ».
❷ Point de branchement %10. Durée du son dans A.
❸ Point de branchement %11. Bip !
❹ Point de branchement %12. Temporisation fréquence.
❺ Boucle durée.
❻ Tic suivant.
❼ Point de branchement %13. Pointeur sur case courante.
❽ Point de branchement %14.
❾ Inversion entre les cases de départ et d'arrivée.
❿ On conserve la couche malgré l'inversion des cases.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	81E	CSRB		
	1580	DAT0=A	1	
	15D0	DAT1=C	1	
	874	?ST=1	4	❶
	C1	GOYES	%15	❷
	80843	ABIT=0	3	
	305	C=(1)	#5	
	902	?A=C	P	❸
	F3	GOYES	%18	
	14B	A=DAT1	B	
	80843	ABIT=0	3	
	902	?A=C	P	
	23	GOYES	%18	
%15				❹
#471h	137	CD1ex		
	135	D1=C		
	D5	B=C	A	
	860	?ST=0	0	❺
	B4	GOYES	%21	
	D0	A=0	A	❻
	22	P=	2	
%16				❼
#482h	308	C=(1)	#8	
	801	OUT=C		❽
	D2	C=0	A	
	801	OUT=C		
%17				❾
#48Dh	B66	C=C+1	B	
	5CF	GONC	%17	
	818604	A=A+#5	B	❿

❶ Test : est-ce un déplacement ?
❷ Si oui, on modifie le pointeur désignant la case centrale (case active).
❸ Inversion, si ce n'est pas un déplacement et s'il s'agit d'un carré.
❹ Point de branchement %15. B contient le pointeur sur la case active.
❺ Test : le son est-il autorisé ?
❻ Durée dans A.
❼ Point de branchement %16.
❽ Bip avec volume moyen.
❾ Point de branchement %17. Temporisation fréquence.
❿ Boucle durée.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	58E	GONC	%16	
	20	P=	0	❶
	482	GOC	%21	
%18				❷
#4A1h	860	?ST=0	0	
	32	GOYES	%21	❸
	D0	A=0	A	
	D2	C=0	A	
	28	P=	8	
%19				❹
#4ACh	80F2	CPex	2	
	801	OUT=C		
	AE6	C=A	B	
%20				❺
#4B6h	B66	C=C+1	B	
	5CF	GONC	%20	
	818601	A=A+#2	B	❻
	59E	GONC	%19	
	20	P=	0	
%21				❼
#4C7h	6BCB	GOTO	$0	
#4CBh				
$CHRONO				❽
#000h	1BB3100	D0=(5)	#0013B	
	1E0050	D1=(4)	#0500	❾
14A	A=DAT0	B		❿
	80860	?ABIT=0	0	

❶ Affichage zoom. Un autre son est émis si ce n'est qu'une inversion.
❷ Point de branchement %18.
❸ Test : le son est-il autorisé ? Durée dans A.
❹ Point de branchement %19. Bip !
❺ Point de branchement %20. Temporisation fréquence.
❻ Boucle durée.
❼ Point de branchement %21. Affichage zoom.
❽ Procédure $CHRONO. Le compteur d'adresses repart à #000h.
❾ D1 pointe sur le chronomètre.
❿ Lecture de l'horloge (1/2 seconde).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	80	GOYES	%1	❶
	841	ST=0	1	
	5D3	GONC	%4	
%1				❷
#01Dh	871	?ST=1	1	
	83	GOYES	%4	
	851	ST=1	1	
%2				❸
#025h	14B	A=DAT1	B	
	05	SETDEC		
	A0C	A=A-1	P	❹
	04	SETHEX		
	1590	DAT1=A	1	❺
	542	GONC	%4	
	133	AD1ex		
	131	D1=A		
	80860	?ABIT=0	0	❻
	90	GOYES	%3	
	305	C=(1)	#5	
	15D0	DAT1=C	1	
%3				❼
#04Ah	170	D1=D1+	#1	
	80862	?ABIT=0	2	
	3D	GOYES	%2	
	6F42	GOTO	$PERDU	❽
%4				❾
#058h	110	A=R0		
	346D100	C=(5)	#001D6	
	CA	A=A+C	A	❿
	130	D0=A		

❶ On teste la parité toute les deux secondes. Attente si imparité.
❷ Point de branchement %1. Test : est-ce un changement.
❸ Point de branchement %2. On modifie le chronomètre.
❹ Défilement chiffre par chiffre.
❺ S'il n'y a pas de retenue, c'est fini.
❻ Test : est-ce une dizaine ? On fixe le maximum à 59 et non à 99 ?
❼ Point de branchement %3. Modification des heures.
❽ Fin du jeu au bout de deux heures.
❾ Point de branchement %4.
❿ D0 pointe sur l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1D00	D1=(2)	#00	❶
	14B	A=DAT1	B	
	301	C=(1)	#1	
	7120	GOSUB	$AFF	❷
#075h	180	D0=D0-	#1	
	1E2050	D1=(4)	#0502	❸
	14B	A=DAT1	B	
	301	C=(1)	#1	
	7E00	GOSUB	$AFF	
#088h	180	D0=D0-	#1	
	1E4050	D1=(4)	#0504	❹
	14B	A=DAT1	B	
	D2	C=0	A	
$AFF				❺
%1				❻
#096h	102	R2=A		
	06	RSTK=C		❼
	D2	C=0	A	
	31D4	C=(2)	#4D	
	81B4	A=PC		
#0A5h	CA	A=A+C	A	❽
	D2	C=0	A	
	81A01A	C=R2	P	
	CA	A=A+C	A	
	CA	A=A+C	A	❾
	F2	CSL	A	
	CA	A=A+C	A	
	131	D1=A		
	308	C=(1)	#8	❿

❶ On pointe sur les secondes (deux chiffres).
❷ Affichage du nombres de secondes.
❸ On pointe sur les minutes (deux chiffres) et on affiche ce nombre.
❹ On pointe sur les heures (un chiffre).
❺ Procédure $AFF (commence en #096h).
❻ Point de branchement %1. R2 sauvegarde les quartets.
❼ Le compteur de chiffres est sauvé dans la pile RSTK.
❽ On pointe sur les données « chiffres ».
❾ A = A + (C × #18h)
❿ Compteur de lignes.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%2				❶
#0BDh	14B	A=DAT1	B	
	148	DAT0=A	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❷
	161	D0=D0+	#2	
	171	D1=D1+	#2	❸
	A0E	C=C-1	P	
	5AE	GONC	%2	❹
	13A	ADOxs		
	334310	C=(4)	#0134	❺
	EA	A=A-C	A	
	138	D0=AS		
	112	A=R2		
	F4	ASR	A	❻
	07	C=RSTK		
	A0E	C=C-1	P	❼
	58A	GONC	%1	
	01	RTN		
$CHIFFRES				❽
#0F2h	E1121	*Données graphiques*		
	21212	*Données graphiques*		
	12121	*Données graphiques*		
	2E180	*Données graphiques*		
	C0808	*Données graphiques*		
	08080	*Données graphiques*		
	8080C	*Données graphiques*		
	1E112	*Données graphiques*		
	12018	*Données graphiques*		
	04020	*Données graphiques*		
	10F3E	*Données graphiques*		
	11202	*Données graphiques*		

❶ Point de branchement %2. Transfert des données vers l'écran.
❷ Passage à la ligne du dessous.
❸ Donnée suivante.
❹ Boucle sur 9 lignes.
❺ Chiffre suivant sur l'écran.
❻ Chiffre suivant.
❼ Boucle sur le chiffres.
❽ Procédure $CHIFFRES (codage des sprites chiffres).

Adresses	Codes hexadécimaux	Mnémoniques	Renvois
	02C10	*Données graphiques*	
	20212	*Données graphiques*	
	E1018	*Données graphiques*	
	14121	*Données graphiques*	
	11F30	*Données graphiques*	
	10101	*Données graphiques*	
	F3101	*Données graphiques*	
	0F102	*Données graphiques*	
	02021	*Données graphiques*	
	2E1E1	*Données graphiques*	
	12101	*Données graphiques*	
	0F112	*Données graphiques*	
	1212E	*Données graphiques*	
	1F302	*Données graphiques*	
	02018	*Données graphiques*	
	04040	*Données graphiques*	
	4040E	*Données graphiques*	
	11212	*Données graphiques*	
	12E11	*Données graphiques*	
	21212	*Données graphiques*	
	E1E11	*Données graphiques*	
	21212	*Données graphiques*	
	E3020	*Données graphiques*	
	212E1	*Données graphiques*	
$SPRITES			❶
#1A6h	10003	*Données graphiques*	
	00070	*Données graphiques*	
	00F00	*Données graphiques*	
	07100	*Données graphiques*	
	D200D	*Données graphiques*	
	7005B	*Données graphiques*	
	00D61	*Données graphiques*	
	0DF30	*Données graphiques*	
	5B50D	*Données graphiques*	
	6B0DF	*Données graphiques*	
	F130C	*Données graphiques*	
	3FFF7	*Données graphiques*	
	FFF3E	*Données graphiques*	
	106CF	*Données graphiques*	
	F586B	*Données graphiques*	

❶ Procédure $SPRITES. Codage des différentes figures (triangles, etc.).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	50D65	*Données graphiques*		
	0EF50	*Données graphiques*		
	4B508	*Données graphiques*		
	6500F	*Données graphiques*		
	500A5	*Données graphiques*		
	00470	*Données graphiques*		
	08700	*Données graphiques*		
	07000	*Données graphiques*		
	60004	*Données graphiques*		
#21E	00000	*Données graphiques*		❶
	00000	*Données graphiques*		
	00000	*Données graphiques*		
	00000	*Données graphiques*		
	00000	*Données graphiques*		
	00000	*Données graphiques*		
	00FFF	*Données graphiques*		
	75145	*Données graphiques*		
	BFF65	*Données graphiques*		
	00570	*Données graphiques*		
	07500	*Données graphiques*		
	55005	*Données graphiques*		
	50053	*Données graphiques*		
	00670	*Données graphiques*		
	07D08	*Données graphiques*		
	5B1C6	*Données graphiques*		
	73675	*Données graphiques*		
	6353D	*Données graphiques*		
	567B6	*Données graphiques*		
	7FFF7	*Données graphiques*		
	BDD66	*Données graphiques*		
	B63C0	*Données graphiques*		
	818DD	*Données graphiques*		
	00B60	*Données graphiques*		
	06300	*Données graphiques*		
	C10	*Données graphiques*		
$GAGNE				❷
#29Eh	850	ST=1	0	
	450	GOC	$FIN	

❶ Codage du « blanc » (pièce nulle) puis des carrés.
❷ Procédure $GAGNE. On commence par effacer l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$PERDU				❶
#294h	840	ST=0	0	
$FIN				❷
%1				❸
#2A7h	32FF7	C=(3)	#7FF	
	8FCEE10	GOSBVL	#01EEC	
	93E	?C≠0	X	
	1F	GOYES	%1	
%2				❹
#2B8h	32FF7	C=(3)	#7FF	
	8FCEE10	GOSBVL	#01EEC	
	93A	?C=0	X	
	80	GOYES	%3	
	D2	C=0	A	
	81B3	PC=C		
%3				❺
#2CFh	111	A=R1		
	3178	C=(2)	#87	❻
	130	D0=A		
%4				❼
#2D9h	1527	A=DAT0	W	
	860	?ST=0	0	
	80	GOYES	%5	❽
	A74	A=A+A	W	
	B24	A=A+1	XS	
%5				❾
#2E8h	BFC	A=-A-1	W	
%6				❿
#2EBh	1507	DAT0=A	W	

❶ Procédure $PERDU. Inversion vidéo.
❷ Procédure $FIN.
❸ Point de branchement %1. Attente du relâchement des touches.
❹ Point de branchement %2. Reset si pression sur une touche.
❺ Point de branchement %3.
❻ #880h quartets. D0 pointe sur le début de l'écran.
❼ Point de branchement %4. Lecture de l'écran.
❽ On efface progressivement l'écran si le joueur a gagné.
❾ Point de branchement %5. Inversion vidéo si le joueur a perdu.
❿ Point de branchement %6.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	16F	D0=D0+	#10	❶
	A6E	C=C-1	B	
	53E	GONC	%4	❷
	3400060	C=(5)	#06000	
	860	?ST=0	0	
	80	GOYES	%7	❸
	330002	C=(4)	#2000	
%7				❹
#30Ah	CE	C=C-1	A	
	5DF	GONC	%7	
	48A	GOC	%2	❺
#312h				❻

❶ Quartets suivants.
❷ Boucle les #880h quartets.
❸ Temporisation plus courte en cas de victoire.
❹ Point de branchement %7. Temporisation.
❺ On retourne vers la procédure de test des touches.
❻ Fin du code.

3

Exécuter le programme

La méthode la plus académique pour exécuter le programme consiste à sauvegarder l'objet code exécutable dans une variable 'CODE'. Cela dit, dans le cas du jeu ALPHA, il nous semble intéressant de ne mémoriser que l'objet chaîne de caractères ('ALPHA48G'). On évite ainsi la multiplication des variables et la mobilisation en permanence de la mémoire affectée à la sauvegarde de l'objet code exécutable.

Pour exécuter le programme, on pourra donc procéder de la façon suivante :

- la chaîne de codes hexadécimaux étant enregistrée dans 'ALPHA48G', on la place au niveau 1 de la pile,
- au niveau 1 de la pile apparaît "CCD200E70084F808F...

• lancez le programme 'ASS' qui va prendre la chaîne hexadécimale comme argument,
• au niveau 1 de la pile **"CCD200E70084F808F...** est remplacé par **Code** *,
• Il suffit ensuite d'appuyer sur la touche [EVAL] pour lancer le jeu.

* A ce stade, il est aussi possible de saisir 'CODE' et d'appuyer sur [STO] pour mémoriser le code exécutable dans la variable 'CODE' qu'il vous suffira ensuite d'évaluer pour exécuter le jeu.

4

Comment jouer ?

ALPHA est un jeu d'association de pièces. Il faut assembler deux pièces complémentaires pour créer un carré. Créez 400 carrés en moins de deux heures et vous aurez gagné ! Vous jouez sur une sphère couverte de 256 cases. Seules neuf cases sont visibles à un instant donnée. La case centrale est la case active et pourra interagir avec les huit cases qui l'entourent. Pour faire réagir le contenu de la case centrale avec le contenu de l'une des huit cases qui l'entourent, il suffit d'appuyer sur les touches [1], [2], [3], [4], [6], [7], [8] et [9] du pavé numérique. Ainsi, pour faire réagir le contenu de la case centrale avec celui de la case se trouvant en bas à droite, il suffira d'appuyer sur la touche [3].

Retenez que c'est toujours la case centrale qui réagit avec celles qui l'entoure. Les figures utilisées sont les suivantes : triangles (il en existe de quatre différents), le nœud papillon, le sablier, le carré, la pièce blanche, et la pièce noire. Les triangles, le nœud papillon et le sablier sont les pièces dites « mineures ».

Les différentes réactions possibles entre les figures sont les suivantes :
• l'assemblage de deux triangles complémentaires, dans ce cas, un carré se forme,
• l'assemblage d'un nœud papillon avec un sablier, dans ce cas, un carré se forme,
• la réaction entre deux carrés provoque la création de pièces mineures,

- la réaction entre une pièce mineure et un carré provoque un échange des positions,
- dans tous les autres cas, la case de destination choisie devient la case centrale (on se déplace sur la sphère).

A chaque fois qu'une case vide apparaît (à la suite, par exemple, de la création d'un carré), elle rend visible la seconde couche de figures (en inversion vidéo). Les cases de la première couche ont un fond blanc alors que celles de la seconde couche ont un fond noir.

Seul le carré peut passer de la première couche (fond blanc) à la seconde couche (fond noir) par simple inversion des positions. Ce sont donc vos carrés qu'il faudra exploiter pour faire « monter » sur la première couche ou « descendre » vers la seconde couche les pièces de votre choix (en échangeant la position d'un carré de l'une des couches avec la position d'une pièce mineure de l'autre couche). En résumé, voici les fonctions de chacune des touches utilisées pendant une partie :

- [SIN] provoque une pause,
- [ON] relance une partie interrompue en appuyant sur [SIN],
- [MTH] [MTH] (deux pressions) met fin au jeu. Vous pouvez aussi appuyer sur [MTH] puis sur une touche quelconque,
- [1] inverse la case centrale et la case en bas à gauche,
- [2] inverse la case centrale et la case en bas au centre,
- [3] inverse la case centrale et la case en bas à droite,
- [4] inverse la case centrale et la case à gauche,
- [6] inverse la case centrale et la case à droite,
- [7] inverse la case centrale et la case en haut à gauche,
- [8] inverse la case centrale et la case en haut au centre,
- [9] inverse la case centrale et la case en haut à droite.

5
Retrouver les fichiers sur la disquette

Les chemins d'accès aux différents fichiers sont :

- pour 'ASS', DUNOD_1\G_GX\ASS,
- pour 'ALPHA48G', DUNOD_1\G_GX\OB_P_OBG\ALPHA_G\ALPHA48G,
- pour 'CODE', DUNOD_1\G_GX\OB_P_OBG\ALPHA_G\CODE.

33

Star pour HP-48 G et GX

1
Le jeu

1.1
Présentation

Contrairement à ALPHA qui brille par la simplicité de sa mise en œuvre, STAR nécessite un programme lanceur écrit en RPL. Ce programme, au demeurant très simple, est chargé de la mise en place d'un objet graphique dans la pile.

Les adeptes du jeu sur Macintosh retrouveront avec STAR les joies des Tinies™, célèbre jeu de l'ordinateur à la pomme. En effet, on retrouve avec STAR des principes de jeu assez proches de ceux qui ont fait le succès des Tinies™ : vous déplacez un sprite qui ne peut s'arrêter qu'en rencontrant une paroi. Pour passer au tableau suivant, vous devrez « gober » toutes les pastilles rencontrées... Comme nous l'avons dit, vous êtes obligé de vous déplacer en vous heurtant aux murs, heureusement, vous pouvez déplacer un carré, sorte de « petit mur mobile », pour mieux contrôler vos déplacements. Bien entendu, ce dernier est soumis aux mêmes règles de déplacements que votre sprite gobeur !

Bref, un cocktail Pac-Man™/Tinies™ boulimique et rebondissant !

1.2
Liste des objets

Neuf objets sont associés à la mise en œuvre de ce jeu :

- 'STAR48G' est la chaîne de codes hexadécimaux qui devra être traitée avec le programme ASS pour générer le code exécutable 'CODE',
- 'CODE' est le code exécutable généré à partir de 'STAR48S',
- 'ASS' est le programme permettant de générer 'CODE' à partir de 'STAR48G',
- 'STAR' est le programme lanceur (écrit en RPL),
- 'T1', 'T2', 'T3' et 'T4' sont des variables intermédiaires qui facilitent la saisie de 'TABL',
- 'TABL' est un objet graphique (GROB).

'STAR48G', 'ASS', 'T1', 'T2', 'T3' et 'T4' sont des variables utilisées pour le développement, l'édition et l'introduction de ALPHA mais ne sont absolument pas indispensables à l'exécution du programme.

Seules les variables 'STAR' (lanceur RPL), 'CODE' (code exécutable) et 'TABL' (objet graphique) sont indispensables à l'exécution du programme.

2
Les objets

2.1
Le lanceur RPL

Le listing est un modèle de simplicité :

```
« TABL -6 CODE »
```

Cet objet est à enregistrer dans une variable nommée 'STAR'.

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à #D011h,
- au niveau 1, une longueur égale à 27,5 octets.

2.2
L'objet graphique

L'objet graphique 'TABL' est assez long. Il vous est donc possible de le saisir sous la forme de quatre chaînes de caractères 'T1', 'T2', 'T3' et 'T4' qu'il suffira de concaténer (une concaténation est une juxtaposition de chaînes de caractères). Cette méthode de saisie vous est présentée dans le chapitre présentant la version de STAR destinée aux HP-48 S et SX.

Saisissez l'objet ci-dessous en ne tenant compte que des trois premiers espaces (ces trois espaces à saisir sont signalés par le caractère «_»). Tous les autres espaces ainsi que tous les sauts de ligne doivent être ignorés. Cet objet est à placer dans une variable nommée 'TABL'.

```
GROB_1_791_
B 0F0 A01 FFFF 500B 7008 1C08 30C8 100C 1308
          1038 FFFF 0000 8000 0100 0200 0010
          8021 4402 2800 0000
 9 A01 2B4 B8A4 1008 0011 500A 2020 1058 9821
          2008 4084 0000 2080 8880 0280 0200
          0200 6000 0000 0000
 F A01 8F1 FFFF 5008 D0C8 1088 1608 320A 702B
          1038 FFFF 0000 8000 0000 0140 2802
          4421 8290 0804 0000
 F C96 A01 FFFF 5008 1028 551D BA8A 554D 1028
          9088 FFFF 0000 0000 0000 AAA2 0050
          AAA2 0000 0000 0000
 8 0F0 0B5 EFF7 1988 5029 190C 3049 111C 545A
          5288 EFF7 0000 0000 0980 0000 0084
          0400 0004 2400 0000
 9 AB3 486 FFF0 1101 B002 1004 1008 320C 9008
          1309 FFFF 0000 A010 0010 0010 0010
          0010 0010 8010 0000
 7 886 696 FFFF 1008 D4E6 AA44 CE42 AA46 1008
          5088 FFFF 0000 0000 2200 0022 0000
          4400 0000 2040 0000
```

```
A CA5 895 EFFF 30C8 1819 14D8 591C 900D 5188
          100C FFF7 0000 0001 0280 0201 8000
          4100 0805 0000 0000
D A01 8C3 EFF2 1105 2088 9815 1029 2818 400A
          8008 0FF7 0000 0082 0405 0120 2490
          0020 0005 0000 0000
C 0C4 4E1 EFF0 100F 9838 2428 9028 A42A C934
          A208 9CF7 0000 0040 2500 0212 2400
          0805 0002 0000 0000
E A01 CA5 FFFF 1018 3009 160C 7828 168D 1048
          32CD EFF6 0000 0000 81C0 2040 0614
          0000 2094 0002 0000
A 0F0 EB4 E91C 37EB 1008 513B 992A 513B 1008
          1008 FFFF 0000 4010 8060 8800 0480
          0000 0020 0400 0000
A 2F0 486 EFF7 3009 D02A 5008 5008 100A 500C
          5188 EFF7 0000 8000 0110 0800 2002
          0400 2A00 0400 0000
C 4E1 EC2 FFFF 500A 9109 1008 1208 1008 9009
          500A FFFF 0000 A005 0002 0100 0400
          0010 0002 A005 0000
C 0D2 0D3 EFE7 1018 142A 9888 5015 9408 2004
          1008 EFF7 0000 4020 4080 4600 0400
          4004 0800 2004 0000
F 2E1 4F0 FFFF 108C 3048 30CD 1A0C 1108 1F0C
          102D FFFF 0000 2050 00A4 0200 0410
          8206 8001 0002 0000
F 801 A96 FFFF 9009 542A 1A58 9009 1188 5A5A
          1018 FFFF 0000 0002 0A50 8420 0102
          0241 0400 0840 0000
F 886 201 FFFF 1108 1249 11A8 9018 198A 1229
          902A FFFF 0000 0012 01A0 8040 4180
          0201 0882 0004 0000
B E96 0B5 FFFF D068 904A 1C09 101A 1248 5C2B
          D049 FFFF 0000 2004 4000 8012 0800
          0400 0240 0022 0000
C 2D3 CB3 FFFF 1048 3C2C 528A 1998 514A 343C
          1208 FFFF 0000 00A0 0242 0020 2000
          0400 4240 0500 0000 0
```

Note : chaque groupe de trois lignes correspond à un tableau de jeu.

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez, au niveau 2, une somme de contrôle égale à #A263h, et au niveau 1, une longueur égale à 801 octets.

2.3
Le code hexadécimal

a
La chaîne à saisir

Cet objet est à enregistrer dans une variable nommée 'STAR48G'. La chaîne de codes hexadécimaux est à saisir entre les guillemets " et ", sans espace ni saut de ligne.

```
"CCD20 52400 84F80 8F082 01433 4FE3A
28A2C 01421 64808 C1741 43104 1F058
0814F 808B3 50851 1BB01 00314 814CD
219E2 15C01 ED860 14710 81853 0F15C
0114D 8D231 41C1D 910B1 35118 134AF
03178 15071 6FA6E 55F14 FD717 0118D
01533 CA101 172D0 1533C A1021 72134
30806 13713 5D5AC 2147F 2C65A 40634
BE200 81B4C A1313 07061 43146 0E3E1
50316 F16F1 61172 07A0E 5ED13 23201
1EA13 00716 1B465 CA132 D231E CCA13
0D913 51730 7A0E4 60697 F1181 34308
06AC2 147F2 C65B1 06302 79121 32D23
1CCEA 13007 161B4 65BD1 32D23 1ECCA
13017 307A0 E58B1 37D51 11130 30175
D1112 130D2 79C18 401BB 0100D 215C0
87090 30415 C0320 048FC EE108 08AFB
011BD 5618E D0320 408FC EE108 08A46
081B2 808A0 60E4E 4808A 260CC CC8AC
06808 A1A0D 231EE CA320 808FC EE108
08A1A 0D231 EEEA8 AC138 08A06 28608
08405 50850 DA330 00CCE 5DF31 51962
476C3 F1198 60501 1ADE0 6CAD2 3198C
A1301 4A30A 90660 611F8 08707 F8086
1C487 0BE86 17231 08DA3 20088 01630
0D280 1D6A6 E5CFA 6C53E A0F57 1D913
514B9 0C606 36D6C 6D111 86050 112D2
3144C A1303 03061 4A311 80E66 14816
F16F1 6107A 0E51E 07061 32CAD 231CC
EA130 86080 10255 0101D 28704 0E673"
```

```
10320 12A3E 5CF07 DA640 FD0A8 A1370
634A4 000F0 C281B 4CA13 13050 614B1
4E0E6 E1481 6F16F 16117 107A0 E5FD0
71350 1EF09 213C1 D013A 19415 21EF0
0000C 3C3C3 C3000 00000 81C2C 38100
00000 08142 4281
```

Placez dans la pile une copie de l'objet 'STAR48G', activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à ####h,
- au niveau 1, une longueur égale à #### octets.

Pour générer l'objet 'CODE' à partir de 'STAR48G', procédez de la façon suivante :

- placez la chaîne de codes hexadécimaux (variable 'STAR48S') au niveau 1 de la pile, **"CCD205240084F808F...** s'affiche,
- exécuter le programme ASS,
- au niveau 1 de la pile doit apparaître **Code**,
- placez-vous dans le répertoire DSTAR,
- saisissez le nom **'CODE'**,
- appuyez sur [STO].

Placez dans la pile une copie de l'objet 'CODE', activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à # 2D99h,
- au niveau 1, une longueur égale à 1069 octets.

b *Codes commentés*

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$$				❶
	CCD20			❷
#000h	52400			
	84F	ST=0	15	
	808F	INTOFF		❸
	08	CLRST		
	20	P=	0	

❶ Procédure $$ d'initialisation.
❷ Prologue du code exécutable puis longueur totale hors prologue.
❸ Inhibition des interruptions.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	143	A=DAT1	A	❶
	34FE3A2	C=(5)	#2A3EF	
	8A2	?A=C	A	❷
	C0	GOYES	%1	
	142	A=DAT0	A	❸
	164	DO=DO+	#5	
	808C	PC=(A)		
%1				
#029h	174	D1=D1+	#5	❹
	143	A=DAT1	A	
	104	R4=A		
	1F05808	D1=(5)	#80850	
	14F	C=DAT1	B	❺
	808B3	?CBIT=1	3	
	50	GOYES	%2	
	851	ST=1	1	❻
%2				
#046h	1BB0100	D0=(5)	#0010B	
	3148	C=(2)	#84	❼
	14C	DAT0=C	B	
	D2	C=0	A	
	19E2	D0=(2)	#2E	❽
	15C0	DAT0=C	1	
	1ED860	D1=(4)	#068D	
	147	C=DAT1	A	❾
	108	R0=C		
	185	D0=D0-	#6	
	30F	C=(1)	#F	❿
	15C0	DAT0=C	1	

❶ Le niveau 1 de la pile RPL est lu.
❷ Test : le niveau 1 de la pile RPL contient-il -6 ?
❸ Si le test est négatif, on rend la main au RPL.
❹ Point de branchement %1. R4 désigne le début du GROB tableau.
❺ Test : flag bip RPL ?
❻ On pose ST1 = 1 si le test est positif.
❼ En #046h, point de branchement %2. Extinction du sablier, allumage de α.
❽ On limite encore les interruptions.
❾ Lecture de la bitmap écran. R0 désigne le début de l'écran.
❿ Les menus horizontaux en bas de l'écran sont effacés.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$0				❶
#074h	114	A=R4		
	D8	B=A	A	
	D2	C=0	A	
	3141	C=(2)	#14	❷
	C1	B=B+C	A	
$1				
#081h	D9	C=B	A	❸
	10B	R3=C		
	135	D1=C		❹
	118	C=R0		
	134	D0=C		❺
	AF0	A=0	W	
	3178	C=(2)	#87	❻
%1				
#096h	1507	DAT0=A	W	❼
	16F	D0=D0+	#10	❽
	A6E	C=C-1	B	
	55F	GONC	%1	
	14F	C=DAT1	B	
	D7	D=C	A	❾
	170	D1=D1+	#1	
	118	C=R0		
	D0	A=0	A	
	1533	A=DAT1	X	
	CA	A=A+C	A	
	101	R1=A		❿
	172	D1=D1+	#3	
	D0	A=0	A	

❶ Début de la procédure $0 (tableau n°1).
❷ On pointe sur le contenu de TABL (objet graphique).
❸ Procédure $1 (affichage du tableau). Adresse du tableau sauvegardée.
❹ D1 contient le début du tableau.
❺ D0 contient le début de l'écran.
❻ #880h quartets.
❼ Point de branchement %1.
❽ Tout l'écran est effacé.
❾ Lecture du nombre de pastilles à gober.
❿ R1 = R0 + position du gobeur

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1533	A=DAT1	X	
	CA	A=A+C	A	❶
	102	R2=A		
	172	D1=D1+	#3	
#0CAh	134	D0=C		❷
	308	C=(1)	#8	
%2				
#0D0h	06	RSTK=C		❸
	137	CD1ex		
	135	D1=C		
	D5	B=C	A	❹
	AC2	C=0	S	
	147	C=DAT1	A	
	F2	CSL	A	❺
%3				❻
#0E2h	C6	C=C+C	A	
	5A4	GONC	%5	❼
	06	RSTK=C		
	34BE200	C=(5)	#002EB	
	81B4	A=PC		
#0F4h	CA	A=A+C	A	
	131	D1=A		❽
	307	C=(1)	#7	
%4				❾
#0FCh	06	RSTK=C		
	143	A=DAT1	A	
	146	C=DAT0	A	
	0E3E	A=A!C	X	❿
	1503	DAT0=A	X	

❶ R2 = R0 + position du carré.
❷ Affichage des bords. On pointe sur le début de l'écran.
❸ Point de branchement %2. Sauvegarde du compteur.
❹ B sauvegarde D1.
❺ On pointe sur la première colonne de la première ligne.
❻ Point de branchement %3.
❼ Test : est-ce un bord ? Sauvegarde de la ligne.
❽ On pointe sur les données graphiques relatives à l'obstacle.
❾ Affichage de l'obstacle (en D0).
❿ On effectue un « OU » logique avec le contenu de l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	16F	D0=D0+	#10	❶
	16F	D0=D0+	#10	
	161	D0=D0+	#2	
	172	D1=D1+	#3	
	07	C=RSTK		
	A0E	C=C-1	P	❷
	5ED	GONC	%4	
	132	AD0ex		
	32011	C=(3)	#110	❸
	EA	A=A-C	A	
	130	D0=A		
	07	C=RSTK		❹
%5				❺
#12Fh	161	D0=D0+	#2	
	B46	C=C+1	S	❻
	5CA	GONC	%3	
	132	AD0ex		
	D2	C=0	A	
	31EC	C=(2)	#CE	
	CA	A=A+C	A	❼
	130	D0=A		
	D9	C=B	A	
	135	D1=C		❽
	173	D1=D1+	#4	
	07	C=RSTK		
	A0E	C=C-1	P	❾
	460	GOC	%6	
	697F	GOTO	%2	
%6				❿
#15Ah	118	C=R0		

❶ On passe à la ligne suivante.
❷ Boucle sur 8 lignes.
❸ On revient au début de la colonne en cours de traitement.
❹ Restauration de la ligne.
❺ Point de branchement %5.
❻ Passage à la colonne suivante (en restant sur la même ligne).
❼ On passe à la ligne suivante.
❽ On passe à la ligne suivante (dans le tableau).
❾ Boucle sur 9 lignes.
❿ Point de branchement %6.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	134	D0=C		❶
	308	C=(1)	#8	
%7				
#163h	06	RSTK=C		❷
	AC2	C=0	S	
	147	C=DAT1	A	
	F2	CSL	A	❸
%8				❹
#16Dh	C6	C=C+C	A	
	5B1	GONC	%9	
	06	RSTK=C		
	302	C=(1)	#2	
	7912	GOSUB	$6	❺
#17Bh	132	ADOex		
	D2	C=0	A	
	31CC	C=(2)	#CC	
	EA	A=A-C	A	❻
	130	D0=A		
	07	C=RSTK		❼
%9				❽
#18Bh	161	D0=D0+	#2	
	B46	C=C+1	S	
	5BD	GONC	%8	
	132	AD0ex		
	D2	C=0	A	
	31EC	C=(2)	#CE	❾
	CA	A=A+C	A	
	130	D0=A		
	173	D1=D1+	#4	❿
	07	C=RSTK		

❶ On pointe sur le début de l'écran.
❷ Point de branchement %7. Le compteur est sauvegardé.
❸ On pointe sur la première colonne.
❹ Point de branchement %8. Test : est-ce un bord ? La ligne est sauvée.
❺ En D0, affichage d'une pastille à gober.
❻ Retour au début de la colonne.
❼ restauration de la ligne.
❽ Point de branchement %9. Passage à la colonne suivante.
❾ Passage à la ligne suivante (1 ligne de jeu = 7 lignes écran).
❿ Passage à la ligne suivante dans le tableau.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	A0E	C=C-1	P	❶
	58B	GONC	%7	
	137	CD1ex		❷
	D5	B=C	A	
#1B2h	111	A=R1		❸
	130	D0=A		
	301	C=(1)	#1	❹
	75D1	GOSUB	$6	
#1BFh	112	A=R2		❺
	130	D0=A		
	D2	C=0	A	❻
	79C1	GOSUB	$6	
#1CBh	840	ST=0	0	❼
$2				❽
#1CEh	1BB0100	D0=(5)	#0010B	
	D2	C=0	A	
	15C0	DAT0=C	1	❾
	870	?ST=1	0	
	90	GOYES	%1	
	304	C=(1)	#4	
	15C0	DAT0=C	1	
%1				
#1E7h	32004	C=(3)	#400	❿
	8FCEE10	GOSBVL	#01EEC	
	808AF	?CBIT=0	15	
	B0	GOYES	%2	
	11B	C=R3		
	D5	B=C	A	

❶ Boucle sur 9 lignes.
❷ On sauvegarde le début du tableau suivant.
❸ On calcule l'emplacement sur l'écran.
❹ Affichage du gobeur.
❺ On calcule l'emplacement sur l'écran.
❻ On affiche un carré.
❼ Ici, la pièce mobile active est le gobeur (et non le carré !).
❽ Procédure $2 (détection d'une pression sur les touches).
❾ Suppression du témoin α (sauf en cas de déplacement du gobeur).
❿ Point de branchement %1. Reprise au début si pression sur [ON].

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	618E	GOTO	$1	
%2				❶
#203h	D0	A=0	A	
	32040	C=(3)	#040	
	8FCEE10	GOSBVL	#01EEC	❷
	808A4	?CBIT=0	4	
	60	GOYES	%3	
	81B2	PC=A		
%3				❸
#21Ch	808A0	?CBIT=0	0	
	60	GOYES	%4	❹
	E4	A=A+1	A	
	E4	A=A+1	A	
%4				❺
#227h	808A2	?CBIT=0	2	
	60	GOYES	%5	❻
	CC	A=A-1	A	
	CC	A=A-1	A	
%5				❼
#232h	8AC	?A≠0	A	
	06	GOYES	$3	
	808A1	?CBIT=0	1	
	A0	GOYES	%6	❽
	D2	C=0	A	
	31EE	C=(2)	#EE	
	CA	A=A+C	A	
%6				❾
#246h	32080	C=(3)	#080	
	8FCEE10	GOSBVL	#01EEC	
	808A1	?CBIT=0	1	❿

❶ Point de branchement %2
❷ Test : appui sur la touche [STO] ? Dans l'affirmative, reset système.
❸ Point de branchement %3.
❹ Test : appui sur la touche [→] ?
❺ Point de branchement %4.
❻ Test : appui sur la touche [←] ?
❼ Point de branchement %5.
❽ Test : appui sur la touche [↓] ?
❾ Point de branchement %6.
❿ Test : appui sur la touche [↑] ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	A0	GOYES	%7	
	D2	C=0	A	
	31EE	C=(2)	#EE	
	EA	A=A-C	A	
%7				❶
#261h	8AC	?A≠0	A	
	13	GOYES	$3	❷
	808A0	?CBIT=0	1	
	62	GOYES	%11	
	860	?ST=0	1	
	80	GOYES	%8	
	840	ST=0	1	
	550	GONC	%9	
%8				❸
#278h	850	ST=1	1	
%9				❹
#27Bh	DA	A=C	A	
	33000C	C=(4)	#C000	
%10				❺
#283h	CE	C=C-1	A	
	5DF	GONC	%10	❻
	3151	C=(2)	#15	
	962	?A=C	B	❼
	47	GOYES	$4	
%11				❽
#291h	6C3F	GOTO	$2	
$3				❾
#295h	119	C=R1		
	860	?ST=0	0	❿

❶ Point de branchement %7.
❷ Test : un déplacement a-t-il été demandé ? Test : appui sur [NXT] ?
❸ Point de branchement %8.
❹ Point de branchement %9.
❺ Point de branchement %10.
❻ Temporisation.
❼ Test : appui sur [PRG]-[VAR]-[NXT] (mode « tricheur ») ?
❽ Point de branchement %11.
❾ Procédure $3 : détection d'une collision.
❿ On pointe, selon le sprite actif, sur le gobeur ou sur le carré.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	50	GOYES	%1	
	11A	C=R2		
%1				❶
#2A0h	DE	ACex	A	
	06	RSTK=C		
	CA	A=A+C	A	
	D2	C=0	A	
	3198	C=(2)	#89	❷
	CA	A=A+C	A	
	130	D0=A		
	14A	A=DAT0	B	❸
	30A	C=(1)	#A	
	906	?A≠C	P	❹
	60	GOYES	%3	
%2				❺
#2BBh	611F	GOTO	$2	
%3				❻
#2C0h	80870	?ABIT=1	0	
	7F	GOYES	%2	❼
	80861	?ABIT=0	1	
	C4	GOYES	$5	
	870	?ST=1	0	
	BE	GOYES	%2	❽
	861	?ST=0	1	
	72	GOYES	%6	❾
	3108	C=(2)	#80	
	DA	A=C	A	
%4				❿
#2DEh	32008	C=(3)	#800	
	801	OUT=C		

❶ Point de branchement %1. Un décalage est ajouté puis sauvegardé.
❷ Identification à l'aide d'une descente de 4 lignes.
❸ On lit l'écran (en fait, la case d'arrivée).
❹ Test : s'agit-il d'un bord ? En ce cas, c'est la fin du déplacement.
❺ Point de branchement %2. Fin du déplacement.
❻ Point de branchement %3.
❼ Test : s'agit-il du gobeur ou du carré ? Test : s'agit-il d'une pastille ?
❽ Arrêt si le carré est utilisé.
❾ Test : le son est-il autorisé ? Durée puis fréquence du son.
❿ Point de branchement %4. Bip avec volume moyen.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	6300	NOP4		
	D2	C=0	A	
	801	OUT=C		
	D6	C=A	A	
%5				❶
#2F1h	A6E	C=C-1	B	
	5CF	GONC	%5	
	A6C	A=A-1	B	❷
	53E	GONC	%4	
%6				❸
#2FDh	A0F	D=D-1	P	
	571	GONC	$5	
$4				❹
#303h	D9	C=B	A	
	135	D1=C		❺
	14B	A=DAT1	B	
	90C	?A≠0	P	❻
	60	GOYES	%7	
#310h	636D	GOTO	$0	
%7				❼
#314h	6C6D	GOTO	$1	
$5				❽
#318h	111	A=R1		
	860	?ST=0	0	❾
	50	GOYES	%1	
	112	A=R2		
%1				❿
#323h	D2	C=0	A	

❶ Point de branchement %5. Temporisation fréquence.
❷ Boucle durée, fréquence du son plus élevée.
❸ Point de branchement %6. Une pastille est supprimée.
❹ Procédure $4 (mode « tricheur »).
❺ On pointe sur le tableau suivant.
❻ Test : y a-t-il des pastilles à gober dans le nouveau tableau ?
❼ Point de branchement %7. S'il n'y a pas de pastille, tableau suivant.
❽ Procédure $5 (déplacement).
❾ On pointe sur le gobeur ou sur le carré.
❿ Point de branchement %1. Un décalage est ajouté.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	3144	C=(2)	#44	
	CA	A=A+C	A	
	130	D0=A		
	303	C=(1)	#3	❶
%2				❷
#331h	06	RSTK=C		
	14A	A=DAT0	B	
	3118	C=(2)	#81	❸
	0E66	A=A&C	B	
	148	DAT0=A	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❹
	161	D0=D0+	#2	
	07	C=RSTK		
	A0E	C=C-1	P	❺
	51E	GONC	%2	
	07	C=RSTK		
	06	RSTK=C		
	132	AD0ex		
	CA	A=A+C	A	
	D2	C=0	A	❻
	31CC	C=(2)	#CC	
	EA	A=A-C	A	
	130	D0=A		
	860	?ST=0	0	
	80	GOYES	%3	
	102	R2=A		❼
	550	GONC	%4	
%3				❽
#371h	101	R1=A		

❶ Quatre lignes...
❷ Point de branchement %2. Il convient ici de sauvegarder le compteur !
❸ On ne conserve que les extrémités.
❹ On passe à la ligne inférieure.
❺ Boucle sur 4 lignes.
❻ Désignation de la case d'arrivée.
❼ Modification du gobeur ou du carré.
❽ Point de branchement %3.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%4				❶
#374h	D2	C=0	A	
	870	?ST=1	0	
	40	GOYES	%5	❷
	E6	C=C+1	A	
%5				❸
#37Dh	7310	GOSUB	$6	
	32012	C=(3)	#210	
%6				❹
#386h	A3E	C=C-1	X	
	5CF	GONC	%6	
	07	C=RSTK		
	DA	A=C	A	
#390h	640F	GOTO	$3	❺
$6				❻
#394h	D0	A=0	A	
	A8A	A=C	P	❼
	137	CD1ex		
	06	RSTK=C		❽
	34A4000	C=(5)	#0004A	
	F0	ASL	A	
	C2	C=C+A	A	
	81B4	A=PC		❾
#3ADh	CA	A=A+C	A	
	131	D1=A		
	305	C=(1)	#5	
%1				❿
#3B5h	06	RSTK=C		
	14B	A=DAT1	B	

❶ Point de branchement %4.
❷ Affichage du gobeur ou du carré.
❸ Point de branchement %5.
❹ Point de branchement %6. Temporisation.
❺ Poursuite du déplacement.
❻ Procédure $6 (affichage des sprites).
❼ Lecture du sprite à afficher.
❽ D1 est sauvegardé.
❾ D1 pointe sur la description du sprite
❿ Point de branchement %1.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	14E	C=DAT0	B	❶
	0E6E	A=A!C	B	
	148	DAT0=A	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❷
	161	D0=D0+	#2	
	171	D1=D1+	#2	
	07	C=RSTK		
	A0E	C=C-1	P	❸
	5FD	GONC	%1	
	07	C=RSTK		
	135	D1=C		❹
	01	RTN		
$BORD				❺
#3DFh	EF092	*Données graphiques*		
	13C1D	*Données graphiques*		
	013A1	*Données graphiques*		
	94152	*Données graphiques*		
	1EF0	*Données graphiques*		
$SPRITES				❻
#3F7h	0000C	*Données graphiques*		
	3C3C3	*Données graphiques*		
	C3000	*Données graphiques*		
	00000	*Données graphiques*		
	81C2C	*Données graphiques*		
	38100	*Données graphiques*		
	00000	*Données graphiques*		
	08142	*Données graphiques*		
	4281	*Données graphiques*		
#423h				❼

❶ Un « OU » logique sert au transfert du graphisme vers l'écran.
❷ Passage à la ligne du dessous.
❸ Boucle sur 6 lignes.
❹ D1 est restauré.
❺ Procédure $BORD (données graphiques décrivant les obstacles).
❻ Procédure $SPRITES (données graphiques décrivant les pièces mobiles).
❼ Fin du code en #423h.

3

Exécuter le programme

Pour exécuter le programme, vous devez réunir dans le répertoire DSTAR (spécialement créé), les trois objets suivants :

- 'STAR', le lanceur RPL,
- 'TABL', l'objet graphique,
- 'CODE', le code exécutable

Pour jouer, il suffit ensuite de lancer l'exécution du programme RPL 'STAR'. Vous pouvez activer ou non le son, pour cela activez ou désactivez le flag beep RPL.

4

Comment jouer ?

Le but du jeu est "l'absorption" par le gobeur (le rond plein) de toutes les pastilles (les ronds vides). Le carré permet de se sortir des situations difficiles en offrant des « surfaces de rebondissement mobiles ». Votre gobeur et le carré ne peuvent se déplacer que d'une paroi vers une autre paroi. Votre gobeur avale tout sur son passage. Pour jouer, utilisez les touches suivantes :

- les quatre flèches [←], [↑], [→] et [↓] permettent de déplacer votre gobeur ou le carré,
- la touche [NXT] permet de passer de la manipulation du gobeur à celle du carré (et inversement). Lorsque le témoin α est allumé, cela signifie que vous déplacez le gobeur (ce témoin est éteint quand vous manipulez le carré),
- une pression sur [ON] permet de reprendre à partir du début du tableau (ce qui est indispensable en cas de blocage),
- une pression sur [STO] permet de quitter le jeu.

Les tricheurs et les joueurs un peu perdus seront heureux d'apprendre qu'en appuyant simultanément sur les touches [PRG]-[VAR]-[NXT] ils pourront passer directement au tableau suivant...

Il vous est possible de créer vos propres tableaux. Reportez-vous au paragraphe 5 du chapitre présentant la version de STAR destinée aux HP-48 S et SX.

5
Retrouver les fichiers sur la disquette

Les chemins d'accès aux objets utilisés sont les suivants :

- 'CODE' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\CODE,
- 'STAR' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\STAR,
- 'STAR48G' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\STAR48G,
- 'TABL' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\TABL,
- 'ASS' : DUNOD_1\G_GX\ASS,
- 'T1' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\T1,
- 'T2' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\T2,
- 'T3' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\T3,
- 'T4' : DUNOD_1\G_GX\OB_P_OBG\STAR_G\T4.

34

Xalk pour HP-48 G

1

Le jeu

1.1

Présentation

Xalk est un jeu de la célèbre famille des Tetris™ qui comblera tous les amateurs de jeux d'assemblage. Comme pour tous les jeux de ce type, des pièces tombent du ciel (cela dit, dans le cas de XALK, elles arrivent par la gauche !), il vous faut les assembler convenablement car les lignes de pièces identiques disparaissent. La place libérée permet d'accueillir de nouvelles pièces... Vous perdez quand un empilement de pièces atteint la largeur de l'écran. Trois pièces identiques disparaissent lorsqu'elles sont alignées horizontalement ou verticalement. Pour augmenter l'intérêt du jeu, des coups spéciaux (cascade, combinaison, etc.) imposent l'élaboration d'une véritable stratégie. Selon le niveau choisi, de 5 à 8 figures différentes vous seront proposées, la difficulté du jeu dépend directement de ce nombre de figures.

Pour la petite histoire, Xalk est une adaptation de Klax™ sur Amstrad CPC. Klax™ est lui-même un parent éloigné de Tetris™... La généalogie des jeux informatiques existe !

1.2
Liste des objets

Quatre objets sont associés à la mise en œuvre de ce jeu :

- 'XALK48G' est la chaîne de codes hexadécimaux qui devra être traitée à l'aide du programme ASS afin d'obtenir l'objet 'CODE',
- 'CODE' est le programme assembleur proprement dit, il est obtenu en traitant 'XALK48G' à l'aide de ASS,
- 'XALK' (ou 'LXG', au choix) est le programme lanceur RPL,
- 'ASS' génère 'CODE' à partir de 'XALK48G'.

'XALK48G' et 'ASS' ne sont pas nécessaires à l'exécution du programme. Seules les variables 'XALK' (lanceur RPL) et 'CODE' (code exécutable) sont nécessaires pour jouer.

2
Les objets

2.1
Le lanceur RPL

Cet objet est à enregistrer dans une variable nommée 'XALK' (ou 'LXG'). Saisissez-le entre « et » qui sont les délimiteurs d'un programme RPL.

```
« DO CLEAR "XALK 48G"
":NIVEAU 1→4:"
INPUT OBJ→ DTAG
  UNTIL { 1 2 3 4 }
OVER POS
  END #0h DUP DUP
DUP2 PDIM 2 →LIST
PVIEW PICT { #0h
#37h } "SCORE:" 3
```

```
→GROB REPL + 3 +
-9 CODE
 »
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à # 371Eh,
- au niveau 1, une longueur égale à 179,5 octets.

2.2
Le code hexadécimal

a
La chaîne à saisir

Cette chaîne de codes hexadécimaux est à enregistrer dans une variable nommée 'XALK48G'. Saisissez la chaîne sans espace ni saut de ligne mais entre deux délimiteurs " et ".

```
CCD20 A8700 20143 34E24 A28A2 C0142
16480 8C84F 808F0 81741 47135 17914
F1F50 40814 D81B4 349E5 00CA1 00320
A0CA1 017BC 4AF21 9C017 515C0 15571
9E21C C15C0 14517 73292 A1451 E0580
14F80 8B350 8561E 60401 4FF61 4D1C0
14F75 74D1D 37E24 E5E57 62484 08448
45171 1471C 10632 4008F CEE10 808A2
B21B0 0100D 215C0 32200 8FCEE 10808
A2FE3 0815C 0D280 8B580 8415C 28717
28513 04901 C1E57 F63CD 14A96 CC078
83E57 F9332 8008F CEE10 808B0 80842
51187 2C085 08526 78080 8B266 808B5
80843 5B287 36285 3D28A 1C1CD 7A03E
514A9 6CC07 323CD 7A330 720CE 4046C
1FD1D 22880 F2866 50801 D4CC5 DFB65
59E20 32800 8FCEE 10808 A2FED 281B3
E779A 2CF14 A96CD 172C2 E779D 2DBB0
64B08 70BD6 5AE71 F3D29 03291 C0AF2
```

CCD 20

```
15F08 18FA2 75032 2D230 2D530 1D7F7
30886 6A080 1D280 1D4CC 5DFCF 56E20
D2304 D51DD 1D3E7 74221 6F16F 16114
E14D1 70B07 56ECD 5DDD2 304D5 D3E7D
4F0E0 33954 0E213 D30E9 8BE31 4BE71
70908 DE14F 9065E E7170 14F90 68D74
E1CF7 ED1CF 78D1D 231F6 7BE2C D54AD
3E733 A340C B13D1 4B908 3617E 14F90
60717 E14F1 CE1CE 1CE90 6D414 F906E
11CE1 4F906 31D23 04D57 571CD 7F61D
2D577 61E57 161E5 7B51D 231F6 7E62B
07588 42314 F9061 FD2E6 5EC1C E1CE1
4F906 CD1CE 14F90 61DD2 3025D A8746
068B0 D2304 D5D23 0FD77 6D014 A90CD
31811 4A908 23855 30914 A1611 48181
D0148 16F16 F161A 0E52E D231F 277D1
CF301 9071B CD55A 86590 84564 4E1E5
040AF 215F0 818FA 205C6 C67BF 01C41
43329 F18BA A1818 F8914 11C70 5143E
41410 4D134 BA300 D7742 01601 ED040
30578 C0169 1E004 03027 8B069 FB1BD
86081 46CBC BD4C4 E2C4C 4E2F0 C2C4C
21340 175DF D0309 14816 F16F1 61A0E
50F01 78BFD 215F0 C6C61 10CAC 6C6CA
13130 914B1 4816F 16F16 1171A 0E5AE
1E604 0011B 83100 14A80 84398 65F17
11590 01177 05153 7A7A1 51704 01153
7D7CE 814A0 E59FA F606D 1A851 11C5C
081DF 1C013 13080 614BD B808B 0B0D2
15435 60C4C 414E0 E6A14 C16F1 6F161
17107 A0E59 C3413 10013 2EA13 0DB80
8A050 16007 812A0 F5880 1854D ADD06
2880F 28665 08010 6D6CE 5DF07 CD56E
2007D 5D2B3 65205 9FB36 52059 F01F7
14141 41414 14141 4F7F7 55B65 5B655
B655B 6F7F7 F7F7F 7F7F7 F7F7F 7F7F7
94D51 4A2A2 14D59 4F7F7 41E38 06363
80E34 1F7F7 D5943 67777 3694D 5F7F7
3600C 12222 C1003 6F7E3 B6555 55555
5555B 6E3E0 11111 11111 1111E 04060
40404 04040 40E0E 01111 80402 01010
F1E01 10101 E0010 111E0 80C0A 090F1
80808 080F1 1010F 00101 0111E 0C020
1010F 01111 11E0F 10101 80402 02020
```

```
20E01 11111 E0111 111E0 E0111 111E1
01018 060
```

Placez dans la pile une copie de la chaîne de caractères stockée dans 'XALK48G', activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à # 5AC0h,
- au niveau 1, une longueur égale à 1933 octets.

Pour générer l'objet 'CODE' à partir de la chaîne 'XALK48G' :

- placez la chaîne de codes hexadécimaux (variable 'XALK48G') au niveau 1 de la pile, **"CCD20A87002014334...** s'affiche,
- exécuter le programme ASS,
- au niveau 1 de la pile doit apparaître **Code**,
- si ce n'est pa encore fait, placez-vous dans le répertoire DXALK,
- saisissez le nom **'CODE'**,
- appuyez sur la touche [STO],
- appuyez sur la touche [VAR], le nom **CODE** doit apparaître dans la liste des variables.

Rappelons que pour placer 'CODE' dans la pile sans l'évaluer, vous devez appuyer sur [↱] avant de choisir l'article CODE du menu. Vérifiez l'intégrité de la variable 'CODE' en plaçant au niveau 1 de la pile une copie de cet objet (**Code** doit apparaître au niveau 1), activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à # 3E94h,
- au niveau 1, une longueur égale à 967,5 octets.

b *Codes commentés*

Adresses	Codes hexadécimaux	Mnémoniques	Renvois
$$			❶
	CCD20		❷
#000h	A8700		❸
	20	P= 0	

❶ Procédure **$$** (initialisation).
❷ Prologue d'un objet de type code exécutable.
❸ Longueur hors prologue du code (en nombre de quartets).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	143	A=DAT1	A	
	34E24A2	C=(5)	#2A42E	❶
	8A2	?A=C	A	
	C0	GOYES	%1	
	142	A=DAT0	A	
	164	D0=D0+	#5	
	808C	PC=(A)		
%1				❷
#020h	84F	ST=0	15	
	808F	INTOFF		
	08	CLRST		❸
	174	D1=D1+	#5	
	147	C=DAT1	A	❹
	135	D1=C		
	179	D1=D1+	#A	
	14F	C=DAT1	B	❺
	1F50408	D1=(5)	#80405	
	14D	DAT1=C	B	❻
	81B4	A=PC		
#046h	349E500	C=(5)	#005E9	
	CA	A=A+C	A	❼
	100	R0=A		
	320A0	C=(3)	#0A0	
	CA	A=A+C	A	❽
	101	R1=A		
	7BC4	GOSUB	$NEXT	❾
#060Ch	AF2	C=0	W	
	19C0	D0=(2)	#0C	
	175	D1=D1+	#6	❿
	15C0	DAT0=C	1	

❶ Test : a-t-on -9 au niveau 1 de la pile ? Dans la négative, reset système.
❷ Point de branchement %1. Inhibition des interruptions.
❸ On pose ST1/2/3 = 0.
❹ On lit le niveau de jeu (au niveau 2 de la pile RPL).
❺ On lit le niveau (il s'agit d'un nombre hexa).
❻ On mémorise le niveau en mémoire.
❼ R0 contient l'adresse de la table des sprites cubes.
❽ R1 contient l'adresse de la table des sprites chiffres.
❾ On mémorise le cube suivant.
❿ Suppression du témoin « sablier » en haut de l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1557	DAT1=C	W	❶
	19E2	D0=(2)	#2E	
	1CC	D1=D1-	#D	
	15C0	DAT0=C	1	
	145	DAT1=C	A	❷
	177	D1=D1+	#8	
	3292A	C=(3)	#A29	
	145	DAT1=C	A	
	1E0580	D1=(4)	#0850	
	14F	C=DAT1	B	
	808B3	?CBIT=1	3	❸
	50	GOYES	$0	
	856	ST=1	6	❹
$0				❺
#09Eh	1E6040	D1=(4)	#0406	
	14F	C=DAT1	B	❻
	F6	CSR	A	
	14D	DAT1=C	B	
	1C0	D1=D1-	#1	
	14F	C=DAT1	B	❼
	7574	GOSUB	$NEXT	
#0B6h	D1	B=0	A	
	D3	D=0	A	
	7E24	GOSUB	$SPR1	
#0BEh	E5	B=B+1	A	
	E5	B=B+1	A	❽
	7624	GOSUB	$SPR1	
$1				❾
#0C6h	840	ST=0	0	❿
	844	ST=0	4	

❶ On pose score = 0. On pointe en #0012E.
❷ Vitesse (affichage) en #80400h et vitesse (compteur) en #80408h.
❸ Test : flag bip RPL ?
❹ Si le flag bip est armé, on a ST6 = 1 (flag son pour le jeu).
❺ Procédure $0 (cube suivant).
❻ Lecture du cube suivant qui devient courant et est mémorisé.
❼ Lecture du niveau, calcul, stockage et affichage du cube suivant.
❽ Affichage du cube courant au centre à gauche
❾ Procédure $1 (test des touches).
❿ Initialisation des pressions sur [÷] et du flag « triplets » (ST4).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	845	ST=0	5	❶
	171	D1=D1+	#2	
	147	C=DAT1	A	
	1C1	D1=D1-	#2	
%1				❷
#0D8h	06	RSTK=C		
	32400	C=(3)	#004	
	8FCEE10	GOSBVL	#01EEC	❸
	808A2	?CBIT=0	2	
	B2	GOYES	%3	❹
	1B00100	D0=(5)	#00100	
	D2	C=0	A	
	15C0	DAT0=C	1	
%2				❺
	32200	C=(3)	#002	
	8FCEE10	GOSBVL	#01EEC	❻
	808A2	?CBIT=0	2	
	FE	GOYES	%2	
	308	C=(1)	#8	
	15C0	DAT0=C	1	
	D2	C=0	A	
%3				❼
	808B5	?CBIT=1	5	
	80	GOYES	%4	❽
	841	ST=0	1	
	5C2	GONC	%5	
%4				❾
	871	?ST=1	1	
	72	GOYES	%5	
	851	ST=1	1	❿

❶ Initialisation du flag « cascade » (ST5). Charge la vitesse (compteur).
❷ Point de branchement %1. Sauvegarde de la temporisation.
❸ Test des touches [←] et [5].
❹ Test : pression sur [5] ? Si oui, on éteint l'écran.
❺ Point de branchement %2.
❻ Attente d'une pression sur [2] pour allumer à nouveau l'écran.
❼ Point de branchement %3.
❽ Test : pression sur [←] ?
❾ Point de branchement %4. Test : [←] déjà pressée auparavant ?
❿ Désensibilisation de [←].

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	304	C=(1)	#4	
	901	?B=C	P	❶
	C1	GOYES	%5	
	E5	B=B+1	A	
	7F63	GOSUB	$SPRCALC	❷
#139h	CD	B=B-1	A	
	14A	A=DAT0	B	
	96C	?A≠0	B	❸
	C0	GOYES	%5	
	7883	GOSUB	$SPRO	
#147h	E5	B=B+1	A	
	7F93	GOSUB	$SPR1	
%5				❹
#14Dh	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	
	808B0	?CBIT=1	0	
	80	GOYES	%6	❺
	842	ST=0	2	
	511	GONC	%7	
%6				❻
	872	?ST=1	2	
	C0	GOYES	%7	❼
	850	ST=1	0	
	852	ST=1	2	
#171h	6780	GOTO	$3	
%7				❽
	808B2	?CBIT=1	2	
	66	GOYES	$2b	❾
	808B5	?CBIT=1	5	
	80	GOYES	%8	❿

❶ Test : le cube courant est-il sur le bord inférieur de l'écran ?
❷ On consulte le contenu de la case au-dessous du cube courant.
❸ Test : cette case est-elle vide ? Si oui, le cube courant descend.
❹ Point de branchement %5. Test : pression sur [α], [8] ou [÷] ?
❺ Test : pression sur [÷] ?
❻ Point de branchement %6.
❼ Test : la touche [÷] a-t-elle déjà été pressée ?
❽ Point de branchement %7.
❾ Test : pression sur [8] ? Si oui, fin du jeu et reset système.
❿ Test : pression sur [α] ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	843	ST=0	3	
	5B2	GONC	%9	
%8				❶
	873	?ST=1	3	
	62	GOYES	%9	❷
	853	ST=1	3	
	D2	C=0	A	
	8A1	?B=C	A	❸
	C1	GOYES	%9	
	CD	B=B-1	A	
	7A03	GOSUB	$SPRCALC	❹
#19Eh	E5	B=B+1	A	
	14A	A=DAT0	B	
	96C	?A≠0	B	❺
	C0	GOYES	%9	
	7323	GOSUB	$SPRO	❻
#1ACh	CD	B=B-1	A	
	7A33	GOSUB	$SPR1	
%9				❼
#1B2h	07	C=RSTK		
	20	P=	0	
	CE	C=C-1	A	
	404	GOC	$3	❽
	6C1F	GOTO	%1	
$2				❾
%a				❿
#1BFh	D1	B=0	A	
	D2	C=0	A	
	28	P=	8	

❶ Point de branchement %8.
❷ Test : avait-on déjà appuyé sur [α] ?
❸ test : le cube en mouvement est-il sur le bord supérieur de l'écran ?
❹ Quel le contenu de la case au-dessus du cube en mouvement ?
❺ La case au-dessus du cube en mouvement est-elle vide ?
❻ Si oui, on efface le cube et on le réaffiche sur la ligne du dessus.
❼ Point de branchement %9.
❽ Temporisation entre deux tests des touches.
❾ Procédure $2 (fin du jeu).
❿ Point de branchement %a.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%1				❶
	80F2	CPex	2	
	866	?ST=0	6	
	50	GOYES	%2	❷
	801	OUT=C		
%2				❸
	D4	A=B	A	
%3				❹
	CC	A=A-1	A	
	5DF	GONC	%3	
	B65	B=B+1	B	
	59E	GONC	%1	❺
	20	P=	0	
%b				❻
#1E0h	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	
	808A2	?CBIT=0	2	
	FE	GOYES	%b	
	D2	C=0	A	
	81B3	PC=C		❼
$3				❽
#1F9h	E7	D=D+1	A	
	79A2	GOSUB	$SPRCALC	
#1FFh	CF	D=D-1	A	
	14A	A=DAT0	B	
	96C	?A≠0	B	❾
	D1	GOYES	$4	
	72C2	GOSUB	$SPRO	❿
#20Dh	E7	D=D+1	A	

❶ Point de branchement %1. On alterne OUT #800h et OUT #000h...
❷ ...sauf si le flag bip RPL est désactivé.
❸ Point de branchement %2.
❹ Point de branchement %3. Temporisation fréquence.
❺ Boucle durée.
❻ Point de branchement %b. Attente d'une pression sur la touche [8].
❼ Reset système.
❽ Procédure $3 (un cran à droite !).
❾ Test : y a-t-il une figure sur la droite de la figure en mouvement ?
❿ On efface la figure active puis on l'affiche à nouveau un cran à droite.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	79D2	GOSUB	$SPR1	
#213h	DB	C=D	A	
	B06	C=C+1	P	❶
	4B0	GOC	$4	
	870	?ST=1	0	❷
	BD	GOYES	$3	
	65AE	GOTO	$1	
$4				❸
#224h	71F3	GOSUB	$TEMPO	❹
	D2	C=0	A	
	903	?C=D	P	
	29	GOYES	$2a	❺
	1C0	D1=D1-	#1	
	AF2	C=0	W	
	15F0	C=DAT1	1	❻
	818FA2	C=C-3	A	
	7503	GOSUB	$SCORE	❼
#243h	22	P=	2	
	D2	C=0	A	
	302	C=(1)	#2	
	D5	B=C	A	❽
	301	C=(1)	#1	
	D7	D=C	A	
	F7	DSR	A	
	308	C=(1)	#8	
%1				❾
	866	?ST=0	6	
	A0	GOYES	%2	❿
	801	OUT=C		

❶ Test : D = 15 ? Sommes-nous sur la colonne (n°#Fh) de droite ?
❷ On teste les touches sauf en cas de pression sur [÷].
❸ Procédure $4 (cube posé).
❹ Petite pause.
❺ Test : une pièce est-elle coincée à gauche ? En ce cas, fin du jeu.
❻ Lecture du niveau.
❼ On modifie le score en fonction du niveau.
❽ B contient la fréquence, et D, la durée.
❾ Point de branchement %1.
❿ Son avec volume faible (sauf si le flag bip RPL est désactivé).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D2	C=0	A	
	801	OUT=C		
%2				❶
	D4	A=B	A	
%3				❷
	CC	A=A-1	A	
	5DF	GONC	%3	❸
	CF	D=D-1	A	
	56E	GONC	%1	
	20	P=	0	
$5				❹
#271h	D2	C=0	A	
	304	C=(1)	#4	
	D5	B=C	A	❺
	1DD1	D1=(2)	#1D	
%1				❻
	D3	D=0	A	
	E7	D=D+1	A	
%2				❼
	7422	GOSUB	$SPRCALC	
#284h	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❽
	161	D0=D0+	#2	
	14E	C=DAT0	B	
	14D	DAT1=C	B	❾
	170	D1=D1+	#1	
	B07	D=D+1	P	
	56E	GONC	%2	❿
	CD	B=B-1	A	

❶ Point de branchement %2.
❷ Point de branchement %3.
❸ Temporisation fréquence et boucle durée.
❹ Procédure $5 (création du tableau de traitement des cascades).
❺ B est le compteur des 5 lignes. Le tableau se trouve en #8041Dh.
❻ Point de branchement %1. D est le compteur des 15 colonnes.
❼ Point de branchement %2. On pointe sur le sprite (coordonnées (B,D)).
❽ On descend d'une ligne pour identification.
❾ Transfert écran/tableau puis donnée suivante.
❿ Boucle les 15 colonnes.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	5DD	GONC	%1	❶
$6				❷
#2A1h	D2	C=0	A	
	304	C=(1)	#4	
	D5	B=C	A	❸
%1				❹
	D3	D=0	A	
	E7	D=D+1	A	
	D4	A=B	A	
	F0	ASL	A	
	E0	A=A-B	A	
	339540	C=(4)	#0459	
	E2	C=C-A	A	
	13D	D1=CS		
%2				❺
	30E	C=(1)	#E	
	98B	?C≤D	P	❻
	E3	GOYES	%3	
	14B	A=DAT1	A	❼
	E7	D=D+1	A	
	170	D1=D1+	#1	
	908	?A=0	P	❽
	DE	GOYES	%2	
	14F	C=DAT1	B	❾
	906	?A≠C	P	
	5E	GOYES	%2	❿
	E7	D=D+1	A	
	170	D1=D1+	#1	
	14F	C=DAT1	B	

❶ Boucle sur 5 lignes (après la boucle sur les 15 colonnes).
❷ Procédure $6 (recherche horizontale).
❸ B est le compteur des 5 lignes.
❹ Point de branchement %1. D pointe sur la 1ère colonne. A = B × 15.
❺ Point de branchement %2.
❻ On s'arrête deux figures avant la fin de la ligne (sur la droite).
❼ Lecture de la case courante (coordonnées (B,D)).
❽ La case courante doit contenir une figure.
❾ Lecture de la case suivante (colonne suivante de la même ligne).
❿ Test : identité ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	906	?A≠C	P	
	8D	GOYES	%2	❶
	7AE1	GOSUB	$SPRO	❷
#2EBh	CF	D=D-1	A	
	7ED1	GOSUB	$SPRO	
#2F1h	CF	D=D-1	A	
	78D1	GOSUB	$SPRO	❸
#2F7h	D2	C=0	A	
	31F6	C=(2)	#6F	
	7BE2	GOSUB	$BEEP	
%3				❹
#301h	CD	B=B-1	A	
	54A	GONC	%1	
$7				❺
#306h	D3	D=0	A	
	E7	D=D+1	A	
%1				❻
	33A340	C=(4)	#043A	
	CB	C=C+D	A	❼
	13D	D1=CS		
	14B	A=DAT1	B	❽
	908	?A=0	P	
	36	GOYES	%4	❾
	17E	D1=D1+	#F	
	14F	C=DAT1	B	
	906	?A≠C	P	
	07	GOYES	%6	❿
	17E	D1=D1+	#F	
	14F	C=DAT1	B	

❶ Test : identité ?
❷ Ici, on a un triplet horizontal de figures identiques.
❸ On efface le triplet à l'écran mais le triplet demeure dans le tableau.
❹ Point de branchement %3. Boucle sur 5 lignes.
❺ Procédure $7 (recherche horizontale). On pointe sur la 1ère colonne.
❻ Point de branchement %1.
❼ On pointe sur le tableau créé en $5.
❽ Lecture de B2 (figure au centre).
❾ Test : est-ce un blanc ? Non ? Alors lecture de B1 (au-dessus de B2).
❿ Test B1 = B2 ? Si oui, lecture de B0.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1CE	D1=D1-	#F	
	1CE	D1=D1-	#F	
	1CE	D1=D1-	#F	❶
	906	?A≠C	P	
	D4	GOYES	%5	❷
	14F	C=DAT1	B	
	906	?A≠C	P	
	E1	GOYES	%2	❸
	1CE	D1=D1-	#F	
	14F	C=DAT1	B	
	906	?A≠C	P	
	31	GOYES	%2	❹
	D2	C=0	A	
	304	C=(1)	#4	❺
	D5	B=C	A	
	7571	GOSUB	$SPRO	❻
#35Ah	CD	B=B-1	A	
	7F61	GOSUB	$SPRO	❼
%2				❽
#360h	D2	C=0	A	
%3				❾
	D5	B=C	A	
	7761	GOSUB	$SPRO	
#368h	E5	B=B+1	A	
	7161	GOSUB	$SPRO	
#36Eh	E5	B=B+1	A	
	7B51	GOSUB	$SPRO	
#374h	D2	C=0	A	
	31F6	C=(2)	#6F	
	7E62	GOSUB	$BEEP	❿

❶ Préparation de la lecture de B3.
❷ Test : B0 = B2 ?
❸ Test : B2 = B3 ?
❹ Test : B2 = B4 ?
❺ Test : toute la ligne contient-elle la même figure ?
❻ On efface B4 à l'écran.
❼ On efface B3 à l'écran.
❽ Point de branchement %2. On efface B0, B1 et B2 à l'écran.
❾ Point de branchement %3. On efface le triplet pointé par B.
❿ Bip ! Temporisation.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%4				❶
#37Eh	B07	D=D+1	P	
	588	GONC	%1	❷
	423	GOC	$8	
%5				❸
	14F	C=DAT1	B	
	906	?A≠C	P	
	1F	GOYES	%4	❹
	D2	C=0	A	
	E6	C=C+1	A	
	5EC	GONC	%3	
%6				❺
	1CE	D1=D1-	#F	
	1CE	D1=D1-	#F	
	14F	C=DAT1	B	
	906	?A≠C	P	
	CD	GOYES	%4	❻
	1CE	D1=D1-	#F	
	14F	C=DAT1	B	
	906	?A≠C	P	
	1D	GOYES	%4	
	D2	C=0	A	
	302	C=(1)	#2	
	5DA	GONC	%3	❼
$8				❽
#3B7h	874	?ST=1	4	
	60	GOYES	%1	❾
#3BCh	68B0	GOTO	$10	
%1				❿
	D2	C=0	A	

❶ Point de branchement %4.
❷ Boucle sur 15 colonnes.
❸ Point de branchement %5.
❹ Test : B2 = B3 ? On efface B1, B2 et B3.
❺ Point de branchement %6.
❻ Test : B2 = B3 ? Test : B2 = B4 ?
❼ On efface B2, B3 et B4.
❽ Procédure $8 (cascades).
❾ Test : des triplets ont-ils été effacés ?
❿ Point de branchement %1.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	304	C=(1)	#4	
	D5	B=C	A	❶
%2				❷
	D2	C=0	A	
	30F	C=(1)	#F	
	D7	D=C	A	
%3				❸
	76D0	GOSUB	$SPRCALC	
#3D2h	14A	A=DAT0	B	
	90C	?A≠0	P	
	D3	GOYES	%5	❹
	181	D0=D0-	#2	
	14A	A=DAT0	B	❺
	908	?A=0	P	
	23	GOYES	%5	
	855	ST=1	5	❻
	309	C=(1)	#9	
%4				❼
	14A	A=DAT0	B	
	161	D0=D0+	#2	
	148	DAT0=A	B	❽
	181	D0=D0-	#2	
	D0	A=0	A	
	148	DAT0=A	B	❾
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	❿
	A0E	C=C-1	P	
	52E	GONC	%4	
	D2	C=0	A	

❶ B est le compteur des 5 lignes de l'écran de jeu.
❷ Point de branchement %2. On commence par la colonne de droite.
❸ Point de branchement %3. Lecture du sprite en (B,D).
❹ Recherche d'un blanc.
❺ Si un blanc est trouvé, on cherche une figure sur la gauche.
❻ Activation du flag ST5 (cascade). 10 lignes.
❼ Point de branchement %4.
❽ Transfert du cube inter-écran.
❾ Efface le cube à sa précédente position.
❿ On passe à la ligne du dessous (boucle les 10 lignes).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	31F2	C=(2)	#2F	
	77D1	GOSUB	$BEEP	❶
%5				❷
#415h	CF	D=D-1	A	
	301	C=(1)	#1	
	907	?C≠D	P	
	1B	GOYES	%3	
	CD	B=B-1	A	❸
	55A	GONC	%2	
$9				❹
#424h	865	?ST=0	5	
	90	GOYES	%1	
	845	ST=0	5	❺
	644E	GOTO	$5	
%1				❻
	1E5040	D1=(4)	#0405	
	AF2	C=0	W	
	15F0	C=DAT1	1	❼
	818FA2	C=C-3	A	
	05	SETDEC		
	C6	C=C+C	A	❽
	C6	C=C+C	A	
	7BF0	GOSUB	$SCORE	❾
#44Dh	1C4	D1=D1-	#5	
	143	A=DAT1	A	
	329F1	C=(3)	#1F9	
	8BA	?A≤C	A	❿
	A1	GOYES	$10	
	818F89	A=A-10	A	

❶ Bip !
❷ Point de branchement %5.
❸ Boucle sur 14 colonnes et 5 lignes.
❹ Procédure $9 (score et vitesse).
❺ En cas de cascade, on recommence à partir de la création du tableau.
❻ Point de branchement %1.
❼ Lecture du niveau.
❽ Multiplication des niveaux.
❾ Score = niveau × 5 (voir $4). Lecture de la vitesse (compteur).
❿ Test : la vitesse est-elle encore modifiable ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	141	DAT1=A	A	❶
	1C7	D1=D1-	#8	
	05	SETDEC		
	143	A=DAT1	A	
	E4	A=A+1	A	
	141	DAT1=A	A	❷
	04	SETHEX		
$10				❸
#475h	D1	B=1	A	
	34BA300	C=(5)	#003AB	
	D7	D=C	A	
	7420	GOSUB	$SPRCALC	❹
#484h	160	D0=D0+	#1	
	1ED040	D1=(4)	#040D	
	305	C=(1)	#5	❺
	78C0	GOSUB	$AFF	
#494h	169	D0=D0+	#A	
	1E0040	D1=(4)	#0400	❻
	302	C=(1)	#2	
	78B0	GOSUB	$AFF	
#4A4h	6BFB	GOTO	$0	❼
#4A8h				
$SPRCALC			❽	
#000h	1BD8608	D0=(5)	#8068D	❾
	146	C=DAT0	A	

❶ Si ce test est positif, on modifie la vitesse (compteur).
❷ Modification de la vitesse (affichage).
❸ Procédure $10 (affichage).
❹ Adresse sur écran puis données du score.
❺ Nombre de quartets (6), affichage du score.
❻ Données relatives à la vitesse, nombre de quartets (3).
❼ Affichage de la vitesse puis passage à la figure suivante.
❽ Procédure $SPRCALC. En #4A8h, le compteur d'adresses est remis à zéro, on repart donc à #000h pour l'adressage des routines.
❾ Lecture de la bitmap écran pour que D0 pointe au début de l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	CB	C=C+D	A	❶
	CB	C=C+D	A	
	D4	A=B	A	
	C4	A=A+A	A	
	E2	C=C-A	A	
	C4	A=A+A	A	
	C4	A=A+A	A	
	E2	C=C-A	A	
	F0	ASL	A	
	C2	C=C+A	A	
	C4	A=A+A	A	
	C2	C=C+A	A	
	134	D0=C		
	01	RTN		
$SPR0				❷
#027h	7FDF	GOSUB	$SPRCALC	❸
	D0	A=0	A	
	309	C=(1)	#9	❹
%1				❺
	148	DAT0=A	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	❻
	A0E	C=C-1	P	
	50F	GONC	%1	❼
	01	RTN		
$SPR1				❽
#044h	78BF	GOSUB	$SPRCALC	❾

❶ On ajoute successivement à D0 : #2h×D (*colonne*), puis -#2h×B, puis -#8h×B, puis #80h×B, et enfin, #100h×B (*ligne*).
❷ Procédure $SPR0.
❸ D0 correspond à (B,D) sur l'écran.
❹ 10 lignes.
❺ Point de branchement %1. On efface le sprite (B,D) à l'écran.
❻ Passage à la ligne du dessous.
❼ Boucle sur 10 lignes.
❽ Procédure $SPR1.
❾ D0 correspond à (B,D).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D2	C=0	A	❶
	15F0	C=DAT1	1	
	C6	C=C+C	A	
	C6	C=C+C	A	
	110	A=R0		
	CA	A=A+C	A	❷
	C6	C=C+C	A	
	C6	C=C+C	A	
	CA	A=A+C	A	
	131	D1=A		
	309	C=(1)	#9	❸
%1				❹
	14B	A=DAT1	B	
	148	DAT0=A	B	❺
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	❻
	171	D1=D1+	#2	
	A0E	C=C-1	P	
	5AE	GONC	%1	
	1E6040	D1=(4)	#0406	
	01	RTN		
$NEXT				❼
#083h	1B83100	D0=(5)	#00138	❽
%1				❾
	14A	A=DAT0	B	
	80843	ABIT=0	3	❿
	986	?A>C	P	
	5F	GOYES	%1	

❶ D1 correspond au numéro du sprite à afficher.
❷ D1 = R0 + C × #14h.
❸ 10 lignes !
❹ Point de branchement %1.
❺ Transfert du sprite depuis la table des sprites vers l'écran.
❻ Passage à la ligne du dessous puis donnée suivante dans le tableau.
❼ Procédure $NEXT.
❽ On pointe sur l'horloge (pour obtenir un nombre aléatoire).
❾ Point de branchement %1.
❿ Plus de hasard ! Comparaison avec le niveau (nombre de sprites).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	171	D1=D1+	#2	
	1590	DAT1=A	1	❶
	01	RTN		
$SCORE				❷
#0A0h	177	D1=D1+	#8	
	05	SETDEC		
	1537	A=DAT1	W	
	A7A	A=A+C	W	
	1517	DAT1=A	W	❸
	04	SETHEX		
	01	RTN		
$AFF				❹
#0B4h	1537	A=DAT1	W	❺
	D7	D=C	A	
	CE	C=C-1	A	
%1				❻
	814	ASRC		
	A0E	C=C-1	P	
	59F	GONC	%1	
	AF6	C=A	W	❼
%2				❽
	06	RSTK=C		
	D1	B=0	A	
	A85	B=C	P	❾
	111	A=R1		
	C5	B=B+B	A	
	C0	A=A+B	A	❿
	81D	BSRB		

❶ Mémorisation du numéro du prochain sprite.
❷ Procédure $SCORE. On pointe en mémoire sur le score.
❸ Modification du score (ajout de niveau ou d niveau × 4).
❹ Procédure $AFE.
❺ Lecture des chiffres à afficher. D correspond au nombre de quartets.
❻ Point de branchement %1. Premier quartet affichable sur champ P.
❼ C contient les chiffres à afficher.
❽ Point de branchement %2. Sauvegarde des chiffres dans la pile RSTK.
❾ B = numéro du sprite à afficher. A = début de la table.
❿ A = A + B × #2h.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	F1	BSL	A	
	C0	A=A+B	A	❶
	131	D1=A		❷
	308	C=(1)	#8	
%3				❸
	06	RSTK=C		
	14B	A=DAT1	B	
	DB	C=D	A	
	808B0	?CBIT=1	0	
	B0	GOYES	%4	❹
	D2	C=0	A	
	1543	DAT0=C	X	❺
	560	GONC	%5	
%4				❻
	C4	A=A+A	A	
	C4	A=A+A	A	
%5				❼
	14E	C=DAT0	B	
	0E6A	C=C!A	B	❽
	14C	DAT0=C	B	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	161	D0=D0+	#2	❾
	171	D1=D1+	#2	
	07	C=RSTK		
	A0E	C=C-1	P	
	59C	GONC	%3	❿
	3413100	C=(5)	#00131	
	132	AD0ex		
	EA	A=A-C	A	

❶ A = A + B × #10h.
❷ D1 désigne le sprite à afficher dans la table.
❸ Point de branchement %3. Lecture des données graphiques (chiffres).
❹ Test : parité du compteur ?
❺ On affiche trois quartets blancs si le compteur est pair.
❻ Point de branchement %4. Décalage du chiffre (compteur impair).
❼ Point de branchement %5.
❽ Opération « OU » logique avec l'affichage.
❾ Passage à la ligne du dessous et donnée suivante dans le tableau.
❿ Boucle sur 9 lignes du sprite « chiffre ».

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	130	D0=A		❶
	DB	C=D	A	
	808A0	?CBIT=0	0	❷
	50	GOYES	%6	
	160	D0=D0+	#1	
%6				❸
	07	C=RSTK		
	812	CSLC		❹
	A0F	D=D-1	P	
	588	GONC	%2	
	01	RTN		
$BEEP				❺
#144h	854	ST=1	4	❻
	DA	A=C	A	
	DD	BCex	A	
	06	RSTK=C		
	28	P=	8	
%1				❼
	80F2	CPex	2	
	866	?ST=0	6	❽
	50	GOYES	%2	
	801	OUT=C		
%2				❾
	06	RSTK=C		
	D6	C=A	A	
%3				❿
	CE	C=C-1	A	
	5DF	GONC	%3	
	07	C=RSTK		

❶ On passe au chiffre suivant sur l'écran.
❷ Ajustement du pointeur écran selon la parité du compteur de quartets.
❸ Point de branchement %6.
❹ Passage au quartet suivant puis boucle sur les quartets.
❺ Procédure $BEEP.
❻ Activation du flag triplet (ST4). Nouveau traitement après cascade.
❼ Point de branchement %1.
❽ Bip avec volume normal (sauf si le flag bip RPL est désactivé).
❾ Point de branchement %2.
❿ Point de branchement %3. Temporisation fréquence.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	CD	B=B-1	A	
	56E	GONC	%1	❶
	20	P=	0	
	07	C=RSTK		
	D5	B=C	A	❷
$TEMPO				❸
#171h	D2	C=0	A	
%1				❹
	B36	C=C+1	X	
	520	NOP3		
	59F	GONC	%1	❺
%2				❻
	B36	C=C+1	X	
	520	NOP3		
	59F	GONC	%2	
	01	RTN		

Adresses	Codes hexadécimaux	Renvois
$SPRITES		❼
#187h	F7141 41414 14141 414F7 F755B 655B6 55B65 5B6F7 F7F7F 7F7F7 F7F7F 7F7F7 F794D 514A2 A214D 594F7 F741E 38063 6380E 341F7 F7D59 43677 77369 4D5F7 F7360 0C122 22C10 036F7 E3B65 55555 55555 5B6E3	
$CHIFFRES		❽
#227h	E0111 11111 11111 1E040 60404 04040 4040E 0E011 11804 02010 10F1E 01101 01E00 10111 E080C 0A090 F1808 08080	

❶ Boucle durée.
❷ Restauration de B (colonne courante).
❸ Procédure $TEMPO.
❹ Point de branchement %1.
❺ Double temporisation (les HP-48 G sont plus rapides que les HP-48 S !).
❻ Point de branchement %2.
❼ Procédure $SPRITES. Codage des figures mobiles (sprites du jeu).
❽ Procédure $CHIFFRES. Codage des chiffres.

Adresses	Codes hexadécimaux	Renvois
	F1101 0F001 01011 1E0C0 20101 0F011 1111E 0F101 01804 02020 2020E 01111 11E01 11111 E0E01 11111 E1010 18060	
#2D6h		❶

❶ Fin des routines en #2D6h.

Affectations des registres et affectations des drapeaux

Les versions de XALK destinées aux HP-48 S/SX et G utilisent les registres et les drapeaux de la même façon. Reportez-vous au chapitre présentant le listing de XALK pour HP-48 S/SX. Les affectations des registres et des drapeaux sont présentées juste après les codes hexadécimaux commentés.

Les registres utilisés sont P, A, C, B, D, D0 et D1.

Les drapeaux (flags) utilisés sont : ST0, ST1, ST2, ST3, ST4, ST5, ST6 e ST15.

Zones utilisées en RAM

Les zones utilisées en RAM sont celles normalement réservées à l'interface série RS 232. Il s'agit des adresses comprises entre #80400h et #80600h.

Adresses de départ	Affectations de la RAM RS232C
R0	Table des sprites (figures cubiques)
R1	Table des sprites (chiffres)
#80400h	Vitesse (affichage) variant entre 0 et 200
#80405h	Niveau +3 (4, 5, 6 ou 7) d'où nombre de figures
#80406h	Figure courante (en cours de chute)
#80407h	Figure suivante (affichée en haut à gauche)
#80408h	Vitesse (compteur)
#8040Dh	Score
#8041Dh	Tableau créé par les cascades (taille 15×5)

3
Exécuter le programme

Si ce n'est pas encore fait, créez un répertoire DXALK et placez-y les objets suivants :

- 'XALK', le lanceur en RPL,
- 'CODE', le programme proprement dit.

La chaîne des codes hexadécimaux 'XALK48G' et le programme 'ASS' ne sont absolument pas nécessaires à l'exécution du programme.

Pour lancer le jeu, il suffit d'évaluer XALK (le lanceur RPL). Les deux objets indispensables que sont 'XALK' et 'CODE' devront bien entendu se trouver dans le répertoire DXALK.

4
Comment jouer ?

Des figures arrivent de la gauche de l'écran, à vous de les empiler correctement sur 5 colonnes. Vous aurez perdu lorsqu'une colonne aura atteint toute la largeur de l'écran. Le lanceur RPL vous demande le niveau du jeu, de 1 à 4. Au niveau 4 vous devrez faire face à 8 figures différentes pour 5 colonnes !

Sachez que trois figures identiques disparaissent lorsqu'elles sont alignées horizontalement ou verticalement (les diagonales ne sont pas reconnues). Il existe plusieurs coups spéciaux :

- une cascade s'obtient en éliminant plusieurs triplets à l'aide d'une unique figure. En fait, une figure provoque l'élimination d'un triplet. Cette élimination provoque la chute vers la droite des figures se trouvant à gauche du triplet éliminé. C'est ainsi qu'une seule figure permet d'éliminer plusieurs triplets,
- une combinaison est l'élimination de plusieurs triplets à l'aide d'une unique figure sans avoir pour autant recours à la cascade.

Autrement dit, les figures ne tombent pas. Une combinaison peut par exemple s'obtenir en élimant un triplet horizontal et un triplet vertical à l'aide d'une unique figure,

• cinq figures peuvent disparaître simultanément, à vous de savoir constituer une colonne ou une ligne de cinq figures identiques. Notez qu'une ligne ou une colonne de 4 figures ne permet l'élimination que de 3 figures... C'est 3 ou 5 !

A l'écran apparaissent la prochaine figure (en haut à gauche), le score (en bas à gauche) et la vitesse de chute (en bas à droite).

Au niveau 1, chaque figure posée rapporte 1 point, chaque triplet éliminé rapporte 5 points. Au niveau 4, une figure posée et un triplet éliminé rapportent respectivement 4 et 20 points. Les touches à utiliser pendant le jeu sont les suivantes :

- [α] provoque un déplacement vers le haut,
- [↰] provoque un déplacement vers le bas,
- [÷] lâche le carré dans la colonne,
- [5] interrompt temporairement la partie,
- [2] permet de reprendre la partie,
- [8] sert à quitter le jeu.

Appuyer sur [8] est le seul moyen normal de quitter le jeu. Le son est fonction du flag beep RPL (qui n'est pas modifiable pendant le jeu).

5

Retrouver les fichiers sur la disquette

Les chemins d'accès aux différents fichiers relatifs aux variables associées à ce jeu sont :

- 'XALK48G' : DUNOD_1\G_GX\OB_P_OBG\XALK_G\XALK48G,
- 'CODE' : DUNOD_1\G_GX\OB_P_OBG\XALK_G\CODE,
- 'XALK' : DUNOD_1\G_GX\OB_P_OBG\XALK_G\XALK,
- 'ASS' : DUNOD_1\G_GX\ASS.

35

Seth pour HP-48 G

1
Le jeu

1.1
Présentation

Eviter flaques d'huile et obstacles tout en ramassant les pompes à essence à bord d'une voiture démunie de frein... Tel est le programme que vous propose SETH pendant quinze tours d'une piste longue de 6 km auxquels s'ajoute un final de 10 km ! Les 10 voitures à votre disposition ne seront pas de trop !

1.2
Liste des objets

En plus du code exécutable, SETH nécessite un programme lanceur écrit en RPL. Contrairement à la version pour HP-48 S/SX, le code exécutable de SETH pour HP-48G comprend les objets graphiques (GROB) qui étaient indépendants dans le version S/SX.

Quatre objets sont associés à la mise en œuvre de la version pour HP-48 G de SETH :

- 'SETH48G' est la chaîne de codes hexadécimaux qui devra être traitée avec le programme ASS pour générer le code exécutable 'CODE',
- 'CODE' est le code exécutable générer avec ASS à partir de 'SETH48G',
- 'ASS' est le programme permettant de générer 'CODE' à partir de 'SETH48G',
- 'RPL' est le programme lanceur écrit en RPL.

'SETH48G' et 'ASS' ne sont absolument pas nécessaires pour exécuter le programme. Par contre, 'RPL' (le programme lanceur en RPL) et 'CODE' (le code exécutable) doivent absolument se trouver dans le répertoire DSETH que vous devez avoir à créé.

2
Les objets

2.1
Le lanceur RPL

Le lanceur est un programme écrit en RPL. Il est à enregistrer dans une variable portant le nom 'RPL' (ou 'LANC'). Il doit être saisi entre « et » qui sont les délimiteurs d'un objet de type programme RPL.

```
« CLEAR #90h #0h
PDIM PICT { #18h
#18h } { #0h #0h
} PVIEW
"SETH" 3
→GROB REPL
 -1 WAIT
  IF 16.6 ==
  THEN -7
  ELSE -9
```

```
  END ERASE { #0h
#0h } PVIEW CODE
»
```

Placez dans la pile une copie de l'objet 'RPL' (ou 'LANC', au choix), activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à # BC95h,
- au niveau 1, une longueur égale à 201 octets.

2.2
Le code hexadécimal

a
La chaîne à saisir

Cet objet est à enregistrer dans une variable nommée 'SETH48G'. Saisissez la chaîne de codes hexadécimaux présentée ci-dessous entre " et ". Ne saisissez ni les espaces ni les sauts de ligne.

```
"CCD20 06210 14381 48143 24A29 32C01
42164 808C8 4F808 F0881 08103 14096
65085 92081 B4348 7F00C A1043 34300
EA103 310DE A1023 164EA 10131 0EEA1
00131 1BC01 0015C 019E2 15C01 B0580
814E8 08B35 08521 AD860 14213 030D1
53715 0716F 17F80 8B050 163A0 E54E1
6F16F 16313 6108A F2351 00101 1E004
08696 21711 43329 99932 E017B 14B30
3982C 0D281 B3155 78408 411EF 04032
06515 53AF2 32A0D AFA32 11017 31517
17BCE 56F33 22221 D3B14 534FB 80081
B4C20 61353 10114 D8531 E3140 73871
D3114 B3029 02113 400D0 0CE5D F40E0
71353 13414 D74B6 7D468 6223D 23032
2D730 5D530 8801D 2801D 4CC5D F818F
9F57E CF5DD 20347 83001 10CA1 3014A
1E214 01590 8EE89 01B43 10084 3146C
65508 53307 D5110 130D2 31681 52781
C158E 16ECE 5FE7C 16320 408FC EE108"
```

```
08A05 08528 08A25 08423 28008 FCEE1
0808A 280D2 81B38 08A09 21B00 100D2
15C03 20048 FCEE1 0808A FFE30 815C0
90913 1DD0D 015B0 86180 818F0 83400
C00F0 C4C4E 2CE5D FCD65 4F175 31111
7B15B B1CB1 59B17 BA6E5 BE1D2 114B7
C271B 83100 1DFDA F2155 71CB1 4B14E
90AF0 E4808 A060C CCC17 B1491 C914B
E4808 A160C CCC17 B1491 C114F EA305
9826A 30798 6E917 31371 35D7D 1A881
DD0D2 15F08 1B463 40000 09000 70009
A00B0 009CF 0A000 9B007 C009A 00BF0
09C00 7AB09 CF079 B09AB FE6C6 C6CA1
31147 0614A 90471 DB808 6440C 91353
0615D 014A9 6471D B8086 440C9 13530
815D0 CDCDE 71EFD 4014F 17114 BEA30
6982A 214A8 08720 28087 391D2 15E08
B59FC B1353 0515D 02081 507D5 30314
A8084 28084 3A0C4 70F55 7F14A 80873
74310 A9E6E 31564 9C59F D2812 C3816
DB135 D915D 030F9 05813 0715D 01709
45501 C115D 01E01 4014B A6C14 95453
18414 D1D3E 37222 22222 15D71 D5014
B3161 962B2 05C4D 6C4C2 041C4 14B96
6411D 3E373 33333 3315D 7110D 23122
CA320 E7C2D 7D813 03060 6D278 1307A
0E52F 34021 001DE 014BA 0C490 C1E35
5FD41 30D01 5B01D FD14F EA343 E408A
62D51 4B171 14FB6 21711 33A6A DC131
13313 1313E 9E2D0 9E080 14F55 03017
A9217 01361 348B3 3D844 32800 8FCEE
10808 A5018 43854 86050 85332 4008F
CEE10 808A5 51874 91843 85487 05085
38646 07592 1E014 014B3 1049E 67117
614B3 02902 D0E69 02606 AE084 1D81D
0014B 05B64 04149 5606F C6303 901C2
D231F 7D5D7 2880F 28625 0801D 4CC5D
FCF5A E206B 90174 14B05 E4041 49840
17714 BE415 90304 902A0 30E90 63417
0D015 B0D6E 41590 110F2 F2CAF 6C6CA
13072 A1110 C4136 EA344 E600C A1307
88186 262D2 30ED5 311FD 72880 F2801
D4CC5 DFCF5 FE201 D7015 B5AF2 30705
A7A04 15953 45110 0110E A1307 43016
```

```
C1E00 40301 74201 641E5 04030 17410
16B1E 20403 02740 06D3A D715F 5AF51
11D2A 89EAF 281EC A1393 0614B 15801
6F16F 16317 0A0E5 9E132 D231D FEA13
0BF5A 0F5BB 01110 345E4 00C20 61341
13131 30C14 31401 6F16F 16317 3A0E5
AE320 048FC EE108 08AFF E0713 4D030
C1401 6F16F 163A0 E50F0 13406 30011
0CA13 0304D A1370 6D2A8 6F211 2CA13
13170 14B14 816F1 6F163 171CE 5BE07
13501 D2311 1D732 E11D5 D273B F136E
9134C F5FE0 13456 20011 08637 0325A
4CA13 014A7 7A232 06311 0CA13 0D271
7F1D3 13132 14BE4 86360 CCCC1 49171
808B0 50177 A6E5E D1DE0 11034 02100
D5873 B0320 D7CAF 914F0 6DDCA A0D5A
F131E A1300 7D731 F8D53 142E1 A0F5A
F1537 15078 73A11 8F1CF A6D59 E17F1
37134 5E016 F17FA 6D51D D21FE 04081
5F0D7 17631 1106D B8637 0306E B14B1
C1B6A A6448 214BB 62808 B7B11 33D8A
6A131 17314 FD413 15503 01796 E17D1
3234E 1100E A1300 7CE5F 93458 30011
0CA13 014A3 02906 93870 80850 55084
08623 2D231 F8D5D 22880 F2801 D4CC5
DFCD5 FE204 16304 90680 8515B C3079
06F41 D2014 705E6 04155 3D230 80630
FD5D7 2880F 28625 0801D 4CC5D FCF5A
E80F2 CE5DF 07CE5 FC6D8 D30C9 06D51
D7015 B5AF2 31040 5A7A0 41595 17714
BB041 59043 88627 C31FF D5315 3D732
00880 16300 D2801 D4CC5 DFCF5 5E4B9
30F90 63979 8D758 D8620 0D1D2 B2530
FD728 80F28 01D4C C5DF8 1861F 5BECD
80F28 01D4C C5DF8 1869F 5BECF 5DC20
01808 70008 08610 03029 02001 10340
6300C A1303 0671B C31D7 D5110 13015
27BFC 15071 6F310 AA6E5 CFA6D 54ED0
D1D2D 3CC28 B6545 1B675 6FD7D 6DFC7
C75A0 D8A6C 42180 F2862 50801 62DFD
5D280 1CD5D F31F1 D531C F1101 3015A
7A741 58716 7A6E5 EEA6E 5CFA6 D58D1
E2040 14305 A3C04 15134 D1110 34101
00EA1 30E67 B1B8C 584F3 4B400 08627
```

CCD 20

```
481B4 CA131 32005 A3E5C F1439 08B21
7214F 171D7 D2288 0F280 1D8A3 D5CFA
6F5DE 2046C 721BD 281B3 1010F 1010A
10105 1010F 7D00A 5E005 5E00A 10105
1010A E0105 101FF 07D00 67D00 37D00
3F110 27D00 3DA00 37D00 90110 13011
41313 183D5 3200C A3E5C F15BB 158B1
6F16F 16317 BA6D5 DD343 80006 11F77
73707 77777 00550 73073 00527 70737
11551 25555 55005 72520 52005 21125
25115 51255 55550 05725 20520 05211
25257 15370 55555 50035 05275 20052
37052 54155 12555 55500 55252 05200
52142 52541 55125 55555 00552 52052
00221 42525 77757 07777 77005 50770
77002 27707 77755 55573 22222 77447
11774 46447 55574 44711 74477 11755
77442 22275 57557 75574 47000 00000
00000 000FF BA5DB A5DBA 5DFFF 000F0
00F00 0F000 F0F0F 0F0F0 F0F0F 0F0FE
64BFB F64FE F0707 E24FF FF247 E7050
7621E DB762 3660E 7FE9B FFF93 FFDE7
DBE7B DFFE7 42E79 98D9F F7E3C 7CFEE
6C64F C99BA 9AB49 0F0C3 24DBD BDBDB
24C38 14266 99996 64281 FFF71 00493
15543 55435 54555 45554 95549 55415
93151 004FF F7FFF FFFFF FFF71 00000
00000 41000 8FF00 00410 00700 70004
100E0 00830 04108 10000 C0041 06000
00030 41010 00000 40418 00000 00804
14000 00000 14140 00000 00141 20000
00002 41200 00000 02411 00000 00044
11000 00000 44110 00000 00441 10000
00004 41300 00000 06413 00000 00064
17000 00000 74160 00F70 00341 A00C0
81082 41210 20020 42414 60100 40314
14811 004C0 14180 E200A 38041 01070
07040 41010 8FF00 40410 20000 00204
10400 00001 04108 10000 C0041 00200
00200 4100C 00081 00410 00700 70004
1000C FF100 04100 04001 00041 00080
80000 41000 80800 00410 00808 00004
10000 14000 04100 00140 00041 00081
C0000 41000 E1C30 00410 00F1C 70004
```

```
1000F 1C700 04100 8F3EF 00041 008FF
FF000 41008 FFFF0 00410 00000 00004
99E21 0D5D1 449A2 61055 50449 9EE10
55D14 49A2A 14555 00499 E21CD DD144
10000 00000 04FFF FFFFF FFF7
```

Placez dans la pile une copie de l'objet, activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à # 68B0h,
- au niveau 1, une longueur égale à 4709 octets.

Pour générer 'CODE' (le code exécutable) à partir de 'SETH48G', procédez de la façon suivante :

- placez la chaîne de codes hexadécimaux (variable 'SETH48G') au niveau 1 de la pile **"CCD200621014381...** s'affiche,
- exécuter le programme ASS,
- au niveau 1 de la pile doit apparaître **Code**,
- si ce n'est pas encore fait, placez-vous dans le répertoire DSETH (à créer s'il n'existe pas déjà),
- saisissez le nom **'CODE'**,
- appuyez sur la touche [STO].

Placez dans la pile une copie de l'objet code exécutable (variable 'CODE'), activez la commande BYTES et vérifiez que vous obtenez :

- au niveau 2, une somme de contrôle égale à # F350h,
- au niveau 1, une longueur égale à 2354,5 octets.

Codes commentés

Adresses	Codes hexadécimaux	Mnémoniques	Renvois
$$			❶
	CCD20		❷
#000h	**06210**		❸
	143	**A=DAT1 A**	❹
	814	**ASRC**	

❶ Procédure **$$** d'initialisation.
❷ Prologue d'un objet de type code exécutable.
❸ Longueur hors prologue du code.
❹ Lecture du premier niveau de la pile RPL.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	814	ASRC		
	324A2	C=(3)	#2A4	
	932	?A=C	X	❶
	C0	GOYES	%1	❷
	142	A=DAT0	A	
	164	D0=D0+	#5	
	808C	PC=(A)		
%1				❸
#022h	84F	ST=0	15	
	808F	INTOFF		
	08	CLRST		❹
	810	ASLC		
	810	ASLC		
	3140	C=(2)	#04	❺
	966	?A≠C	B	
	50	GOYES	%2	
	859	ST=1	9	❻
%2				❼
#03Dh	20	P=	0	
	81B4	A=PC		
	3487F00	C=(5)	#00F78	
	CA	A=A+C	A	
	104	R4=A		❽
	334300	C=(4)	#0034	
	EA	A=A-C	A	
	103	R3=A		❾
	310D	C=(2)	#D0	
	EA	A=A-C	A	
	102	R2=A		❿
	3164	C=(2)	#46	

❶ Test : a-t-on -7 ou -9 au niveau 1 de la pile RPL ?
❷ Si le test est négatif, on retourne au RPL.
❸ Point de branchement %1. Inhibition des interruptions.
❹ ST2 = 0 (flag son).
❺ Test : a-t-on -7 au premier niveau de la pile RPL ?
❻ Si le test est positif (ST9 = 1) on reprend une partie en cours.
❼ Point de branchement %2.
❽ Désignation (R4) du sprite « coupe de fin ».
❾ Désignation (R3) du sprite « ON ».
❿ Désignation (R2) des sprites objets.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	EA	A=A-C	A	
	101	R1=A		❶
	310E	C=(2)	#E0	
	EA	A=A-C	A	
	100	R0=A		❷
	131	D1=A		
	1BC0100	D0=(5)	#0010C	❸
	15C0	DAT0=C	1	
	19E2	D0=(2)	#2E	❹
	15C0	DAT0=C	1	
	1B05808	D0=(5)	#80850	
	14E	C=DAT0	B	
	808B3	?CBIT=1	3	❺
	50	GOYES	%4	
	852	ST=1	2	
%4				❻
#09Fh	1AD860	D0=(4)	#068D	
	142	A=DAT0	A	
	130	D0=A		
	30D	C=(1)	#D	
%5				❼
#0AEh	1537	A=DAT1	W	
	1507	DAT0=A	W	
	16F	D0=D0+	#10	
	17F	D1=D1+	#10	❽
	808B0	?CBIT=1	0	
	50	GOYES	%6	❾
163	D0=D0+	#4		
%6				❿

❶ Désignation (R1) des sprites chiffres.
❷ Désignation (R0) des sprites score. Sauvegarde de R0.
❸ Suppression du témoin « sablier ».
❹ Toujours moins d'interruptions... Lecture du flag beep RPL.
❺ ST2 = 1 si le son est autorisé (flag RPL activé).
❻ Point de branchement %4. D0 désigne la bitmap écran.
❼ Point de branchement %5. Affiche 16 quartets du sprite score.
❽ Données suivantes...
❾ Si le compteur C est pair, on passe à la ligne suivante.
❿ Point de branchement %6.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#0C6h	A0E	C=C-1	P	
	54E	GONC	%5	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❶
	163	D0=D0+	#4	
	136	CD0ex		❷
	108	R0=C		
	AF2	C=0	W	
	35100101	C=(6)	#101001	❸
	1E0040	D1=(4)	#0400	
	869	?ST=0	9	
	62	GOYES	%7	❹
	171	D1=D1+	#2	
	143	A=DAT1	A	
	32999	C=(3)	#999	❺
	932	?A=C	X	
	E0	GOYES	%7	
	17B	D1=D1+	#C	
	14B	A=DAT1	B	
	303	C=(1)	#3	❻
	982	?A<C	P	
	C0	GOYES	$0	
%7				❼
#10Fh	D2	C=0	A	
	81B3	PC=C		
%8				❽
#115h	1557	DAT1=C	W	
$0				❾
#119h	840	ST=0	0	❿

❶ On saute une ligne puisque l'écran a été effacé par le lanceur.
❷ R0 pointe sur le début de la 9ème ligne.
❸ Champ de vision initial (km=01, vies=005, tour=01, score=000000).
❹ Test : veut-on reprendre une partie ?
❺ Test : avait-on perdu pendant la précédente partie ? Oui, alors reset.
❻ Test : le champ de vision a-t-il été détérioré (transmission IR) ?
❼ Point de branchement %7. Reset si le test est positif.
❽ Point de branchement %8. Sauvegarde des données.
❾ Procédure $0 (Départ).
❿ Désactivation inverseur (ST0) et désactivation accélérateur (ST1).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	841	ST=0	1	
	1EF040	D1=(4)	#040F	
	32065	C=(3)	#560	❶
	1553	DAT1=C	X	
	AF2	C=0	W	
	32A0D	C=(3)	#D0A	❷
	AFA	A=C	W	
	32110	C=(3)	#011	
	173	D1=D1+	#4	❸
%1				❹
#141h	1517	DAT1=A	W	
	17B	D1=D1+	#A	
	CE	C=C-1	A	❺
	56F	GONC	%1	
	332222	C=(4)	#2222	❻
	1D3B	D1=(2)	#B3	
	145	DAT1=C	A	
#15Ah	34FB800	C=(4)	#008BF	
	81B4	A=PC		❼
	C2	C=C+A	A	
	06	RSTK=C		
	135	D1=C		❽
	3101	C=(2)	#10	
	14D	DAT1=C	B	
	853	ST=1	3	
	1E3140	D1=(4)	#0413	❾
%2				❿
#17Ch	7387	GOSUB	$APPUI2	
#180h	1D31	D1=(2)	#13	
	14B	A=DAT1	B	

❶ Compteur de kilomètres valant #56h et compteur essence valant #0h.
❷ 1ère case (coordonnée 2), #0Ah. Dernière case (coordonnée 5), #0Dh.
❸ On pointe sur le début du tableau.
❹ Point de branchement %1. Création d'un contenu vide.
❺ Colonne suivante.
❻ Création d'une ligne de départ dans le tableau.
❼ On pointe sur la procédure $APPUI.
❽ 01:RTN (on court-circuite l'affichage de la voiture).
❾ On pointe sur les données.
❿ Point de branchement %2. On décale tous les pointeurs du tableau.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	302	C=(1)	#2	❶
	902	?A=C	P	
	11	GOYES	%4	
	3400D00	C=(5)	#00D00	
%3				❷
#196h	CE	C=C-1	A	
	5DF	GONC	%3	
	40E	GOC	%2	
%4				❸
#19Eh	07	C=RSTK		
	135	D1=C		
	3134	C=(2)	#43	
	14D	DAT1=C	B	
	74B6	GOSUB	$VOITURE	❹
#1AEh	7D46	GOSUB	$ON	❺
#1B2h	862	?ST=0	2	
	23	GOYES	$1	❻
	D2	C=0	A	
	303	C=(1)	#3	❼
	22	P=	2	
	D7	D=C	A	
%7				❽
#1C0h	3054	C=(1)	#5	
	D5	B=C	A	
%8				❾
#1C5h	308	C=(1)	#8	
	801	OUT=C		
	D2	C=0	A	
	801	OUT=C		
	D4	A=B	A	❿

❶ Test : la route est-elle centrée ?
❷ Point de branchement %3. Petite temporisation.
❸ Point de branchement %4. Si la route est centrée, réactivation de $APPUI2.
❹ Affichage de la voiture.
❺ Attente d'une pression sur la touche [ON].
❻ Test : bip autorisé ?
❼ Oui, alors 4 bips !
❽ Point de branchement %7. Fréquence dans B.
❾ Point de branchement %8. Bip au volume atténué.
❿ Temporisation fréquence.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%9				❶
#1D2h	CC	A=A-1	A	
	5DF	GONC	%9	
	818F9F	B=B-#10	A	❷
	57E	GONC	%8	
	CF	D=D-1	A	❸
	5DD	GONC	%7	
	20	P=	0	
$1				❹
#1E7h	3478300	C=(5)	#00387	
	110	A=R0		
	CA	A=A+C	A	
	130	D0=A		❺
	14A	A=DAT0	B	
	1E2140	D1=(4)	#0412	
	1590	DAT1=A	1	❻
	8EE890	GOSUBL	#SCRASH	
#209h	1B43100	D0=(5)	#00134	
	843	ST=0	3	
	146	C=DAT0	A	❼
	C6	C=C+C	A	
	550	GONC	%1	
	853	ST=1	3	
				❽
#21Eh	307	C=(1)	#7	❾
	D5	B=C	A	
%1				❿
#223h	110	A=R0		
	130	D0=A		

❶ Point de branchement %9.
❷ Boucle durée (plus aigu).
❸ Le son est produit 5 fois.
❹ Procédure $1 (scrolling).
❺ Test : collision sur la droite ?
❻ Stockage du quartet représentatif du sprite à droite de la voiture.
❼ ST3 prend une valeur aléatoire en fonction de #00138 (horloge).
❽ Début de traitement du défilement de l'écran.
❾ 8 décalages.
❿ Point de branchement %1. On pointe en bas à droite de l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D2	C=0	A	
	3168	C=(2)	#86	❶
%2				❷
#22Fh	1527	A=DAT0	W	
	81C	ASRB		❸
	158E	DAT0=A	15	
	16E	D0=D0+	#F	
	CE	C=C-1	A	
	5FE	GONC	%2	
	7C16	GOSUB	$VOITURE	❹
#246h	32040	C=(3)	#040	
	8FCEE10	GOSBVL	#01EEC	❺
	808A0	?CBIT=0	0	
	50	GOYES	%3	
	852	ST=1	2	❻
%3				❼
#25Ch	808A2	?CBIT=0	2	
	50	GOYES	%4	
	842	ST=0	2	❽
%4				❾
#266h	32800	C=(3)	#800	
	8FCEE10	GOSBVL	#01EEC	
	808A2	?CBIT=0	2	
	80	GOYES	%5	
	D2	C=0	A	
	81B3	PC=C		
%5				❿
#27Fh	808A0	?CBIT=0	0	
	92	GOYES	%7	

❶ Compteur de #87h × #15d quartets.
❷ Point de branchement %2.
❸ Scrolling vers la gauche.
❹ Affichage de la voiture.
❺ Test : pression sur [→] ?
❻ Si oui, activation du flag son (ST2).
❼ Point de branchement %3. Test : pression sur [←] ?
❽ Si oui, désactivation du flag son (ST2).
❾ Point de branchement %4. Test : pression sur [8] ? Si oui, reset.
❿ Point de branchement %5. Test : pression sur [÷] ? Si oui, pause.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	1B00100	D0=(5)	#00100	
	D2	C=0	A	❶
	15C0	DAT0=C	1	
%6				❷
#293h	32004	C=(3)	#400	
	8FCEE10	GOSBVL	#01EEC	
	808AF	?CBIT=0	15	❸
	FE	GOYES	%6	
	308	C=(1)	#8	
	15C0	DAT0=C	1	
%7				❹
#2ADh	909	?B=0	P	
	13	GOYES	%10	
	1DD0	D1=(2)	#0D	
	D0	A=0	A	
	15B0	A=DAT1	1	❺
	861	?ST=0	1	
	80	GOYES	%8	❻
	818F08	A=A+#9	A	
%8				❼
#2C7h	3400C00	C=(4)	#00C00	
	F0	ASL	A	❽
	C4	A=A+A	A	
	C4	A=A+A	A	
	E2	C=C-A	A	
%9				❾
#2D6h	CE	C=C-1	A	
	5DF	GONC	%9	
	CD	B=B-1	A	❿
	654F	GOTO	%1	

❶ Extinction de l'écran en cas de pression sur [÷] ?
❷ Point de branchement %6.
❸ Attente d'une pression sur [ON]. Allumage de l'écran.
❹ Point de branchement %7. Pas de temporisation si 8ème décalage.
❺ Lecture du numéro du tour.
❻ Test : accélérateur ?
❼ Point de branchement %8.
❽ Calcul de la temporisation selon numéro du tour et accélérateur.
❾ Point de branchement %9. Temporisation entre deux décalages.
❿ Décalage suivant.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%10				❶
#2E1h	175	D1=D1+	#6	
	3111	C=(2)	#11	
%11				❷
#2E8h	17B	D1=D1+	#4C	
	15BB	A=DAT1	12	
	1CB	D1=D1-	#C	
	159B	DAT1=A	12	❸
	17B	D1=D1+	#C	
	A6E	C=C-1	B	
	5BE	GONC	%11	
	1D21	D1=(2)	#12	
	14B	A=DAT1	B	❹
	7C27	GOSUB	$COLLISION	
$2				❺
#30Ah	1B83100	D0=(5)	#00138	❻
	1DFD	D1=(2)	#DF	
	AF2	C=0	W	❼
	1557	DAT1=C	W	
	1CB	D1=D1-	#C	
	14B	A=DAT1	B	❽
	14E	C=DAT0	B	
	90A	?C=0	P	
	F0	GOYES	%1	❾
	E4	A=A+1	A	
	808A0	?CBIT=0	0	❿
	60	GOYES	%1	
	CC	A=A-1	A	
	CC	A=A-1	A	

❶ Point de branchement %10. On pointe sur le début du tableau.
❷ Point de branchement %11. Lecture de l'une des 18 colonnes.
❸ Transfert des colonnes vers la gauche. Boucle sur 18 colonnes.
❹ Test : collision à l'emplacement de la voiture ?
❺ Procédure $2 : création de la 18ème colonne.
❻ D0 pointe sur l'horloge.
❼ Suppression de la 18ème colonne dans le tableau.
❽ Lecture de l'ancienne position relative à R0 (bord de la route).
❾ Tendance à monter...
❿ Probabilité de 1/2 pour choix entre montée et descente...

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%1				❶
#337h	17B	D1=D1+	#C	
	149	DAT1=A	B	
	1C97	D1=D1-	#A	
	14B	A=DAT1	B	❷
	E4	A=A+1	A	
	808A1	?CBIT=0	1	❸
	60	GOYES	%2	
	CC	A=A-1	A	
	CC	A=A-1	A	
%2				❹
#350h	17B	D1=D1+	#C	
	149	DAT1=A	B	
	1C1	D1=D1-	#2	
	14F	C=DAT1	B	❺
	EA	A=A-C	A	
	305	C=(1)	#5	
	982	?A<C	P	
	6A	GOYES	$2	❻
	307	C=(1)	#7	
	986	?A>C	P	❼
	E9	GOYES	$2	
#36Eh	173	D1=D1+	#4	❽
	137	CD1ex		
	135	D1=C		
	D7	D=C	A	❾
	D1	B=0	A	❿
	A88	B=A	P	

❶ Point de branchement %1. Stockage de la nouvelle position R0.
❷ Lecture de l'ancienne seconde position.
❸ Probabilité de 1/2.
❹ Point de branchement %2. Stockage de la nouvelle seconde position.
❺ Lecture de la largeur de la route.
❻ Si la largeur est hors norme...
❼ ...nouveau calcul des positions.
❽ On pointe sur la première case de la 18ème colonne.
❾ Pointeur D1 dans D.
❿ On place la largeur de la colonne dans B.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
#37Eh	1DD0	D1=(2)	#0D	
	D2	C=0	A	❶
	15F0	C=DAT1	1	
	81B4	A=PC		❷
	6340	GONC	%3	
#390h	00009	*Données tableau*		❸
	00070			
	009A0			
	0B000			
	9CF0A			
	0009B			
	007C0			
	09A00			
	BF009			
	C007A			
	B09CF			
	079B0			
	9ABF			
%3				❹
#3D0h	E6	C=C+1	A	
	C6	C=C+C	A	
	C6	C=C+C	A	
	CA	A=A+C	A	
	131	D1=A		
	147	C=DAT1	A	❺
	06	RSTK=C		
#3E0h	14A	A=DAT0	B	
	904	?A≠B	P	❻
	71	GOYES	%5	
	DB	C=D	A	❼
	80864	?ABIT=0	4	

❶ Lecture du numéro du tour.
❷ Calcul du PC.
❸ Début du tableau.
❹ Point de branchement %3. Calcul d'une position dans le tableau.
❺ On place dans RSTK les types de sprites à afficher.
❻ Probabilité = 1/16...
❼ ...soit en haut...

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	40	GOYES	%4	❶
	C9	C=C+B	A	
%4				❷
#3F3h	135	D1=C		
	306	C=(1)	#6	❸
	15D0	DAT1=C	1	
%5				❹
#3FDh	14A	A=DAT0	B	
	964	?A≠0	B	❺
	71	GOYES	%7	
	DB	C=D	A	
	80864	?ABIT=0	4	
	40	GOYES	%6	❻
	C9	C=C+B	A	
%6				❼
#410h	135	D1=C		
	308	C=(1)	#8	
	15D0	DAT1=C	1	
%7				❽
#41Ah	CD	B=B-1	A	
	CD	B=B-1	A	
	E7	D=D+1	A	
	1EDB40	D1=(4)	#04BD	
	14F	C=DAT1	A	❾
	171	D1=D1+	#1	
	14B	A=DAT1	A	
	EA	A=A-C	A	
	306	C=(1)	#6	❿

❶ ...soit en bas ! Ce choix s'effectue selon l'horloge.
❷ Point de branchement %4.
❸ Stockage de la voiture détruite dans le contenu de la 18ème colonne.
❹ Point de branchement %5.
❺ Probabilité = 1/256... soit en haut (début contenu)...
❻ ...soit en bas (fin contenu), selon l'horloge.
❼ Point de branchement %6. Stockage d'une vie dans la 18ème colonne.
❽ Point de branchement %7. On ne se trouve pas au bord de la route.
❾ Calcul de la largeur (calcul de la différence des positions).
❿ On fixe la largeur minimum à 5 positions possibles.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	982	?A<C	B	
	A2	GOYES	%9	
	14A	A=DAT0	B	
	80872	?ABIT=1	2	❶
	02	GOYES	%9	
	80873	?ABIT=1	3	
	91	GOYES	%9	
	D2	C=0	A	
%8				❷
#44Ch	15E0	C=DAT0	1	
	8B5	?B<C	A	
	9F	GOYES	%8	
	CB	C=C+D	A	❸
	135	D1=C		
	305	C=(1)	#5	❹
	15D0	DAT1=C	1	
%9				❺
#461h	20	P=0		
	815	BSRC		❻
	07	C=RSTK		
	D5	B=C	A	❼
	303	C=(1)	#3	
%10				❽
#46Dh	14A	A=DAT0	B	
	80842	ABIT=0	2	❾
	80843	ABIT=0	3	
%11				❿
#47Ah	A0C	A=A-1A	P	
	470	GOC	%12	

❶ Pseudo-hasard.
❷ Point de branchement %8. La voiture adverse doit être sur la route...
❸ On calcule la place dans le contenu.
❹ Mémorisation de la position (voiture adverse) dans la 18ème colonne.
❺ Point de branchement %9.
❻ Largeur placée dans B(S).
❼ Sprites à afficher placés dans B(A).
❽ Point de branchement %10. Calcul d'un chiffre entre 0 et 3...
❾ ...selon l'horloge (dans A).
❿ Point de branchement %11. Copie de ce chiffre dans B (P).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	F5	BSR	A	
	57F	GONC	%11	
%12				❶
#485h	14A	A=DAT0	B	
	80873	?ABIT=1	3	
	74	GOYES	%15	
	310A	C=(2)	#A0	
	9E6	?A>C	B	
	E3	GOYES	%15	
%13				❷
#498h	1564	C=DAT0	S	
	9C5	?B<C	S	
	9F	GOYES	%13	
	D2	C=0	A	
	812	CSLC		
	C3	D=D+C	A	❸
	816	CSRC		
	DB	C=D	A	
	135	D1=C		
	D9	C=B	A	❹
	15D0	DAT1=C	1	
	30F	C=(1)	#F	❺
	905	?B≠C	P	
	81	GOYES	%15	
	307	C=(1)	#7	❻
	15D0	DAT1=C	1	
	170	D1=D1+	#1	❼
	945	?B≠C	S	❽
	50	GOYES	%14	
	1C1	D1=D1-	#2	

❶ Point de branchement %12. Diminution de la fréquence d'apparition (A < #A0h et bit 3 annulé).
❷ Point de branchement %13. On calcule la place du sprite dans le contenu de la 18ème colonne en fonction de la largeur.
❸ On pointe dans le tableau.
❹ Mémorisation du sprite dans le contenu de la 18ème colonne.
❺ Test : s'agissait-il d'un double obstacle ?
❻ Si oui, on stocke un obstacle simple.
❼ Calcul de l'emplacement du second obstacle.
❽ Obstacle simple.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%14				❶
#4D0h	15D0	DAT1=C	1	
%15				❷
#4D4h	1E0140	D1=(4)	#0410	
	14B	A=DAT1	B	
	A6C	A=A-1	B	
	149	DAT1=A	B	
	545	GONC	$3	❸
	3184	C=(2)	#48	
	14D	DAT1=C	B	❹
	1D3E	D1=(2)	#E3	
	3722222222	C=(8)	#22222222	
	15D7	DAT1=C	8	❺
	1D50	D1=(2)	#05	
	14B	A=DAT1	B	❻
	3161	C=(2)	#16	
	962	?A=C	B	❼
	B2	GOYES	$3	
	05	SETDEC		
	C4	A=A+A	A	❽
	D6	C=C	A	
	C4	A=A+A	A	
	C2	C=C+A	A	
	04	SETHEX		
	1C4	D1=D1-	#5	❾
	14B	A=DAT1	B	
	966	?A≠C	B	
	41	GOYES	$3	
	1D3E	D1=(2)	#E3	❿

❶ Point de branchement %14. Stockage du second obstacle simple.
❷ Point de branchement %15. Décrémentation du compteur kilométrique.
❸ Test : fin d'un kilomètre ?
❹ Initialisation du compteur de kilomètres à #48h.
❺ Stockage de la ligne de fin de kilomètre dans le contenu de la colonne.
❻ Lecture du numéro du tour.
❼ Pas de fin de tour si c'est le 16 ème tour (final).
❽ Calcul tour × #6h.
❾ Test : le numéro du tour est-il égal à un multiple de 6 ?
❿ Si le test est positif on stocke une ligne de fin de tour dans la colonne n°18 par dessus la ligne de fin de kilomètre.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	3733333333	C=(8)	#33333333	
	15D7	DAT1=C	8	
$3				❶
#538h	110	A=R0		
	D2	C=0	A	
	3122	C=(2)	#22	❷
	CA	A=A+C	A	
	320E7	C=(3)	#7E0	
	C2	C=C+A	A	❸
	D7	D=C	A	
	D8	B=A	A	
	130	D0=A		❹
	306	C=(1)	#1	
%1				❺
#554h	06	RSTK=C		
	D2	C=0	A	
	7813	GOSUB	$SPR	
#55Ch	07	C=RSTK		
	A0E	C=C-1	P	❻
	52F	GONC	%1	
	3402100	C=(5)	#00120	❼
	1DE0	D1=(2)	#0E	
	14B	A=DAT1	B	❽
%2				❾
#572h	A0C	A=A-1	P	
	490	GOC	%3	
	C1	B=B+C	A	❿
	E3	D=D-C	A	
	55F	GONC	%2	

❶ Procédure $3 (Affichage de la 18ème colonne).
❷ B pointe sur la 1ère case de la 18ème colonne de l'écran de jeu.
❸ D pointe sur la dernière case de la 18ème colonne de l'écran de jeu.
❹ On pointe en haut de l'écran.
❺ Point de branchement %1. Affichage d'un blanc.
❻ On reproduit 7 fois le motif (toute la colonne).
❼ Décalage entre deux cases d'une colonne de l'écran de jeu.
❽ Lecture du champ de vision.
❾ Point de branchement %2.
❿ Si le champ de vision est réduit, case sous B et case au-dessus de D.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%3				❶
#57Fh	D4	A=B	A	
	130	D0=A		
#584h	D0	A=0	A	
	15B0	A=DAT1	1	❷
	1DFD	D1=(2)	#DF	
	14F	C=DAT1	B	❸
	EA	A=A-C	A	
	343E408	C=(5)	#804E3	
	A62	C=C+A	B	❹
	D5	B=C	A	
	14B	A=DAT1	B	❺
	171	D1=D1+	#2	
	14F	C=DAT1	B	❻
	B62	C=C-A	B	
	171	D1=D1+	#2	
	133	AD1ex		
	A6A	A=A+C	B	❼
	DC	ABex	A	
	131	D1=A		
%4				❽
#5B9h	133	AD1ex		
	131	D1=A		
	313E	C=(2)	#E3	❾
	9E2	?A<C	B	
	D0	GOYES	%5	❿
	9E0	?A>B	B	
	80	GOYES	%5	

❶ Point de branchement %3. Initialisation de D0.
❷ Lecture du champ de vision.
❸ Lecture de la position de la 1ère case (selon R0).
❹ B pointe sur la 1ère case du champ de vision.
❺ Lecture de la 1ère case de la 18ème colonne.
❻ Lecture de la position de la dernière case (on en déduit la largeur).
❼ Fin du contenu (B=1ère case contenu, A=1ère case champ de vision).
❽ Point de branchement %4. D1 désigne la case à afficher à l'écran.
❾ 1ère case de la colonne.
❿ Tests : est-on encore dans le contenu ? n'y est-on plus ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	14F	C=DAT1	B	❶
	550	GONC	%6	
%5				❷
#5D3h	301	C=(1)	#1	
%6				❸
#5D6h	7A92	GOSUB	$SPR	
#5DAh	170	D1=D1+	#1	
	136	CD0ex		
	134	D0=C		
	8B3	?C<D	A	❹
	3D	GOYES	%4	
$4				❺
#5E8h	844	ST=0	4	❻
	32800	C=(3)	#008	
	8FCEE10	GOSBVL	#01EEC	
	808A5	?CBIT=0	5	
	01	GOYES	%1	
	843	ST=0	3	❼
	854	ST=1	4	
	860	?ST=0	0	
	50	GOYES	%1	
	853	ST=1	3	
%1				❽
#60Ch	32400	C=(3)	#004	
	8FCEE10	GOSBVL	#01EEC	
	808A5	?CBIT=0	5	❾
	51	GOYES	%2	
	874	?ST=1	4	❿
	91	GOYES	$5	

❶ Si les tests sont négatifs, on lit la case dans le tableau.
❷ Point de branchement %5. Hors du contenu ? alors affichage d'un bord.
❸ Point de branchement %6. Affichage d'un sprite. D1 = case suivante.
❹ Boucle si D0 est encore dans le champ de vision.
❺ Procédure $4 (test des touches).
❻ Pas de pression sur les touches... Test : pression sur [α] ?
❼ Oui ? alors déplacement vers le haut et activation du flag ST4.
❽ Point de branchement %1. Si inverseur, déplacement vers le bas.
❾ Test : pression sur [←] ?
❿ Test : avait-on déjà appuyé sur [α] ? Si oui, pas de déplacement.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	843	ST=0	3	❶
	854	ST=1	4	
	870	?ST=1	0	
	50	GOYES	%2	❷
	853	ST=1	3	
%2				❸
#632h	864	?ST=0	4	
	60	GOYES	$5	
	7592	GOSUB	$APPUI	
$5				❹
#63Bh	1E0140	D1=(4)	#0410	❺
	14B	A=DAT1	B	
	3104	C=(2)	#40	
	9E6	?A>C	B	❻
	71	GOYES	%1	
	176	D1=D1+	#7	
	14B	A=DAT1	B	
	302	C=(1)	#2	
	902	?A=C	P	❼
	D0	GOYES	%2	
	E6	C=C+1	A	
	902	?A=C	P	
	60	GOYES	%2	
%1				❽
#661h	6AE0	GOTO	$6	
%2				❾
#665h	841	ST=0	1	
	D8	B=A	A	❿
	1D00	D1=(2)	#00	

❶ Déplacement vers le haut (flag ST4 activé).
❷ En cas d'absence d'inverseur, on maintient le déplacement vers le bas.
❸ Point de branchement %2. Pression sur [α] ou sur [↰] ?
❹ Procédure $5 (kilomètre suivant et/ou tour suivant).
❺ Lecture du compteur de kilomètres.
❻ Test : a-t-on une ligne délimitant un kilomètre ?
❼ Test : a-t-on une ligne de fin de kilomètre dans le tableau ?
❽ Point de branchement %1.
❾ Point de branchement %2. inhibition de l'accélérateur.
❿ Sauvegarde de l'identification de la fin de kilomètre ou de tour.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	14B	A=DAT1	B	
	05	SETDEC		
	B64	A=A+1	B	❶
	04	SETHEX		
	149	DAT1=A	B	
	560	GONC	%3	❷
#67Fh	6FC6	GOTO	$GAGNE	
%3				❸
#683h	303	C=(1)	#3	
	901	?B=C	P	❹
	C2	GOYES	%7	
	D2	C=0	A	❺
	31F7	C=(2)	#7F	
	D5	B=C	A	
	D7	D=C	A	
	28	P=	8	
%4				❻
#697h	80F2	CPex	2	
	862	?ST=0	2	
	50	GOYES	%5	
	801	OUT=C		❼
%5				❽
#6A3h	D4	A=B	A	
%6				❾
#6A5h	CC	A=A-1	A	
	5DF	GONC	%6	
	CF	D=D-1	A	❿
	5AE	GONC	%4	
	20	P=	0	
	6B90	GOTO $6		

❶ Modification du numéro de kilomètre.
❷ Test : fin du jeu au bout de 100 km ?
❸ Point de branchement %3.
❹ Test : a-t-on une ligne de fin de tour ? Oui, alors tour suivant ?
❺ Si non, préparation d'un bip (fréquence dans B, durée dans D).
❻ Point de branchement %4. Test : le son est-il autorisé ?
❼ Bip avec volume normal.
❽ Point de branchement %5.
❾ Point de branchement %6. Temporisation (fréquence).
❿ Boucle durée.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%7				❶
#6B5h	174	D1=D1+	#5	
	14B	A=DAT1	B	
	05	SETDEC		
	E4	A=A+1	A	❷
	04	SETHEX		
	149	DAT1=A	B	
	840	ST=0	0	❸
	177	D1=D1+	#8	
	14B	A=DAT1	B	❹
	E4	A=A+1	A	
	1590	DAT1=A	1	
	304	C=(1)	#4	
	902	?A=C	P	
	A0	GOYES	%8	
	30E	C=(1)	#E	
	906	?A≠C	P	❺
	34	GOYES	%9	❻
%8				❼
#6E3h	170	D1=D1+	#1	
	D0	A=0	A	
	15B0	A=DAT1	1	
	D6	C=A	A	
	E4	A=A+1	A	❽
	1590	DAT1=A	1	
	110	A=R0		
	F2	CSL	A	❾
	F2	CSL	A	
	CA	A=A+C	A	
	F6	CSR	A	
	C6	C=C+C	A	
	CA	A=A+C	A	

❶ Point de branchement %7.
❷ Incrémentation du numéro du tour (en décimal).
❸ Suppression de l'inverseur.
❹ Incrémentation du numéro du tour (en hexadécimal).
❺ Réduction du champ de vision...
❻ ...à partir des tours #4h et #Dh.
❼ Point de branchement %8.
❽ Modification du champ de vision.
❾ Champ de vision × #120h.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	130	D0=A		❶
	72A1	GOSUB	$LIGNE	❷
#70Ah	110	A=R0		
	C4	A=A+A	A	
	136	CD0ex		
	EA	A=A-C	A	
	344E600	C=(5)	#006E4	
	CA	A=A+C	A	
	130	D0=A		❸
	7881	GOSUB	$LIGNE	
%9				❹
#724h	862	?ST=0	2	
	62	GOYES	$6	❺
	D2	C=0	A	
	30E	C=(1)	#E	❻
	D5	B=C	A	
	311F	C=(2)	#F1	❼
	D7	D=C	A	
	28	P=	8	
%10				❽
#738h	80F2	CPex	2	
	801	OUT=C		
	D4	A=B	A	
%11				❾
#741h	CC	A=A-1	A	
	5DF	GONC	%11	
	CF	D=D-1	A	
	5FE	GONC	%10	❿
	20	P=	0	

❶ D0 pointe sur la ligne à effacer.
❷ On efface la ligne du haut.
❸ On efface la ligne symétrique par rapport à la ligne centrale.
❹ Point de branchement %9.
❺ Test : le son est-il autorisé ?
❻ Fréquence du bip dans B.
❼ Durée du bip dans D.
❽ Point de branchement %10. Bip avec volume normal.
❾ Point de branchement %11. Temporisation fréquence.
❿ Boucle durée.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$6				❶
#74Dh	1D70	D1=(2)	#07	
	15B5	A=DAT1	6	❷
	AF2	C=0	W	
	307	C=(1)	#7	
	05	SETDEC		
	A7A	A=A+C	W	❸
	04	SETHEX		
	1595	DAT1=A	6	❹
	3451100	C=(5)	#00115	
	110	A=R0		❺
	EA	A=A-C	A	
	130	D0=A		
	7430	GOSUB	$AFF	❻
#779h	16C	D0=D0+	#D	
	1E0040	D1=(4)	#0400	
	301	C=(1)	#1	
	7420	GOSUB	$AFF	❼
#789h	164	D0=D0+	#5	
	1E5040	D1=(4)	#0405	
	301	C=(1)	#1	
	7410	GOSUB	$AFF	❽
#799h	16B	D0=D0+	#C	
	1E2040	D1=(4)	#0402	
	302	C=(1)	#2	
	7400	GOSUB	$AFF	❾
#7A9h	6D3A	GOTO	$1	
#7ADh				❿

❶ Procédure $6 (affichage des données).
❷ Lecture du score.
❸ On ajoute 7 au score.
❹ Mémorisation du nouveau score.
❺ On calcule les adresses des unités sur l'écran.
❻ Affichage du score sur 6 quartets.
❼ Affichage du nombre de kilomètres sur 2 quartets.
❽ Affichage du nombre de tours sur 2 quartets.
❾ Affichage du nombre de vies sur 3 quartets.
❿ Fin du programme principal. Remise à #000h du compteur d'adresses.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
$AFF				❶
#000h	**D7**	**D=C**	**A**	❷
	15F5	**C=DAT1**	**6**	
	AF5	**B=C**	**W**	
%1				❸
#009h	**111**	**A=R1**		
	D2	**C=0**	**A**	
	A89	**C=B**	**P**	
	EA	**A=A-C**	**A**	
	F2	**CSL**	**A**	
	81E	**CSRB**		
	CA	**A=A+C**	**A**	
	139	**D1=AS**		❹
	306	**C=(1)**	**#6**	
%2				❺
#020h	**14B**	**A=DAT1**	**B**	
	1580	**DAT0=A**	**1**	
	16F	**D0=D0+**	**#10**	❻
	16F	**D0=D0+**	**#10**	
	163	**D0=D0+**	**#4**	
	170	**D1=D1+**	**#1**	❼
	A0E	**C=C-1**	**P**	❽
	59E	**GONC**	**%2**	
	132	**AD0ex**		❾
	D2	**C=0**	**A**	
	31DF	**C=(2)**	**#FD**	
	EA	**A=A-C**	**A**	
	130	**D0=A**		
	BF5	**BSR**	**W**	❿
	A0F	**D=D-1**	**P**	

❶ Procédure $AFF.
❷ Nombre de quartets dans D. Quartets eux-mêmes dans B.
❸ Point de branchement %1. A = R1+B×#7h (premier sprite chiffre).
❹ AS à cause de CSRB. Puis on saute 7 lignes.
❺ Point de branchement %2. Transfert de données vers l'écran.
❻ Passage à la ligne du dessous.
❼ Donnée suivante.
❽ Boucle sur 7 lignes.
❾ On pointe sur le chiffre précédent sur l'écran.
❿ Quartet précédent.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	5BB	GONC	%1	❶
	01	RTN		
$ON				❷
#052h	110	A=R0		❸
	345E400	C=(5)	#004E5	
	C2	C=C+A	A	
	06	RSTK=C		❹
	134	D0=C		
	113	A=R3		
	131	D1=A		
	30C	C=(1)	#C	
%1				❺
#06Ch	143	A=DAT1	A	
	140	DAT0=A	A	❻
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	
	163	D0=D0+	#4	
	173	D1=D1+	#4	❼
	A0E	C=C-1	P	
	5AE	GONC	%1	
%2				❽
#084h	32004	C=(3)	#400	
	8FCEE10	GOSBVL	#01EEC	
	808AF	?CBIT=0	15	❾
	FE	GOYES	%2	
	07	C=RSTK		
	134	D0=C		❿
	D0	A=0	A	
	30C	C=(1)	#C	

❶ Boucle sur les quartets.
❷ procédure $ON.
❸ On pointe en bas à droite de l'écran.
❹ Copie de D0 pour effacement du sprite après pression sur [ON].
❺ Point de branchement %1.
❻ transfert des données vers l'écran du sprite « ON ».
❼ Donnée suivante puis boucle sur 13 lignes.
❽ Point de branchement %2.
❾ Attente d'une pression sur [ON].
❿ Initialisation de D0.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	%3			❶
#0A1h	140	DAT0=A	A	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❷
	163	D0=D0+	#4	
	A0E	C=C-1	P	
	50F	GONC	%3	
	01	RTN		
	$VOITURE			❸
#0B5h	3406300	C=(5)	#00360	
	110	A=R0		
	CA	A=A+C	A	❹
	130	D0=A		
	304	C=(1)	#4	❺
	$SPR			❻
#0C7h	DA	A=C	A	
	137	CD1ex		❼
	06	RSTK=C		
	D2	C=0	A	
	A86	C=A	P	
	F2	CSL	A	❽
	112	A=R2		
	CA	A=A+C	A	❾
	131	D1=A		
	3170	C=(2)	#07	
	%1			❿
#0E1h	14B	A=DAT1	B	
	148	DAT0=A	B	

❶ Point de branchement %3. On efface le sprite « ON ».
❷ Passage à la ligne du dessous. Boucle sur 13 lignes.
❸ Procédure $VOITURE.
❹ On pointe sur l'écran à l'emplacement de la voiture.
❺ Préparation du sprite voiture.
❻ Procédure $SPR.
❼ D1 ne sera pas altéré par cette routine.
❽ D1 = R2+#10h×Numéro
❾ R2 pointe sur le sprite objets.
❿ Point de branchement %1. Transfert des données vers l'écran.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	16F	D0=D0+	#10	❶
	16F	D0=D0+	#10	
	163	D0=D0+	#4	
	171	D1=D1+	#2	❷
	CE	C=C-1	A	
	5BE	GONC	%1	
	07	C=RSTK		❸
	135	D1=C		
	01	RTN		
$LIGNE				❹
#0FFh	D2	C=0	A	
	3111	C=(2)	#11	
	D7	D=C	A	
	32E11	C=(3)	#11E	❺
	D5	B=C	A	
%1				❻
#10Eh	D2	C=0	A	
	73BF	GOSUB	$SPR	❼
#114h	136	CD0ex		
	E9	C=C-B	A	❽
	134	D0=C		
	CF	D=D-1	A	
	5FE	GONC	%1	
	01	RTN		
$APPUI				❾
#123h	3456200	C=(5)	#00265	❿
	110	A=R0		
	863	?ST=0	3	

❶ Passage à la ligne du dessous.
❷ Donnée suivante puis boucle sur 8 lignes.
❸ Restauration de D1.
❹ Procédure $LIGNE.
❺ Décélage sur la case suivante sur la même ligne de l'écran.
❻ Point de branchement %1.
❼ Affichage d'un blanc à l'écran.
❽ Case suivante à l'écran. Boucle sur 18 colonnes.
❾ Procédure $APPUI.
❿ On pointe sur la case se trouvant au-dessus de la voiture...

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	70	GOYES	%1	
	325A4	C=(3)	#4A5	❶
%1				❷
#137h	CA	A=A+C	A	
	130	D0=A		❸
	14A	A=DAT0	B	
	77A2	GOSUB	$SCRASH	
#143h	32063	C=(3)	#360	
	110	A=R0		❹
	CA	A=A+C	A	
	130	D0=A		
	D2	C=0	A	
	717F	GOSUB	$SPR	
$APPUI2				❺
#156h	1D31	D1=(2)	#13	❻
	3132	C=(2)	#23	
%2				❼
#15Eh	14B	A=DAT1	B	
	E4	A=A+1	A	
	863	?ST=0	3	
	60	GOYES	%3	
	CC	A=A-1	A	
	CC	A=A-1	A	
%3				❽
#16Ch	149	DAT1=A	B	
	171	D1=D1+	#2	
	808B0	?CBIT=1	0	❾
	50	GOYES	%4	
	177	D1=D1+	#8	❿

❶ ...sauf en cas de descente, dans ce cas on lit la case du dessous.
❷ Point de branchement %1.
❸ Lecture du sprite se trouvant sur la prochaine position de la voiture.
❹ Si on est toujours vivant (!), effacement de la voiture.
❺ procédure $APPUI2.
❻ On pointe sur le début du tableau.
❼ Point de branchement %2. Lecture de la position du bord de la route.
❽ Point de branchement %3. Sauvegarde de la nouvelle position.
❾ On teste la parité du compteur.
❿ Réajustement de pointeur sur la colonne suivante.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
%4				❶
#17Ch	A6E	C=C-1	B	
	5ED	GONC	%2	
#182h	1DE0	D1=(2)	#0E	❷
	110	A=R0		
	3402100	C=(5)	#00120	❸
	D5	B=C	A	
	873	?ST=1	3	
	B0	GOYES	%5	❹
	320D7	C=(3)	#7D0	
	CA	A=A+C	A	
	F9	B=-B	A	
%5				❺
#1A0h	14F	C=DAT1	B	
	06	RSTK=C		
	DD	BCex		
%6				❻
#1A7h	CA	A=A+C	A	
	A0D	B=B-1	P	
	5AF	GONC	%6	
	131	D1=A		❼
	EA	A=A-C	A	
	130	D0=A		❽
	07	C=RSTK		
	D7	D=C	A	
	31F8	C=(2)	#8F	
	D5	B=C	A	❾
	3142	C=(2)	#24	
%7				❿
#1C5h	E1	B=B-C	A	

❶ Point de branchement %4. 18 colonnes à modifier.
❷ D1 pointe sur le champ de vision.
❸ D0 pointe sur le début de l'écran. B est le décalage entre 2 lignes.
❹ En cas de montée, le scrolling commence en fin d'écran (B inversé).
❺ Point de branchement %5. Lecture du champ de vision.
❻ Point de branchement %6. Modification du début du scrolling.
❼ D1 = D0 + ou - #120h
❽ D0 = R0 (+#7D0h ?) + ou - #120h × champ de vision.
❾ Compteur des lignes écran.
❿ Point de branchement %7. Compteur = #6Bh - #24h × champ de vision.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	A0F	D=D-1	P	
	5AF	GONC	%7	
%8				❶
#1CDh	1537	A=DAT1	W	
	1507	DAT0=A	W	
	873	?ST=1	3	❷
	A1	GOYES	%9	
	18F	D0=D0+	#10	
	1CF	D1=D1+	#10	❸
	A6D	B=B-1	B	
	59E	GONC	%8	
	17F	D1=D1+	#10	
	137	CD1ex		❹
	134	D0=C		
	5E0	GONC	%10	
%9				❺
#1F2h	16F	D0=D0-	#10	
	17F	D1=D1-	#10	
	A6D	B=B-1	B	
	51D	GONC	%8	
%10				❻
#1FEh	D2	C=0	A	
	1FE0408	D1=(5)	#8040E	
	15F0	C=DAT1	1	❼
	D7	D=C	A	❽
	176	D1=D1+	#7	
	3111	C=(2)	#11	❾
%11				❿
#214h	06	RSTK=C		

❶ Point de branchement %8. Transfert inter écran.
❷ Test : descente en cours ?
❸ Quartets suivants en cas de descente.
❹ D0 pointe sur les quartets à afficher.
❺ Point de branchement %9. Quartets suivants en cas de montée.
❻ Point de branchement %10.
❼ D contient le champ de vision.
❽ D1 pointe sur la position de la dernière case de la 1ère colonne.
❾ C contient le compteur des 18 colonnes de l'écran de jeu.
❿ Point de branchement %11. Sauvegarde du compteur de colonnes.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	DB	C=D	A	❶
	863	?ST=0	3	
	70	GOYES	%12	
	306	C=(1)	#6	
	EB	C=C-D	A	
%12				❷
#222h	14B	A=DAT1	B	
	1C1	D1=D1-	#2	
	B6A	A=A-C	B	❸
	A64	A=A+A	B	
	482	GOC	%14	
	14B	A=DAT1	B	❹
	B62	C=C-A	B	
	808B7	?CBIT=1	7	
	B1	GOYES	%14	
%13				❺
#23Eh	133	AD1ex		
	D8	B=A	A	❻
	A6A	A=A+C	B	
	131	D1=A		
	173	D1=D1+	#4	
	14F	C=DAT1	B	❼
	D4	A=B	A	
	131	D1=A		
	550	GONC	%15	❽
%14				❾
#257h	301	C=(1)	#1	
%15				❿
#25Ah	796E	GOSUB	$SPR	
#25Eh	17B	D1=D1+	#C	

❶ C = champ de vision (si on descend, C = 6 - champ de vision).
❷ Point de branchement %12. Lecture de la position de la 1ère case.
❸ Test : position de la dernière case inférieure à C ?
❹ Test : position de la première case supérieure à C ? Sprite bord.
❺ Point de branchement %13.
❻ En cas de test négatif, D1 est sauvé dans B.
❼ Lecture du sprite à afficher. restauration de D1.
❽ Si on est hors du contenu, affichage d'un sprite bord.
❾ Point de branchement %14.
❿ Point de branchement %15. Affichage du sprite. Colonne suivante.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	132	ADOex		
	34E1100	C=(5)	#0011E	
	EA	A=A-C	A	❶
	130	D0=A		
	07	C=RSTK		
	CE	C=C-1	A	❷
	5F9	GONC	%11	
				❸
#277h	3458300	C=(5)	#00385	
	110	A=R0		❹
	CA	A=A+C	A	
	130	D0=A		
	14A	A=DAT0	B	
$COLLISION				❺
#289h	302	C=(1)	#2	❻
	906	?A≠C	P	
	93	GOYES	%6	
	870	?ST=1	0	❼
	80	GOYES	%1	
	850	ST=1	0	
	550	GONC	%2	
%1				❽
#29Ch	840	ST=0	0	
%2				❾
#29Fh	862	?ST=0	2	
	32	GOYES	%5	❿
	D2	C=0	A	
	31F8	C=(2)	#8F	

❶ On pointe sur la case suivante de la même ligne sur l'écran de jeu.
❷ Boucle sur 18 colonnes.
❸ Début des tests de collision.
❹ Lecture du quartet situé à la place de la voiture.
❺ Procédure $COLLISION.
❻ Test : inverseur ?
❼ Si oui, on inverse le flag ST0.
❽ Point de branchement %1.
❾ Point de branchement %2. Test : son autorisé ?
❿ Sinon, on affiche la voiture.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D5	B=C	A	❶
	D2	C=0	A	
	28	P=	8	
%3				❷
#2B0h	80F2	CPex	2	
	801	OUT=C		
	D4	A=B	A	
%4				❸
#2B9h	CC	A=A-1	A	
	5DF	GONC	%4	
	CD	B=B-1	A	❹
	5FE	GONC	%3	
	20	P=	0	
%5				❺
#2C5h	416	GOC	%14	
%6				❻
#2C8h	304	C=(1)	#4	
	906	?A≠C	P	
	80	GOYES	%7	
	851	ST=1	1	❼
	5BC	GONC	%2	
%7				❽
#2D6h	307	C=(1)	#7	
	906	?A≠C	P	❾
	F4	GOYES	%15	
%8				❿
#2DEh	1D20	D1=(2)	#02	
	147	C=DAT1	A	

❶ Fréquence dans B.
❷ Point de branchement %3. Bip avec volume normal.
❸ Point de branchement %4. Temporisation fréquence.
❹ Boucle durée (de plus en plus aigu et rapide).
❺ Point de branchement %5.
❻ Point de branchement %6. Test : accélérateur ?
❼ Activation du flag accélérateur. Bip modulé.
❽ Point de branchement %7.
❾ Test : vie ?
❿ Point de branchement %8.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	05	SETDEC		❶
	E6	C=C+1	A	
	04	SETHEX		
	1553	DAT1=C	X	
	D2	C=0	A	
	308	C=(1)	#8	
%9				❷
#2F4h	06	RSTK=C		
	30F	C=(1)	#F	
	D5	B=C	A	❸
	D7	D=C	A	
	28	P=	8	
%10				❹
#2FFh	80F2	CPex	2	
	862	?ST=0	2	
	50	GOYES	%11	
	801	OUT=C		❺
%11				❻
#30Bh	D4	A=B	A	
%12				❼
#30Dh	CC	A=A-1	A	
	5DF	GONC	%12	
	CF	D=D-1	A	❽
	5AE	GONC	%10	
	80F2	CPex	2	
%13				❾
#31Bh	CE	C=C-1	A	
	5DF	GONC	%13	
	07	C=RSTK		
	CE	C=C-1	A	❿

❶ Si le test est positif, on ajoute une vie. Neuf « tics ».
❷ Point de branchement %9.
❸ On place la fréquence dans B et la durée dans D.
❹ Point de branchement %10. Test : le son est-il autorisé ?
❺ Bip volume normal.
❻ Point de branchement %11.
❼ Point de branchement %12. Temporisation.
❽ Boucle durée. P=0, C=#800h.
❾ Point de branchement %13. Temporisation entre deux tics.
❿ Tic suivant.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	5FC	GONC	%9	
%14				❶
#327h	6D8D	GOTO	$VOITURE	
%15				❷
#32Bh	30C	C=(1)	#C	
	906	?A≠C	P	❸
	D5	GOYES	%18	
	1D70	D1=(2)	#07	
	15B5	A=DAT1	6	❹
	AF2	C=0	W	
	3104	C=(2)	#40	
	05	SETDEC		
	A7A	A=A+C	W	
	04	SETHEX		
	1595	DAT1=A	6	
	177	D1=D1+	#8	
	14B	A=DAT1	B	❺
	B04	A=A+1	P	
	1590	DAT1=A	1	❻
	438	GOC	%8	
	862	?ST=0	2	❼
	7C	GOYES	%14	
	31FF	C=(2)	#FF	
	D5	B=C	A	❽
	3153	C=(2)	#35	
	D7	D=C	A	❾
%16				❿
#36Eh	32008	C=(3)	#800	
	801	OUT=C		

❶ Point de branchement %14. Affichage de la voiture.
❷ Point de branchement %15.
❸ Test : essence ?
❹ Si le test est positif, on modifie le score (140).
❺ Incrémentation du compteur essence.
❻ Ajout d'une vie si 16 pompes à essence ramassées sans accident.
❼ Test : son autorisé ?
❽ Fréquence dans B.
❾ Durée dans D.
❿ Point de branchement %16. Bip ! (volume moyen).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	6300	NOP3		
	D2	C=0	A	
	801	OUT=C		
	D4	A=B	A	
%17				❶
#381h	CC	A=A-1	A	
	5DF	GONC	%17	
	CF	D=D-1	A	
	55E	GONC	%16	❷
	4B9	GOC	%14	
%18				❸
#38Eh	30F	C=(1)	#F	
	906	?A≠C	P	
	39	GOYES	%14	❹
	798D	GOSUB	$APPUI	
#39Ah	758D	GOSUB	$APPUI	
#39Eh	862	?ST=0	2	
	00	RTNYES		❺
	D1	B=0	A	
	D2	C=0	A	
	B25	B=B+1	XS	
	30F	C=(1)	#F	
	D7	D=C	A	❻
	28	P=	8	
%19				❼
#3B1h	80F2	CPex	2	
	801	OUT=C		
	D4	A=B	A	
%20				❽
#3BAh	CC	A=A-1	A	

❶ Point de branchement %17. Temporisation fréquence.
❷ Boucle durée.
❸ Point de branchement %18. S'il ne s'agit pas d'une flaque d'huile, on n'effectue pas le traitement réservé aux flaques d'huile.
❹ Si la flaque d'huile est prise de face, on effectue un déplacement double aléatoire, si elle est prise par le côté, on prolonge le déplacement.
❺ Test : son autorisé ? Fréquence dans B.
❻ 16 fois la boucle.
❼ Point de branchement %19. Bip avec volume normal.
❽ Point de branchement %20. Temporisation pour la fréquence.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	5DF	GONC	%20	
	81861F	B=B+#10	B	
	5BE	GONC	%19	❶
	CD	B=B-1	A	
%21				❷
#3CAh	80F2	CPex	2	
	801	OUT=C		❸
	D4	A=B	A	
%22				❹
#3D3h	CC	A=A-1	A	
	5DF	GONC	%22	
	81869F	B=B-#10	B	❺
	5BE	GONC	%21	
	CF	D=D-1	A	
	5DC	GONC	%19	
	20	P=	0	
	01	RTN		
$SCRASH				❻
#3EAh	80870	?ABIT=1	0	❼
	00	RTNYES		
	80861	?ABIT=0	1	
	00	RTNYES		
	302	C=(1)	#2	
	902	?A=C	P	❽
	00	RTNYES		
	110	A=R0		
	3406300	C=(5)	#00360	
	CA	A=A+C	A	
	130	D0=A		❾

❶ Boucle durée (plus grave). On réajuste pour l'effet sonore.
❷ Point de branchement %21.
❸ Bip !
❹ Point de branchement %22. Temporisation fréquence.
❺ Boucle durée (plus aiguë). La boucle est répétée 16 fois.
❻ Procédure $SCRASH.
❼ On sélectionne les sprites qui provoquent un accident, et ce, en fonction de l'identification envoyée dans le registre A.
❽ Seuls, #6h, #Ah et #Eh sont acceptés.
❾ Affichage d'une voiture détruite à l'emplacement de la voiture.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	306	C=(1)	#6	
	71BC	GOSUB	$SPR	
#416h	31D7	C=(2)	#7D	❶
	D5	B=C	A	
	110	A=R0		
	130	D0=A		
%1				❷
#422h	1527	A=DAT0	W	
	BFC	A=-A-1	W	❸
	1507	DAT0=A	W	
	16F	D0=D0+	#10	❹
	310A	C=(2)	#A0	
%2				❺
	A6E	C=C-1	B	
	5CF	GONC	%2	
	A6D	B=B-1	B	
	54E	GONC	%1	❻
	D0	A=0	A	
	D1	B=0	A	❼
	D2	C=0	A	
	D3	D=0	A	
	CC	A=A-1	A	
	28	P=	8	
%3				❽
#44Ch	B65	B=B+1	B	
	451	GOC	%4	
	B67	D=D+1	B	❾
	56F	GONC	%5	
	D7	D=C	A	❿
	D6	C=A	A	

❶ ...sur tout l'écran !
❷ Point de branchement %1.
❸ Inversion vidéo.
❹ 16 quartets à la fois.
❺ Point de branchement %2.
❻ Quartets suivants, durée.
❼ Fréquence 1 puis fréquence 2.
❽ Point de branchement %3. Temporisation fréquence 1.
❾ Temporisation fréquence 2.
❿ On simule **D=A** **A** (instruction inexistante) avec **D7**, **D6** et **DF**.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	DF	CDex	A	
	C7	D=D+D	A	❶
	C7	D=D+D	A	
	5A0	GONC	%6	
%4				❷
#465h	D8	B=A	A	
	A6C	A=A-1	B	
	421	GOC	%7	
%5				❸
#46Dh	80F2	CPex	2	
	862	?ST=0	2	
	50	GOYES	%6	
	801	OUT=C		❹
%6				❺
#479h	62DF	GOTO	%3	
%7				❻
#47Dh	D5	B=C	A	
	D2	C=0	A	
	801	OUT=C		
%8				❼
#484h	CD	B=B-1	A	
	5DF	GONC	%8	
	31F1	C=(2)	#1F	
	D5	B=C	A	
%9				❽
#48Fh	31CF	C=(2)	#FC	
	110	A=R0		
	130	D0=A		❾
%10				❿
#499h	15A7	A=DAT0	8	

❶ Calcul de D = Ax#4h.
❷ Point de branchement %4. B = A. Boucle durée.
❸ Point de branchement %5.
❹ Bip commun aux deux fréquences (volume normal).
❺ Point de branchement %6.
❻ Point de branchement %7.
❼ Point de branchement %8. 32 répétitions (8 quartets de 4 bits).
❽ Point de branchement %9. Tout l'écran.
❾ On pointe au début de l'écran.
❿ Point de branchement %10.

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	A74	A=A+A	W	❶
	1587	DAT0=A	8	
	167	D0=D0+	#8	
	A6E	C=C-1	B	❷
	5EE	GONC	%10	
%11				❸
#4ADh	A6E	C=C-1	B	
	5CF	GONC	%11	
	A6D	B=B-1	B	❹
	58D	GONC	%8	
	1E2040	D1=(4)	#0402	
	143	A=DAT1	A	
	05	SETDEC		
	A3C	A=A-1	X	❺
	04	SETHEX		
	1513	DAT1=A	X	
	4D1	GOC	%12	❻
	110	A=R0		
	3410100	C=(5)	#00101	
	EA	A=A-C	A	
	130	D0=A		❼
	E6	C=C+1	A	
	7B1B	GOSUB	$AFF	❽
#4E5h	8C584F	GOLONG	$0	
%12				❾
#4EBh	34B4000	C=(5)	#0004B	
$SOUND				❿
#4F2h	862	?ST=0	2	
	74	GOYES	%5	

❶ On escamote 1 bit.
❷ Quartets suivants.
❸ Point de branchement %11. Temporisation.
❹ On efface l'écran...
❺ On supprime une vie.
❻ Test : plus de vie ? Alors, le joueur a perdu !
❼ On affiche le nombre de vies.
❽ Nouveau départ s'il reste des vies.
❾ Point de branchement %12. Décalage PC-données (son).
❿ Procédure $SOUND. Test : bip autorisé ?

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	81B4	A=PC		
#4FBh	CA	A=A+C	A	❶
	131	D1=A		
%1				❷
#500h	32005	C=(3)	#500	
%2				❸
#505h	A3E	C=C-1	X	
	5CF	GONC	%2	
	143	A=DAT1	A	❹
	908	?A=0	P	
	B2	GOYES	%5	❺
	172	D1=D1+	#3	
	14F	C=DAT1	B	❻
	171	D1=D1+	#2	
	D7	D=C	A	
	D2	C=0	A	
	28	P=	8	
%3				❼
#522h	80F2	CPex	2	
	801	OUT=C		
	D8	B=A	A	
%4				❽
#52Bh	A3D	B=B-1	X	
	5CF	GONC	%4	
	A6F	D=D-1	B	❾
	5DE	GONC	%3	
	20	P=	0	
	46C	GOC	%1	
%5				❿
#53Ch	721B	GOSUB	$ON	

❶ On pointe sur les données relatives au son.
❷ Point de branchement %1.
❸ Point de branchement %2. Temporisation entre deux sons.
❹ Lecture de la fréquence.
❺ Test : fin ?
❻ Lecture de la durée.
❼ Point de branchement %3. Bip avec un volume normal.
❽ Point de branchement %4. Temporisation fréquence.
❾ Boucle durée.
❿ Point de branchement %5. Attente d'une pression sur [ON].

Adresses	Codes hexadécimaux	Mnémoniques	Renvois
#540h	D2	C=0 A	
	81B3	PC=C	❶
$DONNEES			❷
#546h	1010F		❸
	1010A	*Codage du son*	
	10105	*avec trois quartets*	
	1010F	*pour la fréquence*	
	7D00A	*et deux quartets*	
	5E005	*pour la durée.*	
	5E00A		
	10105		
	1010A		
	E0105		
	101FF		
	0		
#57Eh	7D006		❹
	7D003	*Idem.*	
	7D003		
	F1102		
	7D003		
	DA003		
	7D009		
	0		
$GAGNE			❺
#5A2h	110	A=R0	❻
	130	D0=A	
	114	A=R4	❼
	131	D1=A	
	3183	C=(2) #38	❽

❶ Reset système.
❷ Procédure $DONNEES.
❸ Données utilisées quand le joueur perd.
❹ Données utilisées quand le joueur gagne.
❺ Procédure $GAGNE.
❻ On pointe sur l'écran.
❼ On pointe sur la coupe.
❽ 39 lignes (presque tout l'écran...).

Adresses	Codes hexadécimaux	Mnémoniques		Renvois
	D5	B=C	A	
%1				❶
#5B4h	3200C	C=(3)	#C00	
%2				❷
#5B9h	A3E	C=C-1	X	
	5CF	GONC	%2	
	15BB	A=DAT1	12	❸
	158B	DAT0=A	12	
	16F	D0=D0+	#10	
	16F	D0=D0+	#10	❹
	163	D0=D0+	#4	
	17B	D1=D1+	#C	
	A6D	B=B-1	B	
	5DD	GONC	%1	❺
#5D9h	3438000	C=(5)	#00083	
#5E0h	611F	GOTO	$SOUND	

Adresses	Codes hexadécimaux	Renvois
$SPRITES		❻
#5E4h	77737 07777 77005 50730 73005 27707 37115 51255 55550 05725 20520 05211 25251 15512 55555 50057 25205 20052 11252 57153 70555 55500 35052 75200 52370 52541 55125 55555 00552 52052 00521 42525 41551 25555 55005 52520 52002 21425 25777 57077 77770 05507 70770 02277 0777	❼
#6C4h	75555 57322 22277 44711 77446 44755 57444 71174 47711 75577 44222 27557 55775 57447	❽

❶ Point de branchement %1. Une petite temporisation (c'est plus joli !).
❷ Point de branchement %2.
❸ Transferts de données vers l'écran.
❹ Passage à la ligne du dessous et donnée suivante.
❺ Boucle sur 57 lignes puis décalage PC-Données.
❻ Procédure $SPRITES.
❼ Sprite score (équivalent de R0 pour la version S/SX).
❽ Sprite chiffres (équivalent de R1 pour la version S/SX).

Adresses	Codes hexadécimaux	Renvois
#70Ah	00000 00000 00000 0FFBA 5DBA5 DBA5D FFF00 0F000 F000F 000F0 F0F0F 0F0F0 F0F0F 0FE64 BFBF6 4FEF0 707E2 4FFFF 247E7 05076 21EDB 76236 60E7F E9BFF F93FF DE7DB E7BDF FE742 E7998 D9FF7 E3C7C FEE6C 64FC9 9BA9A B490F 0C324 DBDBD BDB24 C3814 26699 99664 281	❶
#7DAh	FFF71 00493 15543 55435 54555 45554 95549 55415 93151 004FF F7	❷
#80Eh	FFFFF FFFFF F7100 00000 00041 0008F F0000 41000 70070 00410 0E000 83004 10810 000C0 04106 00000 03041 01000 00040 41800 00000 80414 00000 00014 14000 00000 14120 00000 00241 20000 00002 41100 00000 04411 00000 00044 11000 00000 44110 00000 00441 30000 00006 41300 00000 06417 00000 00074 16000 F7000 341A0 0C081 08241 21020 02042 41460 10040 31414 81100 4C014 180E2 00A38 04101 07007 04041 0108F F0040 41020 00000 20410 40000 00104 10810 000C0 04100 20000 20041 00C00 08100 41000 70070 00410 00CFF 10004 10004 00100 04100 08080 00041 00080 80000 41000 80800 00410 00014 00004 10000 14000 04100 081C0 00041 000E1 C3000 41000 F1C70 00410 00F1C 70004 1008F 3EF00 04100 8FFFF 00041 008FF FF000 41000 00000 00499 E210D 5D144 9A261 05550 4499E E1055 D1449 A2A14 55500 499E2 1CDDD 14410 00000 00004 FFFFF FFFFF F7	❸ ❹

❶ Codage du sprite score (équivalent de R2 sur HP 48 S/SX).
❷ Codage du sprite [ON] (équivalent de R3 sur HP 48 S/SX).
❸ Codage du sprite coupe de fin (équivalent de R4 sur HP 48 S/SX).
❹ Fin du code.

3

Exécuter le programme

Pour exécuter le programme, vous devez réunir dans le répertoire DSETH (spécialement créé), les objets suivants :

- 'RPL' (ou 'LANC'), le programme lanceur en RPL,
- 'CODE', le code exécutable.

Les objets de développement tels que la chaîne de codes hexadécimaux ('SETH48G') et le programme ASS peuvent rester dans le répertoire DSETH mais ils ne seront pas utilisés pendant l'exécution du programme. Pour gagner de la mémoire, vous pouvez donc supprimer 'SETH48G' et 'ASS'.

Pour jouer, il suffit d'évaluer le programme 'RPL' qui est le lanceur écrit en langage RPL. Le message « SETH HP-48 G » étant affiché, appuyez ensuite sur [ENTER] pour lancer le jeu. Vous pouvez aussi appuyer simultanément sur [α]-[↱]-[F] pour reprendre une partie en cours (ne pas utiliser cette possibilité si vous venez d'effectuer une transmission par infrarouge). Une pression sur [ON] fera ensuite démarrer votre voiture...

4

Comment jouer ?

Vous disposez de 10 voitures pour effectuer 15 tours de 6 km, un final de 10 km vous permettra ensuite de prouver que vous êtes vraiment le meilleur ! Bien entendu, le jeu se charge de faire accélérer votre voiture au fil des tours... Le seul moyen de ralentir reste l'accident (toujours brutal) ou la victoire finale !

Pour jouer, utilisez les touches suivantes :

- [α] pour monter (vers la gauche par rapport au déplacement),
- [↰] pour descendre (vers la gauche par rapport au déplacement),

- [8] pour quitter le jeu,
- [÷] pour interrompre temporairement une partie,
- [ON] pour reprendre la partie interrompue avec [÷],
- [←] et [→] servent à activer ou à désactiver le son.

Attention ! Le seul moyen de quitter le jeu consiste à appuyer sur la touche [8].

En début de partie, le son, ou l'absence de son, est fonction du flag beep RPL. Cependant, les touches [←] et [→] permettent d'activer ou de désactiver le son en cours de partie.

Quelques astuces :

- anticipez vos déplacements,
- passez sur 16 pompes à essence et gagner ainsi une vie (en fait, une voiture !) supplémentaire,
- utilisez bien les flaques d'huiles en les prenant par le côté,
- maintenez enfoncée la touche [α] ou la touche [↰] pour vous déplacer, cette technique est généralement plus efficace que des pressions successives sur les touches [α] et [↰].

5
Retrouver les fichiers sur la disquette

Les chemins d'accès aux différents fichiers relatifs aux variables associées à ce jeu sont :

- 'SETH48G', DUNOD_1\G_GX\OB_P_OBG\SETH_G\SETH48G,
- 'CODE', DUNOD_1\G_GX\OB_P_OBG\SETH_G\CODE,
- 'RPL', DUNOD_1\G_GX\OB_P_OBG\SETH_G\RPL,
- 'ASS', DUNOD_1\G_GX\ASS.

36

Structure des programmes

Ce chapitre regroupe des informations relatives à la structure des listings. Il vous permettra de localiser facilement un traitement ou une procédure au sein d'un programme. L'étude de la structure d'un programme est indispensable à la compréhension dudit programme. Les adresses sont des adresses relatives. En général, l'adresse #000h correspond au premier quartet de la longueur totale hors prologue qui suit le prologue CCD20 au début du code hexadécimal.

1

Les listings pour HP-48 S et SX

1.1

ALPHA pour HP-48 S et SX

1	#000h	Initialisation
2	#093h	Affichage du zoom
2.1	#093h	L'écran est effacé
2.2	#0ADh	Lecture de la case
2.3	#0EDh	Traitement des figures autres que les triangles
2.4	#122h	Traitement des triangles

3	#16Dh	Inversion des vidéo (blanc au noir)
3.1	#1B5h	Boucle
4	#201h	Triangles restants
5	#21Ah	Attente du relâchement des touches
6	#232h	Test des touches
6.1	#232h	Touches générales
6.2	#2ACh	Touches de direction
7	#310h	Gestion du coup
7.1	#310h	Initialisation des pointeurs
7.2	#35Ah	Réaction entre deux pièces
7.3	#37Ah	Assemblage de deux pièces
7.4	#436h	Inversion de deux pièces (swap)
7.5	#470h	Déplacement

Le compteur d'adresses est remis à #000h au début des procédures (qui sont placées à la suite du code du programme principal).

1	#000h	Chronomètre
2	#096h	Affichage (transfert de données vers l'écran)
3	#0F2h	Codage des chiffres
4	#1A6h	Codage des sprites (figures de jeu)
5	#29Eh	Procédure « gagné ! »
6	#294h	Procédure « perdu ! »
7	#2A7h	Procédure utilisée en fin de jeu

1.2
STAR pour HP-48 S et SX

1	#000h	Initialisation
2	#074h	Tableau de jeu n°1
3	#081h	Affichage du tableau de jeu en cours
3.1	#081h	Lecture des pointeurs
3.2	#0CAh	Affichage des bords
3.3	#15Ah	Affichage des pastilles à gober
3.4	#1B2h	Affichage du gobeur et du carré
4	#1CEh	Tests des touches
5	#295h	Tests des collisions
6	#303h	Mode « tricheur »
7	#318h	Déplacement
8	#394h	Affichage des sprites de jeu
9	#3DFh	Codage des bords du tableau puis des sprites

1.3
XALK pour HP-48 S et SX

1	#000h	Initialisation du jeu
2	#07Ah	Passage à la figure suivante
3	#0A2h	Test des touches
4	#199h	Fin du jeu (perdu !)
5	#1D3h	Chute vers la droite
6	#1FEh	Pose de la figure en cours de chute (figure courante)
7	#24Bh	Création du tableau de traitement des cascades
8	#27Bh	Rechrche horizontale
9	#2E0h	Recherche verticale
10	#391h	Cascades
11	#3FEh	Score et vitesse
12	#44Fh	Affichage

Le compteur d'adresses est remis à #000h au début des procédures (qui sont placées à la suite du code du programme principal).

1	#000h	Procédure de calcul
2	#08Fh	Détermination aléatoire de la prochaine figure
3	#0ACh	Gestion du score
4	#0C0h	Affichage
5	#156h	Bip !
6	#183h	Temporisation

1.4
SETH pour HP-48 S et SX

SETH manipule un tableau de jeu comprenant 18 colonnes et 7 lignes. Un sprite occupe une colonne d'une ligne.

1	#000h	Initialisation du jeu
2	#110h	Départ de la course
2.1	#110h	Données nécessaires
2.2	#151h	Ecran
3	#1DEh	Scrolling
3.1	#215h	Scrolling de l'écran (défilement vers la gauche)
3.2	#23Dh	Tests des touches

3.3	#2D8h	Scrolling du tableau
4	#301h	Création de la 18ème colonne de l'écran de jeu
4.1	#365h	Initialisation des données requises
4.2	#375h	Lecture du tableau par le PC
4.3	#3D7h	Voiture détruite
4.4	#3F4h	Gestion des vie(s) du joueur
4.5	#411h	Voiture adverse
4.6	#458h	Autres sprites
4.7	#4CFh	Ligne de fin de kilomètre ou de fin de tour
5	#52Fh	Affichage de la 18ème colonne de l'écran de jeu
6	#5DFh	Test des touches
7	#632h	Passage au kilomètre suivant (ou au tour suivant)
7.1	#632h	Kilomètre suivant
7.2	#6ACh	Tour suivant
8	#744h	Affichage des données

Le compteur d'adresses est remis à #000h au début des procédures (qui sont placées à la suite du code du programme principal).

1	#000h	Affichage
2	#052h	Attente d'une pression sur la touche [ON]
3	#0B5h	Voiture
4	#0C7h	Affichage des sprites
5	#0FFh	Ligne
6	#123h	Test : collision ?
7	#156h	Scrolling tableau
8	#182h	Scrolling écran
9	#1FEh	Initialisation des données requises
10	#214h	Affichage de la ligne concernée
11	#277h	Test : collision ?
12	#289h	Gestion des collisions
12.1	#289h	Inverseur
12.2	#2C8h	Accélérateur
12.3	#2D6h	Vie
12.4	#32Bh	Essence
12.5	#38Eh	Flaque d'huile
13	#3EAh	Accident
14	#4F2h	Gestion du son
15	#546h	Données de fin de partie
15.1	#546h	Données utilisées en cas d'échec (perdu !)
15.2	#57Eh	Données utilisées en cas de victoire (gagné !)
16	#5A2h	Gagné !

Dans le cas de SETH, les procédures représentent une part très importante du code (plus de 40%).

2
Les listings pour HP-48 G et GX

2.1
ALPHA pour HP-48 G et GX

1	#000h	Initialisation du jeu
2	#093h	Afficahge du zoom
2.1	#093h	L'écran est effacé
2.2	#0ADh	Lecture de la case courante
2.3	#0EDh	Gestion des sprites qui ne sont pas des triangles
2.4	#122h	Gestion des sprites triangles
3	#16Dh	Inversion vidéo
4	#1B5h	Boucle
5	#201h	Triangles restants
6	#21Ah	Attente du relâchement des touches
7	#232h	Test des touches
7.1	#232h	Touches générales
7.2	#2ACh	Touches de direction
8	#310h	Gestion du coup
8.1	#310h	Initialisation des pointeurs requis
8.2	#326h	Blanc
8.3	#35Ah	Réaction entre deux figures
8.4	#37Ah	Assemblage de deux figures
8.5	#436h	Inversion de deux figures (swap)
8.6	#470h	Déplacement des figures

Le compteur d'adresses est remis à #000h au début des procédures (qui sont placées à la suite du code du programme principal).

1	#000h	Chronomètre
2	#096h	Affichage (transfert de données vers l'écran)
3	#0F2h	Codage des chiffres
4	#1A6h	Codage des sprites
5	#21Eh	Codage des sprites (blancs et carrés)
6	#29Eh	Procédure « gagné ! »
7	#294h	Procédure « perdu ! »
8	#2A7h	Procédure utilisée en fin de jeu

2.2
STAR pour HP-48 G et GX

1	#000h	Initialisation
2	#074h	Tableau de jeu n°1
3	#081h	Affichage du tableau de jeu en cours
3.1	#081h	Lecture des pointeurs
3.2	#0CAh	Affichage des bords
3.3	#15Ah	Affichage des pastilles à gober
3.4	#1B2h	Affichage du gobeur et du carré
4	#1CEh	Tests des touches
5	#295h	Tests des collisions
6	#303h	Mode « tricheur »
7	#318h	Déplacement
8	#394h	Affichage des sprites de jeu
9	#3DFh	Codage des bords du tableau
10	#3F7h	Codage des sprites

3
Les listings pour HP-48 G

3.1
XALK pour HP-48 G

1	#000h	Initialisation du jeu
2	#09Eh	Passage à la figure suivante
3	#0C6h	Test des touches
4	#1BFh	Fin du jeu (perdu !)
5	#1F9h	Chute vers la droite
6	#224h	Pose de la figure en cours de chute (figure courante)
7	#271h	Création du tableau de traitement des cascades
8	#2A1h	Recherche horizontale
9	#306h	Recherche verticale

10	#3B7h	Cascades
11	#424h	Score et vitesse
12	#475h	Affichage

Le compteur d'adresses est remis à #000h au début des procédures (qui sont placées à la suite du code du programme principal).

1	#000h	Procédure de calcul
2	#083h	Détermination aléatoire de la prochaine figure
3	#0A0h	Gestion du score
4	#0B4h	Affichage
5	#144h	Bip !
6	#171h	Temporisation
7	#187h	Codage (graphisme) des sprites de jeu
8	#227h	Codage (graphisme) des chiffres

3.2
SETH pour HP-48 G

1	#000h	Initialisation du jeu
2	#119h	Départ de la course
2.1	#119h	Initialisation des données requises
2.2	#15Ah	Affichage de la piste
3	#1E7h	Scrolling
3.1	#21Eh	Scrolling de l'écran
3.2	#246h	Test des touches
3.3	#2ADh	Temporisation entre deux décalages
3.4	#2E1h	Scrolling du tableau
4	#30Ah	Création de la 18ème colonne
4.1	#36Eh	Initialisation des données requises
4.2	#37Eh	Lecture du tableau par le registre PC
4.3	#3E0h	Voiture détruite
4.4	#3FDh	Gestion des vie(s)
4.5	#41Ah	Voiture adverse
4.6	#461h	Autres sprites
4.7	#4D4h	Ligne de fin de kilomètre ou de fin de tour
5	#538h	Affichage de la 18ème colonne
6	#5E8h	Test des touches
7	#63Bh	Kilomètre suivant ou tour suivant
7.1	#63Bh	Kilomètre suivant

7.2	#6B5h	Tour suivant
8	#74Dh	Affichage des données

Le compteur d'adresses est remis à #000h au début des procédures (qui sont placées à la suite du code du programme principal).

1	#000h	Affichage
2	#052h	Gestion d'une pression sur la touche [ON]
3	#0B5h	Voiture
4	#0C7h	Transfert d'un sprite vers l'écran
5	#0FFh	Ligne
6	#123h	Appui sur une touche et test d'une collision
7	#156h	Scrolling tableau
8	#182h	Scrolling écran
9	#289h	Collision
9.1	#289h	Inverseur
9.2	#2C8h	Accélérateur
9.3	#2D6h	Gestion des vies
9.4	#32Bh	Essence
9.5	#38Eh	Flaque d'huile
10	#3EAh	Gestion d'un accident
11	#4F2h	Son
12	#546h	Données de fin de partie
12.1	#546h	Données utilisées en cas d'échec
12.2	#57Eh	Données utilisées en cas de victoire
13	#5A2h	Gagné !
14	#5E4h	Sprites
14.1	#5E4h	Sprites pour le score
14.2	#6C4h	Sprites pour les chiffres
14.3	#70Ah	Sprites pour les différents objets
14.4	#7DAh	Sprite pour la touche [ON]
14.5	#80Eh	Codage du graphisme de la coupe de victoire

Bibliographie

Ci-dessous, la liste des livres consultés lors de la rédaction du présent ouvrage.

P. Courbis, S. Lalande. – *Voyage au centre de la HP-48*, Angkor, 1991.
P. Courbis. – *Voyage au centre de la HP-48 G/GX*, Angkor, 1993.
J.-M. Ferrard. – *Les secrets de la HP-48*, D3i, 1993.
M. Cornillault. – *HP-48 : Mathématiques en prépa*, Dunod, 1994.
P. Kezirian. – *Assembleur HP-48*, D3i, 1994.
O. Nachba, C. Lecourt. – *Initiation à l'assembleur*, Angkor, 1994.

Ci-dessous, la liste des bulletins et périodiques consultés lors de la rédaction du présent ouvrage.

Haute Performance, n°2 à 11, édité par le club FFJM,
Poche Computers, n°2 et 3, édité par Microcards,
CHP, bulletin du club hexagone programmation
Texte divers en provenance de BBS ou de 3615 Calculator

Utilisation de la disquette

Quel que soit le protocole de communication utilisé, paramétrez vos logiciels de communication en mode binaire (et non en mode ASCII).

Le contenu de la disquette jointe à ce livre n'appartient pas au domaine public. Son utilisation implique l'achat du livre. Il est donc formellement interdit (loi du 11 mars 1957) de diffuser le contenu de la disquette jointe à ce livre.

La disquette a été formatée sur un PC sous Windows 95. Elle est lisible par les PC et les Macintosh munis d'un lecteur FDHD. La disquette contient :

- un répertoire DUNOD_EX rassemblant les exemples du premier chapitre,
- un répertoire DUNOD_1 rassemblant les différents objets de la troisième partie de ce livre.

Pour connaître le chemin d'accès à un fichier, consulter au chapitre 27 le paragraphe consacré à la disquette.

■ Réglage d'une HP-48 S/SX

Placez-vous dans le répertoire où devront être rangés les fichiers reçus. Le menu de *I/O Setup* est accessible par [←] [PRG] [F]. Utilisez ensuite les touches [A] à [F] pour remplir les rubriques comme ci-dessous.

```
IR/WIRE : WIRE
ASCII/BINARY : BINARY
BAUD : 9600
PARITY : NONE 0.
```

Revenez ensuite au menu principal de I/O et choisissez l'article RECV du menu pour que la HP-48 reçoive les données par son port série.

■ Réglage d'une HP-48 G/GX

Placez-vous dans le répertoire où devront être rangés les fichiers reçus. Appuyez sur [↱] puis sur [1]. Choisissez *Transfer...* dans la liste. Pour sélectionner une rubrique, déplacez-vous avec les flèches de déplacement. Pour changer le contenu d'une rubrique, appuyez sur [+/-]. Affectez aux différentes rubriques les paramètres suivants :

```
PORT : WIRE
TYPE : KERMIT
FMT : BIN
BAUD : 9600
PARITY : NONE
```

Choisissez ensuite l'article RECV du menu pour que la HP-48 reçoive les données par son port série.

■ Réglage d'un PC utilisant KERMIT pour MS-DOS

Votre souris occupe généralement le port 1, à l'invite `Kermit-MS>`, saisissez donc : **`SET PORT 2`**. Ceci est valable si votre HP-48 est reliée au PC par le port série n°2. Saisissez ensuite **`SET BAUD 9600`** et **`SET PARITY NONE`**. Vous pouvez saisir **`HELP`** pour obtenir des informations sur la syntaxe des commandes. Tapez **`EXIT`** pour quitter le programme.

En admettant que votre lecteur de disquette 3,5' soit le lecteur A, pour transférer le fichier CD01 du dossier DUNOD_EX vous saisirez : **`SEND A:\dunod_ex\cd01`**

■ Réglage d'un Macintosh

Votre Macintosh doit être muni d'un lecteur FDHD (c'est le cas de tous les Macintosh depuis 1989) et du tableau de bord *Echange PC/Mac* ou de *Apple File Exchange*. Il existe de nombreux utilitaires de communication offrant le protocole KERMIT sur Macintosh. Quel que soit cet utilitaire, choisissez le port série convenable, 9600 bps pour la vitesse, pas de parité et le mode binaire. Si vous utilisez Apple File Exchange, choisissez la traduction par défaut.

Les clubs d'utilisateurs

■ Clubs francophones d'utilisateurs de HP-48

48SX Tant
Adresse : Lijsterlaan 31, 2665 TH Bleiswijk (Pays-Bas)
Prix au numéro : 35 FF
Activité : bulletin d'information et de programmation
Remarque : l'un des clubs francophones les plus sérieux

Club Calculatrices
Adresse : SETORGA, BP 203-16, 75765 Paris Cedex 16
Cotisation : aucune
Activité : bulletin d'information et de programmation
Remarque : club ouvert à tous les types de calculatrices

CHP - Club Hexagone Programmation
Adresse : CHP, 9 route d'Orléans, 63200 Riom
Cotisation : 30 FF par numéro
Activité : bulletin d'information et de programmation
Remarque : petit club très sympathique

Haute Performance
Adresse : FFJM, 1 av. Foch, 94700 Maisons Alfort
Cotisation : 125 FF par an
Activité : bulletin d'information et de programmation

PPC-Paris
Adresse : BP 604, 75028 Paris Cedex 01
Cotisation : 350 FF par an
Activités : réunions et bulletin de programmation

HP-Toulon
Adresse : HP Rafy, 26 allée des roses, 83190 Ollioules
Cotisation : 70 FF par an
Activité : bulletin d'information et de programmation

■ Serveur Minitel

3615 CALCULATOR
Editeur : Suite 1024
Coût de la connexion : 0,13 F puis 3 U.T./min.
Remarque : le plus important serveur dédié aux calculatrices

■ Presse traditionnelle

Poche Computers
Editeur : Microcards, 42220 Burdignes
Abonnement : 180 FF pour 6 numéros
Remarque : le seul magazine dédié aux calculatrices

Génération Electronique
Editeur : Publications Ventillard
Prix au numéro : 15 FF
Remarque : journal d'électronique avec une rubrique mensuelle consacrée aux calculatrices.

Glossaire

@ Symbole généralement utilisé pour représenter une adresse en mémoire.

A (champ) Ce champ correspond aux 5 quartets de poids faible d'un registre de travail.

A (registre) Registre de travail offrant de nombreuses possibilités de traitement.

Accès aléatoire Accès immédiat à un point donné de la mémoire représenté par son adresse.

Accès séquentiel Accès à un point donné d'une mémoire après avoir parcouru toute cette mémoire depuis son début jusqu'au point recherché.

Access pointer Voir *pointeur étendu.*

Adresse Numéro d'un quartet en mémoire. Les adresses sont codées en mémoire à l'aide de 5 quartets de #00000h à #FFFFFh.

Algorithme Décomposition d'un travail à effectuer en travaux élémentaires compte tenu de l'exécutant.

Alpha-lock Verrouillage du clavier en mode alphabétique.

And Voir *Et logique.*

Arborescence (menus) Hiérarchie des menus permettant l'appel d'un sous-menu à partir d'un article d'un autre menu.

Arborescence (répertoires) Emboîtement des répertoires (dossiers) les uns dans les autres.

Arité Nombre d'arguments ou de paramètres qu'attend un opérateur, une commande ou une fonction.

ASCII *American Standard Code for Information Interchange* Table faisant correspondre la valeur d'un octet (2 quartets) avec un caractère usuel (lettre, chiffre, symbole, etc.).

Assembleur (langage) Langage de programmation permettant de contrôler la machine au plus bas niveau en intervenant directement au niveau du microprocesseur. L'assembleur est une représentation « lisible » du langage machine qui lui n'est constitué que de 0 et de 1. L'assembleur permet de représenter ces 0 et ces 1 par des chiffres hexadécimaux (représentations d'un groupe de 4 bits, ou quartets) ou par des mnémoniques.

Assembleur (programme) Programme chargé de remplacer les mnémoniques utilisés en langage assembleur par les codes hexadécimaux correspondants, ou mieux, par le programme exécutable. Un assembleur est un outil de programmation en langage assembleur (on dit aussi langage d'assemblage).

B (champ) Deux quartets de poids faible d'un registre de travail.

B (registre) L'un des quatre registres de travail.

Balayage Rafraîchissement ligne par ligne du contenu de l'écran.

Bank switching Sélection de l'une des portions de 128 ko d'une carte en port 2 sur HP-48 GX.

Base 10 Base de numération décimale usuelle.

Base 16 Base de numération hexadécimale utilisée en informatique car un chiffre hexadécimal peut représenter toutes les combinaisons possibles de 4 bits.

Base 2 Base de numération binaire. Les seuls chiffres autorisés sont 0 et 1 représentant les deux seuls états logiques reconnus en électronique numérique.

Base de numération Détermine les chiffres utilisables (0 et 1 en base 2, de 0 à 9 en base 10, de 0 à 9 et de A à F en base 16).

BCD Voir *DCB.*

Bibliothèque Objet particulier réunissant d'autres objets nommés. Une bibliothèque a une structure à rapprocher de celle d'un répertoire mais doit aussi être considérée comme un ensemble de commandes.

Binaire Voir *Base 2.*

Binary digit Voir *Bit.*

Bit Nombre binaire. Un bit vaut 0 ou 1. Un groupe de 4 bits est un quartet, un groupe de 8 bits est un octet.

Bit armé Bit ayant la valeur 1.

Bit désarmé Bit ayant la valeur 0.

Branchement Reprise de l'exécution d'un programme à partir d'une instruction appelée « point de branchement » et ne se trouvant pas immédiatement après la dernière instruction exécutée.

Buffer Voir *Mémoire tampon.*

Byte Voir *Octet.*

BYTES Commande accessible en RPL et permettant de connaître le nombre de quartets occupés en mémoire par un objet ainsi que le checksum (somme de contrôle) de celui-ci.

C Registre de travail offrant le plus de possibilités.

Caractère Lettre, chiffre ou symbole codé sur 1 octet (c'est-à-dire 2 quartets ou 8 bits).

Chaîne de caractères Ensemble de caractères juxtaposés. L'objet chaîne de caractères accessible à l'utilisateur est délimité par " et ".

Champ d'un registre Partie nommée d'un registre de travail. Une opération sur un registre peut être limitée à l'un de ses champs.

Checksum Somme de contrôle ou CRC permettant de vérifier l'intégrité d'un objet. La commande BYTES en RPL renvoie - entre autres - le checksum d'un objet de l'utilisateur.

Chemin courant Liste des répertoires ouverts pour accéder aux variables actuellement disponibles (affichées après une pression sur [VAR]). Le chemin courant est affiché en haut à gauche de l'écran. Le premier répertoire du chemin courant est le répertoire racine (HOME) alors que le dernier répertoire du chemin courant est le répertoire courant (celui dont les variables sont actuellement disponibles).

Chemin d'accès Liste des répertoires à ouvrir pour accéder à une variable.

Chip « Puce électronique » ou module électronique, composant électronique prenant en charge une tâche déterminée.

Code de redondance cyclique CRC, voir *Checksum*.

Coma Mode de fonctionnement « basse consommation » de la HP-48.

Complément Inversion de bits (voir les chapitres d'initiation à la logique).

CRC Code de redondance cyclique, voir *Checksum*.

D L'un des 4 registres de travail.

D0 Registre contenant une adresse.

D1 Registre contenant une adresse.

Daisy chain Chaîne logique entre les différents modules de mémoire. Détermine la configuration de la mémoire.

DCB Décimal codé binaire. Représentation sur un quartet des chiffres de 0 à 9. Comme un quartet peut aussi représenter les chiffres hexadécimaux A à F, le DCB consiste à interdire ces représentations pour se limiter aux nombres de la base 10.

Décimal codé binaire Voir *DCB*.

Décrémentation Diminution, généralement par pas de 1.

Désassembleur Programme chargé de transformer un code exécutable en code hexadécimal pouvant être édité.

Drapeau Bit (ou, plus rarement, un quartet) chargé d'interdire ou d'autoriser une fonction. Les drapeaux peuvent être comparés à des interrupteurs paramétrant le fonctionnement de la calculatrice en « allumant » ou en « éteignant » certaines fonctions.

Drapeau armé Le bit représentant le drapeau vaut 1. Voir aussi *Drapeau*.

Drapeau désarmé Le bit représentant le drapeau vaut 0. Voir aussi *Drapeau*.

Editer Créer ou modifier des données. A l'origine le verbe éditer est synonyme de corriger.

Editeur Programme utilisé pour créer ou modifier des données.

Entrées-sorties Ensemble de moyens dont dispose la machine pour échanger des informations.

Environnement Ensemble de variables visibles à un instant donné. Dans le cadre d'une utilisation classique de la HP-48, les variables visibles sont celles du répertoire courant et celles de tous les répertoires du chemin courant. L'utilisation des applications intégrées telles que Equation Writer ou Matrix vous place dans un environnement particulier.

Epilogue Ensemble de quartets (5 généralement) chargé de signaler la fin du code décrivant un objet.

Equation Writer Application intégrée à la HP-48 et chargée de faciliter l'introduction

d'expressions mathématiques. Cette application se charge de la conversion depuis la notation algébrique directe vers la notation polonaise inversée (RPN). L'application Equation Writer constitue un environnement personnalisé. Voir *RPN.*

Et logique Opération logique (produit logique) dont la table de vérité est :

0 *and* 0 = 0
0 *and* 1 = 0
1 *and* 0 = 0
1 *and* 1 = 1

Evaluation Obtention de la valeur d'une variable. Dans le cas d'une variable non exécutable (nombre, caractère, chaîne de caractères, etc.), l'évaluation de la variable correspond à la valeur constante de la variable (selon le cas, le nombre, le caractère, etc.). Si la variable est exécutable (programme RPL, code exécutable, etc.), son évaluation correspond à l'exécution du programme.

Flag Voir *Drapeau.*

Flag armé Voir *Drapeau armé.*

Flag désarmé Voir *Drapeau désarmé.*

Garbage collection Opération effectuée régulièrement par un système informatique afin de libérer les espaces occupés par des variables devenues inutiles. La zone où sont stockés les objets temporaires est débarrassée des objets temporaires inutilisés lors d'une Garbage collection.

Glaneur de cellules Voir *Garbage Collection.*

GROB Objet graphique en mode point (objet graphique bitmap). Il s'agit du codage pixel par pixel d'une image destinée à l'affichage. Voir *Pixel.*

GROB courant Objet graphique pour l'instant affiché. En fonctionnement normal, la HP-48 peut passer de l'affichage de la pile à celui de l'environnement graphique GRAPH ou PICTURE (selon la version). C'est donc soit le GROB pile, soit le GROB environnement graphique qui est affiché et est donc le GROB courant.

GROB environnement graphique Objet graphique décrivant pixel par pixel l'affichage à l'écran de l'environnement GRAPH ou PICTURE (selon la version). Voir *Pixel.*

GROB menus Objet graphique décrivant pixel par pixel le contenu de la barre des menus en bas de l'écran. Voir *Pixel.*

GROB pile Objet graphique décrivant pixel par pixel la représentation à l'écran du contenu de la pile. Voir *Pixel.*

Hexa Voir *Hexadécimal.*

Hexadécimal Utilisation de la base 16 pour les calculs. Voir aussi *Base 16.*

HOME Nom donné au répertoire racine de la HP-48. Voir *Répertoire racine.*

HP-48 G Version apparue en 1993, dotée de 32 ko de mémoire vive interne et de 512 ko de ROM mais dépourvue de lecteur de carte.

HP-48 GX Version apparue en 1993, dotée de 128 ko de mémoire vive interne, de 512 ko de ROM et de deux lecteurs de cartes dont l'un supporte jusqu'à 4 Mo.

HP-48 S Version simplifiée de la HP-48 SX apparue en 1991. Dépourvue de lecteur de cartes.

HP-48 SX Première calculatrice de la série HP-48 apparue en 1989. Dotée de 32 ko de mémoire vive interne, de 256 ko de ROM et de deux lecteurs de cartes.

I/O Input/Output Voir *Entrées/sorties.*

IN et OUT (registres) Registres utilisés pour connaître les touches pressées.

Incrémentation Augmentation, généralement par pas de 1.

Indicateur Voir *Témoin.*

Kermit Protocole de transmission utilisé par la HP-48 pour échanger des données via le câble série (RS 232) ou les diodes infrarouge. Voir aussi *RS 232.*

Kilo octet 1024 octets (2^{10}). Voir *Octet.*

Langage machine Langage constitué d'ordres directement compréhensibles par le microprocesseur codés par des 0 et des 1 (c'est un langage binaire). Le langage assembleur n'est qu'une représentation « humainement lisible » du langage machine.

LCD Procédé d'affichage par cristaux liquides utilisé pour les écrans de la plupart des calculatrices.

Liaison (communication) Lien physique (câble) ou optique (faisceau infrarouge) entre la HP-48 et un autre dispositif (un micro-ordinateur, une autre HP-48, l'imprimante HP, etc.).

Liaison (variable) Association d'un objet à un nom (placé entre ' et ') afin de créer une variable qui n'est qu'un couple constitué d'un objet et de son nom (ce nom est aussi un objet). Un environnement est un ensemble de liaisons, et donc, un ensemble de variables. Voir *Environnement, Variable.*

Library Voir *Bibliothèque.*

Masquage des mémoires Le SATURN a une capacité d'adressage limitée à #FFFFFh adresses différentes. Cela impose l'utilisation d'une même adresse au sein de plusieurs modules de mémoire différents. Pour éviter les confusions, il existe une hiérarchie entre les modules de mémoire. Si deux modules utilisent la même adresse, celle-ci, lorsqu'elle employée, désigne un quartet du module de mémoire le plus prioritaire. Un quartet de même adresse dans un autre module moins prioritaire est donc masqué. Un module de mémoire masqué est accessible après une reconfiguration de la mémoire. Voir *Mémoire.*

Matrix Writer Application HP d'édition de matrices (et de vecteurs).

Méga octet 1048576 octets (2^{20}). Voir *Octet.*

Mémoire Dispositif électronique de sauvegarde des données informatiques. Il existe six modules de mémoire dans les HP-48 (G/GX) : la mémoire vive des entrées/sorties, la mémoire vive interne, le gestionnaire de partitions, la carte en port 1, la carte en port 2 et la mémoire morte.

Mémoire masquée Voir *Masquage des mémoires.*

Mémoire morte Mémoire dont le contenu a été figé lors de la construction de la machine, il est donc impossible d'y écrire des données. Elle contient toutes les données et toutes les fonctions disponibles en standard ainsi que certains programmes (notamment, pour le démarrage de la calculatrice). Sa capacité est de 256 ko sur les HP-48 S/SX et de 512 ko sur HP-48 G/GX.

Mémoire tampon Mémoire de faible capacité (en général, il s'agit d'une partie de la RAM système ou de la RAM I/O) chargée d'absorber un flot de données entrantes ou de conserver peu de temps des données à émettre vers un périphérique.

Mémoire vive Mémoire où il est à la fois possible d'écrire et de lire des données. Les premiers quartets de la mémoire vive cor-

respondent à la mémoire vive réservée au système. Les quartets suivants contiennent - entre autres - les variables créées par l'utilisateur (programmes, données diverses, etc.). La mémoire vive interne (celle de la HP-48 proprement dite) peut être complétée par des cartes de mémoire vive.

Mémoire vive des entrées/sorties Module de mémoire ayant le plus haut niveau de priorité. Contient des données relatives aux entrées/sorties (avec l'écran entre autres).

Mémoire vive interne Mémoire vive de la HP-48 sans ajout de carte de mémoire vive.

Mémoire vive réservée au système Les premiers quartets de la mémoire vive sont réservés à la HP-48 qui y stocke de nombreux paramètres.

Menu courant Menu affiché à l'instant considéré.

Microprocesseur Organe d'un système micro-informatique chargé des traitements.

Mnémoniques Représentations mnémotechniques des codes hexadécimaux utilisés en assembleur. Ces codes hexadécimaux sont eux-mêmes la représentation directe des bits (des 0 et des 1) constituant le langage machine.

Mode courant Mode d'utilisation de la calculatrice à l'instant considéré.

Nibble Voir *Quartet.*

Nombre binaire voir *Bit.*

Non logique Opération logique consistant à remplacer tous les bits à 1 d'un mot binaire par des bits à 0 et inversement.

Not Voir *Non logique.*

Objet Donnée « encapsulée » dans la mémoire. Un objet est d'un type défini par son prologue et peut être associé à un objet de type nom pour créer une variable.

Octet Groupe de 8 bits, c'est-à-dire, de 2 quartets. Voir *Bit.*

OR Voir *OU logique.*

Ou logique Opération logique (somme logique) dont la table de vérité est :

0 *or* 0 = 0
0 *or* 1 = 1
1 *or* 0 = 1
1 *or* 1 = 1

Ou logique exclusif Opération logique dont la table de vérité est :

0 *xor* 0 = 0
0 *xor* 1 = 1
1 *xor* 0 = 1
1 *xor* 1 = 0

P (champ) Le registre P contient un nombre compris entre #0h et #Fh. Ce nombre est le numéro du quartet constituant le champ P des registres de travail.

P (registre) Le nombre compris entre #0h et #Fh que contient ce registre détermine le numéro du quartet constituant le champ P des registres de travail.

Partition L'une des tranches de 128 ko d'une carte pour le port 2 des HP-48 GX.

PC Registre du SATURN contenant l'adresse de la prochaine instruction à exécuter.

Pile Structure de données de type LIFO (*Last In First Out*), « le dernier introduit est le premier sorti ».

Pixel *Picture Element.* Point de l'écran.

Poids Valeur d'une portion d'un mot binaire par rapport au reste du mot. Les bits de

poids fort (MSB, bits les plus significatifs) se trouvent à gauche du mot binaire. Les bits de poids faible (LSB, bits les moins significatifs) sont à droite du mot binaire. De même, on peut par exemple diviser un octet en deux quartets, l'un de poids fort, l'autre de poids faible.

Pointeur Registre ou champ en mémoire contenant une adresse en mémoire.

Pointeur étendu Objet créé en associant l'adresse d'un objet placé dans une mémoire masquée et l'adresse d'une routine rendant accessible ladite mémoire masquée.

Priorité des mémoires Elle détermine le module de mémoire utilisé lorsqu'une même adresse existe dans plusieurs modules. En configuration normale et sur HP-48 G/GX, les modules sont, par ordre de priorité décroissante : la RAM I/O, la RAM interne, le gestionnaire de partitions, la carte du port 1, celle du port 2 et la ROM. Voir *Mémoire.*

Processeur Voir *Microprocesseur.*

Program Counter Voir *PC.*

Prologue Champ d'un objet déterminant le type d'un objet. Plus généralement, entête représentative d'une zone de mémoire.

Quartet Groupe de 4 bits. Un quartet est représenté par un nombre hexadécimal. Voir aussi *Base 16, Bit, Hexadécimal* et *Octet.*

R0, R1, R2, R3 et **R4** sont les noms des registres de sauvegardes. On y sauvegarde les registres de travail.

Racine Voir *Répertoire racine.*

Rafraîchissement Renouvellement régulier du contenu de l'écran à l'aide d'une image identique à la précédente si l'affichage n'a pas été modifié.

RAM Voir *mémoire vive.*

RAM I/O Voir *mémoire vive des entrées/sorties.*

RAM système Voir *mémoire vive réservée au système.*

Ramasse-miettes Voir *Garbage Collection.*

Récupérateur de mémoire Voir *Garbage Collection.*

Registre Petite mémoire interne du microprocesseur. Le microprocesseur effectue ses traitements à partir du contenu des registres. Voir *registre.*

Registre de calcul Voir *Registre de travail.*

Registre de sauvegarde Voir *R0, R1, R2...*

Registre de travail Registres A, B, C et D. Ce sont les registres qui offrent le plus de possibilités de traitement.

Répertoire Sorte de dossier dans lequel se trouve un ensemble de variables. Un répertoire peut contenir d'autres répertoires qui peuvent, eux-mêmes, en contenir encore d'autres. C'est cet emboîtement de répertoires qui définit l'arborescence de la mémoire. En fait, un répertoire est une variable contenant des noms de variables.

Répertoire racine Ce répertoire contient tous les autres et se trouve donc à la « racine » de l'arborescence des répertoires. Ce répertoire s'appelle HOME dans le cas de la HP-48. Voir *Répertoire.*

Retour de sous-programme Saut vers la dernière adresse placée dans la pile RSTK.

ROM Voir *mémoire morte.*

RPL (environnement RPL) Nom souvent donné à l'interface de la HP-48 dans le

cadre de son utilisation normale (pile, langage RPL, fonctions et outils intégrés, etc.).

RPL (langage) *Reverse Polish LISP* Langage de programmation standard de la HP-48. A la fois apparenté au LISP et au FORTH. Utilise la notation polonaise inversée (RPN). Voir *RPN.*

RPN *Reverse Polish Notation* Notation postfixe imposant l'introduction des arguments d'une fonction avant celle-ci.

RS 232 Norme de communication en série utilisée pour la communication par câble entre la HP-48 et un micro-ordinateur.

RSTK *Return Stack* Pile des retours contenant les adresses de retours de sous-programme. Cette pile est apparentée à un registre de 8 niveaux de 5 quartets. Chaque niveau peut accueillir une adresse. Deux de ses niveaux doivent rester disponibles en permanence

SATURN Microprocesseur de la HP-48. Voir *Microprocesseur.*

Saut Voir *Branchement.*

ST *Status Bits* Registre de 16 bits. Chacun de ces bits a un rôle de drapeau pour le fonctionnement du SATURN. Voir *Drapeau Registre* et *SATURN.*

Table de vérité Tableau résumant le comportement d'une fonction logique pour toutes les valeurs possibles de ses arguments.

Tampon Voir *Mémoire tampon.*

Témoin Symbole qui peut être affiché en haut de l'écran pour indiquer un mode de saisie particulier (touche α, touches shift), une sonnerie, une transmission en cours, etc.

Test Comparaison du contenu d'un registre de travail avec zéro ou avec le contenu d'un autre registre de travail. Le test peut vérifier l'égalité, l'inégalité, la supériorité stricte ou large et l'infériorité stricte ou large. Si la condition testée est remplie, un saut relatif court est exécuté. Voir *Saut.*

Tétrade Voir *Quartet.*

Variable Association d'un objet à un nom afin de stocker en mémoire cet objet sous le nom qui vient de lui être affecté.

Visibilité des mémoires La mémoire visible est la mémoire accessible à l'instant considéré. Voir *Masquage des mémoires.*

Visibilité des variables Les seules variables visibles sont celles appartenant aux répertoires du chemin courant. Voir *Répertoire, Variable, Chemin d'accès* et *Chemin courant.*

W (champ) Ce champ correspond aux 16 quartets d'un registre de travail (le champ W correspond à l'intégralité du registre).

WP (champ) Ce champ est l'ensemble de quartets depuis le quartet n°0 d'un registre de travail jusqu'à son quartet n°P. P est ici la valeur (comprise entre #0h et #Fh) que contient le registre P. Voir aussi *P (champ)* et *P (registre).*

XOR Voir *OU logique exclusif.*

Index

A

A (champ) 26, 147
A (registre) 24, 145, 163
ABEX 180
ABIT=0 179
ABIT=1 179
Absorption 121
Accès à une bibliothèque 327
Accès au port 1 153
Accès au port 2 153
ACEX 180
Activation de l'écran 369
Addition binaire 89
Addition d'un champ 182
Addition d'une constante 183
Addition fractionnée 90
Adresse 134, 140
Adresse d'un pixel 370
Adresse de départ d'un module 240
Adresse de l'écran (bitmap) 337
Adresse de la bitmap menu 338
Adresse de la RAM interne 336
Adresse des objets temporaires 358
Adresse du début de la mémoire vive 345
Adresse du GROB non affiché (inactif) 345
Adresse du GROB Pict 345
Affichage 64
Affichage d'un nombre 71
Algèbre de Boole 115, 119
Algorithme 132
Allumer un point isolé 67
ALPHA pour HP-48 G et GX 539
ALPHA pour HP-48 S et SX 403
Appel de sous-programme 57, 216
Architecture de base 133
Armer le bit de retenue 136
ASCII 125
ASCII 7 bits 125
ASCII HP-48 129
ASCII standard 126
ASMFlash 76
ASS 29, 30, 292, 392
Assembleur 16
Assistance technique 702
Associativité 121

B

B (champ) 26, 147
B (registre) 24, 145, 163
Bank driver 153
Bank switcher 154
Base 1 78
Base 10 78, 106
Base 16 103, 105
Base 2 77
Base 60 77
Base 8 103
BCD 111
BCEX 180
Bibliographie 681
Bibliothèque 314
Binaire 77

Binaire naturel 77
Binaire signé 91
Binaire système 294
Binary digit 80
Bit 19, 80
Bit de retenue 76, 136, 144, 184, 227
Bitmap 64
Bitmap écran 337
Bitmap menu 338
Bits de statut 144, 232
Bits de statut matériel 234
Blocage de l'affichage 371
Boole 115
Boucle 55
Bus 134, 151, 236
Bus multiplexé 151
Bus [0:3] 151
Byte 81

C

C 24, 145
C=IN 60, 368
C=RSTK 231
CAEX 180
Calculator 703
Calculs en base 2 89
Calculs en DCB 113
Calculs sexagésimaux 77
Caractère 299
Carré de la base 79
Carry 136, 144, 145, 184, 227
CBIT=0 179
CBIT=1 179
Central Processing Unit 133
Chaîne de caractères 303
Champ d'un registre 25, 147, 148
Champs de la RAM système 339
Champs définis par l'utilisateur 148
Checksum 31, 333
Circuit de décodage 135
Circuit logique 92
Clubs 685
Codage des quartets 103, 108
Codage par complément à 2 94
Code ASCII 42, 125
Code buffer de clavier 289
Code exécutable 324
Codes hexadécimaux 249, 269
Codes WSLOG 385
Commutativité 121
Comparaison de registres 221
Complément à 1 93, 187
Complément à 2 92, 93, 186
Compteur ordinal 27, 135, 143
CONFIG 240
Configuration de la mémoire 240
Configuration matérielle du SATURN 149
Configurer un module de mémoire 242
Conversion 107
Coordonnées du premier pixel 370
Copie d'un registre 180
Copie vers un registre de sauvegarde 193
CPU 133
CRC 31, 333, 376, 396
Création d'objet avec ASS 76
Création de code exécutable 35
Cycles du microprocesseur 136

D

D 24, 145
D0 24, 25, 145, 199
D1 24, 25, 39, 145, 199
Daisy-chain 155, 165
Daisy-in 155
Daisy-out 155
DALPHA 390
DCB 111, 139
DCB compacté 112
DCB étendu 112
DCEX 180
Décalage d'un bit 191
Décimal 78
Décimal codé binaire 111
Décodage 135
Décodage des instructions 138
Déconfiguration d'un module de mémoire 241
Décrémentation 184
Délimiteurs 395
Délimiteurs d'objets 329
Dépassement de capacité 113
Déplacement de la totalité de la RAM 375
Désactivation de l'écran 369
Désarmer le bit de retenue 136
Détecter une pression sur une touche 60
Différence 92

Digit 80
Digital 80
Diminuteur 92
Dimunuende 92
Disquette jointe au livre 683
Distributivité 121
Division binaire 99
Division de A(W) par C(W) 374
Division par décalage 189
Données de bibliothèque 321
Données traitantes 22, 131
Données traitées 22, 131
Double précision 102, 329
Drapeau 136, 144
Drapeaux du système 362
Drapeaux internes 359
Drapeaux utilisateur 363
DSETH 390
DSTAR 390
DXALK 390

E

Ecriture 28
Ecriture en mémoire 205
Electronique numérique 77
Emission d'un son 62
ENIAC 77
Entier 304
Entrées et sorties 143
EqLIb 133
ET logique 117, 188
ET-NON 118
Etat (du microprocesseur) 137
Etat d'un pixel 371
Exécution d'une instruction 135
Exécution d'une routine en ROM cachée 375
Expression algébrique 310
Extinction (coma) 378
Extinction de l'écran 332

F

Flag 136
Flux de données 28
Fonctions logiques 115
Forme canonique d'un nombre 81
Forth 14

G

Gérer le résultat d'un test 227
Gestionnaire de module 155
Glossaire 687
GOC 211
GONC 54
GOSBVL 01160 60
GOSBVL 0679B 37
GOSBVL 067D2 37
Goto 51
GOYES 45
GROB 395
GROB Menu 64
GROB Pict 64
GROB Stack 64
Grouper les bits 80

H

Hardware du SATURN 149
Hardware status bits (HS ou HST) 144
Hash table 316
Hauteur du menu 337
Hexadécimal 19, 106
Hiérarchie des mémoires 154
HOME 306
Hot-line 702
HP Saturn Assembler 76
HP-28 137
HP-71 137, 147, 164
HS 28, 136, 144, 234
HST 136, 144, 234

I

IN 28, 145, 231, 289
Infériorité et supériorité larges 225
Infériorité et supériorité strictes 224
Initialisation du buffer de clavier 367
Instructions 135, 173, 249, 269
Instructions équivalentes 176
Instructions relatives au bus 236
Instructions sans effet 238
Interruption 138, 236
INTOFF 37
INTON 37
Inversion (d'un bit) 118
Inversion des quartets 141
Inversion vidéo 64

J
Jeu d'instructions 173

L
LA 177, 178
LAHEX 178
Langage assembleur 16, 132
Langage d'assemblage 132
Langage de bas niveau 132
Langage de haut niveau 132
Langage machine 16
Langage RPL 15
Langage RPL système 15
LC 178
LCHEX 178
Least significant bit 86
Lecture 28
Lecture en mémoire 203
Link table 319
Lisp 14
Liste 305
Liste des clubs 685
LOAD DP 151
LOAD PC 151
Logique 115
Longueur d'un mot binaire 100
LSB 86

M
M 26, 147, 163
Machine physique 132
Machine virtuelle 132
Marge à droite 64, 337
Marge à gauche (décalage à gauche) 64, 332
Masquage de la pile 370
Masque IN 289
Masque OUT 289
Masques de clavier 289
Max-terme 123
Mémoire 22, 152
Mémoire (HP-48 G) 167
Mémoire (HP-48 GX) 168 à 171
Mémoire de masse 133
Mémoire disponible 377
Mémoire masquée 239
Mémoire vive 133
Mémoire vive des entrées/sorties 331
Mémoire vive réservée au système 339
Message table 319
Microcycle 136, 174
Microprocesseur 23, 131
Min-terme 123
Mnémoniques 249, 269
Mode décimal 235
Mode hexadécimal 235
Module de mémoire 23, 155, 166
Most significant bit 86
Mot binaire 81
MP 144, 234
MSB 86
Multiplication binaire 97
Multiplication de A(A) par C(A) 373
Multiplication de A(W) par C(W) 373
Multiplication par additions successives 97
Multiplication par décalage 189
Multiplication par décalage et addition 98

N
NAND 118
Niveau bas (niveau logique) 77
Niveau haut (niveau logique) 77
Nom global 325
Nom local 326
Nombre complexe 297
Nombre complexe double précision 298
Nombre rationnel 86
Nombre réel 295
Nombre réel double précision 296
NOP 3 238
NOP 4 238
NOP 5 238
NOR 118
Notation polonaise inversée 14
Notation scientifique 102
Notations négatives 101
Nullité d'un registre 222
Numérique 80

O
Objet avec étiquette 312
Objet avec unité 311
Objet graphique 313
Objet taggué 312
Objets 291
Objet graphique 396

Objets réservés 322
Octet 80
Opérations logiques 187
OU logique 116, 187
OU-NON 118
OUT 28, 145, 231, 289
OUT=C 60
OUT=C puis C=IN 368

P

P (champ) 27, 164
P (registre) 27, 145, 229
P= 0 37
Partitions de 128 ko 153
PC 24, 25, 135, 145
Périphérique 138
Pile des retours 144, 231
Poids faible 85
Poids fort 85
Point de branchement 399
Pointer 399
Pointeur 24
Pointeur d'instruction 135
Pointeur de champ 145
Pointeur étendu 245, 322
Précision d'un mot binaire 100
Premier théorème de De Morgan 122
Pression sur une touche 60
Primitives RPL 328
Procédure 399
Processus de développement 18
Produit logique 117, 188
Program Counter 135
Programme 132
Programme assembleur 18, 76
Programme RPL 323
Prologue 02A2C 42
Prologue 02DCC 33
Propagation d'une retenue 113

Q

Quartet 19, 80, 103
Quartet de poids faible 86
Quartet de poids fort 86

R

R0 24
R1 24
R2 24
R3 24
R4 24
Rafraîchissement de l'écran 332
RAM 133, 152
RAM I/O 331
RAM système 339
Random Access Memory 133
Rang 87
Read Only Memory 133
Récupération des registres 366
Registre 24, 135, 136, 139, 142, 159
Registre d'adresse 159
Registre de calcul 142
Registre de sauvegarde 24, 142
Registre de travail 24, 142, 175
Registre pointeur 24, 143
Registres (notations) 162
Relations fondamentales (algèbre de Boole) 121
Répertoire 306
Répertoire racine 306
Retenue 144, 145, 186, 227
ROM 133, 152
ROM cachée (masquée) 378, 172, 239
Rotation d'un quartet 190
Routines d'accès à la ROM masquée 245
Routines en mémoire morte 365
RPL 14, 15
RPL système 15
RPN 14
RSTK 28, 144, 162, 231
RTNSXM 144
RTNYES 45

S

S 26, 147, 163
SATURN 110, 133, 137, 149
Saut absolu 56
Saut relatif conditionnel 52
Saut relatif inconditionnel 49
Sauts 209
Sauts relatifs, absolus et indirects 213
Sauvegarde 320
Sauvegarde des registres 366
SB 144, 234
Second théorème de De Morgan 122
SETDEC 12

SETH pour HP-48 G 619
SETH pour HP-48 S et SX 485
SETHEX 37
Shannon 115
Simplification des expressions 120
Somme logique 116, 187
Sous-programme 216
Sous-répertoire 308
Soustraction 90, 185
Soustraction binaire 92
Soustraction d'une constante 185
Soustraction par complément à 2 93
Sprite 399
SR 144, 234
ST 28, 136, 144, 232
STAR pour HP-48 G et GX 567
STAR pour HP-48 S et SX 431
Status bits (ST) 144
Structure de la mémoire 152
Structure des exemples de programme 673
Structure des objets 291
SWAP 39
Symétrie 121

T

Tableau 300
Tableau indexé 302
Taille d'un module de mémoire 240
Taille de la mémoire vive interne 354
Télématique 685
Témoins 333, 334
Test de l'état d'un bit 226
Tests 45, 209, 219
Théorème de De Morgan 122
Transfert bit à bit par le port infrarouge 381
Transitivité 121
Transmission en parallèle 134
Transmission en série 134

U

UAL 135, 138, 151
UCT 133
UNCNFG 240
Unité arithmétique et logique 135
Unité d'information 140
Unité de séquencement et de commande 135
Utilisation d'un signe 90

V

Valeur absolue plus le signe 90
Valeur initiale d'un registre 177
Virgule fixe 101
Virgule flottante 101

W

W 26, 146, 163
WP 27, 164
WSLOG 379, 385

X

X 26, 147, 164
XALK pour HP-48 G 589
XALK pour HP-48 S et SX 455
XM 144, 234
XS 26, 147, 164

La collection

CALCULATRICES EFFICACES

Les grands atouts de la "petite informatique"

- **Casio fx 7900/9900 : permis de conduire**
- **Votre Casio fx 6800G**
- **Casio fx : 300+*150* programmes**
- **Casio fx : faites vos jeux !**
- **Casio fx : jeux et graphisme**
- **Casio fx pour le bac !**
- **Casio fx : programmez votre succès !**
- **Casio fx : maths au lycée**
- **Programmation efficace sur Casio fx**
- **HP-48 G/GX : permis de conduire**
- **HP-48 G/GX pour le bac !**
- **HP-48 G/GX/S/SX : mathématiques en prépa**
- **HP-48 G/GX/S/SX : physique et chimie en prépa**
- **HP-48 G/GX/S/SX : faites vos jeux en assembleur !**
- **TI-81 : programmez votre succès !**
- **TI-81 : 300 programmes**
- **Jeux et graphisme sur TI-81**
- **TI-81 : le "top" des jeux !**
- **TI-82 : permis de conduire**
- **TI-82 : mathématiques au lycée**
- **TI-82 : programmes pour le lycée**
- **TI-82 : le "top" des jeux !**
- **TI-82/85 pour le bac !**
- **TI-85 : permis de conduire**
- **TI-85 du lycée à la prépa**
- **TI-85 : le "top" des jeux !**
- **Sharp EL 9200/9300 : faites vos jeux !**
- **Finance et gestion sur calculatrices**

DUNOD

Assistance technique

La « hot-line » des programmeurs sur HP-48

Un problème dans le cadre de l'utilisation de ce livre ?
Une question urgente ?

Le CLUB CALCULATRICES met a votre disposition un service d'assistance technique et répond à vos questions en 48 heures. Voici comment utiliser ce nouveau service :

- à l'aide d'un simple minitel, connectez-vous au service télématique 3615 CALCULATOR,
- sur ce service créez une boîte aux lettres (BAL) pour pouvoir recevoir des messages,
- posez votre question en laissant un message dans la boîte aux lettres CLUB,
- vous trouverez dans votre boîte aux lettres la réponse à votre question (délai à prévoir : 24 à 48 heures).

Ne restez pas bloqué par un petit problème ! Le CLUB est là pour vous aider !

Correspondance

Pour écrire à l'auteur :

Loïc Fieux/Collection "Calculatrices"
Dunod
15 rue Gossin
92543 Montrouge Cedex

VOTRE CALCULATRICE A TROUVE SON MAÎTRE !

- Plus de 12.000 écrans !
- Des milliers de programmes !
- Informations, Scoops...
- Trucs et Astuces
- Messagerie et Contacts
- Petites Annonces
- Echanges, Achats, Ventes
- Micro-Informatique
- *Casio™, HP™, T I™, Sharp™, etc.*

Editeur SUITE 1024 • 0,13 F Cx puis 3 U T min

3615 CALCULATOR

Le minitel de votre HP-48 !

GRATUIT !

Le CLUB CALCULATRICES

Vous avez des idées, vous trouvez des astuces, vous programmez et vous souhaitez savoir ce que d'autres ont pu découvrir... Alors sachez qu'il existe un club d'utilisateurs de calculatrices graphiques (fx, TI, hp, EL...). Le principe est simple : vous envoyez vos éventuelles découvertes et nous vous faisons parvenir la "Lettre du Club Calculatrices" qui rassemble les trouvailles de nos membres. Envoyez-nous dès aujourd'hui le coupon ci dessous, vous recevrez gratuitement la "Lettre du Club Calculatrices" !

Cochez les cases !

❑ Oui, je souhaite recevoir gratuitement la "***Lettre du Club Calculatrices***", je ne prends aucun engagement en envoyant ce coupon.
❑ Je vous envoie des astuces et/ou des programmes afin qu'ils soient publiés dans la "***Lettre du Club Calculatrices***". ❑ Je ne vous envoie ni programme ni astuce.

HP-48

Nom :.....Prénom :.....

Adresse :.....

Code postal .....Ville : .....Pays :.....

Complétez le coupon ci-dessus et envoyez-le à :
SETORGA • CLUB CALCULATRICES • BP 203-16 • 75765 Paris Cedex 16

QU'EN PENSEZ-VOUS ?

Afin de veiller à la qualité de nos livres, adressez-nous vos remarques, critiques et suggestions sur le présent ouvrage :

HP-48 : faites vos jeux en assembleur

Ce livre vous donne-t-il toute satisfaction ? .....

Avez-vous des commentaires à formuler ? .....

.....

.....

.....

.....

.....

.....

Quelle calculatrice pensez-vous acheter à l'avenir ? .....

.....

❒ M. ❒ Mme ❒ Mlle Nom.....Prénom .....

Profession ou activité .....

Adresse .....

Code postal |_|_|_|_|_| Ville .....

Pays .....

A découper et à envoyer à :

DUNOD / Courrier des lecteurs
BP 20
92122 MONTROUGE Cedex

En application de l'article 27 de la loi 78-17 Informatique et Liberté, vous disposez d'un droit d'accès et de rectification pour toute information vous concernant sur notre fichier. Dunod Editeur peut être amené à communiquer ces informations aux organismes qui lui sont liés contractuellement, sauf opposition de votre part notifiée par écrit.

Imprimé en France. - JOUVE, 18, rue Saint-Denis, 75001 Paris
N° 228619E. - Dépôt légal : Août 1995

CALCULATRICES

HP 48
faites vos jeux en assembleur

Disquette 3" 1/2 PC/Mac incluse, pour charger directement les programmes du livre sur votre HP 48, à l'aide du kit de connexion disponible chez votre revendeur Hewlett-Packard.

Le meilleur de votre HP 48 !

1 • L'initiation complète et progressive pour le débutant !

Dès le premier chapitre, le débutant sera guidé pas à pas pour la réalisation de son premier programme en assembleur ! De nombreux exemples abondamment commentés vous aideront à développer très vite vos propres programmes. Un véritable cours vous permettra d'assimiler toutes les notions indispensables pour programmer en assembleur : le binaire naturel, les calculs en base 2, l'hexadécimal, le DCB, les fonctions logiques, le code ASCII, l'architecture de la HP-48, le SATURN et son jeu d'instructions, etc. Bref, tout a été mis en œuvre pour vous faire progresser rapidement.

2 • Toutes les références dont vous devez disposer !

L'initié dispose d'un ensemble complet et cohérent d'informations pour le développement de programmes performants en assembleur, ce livre présente et explique : la structure des objets, la RAM du système, la RAM I/O, les codes WSLOG, les masques de clavier, quelques routines utiles en mémoire morte et bien d'autres choses !

3 • Les jeux

Des listings commentés de jeux en assembleur prêts à l'emploi vous permettront de comprendre comment se développe un jeu... Regardez comment d'autres font, et... A vous de jouer !

4 • La disquette

Lisible sur PC et Macintosh™, elle contient tous les exemples et tous les jeux développés dans ce livre. Bien sûr, le livre est aussi utilisable sans sa disquette.

Bref, tout - et le reste - pour vous jouer de votre HP-48 !

9 782100 023585

ISBN 2 10 002358 6 Code 042 358

Environnement : ***Calculateurs Hewlett-Packard HP 48 G, GX, S et SX***

Thème : ***Langage assembleur (initiation, références et applications ludiques)***

Utilisateur : ***Etudiant, Ingénieur, etc.***

DUNOD
EDITEUR