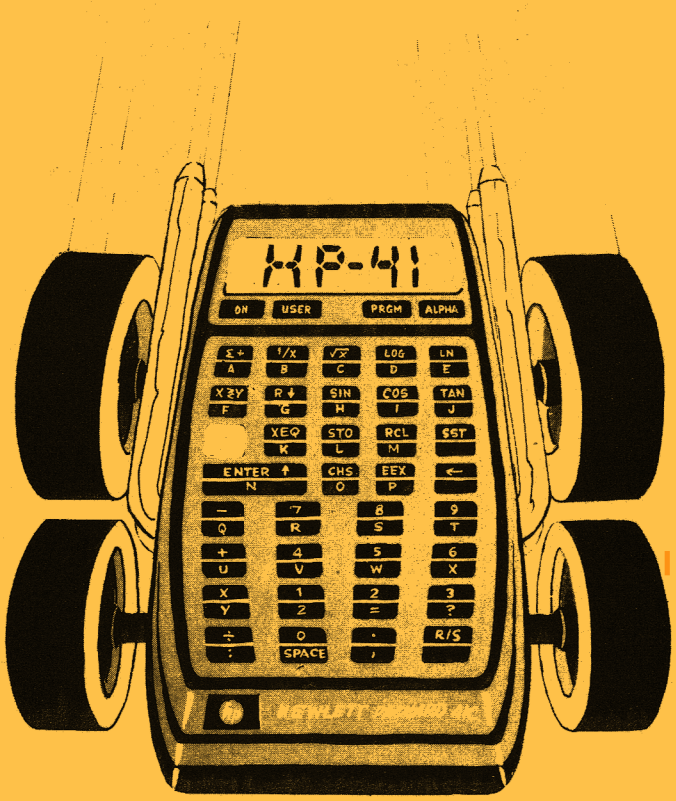
HP-41
LA PROGRAMMATION SYNTHETIQUE C'EST FACILE!

PAR KEITH JARETT



POUR HP-41C, HP-41CV OU HP-41CX

Editions du Cagire
77 rue du Cagire
31100 Toulouse France

LA PROGRAMMATION SYNTHETIQUE C'EST FACILE!

LA PROGRAMMATION SYNTHETIQUE C'EST FACILE!

PAR KEITH JARETT

Traduit de l'américain par Gilles Barret

77 rue du Cagire 31100 Toulouse France
SARL au capital de 20000F
ISBN 2-86811-007-X

Remerciements : Ce livre n'aurait jamais existé sans l'existence de PPC, club d'utilisateurs qui a inventé et développé la programmation synthétique dès 1979, date de la sortie de la 41C. Plusieurs membres de PPC ont largement contribué à l'élaboration des techniques détaillées dans cet ouvrage.

Un grand nombre d'entre elles sont dues à Clifford Stern, l'un des grands maîtres de la programmation synthétique. Clifford fut mon support technique pour l'écriture de ce livre, écrivant plusieurs programmes spécialement pour ce livre et traquant les erreurs lors de sa rédaction.

Beaucoup d'autres membres de PPC ont également contribué indirectement à l'écriture de ce livre, grâce à leurs découvertes et réalisations qui ont fait progresser la programmation synthétique ces dernières années. Richard Nelson, le fondateur de PPC, a droit à toute ma reconnaissance pour son dévouement sans borne envers PPC tout au long de ses 8 années d'existence.

Je dédie ce livre à ma femme, Catherine Van De Rosytne, qui a patiemment supporté ma dévotion à la 41C, et dont l'aide pour la préparation de ce livre a été inestimable.

La Carte des Codes plastifiée à la fin de ce livre est un outil de première importance pour une utilisation aisée de la programmation synthétique. Son maniement est décrit au premier chapitre. Pour plus de détails, consultez l'appendice D et l'appendice C, paragraphe 10.

Pour tout renseignement de prix concernant ce livre, écrivez à : **Editions du CAGIRE,** 77 rue du Cagire 31100 TOULOUSE FRANCE.

Joignez une enveloppe à votre adresse pour obtenir une réponse plus rapide. Des distributeurs et libraires sont recherchés partout dans le monde.

Table des matières

page

5 **INTRODUCTION -- QU'EST-CE QUE LA PROGRAMMATION SYNTHETIQUE ?**

Qu'est-ce que la programmation synthétique ?

8 **Chapitre 1 -- CREER VOTRE 1ère INSTRUCTION SYNTHETIQUE**

Comment assigner et utiliser le Byte Grabber
Comment utiliser la **Carte des Codes Plastifiée**

14 **Chapitre 2 -- LES INSTRUCTIONS SYNTHETIQUES LES PLUS COURANTES**

14 2A. TONEs synthétiques
16 2B. Exposants synthétiques
19 2C. Contrôle du registre des drapeaux
22 2D. Contrôle du pointeur de programme
24 2E. Chaînes de caractères synthétiques
29 2F. L'instruction TEXT 0
30 2G. Utilisation du registre alpha pour le stockage des données
34 2H. Utilisation des autres registres d'état pour le stockage

37 **Chapitre 3 -- CHARGEMENT D'OCTETS**

Comment créer et utiliser le programme **LB**
Comment créer n'importe quelle instruction synthétique

48 **Chapitre 4 -- ASSIGNATION DE FONCTIONS SYNTHETIQUES**

48 4A. Programmes d'assignement
54 4B. Le "chargeur d'octets du pauvre"
58 4C. Préaffichage des pseudo-XROMs
59 4D. L'assignement de RCL b
61 4E. Sauvegarde et Rappel des alarmes

67 **Chapitre 5 -- COMPRENDRE L'EDITION DE PROGRAMMES SUR LA HP-41**

Créer l'instruction synthétique F0
Déchiffrer les octets en mémoire programme

71 **Chapitre 6 -- STRUCTURE MEMOIRE DE LA HP-41**

71 6A. Structure de la mémoire
79 6B. Première application des registres d'état : suspension des assignements
83 6C. Application No2 des registres d'état : renumérotation des registres

90 **SOLUTIONS AUX PROBLEMES**

94 **APPENDICE A -- TEMPS D'EXECUTION**

Conseils pour accélérer vos programmes
Temps d'exécution des instructions les plus courantes

Comment mesurer les temps d'exécution

98 **APPENDICE B -- CODE MORSE ET STO b**

Faire du morse à 16 mots/minute

102 **APPENDICE C -- REFERENCES**

Revues, livres,etc...

105 **APPENDICE D -- LA CARTE DES CODES PLASTIFIEE**

Description, légende, reproduction grand format

106 **APPENDICE E -- CODES-BARRES DES PROGRAMMES**

107 **Légende de la table des codes**

127 **ADDENDUM :** Petits détails utiles

INTRODUCTION

QU'EST-CE QUE LA PROGRAMMATION SYNTHETIQUE ?

Vous êtes-vous déjà demandé pourquoi la HP-41 ne permet pas d'utiliser plus de 10 TONES ? Ou peut-être vous êtes vous demandé pourquoi vous ne pouviez pas stocker ni rappeler des nombres dans le registre ALPHA. **LA PROGRAMMATION SYNTHETIQUE, C'EST FACILE** | vous apprendra comment passer outre à ces limitations et ajouter un jeu complet de nouvelles fonctions à votre HP-41. Par exemple, vous découvrirez :

-Des techniques permettant de rendre vos programmes plus rapides, plus courts, plus économes en registres de données.

-Trois à six nouveaux registres "fourre-tout" similaires à ceux de la pile pour un usage général.

-21 nouveaux caractères affichables dont les parenthèses, les guillemets, le symbole ¤ et bien d'autres.

-Plus de 100 nouveaux TONES.

-De nouvelles possibilités de traitement du registre ALPHA.

-Suspension et réactivation des affectations de touches en mode USER.

-Combinaison quelconque des 56 flags (utilisateurs et systèmes) modifiable en une seule instruction.

-Renumérotation des registres de données sous contrôle du programme pour éviter les recouvrements dans les sous-programmes.

La création et l'utilisation des instructions synthétiques est appelée **Programmation Synthétique.**

Les instructions synthétiques sont celles qui ne peuvent être entrées au clavier par les moyens habituels. Des centaines d'instructions synthétiques existent. Elles vont des TONES non standards jusqu'aux très puissantes instructions qui affectent les registres d'état du système. La programmation synthétique n'est pas dangereuse pour votre HP-41, mais peut occasionner des "crashs" (clavier inopérant et/ou MEMORY LOST) comme vous le verrez certainement lorsque vous ferez vos premiers pas. La programmation synthétique fonctionne sur tous les calculateurs de la série 41, y compris les 41CV et 41CX, quelle que soit leur date de fabrication. Elle dépend uniquement d'aspects fondamentaux du fonctionnement interne (le système d'exploitation) qui est commun à toutes les 41.

Un exemple simple de la puissance et la beauté de la programmation synthétique est démontré par les deux programmes que vous trouverez à la fin de cette introduction. Celui de gauche est standard (non synthétique), et imprime "Hewlett-Packard". Il occupe 40 octets de mémoire programme (pour plus de détails voir chapitre 1). Le programme de droite utilise une instruction synthétique pour faire la même chose en 20 octets, soit exactement la moitié. Dans cet exemple, qui sera plus détaillé à la section 2E, la programmation synthétique passe outre la difficulté d'accès aux minuscules sur l'imprimante de la 41.

Vous n'aurez pas besoin de devenir un expert pour tirer parti de la programmation synthétique. Après avoir lu ce livre, vous en connaîtrez suffisamment et serez confiant pour créer et utiliser rapidement et sans difficulté n'importe quelle programme synthétique de la librairie des utilisateurs HP, le PPC Calculator Journal[1], ou n'importe quelle autre source. Il sera également discuté des applications les plus courantes de la programmation synthétique, afin que vous puissiez les utiliser dans vos

propres programmes.

Ce livre se veut une introduction facile et appliquée de la programmation synthétique de la 41. Il comporte les techniques simples les plus récentes pour vous permettre facilement d'essayer les exemples au fur et à mesure que vous les lirez.

Le domaine traité dans **LA PROGRAMMATION SYNTHETIQUE C'EST FACILE** a été volontairement limité, afin de ne pas compliquer cette introduction à la programmation synthétique. Les détails ont souvent été omis, mais des références permettront aux lecteurs désireux d'approfondir leurs connaissances d'aller plus loin dans ce domaine. L'utilisateur classique de la programmation synthétique apprendra tout ce dont il aura besoin dans ce livre. Pour les autres, ce livre est une base de départ pour accéder au monde grandissant de la littérature dédiée à la programmation synthétique.

Si vous possédez un PPC ROM[2], votre progression dans la lecture de ce livre peut être accélérée en utilisant ses applications avancées comme les programmes d'assignations ou de chargement d'octets. Si vous avez simplement le calculateur, vous devrez suivre plus précisement le contenu de ce livre pour permettre à votre système d'accéder à toutes les possibilités de la programmation synthétique. De toute façon, cette tâche ne sera pas très difficile.

Hewlett-Packard ne cautionne pas l'utilisation de la programmation synthétique. Quoique certaines personnes chez HP connaissent la programmation synthétique, HP ne répond pas aux questions des utilisateurs concernant la programmation synthétique. Aussi, **ne vous adressez pas à HP** pour vous renseigner sur la programmation synthétique. Lisez ce livre et les autres sources d'information (cf. appendice C) pour trouver la réponse à vos questions.

La chose la plus importante que vous apportera ce livre est l'accès à tous les programmes synthétiques publiés. Beaucoup d'entre eux, en particulier ceux du PPC ROM, résolvent des problèmes qui ne seraient accessibles à aucun programme standard. Après avoir lu ce livre, les programmes synthétiques ne vous sembleront plus mystérieux, ni inaccessibles. Il y a des centaines de programmes synthétiques dans le PPC Calculator Journal ou ailleurs qui donneront à votre 41 des possibilités dont vous n'auriez jamais osé rêver.

---

1. Le PPC Calculator Journal (PPC CJ) est une publication du Personnal Programming Center (PPC), une association californienne à but non lucratif dédiée à la programmation individuelle. PPC a plusieurs milliers de membres, dont la plupart sont des "mordus" de la 41. Les membres de PPC sont à l'origine de toutes les découvertes sur la programmation synthétique, depuis la première description de la programmation synthétique par William C. Wickes dans PPC CJ en 1979. Le PPC Calculator Journal reste la première source d'information concernant les derniers développements de la programmation synthétique. Pour obtenir des renseignements sur l'abonnement à PPC CJ, voir appendice C.

2. Le PPC ROM est un custom ROM (Module enfichable de programmes en mémoire

morte) conçu par les membres de PPC et fabriqué par Hewlett-Packard. Il contient 122 programmes, dont la plupart contiennent des instructions synthétiques. Le manuel est un livre colossal de 492 pages et n'a certainement jamais été lu entièrement par quelqu'un. Voir l'appendice C pour savoir comment obtenir le PPC ROM. Les programmes du PPC ROM sont signalés dans ce livre par un nom en blanc sur noir comme MK.

```
NONSYNTHETIC:

01 "H"                    Hewlett-Packard
02 ACA
03 SF 13
04 "EWLETT-"
05 ACA                      SYNTHETIC:
06 CF 13
07 "P"                    01 "Hewlett-Packard"
08 ACA                    02 AVIEW
09 SF 13                  03 END
10 "ACKARD"
11 ACA
12 PRBUF                                  CAT 1
13 CF 13                  END       40 BYTES
14 END                    END       20 BYTES
```

CHAPITRE UN

CREER VOTRE PREMIERE INSTRUCTION SYNTHETIQUE

Un nombre décimal (base 10) xyz a pour valeur $x.10^2+y.10+z.1$ où x, y, z sont des chiffres de 0 à 9. De même en binaire (base 2) le nombre qrst (exprimé en base 2) a pour valeur $q.2^3+r.2^2+s.2+t$ où q, r, s, t sont des chiffres égaux à 0 ou à 1. q est le chiffre de poids 8, r est le chiffre de poids 4, etc... Par exemple en base 2 : 1011=8+2+1=11 et 11111111 = 128+64+32+16+8+4+2+1=255.

Un nombre hexadécimal (base 16) uv a pour valeur u.16+v où u et v sont des chiffres héxadécimaux de zéro à quinze. Comme il n'existe pas de chiffre pour représenter les nombres dix à quinze (décimal), il est convenu de les emprunter à l'alphabet: A=10, B=11, C=12, D=13, E=14, F=15. Par exemple C5=12.16+5=197 et FF=15.16+15=255. Dans la suite, le diminutif "hex" sera employé pour désigner les nombres hexadécimaux (base 16).

Si vous n'êtes pas habitué aux bases 2 et 16, lisez de nouveau les deux précédents paragraphes et refléchissez-y. Tout comme la suite de ce chapitre, tout doit "couler de source" après deux lectures. Continuons, nous allons commencer à nous amuser dès la fin de ce chapitre.

L'unité de base de la mémoire de la HP-41 est appelée **octet.** Un octet est une suite de 8 bits (nombres binaires, **bi**nary digi**t**s en anglais) et sa valeur peut donc aller de 00000000 base 2 à 11111111 base 2, soit 0 à 255 en base 10. Bien qu'un octet ne puisse prendre que 256 valeurs différentes, il y a des centaines d'instructions différentes sur la 41. Les instructions STO et RCL à elles seules peuvent comporter plus de 400 formes différentes. Cette variété est obtenue en allouant plus d'un octet pour le codage de certaines instructions. Les instructions simples comme +, LOG, et MOD occupent un seul octet en mémoire programme. Les instructions comme VIEW 14, RCL 99, et SIGMA REG IND X nécessitent 2 octets : un pour le nom de la fonction ou préfixe, un second pour l'argument ou suffixe. Quelques instructions nécessitent 3 octets, alors que les chaînes de caractères réclament jusqu'à 16 octets (pour une chaîne de 15 caractères).

Les instructions synthétiques peuvent être obtenues par le masquage de l'octet préfixe des instructions à deux octets, obtenu par une procédure simple décrite dans ce chapitre et le suivant. Comme vous le verrez, le fait d'enlever le préfixe libère l'octet suffixe qui peut ainsi se transformer en préfixe en s'attachant le ou les octets suivants. En choisissant avec soin les instructions de départ, nous pouvons créer une grande quantité d'instructions synthétiques en enlevant le préfixe du départ. Afin d'enlever ces préfixes, nous utilisons un assignement appelé le "Byte Grabber" (dérobeur d'octet), découvert par Erwin Gosteli après de longue séances de recherche avec Jack Baldrige. Soit dit en passant, tous deux sont membres de PPC et leurs découvertes furent relatées dans le PPC Calculator Journal (voir appendice C, paragraphe 1). En fait, toutes les personnes mentionnées dans ce livre pour leurs découvertes ou leurs programmes sont membres de PPC.

Du fait que le Byte Grabber n'est pas un assignement standard, il doit être créé de façon particulière. Vous ne serez certainement pas capable de comprendre pour l'instant le pourquoi de la procédure décrite, suivez simplement les instructions pas à pas et avec le plus grand soin. Vous pourrez remettre votre casquette d'individu pensant lorsque vous aurez assigné le Byte Grabber.

Prenez maintenant votre HP-41 si vous ne l'avez déjà devant vous. Si

vous aviez dans l'idée de lire d'abord ce livre, puis d'essayer les exemples qu'il contient, **oubliez vos résolutions !**

Les exemples sont la partie la plus importante du processus d'apprentissage. Essayer les exemples rendra également le texte plus facile à suivre. Lorsque vous lisez "allez à la ligne 05 et effacez-la", ne vous demandez pas ce qu'est la ligne 05. Essayer les exemples au fur et à mesure peut sembler une perte de temps lors de la lecture, mais ce procédé vous en fera gagner car vous n'aurez pas besoin de lire et de relire.

Si vous avez un PPC ROM, sautez directement au paragraphe 12.

Si vous n'avez pas le PPC ROM, vous pouvez assigner le Byte grabber en suivant très soigneusement une procédure découverte par Keith Kendall. Suivez **très précisement** les instructions qui suivent ou vous n'aurez plus qu'à tout recommencer depuis le début (paragraphe 1). Il se peut qu'il vous faille plusieurs essais pour arriver au résultat, soyez patient.

1. Réinitialisez votre machine en faisant MEMORY LOST. Rappelez vous que ceci est obtenu en maitenant la flèche de correction enfoncée lorsque vous allumez la machine, puis en relâchant cette touche. Il existe une autre technique, plus compliquée, qui permet d'assigner le Byte grabber sans faire de MEMORY LOST, mais vous devez considérer ce paragraphe comme un rite initiatique pour accéder à la programmation synthétique. Ce ne sera certainement pas la dernière fois que vous ferez MEMORY LOST.
2. ASN "+" à la touche LN. (Pressez: SHIFT (touche jaune) ASN ALPHA SHIFT + ALPHA LN). Cet assignement sera remplacé par le Byte grabber.
3. ASN "DEL" à la touche LOG. (Pressez: SHIFT ASN ALPHA D E L ALPHA LOG.)
4. Passez en mode programme. Vous devez voir 00 REG 45. (41C et CV)
5. Faites CAT 1 (toujours en mode programme) et pressez R/S immediatement, avant que l'affichage ne clignote. Répétez cette séquence si vous n'avez pas pressé R/S assez vite.
6. Passez en mode ALPHA, puis pressez la flèche de correction avec le .END. terminal affiché.
7. Vous devez voir la ligne de programme 4094 RCL 01. L'origine de ce numéro de ligne pour le moins étrange est expliquée à la section 6A. Un "bug" (c'est à dire une erreur) dans les microprogrammes internes de la HP-41 vous a permis de sortir des limites autorisées de la mémoire programme. Vous vous trouvez maintenant dans la zone des registres fourre-tout du système. Plus de détails sur ceci seront donnés au chapitre 6. Sortez maintenant du mode ALPHA en pressant de nouveau la touche ALPHA.
8. GTO .005 Vous pouvez presser LN pour introduire 005 pour économiser les pressions de touches. Vous devez voir 05 LBL 03. Vous êtes maintenant dans la zone des assignements, ceci sera également discuté à la section 6A. Il faut maintenant retirer le "+" bouche-trou et le remplacer par l'assignement synthétique du Byte grabber. Comme le calculateur "pense" qu'il se trouve encore en zone programme, on fera cette substitution en introduisant au clavier des lignes de programme qui correspondent aux données nécessaires à l'assignement du byte grabber. Cette correspondance n'est pas évidente, aussi ne cherchez pas à comprendre le sens de ces instructions.
9. DEL 003.Vous pouvez vous éviter de nombreuses pressions de touches

en pressant USER afin d'activer l'assignement de la fonction DEL que vous avez fait sur la touche LOG ; LOG, SQRT (la touche racine carrée). Vous devez voir 04 STO 01. Vous avez maintenant détruit l'assignement de la fonction +. Nous allons le remplacer par le byte grabber.

10. Entrez la chaîne de caractères "?AAAAAA". Si vous n'avez pas de module de fonctions d'extension mémoire connecté (ni une 41-CX), vous allez voir 05 "?A . Les cinq derniers A sont allés par delà la fin de la mémoire, dans la zone correspondant à la première partie de la mémoire étendue, et apparaissent comme des caractères "fantômes".
11. Sortez du mode PRGM et faites GTO.. ou CAT 1 pour sortir des registres d'assignement. Sautez le paragraphe 12 et continuez ensuite la lecture du texte.
12. Si vous avez le PPC ROM, ou si vous avez déjà lu le chapitre 4 et avez une copie du programme MK (Make Key assignments), assignez le byte grabber à l'aide de la procédure suivante plutôt que celle des paragraphes 1 à 11 ci-dessus :

    a.) Détruisez toutes les alarmes stockées par le module TIME qui pourraient être présentes.

    b.) ASN ALPHA ALPHA LN (ceci détruit tout assignement précédemment fait sur la touche LN)

    c.) XEQ MK ou MK

    d.) Lorsque le message PRE↑POST↑KEY apparaît, introduisez 247 ENTER↑ 63 ENTER↑ 15 R/S. Lorsque le programme s'arrêtera de nouveau, ce sera fait. Vous pouvez effacer le message PRE↑POST↑KEY, mais ce n'est pas nécessaire.

Si vous avez suivi à la lettre la procédure décrite plus haut, le byte grabber doit être assigné à la touche LN. Mais ne l'essayez pas tout de suite; le byte grabber peut être dangereux si vous n'êtes pas prudent. Si vous pressez LN en mode USER et **maintenez la touche enfoncée,** vous devez voir XROM 28,63 suivi du message NULL, indiquant que la limite de temps pour relâcher la touche a été dépassée. Lorsque le message NULL apparaît, l'opération déclenchée par le byte grabber est annulée et l'on peut relâcher la touche sans risque. Dans quelques pages, vous utiliserez le byte grabber, aussi ne soyez pas impatient. Quelques informations maintenant vous éviteront de nombreux MEMORY LOST plus tard.

Si vous possédez un lecteur de cartes, enregistrez une carte d'état (XEQ ALPHA W S T S ALPHA) pour conserver l'assignement du byte grabber. De cette façon, si jamais vous obteniez MEMORY LOST par la suite, vous pourrez retrouver votre assignement du byte grabber en lisant la piste 2 de la carte d'état. Il vous suffira alors d'effacer avec la flèche de correction le message vous demandant d'introduire la piste 1 de la carte.

NOTE : Lorsque vous verrez écrit BG (diminutif employé pour Byte Grabber) dans la suite du texte, cela signifie : la touche à laquelle est assigné le byte grabber, dans notre cas LN. Excepté dans les cas mentionnés, la touche du byte grabber doit être pressée en mode USER et en **mode PRGM.**

ATTENTION : Ne pressez pas BG sans précaution en mode PRGM. Si vous la pressez avec le pointeur de programme sur ou juste avant un END, vous pourriez être contraint de faire MEMORY LOST pour retrouver l'usage du catalogue 1. (La première des choses à essayer dans ce cas est de faire BST jusqu'à la ligne qui était affichée lorsque vous avez pressé BG, puis refaire BG.) Si jamais votre clavier devient inopérant, retirez les piles

(ou les batteries), ainsi que l'imprimante si elle est connectée, puis replacez-les au bout de quelques secondes. Si cela ne suffit pas, essayez d'allumer et d'éteindre plusieurs fois de suite la HP-41 sans ses piles. Retirez également tous les modules enfichables (particulièrement les QUAD MEMORY, X-MEMORY et X-FUNCTIONS). Quelques très rares "crash" ("plantages") nécessitent de retirer les piles toute une nuit.

NOTE DE L'EDITEUR : Des "crash durs", entraînant la destruction de l'unité logique de la HP-41 avec retour en service après vente, se produisent parfois. Ils sont dus le plus souvent à des problèmes d'électricité statique. En tous cas, la programmation synthétique ne peut en aucun cas être tenue pour responsable d'un tel accident.

Pasez maintenant en mode PRGM, GTO.. puis entrez ces instructions, que nous utiliserons bientot :

```
01 ENTER↑
02 X<> 88
03 STO IND 31
04 PI
```

La ligne 01 est une instruction ENTER↑ ordinaire. La ligne 02 est obtenue en frappant XEQ, ALPHA, X, SHIFT COS, SHIFT TAN, ALPHA, 8, 8. Comme vous pourrez le lire dans le manuel d'utilisation de la 41, celle-ci contient beaucoup de fonctions qui n'ont pu être implémentées au clavier. Les fonctions telles que X<> qui ne sont pas présentes au clavier doivent être introduites en pressant XEQ, ALPHA, nom de la fonction, ALPHA. Les caractères obtenus en mode ALPHA en pressant SHIFT, tels < et > ne sont malheureusement pas visibles au clavier. Vous devez donc retourner votre HP-41 et regarder l'aide-mémoire collé au dos pour localiser la touche que vous cherchez (sauf HP-41 CX).

Au cas où vous n'auriez pas su entrer les instructions ci-dessus, la ligne 03 s'obtient par STO, SHIFT, 3, 1. La fonction PI peut être obtenue en pressant SHIFT, 0.

Avant d'utiliser le byte grabber, vous devez connaître quelques détails sur les octets. Laissez donc votre calculateur de côté pendant le temps nécessaire à l'assimilation des deux pages qui suivent.

En programmation synthétique, il est souvent pratique d'exprimer les 256 valeurs possibles d'un octet en hexadécimal (base 16). En partageant les 8 bits d'un octet en deux groupes de quatre bits, et en convertissant chaque groupe en un nombre hexadécimal, nous obtenons une représentation d'un octet par deux chiffres. En base 16, les lettres A à F représentent les nombres décimaux 10 à 15. Voici la table de conversion des groupes de 4 bits :

binaire	**hex**	**décimal**
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11

1100	C	12
1101	D	13
1110	E	14
1111	F	15
1 0000	10	16

Par exemple 0100 1101 base 2 = 4D base 16
1111 0001 base 2 = F1 base 16.

Prenez votre **Carte des codes plastifiée** (la carte 7,5*15,5 mm fournie avec ce livre) ou référez-vous à la table de l'appendice D. La table des octets contenue dans la carte des codes est la pierre de rosette de la programmation synthétique, donnant toutes les équivalences nécessaires à la création d'instructions synthétiques.

L'octet est basé sur une représentation rc_{16} où r est le numéro du rang (de 0 à 15) et c est le numéro de la colonne. Les rangs 0 à 7 sont la première partie de la table ; les rangs 8 à F sont la seconde. En haut de chaque case de la table se trouve la fonction primaire, ou préfixe, interprétation de la valeur de cet octet. Immédiatement au-dessous se trouve l'interprétation en tant que suffixe. Au bas de chaque case se trouve la valeur décimale de cet octet. A droite se trouvent les caractères affichés respectivement par l'écran et l'imprimante lorsqu'ils recoivent cet octet ; plus de détails à la section 2E.

Prenons pour exemple l'instruction ENTER↑ que vous avez entré à la ligne 01. Comme nous trouvons ENTER↑ dans la zone préfixe (le haut de la case) à l'intersection de la ligne 8 et de la colonne 3, nous en déduisons que ENTER↑ est codé de façon interne : 83 (hex). Le bas de la case nous indique que l'équivalent décimal de 83 hex est 131. Vous n'avez pour l'instant pas besoin de cet équivalent décimal, mais vous le trouverez bien pratique lorsque nous aborderons le chapitre 3.

Etudions maintenant le X<> 88 de la ligne 02. Nous trouvons X<> à l'intersection de la ligne C et de la colonne E, et 88 dans la partie de la case réservée au suffixe à l'intersection de la ligne 5 et de la colonne 8. Ceci traduit le fait que X<> 88 , une instruction codée sur deux octets, est représentée par CE 58 hex, occupant deux octets consécutifs en mémoire. La ligne 03 contient STO IND 31. STO apparaît ligne 9 colonne 1 et IND 31 ligne 9 colonne F. Ainsi STO IND 31 est composée des deux octets consécutifs 91 9F. La ligne 04, PI, est représentée par 72 hex (ligne 7, colonne 2). Notez que les numéros de lignes ne sont pas stockés en mémoire. La HP-41 calcule les numéros de ligne en comptant les instructions depuis le début de la mémoire programme.

Supposons que nous puissions nous débarrasser de l'octet X<> (l'octet CE hex) dans l'instruction X<> 88. Le suffixe 88 (58 hex) va être livré à lui-même, devenant l'instruction E↑X-1 (voir la zone préfixe de la case 58 hex).

L'assignement du byte grabber nous permet de nous débarrasser facilement du préfixe (l'octet de tête) des instructions. Pour cette raison, on l'appelle parfois "masqueur de préfixe". Le byte grabber opère toujours sur la ligne de programme **suivant** celle qui est affichée, dérobant l'octet de tête.

Reprenez maintenant votre HP-41, allumez-la, et vérifiez que votre programme est toujours intact en passant en mode PRGM et en pressant SST pour le lire pas à pas.

Pour illustrer le comportement de masqueur de préfixe du byte grabber sur l'instruction X<> 88, exécutez tout d'abord PACK (XEQ ALPHA P A C K ALPHA). Ne faites **pas** GTO.. car vous devez rester où vous êtes dans le

programme et GTO.. a la détestable manie de rajouter un END à votre programme et de vous en faire sortir. Assurez-vous que vous êtes en mode USER, puis tapez GTO .001 (pour vous positionner sur l'instruction **précédant** le X<> 88). Passez en mode PRGM si vous n'y êtes pas déjà, puis BG (pressez la touche LN). Vous devez voir apparaître une instruction pour le moins étrange : 02 ᵀ ? ⊠

La Nova (nom donné aux 14 segments allumés) à la fin de la chaîne de caractères est, ou plutôt était, la partie X<> de X<> 88. L'octet CE hex a été dérobé, laissant le suffixe devenir une instruction à part entière. SST et vous voyez :

03 E↑X-1

tel que nous l'avions prévu.

Relisez cet exemple jusqu'à ce que vous soyez à l'aise. Lorsque vous aurez compris la structure de la mémoire (par octets) et le rôle du byte grabber (voir figure 1.1), vous pourrez alors envisager sereinement la programmation synthétique.

Qu'adviendrait-il si nous dérobions le préfixe STO de l'instruction STO IND 31 ? D'après la table, le suffixe IND 31 deviendrait une instruction TONE. Mais attendez un peu. L'instruction TONE nécessite la présence d'un suffixe ; après tout, chaque TONE est une instruction à deux octets. Où donc ce TONE nouvellement créé va-t-il aller chercher son suffixe ? Voyons cela. BG sur la ligne 003 (GTO .003 si vous n'y êtes pas déjà et pressez LN en mode PRGM) pour dérober l'octet codant STO. SST et voyez : 05 TONE Y, une instruction synthétique ! Un rapide coup d'oeil sur la table à l'intersection de la ligne 7 et de la colonne 2 revèle que le TONE a absorbé l'instruction PI pour la transformer en suffixe Y (voir figure 1.1). Il est raisonnable de penser que le TONE a pris son suffixe dans l'instruction qui le suivait dans le programme -- il fallait bien qu'il le prenne quelque part.

Vous pouvez faire SST sur la ligne 05 en mode RUN (c'est-à-dire hors du mode PRGM) pour écouter votre nouveau TONE synthétique. BST et SST pour le réécouter si cela vous chante. Il existe plus de 100 autres TONEs synthétiques qui ne demandent qu'à ce que vous les expérimentiez.

valeur hex de l'octet	instructions programme	instructions après BG
8 3	ENTER↑	ENTER↑
C E	X<>	ᵀ ? ⊠
5 8	88	E↑X-1
9 1	STO	ᵀ ? ⊠
9 F	IND 31	TONE
7 2	PI	Y

Figure 1.1 **Transformation des instructions par le byte grabber.**

CHAPITRE DEUX

LES INSTRUCTIONS SYNTHETIQUES LES PLUS COURANTES

Ce chapitre présente les huit types d'instructions synthétiques les plus utilisées. Que vous soyez ou non appelés à écrire des programmes synthétiques "exotiques", vous serez amenés à utiliser ces instructions faciles d'accès dans votre programmation de tous les jours. Les types d'instructions décrites dans ce chapitre sont :

A. Les TONEs synthétiques qui personnalisent vos programmes.

B. Les exposants synthétiques (forme abrégée) qui prennent moins de place en mémoire.

C. Le contrôle des drapeaux, utilisé pour préserver l'état de l'affichage lors de la construction des messages.

D. Contrôle du pointeur de programme, qui permet de figer le "canard".

E. Les chaînes de caractères synthétiques, qui permettent l'utilisation des parenthèses ou des minuscules.

F. L'instruction TEXT 0, équivalente à l'instruction NOP (pas d'opération) de la HP-25.

G. Utilisation du registre ALPHA comme registre de stockage temporaire ne perturbant pas les registres de données.

H. Utilisation de certains registres scratch du système comme registres de stockage temporaire.

Des exemples d'instructions synthétiques sont présentés dans ce chapitre, ainsi qu'un suivi pas à pas de la procédure pour les obtenir. Ces procédures utilisent l'assignement du byte grabber fait au premier chapitre. Les possesseurs du PPC ROM peuvent se passer de ces procédures en créant directement les instructions avec le programmes LB (Load Bytes, "chargeur d'octets") du ROM. Les données à fournir au programme LB seront fournies pour chaque exemple. Si l'instruction synthétique comprend deux octets et n'est pas une entrée numérique, la routine (programme) MK du PPC ROM peut être employée à la place de LB pour obtenir une assignation de la fonction désirée. Il est néanmoins recommandé aux possesseurs du PPC ROM d'essayer d'entrer certains exemples de ce livre avec le byte grabber plutôt qu'avec l'aide de LB et MK.

Pour ceux qui n'ont pas le PPC ROM, une version simplifiée de LB sera décrite au chapitre 3, avec la description des instructions nécessaires à l'utilisation du byte grabber pour charger le programme. Vous pouvez le taper tout de suite, mais vous en apprendrez plus sur l'utilisation du byte grabber si vous attendez d'être arrivé au chapitre 3 pour taper et utiliser LB.

2A. TONEs synthétiques

Ainsi qu'il est écrit à la fin du premier chapitre, il existe plus de 100 TONEs synthétiques, très variables en durée et hauteur. Parmi les 16 fréquences possibles, les dix premières sont les fréquences des TONE 0 à TONE 9. La durée des TONEs synthétiques va de quelques millisecondes (ces TONEs ressemblent à des "clics") jusqu'à plusieurs secondes. Pour beaucoup d'applications demandant de jeter un oeil sur l'afficheur, il est préferable d'utiliser un son aigu et de courte durée. TONE 89 en est un bon exemple. On peut le créer comme suit : détruisez les lignes restantes utilisées au premier chapitre et tapez celles-ci :

```
01 ENTER↑                    Valeurs à entrer pour LB/MK
```

```
02 STO IND 31            TONE 89 = 159,89
03 SIN
```

Maintenant, toujours en mode PRGM, GTO .001 et BG (pressez LN en mode USER). Comme toujours, vous voyez apparaître une chaîne de caractères du genre : 02 ᵀ ? Ø. SST pour voir votre nouvelle instruction synthétique 03 TONE 9. Elle peut vous paraître non synthétique mais vous entendrez bientôt la preuve qu'elle l'est.

L'octet IND 31 (9F hex) est devenu une instruction synthétique après que l'octet de STO ait été dérobé. L'octet SIN (ligne 5 colonne 9 de la table c'est-à-dire 89 décimal) est devenu le paramètre du TONE. Les TONEs synthétiques de 10 à 101 apparaissent à l'affichage seulement comme TONE suivi du chiffre de droite du paramètre décimal. Dans notre exemple, TONE 89 apparaît comme TONE 9. Les autres TONEs apparaissent avec pour suffixe une lettre tel le TONE Y que nous avions obtenu au premier chapitre.

Passez en mode RUN et SST pour écouter le TONE 89. Il pourrait bien devenir l'un de vos préférés.

La table 2.1 rassemble tous les TONEs synthétiques que vous pouvez obtenir. La fréquence du TONE est déterminée par sa colonne dans la table. Les fréquences correspondant aux colonnes A à F étendent vers le bas les possibilités standard en ce domaine, la plus haute fréquence synthétique, colonne F, se trouvant juste en-dessous de celle de TONE 0, la fréquence standard la plus basse.

La durée de chaque TONE, exprimée en secondes, est donnée dans la table. Cette durée est celle que met la HP-41 pour exécuter le TONE ; ainsi la durée audible pourra être beaucoup plus courte pour les TONEs très brefs. Les durées peuvent légèrement varier d'une machine à l'autre, suivant leur date de production. Par exemple, TONE Z dure 0.64 seconde sur les HP-41 récentes alors qu'il ne durait que 0.061 seconde sur les anciennes.

Tel que vous le verrez en examinant la table, les TONEs 37 et 38 sont les plus courts, 0.020 seconde chacun. L'exemple suivant montre un de leurs emplois possibles. Effacez l'exemple précédent et entrez les lignes suivantes :

```
01 DEG                   Valeurs pour LB/MK
02 CLX
03 LBL 01
04 STO IND 31            TONE 37 = 159,37
05 RCL 05
06 SIN
07 SQRT
08 STO IND 31            TONE 38 = 159,38
09 RCL 06
10 SIN
11 SQRT
12 GTO 01
```

GTO .007, BG, et détruisez la chaîne de caractères. SST pour voir TONE 8 (en fait TONE 38). GTO .003, BG, et effacez la chaîne de caractères. SST vous devez voir TONE 7 (en fait TONE 37). Sortez du mode PRGM, RTN, puis R/S. Quoique l'horloge interne de la HP-41 ne soit pas à quartz, ce programme imite assez bien le tic-tac d'une pendule.

Les TONEs synthétiques ont d'autres applications. Voir l'appendice B pour créer un programme qui génère rapidement du code Morse en utilisant

des TONEs synthétiques. Vous pouvez utiliser la table 2.1 pour vous aider à choisir les TONEs adaptés à vos besoins. Vous pouvez choisir une fréquence et une durée et chercher le TONE qui a les caractéritiques les plus proches. La table 2.1 et la figure 2.1 ont étés reproduites avec l'autorisation de Robert E. Swanson, qui les a écrites pour l'ouvrage HP-41/ HP-IL SYSTEM DICTIONNARY.

2.B Exposants synthétiques

Si vous pressez EEX CHS 3 en mode RUN, vous obtenez 1. 10^{-3} dans le registre X. Mais si vous faites de même en mode PRGM, vous obtenez l'instruction 1E-3 même si vous n'aviez pressé que E-3. Le calculateur persiste à introduire le 1 superflu, perdant un octet d'encombrement mémoire. Maintenant que nous avons le byte grabber, je parie que vous savez déjà comment se débarrasser de ce 1. Effacez l'exemple précédent et frappez :

```
01 ENTER↑                    Valeurs pour LB
02 1E-3                      E-3 = 27, 28, 19
```

PACK (ceci est nécessaire). Comme dans le premier chapitre, vous devez presser XEQ ALPHA P A C K ALPHA, et **pas** GTO.., qui serait plus facile à frapper mais déplacerait le pointeur, ce qui vous obligerait à revenir dans votre programme. Vous pouvez gagner un peu de temps en assignant PACK à une touche ; tapez ASN ALPHA P A C K ALPHA et pressez une touche qui ne soit pas déjà assignée et que vous voulez conserver.

Faites maintenant GTO .001 et BG. Détruisez la chaîne de caractères. La nova à la fin de la chaîne est le 1 dérobé. SST et voyez 02 E-3 , un exposant synthétique souvent appelé exposant sous forme réduite.

Vous pouvez expérimenter cette instruction en faisant SST en mode RUN. Vous découvrirez que E-3 fonctionne comme 1E-3. Il sauve bien évidemment un octet de mémoire, mais vous devez savoir qu'il s'éxecute également plus rapidement que 1E-3.

On peut également gagner en vitesse d'exécution, mais pas en place mémoire, en utilisant le point décimal à la place du chiffre 0 pour une entrée nulle, et E en lieu et place de 1 pour une entrée unitaire. Le point décimal seul n'est pas une instruction synthétique mais le E seul en est une. Pour l'obtenir, dérobez le préfixe STO d'une instruction STO 27. La case ligne 1 colonne B de la table des codes montre que le suffixe devient une instruction EEX.

Il est écrit plus haut qu'il est nécessaire de PACKer (compacter) la mémoire pour enlever le 1 de tête d'une instruction avec exposant. La raison en est que les entrées de données numériques dans les programmes sont précédées par un octet nul invisible (ligne 0 colonne 0) qui sert seulement à séparer le premier chiffre de l'instruction précédente. Ne confondez pas l'octet nul avec le message NULL (annulé) qui apparaît lorsque vous maintenez une touche enfoncée pendant 2 secondes après que le nom de la fonction apparaisse à l'écran. Comme son nom l'indique, un octet nul est un marqueur qui ne fait rien à l'éxécution (sauf s'il est suffixe dans une instruction comme X<> 00 ou ΣREG 00). Les octets nuls, qui sont toujours invisibles sauf s'il sont inclus dans des chaînes de caractères, sont crées lorsque des instructions sont effacées et sont enlevés en exécutant PACK. Ce comportement sera décrit et expliqué au chapitre 5.

Dans le premier exemple de cette section, nous avons utilisé PACK pour enlever le nul que la HP-41 avait inséré entre 01 ENTER↑ et 02 1E-3. Si la ligne 01 avait été une donnée numérique, le nul n'aurait pu être enlevé par

Table des TONE HP-41C/CV : durées d'exécution et numéros XROM

Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.27	0.27	2.08	2.42	3.37	0.67	2.30	0.35
Ø													0.80		
60,00	60,01	60,02	60,03	60,04	60,05	60,06	60,07	60,08	60,09	60,10	60,11	60,12	60,13	60,14	60,15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1.82	0.32	1.43	0.29	0.48	0.94	0.45	0.82	0.29	0.49	4.70	3.23	1.75	3.85	3.46	2.37
1															
60,16	60,17	60,18	60,19	60,20	60,21	60,22	60,23	60,24	60,25	60,26	60,27	60,28	60,29	60,30	60,31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
.022	1.10	2.25	1.90	1.17	.020	.020	0.35	0.65	0.49	0.83	0.43	3.80	1.71	1.29	0.12
2															
60,32	60,33	60,34	60,35	60,36	60,37	60,38	60,39	60,40	60,41	60,42	60,43	60,44	60,45	60,46	60,47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
0.50	0.26	2.04	1.85	0.29	0.14	0.75	0.77	0.62	.046	4.07	3.99	3.19	3.77	0.93	0.27
3											0.41		0.39		
60,48	60,49	60,50	60,51	60,52	60,53	60,54	60,55	60,56	60,57	60,58	60,59	60,60	60,61	60,62	60,63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
1.79	2.29	0.16	0.19	1.01	0.25	.072	0.21	0.13	0.15	3.58	0.28	3.60	3.30	0.85	0.87
4		0.40				.032									
61,00	61,01	61,02	61,03	61,04	61,05	61,06	61,07	61,08	61,09	61,10	61,11	61,12	61,13	61,14	61,15
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
.075	0.22	1.68	0.72	0.30	1.16	0.46	.093	0.56	.038	2.61	0.39	3.12	3.78	0.30	2.45
5															
61,16	61,17	61,18	61,19	61,20	61,21	61,22	61,23	61,24	61,25	61,26	61,27	61,28	61,29	61,30	61,31
96	97	98	99	100	101	102 A	103 B	104 C	105 D	106 E	107 F	108 G	109 H	110 I	111 J
0.62	2.21	0.41	1.21	0.11	1.27	0.96	0.80	0.64	0.45	2.26	0.43	3.54	0.31	2.00	2.33
6										0.23					*0.33*
61,32	61,33	61,34	61,35	61,36	61,37	61,38	61,39	61,40	61,41	61,42	61,43	61,44	61,45	61,46	61,47
112 T	113 Z	114 Y	115 X	116 L	117 M	118 N	119 O	120 P	121 Q	122 R	123 a	124 b	125 c	126 d	127 e
0.25	.061	0.55	1.19	0.40	1.07	0.22 \	0.78]	0.13 +	0.32 _	0.29 "	4.38	0.73	3.77	3.45	2.84
7 *1.66*	*0.64*	*1.40*	*0.48*	*1.20*											
61,48	61,49	61,50	61,51	61,52	61,53	61,54	61,55	61,56	61,57	61,58	61,59	61,60	61,61	61,62	61,63

Dans chaque case se trouve le numéro décimal du TONE, la durée d'exécution et le numéro XROM. Sur les modèles récents, la durée est inférieure d'environ ,015 secondes et dépend de la date de fabrication.


HEX Col.
A B C D E F Ø 1 2 3 4 5 6 7 8 9
Rel. freq.
-6 -5 -4 -3 -2 -1 Ø 1 2 3 4 5 6 7 8 9
Freq. (Hz)
105 113 121 131 143 158 175 197 225 263 315 394 525 629 788 1051
TONE EXECUTION TIME (sec)
5.0
2.0
1.0
0.5
0.2
0.1
.05
.02
R. Swanson
A-flat A B C D-flat E-flat F G A C E-flat G C E-flat G C

PACK. Il aurait été nécessaire pour délimiter les lignes 01 et 02. Excepté dans ce cas particulier, PACK enlève toujours les nuls.

Mais il existe une autre façon de retirer ce nul. On peut tout simplement entrer une instruction codée sur un octet pour remplir l'espace laissé ouvert par le nul. Essayons ceci sur l'exemple de E-3. Effacez la ligne 02 et frappez :

```
01 ENTER↑
02 1E-3
```

Il y a maintenant un nul invisible entre les lignes 01 et 02. Comme nous voulons dérober le 1 de 1E-3, et non pas le nul, nous remplissons d'abord le nul. GTO .001, ou simplement BST, et frappez RDN (roll down : permutation circulaire vers le bas de la pile). C'est une instruction codée sur un octet qui remplaçera l'octet nul. Maintenant BG pour capturer le 1 de tête. Appuyez deux fois sur la flèche de correction et vous obtenez :

```
01 ENTER ↑
02 E-3
```

Les deux pressions de touches introduites ici suppriment la nécessité de PACKer. Ceci peut être particulièrement avantageux si vous devez introduire des exposants synthétiques dans un long programme qui demande plusieurs secondes pour être compacté.

Note de l'éditeur : Une méthode plus rapide consiste simplement à BG deux fois, La première dérobe le nul, la deuxième dérobe le 1.

Le chapitre 5 reprendra dans le détail le comportement évanescent des nuls. Il décrit une procédure synthétique pour les rendre visibles. Les programmeurs synthétiques ambitieux qui veulent parvenir à réaliser des choses complexes comme créer une instruction -E doivent savoir que lorsqu'on veut introduire un signe négatif dans une donnée numérique, l'octet approprié est ligne 1 colonne C (NEG), et **non** ligne 5 colonne 4 (CHS). La touche CHS opère de deux manières : rendre négative une entrée numérique et rendre négatif un nombre existant en mémoire.

2C. Contrôle du registre des drapeaux

Généralement, lorsqu'un programme construit un message en alpha contenant des nombres, le mode d'affichage est altéré. Par exemple, la séquence suivante demande des entrées numérotées 1 à 10 et les stocke dans les registres 1 à 10. Elle a l'effet indésirable de modifier le format d'affichage en FIX 0 (lignes 03 et 04). La programmation synthétique permet d'éviter facilement cette altération de l'affichage dans des cas comme celui-ci :

```
01 1.01          Index de contrôle des registres 1 à 10
02 STO 00
03 FIX 0
04 CF 29         Ces deux derniers pas sont nécessaires
05 LBL 01        pour que le numéro du registre apparaisse
06 "INPUT "      à l'affichage sans le point décimal.
07 ARCL 00       (Remarquer l'espace suivant le T)
08 "┣?"          Ajoute le numéro du registre
09 TONE 9
10 PROMPT
11 STO IND 00    Stocke l'entrée dans le registre courant
12 ISG 00        Ajoute 1 à l'index de contrôle
```

```
13 GTO 01
```

La ligne 06 est XEQ ALPHA T O N E ALPHA 9. La ligne 07 est obtenue par ALPHA shift RCL O O, tandis que la ligne 08 est ALPHA shift XEQ 3 ALPHA.

Il est temps de faire une légère digression à propos des drapeaux. Un drapeau ne pouvant prendre que deux états, levé ou baissé, il est logique que le calculateur utilise un bit (**b**inary dig**it**) pour représenter un drapeau. Dans la pratique, l'état levé est représenté par 1 et l'état baissé par 0. Nous avons vu au premier chapitre qu'un octet est constitué de 8 bits. Le manuel d'utilisation de la HP-41 révèle qu'un registre est constitué de sept octets. Il y a donc 8*7 = 56 bits dans un registre. Si le nombre 56 vous paraît familier, peut-être est-ce parce que la HP-41 possède 56 drapeaux (utilisateur et systême), numérotés de 0 à 55. Il n'est donc pas étonnant que ces 56 drapeaux occupent exactement un registre de la HP-41.

Le registre des drapeaux est l'un des seize registres d'état de la HP-41. Vous connaissez déjà les cinq premiers : les registres T, Z, Y, X, et L qui composent la pile opérationnelle. Les noms des autres se trouvent dans la ligne 7 de la table des codes. Le nom du registre des drapeaux est **d** (ligne 7 colonne E).

Revenons maintenant à nos moutons. Nous voulions conserver le format d'affichage pendant la construction d'un message contenant des nombres. Pour ce faire, nous pouvons faire RCL d avant de construire le message, sauvant ainsi le contenu original du registre d dans X. Après avoir constitué le message, nous ferons STO d, transférant alors le contenu original de d sauvegardé en X dans le registre des drapeaux. Ceci restaurera les 56 drapeaux du départ, y compris ceux codant le format d'affichage.

Pour l'exemple donné plus haut, ceci se fait de la façon suivante Tapez :

```
01 1.01              Valeurs pour LB/MK
02 STO 00
03 LBL 01
04 "INPUT "
05 STO IND 16        RCL d = 144,126
06 AVIEW
07 FIX 0
08 CF 29
09 ARCL 00
10 STO IND 17        STO d = 145,126
11 AVIEW
12 "├?"
13 TONE 9
14 PROMPT
15 STO IND 00
16 ISG 00
17 GTO 01
```

GTO .009, BG, et effacez la chaîne de caractères. SST et voyez STO d. GTO .004, BG, flèche de correction, et SST pour voir RCL d. L'octet IND 17 (ligne 9 colonne 1) devient STO, L'octet IND 16 (ligne 9 colonne 0) devient RCL et les deux instructions AVIEW (ligne 7 colonne E) se transforment en suffixes d. Cette version du programme réclame des entrées à stocker dans

```
01 1.01
02 STO 00

03♦LBL 01
04 "INPUT "
05 RCL d
06 FIX 0
07 CF 29
08 ARCL 00
09 STO d
10 "⊢?"
11 TONE 9
12 PROMPT
13 STO IND 00
14 ISG 00
15 GTO 01
```

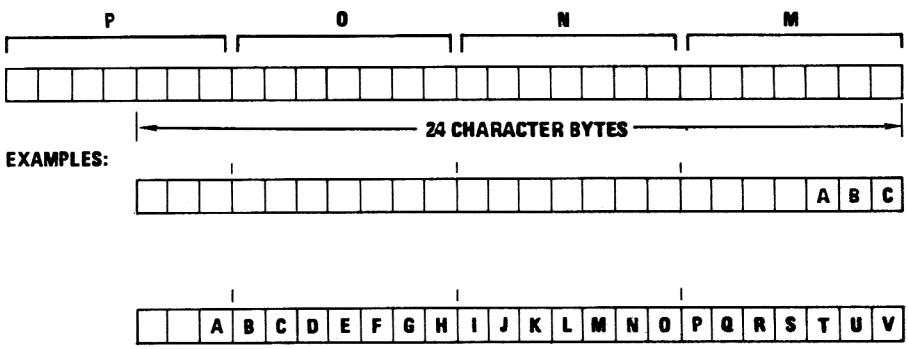


Figure 2.2 : Le Registre ALPHA. Les chaines de caractères de 1 à 24 caractères de long sont toujours justifiées à droite. Les positions non utilisees à gauche sont des nuls invisibles.

les registres 1 à 10. Lorsqu'il est terminé, le format d'affichage est resté inchangé, plutôt que d'avoir ce FIX 0 peu sympathique.

La combinaison RCL d / STO d peut être utilisée n'importe où lorsque vous voulez préserver le format d'affichage, le mode angulaire, ou d'autres drapeaux. Le contenu original du registre des drapeaux peut être stocké n'importe où dans la pile opérationnelle, mais ne doit pas être stocké dans un registre de données. Les données rappelées d'un registre de données subissent en effet une **normalisation.** Si la configuration des 56 bits n'est pas reconnue par la HP-41 comme étant une forme numérique ou alphanumérique, elle change les bits nécessaires pour la transformer en valeur numérique ou alphanumérique.

Le principe détaillé de la reconnaissance par la machine des bits adéquats sort du cadre de cet ouvrage, mais pour notre usage une règle simplifiée de la normalisation suffira : Une combinaison quelconque de 56 bits dont les quatre premiers sont 0001 peut être stockée dans un registre de données et rappelée sans crainte ; si les quatre premiers bits sont différents de 0001, la donnée est sujette à normalisation (c'est-à-dire à une altération possible) lorsqu'elle sera rappelée. Ceci n'est évidemment pas gênant si la donnée est numérique ou alphanumérique. La normalisation n'est gênante que lorsqu'on travaille sur des données non standard tel le contenu du registre des drapeaux.

Si vous souhaitez stocker un jeu de drapeaux dans un registre de données, vous devez avant tout mettre à 0001 les quatre premiers bits. Ceci peut être fait facilement comme dans l'exemple qui suit. Effacez l'exemple précédent à l'exclusion des instructions RCL d et STO d. Puis GTO .000 et tapez :

```
01 CF 00          Ces quatre premières lignes mettent
02 CF 01          à 0001 les quatre premiers bits
03 CF 02          du registre des drapeaux.
04 SF 03
05 RCL d
06 STO 01
07 GRAD
08 SF 01
09 CF 03
10 STOP
11 RCL 01
12 STO d
```

Sortez du mode PRGM, RTN et R/S. Notez que le drapeau 1 est levé et que le mode GRAD est actif. R/S de nouveau pour voir les drapeaux reprendre leurs états initiaux, avec les drapeaux 0, 1, 2 baissés et le drapeau 3 levé. Si vous ne craignez pas d'essayer un exemple qui laissera vos drapeaux en désordre, vous pouvez changer la ligne 01 par SF 00 et vérifier que beaucoup de drapeaux changent d'état lors de l'exécution du programme. Pour remettre les choses dans l'ordre plus rapidement, vous pouvez utiliser une copie du contenu original de d qui se trouve dans Y après l'exécution du programme. Comme cette copie n'a pas été stockée dans un registre de données, elle n'a pas été altérée. Tapez juste RDN, GTO .012 et SST pour restaurer l'état des drapeaux.

2D. Contrôle du pointeur de programme

La HP-41 maintient la position du pointeur de programme dans un de ses registres d'état. Ce pointeur désigne quelle ligne sera affichée en mode

PRGM. Le registre d'état contenant la position du pointeur de programme (ainsi que certaines adresses de retour -ceci est décrit à la section 6A de ce livre ainsi que dans le manuel d'utilisation du PPC ROM à la rubrique "Line by Line Analysis of LR") est appelé registre **b** du système d'exploitation de la HP-41.

Pour illustrer la simplicité du contrôle du pointeur programme sur la HP-41, essayez l'exemple suivant. Effacez l'exemple précédent et tapez :

```
01 ENTER↑              Valeurs pour LB/MK
02 STO IND 16          RCL b = 144,124
03 MEAN
04 STO IND 31          TONE 89 = 159,89
05 SIN
06 STO IND 17          STO b = 145,124
07 MEAN
```

GTO .005, BG, flèche de correction, GTO .003, BG, flèche de correction, GTO .001, BG, flèche de correction deux fois, et PACK (**pas** GTO..). Passez en mode RUN, RTN, et R/S. Vous entendrez le rapide crépitement du TONE 89 répété. Le "canard volant" est figé sur place.

Comment cela fonctionne-t-il ? L'instruction RCL b copie le pointeur programme dans le registre X. TONE 89 est exécuté, puis STO b remet le pointeur programme à la valeur stockée en X (celle qu'il occupait précédemment). Lorsque le pointeur fut rappelé en X, l'instruction suivante à exécuter était TONE 89. C'est pour cela que STO b transfère l'exécution sur l'instruction TONE 89. Si vous faites RTN et SST plusieurs fois, vous vérifierez qu'il se déroule de la façon suivante : RCL b, TONE 89, STO b, TONE 89, etc...

La raison pour laquelle le canard se tient immobile pendant l'exécution est simple. Le canard se déplace d'une position vers la droite chaque fois que le programme passe sur un LBL. Mais il n'y a pas de label dans ce programme, seulement une boucle. Aussi le canard n'a aucune raison de bouger.

L'exemple suivant répond à la question banale : Quelle est la boucle infinie la plus rapide sur la HP-41 ? La réponse tient en deux lignes. Effacez le TONE 89 de l'exemple précédent, puis PACK. Vous avez maintenant :

```
01 RCL b
02 STO b
```

Si vous tapez RTN, puis SST plusieurs fois, vous voyez l'enchaînement RCL b, STO b, STO b, STO b, STO b, etc ... à l'infini bien que le numéro de ligne n'arrête pas de croître. Pendant une exécution pas à pas, la HP-41 incrémente toujours le numéro de ligne à moins que le programme exécute un GTO, XEQ, RTN ou un END, auxquels cas le numéro de ligne est recalculé. Le calculateur ne reconnaît pas STO b comme une instruction de branchement, aussi il ne s'embête pas à recalculer le numéro de ligne. Si votre doigt sur SST fut très persévérant, vous découvrirez que le numéro de ligne est allé jusqu'à 4094 avant de retomber à 02. Comme vous l'apprendrez à la section 6A, le nombre 4095 a une signification particulière pour le programme interne de la HP-41. Ce nombre signifie que le numéro de ligne doit être recalculé.

Pour l'exécution normale (et non pas à pas avec SST), le calculateur ne remet pas à jour le numéro de ligne à chaque instruction. Ceci ralentirait en effet de façon significative la vitesse d'exécution.

Des techniques avancées de programmation synthétique sont nécessaires pour un emploi efficace de toute la puissance de l'instruction STO b. Le programme de code Morse ultra-rapide de l'appendice B utilise le branchement indirect précompilé, une application directe du contrôle de pointeur. De même, la séquence : 0, STO b, GTO .002 est une façon simple de positionner le pointeur dans les registres d'assignement. Des précisions sur la façon dont les informations sont stockées dans les registres d'assignement pourront être trouvées dans le manuel d'utilisation du PPC ROM à la rubrique "Background for MK".

2E. Chaînes de caractères synthétiques

La différence principale entre une HP-41 et ses prédécesseurs est qu'elle permet l'accès aux chaînes alphanumériques. Cette possibilité permet de commenter les demandes de données et les sorties. Néanmoins le jeu de caractères disponible peut paraître limité. Ainsi, il n'existe ni parenthèses ni guillemets.

La programmation synthétique permet d'obtenir 21 nouveaux caractères, utilisables dans les lignes de texte, tels les parenthèses, les guillemets, l'apostrophe, le symbole &, et bien d'autres. Ces caractères synthétiques peuvent être inclus dans des chaînes de caractères par un procédé que nous décrirons plus loin. Les programmes du PPC ROM permettent d'utiliser deux alternatives à cette méthode. La plus simple est d'utiliser LB pour créer directement des chaînes synthétiques. La méthode de transfert par Q est également utilisable à l'aide d'un programme comme DC. La première de ces méthodes sera décrite au chapitre trois. La seconde sera présentée à la section 4B.

La méthode permettant de créer des chaînes de caractères synthétiques à l'aide du Byte Grabber, décrite dans cette section, est très simple et demande très peu de préparation (juste l'assignement du Byte Grabber). Malgré la disponibilité d'autres méthodes, suivez cet exemple utilisant le Byte Grabber. Vous trouverez certainement qu'il est le plus pratique pour créer une ou deux chaînes synthétiques.

Les possesseurs d'imprimante ou du module d'extension de fonctions (X-Functions) peuvent être habitués (à travers les fonctions BLDSPEC et XTOA) à employer des méthodes plus lourdes pour créer de nouveaux caractères. Dans cette section, nous démontrerons que les lignes de caractères synthétiques économisent beaucoup d'octets par rapport aux méthodes classiques utilisant BLDSPEC ou XTOA.

La structure d'une chaîne de n caractères est relativement simple. Un octet Fn (hex) ligne F colonne n, précède n octets, chacun représentant un caractère. C'est pourquoi il faut n+1 octets pour contenir une chaîne de n caractères. La correspondance octet-caractère est donnée dans la table des codes, incluse dans la **carte des codes plastifiée.** Par exemple, l'octet ligne 5 colonne F s'affiche et s'imprime _. Certains caractères apparaissent de façon fort différente à l'affichage et à l'impression. Par exemple, l'octet ligne 0 colonne 4 s'affiche ↑ mais à l'impression donne ∝ . Un octet est interprété comme un caractère seulement s'il est précédé par un octet de la ligne F qui fait passer l'octet en question pour un caractère. En l'absence d'un octet de la ligne F, les octets sont interprétés de la façon habituelle, comme des instructions ou des suffixes de l'instruction précédente. Les octets de la ligne F peuvent être considérés comme des instructions (appelées TEXT) qui nécessitent des octets suffixes. La différence entre les instructions TEXT et la plupart des autres instructions est que le nombre d'octets suffixes est variable et qu'une instruction TEXT donne une interprétation très différente de ces octets en

les considérant comme des caractères.

Les chaînes de caractères synthétiques peuvent être créées en utilisant le Byte Grabber, en 4 opérations. On créera tout d'abord une chaîne de caractères contenant le même nombre de caractères que celle désirée en remplaçant les caractères synthétiques à créer par des X. Puis l'instruction préfixe TEXT sera dérobée. Ceci libèrera les octets suffixes qui seront interprétés comme des instructions. Sous cette forme, les X pourront être remplacés par les instructions correspondant aux caractères synthétiques à créer. On relâchera finalement le préfixe TEXT dérobé, qui capturera les instructions qui le suivent, les transformant de nouveau en caractères.

Un exemple éclaircira cette exposé. Supposez que nous voulions créer la chaîne "HP'S #1". Effacez l'exemple précédent et tapez :

```
01 ENTER↑          Entrées pour LB
02 "HPXS X1"       247, 72, 80, 39, 83, 32, 35, 49.
```

GTO .001 et BG mais **ne pressez pas la touche de correction.** La chaîne affichée contient le préfixe TEXT 7 que nous utiliserons plus tard. SST plusieurs fois et vous voyez maintenant :

```
01 ENTER↑
02 " ?    Ø"
03 Σ-
04 LN
05 E↑X-1
06 Y↑X
07 RCL 00
08 E↑X-1
09 STO 01
```

Les lignes 03 à 09 correspondent chacune à un caractère de la chaîne de départ. Ainsi, RCL 00 correspond à l'espace. La ligne 2 colonne 0 de la table des codes confirme cette correspondance. Nous devons maintenant remplacer les instructions E↑X-1 qui correspondent aux X. Faisons GTO .008 et effaçons le E↑X-1. Nous voulons mettre un # à la place. En consultant la ligne 2 colonne 3 de la table, nous découvrons que l'instruction correspondante est RCL 03. Entrons RCL 03 à la place de la ligne 08. Faisons maintenant GTO .005 et effaçons le E↑X-1. La table nous dit d'utiliser RCL 07, ligne 2 colonne 7, à la ligne 05 du programme pour obtenir l'apostrophe.

Si vous avez suivi ces instructions à la lettre, vous n'avez pas besoin de PACKer, néanmoins cela ne peut pas nuire. Vous devez obtenir maintenant :

```
01 ENTER↑
02 " ?    Ø"
03 Σ-
04 LN
05 RCL 07
06 Y↑X
07 RCL 00
08 RCL 03
09 STO 01
```

Faites maintenant GTO .001 puis BG. Vous avez dérobé le préfixe TEXT

de la ligne 02. Ceci libère le point d'interrogation et la nova qui deviennent des instructions. SST et vous verrez le point d'interrogation se changer en STO 15 (voir ligne 3 colonne F). SST de nouveau pour s'apercevoir que la nova a repris son identité de TEXT 7 et ce faisant, a capturé les 7 octets qui la suivaient, les transformant en caractères. Nous avons donc maintenant :

```
01 ENTER↑
02 " ?    ⊠"
03 STO 15
04 "HP'S #1"
```

Si vous avez une imprimante vous pouvez comparer comment ces caractères s'impriment et comment ils s'affichent. Si vous n'en possédez pas, consultez la partie en bas à droite de chaque case de la table des codes pour savoir comment l'octet apparaît à l'impression. Vous découvrirez que l'apostrophe et le symbole # s'impriment comme prévu, mais que la nova ne laisse aucune trace. Ce comportement évanescent caractérise tous les caractères des lignes 8 à F lors de l'impression. Ce point sera détaillé plus loin dans la suite de cette section.

L'instruction APPEND est tout à fait particulière dans son implémentation. Une instruction APPEND est une instruction TEXT dont le premier caractère est le caractère ⊢ (ligne 7 colonne F). Comme le caractère APPEND prend la première place et qu'une chaîne de caractères ne peut dépasser quinze caractères, on ne peut ajouter que quatorze caractères dans une chaîne commençant avec APPEND. Si le caractère APPEND est inséré synthétiquement dans une chaîne ailleurs qu'en première position, il perd son rôle de "caractère de contrôle" et devient un caractère ordinaire.

Entrons des caractères synthétiques après une instruction APPEND. Tapez :

```
01 ENTER↑
02 "⊢ABCDEFGHIJKL"
```

GTO .001 puis BG et n'effacez pas la ligne. La ligne de texte créée par le Byte Grabber contiendra le TEXT 13 de l'ancienne ligne 02 jusqu'à ce que vous ayez terminé de travailler. SST pour lister ceci :

```
01 ENTER↑
02 " ?    ⊠"
03 CLD
04 -
05 *
06 /
07 X<Y?
08 X>Y?
09 X<=Y?
10 Σ+
11 Σ-
12 HMS+
13 HMS-
14 MOD
15 %
```

La ligne 03 est le caractère de contrôle APPEND (ligne 7 colonne F). Les lignes 04 à 15 correspondent aux caractères A à L. Consultez la ligne 4 de la table pour voir la correspondance. Faites maintenant GTO .004 et DEL

012 (XEQ ALPHA D E L ALPHA 0 1 2). Ceci détruit les lignes 04 à 15. Nous allons remplacer ces 12 caractères par des caractères synthétiques. Nous avons simplement à entrer les instructions correspondant aux caractères que nous voulons obtenir. Entrez donc les instructions :

instruction	**caractère**
04 -	A
05 LBL 00	天
06 LBL 11	ト
07 RCL 02	"
08 RCL 08	(
09 RCL 09	)
10 STO 11	;
11 ASIN	\
12 DEC	_
13 CLD	├
14 1/X	┬
15 +	@

Faites maintenant PACK pour vous assurer qu'il n'y a pas de nul présent. Effacez la ligne 04 pour créer un nul, puis GTO .001, BG, et flèche de correction. Vous devez voir :

```
01 ENTER↑
02 STO 15
03 "├¯天ト"();\_┬@"
```

Les données à fournir à LB pour cet exemple sont : 253, 127, 0, 1, 12, 34, 40, 41, 59, 92, 95, 127, 96 et 64.

Mettez "ABC" dans le registre ALPHA et exécutez la ligne 03. Le registre ALPHA contiendra alors "ABC¯天ト"();_├┬@". Si vous faites CLA et exécutez la ligne 03 vous aurez une surprise. Le registre ALPHA contiendra "天ト"();_├┬@". Le caractère nul (qui apparaît comme surligné "¯") a disparu ! La règle générale est que les nuls n'apparaissent que s'ils sont au milieu ou en fin d'une chaîne dans le registre ALPHA.

Si vous faites ASTO X, même les nuls intérieurs et terminaux seront invisibles dans le registre X, mais ils seront toujours présents. Ceci peut être vérifié en essayant le test X = Y?. La réponse sera NO (non) si, par exemple, le registre X contient un nul invisible et pas le registre Y, même si tous deux s'affichent de la même façon. Ce comportement n'est pas assez utile pour justifier un exemple, mais vous devez être mis en garde que le fait de visionner par VIEW une chaîne stockée par ASTO ne donnera pas l'image exacte de la chaîne si celle ci contient des nuls. Vous devez utiliser ARCL puis AVIEW si vous avez des doutes.

Les possesseurs d'imprimante savent peut être que la fonction BLDSPEC peut être utilisée pour créer n'importe quel caractère synthétique. Par exemple, la séquence :

```
01 .           (point décimal)
02 X<>Y
03 BLDSPEC
04 PRX
```

créera un caractère correspondant à la valeur décimale (0 à 127) contenue dans X. Elle l'imprimera ensuite.

Essayez 38, GTO .001, R/S et vous obtiendrez le symbole ¤, qui est un caractère synthétique. La case ligne 2 colonne 6 de la table des codes montre pour comparaison les deux versions de ce symbole : à l'affichage et à l'impression. Essayez 5, R/S et vous obtiendrez l' homme à un seul bras ⅄ à l'affichage et la lettre grecque β sur l'imprimante. La case ligne 0 colonne 5 prévoyait bien ce résultat. Un grand nombre des 128 caractères standards de l'imprimante apparaissent à l'affichage comme des novas.

Ceci est dû au fait que l'afficheur à 14 segments n'a pas une aussi grande souplesse que la matrice de points qu'utilise l'imprimante.

Les possesseurs du module de fonctions d'extension mémoire (X-Fonctions) ou d'une HP-41CX disposent d'une puissante instruction, XTOA, qui peut être utilisée afin de créer des caractères synthétiques. XTOA est une alternative beaucoup plus rapide du programme DC du PPC ROM. Assignez XTOA (ou DC) à une touche qui vous convient et tapez CLA, 38, XTOA. Passez en mode ALPHA et vous devez voir le caractère synthétique & . Si vous faites maintenant ALPHA (pour sortir du mode ALPHA), 5, XTOA, ALPHA (pour y revenir), vous voyez &⅄ . Le caractère "homme à un seul bras" (de code décimal 5) a été ajouté au contenu du registre ALPHA. Pour comparer les versions de l'imprimante, exécutez PRA.

Les possesseurs d'imprimante apprécieront l'économie d'octets réalisée en utilisant des chaînes de caractères synthétiques pour générer des minuscules ou mélanger majuscules et minuscules. Considérons la méthode classique pour imprimer "Hewlett-Packard" :

```
01 "H"
02 ACA          (charge H dans la mémoire tampon d'impression)
03 SF 13        (passe en minuscules)
04 "EWLETT-"
05 ACA          (ajoute les minuscules à la mémoire tampon)
06 CF 13        (revient en majuscules)
07 "P"
08 ACA
09 SF 13
10 "ACKARD"
11 ACA
12 PRBUF        (imprime le contenu de la mémoire tampon)
13 CF 13
```

Cette monstruosité demande 37 octets de mémoire, à comparer aux 18 octets nécessités par la chaîne synthétique "Hewlett-Packard" suivie d'une instruction PRA. De plus chaque changement de mode, entre les majuscules et les minuscules par exemple, utilise un octet dans la mémoire tampon de l'imprimante. La méthode utilisant des chaînes synthétiques économise la mémoire tampon autant que la mémoire programme. Bien sûr, la plupart des minuscules (toutes sauf a, b, c, d, e) apparaissent comme des novas à l'affichage, bien que la ligne s'imprime correctement dans un listage du programme. Si vous tolérez cet affichage quelque peu nébuleux, vous pouvez gagner de précieux octets en utilisant les chaînes synthétiques là où vous voulez employer des minuscules ou un mélange majuscules et minuscules.

Les chaînes synthétiques ont d'autres applications que la création de caractères non standards à l'affichage. Elles permettent d'entrer facilement et rapidement des octets sous contrôle d'un programme. Les programmes chargeurs d'octets (chapitre 3), ou d'assignement (chapitre 4), et d'autres programmes synthétiques très puissants utilisent intensivement les chaînes de caractères synthétiques. En utilisant le premier exemple de cette section, nous allons illustrer la simplicité des chaînes synthétiques en

comparant cette méthode à la plus performante des alternatives, l'instruction XTOA du module X-Fonctions.

But : Créer la chaîne "HP'S ₰1"

Meilleure méthode :L'instruction synthétique 01 "HP'S ₰1"

Nombre d'octets : 8 Vitesse d'exécution : rapide

Meilleure alternative : XTOA ou DC

```
01 "HP"
02 39
03 XTOA (ou XROM DC)
04 "⊦S"
05 35
06 XTOA (ou XROM DC)
07 "⊦1"
```

Nombre d'octets : 18 Vitesse d'exécution : lente

Les possesseurs d'imprimante qui aiment utiliser BLDSPEC pour se fabriquer leurs caractères personnalisés peuvent économiser beaucoup d'octets et accélérer leurs programmes en utilisant les chaînes de caractères synthétiques. La séquence : chaîne de 7 caractères, RCL M, ACSPEC, remplaçe la séquence : nombre, BLDSPEC, nombre, BLDSPEC, ..., nombre, BLDSPEC, ACSPEC. L'instruction RCL M sera expliquée à la section 2G. Des précisions sur la correspondance entre les codes à fournir à la fonction BLDSPEC et la chaîne de 7 caractères à introduire peut être trouvée dans le manuel d'utilisation du PPC ROM à la page du programme BL.

Pour ceux qui seraient tentés par des choses plus exotiques, les chaînes synthétiques nécessitent souvent d'introduire des octets des lignes 9 à F de la table des codes, qui correspondent aux instructions codées sur plusieurs octets. La technique du Byte Grabber décrite précédemment ne permet généralement pas la création de telles chaînes. La méthode la plus simple pour créer ces instructions consiste à utiliser un programme chargeur d'octets, tel celui que vous découvrirez au chapitre trois. Mais **attention !** Les chaînes contenant des octets des lignes 8 à F apparaissent comme prévu à l'affichage mais donnent des choses étranges à l'impression. Ces octets des lignes 8 à F s'affichent tous comme des novas. S'ils sont imprimés avec la fonction PRA, ils apparaîtront tels qu'ils sont décrits dans la table des codes. Par exemple, le caractère ligne C colonne D donne une nova à l'écran, mais la lettre M à l'impression. Néanmoins, si vous **listez** un programme contenant ces caractères, ils **n'apparaîtront pas** et ne seront même pas remplaçés par des espaces pour marquer leur présence. Certains de ces caractères, ceux qui ont étés ombrés dans la table des codes, auront de plus un comportement parasite lors de l'impression (laissant des espaces, passant en minuscules, etc...). Si ceci vous arrive, positionnez vous manuellement (GTO) sur la ligne suivante et LISTez la suite du programme (instruction LIST). En principe, les listages en mode NORMAL trahissent la présence de caractères synthétiques car les numéros de lignes contenant ces caractères invisibles sont décalés.

2F. L'instruction TEXT 0

La HP-41 autorise (en mode PRGM) des chaînes longues de 15 caractères ou 14 en plus du symbole APPEND. Le premier octet d'une chaîne de caractères appartient à la ligne F de la table des codes, le numéro de colonne dénotant le nombre de caractères de la chaîne.

Mais qu'en est-il de la colonne 0 ? Par extension de la règle

précédente, un octet ligne F colonne 0 caractériserait donc une chaîne de longueur nulle. On peut donc penser que TEXT 0 est équivalent à CLA. Voyons cela. Tapez :

	entrée pour LB:	pour MK:
01 "ABC"		
02 STO IND T	240	240, 240

Pour entrer la ligne 02, pressez STO, SHIFT, . (point décimal),9(T). GTO .001, BG, et flèche de correction. Le STO a été dérobé et le IND T (ligne F colonne 0) s'est transformé en instruction TEXT 0. Cette instruction apparaît comme le symbole texte tout seul. A l'impression, on obtient "" (rien entre les guillemets). Lancez maintenant le programme et passez en mode ALPHA. Surprise ! La chaîne "ABC" qui a été chargée en ALPHA par la ligne 01 est toujours là. L'instruction TEXT 0 n'est donc pas l'équivalent de CLA. D'autres manipulations revèlent que TEXT 0 n'a aucun effet sur le registre ALPHA, ni sur aucun autre registre (y compris celui des drapeaux). TEXT 0, comme la plupart des autres instructions de programme, valide les mouvements de pile. (consultez le **Manuel d'utilisation** pour des précisions sur les mouvements de la pile).

Quelle peut donc bien être l'utilité de l'instruction TEXT 0 si elle ne fait rien ? Supposons que vous vouliez incrémenter un nombre entier inconnu se trouvant dans le registre Y sans bouleverser la pile. ISG Y fera cela mais sautera également une ligne de programme si Y n'est pas négatif. C'est pourquoi nous devons faire suivre ISG Y par une instruction qui n'affectera pas l'état du calculateur si elle est exécutée. TEXT 0 est l'instruction qu'il nous faut. De plus, c'est la seule instruction de ce type de la HP-41 qui n'occupe qu'un seul octet. Des instructions qui ne font rien telle TEXT 0 sont appelées des NOPs (**No** **OP**eration : pas d'opération). Une touche NOP existait sur les HP-25, HP-33, HP-55 et d'autres calculateurs. La programmation synthétique a donc donné cette "fonction" à la HP-41. Vous trouverez des séquences telles :

```
01 ISG X
02 TEXT 0
```

dans beaucoup de programmes synthétiques. Vous pouvez utiliser une telle séquence n'importe où vous avez besoin d'une "incrémentation sans saut". Bien sûr, TEXT 0 peut également être utilisé après une instruction DSE pour décrémenter sans sauter.

2G. Utilisation du registre ALPHA pour le stockage des données

Nous avons vu qu'un octet de mémoire programme était nécessaire pour représenter chaque caractère d'une chaîne. Nous pouvons donc nous attendre à ce que les 24 caractères du registre ALPHA nécessitent 24 octets de mémoire, soit 24/7 = 3 registres et 3 octets. Ces registres, ainsi que ceux de la pile, le registre des drapeaux et d'autres, se trouvent dans une zone à part de la mémoire appelée fourre-tout ou registres d'état. Le nom registres d'état provient du fait que la fonction WSTS (**W**rite **ST**atu**S** : écriture de l'état) du lecteur de cartes, écrit le contenu de ces registres sur la piste 1 de la carte d'état.

Le registre des drapeaux et le pointeur programme pouvant être modifiés directement par des instructions synthétiques ; peut-être pouvons-nous avoir accès aux 3+ registres qui composent le "registre" ALPHA. Les octets suffixes du registre des drapeaux et du pointeur programme se trouvent à la ligne 7 de la table, respectivement aux colonnes E et C. Vous

avez certainement pensé que les autres suffixes de la ligne 7 correspondent aux autres registres d'état. Mais avant de commencer à essayer quelques manipulations, prenez garde ! Vous pouvez rappeler en toute confiance (par RCL) le contenu de n'importe lequel des registres d'état (la "normalisation" décrite à la section 2C ne concerne pas les registres d'état), mais ne modifiez pas leurs contenus sans savoir ce que vous faites, sinon attendez vous au pire. Ainsi, si vous mettez à zéro le registre c, **vous obtiendrez MEMORY LOST.**

Le registre ALPHA occupe les registres d'état nommés M, N, O et une partie du registre P. Tant que vous ne tenez pas à conserver ce qui se trouve en ALPHA, vous pouvez utiliser librement M, N et O, comme vous le feriez avec des registres de données. Avec ce que vous connaissez du Byte Grabber, vous devriez pouvoir créer le programme suivant :

```
01 LBL "RSHF"
02 CLX
03 X<> O
04 X<> N
05 X<> M.
```

Si vous ne vous en sortez pas, voyez la réponse à la fin de cette section.

Pour le moment, intéressons-nous à l'instruction X<> M. Essayez la séquence CLA, 1.274065002 E-40, X<> M. Pour faire X<> M vous pouvez faire GTO.005 et SST en mode RUN (hors PRGM). Passez maintenant en mode ALPHA et vous voyez ☒'@e)┬ . Que s'est-il passé ? Référons-nous à la table des codes pour identifier les 7 octets formant la chaîne de caractères. Les 7 octets sont les suivants :

Valeur Hexadécimale des octets	01	27	40	65	00	29	60
Sous forme de caractères	☒	'	@	e		)	┬
Registre sous forme numérique	+1,	27	40	65	00	2E-	40

signe mantisse (10 chiffres) signe exposant

Les quatorze chiffres hexadécimaux représentant les sept octets sont 01274065002960. Les dix chiffres du registre X sont immédiatement reconnaissables dans les second à onzième chiffres hexadécimaux. Le premier de ces 14 chiffres code le signe : 0 représente un nombre positif, 9 un nombre négatif et 1 une donnée alphanumérique. Les trois derniers chiffres représentent l'exposant et son signe. Si le douzième chiffre est nul, l'exposant est positif ; si le douzième chiffre est 9, l'exposant est négatif. Les 2 derniers chiffres sont les chiffres de l'exposant si celui-ci est positif. S'il est négatif, les deux derniers chiffres sont 100 plus l'exposant négatif. Dans notre cas, l'exposant est -40, les deux derniers chiffres sont donc 100+(-40) = 60. Voici une règle simple qui marche aussi bien pour les exposants positifs que négatifs : ajoutez 1000 à l'exposant avec son signe, gardez seulement les trois derniers chiffres du résultat. Ceci donne la représentation correcte de l'exposant. Dans notre cas : 1000+(-40) = 960.

Si nous exécutons GTO .005 et SST pour exécuter de nouveau X<> M, le nombre 1.274065002 E-40 revient dans le registre X et ALPHA est à nouveau vide. Essayons autre chose. Avec le même nombre en X, exécutons X<> M, passons en mode ALPHA, appuyez sur APPEND (SHIFT K), flèche de correction, puis A. Vous avez maintenant la chaîne ☒'@e)A . Sortons du mode ALPHA et

exécutons X<> M encore une fois : nous obtenons 1.274065002 E-59. Le caractère A ayant pour code hexadécimal 41, l'exposant devient 41-100 = -59.

Essayez maintenant d'explorer seul les correspondances entre les nombres et les chaînes de sept caractères en utilisant l'instruction X<> M. Beaucoup de nombres génèreront des novas. Vous devez savoir que si vous mettez une chaîne dans le registre X en utilisant X<> M, le résultat peut se comporter étrangement si les deux chiffres codant le signe ne sont pas 0 ou 9, ou s'il se trouve des chiffres hors de l'intervalle 0-9 (c'est-à-dire non décimaux).

Lorsque vous utiliserez M comme registre de stockage, vous ne vous préoccuperez certainement pas de savoir si ce nombre apparaît comme une chaîne, mais l'équivalence caractère/nombre peut être très utile dans certaines applications avancées de la programmation synthétique. Par exemple, si nous voulons entrer le nombre $1.274065002 \times 10^{-40}$ dans un programme, nous pouvons économiser 5 octets de mémoire en utilisant "Ŧ'ⓐe)Ŧ" suivi de RCL M.

Les instructions X<> N et X<> O se comportent de la même manière que X<> M. La différence réside dans le fait que X<> M place le nombre dans les 7 positions **les plus à droite** du registre ALPHA. Les instructions X<> N et X<> O donnent accès aux deux groupes suivants de sept caractères en allant de la droite vers la gauche. La figure 2.2 éclaircira un peu tout cela. Vous pouvez également essayer ce court exemple. Chargez "ABCD EFGHIJKLMNOPQRSTUV" dans le registre ALPHA. Exécutez CLX puis X<> O (faites CLX, GTO.002, SST, SST). Le registre ALPHA contient maintenant "A IJKLMNOPQRSTUV". Les sept caractères qui occupaient le registre O (voir figure 2.2) ont étés remplacés par les surlignés qui correspondent aux octets nuls (ligne 0 colonne 0). Le registre O contient maintenant le nombre zéro. Exécutez X<> N et ALPHA contiendra "A BCDEFGHPQRSTUV". Faites maintenant X<> O et vous obtenez "AIJKLMNOBCDEFGHPQRSTUV". Donc, en plus de leur utilité pour le stockage des données, les instructions STO, RCL et X<> sur les registres d'état M, N et O peuvent être utilisées pour morceler et recombiner des chaînes dans le registre ALPHA. Ces propriétés de traitement de chaînes sont intensivement utilisées dans les applications avancées de la programmation synthétique afin d'isoler des octets pour les décoder ou remplacer certains octets d'une chaîne.

Une manipulation de chaîne facilement accessible est un décalage du registre ALPHA de 7 octets vers la droite. Le programme "RSHF" réalise ce décalage sur des chaînes allant jusqu'à 21 caractères, détruisant les sept caractères les plus à droite.

```
01 LBL "RSHF"
02 CLX
03 X<> O
04 X<> N
05 X<> M (voir fig. 2.2, p. 21).
```

Par exemple "ABCDEFGHIJKLMNOP", XEQ "RSHF", retourne "ABCDEFGHI". Vous pouvez faire SST en mode ALPHA pour voir comment "RSHF" fonctionne.

Voyons maintenant comment l'accès aux registres d'état M, N et O peut nous servir pour manipuler des nombres. Avoir trois registres à part peut grandement soulager les problèmes de conflits de registres. Vous pouvez maintenant réécrire beaucoup de vos sous-programmes afin qu'il n'utilisent plus aucun registre de données. Ceci les rend compatibles avec n'importe quel programme qui utilise seulement des registres de données. Par exemple, beaucoup de routines du PPC ROM n'utilisent pas de registre de données,

pour que le programme appelant ces routines puisse utiliser n'importe lequel de ces registres. Pour améliorer cette compatibilité, il est souhaitable de ne pas utiliser ce qui reste dans M, N et O après avoir appelé un sous-programme.

Des sous-programmes très courts peuvent souvent se servir d'une partie du registre ALPHA pour éviter d'utiliser les registres de la pile ou des registres de données. Le but recherché étant d'obtenir une fonction telle les fonctions standards de la machine -sauvant X en LAST X, sauvant le contenu de T (en T), et retournant le résultat en X.

Ecrivons pour illustrer ceci un sous-programme nommé "CNK" qui calculera le nombre de combinaisons de k parmi n :

$$C(n,k) = \frac{n!}{k!(n-k)!} = \frac{(n-k+1)(n-k+2)\ldots n}{k(k-1)\ldots 1}$$

Le programme doit prendre les valeurs de n et k respectivement dans les registres Y et X et retourner le résultat C(n,k) en X. Les anciens contenus des registres Z et T finissent en Y et Z comme dans une fonction standard. La valeur de k doit être sauvée en LAST X et celle de n en T.

Du fait de la complexité du calcul, "CNK" ne peut pas préserver les contenus de Z et T sans utiliser un registre fourre-tout. Nous utiliserons le registre M. Ceci rend "CNK" compatible avec n'importe quel programme utilisant seulement des registres de données. Un exemple de programme "CNK" possible est listé ci-dessous pour que vous puissiez l'essayer.

```
01 LBL "CNK"          Données pour LB/MK
02 -
03 E                        27 ou 27, 0
04 STO M                    145, 117
05 RDN
06 LAST X
07 X>Y ?
08 X<>Y
09 LBL 01
10 X<>Y
11 ISG X
12 TEXT 0                   240 ou 240, 240
13 ST* M                    148, 117
14 X<>Y
15 ST/ M                    149, 117
16 DSE X
17 GTO 01
18 X<>Y
19 RDN
20 X<> M                    206, 117
21 END
```

Pour créer les lignes synthétiques, utilisez STO 27, STO IND 17, RDN, STO IND T, STO IND 20, RDN, STO IND 21, RDN, STO IND 78, RDN. Pour chacune des cinq instructions STO, dérobez le préfixe en vous positionnant sur la ligne précédente en mode PRGM, puis en pressant BG, puis la flèche de correction.

Testez le programme "CNK" en faisant 88 ENTER↑ 3 R/S, puis 88 ENTER↑ 85 R/S. Les deux doivent donner le même résultat : 109 736. Ce résultat est le nombre d'accords possibles avec 3 notes sur un clavier de piano à 88

touches.

Voici maintenant comment "CNK" fonctionne. Au début, X contient la valeur de k et Y la valeur de n. "CNK" met à 1 le registre M à la ligne 04 afin que les instructions ST* M et ST/ M dans la boucle commençant au label 01 fonctionnent comme il faut au premier passage. Après exécution de la ligne 06, M contient 1, X contient k, et Y contient n-k. Les lignes 07 et 08 permutent k et n-k si n-k est la plus petite des deux valeurs. On utilise ici le fait que C(n,k) = C(n,n-k) pour accélérer l'exécution si cela est possible. La boucle au LBL 01 incrémente n-k et multiplie le contenu de M par le résultat. Puis à la ligne 14, k est rappelé dans X, après quoi on divise le contenu de M et on décrémente. A cet endroit (de retour au LBL 01 prêt pour un second passage dans la boucle), X contient k-1, Y contient n-k+1 et M contient (n-k+1)/k, le premier facteur de l'expression de C(n,k) donnée plus haut. La boucle est exécutée k fois, après quoi on trouve 0 en X et n en Y. Les trois dernières lignes mettent Y en T, et rappellent le résultat de M dans X, en remettant M à zéro. Vous pouvez modifier les lignes 04, 13, 15 et 20 de "CNK" pour utiliser le registre 0 plutôt que M. Cela permet de conserver des chaînes de 14 caractères en ALPHA lorsque "CNK" est utilisé.

Voici la procédure pas à pas promise pour créer les instructions d'accès au registre ALPHA. Tapez :

```
01 LBL "RSHF"          données pour LB/MK
02 CLX
03 STO IND 78          X<> 0 = 206, 119
04 CLX
05 STO IND 78          X<> N = 206, 118
06 LAST X
07 STO IND 78          X<> M = 206, 117
08 RDN
```

GTO .006, BG, flèche de correction, GTO .004, BG, flèche de correction, GTO .002, BG et flèche de correction. Vous avez maintenant toutes les instructions synthétiques nécessaires pour "RSHF".

2H. Utilisation des autres registres d'état pour le stockage

Les registres d'état **P**, **Q**, et **a** peuvent être utilisés dans certains cas pour stocker temporairement des données. Des détails sur la façon dont le système d'exploitation de la HP-41 utilise ces registres sont donnés à la section 6A de ce livre, mais nous allons en donner une brève description ici.

Le registre **P** peut être utilisé pour le stockage en cours de programme, mais son contenu sera modifié si une ligne de donnée numérique est exécutée, ou si une opération entraînant l'affichage d'un nombre a lieu.

Le registre **Q** peut également être utilisé, mais son contenu est également susceptible d'être altéré. Si vous effectuez un GTO alpha ou un XEQ alpha (c'est-à-dire, un GTO ou XEQ se référant à un label apparaissant dans le catalogue 1 ou le catalogue 2), vous perdrez le contenu de Q. Ceci ne se produit pas avec les instructions LBL "alpha". Ni avec les instructions XROM, qui ont une structure différente des instructions XEQ alpha, comme nous le verrons au prochain chapitre. Q sera également modifié si vous entrez un nom alpha au clavier pour une instruction GTO, XEQ, ou LBL. Les autres instructions qui modifient Q sont : les entrées numériques,

SIN, COS, R-P, P-R, Y↑X, SDEV et toutes les instructions qui affichent le contenu du registre ALPHA (AVIEW, PROMPT, ou PSE avec AON actif). Le registre d'état Q est utilisé intensivement par l'imprimante HP 82143A pour échanger des informations avec le calculateur. Si vous prévoyez d'avoir une imprimante 82143A connectée pendant l'exécution de vos programmes, évitez d'utiliser le registre Q pour le stockage des données.

Le registre **a** peut être utilisé par n'importe quel programme n'utilisant pas plus de deux niveaux de sous-programmes. Cela signifie que si le programme ne contient pas d'instruction XEQ, il ne doit pas être appelé par un sous-programme qui ne soit pas du premier niveau d'imbrication. Si un programme utilisant le registre a est appelé comme second niveau de sous-programme, le END ou RTN du programme principal appelant ne doit pas être l'instruction arrêtant le programme. En effet, si le registre a n'est pas vide (égal à zéro) une instruction RTN ou END renverra le pointeur à l'adresse contenue dans le registre a. Vous devez également savoir qu'une instruction XEQ ou RTN modifiera le contenu du registre a, en le décalant de 2 octets. N'exécutez jamais PSIZE (du module X-Fonctions) avec quelque chose dans le registre a. Le calculateur penserait que vos données sont des adresses de retour devant être réévaluées en fonction de la nouvelle partition mémoire (SIZE). Tout ceci devrait s'éclaircir après avoir lu le chapitre 6.

Problèmes (Solutions après le chapitre 6)

2.1 En utilisant TONE P et l'instruction standard TONE 8, construire une séquence produisant le code Morse pour "CQ" (tâ-ti-tâ-ti, tâ-tâ-ti-tâ).

2.2 En utilisant le Byte Grabber, fabriquez l'instruction -E1.
Petite aide : faites d'abord E1

2.3 En utilisant RCL d/STO d, écrivez un court programme pour voir les dix chiffres d'un nombre en X sans modifier le format d'affichage.
Petite aide : Modifiez la routine ci-dessous pour restaurer le format d'affichage.

```
01 LBL "VX"
02 "  "  (2 espaces)
03 SCI 9
04 ARCL X
05 AVIEW
06 END
```

2.4 En utilisant une boucle STO b/RCL b, calculez le nombre d'or x = 1+1/x, en affichant les itérations successives.

2.5 a) Construisez une séquence utilisant une chaîne synthétique qui affichera le message "X(n)=?", ou n est un entier contenu dans le registre 00.

b) Modifiez cette séquence pour conserver le format d'affichage.

2.6 Construisez une séquence d'affichage qui affichera "OUT=xμV" sans modifier le format d'affichage, où x est à rappeler par ARCL du registre X en format FIX 2.

2.7 Construisez une véritable fonction MOD qui fonctionne comme une fonction standard. Les registres Z et T doivent être sauvegardés, x

conservé dans L, y mod x dans Y et (y-(y mod x))/x dans X. Vous aurez besoin d'utiliser un registre scratch, par exemple M.

2.8 En utilisant le Byte Grabber, construisez l'instruction de deux octets F1 F0 hex (une chaîne de 1 caractère, ce caractère étant F0 hex).

CHAPITRE TROIS

CHARGEMENT D'OCTETS

Si vous avez suivi les exemples du deuxième chapitre en utilisant le Byte Grabber, vous serez certainement d'accord avec le fait que le Byte Grabber est un outil puissant pour créer un grand nombre d'instructions synthétiques différentes. Néanmoins, si vous devez créer plusieurs instructions synthétiques à la suite, une autre approche sera bien plus rapide. Un programme spécial, appelé chargeur d'octets (byte loader), sera utilisé pour créer les instructions désirées, les chargeant directement en mémoire programme. Vous avez simplement besoin de lui fournir la valeur décimale (0 à 255) de chaque octet de la séquence.

La théorie concernant les chargeurs d'octets est décrite dans le manuel d'utilisation du PPC ROM à la section LB. Plusieurs membres de PPC, parmi lesquels William Cheeseman, Roger Hill, John McGechie, William Wickes, et l'auteur ont été les pionniers dans l'écriture de programmes chargeurs d'octets. Ce livre se limitera à présenter les différentes applications des programmes chargeurs d'octets.

Il vous sera présenté dans ce chapitre trois programmes chargeurs d'octets différents. Le premier d'entre eux, appelé "LB" (Load Bytes) ne nécessite qu'une HP-41 de base pour fonctionner. Ce programme, écrit par Clifford Stern, occupe 214 octets et tient donc sur une seule carte magnétique.

Le second est le programme LB inclus dans le PPC ROM, superbe chargeur d'octets écrit par Roger Hill. Si vous avez un PPC ROM, familiarisez-vous avec le fonctionnement de LB. Il est similaire, mais pas tout à fait identique à celui de "LB".

Le troisième chargeur d'octets, appelé "LBX", demande la présence du module de fonctions d'extension mémoire. Ce programme, également écrit par Clifford Stern, est une version plus courte et plus rapide de "LB" qui utilise intensivement des fonctions du module X-Fonctions comme XTOA. Si vous voulez utiliser "LBX" référez-vous à la section 3.5 pour y trouver le listage du programme.

Malgré sa compacité, "LB" fait pratiquement les mêmes choses que la version LB du PPC ROM, seuls la possibilité d'interruption et les routines de nettoyage lui manquent. Toute la souplesse de la version du ROM n'aurait pu être introduite sans allonger de façon significative le programme. Les programmes en ROM ne sont pas aussi contraignants sur le plan de la longueur car ils n'utilisent pas du tout de mémoire utilisateur. En tout cas, ce que "LB" perd en souplesse, il le regagne en rapidité. Si vous avez un module X-Fonctions, vous utiliserez certainement "LBX" (voir section 3.5), puisqu'il est à la fois plus court et plus rapide que "LB".

Si vous avez accès à un lecteur optique de codes-barres (Wand), vous pouvez entrer "LB" ou "LBX" directement en lisant les codes-barres. L'appendice E contient les codes-barres de tous les programmes utilitaires contenus dans ce livre, vous permettant l'accès à une méthode rapide et sans risque d'erreur pour entrer ces programmes synthétiques dans votre HP-41. N'oubliez pas d'utiliser une feuille de plastique transparent que vous placerez sur les codes-barres avant de les lire afin de les protéger du frottement du lecteur qui pourrait les endommager. Bien sûr, si vous voulez acquérir une plus grande maîtrise du Byte Grabber, vous pouvez oublier les codes-barres pour un temps. Les Editions du Cagire peuvent également vous fournir une cassette contenant tous les programmes de ce livre, si vous avez la chance de posséder un lecteur de cassettes.

Si vous n'avez ni PPC ROM, ni module X-Fonctions, commencer à rentrer ces instructions vous permettra de créer les lignes synthétiques du programme "LB" de Clifford Stern :

```
01 ENTER↑
02 STO IND 16            (Pressez STO SHIFT 1 6)
03 MEAN                  (Pressez XEQ ALPHA M E A N ALPHA)
04 STO IND 17
05 RDN
06 STO IND L             (Pressez STO SHIFT . L)
07 CLD                   (Pressez XEQ ALPHA C L D ALPHA)
08 ENTER↑
09 ENTER↑
10 LBL 01
11 STO IND 78
12 RDN
13 STO IND 78
14 AVIEW                 (Pressez ALPHA SHIFT R/S ALPHA)
15 STO IND 78
16 AVIEW
17 STO IND 17
18 RDN
19 STO IND 78
20 AVIEW
21 STO IND 78
22 AVIEW
23 STO IND 78
24 RDN
25 STO IND 17
26 LAST X
27 STO IND 78
28 LAST X
29 STO IND 78
30 SDEV
31 STO IND 17
32 SDEV
33 STO IND Y             (Pressez STO SHIFT . Y)
34 CLD
35 ENTER↑
36 STO IND 78
37 SDEV
38 STO IND 16
39 RDN
40 STO IND 17
41 SDEV
```

Maintenant dérobez et détruisez les octets STO des lignes 40, 38, et 36. (Par exemple pour la ligne 40 : GTO .039, BG, flèche de correction) effacez la ligne 35 (ne PACKez pas) puis dérobez et effacez les octets STO des lignes 33, 31, 29, 27, 25, 23, 21, 19, 17, 15, 13 et 11. Détruisez les lignes 08 et 09 (ne PACKez pas), puis dérobez et détruisez les octets STO des lignes 06, 04 et 02. Effacez la ligne 01 et entrez les lignes non synthétiques nécessaires pour compléter le listage de "LB" donné ci-dessous. La ligne 61 est une chaîne de caractères contenant seulement un espace. Utilisez 1E4 à la ligne 72. Si vous le désirez, utilisez le Byte Grabber pour enlever le 1 de tête. En fait, si vous vous imprégnez de

```
01♦LBL 01
02 CLST
03 BEEP
04 STOP
05 GTO "++"

06♦LBL "LB"
07 FS? 50
08 GTO 02
09 1
10 ENTER↑
11 ENTER↑
12 CLA
13 CF 21
14 AVIEW
15 -10
16 GTO "++"

17♦LBL 02
18 7
19 /
20 INT
21 FIX 0
22 CF 29
23 ARCL X
24 "⊢ REGS."
25 TONE 8
26 AVIEW
27 PSE
28 RCL b
29 STO [
30 "⊢♦♦x̄"
31 X<> [
32 X<> d
33 CF 04
34 CF 05
35 CF 06
36 FS?C 07
37 SF 05
38 FS?C 08
39 SF 06
40 FS?C 09
41 SF 07
42 FS?C 10
43 SF 09
44 FS?C 11
45 SF 10
46 FS?C 12
47 SF 11
48 X<> d
49 INT
50 DEC
51 1
52 +
53 .1
54 %
55 +
56 +

57♦LBL 03
58 1.007
59 ENTER↑

60♦LBL 04
61 " "
62 ARCL Y
63 "⊢?"
64 AVIEW
65 STO [
66 RDN
67 STOP
68 FC?C 22
69 GTO 05
70 OCT
71  E4
72 +
73 X<> d
74 FS?C 19
75 SF 20
76 FS?C 18
77 SF 19
78 FS?C 17
79 SF 18
80 FS? 15
81 SF 17
82 FS? 14
83 SF 16
84 X<> d
85 X<> [
86 "⊢**"
87 STO \
88 ARCL Y
89 X<> \
90 ISG Y
91 GTO 04
92 SIGN
 93 X<> c
 94 LASTX
95 STO IND T
 96 X<>Y
 97 STO c
 98 R↑
 99 DSE X
100 GTO 03
101 GTO 01

102♦LBL 05
103 "⊢♦"
104 ISG X
105 GTO 05
106 X<> c
107 RCL [
108 STO IND Z
109 X<>Y
110 STO c
111 GTO 01
112 END
```

LBL'LB
END 214 BYTES

Le programme de chargement d'octets "LB" de Clifford Stern

l'esprit de la programmation synthétique, vous désirerez certainement remplacer toutes les lignes d'entrée numérique "1" par des lignes "E".

Si vous utilisez la version avec X-Fonctions de "LB", la procédure décrite plus haut vous fournira toutes les lignes synthétiques dont vous avez besoin (plus quelques autres qu'il vous faudra détruire), mis à part la ligne 34, STO N. Pour obtenir cette ligne, tapez d'abord STO IND 17, LAST X puis dérobez et détruisez l'octet STO.

Remarques : Le suffixe [signifie M
Le suffixe \ signifie N
La ligne 30 est hex F4 7F 00 00 02
La ligne 62 est un simple espace
La ligne 103 est hex F2 7F 00

Comparez **très soigneusement** votre programme et le listage. Comme tout programme utilisant le registre d'état c, toute erreur peut causer un MEMORY LOST lors de l'exécution. C'est pourquoi il est souhaitable d'enregistrer le programme sur une carte magnétique, afin de ne pas avoir à tout retaper à cause d'une erreur mineure. Notez que certaines lignes synthétiques apparaissent différemment à l'écran et sur le listage. Ainsi, la ligne 30 s'affiche ┳├ ─ 0 et la ligne 103 ┳├ . Les instructions comprenant les suffixes M et N apparaissent également différemment à l'écran et sur l'imprimante. M est imprimé [et N apparaît comme \. Cette correspondance, qui est importante pour plusieurs registres d'état, apparaît à la ligne 7 de la table des codes. Par exemple, le suffixe 0 donne] à l'impression.

INSTRUCTIONS :

Voici la procédure d'utilisation du programme "LB" de Clifford Stern. La procédure du programme LB du PPC ROM est pratiquement identique ; les détails sont donnés dans le manuel d'utilisation du PPC ROM.

Plaçez les lignes suivantes à l'emplacement mémoire où vous souhaitez créer un groupe d'instructions synthétiques :

```
LBL "++"
+
+
+
etc...

+
+
XEQ "LB" .
```

Si vous utilisez le PPC ROM, la dernière instruction se transformera en XROM "LB". Le nombre de + entrés doit correspondre au nombre d'octets à créer plus 16.

Si vous n'avez pas introduit les instructions décrites plus haut en séquence, c'est-à-dire si vous êtes revenu en arrière pour insérer d'autres +, vous devez exécuter PACK. Si vous avez inséré un nombre de + qui est multiple de 7, vous n'avez alors pas besoin de PACKer. La raison de ceci vous apparaîtra au cinquième chapitre.

Comme vous utiliserez fréquemment le programme "LB", il est astucieux d'enregistrer la séquence LBL "++" etc... sur une carte magnétique. Si vous tapez 99 instructions + (la ligne 101 se trouvant alors XEQ "LB"), GTO.. et

GTO "++", la séquence tiendra en entier sur une piste de carte. Si vous avez un module d'extension mémoire (X-Memory), vous pouvez faire "++", SAVEP pour créer un fichier en mémoire étendue contenant la séquence LBL "++" etc... Elle pourra être rappelée lorsque nécessaire par la fonction GETP. Le stockage sur carte magnétique présente quand même l'avantage d'être à l'abri des MEMORY LOST.

A cet endroit, vous pouvez sortir du mode PRGM et taper XEQ "LB" ou simplement presser R/S si vous êtes à la dernière ligne du programme. "LB" commencera par vous dire combien de registres sont disponibles pour charger des octets, puis il demandera chacun des sept octets composant chaque registre. Le nombre de registres disponibles est INT((p-10)/7), où p est le nombre de + que vous avez entré. La table 3.1 est une référence rapide permettant de déterminer le nombre de + nécessaires.

Nombre de + nécessaires	Nombre de registres disponibles	Nombre d'octets disponibles
0-16	0	0
17-23	1	7
24-30	2	14
31-37	3	21
10+7n	n	7n

Table 3.1 Nombre de + nécessaires pour "LB"

En réponse à chaque demande de la valeur d'un octet, vous devez simplement entrer son équivalent décimal (0 à 255) et presser R/S. **ATTENTION:** Si vous souhaitez corriger une donnée avant de presser R/S, vous devez presser RDN avant d'entrer la valeur correcte. Ceci est nécessaire car des données très importantes se trouvent dans la pile pendant l'exécution de "LB". Cette remarque ne s'applique pas à la version LB du PPC ROM.

Lorsque vous avez terminé d'entrer tous les octets désirés, pressez simplement R/S sans rien entrer. Cela termine la procédure de chargement d'octets. Si vous n'avez plus de registres disponibles, "LB" se terminera automatiquement. Voyons un exemple.

Supposons que vous vouliez créer une copie du programme "CMOD" du problème 2.6. Rappelez-vous que le listage (que vous trouverez après le chapitre 6) demande les entrées suivantes pour "LB" :

	Entrées pour LB/MK
01 LBL "CMOD"	
02 X<>Y	
03 STO M	145, 117
04 X<>Y	
05 MOD	
06 ST-M	147, 117
07 LAST X	
08 ST/ M	149, 117
09 CLX	
10 X<>M	206, 117

Ces équivalents décimaux peuvent être utilisés pour créer les 4 instructions synthétiques codées sur 2 octets désirées.

Faites donc comme décrit plus haut LBL "++", 24 +, et XEQ "LB". Sortez du mode PRGM et R/S. Vous voyez le message "2 REGS." suivi par la demande "1?". Le message "2 REGS." signifiant que vous pouvez créer jusqu'à 14

octets (2 registres de 7 octets chacun).

En réponse à la demande "1?", entrez le premier équivalent décimal, 145, et R/S. Entrez les réponses à chacune des demandes en suivant le tableau ci-dessous :

Demande	Réponse
1?	145, R/S
2?	117, R/S
3?	147, R/S
4?	117, R/S
5?	149, R/S
6?	117, R/S
7?	206, R/S
1?	117, R/S
2?	R/S

Les sept premières entrées remplissent le premier registre, qui est inséré dans la zone LBL "++". Ceci remet à 1 l'index compteur d'octets (1^{e} octet du 2^{e} registre). Puis la pression de R/S sans entrée de donnée en réponse à "2?" termine le chargement, en complétant le second registre avec des nuls et en le stockant dans la zone "LBL ++" avant d'arrêter l'exécution. Lorsque "LB" s'arrête, vous pouvez presser une fois SST pour vous positionner sur le LBL "++". Vous pouvez passer maintenant en mode PRGM et examiner vos instructions synthétiques. Il est alors aisé d'effacer les + restants et de taper la partie non synthétique du programme "CMOD".

Comme vous pouvez le voir, il n'est pas nécessaire de bien connaître la programmation synthétique pour utiliser le programme "LB". Les seules connaissances requises sont comment obtenir les équivalents décimaux des instructions synthétiques à créer. Au deuxième chapitre, vous avez appris à les déterminer en utilisant la table des codes. Ainsi vous devriez être capable de dire, en regardant la ligne 1 de la table, que -E1 peut être créé avec "LB" en introduisant les valeurs 28, 27 et 17.

Il existe encore de larges zones de la table qui n'ont pas encore été expliquées ici, en particulier les lignes A à E. Ce chapitre évoquera cette zone, ainsi que les références particulières pour obtenir de plus amples informations, si nécessaire.

Ce qui suit est un résumé de la façon de déterminer les équivalents décimaux nécessaires à la création d'une instruction particulière. Dans la plupart des cas, il vous sera indispensable de consulter la table des codes. Les valeurs décimales sont inscrites dans l'angle inférieur gauche de chaque case de la table. Par exemple, le nombre décimal 126 (ligne 7 colonne E) correspond à l'instruction AVIEW, au suffixe d et au caractère **Σ**.

I. Instructions codées sur un octet

Toutes ces instructions sont standard, excepté TEXT 0 (ligne F, colonne 0, décimal 240). Chaque valeur décimale des lignes 0 ou 2 à 8 donnera naissance à une instruction non synthétique d'un octet à moins qu'elle ne soit précédée d'un autre octet réclamant un suffixe.

Les entrées de chiffres se combinent pour former une seule ligne correspondant à une entrée de nombre, sauf si elles sont séparées par un nul ou une autre instruction. Utilisez les valeurs décimales de la ligne 1, colonnes 0 à C pour faire des lignes synthétiques d'entrée de données numériques. Par exemple -E-3 est codé par 28, 27, 28, 19.

II. Instructions codées sur deux octets

Ces instructions ont un préfixe, ou premier octet, appartenant à la zone jaune de la table des codes.

La première catégorie d'instructions à deux octets est celle de la ligne 9, plus les colonnes 8 à D de la ligne A ainsi que les colonnes E et F de la ligne C de la table. Celles-ci prennent le premier octet dans la case contenant le nom de la fonction, plus un second octet dans la case contenant le suffixe désiré. Ainsi STO M est 145, 117 ; TONE C est 159, 104 ; RCL IND N est 144, 246 ; LBL X (label local) est 207, 115.

La seconde catégorie d'instructions à deux octets contient les GTO courts. Ceux-ci prennent leur premier octet à la ligne B et ont un second octet nul. Cet octet nul est rempli par la HP-41 la première fois que ce GTO est exécuté. L'octet de remplacement indique au microprocesseur la longueur du saut et sa direction.

La troisième catégorie d'instructions à deux octets contient les instructions GTO IND et XEQ IND. Celles-ci ont pour premier octet 174 (ligne A colonne E). Le second octet vaut de 0 à 127 pour les GTO IND, et de 128 à 255 pour les XEQ IND. Ainsi 174, 117 est GTO IND M, tandis que 174, 245 est XEQ IND M.

La dernière catégorie d'instructions à deux octets contient tous les XROMs. Ce sont les fonctions des périphériques qui résident dans une mémoire morte extérieure (e**X**ternal **R**ead **O**nly **M**emory). Lorsque le périphérique n'est pas connecté, la fonction apparaît sous la forme XROM i,j , où i et j sont deux nombres décimaux allant de 0 à 63 (en fait de 0 à 31 pour i). Le nombre i désigne l'identité du périphérique -- c'est pourquoi on l'appelle ROM ID. Certains périphériques contiennent deux ROMs de 4 kilo-octets chacune (1 kilo-octets = 1024 octets), chacune d'elles ayant son propre ROM ID. Le nombre j est le numéro d'ordre de la fonction (dans l'ordre du CAT 2) dans la ROM de 4K (kilo-octets).

Les instructions XROM se composent du chiffre hexadécimal A (en binaire 1010) suivi de deux groupes de six bits. Le premier d'entre eux représente, en binaire naturel, le numéro d'identification (0 à 31) de la ROM externe. Par exemple, l'imprimante est XROM 29 et le lecteur de cartes XROM 30. Le second groupe de six bits représente, toujours en binaire naturel, le numéro d'ordre (de 0 à 63) de la fonction à l'intérieur de la ROM externe. Ainsi, WSTS est la dixième fonction du lecteur de cartes. Ceci peut être contrôlé en exécutant CAT 2 avec le lecteur de cartes connecté : WSTS est bien la dixième fonction à apparaître après l'èn-tête du lecteur de cartes. Ainsi WSTS est codé XROM 30,10. En équivalents décimaux, cela donne 167, 138 (voir figure 3.1). En général, les valeurs des codes décimaux de XROM i,j sont :

```
Octet 1 = 160+INT(i/4)
Octet 2 = 64*(i MOD 4)+j
```

```
WSTS = XROM  30,  10
     = 1010 0111  1000 1010
         A   7     8   A
        167       138
```

FIGURE 3.1: Une instruction XROM classique et ses divers codages.

III. Instructions codées sur trois octets

Les instructions codées sur trois octets prennent leur préfixe, ou premier octet, dans la zone verte de la table des codes.

La première catégorie d'instructions codées sur trois octets comprend

les GTOs longs de 3 octets. Tous les GTOs concernant des labels autres que ceux allant de 00 à 14 sont des GTOs longs. Néanmoins avec "LB" vous pouvez créer des GTO longs (3 octets) se branchant sur les labels 00 à 14. Cette possibilité permet d'éliminer la limitation (distance de saut inférieure à 112 octets) normalement associée aux LBL 00 à 14. Cela ne veut pas dire que vous ne pouvez pas atteindre un LBL distant de plus de 112 octets avec un GTO court, mais le branchement sera beaucoup plus lent. Les sauts de plus de 111 octets ne peuvent être mémorisés par un GTO court car la représentation binaire de la longeur du saut ne tient pas dans l'espace alloué pour elle dans l'instruction GTO. Les GTO longs ont plus de place pour stocker la longueur du saut, il n'y a donc pas de contrainte sur cette distance de saut.

Les branchements à un LBL court (00 à 14) éloigné de moins de 112 octets du GTO peuvent utiliser les GTO courts ordinaires, sur deux octets, mais pour des sauts plus longs, vous devrez utiliser les GTO synthétiques longs sur trois octets. La différence entre un GTO 14 sur trois octets et un GTO 99 également sur trois octets, mis à part le fait que le premier est synthétique alors que le second ne l'est pas, est que le premier ne réclame qu'un LBL à un octet (LBL 14) alors que le second nécessite un LBL à deux octets (LBL 99). On obtient donc un gain d'un octet en utilisant un GTO long synthétique codé sur trois octets.

Les GTOs synthétiques nécessitent les entrées décimales suivantes :

Octet 1 = 208
Octet 2 = 0
Octet 3 = 0 à 127

L'octet 3 code le numéro du LBL. Ainsi 208, 0, 1 est un GTO 01 de trois octets tandis que 208, 0, 115 est GTO X (Qui branche l'exécution à un label local LBL X, décimal 207, 115).

La seconde catégorie d'instructions de trois octets comprend les XEQ non alphanumériques. Ils sont très similaires aux GTOs longs. La seule différence réside dans le fait que l'octet 1 est 224. Ainsi 224, 0, 98 représente XEQ 98 ; 224, 0, 116 représente XEQ L (qui branche l'exécution à un LBL L, décimal 207, 116).

Le troisième type d'instructions de trois octets est l'instruction END. Les données à fournir à "LB" pour créer un END sont 192 et 0 suivi d'un troisième nombre déterminant le type de END (voir ci-dessous).

Type du END	3^{e} donnée pour "LB"
END packé	9
END non packé	13
.END. packé	41
.END. non packé	45

Table 3.2: Troisième entrée à fournir à "LB" pour créer un END.

Compactez toujours la mémoire (par PACK) immédiatement après avoir crée un END ou un LBL alphanumérique pour l'incorporer dans le catalogue 1.

Les LBLs et ENDs forment une suite enchaînée dans le catalogue 1 qui part du .END. terminal, et qui code la distance entre l'instruction courante et la suivante en remontant la mémoire. Cette distance est

contenue dans les deux premiers octets de chaque END et LBL alpha. Le codage de la distance est fait sur le même schéma que pour les GTOs longs ou les XEQs, hormis le fait que le bit de direction n'est pas utilisé. (La direction est toujours du bas vers le haut de la mémoire). Les instructions données ici pour créer des END simplifient les choses en permettant au calculateur de PACKer pour remplir les octets codant la distance pour un chaînage correct du catalogue 1.

IV. Chaînes alphabétiques

Les chaînes de caractères nécessitent la présence d'un octet de tête appartenant à la ligne F de la table des codes (240 décimal plus le nombre de caractères dans la chaîne) tel que cela est décrit à la section 2E. Chaque caractère demande alors l'introduction d'une seule valeur, généralement comprise entre 0 et 127. Par exemple "X(5)=?" est représenté par 246 décimal suivi des six octets codant les caractères, ici : 88, 40, 53, 41, 61 et 63.

Les instructions append sont des chaînes de caractères où le caractère APPEND (ligne 7 colonne F = 127 décimal) est le premier de la chaîne. L'octet de tête doit être choisi en tenant compte du symbole APPEND dans la longueur de la chaîne. Ainsi "⊢@" est codée en décimal : 242, 127, 64.

Les GTOs alphanumériques sont simplement des chaînes de caractères précédées par l'octet ligne 1 colonne F (29 décimal). Ainsi les codes 29, 243, 65, 66, 67 représentent GTO "ABC". Les XEQs alphanumériques sont constitués pour leur part de l'octet ligne 1 colonne E (30 décimal) suivi d'une chaîne de caractères. Par exemple XEQ "FX" est codée 30, 242, 70, 88 en décimal. La mystérieuse instruction W^{T} que l'on trouve ligne 1 colonne F a la même constitution que les GTOs et XEQs alphanumériques, mais elle n'a pas d'autre utilité que de "planter" la machine. Il suffit d'enlever et de remettre les piles ou les batteries pour reprendre le contrôle.

Les labels alphanumériques sont composés de 4+n octets, n étant le nombre de caractères composant la chaîne. Les données à fournir à LB sont 192, 0, 241+n, 0, et les codes des n caractères. Ainsi LBL "A", un label global synthétique (global=apparaissant dans le CAT 1), est codé : 192, 0, 242, 0, 65. Si vous voulez assigner ce label synthétique à une touche, vous devez remplacer le second zéro par le code d'assignement de la touche désirée et lever (mettre à 1) le bit correspondant dans le registre d'état e ou ⊢ (voir section 6A). La correspondance entre la touche, l'octet la désignant et la position du bit dans ce registre est donnée dans le manuel du PPC ROM à la rubrique "Background for MK".

Une façon plus commode pour assigner un label global synthétique consiste à utiliser la fonction ASN. Tous les labels synthétiques ne pouvant pas être assignés par ASN peuvent l'être par la fonction PASN du module X-Fonctions. Seuls des cas très particuliers tel LBL ":" ne peuvent être assignés qu'avec PASN.

REMARQUE : Vous devez **toujours** PACKer immédiatement après avoir crée un LBL alpha ou un END pour reformer le chaînage du CATalogue 1.

Utilisez LB jusqu'à ce que la création des instructions synthétiques décrites au deuxième chapitre ne vous pose plus de problème.

PROBLEMES

3.1 Utilisez LB pour créer la séquence suivante :

```
E
STO 0
ST+ 0
```

```
X<> 0
STO M
ISG M
TEXT 0
ΣREG IND M
VIEW 0
FS? IND M
TONE E
"▶▶▶┌"
"┴┌"
ASTO N
VIEW N
```

Cette séquence d'instructions n'est pas particulièrement intéressante, mais elle comprend une vaste gamme d'instructions synthétiques qui peuvent être individuellement très utiles.

3.2 Ecrivez un court programme non synthétique qui fournira les données pour LB lorsque vous lui donnerez les deux paramètres d'un XROM. Pour une entrée i ENTER↑ j, les deux résultats doivent être 160+INT(i/4) et 64*(i MOD 4)+j conformément à ce qui est décrit dans le paragraphe concernant les instructions de deux octets. Ces deux résultats sont les données décimales à fournir à LB pour créer XROM i,j.

Ecrivez une version synthétique de ce programme qui remplace i et j par les deux résultats sans perdre les contenus des registres Z et T de la pile.

3.3 Illustrez l'utilisation des labels locaux synthétiques en créant la séquence :

```
LBL P          (pas LBL "P")
TONE 37        (apparaît à l'affichage comme TONE 7)
GTO P          (pas GTO "P")
```

3.4 Créez un label global synthétique de plus de 7 caractères. Par exemple LBL "RPN CALCULATOR".

3.5 Si vous ne possédez pas le PPC ROM, mais si vous avez le module X-Fonctions, voici une version de "LB" plus concise et plus rapide, également écrite par Clifford Stern. L'utilisation de "LBX" est identique à celle de "LB", et vous pouvez utiliser "LB" pour vous aider à taper "LBX". Les données à fournir à "LB" pour créer "LBX" pourront être trouvées après le sixième chapitre si vous avez des difficultés. Si vous prévoyez d'utiliser "LBX" régulièrement, vous le renommerez certainement "LB" et vous débarrasserez de la version originale de "LB".

```
01♦LBL 01
02 CLST
03 BEEP
04 STOP
05 GTO "++"

06♦LBL "LBX"
07 FS? 50
08 GTO 02
09 1
10 ENTER↑
11 ENTER↑
12 CLA
13 CF 21
14 AVIEW
15 -10
16 GTO "++"

17♦LBL 02
18 7
19 /
20 INT
21 FIX 0
22 CF 29
23 ARCL X
24 "⊢ REGS."
25 TONE 8
26 AVIEW
27 PSE
28 RCL b
29 "*"
30 X<> [
31 -2
32 AROT
33 RDN
34 STO \
35 ASHF
36 SIGN
37 ALENG
38 8
39 Y↑X
40 ATOX
41 *
42 512
43 MOD
44 ATOX
45 +
46 +
47 .1
48 %
49 +
50 +

51♦LBL 03
52 1.007
53 ENTER↑

54♦LBL 04
55 " "
56 ARCL Y
57 "⊢?"
58 AVIEW
59 STO [
60 RDN
61 STOP
62 FC?C 22
63 GTO 05
64 XTOA
65 X<> [
66 ISG Y
67 GTO 04
68 SIGN
69 X<> c
70 LASTX
71 STO IND T
72 X<>Y
73 STO c
74 R↑
75 DSE X
76 GTO 03
77 GTO 01

78♦LBL 05
79 "⊢♦"
80 ISG X
81 GTO 05
82 X<> c
83 RCL [
84 STO IND Z
85 X<>Y
86 STO c
87 GTO 01
88 END
```

```
LBL"LBX
END          160 BYTES
```

CHAPITRE QUATRE

ASSIGNEMENTS SYNTHETIQUES

4A. Programmes d'assignement

Les programmes chargeurs d'octets sont un grand pas en avant du point de vue du confort d'utilisation, par rapport au Byte Grabber. Les programmes d'assignement apportent encore plus de confort à l'utilisateur. Un tel programme peut assigner n'importe quelle instruction de un ou deux octets, qu'elle soit ou non synthétique, et ceci à n'importe quelle touche. Pour un confort d'utilisation maximal, vous pouvez assigner les fonctions synthétiques les plus couramment utilisées et vous servir de LB pour créer les autres instructions synthétiques dont vous avez besoin dans vos programmes.

Les programmes d'assignement sont similaires aux chargeurs d'octets, en cela qu'ils utilisent les codes décimaux pour construire des octets qui seront stockés dans la partie adéquate de la mémoire principale. Plutôt que d'introduire les codes décimaux un à un comme pour LB, vous chargerez les deux codes décimaux et le code de la touche à assigner dans la pile opérationnelle.

Les premiers programmes d'assignement ont été écrits par John McGechie au début de 1980. A cette époque, ils étaient réellement la consécration de "l'art" de la programmation synthétique.

Tout comme pour LB, trois programmes d'assignement différents sont décrits dans ce chapitre. Le premier, appelé "MK" (**M**ake **K**ey assignments : Faire des assignements de touches), nécessite seulement une HP-41 de base. Ce programme occupe seulement trois pistes de deux cartes magnétiques. Il a été écrit par Clifford Stern.

Le second programme d'assignement est **MK** contenu dans le PPC ROM, écrit par Roger Hill. **MK** est réellement un chef-d'oeuvre de la programmation synthétique et est pratiquement insensible aux erreurs d'utilisation. Si vous avez le PPC ROM, relisez le mode d'emploi de **MK** dans le manuel d'utilisation.

Le troisième programme, appelé "MKX", nécessite le module d'extension de fonctions. Ecrit par Tapani Tarvainen, il peut être stocké sur une seule carte magnétique. Il est plus concis et plus rapide que MK ou **MK**, et pardonne plus facilement les erreurs de l'utilisateur. Le listage de MKX est donné à la fin de ce chapitre, au problème 4.4 .

Bien que le programme MK de Clifford Stern soit assez court, il inclut bon nombre des commodités de **MK** du PPC ROM. Comme ce fut le cas pour LB et **LB**, toutes les subtilités de **MK** n'ont pu être incorporées dans MK ; cela l'aurait allongé sensiblement. Néanmoins, la détection d'erreur la plus importante, KEY TAKEN (Touche déjà prise), est présente. Un contrôle des erreurs par l'utilisateur plutôt que par le programme permet d'économiser de nombreux octets.

Si vous possédez un lecteur optique WAND, vous pourrez entrer MK ou MKX directement à partir des codes-barres fournis à l'appendice E. La première fois, il est peut-être préférable de s'habituer au fonctionnement de LB en entrant l'un de ces programmes avec son aide.

MK, qui ne réclame rien d'autre qu'une HP-41 de base, est listé ci-dessous et les équivalents décimaux nécessaires pour la création des instructions synthétiques à l'aide de LB sont également donnés. Après avoir créé ces instructions synthétiques, complétez le programme avec les instructions standard de la manière habituelle. Je vous rappelle que les

suffixes M, N, O, P, Q et ⊢ apparaissent respectivement comme [, \,], ↑, _ , et ⊤ sur les listages de l'imprimante bien que P et Q ne soient pas utilisés dans ce programme.

Remarquez que les lignes 11, 20 et 38 ne sont pas telles qu'elles apparaissent dans le listage. La ligne 20 est particulièrement trompeuse. Consultez la liste des données pour LB qui suit le programme afin de déterminer sa composition ainsi que celle des autres instructions synthétiques.

Données pour LB :

```
Ligne 09 : 206, 125      Ligne 11 : 241, 240*     Ligne 12 : 144,124
Ligne 17 : 206, 117      Ligne 19 : 145, 118
Ligne 20 : 247, 127, 42, 42, 42, 42, 42, 240*
Ligne 21 : 206, 118      Ligne 26 : 145, 124      Ligne 29 : 206, 119
Ligne 38 : 241, 240*     Ligne 40 : 145, 117      Ligne 50 : 145, 118
Ligne 52 : 206, 118      Ligne 53 : 27, 17        Ligne 64 : 240
Ligne 77 : 144, 118      Ligne 90 : 144, 127      Ligne 92 : 144, 122
Ligne 93 : 145, 118      Ligne 96 : 206, 118      Ligne 97 : 206, 126
Ligne 101 : 206, 126     Ligne 102 : 145, 118
Ligne 103 : 247, 127, 0, 0, 0, 42, 42, 42
Ligne 106 : 206, 119     Ligne 108 : 145, 127     Ligne 110 : 145, 122
Ligne 114 : 206, 125     Ligne 115 : 144, 118     Ligne 118 : 144, 118
Ligne 125 : 145, 125     Ligne 131 : 145, 118     Ligne 133 : 27, 20
Ligne 134 : 146, 118     Ligne 135 : 206, 118     Ligne 136 : 206, 126
Ligne 147 : 206, 126     Ligne 148 : 206, 117     Ligne 150 : 145, 118
Ligne 152 : 206, 118     Ligne 153 : 145, 117
```

* indique la présence d'un caractère invisible des lignes 8 à F dans une chaîne de caractères.

Assurez-vous bien que vous avez tapé MK correctement avant de l'utiliser. Tout comme avec LB, vous risquez le MEMORY LOST si le programme est tapé ou utilisé incorrectement. La théorie sur laquelle repose MK est beaucoup trop complexe pour être exposée ici. En fait, écrire un programme d'assignement en SIZE 000 (n'utilisant pas de registre de données) est un des défis les plus en vue de la programmation synthétique. Dans ce livre, nous nous limiterons à l'utilisation de MK.

Mode d'emploi du programme "MK" de Clifford Stern

1.) Si vous utilisez le module horloge, effacez toutes les alarmes. Toute alarme présente dans la machine lors de l'exécution de MK (ou de MK) sera altérée par la normalisation, la rendant ainsi inutilisable. Vous pourrez remettre vos alarmes lorsque vous aurez terminé de créer vos assignements synthétiques. La section 4E présente deux utilitaires qui sauvegardent automatiquement toutes les alarmes en mémoire étendue, et les rappellent ensuite. L'exécution de "SA" (**S**ave **A**larms : sauvegarde des alarmes) avant celle de MK effacera les alarmes après les avoir sauvegardées pour pouvoir les rappeler avec "RA" (**R**ecall **A**larms : rappel des alarmes). Les utilisateurs du PPC ROM doivent être mis en garde de toujours effacer les alarmes avant d'utiliser PK ou toute routine appelant LF (1K, +K, A?, ou F?). Cette restriction sur les alarmes ne s'applique pas au programme "MKX" (voir problème 4.4)

2.) Assurez-vous qu'un nombre suffisant de registres est disponible pour

les assignements avant d'exécuter MK. Le nombre de registres disponibles peut être contrôlé en tapant GTO .000 en mode PRGM. Le nombre d'assignements faisables avec MK est le double du nombre de registres libres, chaque registre contenant deux assignements. Le programme MK du PPC ROM est plus élaboré et peut détecter l'absence de registre disponible, affichant le message "NO ROOM" (plus de place).

3.) Exécutez MK pour initialiser le processus d'assignement. Le programme va chercher le premier registre d'assignement inutilisé afin de ne pas déranger les assignements déjà réalisés.

N'interrompez jamais l'exécution de MK (ou de MKX). Si vous interrompez MK, vous prenez le risque d'obtenir MEMORY LOST. Relancez MK **immédiatement** si vous l'avez interrompu. Si vous interrompez MKX, vous n'obtiendrez pas MEMORY LOST, mais risquez de perdre l'accès au CATalogue 1. C'est pourquoi vous devez relancer MKX **immédiatement et sans essayer de passer en mode PRGM.** Votre tentative pour entrer en mode PRGM pourrait vous faire sortir du programme MKX. Ceci vous contraindrait à faire MEMORY LOST pour reprendre le contrôle de la machine, à moins que vous ne retrouviez le contenu du registre d'état c dans la pile et que vous fassiez STO c. Ceci s'éclairera après la lecture du sixième chapitre.

4.) A la demande "PRE↑POST↑KEY", entrez les trois composantes de l'assignement -la valeur décimale de l'octet 1, ENTER↑, la valeur décimale de l'octet 2, ENTER↑, le code de la touche à assigner (ligne/colonne), R/S. Par exemple pour assigner RCL b à la touche 1/x, il vous faut entrer : 144, ENTER↑, 124, ENTER↑, 12, R/S. L'équivalent décimal du préfixe RCL est 144, celui du suffixe b est 124, et le code de la touche 1/x est 12 (ligne 1 colonne 2 position primaire (non shiftée)). Les deux premiers nombres décimaux doivent être des entiers compris entre 0 et 255, tandis que la troisième entrée doit être un code de touche valide. Un code de touche est un nombre décimal de la forme +-lc, où l est le numéro de ligne de la touche, c son numéro de colonne, et le signe est négatif si la touche est shiftée (position secondaire). Cette forme est la même que celle affichée momentanément par ASN, ou celle requise pour PASN du module X-fonctions. MK et MK vous permettent tous deux d'assigner la touche SHIFT shiftée (code de touche -31), mais MKX ne le permet pas. Si vous assignez une fonction à la touche SHIFT shiftée, choisissez de préférence une fonction réclamant un argument : cela permet d'éviter une exécution accidentelle.
Attention : Ne faites pas PACK, SIZE, ASN et n'éteignez pas la machine lorsque MK est arrêté en attente d'une donnée, à moins que vous n'ayiez terminé de l'utiliser. Ne modifiez pas non plus le contenu du registre alpha ou celui de LAST X.

5.) Lorsque le message "PRE↑POST↑KEY" réapparaît (avec l'indicateur du flag 2 allumé si vous utilisez MK), vous pouvez entrer les trois données nécessaires à l'assignement suivant. Cela complètera un registre d'assignement.

6.) Le message "PRE↑POST↑KEY" réapparaîtra de nouveau (cette fois sans l'indicateur 2 si vous utilisez MK), demandant l'entrée des données pour le premier assignement du registre libre suivant. Suivez les paragraphes 4 et 5 jusqu'à ce que vous ayez terminé tous les assignements désirés. Rappelez vous que vous ne pouvez utiliser plus de registres que le nombre que vous avez vérifié avant de lancer MK.

7.) Lorque vous avez terminé d'effectuer tous les assignements dont vous

```
01♦LBL "MK"
02 CLST
03 CF 02
04 CF 05
05 CF 06
06 CF 21
07 192
08 SIGN
09 X<> c
10 X<> Z
 11 ""
12 RCL b
13 RDN
14 X<> IND L
15 X=Y?
16 GTO 02
17 X<> [
18 "├*"
19 STO \
20 "├*****"
21 X<> \
22 X<> IND L
23 R↑
24 ISG L
25 -
26 STO b

27♦LBL 01
28 "├****"
29 X<> ]
30 "KEY TAKEN"
31 TONE 0
32 AVIEW
33 PSE

34♦LBL 16
35 "PRE↑POST↑KEY"
36 TONE 8
37 AVIEW
 38 ""
39 FS? 02
40 STO [
41 CLST
42 STOP
43 LASTX
44 XEQ 03
45 XEQ 03
46 R↑
47 X<0?
48 SF 05
49 ABS
50 STO \
51 R↑
52 X<> \
53  E1
54 MOD
55 X<>Y
56 LASTX
57 /
58 INT
59 4
60 DSE Z
61 X≠Y?
62 X=0?
63 ISG Z
64 ""
65 ST+ X
66 ENTER↑
67 R↑
68 *
69 ENTER↑
70 R↑
71 +
72 ST+ Y
73 RDN
74 FS? 05
75 +
76 R↑
77 RCL \
78 R↑
79 XEQ 03
80 X<> T
81 X<Y?
82 SF 06
83 36
84 -
85 FS? 06
86 +
87 R↑
88 SIGN
89 FS? 05
90 RCL e
91 FC? 05
92 RCL '
93 STO \
94 FS? 06
 95 "├*"
 96 X<> \
 97 X<> d
 98 FS? IND Z
 99 DSE Y
100 SF IND Z
101 X<> d
102 STO \
103 "├******"
104 FC?C 06
105 "├*"
106 X<> ]
107 FS? 05
108 STO e
109 FC?C 05
110 STO '
111 X<>Y
112 X=0?
113 GTO 01
114 X<> c
115 RCL \
116 FC? 02
117 "├***"
118 RCL \
119 STO IND L
120 FS?C 02
121 ISG L
122 SF 02

123♦LBL 02
124 X<> Z
125 STO c
126 X<>Y
127 GTO 16

128♦LBL 03
129 R↑
130 OCT
131 STO \
132 CLX
133  E4
134 ST+ \
135 X<> \
136 X<> d
137 FS?C 19
138 SF 20
139 FS?C 18
140 SF 19
141 FS?C 17
142 SF 18
143 FS? 15
144 SF 17
145 FS? 14
146 SF 16
147 X<> d
148 X<> [
149 "├**"
150 STO \
151 "├*"
152 X<> \
153 STO [
154 END
LBL'MK
END        313 BYTES
```

Le programme MK

avez besoin, ignorez la nouvelle demande de données, même si votre dernier assignement n'a pas fini de remplir un registre. Néanmoins, si vous terminez avec le flag 2 levé (pour MK seulement), vous perdez un demi-registre à moins que vous ne le remplissiez en utilisant la fonction standard ASN ou la fonction PASN du module X-Fonctions. A l'inverse de MK, ASN et PASN vérifient toujours qu'il n'existe pas de demi-registre inutilisé dans les registres d'assignement avant d'ouvrir un autre registre.

8.) Si vous essayez d'assigner une touche qui l'est déjà, le message "KEY TAKEN" (touche utilisée) apparaîtra à l'affichage. Vous avez alors deux possibilités. (Mais rappelez-vous de ne pas modifier ALPHA ou LAST X). La première consiste à effacer l'assignement se trouvant sur cette touche (ASN, ALPHA, ALPHA, touche), et réintroduire les données pour l'assignement puis R/S. La seconde consiste à réintroduire les deux équivalents décimaux et un nouveau code de touche.

Pour démontrer la puissance de MK, assignons les fonctions synthétiques suivantes :

STO b -11	STO d -12	STO M -13	STO N -14	STO O -15
RCL b 11	RCL d 12	RCL M 13	RCL N 14	RCL O 15
BG -21	X<> d -22	X<> M -23	X<> N -24	X<> O -25

La procédure est la suivante :
1) Effacer à la main les assignements de la première rangée, shiftée et non shiftée, et de la seconde rangée, non shiftée seulement.
2) Vérifier qu'au moins 8 registres (15 assignements à 2 par registre) sont disponibles en faisant GTO.000 en mode PRGM.
3) Sortir du mode PRGM et taper XEQ MK. Entrer les données comme suit :

Drapeau 2 (MK seulement)	Données pour MK, MK , et MKX
Baissé	145, 124, -11, R/S
Levé	144, 124, 11, R/S
Baissé	145, 126, -12, R/S
Levé	144, 126, 12, R/S
Baissé	145, 117, -13, R/S
Levé	144, 117, 13, R/S
Baissé	145, 118, -14, R/S
Levé	144, 118, 14, R/S
Baissé	145, 119, -15, R/S
Levé	144, 119, 15, R/S
Baissé	247, 63, -21, R/S
Levé	206, 126, -22, R/S
Baissé	206, 117, -23, R/S
Levé	206, 118, -24, R/S
Baissé	206, 119, -25, R/S
Levé	Flèche de correction ou rien pour terminer.

Ces fonctions synthétiques sont suffisantes pour créer environ les deux tiers des lignes synthétiques que l'on rencontre couramment. Ainsi, seulement un tiers des lignes synthétiques incluses dans LB et MK n'utilisent pas ces fonctions.

Il est également pratique d'avoir quelques fonctions standards

assignées. Parmi elles notons :

```
ASN X<>Y 21        (Pressez la touche X<>Y pour 21)
ASN RDN  22        (Pressez R↓ pour 22)
ASN SIZE 23        (Pressez SIN pour 23)
ASN PACK 24        (Pressez COS pour 24)
ASN DEL  25        (Pressez TAN pour 25).
```

Les deux premiers de ces assignements élimineront la recherche des labels locaux LBL F et LBL G lorsque vous presserez X<>Y ou RDN en mode USER. Cela accélèrera de façon significative la réponse de la touche dans de nombreux cas. Les autres fonctions sont celles dont il est agréable de disposer rapidement, le choix de la touche étant bien sûr une affaire de goût personnel. PACK et DEL sont très utiles pour l'emploi du Byte Grabber. Celui-ci et LB pouvant être utilisés pour créer n'importe quelle fonction synthétique que vous n'avez pas assignée à une touche.

Bien que l'on utilise généralement ASN pour assigner des fonctions non synthétiques, MK permet de les assigner tout comme des fonctions synthétiques. En réponse à la demande PRE↑POST↑KEY, entrez un seul nombre décimal compris entre 0 et 255, suivi par un code de touche. Pour X<>Y, l'équivalent décimal est 113; pour RDN, 117. Consultez la table des codes pour vérifier la correspondance. Pour les instructions multi-octets, c'est la même idée : DSE est codé 151 ; FC?C est 171 ; END est 192 ; GTO est 208 ; XEQ est 224 ; LBL est 207. Les fonctions non programmables utilisent des valeurs de la ligne 0 de la table. Par exemple, pour assigner SIZE, PACK et DEL en utilisant MK, vous introduiriez simplement les valeurs 6, 10 et 2, respectivement.

Si jamais vous devez assigner STO c ou X<> c à une touche vous devez l'effacer dès que vous avez terminé d'entrer le programme qui a nécessité l'assignement, ou alors vous devez être **très prudent.** Si jamais vous pressiez STO c ou X<> c accidentellement, cela vous conduirait infailliblement à MEMORY LOST.

Pour mon usage personnel, je trouve pratique d'avoir X<> c au clavier. Pour éviter les catastrophes, je l'ai assigné à une touche peu usitée, dans mon cas -21 (normalement CLΣ). Mon clavier redéfini ressemble à ceci :

Colonne :	1	2	3	4	5
Rang 1 shifté			STO M	STO N	STO b
Rang 1 non shifté			RCL M	RCL N	RCL b
Rang 2 shifté	X<> c	X<> d	X<> M	X<> N	X<> O
Rang 2 non shifté	X<>Y	RDN	"EFT"	eGOBEEP	BG
Rang 3		Pas d'assignement			
Rang 4 shifté				DEL	
Rang 5 shifté	PACK				
Rang 6 shifté	SIZE	XROM ▮1	INT	XTOA	
Rang 7 shifté	STO Q			X<>__	
Rang 8 shifté	Q-LOAD				

Je trouve que cet arrangement est facile à retenir et nécessite très peu de basculement en et hors du mode USER lorsque je tape un programme synthétique ou standard.

Les emplacements vides des rangs 1 et 4 permettent d'y placer des assignements temporaires de fonctions ou de programmes.

Au second rang, "EFT" est un programme décrit au problème 4.5. EFT

permet d'exécuter des fonctions du module horloge ou du module X-Fonctions en les appelant par un nombre.

La fonction eGOBEEP est un assignement d'un octet découvert par Robert Edelen. Utilisez les données 0 ENTER↑ 167 ENTER↑ code de touche R/S. Lorsque vous pressez la touche, vous voyez eGOBEEP__. Si vous introduisez un nombre décimal k allant de 0 à 63, vous obtenez XROM 28,k ce qui correspond à une des fonctions du lecteur de cassette. Si vous donnez k entre 64 et 99 vous obtiendrez XROM 29,k-64 ce qui permet de couvrir toutes les fonctions de l'imprimante. Par exemple, PRKEYS est XROM 29,12 , nous l'obtiendrons donc avec eGOBEEP76. La fonction PRP (**P**rint **P**rogram : Impression de programme) nécessite un argument alpha. Si vous tapez eGOBEEP 77, le calculateur ne vous demandera pas d'argument. Ce sera le contenu inversé du registre d'état **Q** qui sera utilisé comme argument, exactement comme pour le Q-Loader, qui est décrit dans les pages qui suivent.

Les assignements de EFT et eGOBEEP économisent beaucoup de temps lorsque vous avez appris les équivalents numériques des fonctions les plus employées. La liste complète des équivalents numériques pour l'utilisation de EFT et eGOBEEP est présentée à la fin de ce chapitre, avec le programme EFT au problème 4.5 .

Aussi sur le rang 2 se trouve le BG, qui demande les entrées décimales 247, 63 plus un code de touche. Au sixième rang, XROM **T1** est une fonction du PPC ROM consistant en une suite de TONEs synthétiques rapides. C'est une alternative plaisante de BEEP, utilisant un octet de plus en mémoire programme. XTQA est un assignement du module X-Fonctions. Son utilité apparaîtra dans la prochaine section.

4B. Le "chargeur d'octet du pauvre"

Les deux derniers assignements du clavier précédent, STO Q et Q-LOAD nécessitent des explications complémentaires. Ainsi que quelques programmes, ces assignements constituent un "chargeur d'octets du pauvre". Assignez ces fonctions aux touches que vous voulez en utilisant MK. Les valeurs décimales à entrer sont 145, 121 pour STO Q et 27, 0 pour Q-LOAD. Vous aurez également besoin du Byte Grabber et de RCL M que vous devez déjà avoir assignés au clavier.

Si vous êtes assez fortuné pour posséder un module X-Fonctions, XTOA vous servira beaucoup comme constructeur d'octets. Si vous avez un PPC ROM, vous pourrez utiliser **DC**. Ces fonctions prennent une valeur décimale comprise entre 0 et 255 dans le registre X et créent l'octet correspondant sous forme de caractère, qui est ajouté au contenu de ALPHA (c'est-à-dire qu'il devient le dernier octet du registre M). Si vous n'avez aucun de ces deux modules, tapez cette courte routine synthétique pour remplacer ces fonctions.

Entrées pour LB :

```
Ligne 03 : 27,  20  Ligne 05 : 206, 126 Ligne 16 : 206, 126
Ligne : 17 206, 117 Ligne 19 : 145, 118 Ligne 22 : 206, 118
Ligne 23 : 145, 117
```

Remarquez que c'est la routine de base de construction d'octets que Clifford Stern a écrite pour ses programmes MK et LB.

Utilisez ASN pour assigner XTOA, **DC**, ou DC, celui que vous utilisez, à une touche adéquate. Nous sommes maintenant prêts. La fonction Q-LOAD crée une chaîne de 7 caractères à partir du contenu inversé du registre d'état Q. Par exemple, pour créer la chaîne "HP'S #1", nous créerons tout d'abord

```
01♦LBL "DC"
02 OCT
03  E4
04 +
05 X<> d
06 FS?C 19
07 SF 20
08 FS?C 18
09 SF 19
10 FS?C 17
11 SF 18
12 FS? 15
13 SF 17
14 FS? 14
15 SF 16
16 X<> d
17 X<> [
18 "┣**"
19 STO \
20 "┣*"
21 CLX
22 X<> \
23 STO [
24 RDN
25 END
LBL'DC
END          54 BYTES
```

```
01♦LBL "RAMBC"
   02 X<>Y
   03 XEQ 01
   04 X<>Y
   05 XEQ 01
   06 -
   07 RTN

08♦LBL "RAMBYT"
   09♦LBL 01
   10 XEQ 03
   11  E41
   12 /
   13 INT
   14 LASTX
   15 FRC
   16  E4
   17 *
   18 DEC
   19 7
   20 *
   21 +
   22 RTN

23♦LBL "ROMBC"
   24 XEQ 02
   25 X<>Y
   26 XEQ 02
   27 -
   28 RTN

29♦LBL "ROMBYT"
   30♦LBL 02
   31 XEQ 03
   32  E37
   33 /
   34 DEC
   35 RTN

   36♦LBL 03
   37 "*"
   38 X<> [
   39 STO \
   40 ASHF
   41 "┣♦♦♦A"
   42 X<> [
   43 X<> d
   44 CF 08
   45 FS?C 09
   46 SF 05
   47 FS?C 10
   48 SF 06
   49 FS?C 11
   50 SF 07
   51 FS?C 12
   52 SF 09
   53 FS?C 13
   54 SF 10
   55 FS?C 14
   56 SF 11
   57 FS?C 15
   58 SF 13
   59 FS?C 16
   60 SF 14
   61 FS?C 17
   62 SF 15
   63 FS?C 18
   64 SF 17
   65 FS?C 19
   66 SF 18
   67 FS?C 20
   68 SF 19
   69 X<> d
   70 END
LBL'RAMBC
LBL'RAMBYT
LBL'ROMBC
LBL'ROMBYT
END          159 BYTES
```

la chaîne "1# S'PH" dans le registre ALPHA, utiliserons RCL M pour la mettre en X, puis le transférerons par STO Q dans le registre Q, puis presserons la touche Q-load. Essayons :

```
CLA
49 XTOA        (Utilisez DC ou DC si vous n'avez pas XTOA.
35 XTOA        Certains de ces caractères ne sont pas
32 XTOA        synthétiques et peuvent être ajoutés directement,
83 XTOA        mais cela n'en vaut sans doute pas la peine.)
39 XTOA
80 XTOA
72 XTOA
```

Arrivé à ce point, vous avez la chaîne "1# S'PH" dans le registre ALPHA. Trouvez maintenant un emplacement dans la mémoire où vous voudriez insérer la chaîne "HP'S #1". Si vous n'y êtes pas déjà, faites GTO.. et utilisez la fin de la mémoire. Lorsque vous êtes à l'endroit ad hoc, sortez du mode PRGM et utilisez les assignements pour faire RCL M, STO Q. Revenez maintenant en mode PRGM et pressez la touche Q-LOAD. Vous allez voir une entrée de donnée synthétique E, provenant de la valeur 27 de l'assignement de Q-LOAD (voir table des codes, ligne 1 colonne B). Pressez SST une fois et voyez la chaîne "HP'S #1". Pressez de nouveau Q-LOAD, vous obtenez les deux instructions synthétiques E et TEXT 0. La première utilisation du Q-Loader (chargeur de Q) a effacé le contenu de Q. La seconde fois, vous avez donc obtenu une chaîne vide. Donc, en plus de la création de chaînes de caractères synthétiques, l'assignement Q-LOAD permet d'obtenir facilement l'entrée numérique E et l'instruction TEXT 0 (NOP).

Mais le Q-Loader révèle toute sa puissance lorsqu'il est utilisé en paralléle avec le Byte Grabber. Vous pouvez utiliser le Q-Loader pour créer des chaînes de caractères, puis dérober et détruire le préfixe indiquant qu'il s'agit d'une chaîne, permettant ainsi aux caractères de se transformer en instructions. L'exemple qui suit, relativement long, illustrera la puissance de ce "chargeur d'octets du pauvre". Suivez son fonctionnement deux ou trois fois jusqu'à ce que vous compreniez son principe et les techniques utilisées.

Dans cet exemple, nous allons créer les instructions synthétiques contenues dans le programme CMOD du problème 2.6 . Ces instructions sont STO M, ST- M, ST/ M et x<> M. Leurs équivalents décimaux sont respectivement 145, 117 ; 147, 117 ; 149, 117 ; 206, 117. Nous commençons par le dernier octet et remontons vers le premier :

```
CLA
206 XTOA
117 XTOA
149 XTOA
117 XTOA
147 XTOA
117 XTOA
117 XTOA
```

Le premier groupe de 7 octets est maintenant prêt à être chargé en mémoire programme. GTO.. et tapez LBL "CMOD" comme en tête. Quittez le mode PRGM, RCL M et STO Q. Revenez en mode PRGM et pressez la touche Q-LOAD. Vous allez voir l'instruction E maintenant bien connue. Ne faites pas SST; pressez plutôt la touche du Byte Grabber. Cela enlève le préfixe text de la chaîne chargée avec le Q-Loader. Appuyez deux fois sur la touche de

correction pour effacer l'octet dérobé et l'instruction E. Vous avez maintenant :

```
01 LBL "CMOD"
02 RDN
03 ST- M
04 ST/ M
05 X<> M
.END.
```

Il reste à charger l'octet STO. Quittez le mode PRGM et tapez :

```
CLA
145 XTOA
```

Faites maintenant GTO "CMOD", RCL M, STO Q, passez en mode PRGM et pressez Q-LOAD. PACK pour enlever les nuls invisibles entre la chaîne nouvellement chargée et les sept octets que nous avions déjà chargés. Toujours sur l'instruction E, pressez BG et deux fois la flèche de correction. Listez le programme avec SST, vous devez voir :

```
01 LBL "CMOD"
02 STO M
03 ST- M
04 ST/ M
05 X<> M
.END.
```

L'octet STO a été chargé dans la chaîne de caractères. Dès qu'il a été libéré, il a absorbé l'octet RDN qui est devenu un suffixe M.

Avec un peu d'entraînement, ce chargeur d'octets du pauvre peut être utilisé pour créer rapidement des instructions synthétiques avec un minimum de préparation. Les seules choses nécessaires sont les assignements de RCL M, STO Q, Q-LOAD et BG, plus un module d'extension de fonctions ou un PPC ROM ou le programme DC, et bien sûr la table des codes.

Il est préférable de ne pas créer des morceaux d'instructions avec le Q-Loader tel que nous l'avons fait dans le premier groupe de sept octets de l'exemple précédent. Il aurait été judicieux de s'arrêter au sixième octet, créant ainsi trois instructions, puis reprendre les deux octets restants pour un second chargement. Cela élimine le temps perdu dans le PACKING. La procédure nécessitant PACK a été illustrée ici car elle est indispensable pour créer des instructions synthétiques longues de plus de sept octets.

La seule limitation de la procédure utilisant le Q-Loader est que les nuls de queue sont supprimés. Donc si vous voulez créer l'instruction F2 7F 00 hex (APPEND un nul), vous devez ajouter une instruction "bidon" tel ENTER↑. Pour cet exemple, la procédure à suivre est donc : CLA, 131 (l'instruction ENTER↑), XTOA, 0 (le nul), XTOA, 127 (append), XTOA, 242 (le préfixe TEXT 2), XTOA, placez-vous à l'endroit désiré, passez en mode RUN, RCL M, STO Q, passez en mode PRGM, Q-LOAD, BG, et deux fois la flèche de correction. Vous devrez alors vous débarrasser de l'instruction ENTER↑ suivant votre nouvelle instruction synthétique. Si l'octet bidon 131 n'avait pas été ajouté, les pas 0, XTOA n'auraient rien fait et vous auriez seulement chargé les deux octets 242, 127.

4C. Préaffichage des pseudo-XROMs

Les seules fonctions non synthétiques de deux octets qui soient assignables à des touches sont les fonctions des périphériques. Lorsque le périphérique correspondant n'est pas connecté, la fonction apparaît comme XROM i,j lorsque la touche est maintenue enfoncée, où i et j sont deux nombres décimaux compris entre 00 et 63. La notation XROM dénote que la fonction assignée réside dans une mémoire morte externe (e**X**ternal **R**ead **O**nly **M**emory). Le nombre i identifie le périphérique, c'est pourquoi i est appelé le numéro d'ID. Certains périph2riques contiennent deux ROMs de 4 kilo-octets chacune, toutes deux ayant leur propre numéro d'ID. Le nombre j donne le numéro d'ordre de la fonction (dans l'ordre du CATalogue 2) à l'intérieur de la ROM de 4 kilo-octets.

Lorsqu'une touche à laquelle est assignée une fonction synthétique de deux octets est pressée, la HP-41 pense que l'assignement est une fonction XROM ordinaire. Si les valeurs décimales des octets d'un assignement sont x et y, les nombres i et j apparaissant dans l'XROM i,j sont :

$$i = 4(x \text{ MOD } 16)+\text{INT}(y/64) \text{ et}$$
$$j = y \text{ MOD } 64 .$$

où MOD représente la fonction modulo (voir MOD dans le manuel de votre HP-41). Par exemple ST+ IND M = 146, 245 apparaît comme XROM 11,53 et TONE Y = 159,114 apparaît comme XROM 61,50. Voir la table des codes pour la correspondance. Le numéro de la colonne du premier octet x est, en fait, x MOD 16. Cela permet à i de prendre 4 valeurs, que l'on trouve à la ligne A de la table, au moins pour les colonnes 0 à 7. Par exemple, ST+ est en colonne 2. En consultant la deuxième colonne de la ligne A, nous trouvons XR8-11, indiquant que le premier des nombres de l'XROM sera 8, 9, 10 ou 11.

La valeur exacte de i est déterminée par le bloc de 4 lignes auquel appartient le second octet y. Les lignes horizontales les plus grasses de la table vous aident à voir les limites des blocs. Les lignes 0 à 3 correspondent à la première valeur possible de i, les lignes 4 à 7, à la seconde, les lignes 8 à B, à la troisième et les lignes C à F, à la quatrième. Si vous déplacez visuellement le second octet jusqu'à la case correspondante dans les lignes 0 à 3, ce qui revient à prendre sa valeur modulo 64, vous pourrez lire directement la valeur de j à la deuxième ligne de la case.

Continuons avec l'exemple de ST+ IND M. Comme le suffixe IND M appartient au quatrième groupe de 4 lignes, la valeur de i est 11. Puis nous translatons par la pensée, le suffixe de la colonne 5 ligne F à la ligne 3 colonne 5, qui est la position correspondante dans le premier groupe de 4 lignes. En regardant l'équivalent décimal de cette case, nous obtenons pour j la valeur 53. Donc ST+ IND M apparaît comme XROM 11,53 .

Les nombres arguments de l'XROM révèlent beaucoup de choses sur la fonction synthétique assignée, mais ne la déterminent pas de façon unique. Par exemple, un assignement de DSE IND 10 s'affiche XROM 30,10 ou WSTS si le lecteur de cartes est présent. Cet assignement ne peut pas être distingué de WSTS jusqu'à ce que la touche soit relâchée. Si vous doutez de l'identité d'un assignement, essayez-le tout d'abord en mode PRGM. Mais, au cas où ce serait le Byte Grabber, ne pressez pas cette touche près du .END. ou d'un END. Rappelez vous les contraintes de l'utilisation du BG décrites au premier chapitre !

4D. L'assignement RCL b

La fonction RCL b est un cas unique parmi les fonctions synthétiques assignables. Contrairement aux autres assignements, qui ne sont pas indispensables si l'on utilise LB, l'assignement de RCL b est plus puissant qu'une instruction RCL b chargée en mémoire programme. Exécutée au clavier, RCL b retourne l'adresse courante du pointeur programme dans le registre X. Exécutée dans un programme, le résultat serait toujours le même, c'est-à-dire l'adresse de l'instruction RCL b en mémoire.

Le résultat retourné par une instruction RCL b est un pointeur codé sur les deux derniers octets du registre X, exprimé sous la forme de quatre chiffres hexadécimaux. Sous sa forme codée, ce pointeur n'est pas spécialement utile. Deux programmes sont présentés ici, qui convertissent le pointeur retourné dans X en un nombre décimal d'octets. Deux autres programmes permettent de déterminer facilement le nombre d'octets entre deux emplacements de la mémoire programme.

Le programme RAMBYT fait exactement la même chose que la routine **PD** du PPC ROM. Pour l'utiliser, placez-vous n'importe où en mémoire à l'aide de CAT 1 et pressez RCL b en mode RUN. Le résultat est un pointeur représentant la position où vous vous trouvez. Exécutez RAMBYT (ou **PD**) pour convertir ce pointeur en sa représentation décimale.

Le programme ROMBYT est semblable à RAMBYT, excepté le fait que le pointeur doit être en ROM. Si vous avez le PPC ROM ou tout autre ROM en langage utilisateur (module d'application), vous pouvez essayer ROMBYT. Positionnez-vous sur un label ou n'importe où ailleurs dans la ROM, pressez RCL b en mode RUN, puis XEQ ROMBYT pour voir le nombre décimal représentant l'adresse du pointeur correspondant.

Les applications les plus courantes des pointeurs programmes sont le comptage du nombre d'octets entre deux instructions de programme. Par exemple, vous pouvez désirer connaître le nombre total d'octets d'un programme. Le programme RAMBC détermine la distance entre deux pointeurs programmes en utilisant RAMBYT pour décoder chaque pointeur, et en soustrayant les deux résultats décimaux. RAMBC a le même usage que la routine **CB** (**C**ount **B**ytes : compteur d'octets) du PPC ROM.

Pour illustrer le fonctionnement de RAMBC, comptons combien d'octets occupe le groupe de routines RAMBC/RAMBYT/ROMBC/ROMBYT. Compactez la mémoire si ce n'est déjà fait. Positionnez-vous sur LBL RAMBC, RCL b en mode RUN, BST (pour venir sur le END), RCL b en mode RUN, et XEQ RAMBC. Le résultat devrait être 156, indiquant que le programme est long de 156 octets, du début de LBL RAMBC jusqu'au début du END. Si vous voulez inclure le END dans votre compte, ajoutez 3 pour obtenir 159. Si la dernière ligne du programme RAMBC est .END. , votre compte peut être augmenté jusqu'à 6 octets de plus. Dans ce cas, vous pouvez faire GTO.. et répéter la séquence ci-dessus pour obtenir le bon compte d'octets.

Divisez par 112 pour savoir combien de pistes de cartes magnétiques le programme nécessitera pour être enregistré. Le END est enregistré sur la carte, mais si vous avez un programme qui fait 112 octets sans le END, vous n'aurez pas besoin de lire la seconde piste. Dans un cas comme celui-ci, vous pouvez effacer le message vous réclamant la seconde piste, tant à la lecture qu'à l'écriture, la seule chose se trouvant sur cette seconde piste étant le END qui n'apporte pas d'information supplémentaire.

Une utilisation plus 'à la pointe' de RAMBC consiste à déterminer si une GTO long (de trois octets) est nécessaire, ou si un GTO court (deux octets) suffira. Les GTOs courts (GTO 00 à GTO 14) doivent seulement être utilisés si la distance de saut est inférieure à 112 octets. Cela permet la compilation du saut lors de la première exécution (la longueur du saut est

alors stockée dans l'instruction elle-même). Les exécutions suivantes seront beaucoup plus rapides, la recherche du label étant alors inutile donc évitée. Seuls les GTOs longs peuvent stocker des distances de saut supérieures à 112 octets, aussi si vous utilisez un label court pour une distance trop longue, la vitesse d'exécution de votre programme sera notablement plus faible, car la recherche se fera à chaque passage.

Afin de déterminer si un GTO sur deux octets et le label sur un octet lui correspondant peuvent être utilisés sans perdre l'avantage de la compilation du saut, entrez le GTO et le LBL aux emplacements souhaités dans le programme. Employez LBL nn et GTO nn, où nn est compris entre 00 et 14 inclus. PACKez pour éliminer les nuls indésirables. Positionnez-vous sur la ligne suivant l'instruction GTO (si cela s'avérait être le .END., insérez une instruction 'bidon' et PACKez de nouveau) puis pressez RCL b en mode RUN. Placez-vous ensuite sur l'instruction LBL (vous pouvez utiliser BST, SST) et pressez RCL b encore une fois. XEQ RAMCB pour connaître la longueur du saut en octets. Si celle-ci est comprise entre -111 et +111 octets, bornes comprises, alors le GTO court est suffisant. Dans le cas contraire, vous devez le remplacer par un GTO long.

Une autre méthode consiste à faire RCL b sur l'instruction GTO, SST pour se positionner sur le LBL, RCL b et XEQ RAMBC. Le résultat doit être compris entre -109 et +113, bornes comprises.

Si vous avez besoin d'un GTO long, vous pouvez en construire un synthétique en introduisant les valeurs 208, 0, nn pour LB, où nn est compris entre 00 et 14. Ou alors vous pouvez entrer la séquence STO IND 80, ISG nn, deux fois BST, BG et flèche de correction pour subtiliser l'octet STO. De toute façon, cela vous permet d'utiliser un LBL nn sur un octet, économisant ainsi un octet par rapport aux instructions standards GTO xx, LBL xx avec xx entre 15 et 99. Une fois créé, un GTO synthétique sur trois octets ne se transformera jamais en un GTO de deux octets, et compilera toujours correctement la distance du saut. Il peut être distingué d'un GTO sur deux octets en utilisant RAMBC pour déterminer sa longueur en octets.

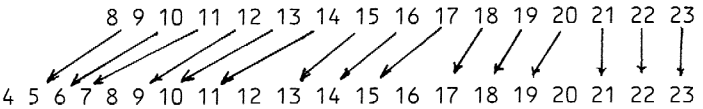
Voici les listages des routines RAMBC, RAMBYT, ROMBC et ROMBYT. ROMBC est bien sûr identique à RAMBC, excepté le fait qu'elle opère sur des pointeurs en ROM (p. 55).

Entrées pour LB :

```
Ligne 11 :  27,  20,  17 Ligne 16 :  27,  20      Ligne 32 :  27, 19, 23
Ligne 38 : 206, 117      Ligne 39 : 145, 118
Ligne 41 : 245, 127, 0, 0, 0, 65
Ligne 42 : 206, 117      Ligne 43 : 206, 126      Ligne 69 : 206, 126
```

Le centre de ce groupe de routines est le sous-programme au LBL 03, qui utilise deux trucs de programmation synthétique avancée. Les quatre premiers pas isolent les deux derniers octets de X dans le registre ALPHA. Ces octets sont alors décalés vers la gauche (ligne 41) et transférés dans le registre des drapeaux. A cet endroit, les 15 bits du pointeur (le bit le plus à gauche n'est pas nécessaire ici) se trouvent être les drapeaux 9 à 23. Les opérations sur les flags sont utilisées pour transformer ces bits en octal (base 8), avec 3 bits par chiffre octal (voir ci-dessous). Cela laisse cinq chiffres octaux dans les flags 4 à 23, les flags 4, 8, 12, 16 et 20 baissés. Ces cinq chiffres octaux sont extraits du registre des drapeaux sous la forme a.bcde 10^{41}. Les opérations arithmétiques courantes peuvent maintenant être utilisées pour séparer les chiffres si nécessaire, après quoi la fonction DEC peut convertir ces chiffres en décimal. Cette astuce consistant à transformer les bits en octal et à les reconvertir en décimal fut découverte par Roger Hill, l'auteur de nombreuses routines du

PPC ROM.



Vous devrez lire le développement sur le format des pointeurs programmes au sixième chapitre pour comprendre la manipulation des chiffres en octal dans les routines RAMBYT et ROMBYT.

4E. Sauvegarde et rappel des alarmes **PPC ROM Nécessaire**

La plupart des programmes d'assignement (excepté MKX -- voir problème 4.4) possèdent une caractéristique commune : ils ne fonctionnent pas correctement si des alarmes sont enregistrées, et ils perturbent également ces alarmes. Une solution consiste à effacer manuellement les alarmes en utilisant la fonction ALMCAT du module horloge. C'est fastidieux et cela nécessite de noter les alarmes pour les réintroduire plus tard.

Si vous avez un module X-Fonctions et un PPC ROM, vous pouvez utiliser les programmes SA (**S**ave **A**larms) et RA (**R**ecall **A**larms) de Clifford Stern pour transférer automatiquement les alarmes en mémoire étendue, puis les rappeler en mémoire principale lorsque vous avez terminé vos assignements. SA utilise la fonction SAVERX du module X-Fonctions qui, contrairement à RCL, permet d'éviter la normalisation des données (voir la section 2C sur la normalisation). En fait, le premier et le dernier registre du bloc des alarmes sont normalisés, mais ceci est rattrapé par RA.

Voici le mode d'emploi des programmes SA et RA :

1) Assurez-vous qu'il y ait au moins un END quelque part avant LBL "SA" dans le CAT 1. Ceci est nécessaire pour permettre le retour en arrière de la ligne 66 avec le rideau baissé. Ceci sera expliqué à la section 6C.
2) Après cette vérification, XEQ "SA" pour sauvegarder les alarmes en X-Mémoire dans un fichier nommé ALM et pour effacer les alarmes de la mémoire principale. Le message DATA ERROR de la ligne 86 apparaît s'il n'y a pas d'alarme présente en mémoire. DUP FL apparaît si le fichier ALM existe déjà en mémoire étendue. Exécutez alors PURFL, puis pressez R/S pour continuer le déroulement de SA. Le message NO ROOM apparaît s'il n'y a pas suffisamment de registres inutilisés en X-Mémoire pour y stocker les alarmes. Vous pouvez alors détruire des fichiers en X-Mémoire et replacer "ALM" dans le registre ALPHA avant de relancer l'exécution.
3) Utilisez le programme d'assignement que vous préférez. Lorsque vous en avez terminé avec lui, XEQ "RA" restaurera vos alarmes et effacera le fichier ALM de la mémoire étendue. Le programme RA utilise la fonction PSIZE (Programmable SIZE) du module X-Fonctions si cela s'avère nécessaire, pour ouvrir les registres voulus sous le .END. afin d'y stocker les alarmes. Si le nombre total de registres disponibles, y compris les registres de données, est insuffisant, vous verrez apparaître le message DATA ERROR à la ligne 15. Si cela arrivait, PACKez ou effacez un programme de la mémoire principale et refaites XEQ "RA". RA se termine par une instruction OFF, nécessitant que vous rallumiez la machine après usage. Cette instruction OFF est utile dans le cas où vous auriez éteint le calculateur après avoir exécuté SA mais avant d'avoir lancé RA. Le module horloge, n'ayant pas trouvé d'alarme à la dernière extinction, n'a pas activé le décompte du temps jusqu'à l'alarme suivante. L'instruction OFF

```
 01♦LBL "RA"          22 FLSIZE       42 XROM "E?"   62 X<> \       81 ENTER↑
02 XROM "F?"          23 ""           43 17          63 STO IND L    82 DSE X
 03 INT               24 RCL [        44 -           64 RDN          83 ATOX
04 XROM "E?" 25 "                    45 X<Y?        65 ISG L        84 "ALM"
 05 X<>Y              26 STO \        46 GTO 03      66 GTO 01       85 CF 25
 06 -                 27 ARCL 00      47  E3         67 CLA          86 CRFLD
 07 SIZE?             28 RCL [        48 /           68 GTO 03       87 +
 08 ENTER↑            29 STO 00       49 +                           88  E3
 09 "ALM"             30 X<> \        50 SIGN        69♦LBL 02       89 /
 10 LASTX             31 DSE Z        51 "0"        70 ARCL IND L    90 +
 11 +                32 STO IND Z     52 X<> [       71 X=0?         91 X<>Y
 12 FLSIZE            33 R↑                          72 CLA          92 X<> c
 13 -                 34 STO c        53♦LBL 01       73 X<> [        93 X<>Y
 14 X<0?              35 "ALM"         54 ""         74 STO IND L     94 SAVERX
 15 SQRT              36 PURFL        55 RCL IND L                   95 XROM "BC"
 16 X<Y?              37 BEEP         56 X<> [       75♦LBL 03        96 X<>Y
 17 PSIZE             38 OFF 57 "├                  76 ATOX          97 STO c
 18 R↑                                 ¡ "         77 R↑            98 BEEP
19 XROM "CX"                         58 X<> \       78 X<> c         99 END
 20 GETR             39♦LBL "SA"     59 X≠Y?        79 LASTX  LBL'RA
 21 R↑               40 XROM "OM"    60 GTO 02      80 INT    LBL'SA
                      41 176         61 ARCL c                END        175 BYTES
```

```
01♦LBL "MKX"       12 STO ]     23 .          33 X≠Y?       44 FC?C 25
 02 "ANUM"         13 X<> [     24 SIGN       34 X<> \      45 ISG L
 03 CF 25    14 "├♦♦♦      B"                 35 X=Y?       46 X=Y?
 04 PASN           15 X<> ]     25♦LBL 01     36 SF 25      47 GTO 01
  05 "xiμ♦"        16 X<> [    26 X<> IND L   37 X=Y?       48 R↑
 06 RCL [          17 STO \     27 X<> [      38 R↑         49 STO c
 07 R↑              18 "├"      28 "├*"       39 "├****"    50 CLST
 08 XTOA           19 X<> ]     29 STO \      40 STO ]      51 END
 09 R↑             20 R↑        30 "├***"      41 "├**"
 10 XTOA           21 X<> c     31 X<> \      42 X<> ]    LBL'MKX
 11 RCL '          22 RCL \     32 "├***"     43 STO IND L END        123 BYTES
```

```
01♦LBL "EFT"     08 CLX       15 SF 25      22 RDN
 02 RCL [        09 64         16 "├┼t|"     23 FS?C 25
 03 CLA          10 +         17 RDN         24 STOP
 04 STO [        11 RCL [     18 X<> [       25 SF 30
 05 AON      12 "pTμu      "  19 X<> a       26 END
 06 PSE          13 X<>Y      20 X<> \     LBL'EFT
 07 AOFF         14 XTOA      21 X<> b     END        58 BYTES
```

active le compte à rebours jusqu'à activation de l'alarme suivante, et permet de vous mettre en garde des alarmes périmées (past-due alarms). Une instruction CLOCK aurait le même effet. Pour un usage comme sous-programme, vous pouvez remplacer OFF par RTN si vous gardez en tête le fait que si le calculateur est éteint pendant que les alarmes sont sauvegardées, le compteur à rebours ne sera pas activé.

Données pour LB :

Ligne 23 : 241, 240*	Ligne 24 : 144, 117	Ligne 25 : 241, 170*
Ligne 26 : 145, 118	Ligne 28 : 144, 117	Ligne 30 : 206, 118
Ligne 34 : 145, 125	Ligne 47 : 27, 19	Ligne 51 : 241, 16
Ligne 52 : 206, 117	Ligne 54 : 241, 240*	Ligne 56 : 206, 117
Ligne 57 : 242, 127, 170*		
Ligne 58 : 206, 117	Ligne 61 : 155, 125	Ligne 62 : 206, 118
Ligne 73 : 206, 117	Ligne 78 : 206, 125	Ligne 88 : 27, 19
Ligne 92 : 206, 125	Ligne 97 : 145, 125	

* indique un caractère invisible des lignes 8 à F de la table des codes (valeurs décimales 128 à 255).

Remarquez que les lignes 25 et 57 contiennent le caractère AA_{16} (décimal 170), qui est un caractère de contrôle de l'imprimante faisant sauter 10 espaces. Ces caractères de contrôle, dont nous avons parlé à la fin de la section 2E, donnent parfois des listages assez curieux. Les caractères ombrés des lignes A à E de la table des codes sont des codes de contrôle de l'imprimante.

Problèmes :

4.1 Relisez les solutions des problèmes du deuxième chapitre et estimez le gain de temps obtenu en utilisant des assignements synthétiques pour taper ces programmes.

4.2 Essayez de taper le programme LB de Clifford Stern en utilisant le 'chargeur d'octets du pauvre' pour créer les instructions suivantes :

```
hex F4 7F 00 00 02
E4
X<> c
STO c
hex F2 7F 00
X<> c
STO c
```

Tapez les intructions synthétiques manquantes en utilisant votre clavier assigné. Vous pourrez alors compléter le programme avec les instructions standards.

4.3 Calculez les paramètres d'un XROM des assignements suivants, et vérifiez vos prédictions :

```
TONE 89
X<> P
ISG IND N
```

4.4 Voici le programme d'assignement MKX, écrit par Tapani Tarvainen, revu

et optimisé par Clifford Stern, qui nécessite le module X-Fonctions. Il emploie une technique tout à fait différente, rendue possible par la présence de la fonction PASN (assignement programmable). En simplifiant, MKX utilise PASN pour créer un assignement 'bidon' sur la touche voulue, puis le retrouve dans les registres d'assignement et le remplace par l'assignement désiré. MKX est suffisamment différent de MK et MK pour justifier la liste d'instructions qui suit :

1) Assurez-vous que vous n'avez pas de LBL "ANUM" apparaissant au CATalogue 1. Si vous exécutez MKX avec un tel label, un octet F0 remplacera le premier octet (le plus à gauche) de tous les registres de la mémoire programme, y compris dans les programmes et les registres de données. Ceci est virtuellement la même chose que MEMORY LOST, car vous déciderez certainement d'en venir à cette extrêmité, plutôt que d'essayer de remettre en ordre ce bazar.
2) Chargez les trois entrées dans la pile et pressez XEQ "MKX". Les trois données à fournir à MKX sont identiques à celles pour MK ou MK. La différence réside dans le fait que vous devez placer les deux valeurs décimales et le code de la touche (dans Z, Y et X comme pour MK) **avant** d'exécuter MKX.
3) Les alarmes ne nécessitent pas d'être sauvegardées ou effacées. Elles ne seront pas altérées.
4) Si vous n'avez pas suffisamment de registres disponibles, vous obtiendrez PACKING, TRY AGAIN à la ligne 04. Le programme pardonne plus facilement les erreurs que MK.
5) Tout comme MK, MKX **n'est pas interruptible.**
6) Si vous essayez d'assigner une touche qui l'est déjà, le nouvel assignement remplacera l'ancien, sans vous signaler ce fait. Si vous voulez éviter ce genre de problème, contrôlez la touche avant d'exécuter MKX.
7) Pour assigner une autre touche, placez les trois nouvelles données dans la pile et relancez MKX ou pressez R/S, le dernier assignement vous ayant laissé au début du programme.
8) On ne perd pas de demi-registre avec MKX. Chaque assignement est traité de la même façon et un nouveau registre n'est ouvert que s'il n'y a pas de trou dans les registres d'assignement (cf. p. 62).

Entrées pour LB :

```
Ligne 05 : 245, 1, 105, 12, 0, 240*
Ligne 06 : 144, 117      Ligne 11 : 144, 122      Ligne 12 : 145, 119
Ligne 13 : 206, 117
Ligne 14 : 247, 127, 0, 0, 0, 240*, 166*, 66
Ligne 15 : 206, 119      Ligne 16 : 206, 117      Ligne 17 : 145, 118
Ligne 18 : 242, 127, 240*
Ligne 19 : 206, 119      Ligne 21 : 206, 125      Ligne 22 : 144, 118
Ligne 27 : 206, 117      Ligne 29 : 145, 118      Ligne 31 : 206, 118
Ligne 34 : 206, 118      Ligne 39 : 245, 127, 42, 42, 42, 0
Ligne 40 : 145, 119      Ligne 41 : 244, 127, 0, 0, 240*
Ligne 42 : 206, 119      Ligne 49 : 145, 125.
```

* indique un caractère de la seconde moitié de la table des codes, normalement invisible dans les listages, mais apparaissant comme une nova à l'affichage.

4.5 Si vous appréciez l'assignement de eGOBEEP qui permet d'accéder facilement à toutes les fonctions de l'imprimante et du lecteur de

cassette, vous souhaiterez peut-être essayer ce court programme écrit par Clifford Stern. Il permet d'obtenir des choses similaires à eGOBEEP pour les modules horloge et X-Fonctions.

Entrez simplement les données dans la pile si besoin est, ENTER↑, puis entrez le numéro d'ordre de la fonction et XEQ "EFT". Le programme EFT s'arrêtera en PAUSE pendant environ une seconde pour vous permettre l'entrée d'un argument en ALPHA comme, par exemple, un nom de fichier. Si l'argument alphanumérique que vous désirez se trouve déjà dans le registre ALPHA, vous n'avez rien à entrer. Les entrées en ALPHA sont limitées à sept caractères. EFT construit une courte suite d'octets contenant l'instruction XROM désirée, puis il exécute cette séquence d'instruction. La séquence est en fait contenue dans les registres d'état a et b.

Il existe deux limitations notables à EFT. La première est que, contrairement à eGOBEEP, EFT ne fonctionne qu'en mode RUN (hors mode PRGM), aussi, il ne peut être utilisé pour entrer des lignes de programme contenant des instructions des modules horloge et de fonctions d'extension. La seconde est que vous ne pouvez employer EFT pour exécuter PSIZE (fonction numéro 30), ou pour exécuter XYZALM (fonction numéro 93) lorsque l'argument contenu en Z doit être différent de zéro. PSIZE modifierait la séquence d'octets contenue dans les registres a et b, séquence qu'EFT est en train d'exécuter. La contrainte de XYZALM est due au fait que le contenu du registre Z est remplacé par zéro lorsque la fonction XYZALM est exécutée depuis les registres d'état. Vous devez également éviter d'employer EFT afin d'exécuter PCLPS (fonction numéro 27) si cela devait effacer EFT lui-même, car vous commenceriez alors à exécuter le contenu des registres d'assignement.

Les lignes 15 à 23 sont présentes pour arrêter l'exécution au moment du retour en mémoire programme en cas d'erreur. Si vous vous arrêtez dans les registres d'état, le processeur met beaucoup de temps pour calculer le numéro de ligne (cf. p. 62).

Les codes-barres du programme EFT sont donnés à l'appendice E.

Entrées pour LB :

Ligne 02 : 144, 117 Ligne 04 : 145, 117 Ligne 11 : 144, 117
Ligne 12 : 247, 145*, 112, 176*, 84, 12, 117, 166*
Ligne 16 : 245, 127, 127, 116, 145*, 124
Ligne 18 : 206, 117 Ligne 19 : 206, 123 Ligne 20 : 206, 118
Ligne 21 : 206, 124

* indique un caractère invisible à l'impression. Le caractère hex A6 (décimal 166) de la ligne 12 fait sauter 6 espaces à l'imprimante.

La fonction (l'en-tête) CARD READER (ou CRD RDR 1F suivant la version) n'est pas accessible par eGOBEEP. Les fonctions du lecteur de cartes (XROM 30) ne sont pas accessibles par eGOBEEP

Codes numériques des fonctions pour "EFT" et EGOBEEP (les numéros XROM sont aussi inclus pour référence).

"EFT"
(XFUNCTIONS, TIME, WAND)

	-EXT FCN 1B	
1	ALENG	25,01
2	ANUM	25,02
3	APPCHR	25,03
4	APPREC	25,04
5	ARCLREC	25,05
6	AROT	25,06
7	ATOX	25,07
8	CLFL	25,08
9	CLKEYS	25,09
10	CRFLAS	25,10
11	CRFLD	25,11
12	DELCHR	25,12
13	DELREC	25,13
14	EMDIR	25,14
15	FLSIZE	25,15
16	GETAS	25,16
17	GETKEY	25,17
18	GETP	25,18
19	GETR	25,19
20	GETREC	25,20
21	GETRX	25,21
22	GETSUB	25,22
23	GETX	25,23
24	INSCHR	25,24
25	INSREC	25,25
26	PASN	25,26
27	PCLPS	25,27
28	POSA	25,28
29	POSFL	25,29
30	PSIZE	25,30
31	PURFL	25,31
32	RCLFLAG	25,32
33	RCLPT	25,33
34	RCLPTA	25,34
35	REGMOVE	25,35
36	REGSWAP	25,36
37	SAVEAS	25,37
38	SAVEP	25,38
39	SAVER	25,39
40	SAVERX	25,40
41	SAVEX	25,41
42	SEEKPT	25,42
43	SEEKPTA	25,43
44	SIZE?	25,44
45	STOFLAG	25,45
46	X<>F	25,46
47	XTOA	25,47

	-TIME- C	
65	ADATE	26,01
66	ALMCAT	26,02
67	ALMNOW	26,03
68	ATIME	26,04
69	ATIME24	26,05
70	CLK12	26,06
71	CLK24	26,07
72	CLKT	26,08
73	CLKTD	26,09
74	CLOCK	26,10
75	CORRECT	26,11
76	DATE	26,12
77	DATE+	26,13
78	DDAYS	26,14
79	DMY	26,15
80	DOW	26,16
81	MDY	26,17
82	RCLAF	26,18
83	RCLSW	26,19
84	RUNSW	26,20
85	SETAF	26,21
86	SETDATE	26,22
87	SETIME	26,23
88	SETSW	26,24
89	STOPSW	26,25
90	SW	26,26
91	T+X	26,27
92	TIME	26,28
93	XYZALM	26,29

	- WAND 1F -	
129	WNDDTA	27,01
130	WNDDTX	27,02
131	WNDLNK	27,03
132	WNDSUB	27,04
133	WNDSCN	27,05
134	"WNDTST	27,06

eGOBEEP
(HP-IL, PRINTER)

	-MASS ST 1H	
1	CREATE	28,01
2	DIR	28,02
3	NEWM	28,03
4	PURGE	28,04
5	READA	28,05
6	READK	28,06
7	READP	28,07
8	READR	28,08
9	READRX	28,09
10	READS	28,10
11	READSUB	28,11
12	RENAME	28,12
13	SEC	28,13
14	SEEKR	28,14
15	UNSEC	28,15
16	VERIFY	28,16
17	WRTA	28,17
18	WRTK	28,18
19	WRTP	28,19
20	WRTPV	28,20
21	WRTR	28,21
22	WRTRX	28,22
23	WRTS	28,23
24	ZERO	28,24
25	--	28,25
26	-CTL FNS	28,26
27	AUTOIO	28,27
28	FINDID	28,28
29	INA	28,29
30	IND	28,30
31	INSTAT	28,31
32	LISTEN	28,32
33	LOCAL	28,33
34	MANIO	28,34
35	OUTA	28,35
36	PWRDN	28,36
37	PWRUP	28,37
38	REMOTE	28,38
39	SELECT	28,39
40	STOPIO	28,40
41	TRIGGER	28,41

	-PRINTER 2D	
65	ACA	29,01
66	ACCHR	29,02
67	ACCOL	29,03
68	ACSPEC	29,04
69	ACX	29,05
70	BLDSPEC	29,06
71	LIST	29,07
72	PRA	29,08
73	"PRAXIS	29,09
74	PRBUF	29,10
75	PRFLAGS	29,11
76	PRKEYS	29,12
77	PRP	29,13
78	"PRPLOT	29,14
79	"PRPLOTP	29,15
80	PRREG	29,16
81	PRREGX	29,17
82	PRZ	29,18
83	PRSTK	29,19
84	PRX	29,20
85	REGPLOT	29,21
86	SKPCHR	29,22
87	SKPCOL	29,23
88	STKPLOT	29,24
89	FMT	29,25
	--	

Le lecteur de cartes n'est pas accessible par EGOBEEP (WROM 30)

CHAPITRE CINQ

COMPRENDRE L'EDITION DE PROGRAMMES SUR LA HP-41

A la section 2B, je vous avais promis d'expliquer comment sont crées les nuls lorsque les programmes sont tapés et édités, et sous quelles conditions ils peuvent être enlevés par PACK. Cette explication sera simplifiée par la construction d'une instruction synthétique très spéciale appelée label FO. Ce label FO est capable d'afficher plusieurs instructions suivantes comme des caractères sans les absorber comme le fait le Byte Grabber.

Contruisez tout d'abord cette instruction spéciale en utilisant LB avec les données 192, 0, 240. Si vous avez BG assigné à une touche, vous pouvez également entrer les instructions STO IND 64, RCL IND T, deux fois BST, BG et la flèche de correction pour enlever l'octet STO. Que vous employiez l'une ou l'autre méthode, vous devez PACKer immédiatement pour que le calculateur incorpore ce nouveau LBL dans le CATalogue 1. Vous avez maintenant un label global synthétique. Il est synthétique car son troisième octet est 240 décimal = FO hexadécimal (d'où le nom LBL FO). Normalement, le troisième octet d'un label apparaissant au CATalogue 1 est 241+n, où n est le nombre de caractères composant le nom du LBL. Un troisième octet de 240 donne une longueur de -1 pour le nom. Il apparaît que le calculateur interprète ce paramètre hautement étrange de façons contradictoires. Pour afficher le label FO en mode PRGM, le processeur utilise n=15, qui est -1 modulo 16. Vous voyez donc LBL^{T} suivi de 15 caractères. Le processeur saute un octet (qui est normalement celui contenant le code de la touche assignée avec ce LBL) et affiche les 15 octets suivants comme des caractères. Néanmoins, si vous faites SST en mode PRGM, vous verrez que ces caractères n'ont pas été réellement absorbés par l'instruction LBL FO.

Un exemple devrait éclaircir ce point. Mais faisons tout d'abord une remarque. Ne faites pas SST sur un label FO en mode RUN et n'exécutez jamais un programme contenant un tel label. Cela 'planterait' la HP-41, rendant le clavier inopérant jusqu'à ce que vous retiriez les piles et les remettiez en place pour reprendre le contrôle. Le fait d'enlever les piles stoppe une 'boucle infinie' interne, ici sans altérer le contenu de la mémoire. Exécuter une instruction label FO conduit à l'un des 'plantages' les moins violents. D'autres (comme utiliser BG sur le .END. et l'effacer) causent un inévitable MEMORY LOST.

Avec votre label FO à l'affichage (en mode PRGM), entrez la séquence : -, *, /, X<Y? (appuyez sur XEQ ALPHA X SHIFT COS Y ? ALPHA), X>Y?, X<=Y?, Σ+, Σ-, HMS+, HMS-, MOD, %, %CH, P-R, R-P, LN, X↑2, SQRT, Y↑X, CHS, E↑X, LOG, 10↑X, E↑X-1, SIN et COS. Revenez maintenant sur le label FO et vous voyez : LBL "BCDEFGHIJKLNOP" (si vous n'obtenez pas cet affichage, PACK et vous devriez l'avoir).

Les caractères B à P sont en fait les instructions * à LN qui suivaient le label FO. Les lignes 4 et 5 de la table des codes montrent la correspondance des instructions et des caractères. Pour illustrer davantage cette correspondance, localisez le /, effacez le et revenez sur le label FO. Vous obtenez : LBL "B DEFGHIJKLMNOP".

Ceci montre que lorsque des instructions sont détruites, elles sont remplacées par des nuls, qui sont normalement invisibles. Le surligné est la représentation d'un octet nul sous forme de caractère. Faites PACK et vous voyez : LBL "BDEFGHIJKLMNOPQ" ce qui montre que le nul a été enlevé par le PACKing.

Le label F0 nous permet de faire une éclatante démonstration du fonctionnement du processeur lorsque des instructions sont insérées dans un programme. Positionnez-vous avec SST sur l'instruction X<Y?, correspondant au caractère D et insérez une instruction +. Revenez sur le label F0 et vous voyez : LBL "BD@ EFGHIJ".

Le caractère @ correspond à l'instruction +. Mais vous ne vous attendiez certainement pas à trouver ces six nuls (caractères surlignés). Cet exemple illustre le fait que si une instruction est insérée là où il n'y avait pas de place (c'est-à-dire s'il n'y a pas suffisamment de nuls à cet endroit), sept octets nuls sont insérés pour la nouvelle instruction, même si un seul était nécessaire. Le reste de la mémoire programme, jusqu'au .END. terminal est décalé d'un registre (sept octets) vers le bas, et le nombre de registres libres diminué d'une unité. (Reportez-vous au sixième chapitre pour trouver la description de l'organisation de la mémoire et l'emplacement des registres libres.) Du fait des opérations sur les registres disponibles au niveau du microprocesseur, le décalage d'un registre entier est plus rapide que le décalage d'un seul octet n'aurait été.

Des insertions pour lesquelles le nombre de nuls déjà présents est suffisant ne perturbent pas le reste de la mémoire programme. Positionnez-vous sur l'instruction + et entrez les instructions STO 01, STO 02, STO 03, STO 04, STO 05 et STO 06. Revenez sur le label F0 et vous voyez : LBL "BD@123456EFGHIJ".

Les six nouveaux octets introduits ont exactement rempli l'espace disponible. Toute insertion supplémentaire ouvrirait un autre groupe de sept octets.

Maintenant que nous avons vu comment les insertions sont traitées par le processeur, vous devez comprendre comment fonctionne le Byte Grabber. Lorsqu'il est pressé en mode PRGM, le Byte Grabber crée un préfixe TEXT 7, suivi d'un octet nul et d'un troisième octet que l'on a toujours pris égal à 63 dans ce livre (MK peut lui donner la valeur que vous désirez). Une instruction TEXT 7 occupe 8 octets en mémoire, dont un est le préfixe TEXT 7 et les autres les sept caractères. Mais le processeur sait seulement qu'il doit faire de la place pour les trois octets à insérer. Dans le cas habituel où il n'y a pas de nuls présents pour l'insertion, il ouvre un espace de sept octets. C'est pourquoi le huitième octet -c'est à dire, le septième caractère- est pris dans le programme existant. La figure 5.1 illustre la capture de cet octet sur l'exemple du premier chapitre.

AVANT

Instructions:	ENTER↑	STO	IND 31	PI
Equivalent hexa:	83	91	9F	72
Equivalent décimal:	131	145	159	114

APRES

Instructions:	ENTER↑	" ? 0"	TONE Y
Equivalent hexa:	83	F7 0 3F 0 0 0 0 91	9F 72
Equivalent décimal:	131	247 0 63 0 0 0 0 145	159 114

Figure 5.1: Création de TONE Y en utilisant le Byte Grabber

Le byte grabber peut être utilisé afin de dérober jusqu'à 5 octets si vous le désirez. Faites simplement PACK ou assurez-vous qu'il n'y a pas de nul avant les octets que vous voulez dérober, exactement comme vous le feriez pour le Byte Grabber ordinaire. Puis, avant de presser BG, insérez un à quatre octets de 'remplissage'. Ainsi, pour dérober deux octets, vous

pouvez insérer une instruction X<>Y avant de presser BG. Nous avons déjà fait cela au second chapitre pour dérober le 1 des exposants sans devoir PACKer. Pour dérober trois octets, vous pouvez insérer le nombre 9 puis BG. Pour dérober quatre octets, insérez EEX 9 puis BG. Pour tous ces cas, l'idée directrice est la même. Le processeur n'a besoin que de trois octets pour le Byte Grabber. Si vous ouvrez 7 octets par une insertion et n'en remplissez que quatre (par exemple en insérant 1E9) et pressez BG, le Byte Grabber se placera dans les trois nuls restants. Mais comme l'instruction TEXT 7 est longue de 8 octets, elle doit prendre ses cinq derniers octets dans le programme existant.

Soyez très prudent lorsque vous dérobez plus d'un octet. Vous pourriez accidentellement dérober une partie du .END. terminal. Si jamais cela vous arrivait, **ne pressez pas la flèche de correction !** Faites immédiatement BST et BG de nouveau pour libérer le .END. de la chaîne de caractères créée par le Byte Grabber.

Vous avez peut-être l'impression que PACK retire tous les nuls d'un programme. Ce n'est pas le cas. Certains nuls sont parfois porteurs d'informations essentielles et ne doivent pas être détruits.

Le premier cas est celui d'un nul situé entre deux lignes successives d'entrées numériques. Revenons à notre label F0 que nous avions laissé dans l'état suivant : LBL "BD@123456EFGHIJ".

Faites SST jusqu'au - (moins) juste devant l'instruction * correspondant au caractère B. Entrons les deux nombres 1E3 et 56. Appuyez deux fois sur la touche ALPHA pour terminer l'instruction 1E3 avant de taper 56. Revenez au label F0. Vous obtenez : LBL " ØØØ ØØ B".

Les trois premières novas sont l'instruction 1E3, tandis que les deux suivantes sont le nombre 56. Faites maintenant PACK et vous obtenez : LBL "ØØØ ØØBD@123456" .

Tous les nuls ont disparu excepté celui qui se trouve entre les deux entrées numériques. Ce nul est nécessaire pour empêcher ces deux lignes de n'en former qu'une seule. C'est pour cette raison qu'un nul situé entre deux entrées numériques n'est pas enlevé par PACK. La présence indispensable de ces nuls pour séparer les instructions explique les nuls que nous avons vus avant de PACKer. Le système d'exploitation de la HP-41 ajoute d'office un nul devant chaque entrée numérique lorsqu'elle est tapée. Ce nul sera détruit par un PACK, sauf si la ligne précédente est également une entrée numérique. De même, il insiste pour qu'il y ait au moins un nul séparant une entrée numérique de l'instruction qui la suit. Dans l'exemple précédent, sept octets ont été ouverts lors de la frappe du 6 de 56. Si ces octets n'avaient pas été ouverts, il n'y aurait pas eu de délimitation entre le 56 et l'instruction suivante. Si cette instruction avait été numérique, le 56 y aurait été amalgamé en créant de ce fait une seule entrée numérique (incorrecte). Comme la HP-41 ouvre sept octets à la fois, sept nuls ont été créés.

Chaque octet nul faisant partie d'une instruction de plusieurs octets n'est pas affecté par PACK. Par exemple l'instruction ST+ 00 apparaît dans un label F0 comme Ø . Le second octet est un nul. Cet octet ne peut pas être altéré par PACK, car il fait partie intégrante de l'instruction et est porteur d'une information, ici un numéro de registre. Connaissant ces règles complexes sur le rôle des nuls, on comprend pourquoi l'instruction PACK met parfois longtemps pour faire son travail.

Il est encore un point obscur à propos des nuls dont nous devons parler. Normalement, lorsque vous entrez une instruction, elle est insérée après l'instruction courante, recouvrant les nuls existants et ouvrant un nouvel espace si nécessaire. Néanmoins, si la ligne courante est un END (ou le .END.), la nouvelle instruction est insérée là où se trouvait le END, et

celui-ci est décalé de sept octets vers le bas. Ceci a lieu même si suffisamment de nuls étaient présents au-dessus du END.

Pour illustrer ce comportement des ENDs, commencez par taper la séquence : label FO, -, *, END. Placez-vous sur le label FO, PACK, et vous voyez LBL "B[illegible] suivi d'autres caractères. Les second, troisième et quatrième caractères visibles sont le END. Effacez maintenant l'instruction *. Si vous insérez une nouvelle instruction * ici, elle prendra exactement la place de l'ancienne. Si toutefois vous faites SST jusqu'au END et insérez une nouvelle instruction *, le résultat est : LBL " B [illegible] et quatre autres caractères.

L'instruction * a été insérée là où se trouvait le END, tandis que celui-ci a été décalé de sept octets vers le bas de la mémoire. Six nouveaux nuls ont étés créés alors qu'aucun d'entre eux n'était vraiment nécessaire. C'est pourquoi il est préférable de ne pas faire d'insertion dans un programme avec le END présent à l'affichage. Faites plutôt BST avant de faire l'insertion pour profiter d'éventuels nuls précédant le END. Bien sûr PACK éliminera ces nuls, mais cette technique vous permettra peut-être d'éviter d'avoir à changer la partition mémoire (le SIZE) pour entrer un programme qui 'tenait'.

Vous remarquerez dans le dernier exemple que le END a changé d'apparence après avoir été déplacé. Ceci est dû au fait que les deux premiers octets d'un END ou d'un label global sont utilisés pour stocker l'adresse relative de l'élément précédent du CATalogue 1.

Aussi si le CATalogue 1 contient LBL "ABC", END, .END., le .END. contient l'adresse du END, le END contient celle du LBL "ABC", et le LBL "ABC" contient une zone adresse vide, indiquant le sommet du CATalogue 1. Le calculateur utilise ce chaînage, en remontant la chaîne depuis le .END. chaque fois qu'il recherche un label global. Le chaînage est également utilisé lors des BST. Lorsque BST est pressée, le calculateur recherche le label global ou le END précédent et redescend pas à pas à partir de là pour trouver l'instruction cherchée. Ceci est nécessaire car le numéro de ligne n'est pas stocké en mémoire programme. Sans commencer à partir d'une position connue du CATalogue 1, le calculateur ne peut savoir si un octet constitue une instruction ou un suffixe pour l'instruction précédente. L'opération BST fonctionne de la seule manière possible, en comptant à partir d'une position connue. Ceci explique pourquoi BST peut prendre tant de temps si on l'exécute près du END d'un long programme qui a seulement un label global à la ligne 01.

Les adresses relatives sont également contenues dans les instructions GTO et XEQ locales (non globales), ce que nous avons vu au troisième chapitre. La première exécution de l'une de ces instructions demande un certain temps correspondant à la recherche du LBL appelé. Mais lorsque cette recherche est terminée, l'adresse est stockée dans l'instruction de saut, permettant un branchement plus rapide lors des exécutions suivantes. A l'aide du label FO, on peut observer les instructions GTO et XEQ avant et après la compilation.

Problèmes :

5.1 Prévoyez le résultat des pas suivants, y compris le nombre et l'emplacement des nuls. Employez le label FO pour vérifier vos prédictions.
a) Entrez les instructions +, 3, -, 4, 5 et *. Insérez >+ et >- après le +. Insérez RCL 05 après le 4.
b) Entrez les instructions +, -, XEQ 00, GTO 99, *, et /. Détruisez le GTO 99 et entrez ST+ 75.

CHAPITRE SIX

STUCTURE MEMOIRE DE LA HP-41. REGISTRES D'ETAT

Ce chapitre complètera vos connaissances sur le fonctionnement et les processus de la HP-41. Certains détails donnés ici ne seront peut-être pas applicables immédiatement, mais ils sont donnés pour vous fournir une référence. Ils sont également un point de départ pour ceux d'entre vous qui voudraient écrire leurs propres programmes synthétiques de manipulation de registres. Même si vous ne prévoyez que d'utiliser les techniques les plus simples de la programmation synthétique, et d'employer des programmes tout prêts comme ceux du PPC ROM ou de la librairie des utilisateurs HP, ces informations vous donneront une idée sur la façon dont fonctionnent ces programmes de manipulation.

6A. Structure de la mémoire

La figure 6.1 illustre l'organisation de la mémoire tant programme que données ou système et mémoire étendue de la HP-41. La mémoire étendue, y compris celle contenue dans le module X-Fonctions, est appelée annexe car les programmes ne peuvent y être exécutés directement. Ils doivent tout d'abord être rappelés en mémoire principale.

L'organisation fonctionnelle de la mémoire principale est donnée figure 6.2 à la page suivante. Les registres de données s'étendent depuis la partition (plus de détails là-dessus lorsque nous parlerons du registre d'état c) jusqu'au sommet de la mémoire principale. Les programmes de l'utilisateur commencent sous la partition et s'étendent jusqu'au .END., qui est déplacé automatiquement par le calculateur si besoin est. Au-dessous du .END. se trouvent les registres disponibles -pour d'autres programmes, pour stocker des alarmes ou pour des assignements. Ils peuvent également être convertis en registre de données en modifiant le SIZE (la partition), ce qui pousse vers le bas toutes les données et les programmes dans les registres libres. Diminuer le SIZE remonte les données et les programmes dans la mémoire, augmentant ainsi le nombre de registres disponibles en perdant des registres de données au sommet de la mémoire. Le nombre de registres disponibles peut être contrôlé à tout moment en faisant GTO.000 en mode PRGM ou en faisant RTN en mode RUN, puis en passant en mode PRGM. Dans les deux cas, il s'affichera 00 REG nn, où nn est le nombre de registres libres.

Au-dessous des registres libres se trouvent les alarmes et les assignements. Les assignements des fonctions du CATalogue 2 (périphériques) et du CATalogue 3 (fonctions de bases) occupent les registres commençant à l'adresse décimale 192 et s'étendent vers le haut. Chaque registre contenant un assignement commence par un octet marqueur F0. Les six autres octets contiennent la paire d'assignements, chacun d'eux requérant trois octets. De ces trois octets, les deux premiers définissent la fonction. Ce sont pour ces deux octets que vous donnez deux valeurs décimales lorsque vous utilisez MK. Le troisième octet définit la touche à laquelle est assignée la fonction.

Les alarmes se trouvent stockées immédiatement au-dessus des registres d'assignement. Chaque alarme emploie un registre pour l'heure de l'alarme, plus d'autres en cas de message ou d'intervalle de repétition associé à l'alarme. Un registre d'en-tête, à la fin du groupe des alarmes, juste

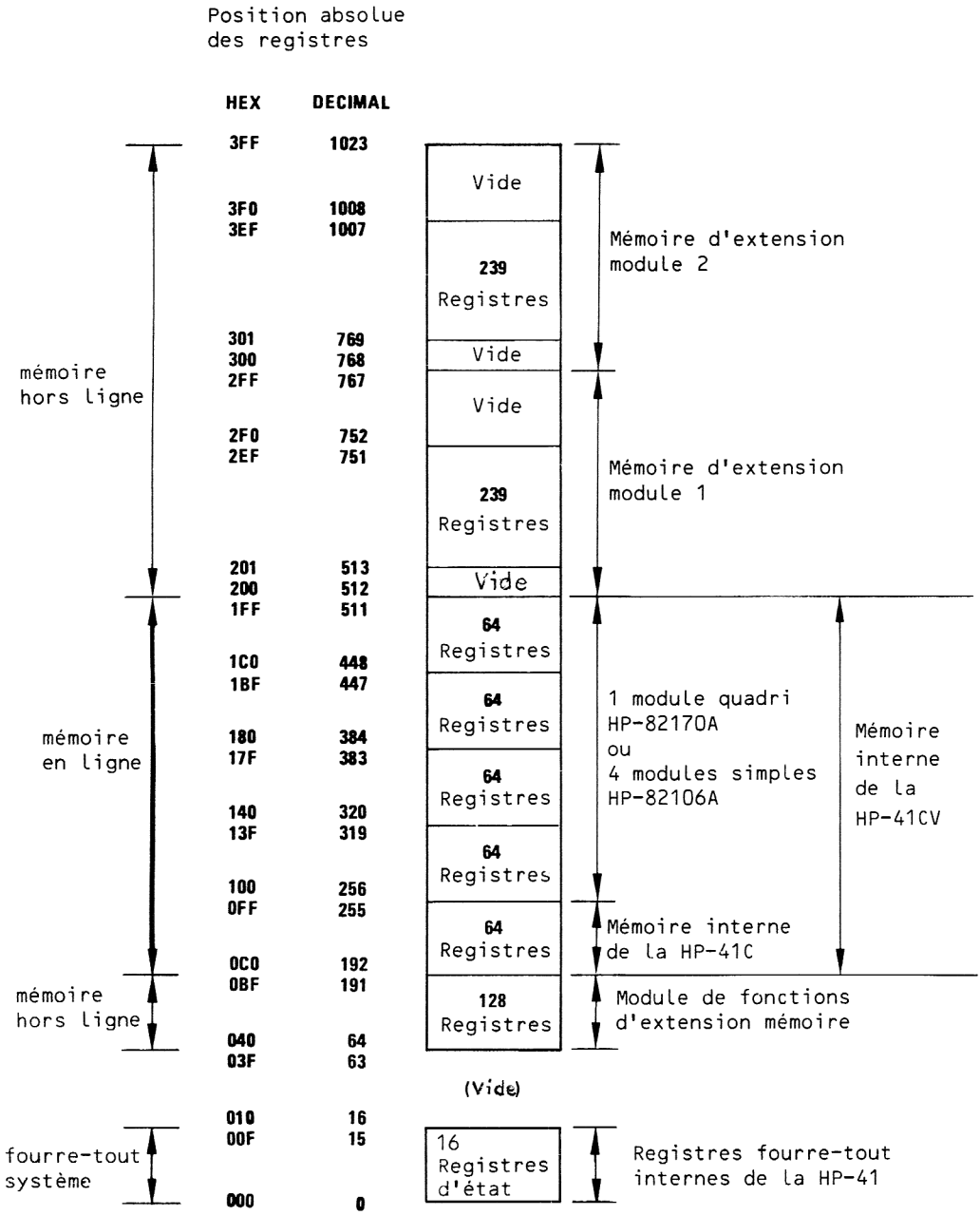


Figure 6.1 : Structure générale
de la mémoire de la HP-41

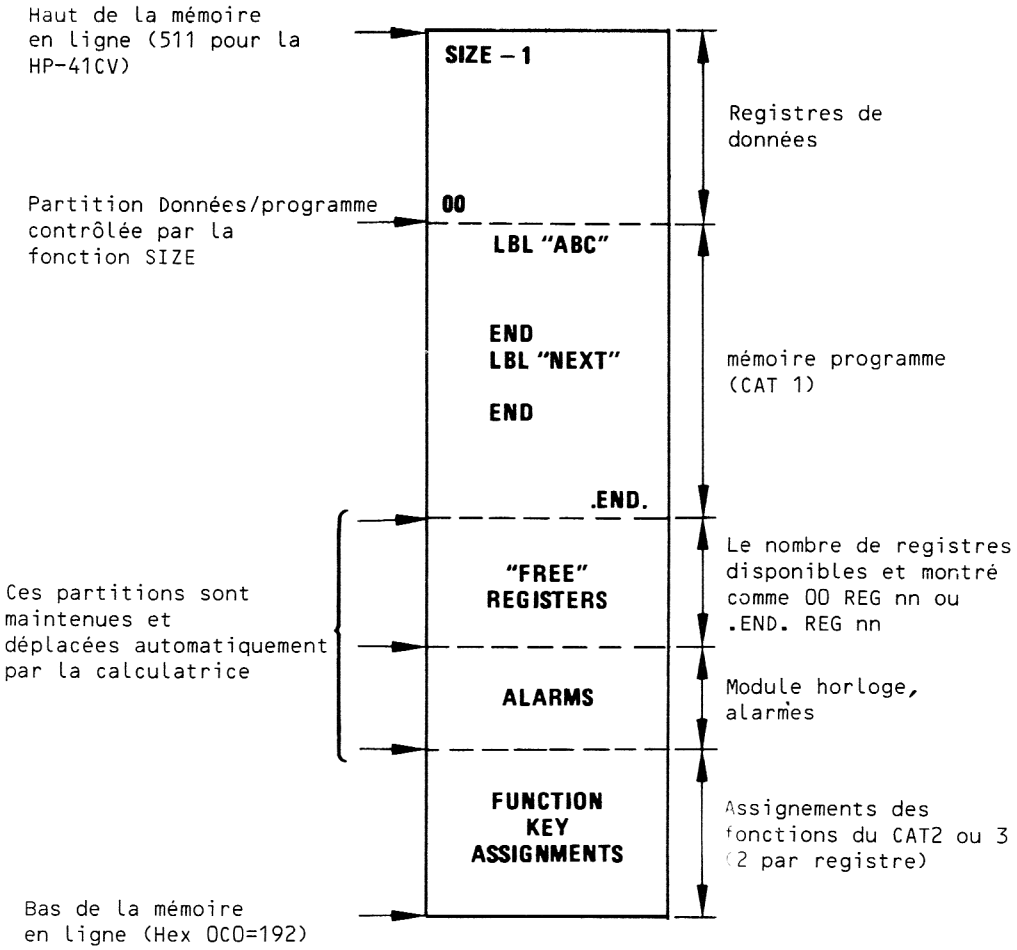


Figure 6.2 : Utilisation de la mémoire en ligne

au-dessus du dernier registre d'assignement, contient le nombre total d'alarmes enregistrées. Un autre registre délimite le sommet du bloc des registres d'alarme.

Ceci termine la description de la structure mémoire de la HP-41, excepté une zone très importante -les registres d'état, ou registres fourre-tout du système. Le nom "registre d'état" provient du fait que le contenu de ces 16 registres est enregistré sur la piste 1 d'une carte d'état lors de l'exécution de la fonction WSTS du lecteur de cartes.

Les 16 registres d'état se trouvent tout à fait en bas de l'espace adressable de la HP-41, aux adresses 0 à 15 (décimal). Leurs noms sont **T, Z, Y, X, L, M, N, O, P, Q, ⊢, a, b, c, d** et **e** respectivement. Vous connaissez déjà certains de ces registres ; les cinq premiers sont décrits dans le manuel d'utilisation de la HP-41, les autres sont présentés au second chapitre. La figure 6.3 résume l'usage que fait le processeur de ces registres.

Les registres de la pile, T, Z, Y, X et L sont accessibles à l'utilisateur par les moyens normaux. En plus des instructions ENTER↑, RDN, R↑ et LAST X présentes sur de nombreux calculateur HP, la HP-41 permet l'accès direct à tous les registres de la pile à l'aide de fonctions telles que RCL Z ou X<> L. La programmation synthétique permet l'emploi de STO, RCL et X<> avec tous les autres registres d'état.

Les registres M, N, O et P contiennent les 24 caractères du registre ALPHA. Les contenus du registre ALPHA sont toujours justifiés à droite dans les registres d'état. L'octet le plus à droite, l'octet 0, du registre M contient le caractère le plus à droite du registre ALPHA. L'octet 1 contient l'avant-dernier caractère, etc... Si le registre ALPHA contient 7 caractères ou moins, seul le registre M est utilisé. Au fur et à mesure de l'entrée des caractères, les autres sont décalés de droite à gauche et poussés dans les registres N, O et P. Lorsque le 24^{e} caractère est introduit, un signal sonore se fait entendre. Si vous ajoutez d'autres caractères, ils pousseront ceux de gauche dans la partie fourre-tout du registre P. Néanmoins, si vous restez en mode ALPHA, ou tout du moins si vous n'avez pas un affichage numérique, les quatre caractères des positions 25 à 28 (les 4 octets les plus à gauche de P) resteront disponibles pour être rappelés par des méthodes synthétiques comme RCL P. Le programme de code Morse de l'appendice B utilise ces 28 caractères disponibles.

Les deux octets de gauche de P sont employés par le processeur dans certains cas. Le premier octet est un codage du format d'affichage (FIX, SCI, ENG, Drapeaux 28 et 29, et le nombre de chiffres). Cet octet est remis à jour par le processeur lorsque un affichage **numérique** est utilisé ou lorsqu'une entrée de donnée est exécutée. Le second octet de P est utilisé lors de ces entrées numériques, qu'elles soient au clavier ou par programme.

L'exécution d'un CATalogue altère également les deux premiers octets de P. Le premier octet contient alors le numéro du catalogue (1, 2 ou 3), tandis que le second contient le numéro de ligne interne dans le catalogue.

Le registre Q est utilisé chaque fois qu'un nom de label ALPHA est épelé. Ceci a lieu lorsque l'instruction LBL est entrée au clavier et lorsque les GTO et XEQ correspondant sont entrés au clavier ou exécutés dans un programme. Le nom du label est placé dans le registre Q, mais **à l'envers.**

Le registre Q est également modifié lors des entrées de données,

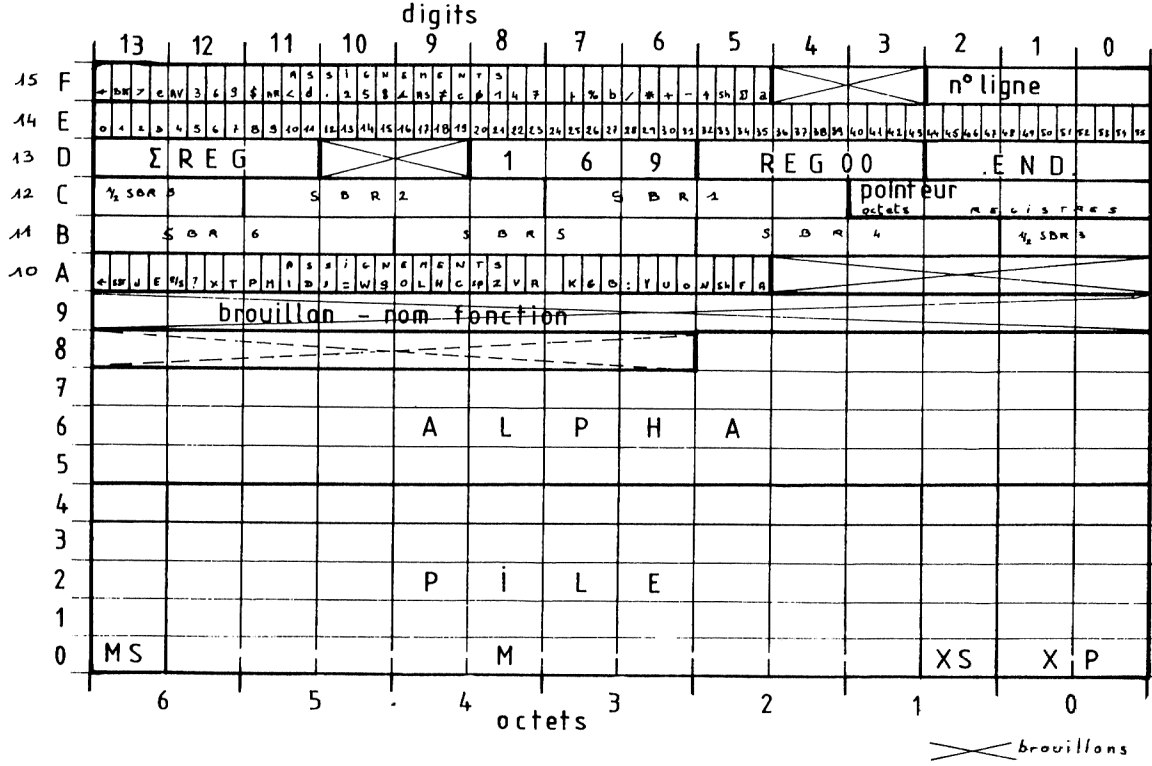


Figure 6.3 : Les registres d'état

manuelle ou en cours de programme. Le nombre est composé en Q avant d'être transféré dans le registre X. Méfiez-vous également du fait que le registre Q est utilisé par l'imprimante si elle est connectée.

Le registre ⊢ contient les bits codant les positions primaires de touches (non shiftées) qui portent un assignement. Les quatre premiers octets et la moitié du cinquième sont réservés à cet usage.
Cela fait partie des astuces développées par les ingénieurs de chez HP permettant d'accélérer l'exécution des fonctions à partir du clavier. Lorsqu'une touche est pressée en mode USER, le processeur recherche le bit correspondant dans le registre ⊢ si cette touche est une position primaire. Si ce bit est à zéro, le processeur sait que la touche n'a pas été assignée et a le choix entre deux actions.

Si la touche en question n'appartient pas au premier rang ou aux touches primaires du second rang (touches ALPHA A-J et a-e), la fonction par défaut, c'est-à-dire celle gravée sur la touche, est exécutée. Si la touche appartient à ces rangs, le calculateur recherche le label local correspondant (A à J ou a à e) dans le programme courant. Si ce label est trouvé, l'exécution du programme démarre à cet endroit. S'il ne l'est pas, le processeur exécute finalement la fonction par défaut.

Si le bit correspondant dans le registre ⊢ est à 1, le processeur sait que la touche a été assignée. Il recherche alors cet assignement, tout d'abord dans les registres d'assignement. Si cet assignement n'est pas trouvé, le processeur contrôle le quatrième octet de chaque label global du CATalogue 1. Cet octet contient l'éventuel assignement du label. Si aucun assignement n'est trouvé (ce qui n'est pas normal), alors une fonction telle CAT, ABS ou 1/X est exécutée.

Grâce aux bits d'assignement, la première partie de ces recherches décrites plus haut s'effectue rapidement. Néanmoins, la recherche du label local peut être très longue si le programme dépasse une centaine de lignes. C'est pourquoi il est astucieux d'assigner X<>Y et RDN (fonctions très employées) à leurs touches par défaut. En mode USER, cet assignement paraissant redondant est prioritaire sur la recherche du label local, permettant ainsi de gagner beaucoup de temps.

Les deux octets et demi les plus à droite du registre ⊢ contiennent le code hexadécimal de la fonction la plus récemment exécutée au clavier. L'imprimante peut également utiliser cette zone.

Les registres a et b contiennent le pointeur programme et la pile de retour des sous-programmes. Chaque pointeur occupe deux octets, représentables par quatre chiffres héxadécimaux. Les octets 0 et 1 du registre b contiennent le pointeur programme courant. Lorsqu'une instruction XEQ est rencontrée, ce pointeur est poussé dans la pile de retour -c'est-à-dire dans les octets 2 et 3 du registre b. Si un autre XEQ est rencontré avant le RTN du premier, le pointeur programme et la première adresse de retour sont décalés de deux octets de plus vers la gauche. La pile de retour contenue dans les registres a et b peut ainsi contenir 6 adresses de retour en attente.

Lorsque l'exécution rencontre une instruction RTN, la première adresse, contenue dans les octets 2 et 3 du registre b est contrôlée. Si elle est nulle, le pointeur courant est retenu et le contrôle est rendu au clavier. Dans le cas contraire, la pile de retour est décalée de deux octets vers la droite, l'ancienne première adresse de retour prenant la place du pointeur de programme. L'exécution reprend donc à cette adresse de la mémoire, un pas plus bas que l'instruction XEQ qui avait envoyé cette adresse dans la pile.

Maintenant, quelque détails sur les pointeurs programmes. Les quatre chiffres hexadécimaux représentant un pointeur programme sont interprétés différemment selon qu'il s'agit d'un pointeur de RAM (**R**andom **A**ccess **M**emory : mémoire à accès aléatoire ou mémoire vive) ou de ROM (**R**ead **O**nly **M**emory : mémoire pouvant être lue seulement ou mémoire morte). Dans le cas de la RAM, les quatre premiers bits pointent l'octet à l'intérieur du registre, et les 12 autres donnent l'adresse absolue du registre à partir du bas de la mémoire. Le format est donc : ObbbOOOrrrrrrrrr, où bbb donne le numéro de l'octet à l'intérieur du registre (exprimé sur trois bits donc variant de 0 à 7 ce qui suffit car la valeur maximum est ici 6 = 0110 base 2) et où rrrrrrrrr donne le numéro du registre (exprimé sur neuf bits car sa valeur maximale est 511 = 000111111111 base 2). Ainsi le nombre 0101000110101110 = hexa 51AE pointe l'octet cinq (donc le sixième) du registre 1AE (= 430 décimal). Les numéros des octets s'étendent de 6 à 0 lorsque le pointeur 'descend' un registre programme. Ainsi 61AE est au dessus de 41AE en mémoire programme et 41AE est au dessus de 61AD.

Les pointeurs de retour en RAM sont identiques aux pointeurs ordinaires, sauf que les trois bits désignant l'octet à l'intérieur du registre sont décalés vers la droite. Ces bits, normalement les second, troisième et quatrième sont décalés aux positions cinq, six et sept. Le format d'un pointeur de retour en RAM est donc : OOOObbbrrrrrrrrr.

Les pointeurs de ROM comprennent une adresse de port dans les quatre premiers bits plus 12 bits codant le numéro de l'octet dans ce port : ppppbbbbbbbbbbb.

L'adresse du port d'un pointeur de ROM n'est pas exactement le numéro de port gravé sur le fond de la machine. La correspondance est :

Adresse du port	Port physique ou périphérique
0	ROM 0 (interne)
1	ROM 1 (interne)
2	ROM 2 (interne)
3	Utilisé uniquement par la 41CX
4	Module de service (S.A.V.)
5	Module horloge ou horloge 41CX
6	Imprimante (IL et non IL)
7	HP-IL et unité de cassette
8	Port 1, 4K inférieurs
9	Port 1, 4K supérieurs
A	Port 2, 4K inférieurs
B	Port 2, 4K supérieurs
C	Port 3, 4K inférieurs
D	Port 3, 4K supérieurs
E	Port 4, 4K inférieurs (lect. de cartes)
F	Port 4, 4K supérieurs

Chaque adresse de port peut contenir une ROM de 4K (4 kilo-octets = 4096 octets = hexa FFF+1). Le numéro d'octet codé sur 12 bits commence à zéro et augmente jusqu'à FFF au fur et à mesure que les instructions sont séquentiellement exécutées.

Autre détail important : Lorsque vous faites RCL b en mode RUN à une ligne spécifique de la mémoire, la valeur du pointeur est généralement un octet au-dessus de l'endroit où vous vous trouvez. Ainsi si une instruction RCL M se trouve dans les octets 6 et 5 du registre 1AE, le pointeur résultant aura pour valeur 01AF, soit un octet plus haut que la position

actuelle. Si des nuls sont présents, le pointeur sera plus haut. En fait, il sera exactement un octet plus haut que le groupe de nuls précédant l'instruction.

Le registre d'état c contient des informations essentielles pour définir la configuration de la mémoire. En se référant à la figure 6.3, nous étudierons le registre c de droite à gauche.

Les trois derniers chiffres hexadécimaux (à droite) du registre c contiennent un pointeur du registre dans lequel se trouve le .END., qui marque la fin de la mémoire accessible à l'utilisateur. Le .END. est toujours positionné dans les trois octets de droite de son registre, précedé par des nuls pour occuper l'espace séparant la dernière instruction du .END.

Les trois chiffres hexadécimaux suivants contiennent l'adresse du premier registre de données, le registre 00. Ce pointeur, souvent appelé 'rideau', marque la séparation de la mémoire programme et des registres de données. Chaque fois que le SIZE est modifié, ce pointeur est ajusté à la nouvelle partition et le contenu de la mémoire est décalé. Plusieurs courts programmes synthétiques ont été écrits pour déplacer ce rideau, transformant des programmes en données ou vice-versa. A la section 6C, vous rencontrerez l'un de ces programmes, ainsi qu'une introduction au déplacements du rideau. Dans les programmes LB et MK se trouvent des séquences qui placent temporairement le rideau à l'adresse 10 hexa = 16 décimal. Ceci permet d'accéder à la mémoire programme ou aux registres dans lesquels sont stockés les assignements, via les instructions STO IND et RCL IND. RCL normalisera, bien entendu, les contenus de ces registres. Les anciens contenus du registre c sont sauvegardés dans la pile ou dans d'autres registres d'état pour les restaurer avant que le programme ne s'arrête. LB et MK illustrent toute la puissance du contrôle du rideau.

Les trois chiffres hexa suivants de c contiennent la "constante de départ à froid". Ces trois chiffres sont 1, 6 et 9 pour toutes les HP-41. Si le processeur découvre que ces chiffres ont été modifiés, il efface toute la mémoire, et affiche le message MEMORY LOST. La raison de cette action est que, le processeur ne modifiant jamais cette constante, toute altération doit provenir d'une coupure d'alimentation. (Personne n'avait pensé à des programmeurs synthétiques vagabonds). Il prévoit alors que d'autres parties de la mémoire peuvent avoir été altérées, la remise à zéro de toute la mémoire permet donc de ne pas fournir à l'utilisateur sans méfiance, des résultats peut-être erronés. La chose principale dont il faut se souvenir est de jamais rien stocker en c sans contrôler ces trois chiffres, sous peine de MEMORY LOST. Entre parenthèses, si le registre situé immédiatement au-dessous du rideau n'existe pas, vous obtenez aussi MEMORY LOST ; alors, faites attention à ce que vous placez en c !

Les quatrième et cinquième chiffres à partir de la gauche ne sont apparemment utilisés ni par le système d'exploitation, ni par l'imprimante.

Les trois chiffres les plus à gauche de c représentent l'adresse la plus basse du bloc des registres statistiques. Par exemple, si le rideau est à l'adresse 1EB hexa, (SIZE 020 avec toute la mémoire), et que vous exécutiez ΣREG 01, le pointeur du >REG en c sera mis à 1EC hexa ce qui est bien 1EB+1.

Le registre d contient les 56 drapeaux. L'octet 6, le plus à gauche, contient les flags 0 à 7, tandis que l'octet 0 contient les flags 48 à 55. Le registre des drapeaux est utilisé comme la pierre angulaire de la programmation synthétique. Jusqu'à la sortie du module X-Fonctions, la plupart des manipulations au niveau du bit devaient se faire en chargeant

des octets dans le registre d. Une fois chargés, les trente premiers bits pouvaient être manipulés comme étant les flags 00 à 29. Un exemple illustrant cette technique est le programme RAMBYT donné au quatrième chapitre. Vous trouverez deux instructions X<> d, séparées par plusieurs lignes de manipulations de flags dans beaucoup de programmes synthétiques du PPC ROM.

Le registre e contient les bits codant les touches assignées en position secondaire (après SHIFT). Ces bits sont basés sur le même principe que ceux codant les touches en position primaire dans le registre ├. Ils occupent aussi les quatre octets et demi les plus à gauche du registre.

Les deux chiffres hexadécimaux suivants, la moitié de l'octet 2 et la moitié de l'octet 1, sont utilisés comme fourre-tout par le processeur.

Les trois derniers chiffres hexa du registre e représentent le numéro de ligne programme. Comme ce numéro n'est pas stocké avec les instructions en mémoire, et comme ces instructions varient en longueur, d'un à plusieurs octets, le processeur doit calculer le numéro de ligne. Ce calcul prend du temps et doit être fait chaque fois que vous exécutez CAT, SST sur une instruction GTO ou XEQ en mode RUN, ou sautez à un emplacement possédant un numéro de ligne inconnu. Comme ce calcul prend du temps, il n'est pas fait en cours de programme. Cela accélère son exécution, mais demande alors beaucoup de temps lorsque vous passez en mode PRGM après l'achèvement de l'exécution. Le processeur ne vous montre pas l'instruction tant qu'il n'a pas calculé son numéro de ligne. Comment sait-il que le numéro doit être recalculé ? C'est simple. Avant que le processeur ne commence à exécuter un programme (l'exécution pas à pas avec SST n'est pas prise en compte ici), il met le numéro de ligne à FFF hexa = 4095 décimal. Le numéro de ligne reste à FFF durant toute l'exécution. Lorsque vous essayez de faire SST ou de passer en mode PRGM, le processeur voit que le numéro de ligne est FFF et recalcule automatiquement le bon numéro en comptant à partir du END précédant le programme dans la mémoire.

La mystérieuse ligne 4094 que vous avez vue au premier chapitre lorsque vous avez crée le Byte Grabber est due au fait que vous avez pressé la flèche de correction en mode ALPHA. Le calculateur a donc décrémenté le numéro de ligne d'une unité sans se rendre compte que le numéro de ligne FFF était invalide. Le RCL 01 que vous avez vu était une instruction fantôme qui apparaît lorsque le registre du pointeur programme (le registre d'état b) contient zéro.

6B. Première application des registres d'état : suspension des assignements

Une des fonctions de compatibilité de la HP-41 avec la HP-67 sont les 15 touches (les deux rangs supérieurs non shiftés et le premier rang shifté) qui, lorsqu'elles sont pressées en mode USER, déclenchent la recherche puis l'exécution du label local correspondant (A-J et a-e). Mais cette recherche est conflictuelle avec un éventuel assignement de la touche, la HP-41 donnant la priorité aux assignements. Combien de fois vouliez-vous vous servir des labels locaux, mais avez rencontré un assignement sur la touche désirée ? Vous avez pressé LOG pour exécuter LBL D, mais vous avez obtenu le résultat de la fonction assignée à cette touche. Ne serait-il pas agréable de pouvoir suspendre ces assignements pendant un temps, puis de les rappeler par la suite ?

Les ruses de la programmation synthétique permettent cela, et le PPC ROM contient deux programmes qui réalisent ce voeu. Vous utiliserez SK pour suspendre les assignements puis RK pour les réactiver.

Pour utiliser SK, tapez simplement un numéro k de registre de données,

puis XEQ SK. Les bits des registres ⊢ et e sont stockés dans les registres k et k+1, tandis que leurs contenus sont remis à zéro. De ce fait, le calculateur pense qu'il n'y a plus aucun assignement. Vous pouvez donc presser la touche LOG pour exécuter LBL D. Tout assignement de fonction ou de label global qui était présent n'est pas détruit, mais suspendu.

Lorsque vous voulez réactiver vos assignements, tapez simplement le même nombre k et XEQ RK. Les contenus des registres de données k et k+1 sont rappelés et stockés dans les registres ⊢ et e. Le calculateur ayant retrouvé les bits corrects dans ces registres, les assignements fonctionnent à nouveau normalement.

Il existe une autre façon de retrouver vos assignements. Vous pouvez lire une carte avec le lecteur de cartes. Il n'est pas important que vous lisiez la carte en mode USER ou non, mais ce doit être une carte programme. Cette technique est utile si vous avez par malheur, modifié les contenus des registres k et k+1 qui contiennent les bits après l'exécution de SK.

Analysons les fonctionnements des routines SK et RK (Suspend and Reactivate Key assignments). Si vous n'avez pas de PPC ROM, entrez SK et RK en utilisant LB :

```
01 LBL "SK"                  Données pour LB
02 SIGN
03 CLX
04 X<> ⊢                     206, 122
05 XEQ 14
06 ISG L
07 TEXT 0                    240
08 .
09 X<> e                     206, 127
10 LBL 14
11 "*"
12 X<> M                     206, 117
13 STO N                     145, 118
14 ASTO IND L
15 RDN
16 RTN
17 LBL "RK"
18 SIGN
19 ARCL IND L
20 hexa F2 7F 00             242, 127,   0
21 ISG L
22 TEXT 0                    240
23 ARCL IND L
24 hexa F3 7F 0F FF          243, 127,  15, 255
25 X<> N                     206, 118
26 STO ⊢                     145, 122
27 X<> M                     206, 117
28 STO e                     145, 127
29 RDN
30 CLA
31 END
```

Le tableau "Analyse du registre ALPHA et de la pile" est un outil indispensable pour suivre pas à pas le fonctionnement des programmes synthétiques. Vous comprendrez toute sa puissance après l'avoir utilisé pour suivre le déroulement de SK et RK.

Lorsque vous exécutez SK, le numéro k est tout d'abord stocké en

Grille d'analyse de la pile et du registre ALPHA

LINE	INSTRUCTION	L	X	Y	Z	T	P	O	N	M
53	LBL"SK"		x	y	z	t				
54	SIGN	x	1							
55	CLX		0							
56	X<>⊦		H							
57	XEQ 14									
62	LBL 14									
63	"*"						Cleared	Cleared	Cleared	*
64	X<> M		------*							1 2 3 4 5 6 7
65	STO N								*	
66	ASTO IND L									
67	RDN		y	z	t	------*				
68	RTN									
58	ISG L	x+1								
59	"" (NOP)									
60	.		0	y	z	t				
61	X<> e		e							
62	LBL 14									
63	"*"						Cleared	Cleared	Cleared	*
64	X<> M		------*							8 9 10 11 12 13 14
65	STO N								*	
66	ASTO IND L									
67	RDN		y	z	t	------*				
68	RTN									
84	LBL"RK"		x	y	z	t				
85	SIGN	x	1							
86	ARCL IND L									* 1 2 3 4 5
87	"⊦⁻"									* 1 2 3 4 5 ⁻
88	ISG L	x+1								
89	"" (NOP)									
90	ARCL IND L								* 1 2 3 4 5	⁻ * 8 9 10 11 12
91	"⊦Φ"								* 1 2 3 4 5 ⁻ *	8 9 10 11 12 0 FF
92	X<> N		⊦'							
93	STO ⊦									
94	X<> M		e'							1 2 3 4 5 ⁻ *
95	STO e									
96	RDN		y	z	t	e'				
97	CLA						Cleared	Cleared	Cleared	
98	RTN									

Grille d'analyse de la pile et du registre ALPHA

LINE	INSTRUCTION	L	X	Y	Z	T	P	Q	N	M

LAST X par la fonction SIGN. Puis une instruction X<> ⊢ est utilisée pour extraire le contenu de ⊢ et le remettre à zéro. Le sous-programme au LBL 14 utilise la fonction ASTO pour stocker une chaîne de six caractères dans le registre k. Cette chaîne consiste en une étoile suivie des cinq premiers octets de l'ancien contenu du registre ⊢. L'étoile est indispensable au cas où le premier octet de ⊢ serait zéro. La séquence "*", X<> M, STO N prépare le registre ALPHA pour une instruction ASTO, comme vous pouvez le voir dans le tableau d'analyse du registre ALPHA et de la pile. Etudiez avec soin le fonctionnement de cette séquence si vous voulez réaliser vos propres programmes synthétiques.

Le reste de la routine SK effectue une opération similaire, extrayant le contenu de e et le remettant à zéro, et stockant une chaîne similaire dans le registre de données k+1.

Lorsque vous exécutez RK, le registre de données k est placé en LAST X par la fonction SIGN. Puis la chaîne de six caractères est "ARCLée" du registre k et décalée d'un octet vers la gauche par ajout d'un nul, bien qu'une étoile aurait aussi bien fait l'affaire. Puis c'est au tour du registre k+1 d'être "ARCLé", décalant la chaîne précédente de six caractères de plus vers la gauche. Deux octets supplémentaires, OF et FF sont ajoutés, causant encore un décalage de deux octets vers la gauche. L'analyse du registre ALPHA montre tous ces mouvements dans le détail.

A cet endroit, le registre N contient les sept octets à stocker dans ⊢, tandis que M contient les octets de e. Les dernières lignes de RK extraient les contenus de N et M, les stockent dans ⊢ et e, et vident ALPHA et la pile. Remarquez que les deux derniers octets de e sont OF et FF, forçant le calculateur à rechercher un numéro de ligne correct. Des versions antérieures de RK stockaient 00 00 dans les deux octets de droite de e, d'où un numéro de ligne incorrect si le programme était listé avec SST ou exécuté pas à pas (mode TRACE).

6C. Application No 2 des registres d'état : renumérotation des registres

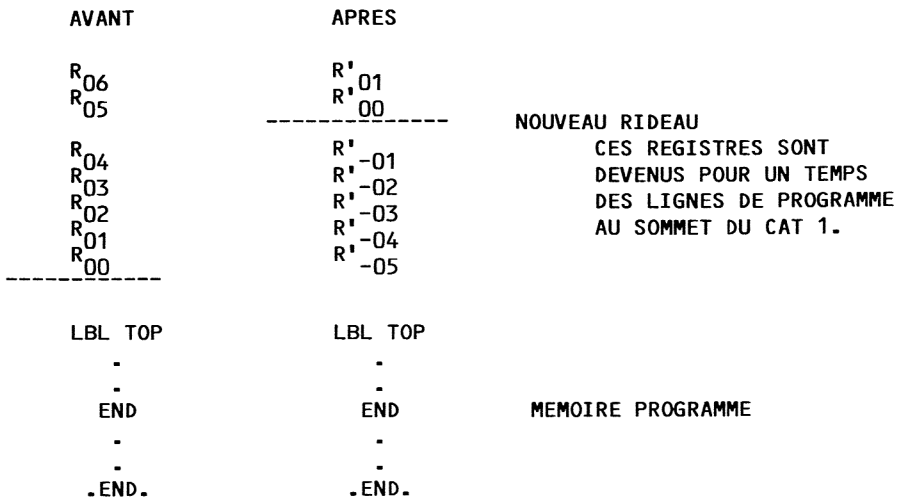
Supposez que vous ayez un programme appelant un autre programme comme sous-programme. Un exemple classique est un programme cherchant les racines d'une équation f(x)=0. Dans ce cas f(x) est calculée par un programme écrit par l'utilisateur. Celui-ci donne le nom du programme de calcul de f(x), que le programme appelant stocke dans un registre de données et appelle, quand besoin est, avec une instruction XEQ IND nn.

En écrivant un tel programme chercheur de racines, vous devez prendre une grave décision. Le programme utilisera quelques registres pour stocker ses données, et il est essentiel que le programme appelé ne modifie pas ces données. Quels que soient les registres que vous avez choisi, il existera toujours une possibilité de conflit entre les programmes appelant et appelé. Vous pouvez essayer d'utiliser les registres 50 et au-dessus pour chercher les racines, en pensant que la définition de la fonction n'utilisera pas ces registres. Mais, même si cela est vrai, vous perdez beaucoup de place. Dans beaucoup de cas, la fonction f(x) utilisera beaucoup moins de 50 registres de données.

La programmation synthétique permet de se sortir de ce mauvais pas. Un court programme synthétique peut déplacer le rideau séparant les registres de données de la mémoire programme, renumérotant ainsi les registres de données.

Par exemple, supposons que le programme cherchant les racines utilise les cinq registres de données 00 à 04. Juste avant d'appeler le programme calculant f(x), le programme appelant appelle la routine synthétique CU (**C**urtain **U**p : lever le rideau) pour monter le rideau de cinq registres. La

figure ci-dessous montre le résultat de cette opération. Bien que le contenu des registres n'ai pas changé, un RCL 00 rappellera maintenant le contenu de ce qui était le registre de données 05.



De même, une instruction RCL 01 retournera le contenu de ce qui se trouvait dans le registre 06. Les registres importants que le programme chercheur de racines devait protéger sont maintenant inaccessibles via les instructions STO et RCL. Les contenus des anciens registres 00 à 04 sont considérés comme une partie de la mémoire programme par le calculateur. En fait, si vous alliez au début du premier programme en mémoire, vous y trouveriez vos données. Bien sûr, elles apparaîtraient comme des instructions et non comme des nombres.

Le point important est qu'après avoir monté le rideau de cinq registres, le chercheur de racines peut appeler la fonction f(x) sans craindre de voir ses données essentielles modifiées. Le programme f(x) pourra avoir libre accès à ce qu'il pense être les registres 00 et au-delà.

Lorsque le programme f(x) rend le contrôle au programme appelant, celui-ci redescend le rideau à sa position primitive. Ceci permet de retrouver les registres originaux et d'accéder de nouveau aux contenus des registres 00 à 04.

Les listages donnés plus loin, des programmes CU et d'un classique chercheur de racines baptisé SOLVE illustrent les principes dont nous venons de discuter. Cette version de CU a été écrite par Tapani Tarvainen, et présente de sensibles améliorations par rapport aux versions plus anciennes.

Données pour LB du programme CU :

```
Ligne 03 : 144, 125      Ligne 04 : 145, 117
Ligne 05 : 245, 127, 0, 0, 0, 33
Ligne 08 : 206, 117      Ligne 09 : 206, 126      Ligne 10 : 145, 119
Ligne 13 : 170, 245      Ligne 15 : 240           Ligne 21 : 168, 245
Ligne 22 : 151, 117      Ligne 27 : 206, 119      Ligne 28 : 206, 126
Ligne 29 : 145, 117      Ligne 30 : 244, 127, 0, 0, 0
Ligne 31 : 206, 118      Ligne 32 : 206, 125
```

```
01♦LBL "SOLVE"
   02 "FNAME?"
   03 AON
   04 STOP
   05 ASTO 00
   06 AOFF
 07 "XGUESS1?"
   08 PROMPT
   09 STO 03
 10 "XGUESS2?"
   11 PROMPT
   12 -
   13 STO 02
   14 RCL 00
   15 LASTX
   16 XEQ 14
   17 STO 01
```

```
18♦LBL 10
   19 RCL 00
   20 RCL 03
   21 XEQ 14
   22 ENTER↑
   23 ENTER↑
   24 X<> 01
   25 -
   26 /
   27 RCL 02
   28 *
   29 CHS
   30 STO 02
   31 RCL 03
   32 +
   33 STO 03
   34 RCL 01
   35 ABS
```

```
   36  E-6
   37 X<=Y?
   38 GTO 10
   39 RCL 03
   40 BEEP
   41 RTN

   42♦LBL 14
   43 4
   44 XEQ "CU"
45 XEQ IND Y
   46 4
   47 CHS
   48 XEQ "CU"
   49 END

LBL'SOLVE
END

     97 BYTES
```

```
01♦LBL "CU"
   02 INT
   03 RCL c
   04 STO [
   05 "├♦♦♦!"
   06 RDN
   07 11
   08 X<> [
   09 X<> d
   10 STO ]
   11 RDN

   12♦LBL 03
13 FS?C IND [
   14 ISG X
   15 ""
   16 2
   17 /
   18 ENTER↑
   19 FRC
   20 X≠0?
   21 SF IND [
   22 DSE [
   23 ABS
   24 -
   25 X≠0?
   26 GTO 03
   27 X<> ]
   28 X<> d
   29 STO [
   30 "├♦♦♦"
   31 X<> \
   32 X<> c
   33 RDN
   34 CLA
   35 END
LBL'CU
END          67 BYTES
```

Les codes-barres des programmes CU et SOLVE sont fournis à l'appendice E. La routine SOLVE demande le nom du programme définissant la fonction f(x) et les deux estimations de la racine, c'est-à-dire la valeur x pour laquelle f(x)=0. Le programme SOLVE utilise ensuite la méthode de Newton pour trouver la racine. Pour ce faire, il évalue f(x) en plusieurs points. Chaque évaluation de f(x) utilise la sous-routine au label 14 qui monte le rideau de 4 registres, appelle f(x) puis abaisse le rideau de 4 registres pour retrouver sa position initiale.

La routine CU monte le rideau du nombre de registres spécifiés en X. Si ce nombre est négatif, le rideau est descendu. Les deux registres Y et Z de la pile sont préservés et finissent en X et Y. Cette caractéristique est utilisée par SOLVE pour sauvegarder le nom de la fonction et la valeur courante de x dans la pile. Une instruction XEQ IND Y est alors suffisante pour appeler la fonction f(x) avec la valeur correcte de x.

Pour essayer le couple SOLVE/CU, faites GTO.. et entrez :

```
01 LBL TEST
02 1/X
03 LAST X
04 -
05 1
06 +
```

Ce court programme calcule f(x)=(1-x)-x+1. En vous référant au problème 2.4, vous confirmerez que la solution de f(x)=0 est x=1+1/x qui est le nombre d'or.

XEQ SOLVE et répondez aux questions posées :

FNAME?	TEST (R/S)
XGUESS 1?	1 (R/S)
XGUESS 2?	2 (R/S)

Après environ 40 secondes, vous entendrez un BEEP et verrez le résultat : 1.618033989. Cet exemple ne démontre pas pleinement les possibilités du couple SOLVE/CU, mais vous pouvez être sûr que SOLVE et CU marcheront comme il faut pour n'importe quelle fonction f(x), sans risque de conflit. Bien sûr, les limitations de la méthode de Newton sont toujours valables. Certaines fonctions instables peuvent poser des problèmes, de même que des estimations très mauvaises. Mais dans la grande majorité des fonctions réelles, ces programmes fonctionnent rapidement et correctement.

Contraintes associées à l'utilisation de CU

1.) Pendant que le rideau est levé, des registres de données sont temporairement devenus des lignes de programme au début de la mémoire. Certains de ces pas de programmes peuvent être des labels. Vous ne devez donc pas exécuter un branchement en arrière dans le premier programme en mémoire lorsque le rideau est levé.

2.) Ne PACKez pas la mémoire avec le rideau levé. Il est plus que probable que les registres protégés contiennent des octets nuls qui seraient détruits par le compactage. Vous pouvez en partie empêcher cette destruction en exécutant PACK avant de monter le rideau. De cette façon, le processeur pense que le premier programme en mémoire est déjà compacté. Assurez-vous également que plusieurs registres sont disponibles (derrière le .END.) avant d'exécuter CU. Ainsi, si vous insérez une instruction dans

un programme, faites un assignement ou stockez une alarme, vous n'obtiendrez pas un PACKing inopportun.

3.) Replacez toujours le rideau à sa position première. C'est une bonne habitude à prendre. Si vous laissez accidentellement le rideau levé, vous devrez aller dans votre premier programme en mémoire, effacer les instructions inutiles du sommet (c'est-à-dire vos données protégées) et faire PACK pour ramener le programme sous le nouveau rideau.

4.) Ne placez pas le rideau juste au-dessus d'un registre inexistant. Ainsi, si vous placez le rideau à l'adresse 16 décimal, c'est tout bon car le registre 15 (le registre e) existe. Mais si vous placez le rideau à l'adresse 17, vous obtenez MEMORY LOST car il n'existe pas de registre à l'adresse 16. Vous pouvez toutefois éviter le MEMORY LOST si vous replacez le rideau à une adresse valide avant que le programme ne s'arrête (MK et LB font cela), mais vous avez tout intérêt à savoir ce que vous faites.

A l'aide du programme CU, vous n'êtes pas limité à un seul programme renumérotant les registres avant d'en appeler un autre. Ce second programme peut lui aussi renuméroter les registres avant d'en appeler un troisième. Ce processus peut être poursuivi à loisir, créant une pile de données à plusieurs niveaux. La séquence permettant à chaque programme de protéger ses données conservées dans les registres 0 à k-1 avant d'appeler un sous-programme est:

```
k
XEQ CU
XEQ sous-programme
-k
XEQ CU
```

La renumérotation à l'aide du rideau donne beaucoup de flexibilité aux programmes. Par exemple, un programme utilisant les registres de données 10 à 19 peut être utilisé en SIZE 010 seulement. Vous n'avez qu'à baisser le rideau de 10 registres avant d'exécuter le programme, transformant ainsi les registres 00 à 09 en registres 10 à 19. N'oubliez pas de replacer le rideau où il se trouvait juste après exécution du programme, sinon un RCL 00 malencontreux pourrait détruire une partie de vos programmes.

Le programme CU de Tapani Tarvainen est fonctionnellement équivalent au programme CU du PPC ROM écrit par Bill Wickes. Si la vitesse est un critère important, vous devez savoir que CU 'tourne' nettement plus vite que CU. Dans le ROM se trouvent également les programmes de contrôle de rideau HD, UD et <C qui sont beaucoup plus rapides. Ces trois programmes ont des limitations que vous devez impérativement connaître avant de les utiliser. Le manuel d'utilisation du PPC ROM contient des informations très utiles dans les descriptifs de CU , HD , UD et <C . L'appendice M du manuel contient encore davantage d'informations essentielles sur les mouvements de rideau.

Fonctionnement du programme CU

Le contenu du registre c est tout d'abord placé dans la portion la plus à droite du registre ALPHA. Puis la ligne 05 ajoute quatre octets. A cet endroit, le registre M qui comprend les sept derniers caractères de ALPHA, contient les trois derniers octets de c, suivis de trois nuls et

d'un octet de code 21 hexadécimal. Le pointeur du rideau réside dans le premier octet et demi de M.

Puis M est manipulé avec les drapeaux. Le pointeur du rideau est codé maintenant par les drapeaux 0 à 11. En fait, les drapeaux 0 et 1 sont toujours baissés, puisque l'adresse du rideau est toujours inférieure ou égale à 512 = 001000000000 en binaire. Les drapeaux originaux sont sauvegardés dans le registre 0 pour être restaurés plus tard, tandis que le nombre 11 est mis en M pour une utilisation future comme index de boucle.

Le mystérieux octet 21 hexa lève les drapeaux 50 et 55. Le drapeau 50 empêche les messages affichés de se déplacer (voir l'exemple 6 du programme IF dans le manuel du PPC ROM). Le drapeau 55 doit être levé pour permettre à CU d'être interrompu ou exécuté pas-à-pas avec une imprimante présente. Si ce drapeau 55 était baissé, les deux drapeaux 21 et 55 seraient levés lors d'une interruption, altérant la portion du registre des drapeaux dans laquelle on a placé l'adresse du .END.

La boucle au LBL 03 réalise une addition binaire dans le registre des drapeaux en utilisant l'élégant algorithme de Tapani. Le nombre binaire représenté par les drapeaux 0 à 11 est converti en décimal, puis ajouté à l'incrément décimal (le nombre de registres dont le rideau doit être déplacé). La somme résultante est reconvertie en binaire et placée dans les drapeaux 0 à 11.

Ce qui rend unique ce programme de Tapani est que la conversion d'un décimal en binaire est faite sur chaque bit avant de passer au suivant. A chaque boucle, un bit de l'adresse courante du rideau est remplacé par le bit conforme à sa nouvelle adresse. Détaillons ce processus pour le bit de poids faible, au premier passage dans la boucle au LBL 03.

Lorsque ce LBL 03 est rencontré la première fois, X contient l'incrément que vous avez demandé. Les lignes 13 et 14 baissent le drapeau 11, le bit "1" de l'adresse du rideau, et ajoute 1 à X si le drapeau 11 est levé. Cela convertit effectivement le bit du drapeau 11 en décimal, en l'ajoutant à X. Le bit du drapeau 11 de la nouvelle adresse sera mis à 1 si et seulement si le nombre maintenant dans X est impair. Si vous ne voyez pas pourquoi, pensez que la nouvelle adresse du rideau est la somme du nombre en X et du nombre binaire représenté par les drapeaux 0 à 11. Comme le drapeau 11 est baissé, ce nombre binaire est divisible par 2. Donc la somme n'est impaire et le drapeau 11 ne doit être levé que si X est impair.

Les lignes 15 à 24 reviennent à lever le drapeau 11 et soustraire 1 de X si X est impair, ou laisser le drapeau 11 baissé et diviser X par 2. Cette division donne un résultat entier puisque l'on s'est assuré que X est pair. L'index des drapeaux est décrémenté d'une unité pour le prochain passage dans la boucle. Le drapeau 11 a donc l'état correct, levé si X était impair, pour la nouvelle adresse du rideau. Les lignes 25 et 26 font progresser l'addition vers le bit de poids immédiatement supérieur si l'incrément n'est pas encore égal à zéro.

Au second passage dans la boucle, le nombre binaire n'est plus long que de onze bits (les drapeaux 0 à 10). Nous devons diviser X par 2 pour qu'il soit un équivalent décimal compatible avec le nouveau bit de poids 1, c'est-à-dire le drapeau 10. Le nombre en X ne représente plus simplement l'incrément voulu pour l'adresse du rideau. Il contient maintenant une composante correspondant à une retenue, si elle était nécessaire, du bit précédent.

Cette fois, le drapeau 10 est baissé et transféré en X, puis levé si et seulement si X est impair. Une fois de plus, X est rendu pair et divisé par 2 pour le prochain passage. Ce processus se répète jusqu'à ce que X soit nul, du fait des divisions successives.

Remarquez qu'il n'a été nulle part question de savoir si X était

positif ou négatif. CU fonctionne de la même manière dans les deux cas. Lorsqu'un drapeau est baissé, X est incrémenté. Lorsqu'un drapeau est levé, X est décrémenté. A chaque passage dans la boucle, X est divisé par 2 jusqu'à ce qu'il devienne éventuellement nul.

Les lignes 27 à 29 extraient le contenu du registre des drapeaux et le place dans le registre M, restaurant la configuration originale des drapeaux et plaçant les trois derniers octets modifiés de c, à côté des quatre premiers octets de c qui occupent toujours les quatre octets de droite de N. Le registre ALPHA est décalé de trois octets vers la gauche à l'aide d'une instruction APPEND. Les sept octets du nouveau registre c se trouvent maintenant dans le registre N. Ils en sont extraits et sont stockés en c. L'instruction X<> c est employée au cas où vous souhaiteriez retrouver plus tard l'ancienne position du rideau en exécutant simplement STO c. Bien sûr, vous devrez tout d'abord trouver l'ancien contenu de c dans la pile, s'il s'y trouve encore.

Les quelques lignes restantes effacent le contenu du registre ALPHA pour laisser place nette, et remettent en ordre la pile. Les anciens Y et Z finissent en X et Y ; Z contient l'ancien contenu de c, et T contient zéro.

Relisez cette explication jusqu'à ce que vous compreniez bien. Il peut être utile de charger la pile en faisant 4 ENTER↑ 3 ENTER↑ 2 ENTER↑ 1 et GTO "CU". Assurez-vous que le SIZE est au moins 001. Puis vous pouvez exécuter CU en mode pas-à-pas pour suivre ce qui se passe dans ce cas simple où l'on doit monter le rideau d'un registre.

Ne vous affolez pas si une petite ou une grande partie de ce chapitre vous ont paru hors de votre portée à la première lecture. Après tout, c'est pour cela que je les ai mises à la fin de ce livre. Rappelez-vous qu'il s'est écoulé deux ans entre les débuts de la programmation synthétique et la découverte de la méthode permettant d'assigner le Byte Grabber à une touche. Il y a certainement encore beaucoup à découvrir de votre HP-41. Peut-être bien que vous serez celui qui en découvrira une partie.

Deuxième chapitre :

2.1 Voici une version possible de CQ :

```
01 LBL CQ                 Données pour LB / MK :
02 RAD
03 CLX
04 TONE 8
05 TONE P                 159, 120
06 TONE 8
07 TONE P                 159, 120
08 SIN
09 TONE 8
10 TONE 8
11 TONE P                 159, 120
12 TONE 8
13 END
```

2.2 Entrez:

```
01 ENTER↑
02 1E1
```

GTO.001, entrez RDN, BG et deux fois la flèche de correction. Vous avez maintenant E1 à la ligne 02. Tapez ensuite STO 28, PACK, BST, BG et flèche de correction. Le PACKing a placé le suffixe 28 juste adjacent à l'instruction E1, en enlevant les nuls. Lorsque le préfixe STO est dérobé, le suffixe 28 devient une instruction NEG et est incorporé à l'instruction E1 adjacente.

Les données pour LB pour créer -E1 sont 28, 27, 17.

2.3

```
01 LBL "VX"                    Données pour LB / MK :
02 "  " (2 espaces)
03 RCL d                  144, 126
04 SCI 9
05 ARCL Y  (pas X car la pile est montée du fait du RCL d)
06 STO d                  145, 126
07 RDN
08 AVIEW
09 END
```

Dans des cas comme celui-ci, vous devez prendre l'habitude de faire AVIEW après le STO d, et non pas avant. Cela évite d'altérer les drapeaux du système. Dans ce cas particulier, l'affichage reviendra à la normale (le nombre montré par AVIEW disparaîtra) à la fin de l'exécution si AVIEW est exécuté avant, car STO d baisse le drapeau 50, le drapeau signalant un message à l'affichage.

2.4 Voici une solution à la recherche du nombre d'or:

```
01 LBL "OR"               Données pour LB / MK :
02 FIX 9
03 E                      27 ou 27,  0
04 RCL b                  144, 124
05 X<>Y
06 1/X
07 E                      27 ou 27,  0
08 +
09 X<>Y
10 VIEW Y
11 STO b                  145, 124
```

Ce programme converge vers une solution à dix chiffres en 8 secondes

2.5 a)

```
01 LBL "PX"               Données pour LB :
```

```
02 FIX 0
03 CF 29
04 "X("                          242,  88,  40
05 ARCL 00
06 "├)=?"                  244, 127,  41,  63
07 PROMPT
```

Pour obtenir les chaînes synthétiques à l'aide du Byte Grabber, entrez : 01 ENTER↑
02 "XX"
03 "├X=?"

GTO.002, BG, GTO.005, flèche de correction, RCL 09, GTO.002, BG, DEL 002, GTO.001, BG, GTO.004, flèche de correction, RCL 08, GTO.001, BG, DEL 002, flèche de correction et entrez les lignes non synthétiques.

b) Pour préserver le format d'affichage, insérez RCL d et STO d comme ci-dessous :

```
01 LBL PX                  Données pour LB / MK :
02 RCL d                        144, 124
03 CF 29
04 FIX 0
05 "X("
06 ARCL 00
07 "├)=?"
08 STO d                        145, 124
09 RDN
10 PROMPT
```

Il est possible d'économiser un octet en remplaçant les lignes 2 et 3 de ce programme par : 02 . (point décimal)

```
03 X<> d  206, 126
```

Cette séquence stocke 0 dans le registre des drapeaux, baissant d'un coup les 56 flags. Nous n'avons donc plus qu'à faire FIX 0 pour obtenir l'état voulu des drapeaux 29 et 36-41 (codant le nombre de chiffres et le type FIX, ENG, SCI). L'ancien contenu de d est en X comme dans le cas précédent, prêt à être replacé dans d par l'instruction STO d. Pour obtenir l'instruction X<> d avec le Byte Grabber, commencez par entrer STO IND 78 suivie de AVIEW. Dérober le préfixe STO et effacez la chaîne créée. Le suffixe IND 78 devient X<> et AVIEW se transforme en suffixe d.

2.6
```
01 LBL "OX"                Données pour LB :
02 RCL d                        144, 126
03 FIX 2
04 "OUT="
05 ARCL Y
06 STO d                        145, 126
07 RDN
08 "├μV"                   243, 127,  12,  86
```

La ligne 08 peut être obtenue à l'aide du Byte Grabber comme suit :
01 ENTER↑
02 "├XV"

GTO.001, BG, GTO.004,flèche de correction, LBL 11, GTO.001, BG, DEL 002, flèche de correction.

2.7
```
LBL "CMOD"                 Données pour LB / MK :
02 X<>Y
03 STO M                        145, 117
04 X<>Y
05 MOD
06 ST- M                        147, 117
07 LAST X
```

```
08 ST/ M                149, 117
09 CLX
10 X<>M                 206, 117
```

Les lignes 01 à 04 sauvegardent y en M et x en L. Puis y MOD x est soustrait au contenu de M. Les lignes 07 à 10 divisent le contenu de M par X, rappellent M en X, et effacent M.

Troisième chapitre :

3.1 GTO.. et entrez LBL ++ suivi d'au moins 45 +, puis XEQ "LB". Sortez du mode PRGM, R/S, et répondez aux demandes comme suit :

Demande	Réponse	Demande	Réponse
1?	27 R/S	1?	1 R/S
2?	145 R/S	2?	4 R/S
3?	119 R/S	3?	5 R/S
4?	146 R/S	4?	6 R/S
5?	119 R/S	5?	242 R/S
6?	206 R/S	6?	127 R/S
7?	119 R/S	7?	96 R/S
1?	145 R/S	1?	154 R/S
2?	117 R/S	2?	118 R/S
3?	150 R/S	3?	152 R/S
4?	117 R/S	4?	118 R/S
5?	240 R/S	5?	R/S
6?	153 R/S		
7?	245 R/S		
1?	152 R/S		
2?	119 R/S		
3?	172 R/S		
4?	245 R/S		
5?	159 R/S		
6?	106 R/S		
7?	244 R/S		

Lorsque le programme s'arrête, vous pouvez presser SST pour revenir sur le LBL ++ et voir vos instructions synthétiques.

3.2 Voici un programme non synthétique simple permettant de calculer les données à fournir à LB pour créer des XROM. Ce programme tire son algorithme du fait que 64*(i MOD 4) est égal à 256*FRC(i/4). Vous trouverez à droite les évolutions de la pile en cours de programme. Lorsqu'il n'y a rien d'écrit, le contenu du registre n'a pas changé par rapport à la ligne précédente.

LBL "XRLB"	L	X	Y	Z	T
X<>Y		i	j	z	t
4		4	i	j	z
/	4	i/4	j	z	z
INT	i/4	INT(i/4)			
X<>Y		j	INT(i/4)		
LAST X		i/4	j	INT(i/4)	z
FRC		FRC(i/4)			
256		256	FRC(i/4)	j	INT(i/4)
*	256	64(i MOD 4)	j	INT(i/4)	
+	64(i MOD 4)	octet 2	INT(i/4)		
X<>Y		INT(i/4)	octet 2		
160		160	j	octet 2	
+	160	octet 1	octet 2	INT(i/4)	INT(i/4)
END					

Pour utiliser XRLB, entrez i ENTER↑ j, puis XEQ "XRLB". Le résultat obtenu dans le registre X est l'octet No 1 en décimal. L'octet No 2 se trouve dans le registre Y.

Voici une version synthétique de XRLB qui ne perturbe pas les registres Z et T. Vous trouverez à droite les contenus importants des registres d'état et de la pile chaque fois que ceux ci changent en cours de programme.

LBL "XRLB"	**N**	**M**	**L**	**X**	**Y**	**Z**	**T**
STO M		j		j	i	z	t
RDN				i	z	t	j
4				4	i	z	t
/			4	i/4	z	t	t
STO N	i/4						
FRC			i/4	FRC(i/4)			
256				256	FRC(i/4)	z	t
*			256	64(i MOD 4)	z	t	t
RCL M				j	64(i MOD 4)	z	t
+			j	octet 2	z	t	t
160				160	octet 2	z	t
ST+ N	160+i/4						
X<> N	160			160+i/4			
INT			160+i/4	octet 1	octet 2	z	t
CLA	0	0					
END				octet 1	octet 2	z	t

3.3 Introduisez au moins 17 + et exécutez LB. Les 7 entrées de données sont : 207, 120, 159, 37, 208, 0, 120.

3.4 Introduisez au moins 31 + et entrez les valeurs décimales 192, 0, 255, 0, 82, 80, 78, 32, 67, 65, 76, 67, 85, 76, 65, 84, 79, 82. Faites PACK pour incorporer ce nouveau label global au chaînage du CATalogue 1.

Comme ce label fait plus de 6 caractères, il ne peut pas faire l'objet d'une instruction GTO IND ou XEQ IND.

3.5 Les entrées appropriées pour LB sont : 144, 124, 206, 117, 206, 118, 145, 117, 206, 117, 206, 125, 145, 125, 242, 127, 0, 206, 125, 144, 117, 145, 125.

Quatrième chapitre :

4.2 Les équivalents décimaux requis sont : 244, 127, 0, 0, 2, 27, 20, 206, 125, 145, 125, 242, 127, 0, 206, 125, 145, 125. GTO.. et entrez LBL LB. Passez en mode RUN, faites CLA, 125, XTOA, 145, XTOA, 125, XTOA, 206, XTOA, 0, XTOA, 127, XTOA, 242, XTOA. GTO "LB", RCL M, STO Q, passez en mode PRGM, Q-LOAD, BG, et deux fois la flèche de correction.

Revenez en mode RUN et faites CLA, 125, XTOA, 145, XTOA, 125, XTOA, 206, XTOA, 20, XTOA, 27, XTOA. GTO "LB", RCL M, STO Q, et passez en mode PRGM. Il n'est pas nécessaire de PACKer, car l'octet 242 ne fait pas partie de l'instruction précédente. Aussi n'est il pas nécessaire qu'il y ait une liaison avec les nouveaux octets. Toujours en mode PRGM sur le LBL "LB", Q-LOAD, BG et deux fois la flèche de correction.

Continuez avec CLA, 2, XTOA, 0, XTOA, 0, XTOA, 127, XTOA, 244, XTOA. GTO LB, RCL M, STO Q, passez en mode PRGM, Q-LOAD, BG et deux fois la flèche de correction. Le fait de ne pas inclure l'octet décimal 2 dans le second groupe d'octets chargés nous a permis d'éviter de compacter la mémoire avant de charger le troisième groupe. En outre, ceci était indispensable du fait de cette impossibilité qu'a le Q-Loader de charger des nuls en fin de chaîne. Nous n'aurions pas pu charger correctement la séquence hexa F4 7F 00 00 toute seule.

4.3 a) XROM 61,25

b) XROM 57,26
c) XROM 27,54

Cinquième chapitre :

5.1 Les séquences d'octets sont données ci-dessous en hexadécimal.

a) 40, 47, 48, 00, 00, 00, 00, 00, 00, 13, 41, 00, 14, 25, 15, 42. Il y avait de la place pour Σ+ (47 hexa), mais Σ- avait ouvert 7 octets. RCL 05 a remplacé le nul qui se trouvait déjà présent entre les deux entrées numériques 4 et 5.

b) 40, 41, E0, 00, 00, 92, 4B, 00, 42, 43. L'instruction ST+ 75 utilise deux des trois octets occupés auparavant par GTO 99.

APPENDICE A : TEMPS D'EXECUTION

En lisant le second chapitre, vous vous êtes peut-être demandé comment l'on pouvait savoir que l'instruction synthétique E était plus rapide à l'exécution que l'entrée numérique 1 ordinaire, ou que le point décimal s'exécute plus vite que le chiffre 0. Du temps de la HP-67, ces résultats étaient obtenus en introduisant une suite de 100 ou plus instructions identiques, puis en mesurant le temps mis pour exécuter cette séquence, et enfin diviser le résultat par le nombre d'instructions entrées. Il est inutile de préciser que cette méthode était à la fois longue et laborieuse.

La programmation synthétique permet l'automatisation de l'entrée de centaines de copies d'une instruction particulière (ou même de copies d'une courte séquence d'instructions). La séquence appropriée est créée, puis stockée, par groupes de 7 octets, dans des registres contigus. Ces octets peuvent ensuite être exécutés comme des instructions en plaçant le code convenable dans le registre de pointeur programme.

Le module horloge HP 82182A permet de mesurer automatiquement la durée d'exécution d'une séquence ou d'instructions stockées de façon synthétique. Clifford Stern a écrit un programme synthétique utilisant le module horloge pour mesurer le temps d'exécution d'un groupe arbitraire d'instructions allant de un à sept octets. Le programme crée et stocke autant de copies de ce groupe qu'il est possible d'en mettre dans la mémoire inutilisée. Puis il exécute ensuite la suite complète de tous les groupes, mesure le temps écoulé, divise le résultat par le nombre de groupes et affiche le temps résultant pour l'exécution d'un groupe.

La table A.1 donne des résultats moyens pour les temps d'exécution des fonctions. Ne sont détaillées que les fonctions pour lesquelles une alternative est possible. Si vous avez besoin de la fonction LOG, il n'est pas nécessaire de savoir combien de temps elle prend, car vous n'avez pas d'autre méthode plus rapide pour calculer le logarithme. Mais, pour incrémenter un registre, il peut vous être utile de savoir que la séquence E, + demandant 78,7 millisecondes est nettement plus lente que la séquence ISG X, TEXT 0 qui ne réclame que 74 millisecondes. Si vous avez besoin de vitesse d'exécution, vous voudrez peut-être utiliser plus de place en mémoire pour l'obtenir. On peut tirer d'autres conclusions de la table des durées :

- R↑ R↑ est plus rapide que RDN RDN ;
- X<> est plus rapide que RCL mais plus lent que STO ;
- Les opérations sur les registres d'état sont toujours plus rapides que celles sur les registres de données ;
- Les GTOs compilés sont très rapides, les XEQs un peu plus lents ;
- Les entrées de données numériques sont très lentes. Ceci est dû au

fait que les registres P et Q doivent être chargés avant X ;

- Pour une exécution plus rapide, employez E plutôt que 1 et le point décimal plutôt que zéro. Notez que CLX, SIGN est une méthode encore plus rapide pour obtenir 1 ;

- Pour une exécution plus rapide des nombres négatifs, utilisez un nombre positif suivi d'une instruction CHS, plutôt qu'une entrée contenant un signe négatif. Pressez ALPHA ALPHA pour terminer l'entrée du nombre positif, puis pressez CHS pour avoir l'instruction CHS à part. CHS est plus rapide que NEG (signe - d'une entrée numérique).

Ces résultats du programme estimant la durée d'exécution sont un autre exemple que la connaissance de la programmation synthétique peut améliorer votre technique générale de programmation.

Si vous avez un PPC ROM, un module d'extension de fonctions et un module horloge, vous pouvez utiliser le programme de Clifford Stern pour faire vos propres mesures. Voici le principe d'utilisation :

1) Assurez-vous qu'il y ait un END présent en mémoire au-dessus de ce programme (par CAT 1). Ceci est nécessaire pour permettre un fonctionnement correct des instructions GTO avec le rideau placé à l'adresse 10 hexa. Pour d'autres explications, se référer aux contraintes de l'utilisation du programme CU à la section 6C.

2) Baissez le drapeau 02 et mettez le SIZE au moins égal à 004. Effacez toutes les alarmes (vous pouvez utiliser le programme SA donné section 4E). Faites **maintenant** tous les assignements que vous désirez. **Ne faites jamais** aucun assignement (sauf un label global) après avoir commencé d'appliquer le paragraphe 3 et avant d'avoir terminé le paragraphe 9.

3) Introduisez le nombre de registres à utiliser pour stocker la séquence d'octets. Ce nombre doit être choisi comme étant un multiple exact du nombre d'octets par groupe d'instructions, les groupes de 1 ou 7 octets sont toujours bons. Par exemple, si le groupe est long de trois octets, le nombre de registres doit être multiple de trois. S'il n'est pas multiple du *nombre d'octets du groupe, vous pourrez obtenir DATA ERROR à la ligne 114.* Si vous donnez un multiple de 60 registres, cela marchera à tous les coups. *Faites XEQ "IN" pour initialiser avec ce nombre de registres. Le programme* ajustera le SIZE si cela est nécessaire, pour fournir le nombre de registres demandés comme registres disponibles sous le .END. Si malgré la modification du SIZE, la mémoire n'est pas suffisante pour donner les registres voulus, un message DATA ERROR apparaîtra à la ligne 49. Si cela se produisait, effacez un programme ou réduisez le nombre de registres réclamés, puis reprenez au début de ce paragraphe.

4) La procédure IN se prolonge automatiquement par le LBL "S", débutant la routine de stockage des instructions. Cette routine S vous demandera d'introduire un groupe de un à sept octets. Entrez un nombre décimal compris entre 0 et 255 pour chaque octet, puis pressez seulement R/S, sans donnée, pour indiquer la fin du groupe. Le groupe d'octets sera dupliqué et stocké dans le bloc de registres initialisés plus haut, se trouvant après le .END. et au-dessus des registres d'assignement.

5) La routine S se termine au LBL "T", drapeau 01 baissé. A cet endroit, la pile est vide. Vous êtes libres de la charger selon votre désir pour s'accorder avec la séquence choisie. Pressez R/S ou XEQ "T" pour commencer la mesure du temps d'exécution. Le résultat, exprimé en millisecondes par groupe d'octets, se trouvera dans le registre X lorsque la mesure sera terminée. S'il arrivait qu'une erreur arrête l'exécution au milieu de la séquence d'instructions, vous devriez alors faire GTO "S" et XEQ 10. Vous pourrez alors entrer une nouvelle séquence d'instructions en suivant les instructions du paragraphe 4, ou tout simplement réintroduire

un argument valide dans la pile et taper XEQ "T".

6) Pour relancer la mesure avec d'autres conditions de départ, rechargez la pile, puis faites XEQ "T" une fois de plus (**ne faites surtout pas R/S** voir paragraphe 9). Si vous souhaitez préparer les contenus du registre ALPHA et de la pile, levez les drapeaux 1 et 2 avant d'exécuter T. Le programme de comptage s'arrêtera pour vous permettre de charger le registre ALPHA (ou la pile). Remarquez que T peut être appelé comme sous-programme pour mesurer les durées de fonctions identiques avec des données de départ différentes.

7) Pour soumettre à la mesure un autre groupe d'instructions, faites de nouveau XEQ "S". Vous pouvez tout d'abord lever le drapeau 01 si vous souhaitez que la mesure se fasse avec une pile vide. Levez les deux drapeaux 01 et 02 si vous devez charger le registre ALPHA pour votre mesure.

8) Pour choisir un nombre différent de registres pour y stocker les instructions, introduisez le nombre voulu, puis XEQ "IN".

9) Pour effacer les registres de stockage après avoir fait vos mesures, pressez RTN et R/S, ou simplement R/S après avoir utilisé la routine LBL "T".

10) Trois autres routines utiles font partie de ce programme. Ce sont toutes des versions de la routine de stockage S qui, elles, ne réclament pas l'introduction des données.

Faites XEQ "1" avec un nombre décimal (entre 0 et 255) en X pour stocker une instruction d'un octet.

Faites XEQ "2" avec un nombre décimal en X pour stocker la séquence : instruction d'un octet, LAST X. Cette séquence est utile pour mesurer les durées de fonctions monadiques telles SIN ou LN.

Faites XEQ "3" avec un nombre décimal en X pour stocker la séquence : instruction d'un octet, X<> L. Ceci est utile pour mesurer les durées de fonctions diadiques comme + ou MOD. Initialisez en remplissant la pile avec l'argument y, puis en plaçant l'argument x dans le registre X et en exécutant "T".

Lorsque vous utilisez "2" ou "3", vous devez mesurer séparément les durées des instructions LAST X ou X<> L et soustraire ces valeurs à celles obtenues afin d'avoir la durée réelle de l'instruction que vous voulez mesurer.

Lorsque vous mesurez les durées d'entrées numériques, vous devez les séparer pour qu'elles ne tournent pas ensemble, sous forme d'une énorme instruction. Utilisez un nul ou LAST X, et décomptez le temps d'exécution du séparateur.

Les codes-barres du programme complet se trouvent à l'appendice E.

Le listage complet du programme est donné ci-après. Quelques lignes synthétiques ont des représentations ambigües à l'impression. Leurs équivalents décimaux sont donnés ci-dessous :

Ligne	**hexa**	**décimal**
10	F2 CE 74	242 206 116
14	F1 76	241 118
65	F7 A6 99 A6 93 6D 1C 85	247 166 153 166 147 109 28 133
68	F5 AC 02 84 A6 94	245 172 2 132 166 148

Les lignes 65 et 68 contiennent des caractères de contrôle de l'imprimante. Le caractère A6 hexa fait sauter 6 espaces ; AC hexa fait sauter 12 espaces.

Rappel des erreurs possibles :

Ligne 49 : DATA ERROR signifie que la mémoire disponible est

insuffisante pour allouer les registres de stockage demandés.

Ligne 114 : DATA ERROR signifie que le nombre d'octets par groupe n'est pas un diviseur du nombre de registres alloués par IN au stockage des séquences d'instructions.

Ligne 115 : NONEXISTENT signifie que vous avez essayé d'introduire un groupe de huit octets. Ce programme n'autorise que des groupes allant de un à sept octets.

Utilisation des registres de données par le programme :

R00 = Scratch (nombre de groupes d'instructions)
R01 = Adresse du rideau abaissé (temporairement placée en c)
R02 = Pointeur de retour de la séquence d'octets stockée
R03 = Nombre de registres de stockage

Si l'un des registres 01 à 03 était altéré, vous devriez réinitialiser (introduire le nombre de registres et XEQ IN).

Table A.1 : Temps d'exécution typiques (en millisecondes)

Opérations de pile :	
ENTER↑	11.7
X<>Y	10.3
RDN	16.9
R↑	12.0
CLX	9.8
LAST X	13.0
CLST	10.5
SIGN	13.3
CHS	12.5
RCL registre d'état	20.3
STO registre d'état	16.8
X<> registre d'état	19.7
CLA	9.5
Instructions diverses :	
LBL 00-14	10.6
LBL à deux octets	13.1
CLD	20.6
TEXT 0	12.3
AON, AOFF	19.0
ADV (sans imprimante)	9.2
BEEP (drapeau 26 levé)	1042.4
(drapeau 26 baissé)	14.9
DEG	19.8
RAD	19.9
GRAD	20.5
PSE	1333.2
Octet nul	5.7
Opérations sur les registres :	
STO 00-15	19.3
STO 16-99	20.6
STO registre d'état	16.8
STO IND 00-99	32.3
STO IND registre d'état	32.1
RCL 00-15	22.8
RCL 16-99	24.1
RCL registre d'état	20.3

RCL IND 00-99	35.7	
RCL IND registre d'état	35.6	
X<> 00-99	23.4	
X<> registre d'état	19.7	
X<> IND 00-99	35.1	
X<> IND registre d'état	35.0	
ST+ 00-99	38.9	
ST+ registre d'état	35.3	
ST- 00-99	40.8	
ST- registre d'état	37.3	
ST* 00-99	46.8	
ST* registre d'état	43.0	
ST/ 00-99	49.5	
ST/ registre d'état	45.8	
ISG X, TEXT 0 (avec saut)	73.2	(x = 1)
(sans saut)	74.4	(x = -1)
DSE X, TEXT 0 (avec saut)	72.9	(x = 1)
(sans saut)	74.0	(x = 2)

Entrées numériques :

0	69.7
1 à 9	59.8
.	61.8
E	53.6
- (NEG, rend négatif un nombre ou un exposant. Seul, place 0 en X.)	60.9

Instructions diverses à plusieurs octets :

GTO 00-14 compilé	17.3
GTO trois octets, compilé	24.5
XEQ compilé	35.2
LBL global:1 caractère	45.4
2 caractères	49.3
3 caractères	51.9

APPENDICE B : CODE MORSE ET STO b

L'idée de faire de la HP-41 une machine capable de générer un code Morse parfait revient à Richard Nelson (le fondateur de PPC). Son programme employait l'instruction synthétique TONE P, mais à cette époque, la programmation synthétique en était à ses débuts, aussi le principe de l'exécution reposait-il sur des techniques classiques. Le résultat conduisait à une transmission d'environ 6 mots par minute. En principe, la licence de radio-amateur vous demande de saisir 13 mots à la minute. Les méthodes conventionnelles sont donc nettement inadaptées pour produire du Morse à ce rythme.

Clifford Stern a écrit un programme qui apporte une solution au problème en utilisant toute la puissance de la programmation synthétique. Pour comprendre la technique utilisée, considérez tout d'abord la boucle suivante qui apparaissait dans une version plus ancienne de ce programme :

```
LBL 01
RCL IND L
XEQ IND X
ISG L
```

```
01 XROM "RF"
 02 AVIEW
03 XROM "LF"
04 XROM "OM"
 05 X<>Y
 06 ISG X
07 XROM "BC"
 08 GTO 13

09*LBL "3"
  10 "t"
11 3
12 GTO 01

13*LBL "2"
14 "v"
15 2
16 GTO 01

17*LBL "1"
18 CLA
19  E

20*LBL 01
21 STO a
22 ASTO X
23 CLA
24 AVIEW
25 CF 29
26 FIX 0
27 X<>Y
28 XTOA
29 ARCL Y
      65 "
  68 "
 30 GTO 16

 31*LBL "IN"
 32 STO 03
33 XROM "F?"
 34 INT
 35 ENTER↑
36 XROM "E?"
 37 X<>Y
 38 -
 39 STO 01
 40 SIZE?
 41 ENTER↑
 42 R↑
 43 +
 44 RCL 03
 45 -
 46 7
 47 -
 48 X<0?
 49 SQRT
 50 4
 51 +
 52 X<Y?
 53 PSIZE
54 XROM "OM"
 55 R↑
 56  E
 57 +
58 XROM "CX"
 59 X<> c
 60 RCL 03
 61  E
 62 +
 63 X<>Y
 64 X<> c
            ne"
 66 RCL [
 67 STO 00
    x̄
           "
69 ASTO IND Z
 70 RDN
 71 X<> c
 72 X<> 01
 73 +
 74 2561
 75 +
 76 7
 77 *
78 XROM "DP"
 79 ASTO 02
 80 BEEP

 81*LBL 10
 82 STOPSW
 83 CLX
 84 SETSW

 85*LBL "S"
 86 CF 29
 87 FIX 0
 88 CLA
 89 CLX

 90*LBL 11
 91 XTOA
 92 ISG a
 93 -
 94 X<> [
 95 "DEC. "
 96 ARCL a
 97 "├?"
 98 AVIEW
 99 STO [
100 STOP
101 FS?C 22
102 GTO 11
103 CLA
104 AVIEW
105 STO [
106 DSE a

107*LBL 16
108 RCL 03
109 7
110 *
111 RCL a
112 /
113 STO 00
 114 OCT
115 GTO IND a

116*LBL 07
117 X<> [
118 X<> ]
119 STO a
120 GTO 12

121*LBL 04
122 FIX 1

123*LBL 05
124 SF 29

125*LBL 06
126*LBL 03
127*LBL 02
128*LBL 01
129 ASTO X
130 17
131 RCL a
132 /
133 INT
134 RCL b
135 ARCL Z
136 DSE Y
137 STO b
138 "├*"
139 FC? 29
140 "├**"
141 RCL a
142  E5
143 /

144*LBL 12
145 RCL 03
146 +
147 ABS
148 RCL 01
149 X<> c
150 RCL ]
151 GTO 09

152*LBL 00
 153  E
154 ST- L
 155 ARCL X
 156 LASTX
 157 R↑
 158 RCL ]

159*LBL 09
160 STO IND Z
 161 DSE Z
 162 GTO 09
 163 DSE a
 164 GTO 00

 165*LBL 13
 166 CLD
 167 X<>Y
 168 STO c
 169 CLST
 170 FC? 02
 171 FC? 01
 172 TONE 8
 173 FC? 01
 174 RTN

 175*LBL "T"
 176 ARCL 02
177 XROM "XE"
 178 SETSW
 179 X<>Y
 180 36 E5
 181 *
 182 RCL 00
 183 /
 184 FIX 9
 185 TONE 8
 186 END
LBL'3
LBL'2
LBL'1
LBL'IN
LBL'S
LBL'T
END          329 BYTES
```

```
GTO 01
```

Les caractères du message ont été stockés dans un groupe de registres de données, et le registre LAST X contient un index de ces registres. L'instruction RCL IND L place un caractère dans le registre X, puis XEQ IND X appelle un court sous-programme générant une suite de TONEs correspondant au caractère en X. Par exemple, si X contient la lettre C, la séquence suivante est exécutée :

```
LBL "C"
TONE 8
TONE P
TONE 8
TONE P
TONE 8
RTN
```

La simplicité de cette procédure est due à l'emploi de labels globaux synthétiques d'un seul caractère. Ils sont utilisés pour trois symboles de ponctuation et les lettres A à J. Les labels non synthétiques pour ces lettres sont locaux, et non globaux, et ne peuvent donc pas être adressés indirectement.

Néanmoins, la vitesse d'exécution est toujours critique avec cette approche. L'instruction XEQ IND X devant rechercher le label dans le CATalogue 1, elle met relativement longtemps à s'exécuter. En fait 16 millisecondes par label sont perdues à remonter le chaînage du CATalogue 1 à partir du .END. jusqu'à trouver le label cherché. Ceci demande un temps non négligeable pour les labels situés au sommet du CATalogue.

La principale trouvaille de ce programme de Morse est de remplacer l'instruction XEQ IND X par STO b pour sauter directement à la séquence voulue. Cela procure non seulement un gain de temps énorme, mais prouve également de façon éclatante que la programmation synthétique rend possible des choses hors de portée des moyens conventionnels, même très élaborés. Dans la pratique, des techniques synthétiques sont employées pour compiler les adresses des branchements indirects.

Il faut connaître quelques détails pour appliquer ce procédé. Premièrement, il faut déterminer l'adresse correcte du branchement. Ceci est réalisé ici en insérant une instruction RCL b avant chaque jeu de TONEs ; par exemple :

```
LBL "C"
RCL b
TONE 8 (STO b fera débuter l'exécution à cet endroit)
TONE P
TONE 8
TONE P
RTN
```

Ces séquences sont appelées avec le drapeau 26 baissé durant la procédure d'initialisation. Les résultats de RCL b sont incorporés dans des codes stockés dans une série de registres de données. On doit également faire attention d'inclure l'adresse de retour dans le code de telle façon que le RTN qui termine chaque séquence ramène l'exécution à l'instruction ISG L.

Pour gagner encore en vitesse, l'instruction GTO 01 est remplacée par un RTN. Une seconde adresse de retour est incorporée en plus de celle dont nous venons de parler pour faire ce travail. Cette seconde adresse est là pour transférer l'exécution directement à l'instruction RCL IND L, éliminant le besoin d'un LBL 01. De plus, RTN est 15% plus rapide qu'un GTO de trois octets compilé.

Le pointeur principal et deux pointeurs de retour utilisent six octets de chaque code pour STO b. L'octet de tête est choisi dans la première ligne de la table des codes pour éviter les problèmes de normalisation lors du rappel des codes stockés dans les registres. (Le fait que le premier octet appartienne à la première ligne de la table garantit que le code sera traité comme une donnée alphanumérique ordinaire). L'octet de gauche étant différent de zéro, il est préférable d'utiliser STOP plutôt que RTN pour arrêter l'exécution.

Dans le systême utilisé ici, les deux pointeurs de retour sont créés par des appels normaux de sous-programmes. Cette technique est plus simple que celle consistant à créer de toutes pièces les pointeurs car elle ne demande pas de calculer la position du programme en mémoire ou de fusionner une adresse de retour avec un pointeur programme. Le premier pointeur de retour est construit par XEQ IND T à la ligne 58, le second par XEQ 05 à la ligne 45. Ainsi, l'instruction RCL b précédant chaque jeu de TONEs fournit-elle le code complet, prêt à être stocké, les deux adresses de retour étant déjà enregistrées à ce moment-là.

Le résultat est un programme produisant du code Morse à 16 mots/minute, ce qui est une amélioration substantielle par rapport aux méthodes conventionnelles. De plus, la capacité du registre ALPHA est utilisée à son maximum, puisque **28** caractères peuvent être introduits en une fois pendant la phase de préparation. Cette capacité est due au fait que le calculateur reste en mode ALPHA lors des entrées de données (voir les informations sur le registre d'état P à la section 6A).

Voici le mode d'emploi du programme MC de Clifford Stern :

1) Exécutez SIZE pour que le nombre de registres soit au moins égal au nombre de caractères du message plus un.

2) XEQ "MC". Entrez un message par groupes de 28 caractères. Le signal sonore indiquant la fin habituelle du registre ALPHA vous signale ici que vous pouvez encore introduire 4 caractères. Pressez R/S pour faire exécuter chaque groupe. Si vous obtenez le message NONEXISTENT, augmentez le SIZE et recommencez.

3) Pressez R/S sans rien entrer pour transmettre le message. Pressez R/S ou XEQ 10 pour répéter ce message.

4) Pour obtenir une sortie moins rapide, insérez n'importe quelle instruction ne modifiant pas LAST X entre les lignes 45 et 46 puis faites de nouveau XEQ "MC". Cette modification augmente l'espacement des caractères.

Si vous avez un lecteur de codes-barres, utilisez les codes-barres donnés à l'appendice E pour charger le programme de Morse en mémoire. Si vous n'avez pas de lecteur, voici quelques conseils pour vous permettre de charger plus rapidement ce programme :

Les assignements suivants vous faciliteront l'entrée de MC à partir du listage ; 159, 120 (TONE P) ; 159, 8 (TONE 8) ; et 205, 0 (le label global contrepartie du Q-Loader). Ce dernier assignement fut découvert par Tom Cadwallader, et peut être employé pour créer les labels synthétiques désirés. Ainsi pour créer LBL "A", entrez XEQ "A" ou LBL "A". Ceci charge le caractère "A" dans le registre Q. Effacez cette instruction si vous étiez en mode PRGM lorsque vous l'avez tapée, et pressez la touche assignée en mode PRGM pour créer LBL "A". Cette méthode fut trouvée par Valentin Albillo, un autre pionnier de la programmation synthétique, et peut être employée pour créer des labels globaux de A à J.

Un procédé différent doit être utilisé pour créer des labels tels deux-points, point et virgule. Une méthode consiste à entrer le signe de ponctuation dans le registre ALPHA, ASTO X, et presser GTO IND X (tout cela en mode RUN). Cela charge le symbole de ponctuation dans le registre Q.

Après l'apparition de NONEXISTENT, passez en mode PRGM et pressez la touche assignée pour obtenir le label global correspondant.

Une alternative consiste à user du Byte Grabber pour synthétiser chacun de ces labels :

```
01 ENTER↑                    Données pour LB :
02 STO IND 66                192,
03 SIN                       0, (n'importe quelle valeur marche)
04 "Z:"                      242, 0, octet du caractère.
```

Presser le Byte Grabber à la ligne 01 retire l'octet STO et crée LBL ":". Il est essentiel de PACKer pour incorporer ces labels globaux dans le chaînage du CATalogue 1, quel que soit le moyen utilisé pour les créer.

Trois des chaînes de caractères de ce programme de code Morse apparaissent à l'impression sous une forme ambigüe. Ce sont :

Ligne	Hexa	Décimal
03	F4 2C 01 80 81	244 44 1 128 129
36	F2 7F 00	242 127 0
53	F2 7F 00	242 127 0

APPENDICE C : PROGRAMMATION SYNTHETIQUE, REFERENCES

Voici une liste de sources d'informations sur la programmation synthétique de la HP-41 :

En anglais :

1. **PPC Calculator Journal,** publié par PPC (Personal Programming Center), une association à but non lucratif californienne, dédiée à la programmation personnelle. Les numéros de juillet 1979 (Volume 6, Numéro 4) à aujourd'hui contiennent une foule d'informations sur la HP-41 en général, et la programmation synthétique en particulier. PPC CJ est toujours la source de programmes, techniques et découvertes synthétiques la plus à la pointe et à la portée de chacun.

Pour obtenir des informations sur l'adhésion à PPC et une liste de prix des anciens numéros de PPC CJ, envoyez une enveloppe timbrée directement à PPC USA, (courrier par avion) à l'adresse suivante :

PPC
POB 9599
Fountain Valley
California, 92728-9599
USA

Pour accélérer le traitement par PPC USA, marquez "New member info plus HP-41 back issues" dans l'angle inférieur gauche de votre enveloppe. Il n'est pas nécessaire d'inclure une lettre ; cela ne ferait que ralentir les choses.

2. **PPC Technical Notes,** publié par Melbourne, le chapitre australien de PPC. PPC TN est d'un format plus réduit que PPC CJ, mais est spécialisé dans la programmation synthétique et microcode (langage machine). Le prix de la souscription est actuellement de 150F par an pour la France et l'Europe. La diffusion pour l'Europe (en anglais) est assurée par les Editions du Cagire (77 rue du Cagire 31100 Toulouse, France).

3. **Datafile,** publié par le chapitre anglais de PPC. Cette publication est relativement récente, mais se consacre largement aux débutants. Pour plus de détails et un bulletin d'adhésion, envoyez une enveloppe timbrée à votre adresse à :

David M. Burch

```
01♦LBL "MC"
02 SF 26
  03 ",:"
04 X<> [
05 X<> d
06 RCL b
07 FC?C 26
08 GTO 01
09 CLA
10 ASTO Z
11 X<> [
12 SIGN
13 ASTO X
14 "≠"
15 ARCL X
16 ASTO b

17♦LBL 01
18 SF 26
19 "CHARACTERS?"
20 PROMPT
21 FC?C 23
22 GTO 06
23 VIEW Z
24 CF 26
25 CLX
26 ENTER↑
27 X<> ↑
28 X=Y?
29 GTO 02
30 SF 05
31 X<> ]
32 X<> \
33 X<> [
34 X<>Y

35♦LBL 02
36 "├♦"
37 X<> ↑
38 X=0?
39 GTO 02
40 STO ↑
41 RDN
42 0
43 FC?C 29
44 GTO 05
45 XEQ 05
46 RCL IND L
47 STO b

48♦LBL 03
49 STO IND L
50 RDN

51♦LBL 04
52 .
53 "├♦"

54♦LBL 05
55 X<> ↑
56 RDN
57 SF 25
58 XEQ IND T
59 ISG L
60 RTN
61 FS?C 25
62 GTO 03
63 FS? 26
64 GTO 07
65 DSE L
66 FC?C 05
67 GTO 01
68 STO ]
69 GTO 04

70♦LBL 06
71 LASTX
72  E3
73 +
74 LASTX
75 /
76 STO 00
77 SIGN
78 R↑
79 STO d
80 RCL 01
81 STO b

82♦LBL 07
83 RCL 00
84 SIGN
85 STOP

86♦LBL 10
87 RCL 01
88 STO b

89♦LBL ":"
90 RCL b
91 TONE 8
92 TONE 8
93 TONE 8
94 TONE ↑
95 TONE ↑
96 TONE ↑
97 RTN

98♦LBL "-"
99 RCL b
100 TONE 8
101 TONE ↑
102 TONE ↑
103 TONE ↑
104 TONE 8
105 RTN

106♦LBL "/"
107 RCL b
108 TONE 8
109 TONE ↑
110 TONE ↑
111 TONE 8
112 TONE ↑
113 RTN

114♦LBL "?"
115 RCL b
116 TONE ↑
117 TONE ↑
118 TONE 8
119 TONE 8
120 TONE ↑
121 TONE ↑
122 RTN
123♦LBL "."
124 RCL b
125 TONE ↑
126 TONE 8
127 TONE ↑
128 TONE 8
129 TONE ↑
130 TONE 8
131 RTN

132♦LBL ","
133 RCL b
134 TONE 8
135 TONE 8
136 TONE ↑
137 TONE ↑
138 TONE 8
139 TONE 8
140 RTN

141♦LBL "0"
142 RCL b
143 TONE 8
144 TONE 8
145 TONE 8
146 TONE 8
147 TONE 8
148 RTN

149♦LBL "9"
150 RCL b
151 TONE 8
152 TONE 8
153 TONE 8
154 TONE 8
155 TONE ↑
156 RTN

157♦LBL "8"
158 RCL b
159 TONE 8
160 TONE 8
161 TONE 8
162 TONE ↑
163 TONE ↑
164 RTN

165♦LBL "7"
166 RCL b
167 TONE 8
168 TONE 8
169 TONE ↑
170 TONE ↑
171 TONE ↑
172 RTN

173♦LBL "6"
174 RCL b
175 TONE 8
176 TONE ↑
177 TONE ↑
178 TONE ↑
179 TONE ↑
180 RTN

181♦LBL "5"
182 RCL b
183 TONE ↑
184 TONE ↑
185 TONE ↑
186 TONE ↑
187 TONE ↑
188 RTN

189♦LBL "4"
190 RCL b
191 TONE ↑
192 TONE ↑
193 TONE ↑
194 TONE ↑
195 TONE 8
196 RTN

197♦LBL "3"
198 RCL b
199 TONE ↑
200 TONE ↑
201 TONE ↑
202 TONE 8
203 TONE 8
```

```
204 RTN

205♦LBL "2"
206 RCL b
207 TONE †
208 TONE †
209 TONE 8
210 TONE 8
211 TONE 8
212 RTN

213♦LBL "1"
214 RCL b
215 TONE †
216 TONE 8
217 TONE 8
218 TONE 8
219 TONE 8
220 RTN

221♦LBL "Z"
222 RCL b
223 TONE 8
224 TONE 8
225 TONE †
226 TONE †
227 RTN

228♦LBL "Q"
229 RCL b
230 TONE 8
231 TONE 8
232 TONE †
233 TONE 8
234 RTN

235♦LBL "J"
236 RCL b
237 TONE †
238 TONE 8
239 TONE 8
240 TONE 8
241 RTN

242♦LBL "X"
243 RCL b
244 TONE 8
245 TONE †
246 TONE †
247 TONE 8
248 RTN

249♦LBL "K"
250 RCL b
251 TONE 8
252 TONE †
253 TONE 8
254 RTN

255♦LBL "V"
256 RCL b
257 TONE †
258 TONE †
259 TONE †
260 TONE 8
261 RTN

262♦LBL "B"
263 RCL b
264 TONE 8
265 TONE †
266 TONE †
267 TONE †
268 RTN

269♦LBL "G"
270 RCL b
271 TONE 8
272 TONE 8
273 TONE †
274 RTN

275♦LBL "W"
276 RCL b
277 TONE †
278 TONE 8
279 TONE 8
280 RTN

281♦LBL "Y"
282 RCL b
283 TONE 8
284 TONE †
285 TONE 8
286 TONE 8
287 RTN

288♦LBL "P"
289 RCL b
290 TONE †
291 TONE 8
292 TONE 8
293 TONE †
294 RTN

295♦LBL " "
296 RCL b
297 FC? 26
298 RTN
299 LASTX
300 LN
301 RTN

302♦LBL "M"
303 RCL b
304 TONE 8
305 TONE 8
306 RTN

307♦LBL "U"
308 RCL b
309 TONE †
310 TONE †
311 TONE 8
312 RTN

313♦LBL "F"
314 RCL b
315 TONE †
316 TONE †
317 TONE 8
318 TONE †
319 RTN

320♦LBL "C"
321 RCL b
322 TONE 8
323 TONE †
324 TONE 8
325 TONE †
326 RTN

327♦LBL "L"
328 RCL b
329 TONE †
330 TONE 8
331 TONE †
332 TONE †
333 RTN

334♦LBL "D"
335 RCL b
336 TONE 8
337 TONE †
338 TONE †
339 RTN

340♦LBL "H"
341 RCL b
342 TONE †
343 TONE †
344 TONE †
345 TONE †
346 RTN

347♦LBL "S"
348 RCL b
349 TONE †
350 TONE †
351 TONE †
352 RTN

353♦LBL "I"
354 RCL b
355 TONE †
356 TONE †
357 RTN

358♦LBL "R"
359 RCL b
360 TONE †
361 TONE 8
362 TONE †
363 RTN

364♦LBL "N"
365 RCL b
366 TONE 8
367 TONE †
368 RTN

369♦LBL "O"
370 RCL b
371 TONE 8
372 TONE 8
373 TONE 8
374 RTN

375♦LBL "A"
376 RCL b
377 TONE †
378 TONE 8
379 RTN

380♦LBL "T"
381 RCL b
382 TONE 8
383 RTN

384♦LBL "E"
385 RCL b
386 TONE †
387 END

LBL'MC
LBL':
LBL'-
LBL'/
LBL'?
LBL'.
LBL',
LBL'0
LBL'9
LBL'8
LBL'7
LBL'6
LBL'5
LBL'4
LBL'3
LBL'2
LBL'1
LBL'Z
LBL'Q
LBL'J
LBL'X
LBL'K
LBL'V
LBL'B
LBL'G
LBL'W
LBL'Y
LBL'P
LBL'
LBL'M
LBL'U
LBL'F
LBL'C
LBL'L
LBL'D
LBL'H
LBL'S
LBL'I
LBL'R
LBL'N
LBL'O
LBL'A
LBL'T
LBL'E
END
845 BYTES
```

Astage
Rectory Lane
Windlesham, Surrey
GU20 6BW
ANGLETERRE

Les demandes venant de l'étranger doivent inclure une enveloppe et une coupon postal international ou deux cartes magnétiques en lieu et place des timbres.

4. **Le PPC ROM User's Manual** (Manuel d'utilisation du PPC ROM), vendu avec le ROM. Ce ROM est un module enfichable conçu par les membres de PPC et fabriqué par Hewlett-Packard. Le PPC ROM contient plus de 60 programmes synthétiques (sur un total d'environ 150 programmes), tous commentés ligne à ligne dans le manuel.

Vous pouvez commander le PPC ROM aux Etats-Unis à PPC (voir adresse plus haut). Il est également disponible en France auprès de PPC-Toulouse. Dans les deux cas, les délais de livraison peuvent varier de huit jours à trois mois, du fait des problèmes de transfert de fonds et de délai de réponse du club américain. Prix PPC-T 1984, 1050F

5. **Le ZENROM,** module enfichable conçu par Zengrange Limited, une société anglaise. Ce module a été conçu spécifiquement pour faire de la programmation synthétique. Il rend toutes les instructions synthétiques aussi faciles à utiliser que les instructions normales. Il permet également d'observer et de modifier tous les coins de la mémoire. Il comporte de plus un assembleur désassembleur pour le langage machine. Prix et disponibilité à demander aux Editions du Cagire.

En français :

6. **MICRO-REVUE** est la revue du principal chapitre français de PPC, et ses 128 pages paraissant tous les deux mois traitent largement de la programmation synthétique. Bien que basé à Toulouse, ce club possède de nombreux adhérents dans toute la France. Bulletin d'adhésion et renseignements à PPC-T 77 rue du Cagire 31100 Toulouse.

7. **Le catalogue des Editions du Cagire** comporte de nombreux titres en français et en anglais consacrés à la HP-41, et en particulier à la programmation synthétique, ainsi que du matériel et des programmes. Demandez ce catalogue gratuit aux Editions du Cagire, 77 rue du Cagire, 31100 TOULOUSE FRANCE.

APPENDICE D : LA CARTE DES CODES PLASTIFIEE

La carte des codes est une carte en plastique qui contient un grand nombre d'informations essentielles à la pratique de la programmation synthétique. Chaque exemplaire de **La programmation synthétique, c'est facile !** est livré avec une carte des codes.

Les deux-tiers gauche de la carte sont occupés par la table des codes. Chaque case de la table illustre les diverses interprétations d'un octet. Reportez vous à la légende de la table des codes ci-après. Ces équivalences sont introduites et expliquées aux premier et second chapitres.

Les caractères apparaissant à l'affichage ne sont pas donnés pour la seconde moitié de la table (lignes 8 à F) car ce sont tous des starbursts (les 14 segments sont allumés). Cela a permis de placer l'intitulé complet des équivalents en tant que suffixe à la seconde ligne de chaque case. Les caractères d'impression donnés sont ceux obtenus par PRA lorsque l'octet en question réside dans le registre ALPHA. Au bas de chaque moitié de la table

se trouvent les équivalents binaires des chiffres hexadécimaux 0 à F.

A la droite de la première moitié de la table se trouve un résumé des utilisations des 56 drapeaux de la HP-41. A la droite de la seconde partie de la table se trouve un bref rappel des données à fournir à LB pour chaque type d'instruction. Le troisième chapitre couvre ce sujet.

Aspects étranges de la table des codes : Les caractères des lignes 8 à F disparaissent lors des **listages** de programmes (pas lors de l'exécution de PRA), excepté ceux qui sont ombrés (grisés) qui donnent à l'imprimante un comportement curieux (voir section 2E). La ligne 0 fournit les données pour MK, de 0 à 15, pour obtenir les fonctions non programmables écrites au-dessus en petites lettres. Voir la section 4A pour plus de détails. La ligne 1 comprend la fonction W^{τ} qui a peu d'effets, excepté enlever le contrôle au clavier jusqu'à ce que les piles soient retirées (pour les détails sur W^{τ}, se reporter à MICRO-REVUE, le journal de PPC-Toulouse). Les octets marqués SPARE forment des instructions NOP (no operation : pas d'opération) de deux octets.

Si cet aperçu rapide vous paraît confus, vous n'avez certainement pas lu les deux premiers chapitres, alors retournez-y et lisez les |

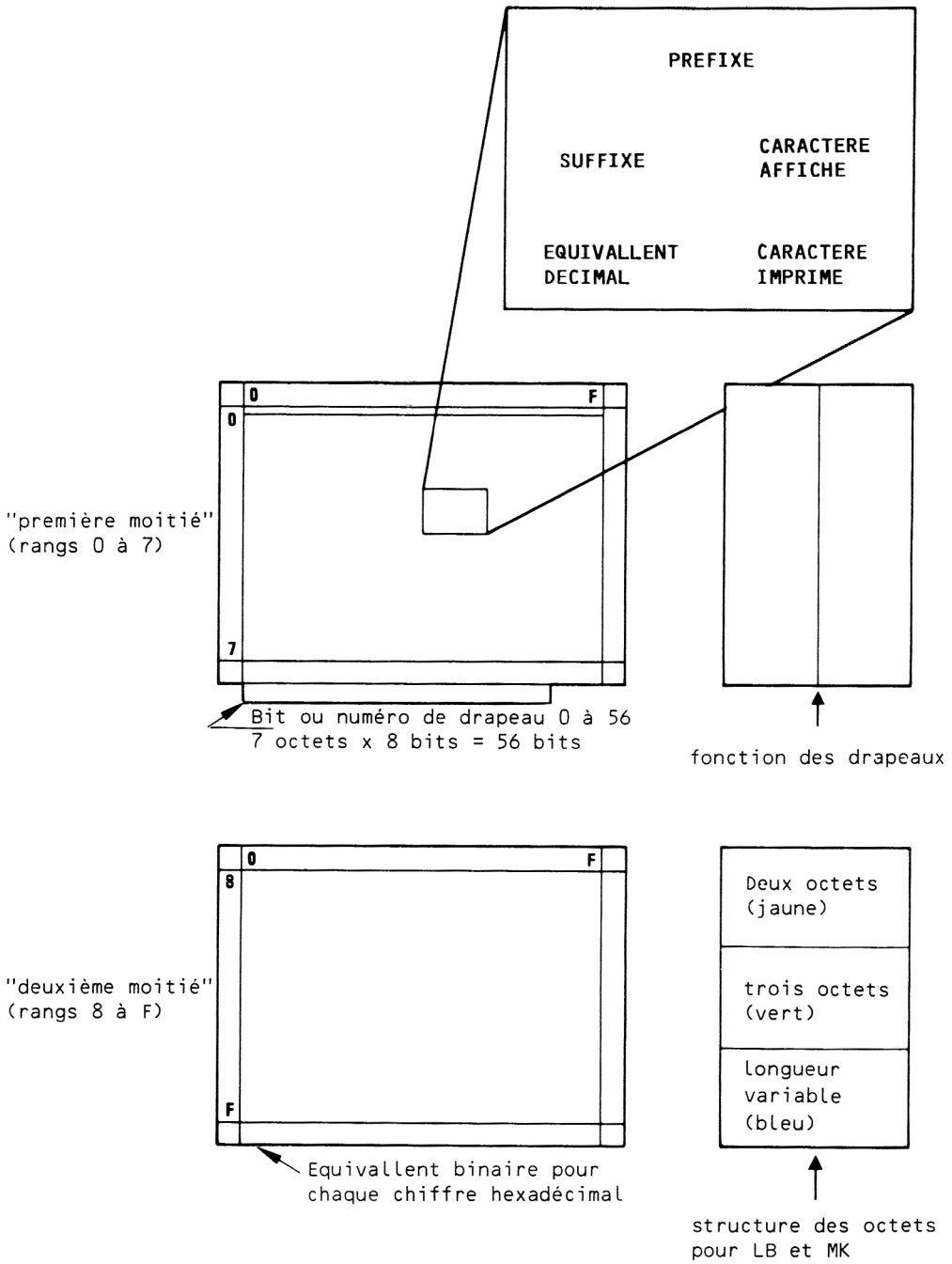
APPENDICE E : CODES-BARRES DES PROGRAMMES

Des codes-barres sont fournis ici pour tous les programmes utilitaires décrits dans ce livre, afin que vous puissiez entrer ces programmes via le lecteur optique Wand HP-82153A. Si vous avez un Wand ou pouvez vous en faire prêter un, ceci vous économisera pas mal de temps. Si vous avez un lecteur de cassettes, vous pouvez également commander la cassette de ces programmes à l'éditeur.

Protégez toujours la surface des codes-barres avec une feuille de plastique transparent, si vous comptez les utiliser souvent. Il peut parfois être utile de placer une feuille de papier blanc (parfois le noir convient mieux) derrière les codes-barres pour augmenter le contraste.

Ces codes-barres ont été essayés et lus correctement. Si vous ne pouvez pas les lire, essayez d'encrer les barres paraissant incomplètes et de lire plus vite en vous aidant d'une règle, ou d'orienter différemment le lecteur. Si tout ceci échoue, essayez un autre lecteur.

Si vous avez un lecteur de cartes, vous devez enregistrer ces programmes au cas où votre chien ferait joujou avec ce livre. D'autres méthodes de stockage sont fournies par l'unité de cassettes HP-IL et la mémoire étendue. Néanmoins, cette dernière ne doit pas être considérée comme un stockage sûr, car elle est soumise au risque de MEMORY LOST.



LEGENDE DE LA TABLE DES CODES PLASTIFIEE

FLAGS (Register d)

00-10 general purpose
11 auto execute
12 doublewide
13 lower case
14 overwrite
15-16 IL printer
 0 0 MAN
 0 1 NORM
 1 0 TRACE
 1 1 TR/STACK
17 record incomplete
18 ┐general use
19 │cleared at
20 ┘turn-on
21 prtr enable
22 num. entry
23 alpha entry
24 range ignore
25 error ignore
26 audio enable
27 USER mode
28 dec./comma
29 digit grouping
30 CAT
31 timer DMY/MDY
32 manual IL I/O
33 IL absolute manual
34 not used
35 not used
36-39 number of digits
40-41 display
 0 0 SCI
 0 1 ENG
 1 0 FIX
 1 1 FIX/ENG
42-43 trig mode
 0 0 DEG
 0 1 RAD
 1 0 GRAD
 1 1 RAD
44 cont. ON
45 system data entry
46 partial key sequence
47 SHIFT
48 ALPHA
49 low BAT
50 message
51 SST
52 PGRM
53 I/O
54 PSE
55 printer existence

Structure of multi-byte instructions

Two-byte instructions
STO 16=145,16 DSE IND 55 =151,183
LBL e =207,127 FS?C IND Y =170,242
RCL b =144,124 TONE 89 =159,89
X<>M=206,117 ST+ IND N =146,246
LBL Q =207,121 VIEW H(109)=152,109

Two-byte special cases
GTO IND=174,reg. XEQ IND=174,128+r
GTO IND 09=174,9 XEQ IND X=174,243
XROM i,j =160+i/4,64(i mod 4)+j
WSTS =XROM 30,10 =167,138
short form GTO =177+label,0
GTO 12 =189,0

Three-byte instructions
long form GTO =208,0,label
GTO 32 =208,0,32
XEQ =224,0,label
XEQ D =224,0,105
END =192,0,9+ sum of status indicators
32(.END.), 4(rePACK), 2(decompile)

Variable length instructions
TEXT =240+n, n character bytes
Append symbol counts as first char.
ᵀ& =241,38 ᵀ⊢)? =243,127,41,63
GTO ᵀ =29,240+n, n character bytes
GTO ᵀXYZ =29,243,88,89,90
XEQ ᵀ =30,240+n, n character bytes
XEQ ᵀA =30,241,65 (synthetic)
LBL ᵀ =192,0,241+n, (key), n chars.
LBL ᵀ: =192,0,242,0,58 (synthetic)

HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	CAT	@c (GTO..)	DEL	COPY	CLP	R/S	SIZE	BST	SST	ON	PACK	←(PRGM)	USR/P/A	2 — —	SHIFT	ASN	
0	NULL 00 ⁻ 0 ◆	LBL 00 01 ⁄ 1 ×	LBL 01 02 ▩ 2 ⁄	LBL 02 03 ▩ 3 ←	LBL 03 04 ⁄ 4 α	LBL 04 05 ⁄ 5 β	LBL 05 06 ⁄ 6 Γ	LBL 06 07 ▩ 7 ↓	LBL 07 08 ▩ 8 Δ	LBL 08 09 ▩ 9 σ	LBL 09 10 ▩ 10 ◆	LBL 10 11 ▩ 11 ⁄	LBL 11 12 ⁄ 12 ρ	LBL 12 13 ∠ 13 ∡	LBL 13 14 ▩ 14 ⁻	LBL 14 15 ▩ 15 ⁄	0
1	0 16 ▩ 16 θ	1 17 ▩ 17 Ω	2 18 ▩ 18 δ	3 19 ▩ 19 ä	4 20 ▩ 20 ä	5 21 ▩ 21 ä	6 22 ▩ 22 ä	7 23 ▩ 23 ö	8 24 ▩ 24 ö	9 25 ▩ 25 Ö	. 26 ▩ 26 Ü	EEX 27 ▩ 27 Æ	NEG 28 ▩ 28 æ	GTO ᵀ 29 ∠ 29 ≠	XEQ ᵀ 30 ▩ 30 £	W ᵀ 31 ▩ 31 ▩	1
2	RCL 00 32 32	RCL 01 33 ; 33 !	RCL 02 34 " 34 "	RCL 03 35 ⁄ 35 #	RCL 04 36 % 36 $	RCL 05 37 % 37 %	RCL 06 38 & 38 &	RCL 07 39 ' 39 '	RCL 08 40 < 40 (	RCL 09 41 > 41)	RCL 10 42 ⁄ 42 *	RCL 11 43 ÷ 43 +	RCL 12 44 , ⁄ 44 ,	RCL 13 45 - 45 —	RCL 14 46 ⊦ 46 -	RCL 15 47 ⁄ 47 ⁄	2
3	STO 00 48 0 48 0	STO 01 49 ⁄ 49 1	STO 02 50 2 50 2	STO 03 51 3 51 3	STO 04 52 4 52 4	STO 05 53 5 53 5	STO 06 54 6 54 6	STO 07 55 7 55 7	STO 08 56 8 56 8	STO 09 57 9 57 9	STO 10 58 :▩ 58 :	STO 11 59 ; 59 ;	STO 12 60 ∠ 60 <	STO 13 61 ≐ 61 =	STO 14 62 ≥ 62 >	STO 15 63 ? 63 ?	3
4	+ 64 P 64 @	— 65 A 65 A	✳ 66 B 66 B	/ 67 C 67 C	X<Y? 68 D 68 D	X>Y? 69 E 69 E	X≤Y? 70 F 70 F	Σ+ 71 G 71 G	Σ− 72 H 72 H	HMS+ 73 I 73 I	HMS− 74 J 74 J	MOD 75 K 75 K	% 76 L 76 L	%CH 77 M 77 M	P→R 78 N 78 N	R→P 79 O 79 O	4
5	LN 80 P 80 P	X↑2 81 Q 81 Q	SQRT 82 R 82 R	Y↑X 83 S 83 S	CHS 84 T 84 T	E↑X 85 U 85 U	LOG 86 V 86 V	10↑X 87 W 87 W	E↑X-1 88 X 88 X	SIN 89 Y 89 Y	COS 90 Z 90 Z	TAN 91 [91 [	ASIN 92 \ 92 \	ACOS 93] 93]	ATAN 94 ↗ 94 ↑	→DEC 95 _ 95 _	5
6	1/X 96 ᵀ 96 ᵀ	ABS 97 a 97 a	FACT 98 b 98 b	X≠0? 99 c 99 c	X>0? 100 d 100 d	LN1+X 101 e 101 e	X<0? A ▩ 102 f	X=0? B ▩ 103 g	INT C ▩ 104 h	FRC D ▩ 105 i	D→R E ▩ 106 j	R→D F ▩ 107 k	→HMS G ▩ 108 l	→HR H ▩ 109 m	RND I ▩ 110 n	→OCT J ▩ 111 o	6
7	CLΣ T ▩ 112 p	X<>Y Z ▩ 113 q	PI Y ▩ 114 r	CLST X ▩ 115 s	R↑ L ▩ 116 t	RDN M [▩ 117 u	LASTX N \ ▩ 118 v	CLX O] ▩ 119 w	X=Y? P↑ ▩ 120 x	X≠Y? Q_ ▩ 121 y	SIGN ⊢ ᵀ ▩ 122 z	X≤0? a ▩ 123 π	MEAN b ▩ 124 !	SDEV c ▩ 125 →	AVIEW d Σ 126 Σ	CLD e ⊦ 127 ⊢	7
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	
	00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31	32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	48 49 50 51	52 53 54 55	← bit numbers in a 7-byte register		

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	DEG IND 00 128 ◆	RAD IND 01 129 ×	GRAD IND 02 130 ÷	ENTER↑ IND 03 131 ←	STOP IND 04 132 α	RTN IND 05 133 β	BEEP IND 06 134 Γ	CLA IND 07 135 ↓	ASHF IND 08 136 Δ	PSE IND 09 137 σ	CLRG IND 10 138 ◆	AOFF IND 11 139 λ	AON IND 12 140 μ	OFF IND 13 141 ∠	PROMPT IND 14 142 ⌐	ADV IND 15 143 ⏎	8
9	RCL IND 16 144 θ	STO IND 17 145 Ω	ST+ IND 18 146 δ	ST− IND 19 147 ä	ST × IND 20 148 ά	ST/ IND 21 149 ä	ISG IND 22 150 ä	DSE IND 23 151 ö	VIEW IND 24 152 ö	Σ REG IND 25 153 ü	ASTO IND 26 154 ü	ARCL IND 27 155 Æ	FIX IND 28 156 æ	SCI IND 29 157 ≠	ENG IND 30 158 £	TONE IND 31 159 ▒	9
A	XR 0-3 IND 32 160	XR 4-7 IND 33 161 !	XR8-11 IND 34 162 "	X12-15 IND 35 163 #	X16-19 IND 36 164 $	X20-23 IND 37 165 %	X24-27 IND 38 166 &	X28-31 IND 39 167 '	SF IND 40 168 (	CF IND 41 169)	FS?C IND 42 170 *	FC?C IND 43 171 +	FS? IND 44 172 ,	FC? IND 45 173 −	GTO XEQ IND IND 46 174 .	SPARE IND 47 175 /	A
B	SPARE IND 48 176 0	GTO 00 IND 49 177 1	GTO 01 IND 50 178 2	GTO 02 IND 51 179 3	GTO 03 IND 52 180 4	GTO 04 IND 53 181 5	GTO 05 IND 54 182 6	GTO 06 IND 55 183 7	GTO 07 IND 56 184 8	GTO 08 IND 57 185 9	GTO 09 IND 58 186 :	GTO 10 IND 59 187 ;	GTO 11 IND 60 188 <	GTO 12 IND 61 189 =	GTO 13 IND 62 190 >	GTO 14 IND 63 191 ?	B
C	GLOBAL IND 64 192 @	GLOBAL IND 65 193 A	GLOBAL IND 66 194 B	GLOBAL IND 67 195 C	GLOBAL IND 68 196 D	GLOBAL IND 69 197 E	GLOBAL IND 70 198 F	GLOBAL IND 71 199 G	GLOBAL IND 72 200 H	GLOBAL IND 73 201 I	GLOBAL IND 74 202 J	GLOBAL IND 75 203 K	GLOBAL IND 76 204 L	GLOBAL IND 77 205 M	X<>-- IND 78 206 N	LBL -- IND 79 207 O	C
D	GTO -- IND 80 208 P	GTO -- IND 81 209 Q	GTO -- IND 82 210 R	GTO -- IND 83 211 S	GTO -- IND 84 212 T	GTO -- IND 85 213 U	GTO -- IND 86 214 V	GTO -- IND 87 215 W	GTO -- IND 88 216 X	GTO -- IND 89 217 Y	GTO -- IND 90 218 Z	GTO -- IND 91 219 [	GTO -- IND 92 220 \	GTO -- IND 93 221]	GTO -- IND 94 222 ↑	GTO -- IND 95 223 _	D
E	XEQ -- IND 96 224 `	XEQ -- IND 97 225 a	XEQ -- IND 98 226 b	XEQ -- IND 99 227 c	XEQ -- IND100 228 d	XEQ -- IND101 229 e	XEQ -- IND102 230 f	XEQ -- IND103 231 g	XEQ -- IND104 232 h	XEQ -- IND105 233 i	XEQ -- IND106 234 j	XEQ -- IND107 235 k	XEQ -- IND108 236 l	XEQ -- IND109 237 m	XEQ -- IND110 238 n	XEQ -- IND111 239 o	E
F	TEXT 0 IND T 240 p	TEXT 1 IND Z 241 q	TEXT 2 IND Y 242 r	TEXT 3 IND X 243 s	TEXT 4 IND L 244 t	TEXT 5 IND M [245 u	TEXT 6 IND N \ 246 v	TEXT 7 IND O] 247 w	TEXT 8 IND P ↑ 248 x	TEXT 9 IND Q _ 249 y	TEXT10 IND ⊢ 250 z	TEXT11 IND a 251 ⌐	TEXT12 IND b 252 |	TEXT13 IND c 253 →	TEXT14 IND d 254 Σ	TEXT15 IND e 255 ⊢	F
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	

For price information and a list of dealers in your area, send a self-addressed stamped envelope to: SYNTHETIX, 1540 Mathews Ave., Manhattan Beach, CA 90266, USA

DECIMAL TO CHARACTER PROGRAM REGISTERS NEEDED: 8

ROW 1 (1 : 6)

ROW 2 (6 : 12)

ROW 3 (13 : 18)

ROW 4 (18 : 25)

ROW 5 (25 : 25)

PROGRAM REGISTERS NEEDED: 31

ROW 1 (1 : 6)

ROW 2 (6 : 15)

ROW 3 (15 : 22)

ROW 4 (22 : 25)

ROW 5 (26 : 31)

ROW 6 (32 : 38)

ROW 7 (38 : 44)

ROW 8 (45 : 53)

ROW 9 (53 : 61)

ROW 10 (61 : 68)

ROW 11 (69 : 76)

ROW 12 (76 : 82)

ROW 13 (83 : 88)

ROW 14 (88 : 95)

ROW 15 (96 : 103)

ROW 16 (103 : 110)

ROW 17 (110 : 112)

ROW 1 (1 : 6)

ROW 2 (6 : 14)

ROW 3 (15 : 21)

ROW 4 (22 : 25)

ROW 5 (25 : 32)

ROW 6 (33 : 42)

ROW 7 (42 : 51)

ROW 8 (52 : 57)

ROW 9 (57 : 65)

ROW 10 (65 : 73)

ROW 11 (73 : 80)

ROW 12 (80 : 87)

ROW 13 (87 : 88)

MAKE KEY ASSIGNMENTS

PROGRAM REGISTERS NEEDED: 45

ROW 1 (1 : 5)

ROW 2 (6 : 12)

ROW 3 (12 : 19)

ROW 4 (19 : 22)

ROW 5 (23 : 28)

ROW 6 (29 : 31)

ROW 7 (31 : 35)

ROW 8 (35 : 40)

ROW 9 (40 : 48)

ROW 10 (48 : 57)

ROW 11 (58 : 67)

ROW 12 (68 : 77)

ROW 13 (78 : 85)

ROW 14 (85 : 93)

ROW 15 (93 : 99)

ROW 16 (99 : 103)

ROW 17 (103 : 108)

ROW 18 (109 : 116)

ROW 19 (116 : 121)

ROW 20 (121 : 128)

ROW 21 (129 : 136)

ROW 22 (137 : 143)

ROW 23 (143 : 149)

ROW 24 (149 : 154)

ROW 25 (154 : 154)

ROW 1 (1 : 3)

ROW 2 (3 : 8)

ROW 3 (8 : 14)

ROW 4 (14 : 18)

ROW 5 (18 : 26)

ROW 6 (27 : 31)

ROW 7 (31 : 37)

ROW 8 (38 : 41)

ROW 9 (41 : 48)

ROW 10 (49 : 51)

PROGRAM REGISTERS NEEDED: 23

ROW 1 (1 : 3)

ROW 2 (4 : 8)

ROW 3 (8 : 14)

ROW 4 (15 : 23)

ROW 5 (23 : 27)

ROW 6 (28 : 31)

ROW 7 (31 : 38)

ROW 8 (39 : 43)

ROW 9 (44 : 50)

ROW 10 (50 : 56)

ROW 11 (57 : 63)

ROW 12 (63 : 69)

ROW 13 (70 : 70)

SAVE ALARMS/ RECALL ALARMS

PROGRAM REGISTERS NEEDED: 25

ROW 1 (1 : 6)

ROW 2 (7 : 14)

ROW 3 (15 : 23)

ROW 4 (23 : 30)

ROW 5 (30 : 36)

ROW 6 (36 : 41)

ROW 7 (41 : 49)

ROW 8 (50 : 57)

ROW 9 (57 : 63)

ROW 10 (64 : 72)

ROW 11 (73 : 81)

ROW 12 (82 : 87)

ROW 13 (88 : 96)

ROW 14 (97 : 99)

EXTENDED FUNCTIONS / TIME
MODULE
PROGRAM REGISTERS NEEDED: 9

ROW 1 (1 : 5)

ROW 2 (6 : 12)

ROW 3 (12 : 16)

ROW 4 (16 : 23)

ROW 5 (24 : 26)

SUSPEND KEY ASSIGNMENTS/
REACTIVATE KEY ASSIGNMENTS
PROGRAM REGISTERS NEEDED: 10

ROW 1 (1 : 5)

ROW 2 (6 : 13)

ROW 3 (14 : 19)

ROW 4 (20 : 25)

ROW 5 (25 : 31)

SOLVE f(x) = 0 for x

PROGRAM REGISTERS NEEDED: 14

ROW 1 (1 : 2)

ROW 2 (2 : 7)

ROW 3 (7 : 10)

ROW 4 (10 : 19)

ROW 5 (20 : 29)

ROW 6 (30 : 39)

ROW 7 (40 : 48)

ROW 8 (48 : 49)

CURTAIN UP

PROGRAM REGISTERS NEEDED: 10

ROW 1 (1 : 5)

ROW 2 (5 : 10)

ROW 3 (11 : 21)

ROW 4 (21 : 29)

ROW 5 (29 : 35)

ROW 6 (35 : 35)

INSTRUCTION TIMER

PROGRAM REGISTERS NEEDED: 47

ROW 1 (1 : 8)

ROW 2 (8 : 13)

ROW 3 (13 : 17)

ROW 4 (17 : 26)

ROW 5 (26 : 31)

ROW 6 (31 : 40)

ROW 7 (40 : 52)

ROW 8 (53 : 61)

ROW 9 (62 : 66)

ROW 10 (66 : 71)

ROW 11 (72 : 79)

ROW 12 (79 : 85)

ROW 13 (86 : 94)

ROW 14 (94 : 98)

ROW 15 (99 : 106)

ROW 16 (107 : 116)

ROW 17 (117 : 124)

ROW 18 (124 : 133)

ROW 19 (134 : 139)

ROW 20 (140 : 147)

ROW 21 (148 : 155)

ROW 22 (156 : 163)

ROW 23 (164 : 172)

ROW 24 (172 : 177)

ROW 25 (178 : 185)

ROW 26 (185 : 186)

PROGRAM REGISTERS NEEDED: 121

ROW 1 (1 : 3)

ROW 2 (4 : 10)

ROW 3 (11 : 18)

ROW 4 (18 : 19)

ROW 5 (20 : 27)

ROW 6 (28 : 35)

ROW 7 (36 : 43)

ROW 8 (43 : 49)

ROW 9 (50 : 58)

ROW 10 (58 : 65)

ROW 11 (65 : 72)

ROW 12 (73 : 83)

ROW 13 (84 : 90)

ROW 14 (91 : 97)

ROW 15 (98 : 102)

ROW 16 (103 : 108)

ROW 17 (108 : 114)

ROW 18 (114 : 120)

ROW 19 (120 : 125)

ROW 20 (126 : 132)

ROW 21 (132 : 137)

ROW 22 (138 : 143)

ROW 23 (143 : 149)

ROW 24 (149 : 155)

ROW 25 (155 : 160)

ROW 26 (161 : 166)

ROW 27 (166 : 173)

ROW 28 (173 : 178)

ROW 29 (178 : 183)

ROW 30 (184 : 189)

ROW 31 (189 : 195)

ROW 32 (196 : 201)

ROW 33 (201 : 206)

ROW 34 (207 : 213)

ROW 35 (213 : 218)

ROW 36 (219 : 224)

ROW 37 (224 : 229)

ROW 38 (230 : 235)

ROW 39 (235 : 242)

ROW 40 (242 : 247)

ROW 41 (247 : 252)

ROW 42 (253 : 258)

ROW 43 (258 : 263)

ROW 44 (264 : 269)

ROW 45 (269 : 275)

ROW 46 (275 : 281)

ROW 47 (281 : 286)

ROW 48 (287 : 292)

ROW 49 (292 : 297)

ROW 50 (298 : 304)

ROW 51 (305 : 310)

ROW 52 (310 : 315)

ROW 53 (316 : 321)

ROW 54 (321 : 327)

ROW 55 (327 : 333)

ROW 56 (334 : 338)

ROW 57 (339 : 344)

ROW 58 (344 : 349)

ROW 59 (350 : 355)

ROW 60 (355 : 360)

ROW 61 (361 : 366)

ROW 62 (366 : 371)

ROW 63 (372 : 377)

ROW 64 (377 : 382)

ROW 65 (383 : 387)

ADDENDUM

Petits détails utiles

L'imprimante ralentit l'exécution

Le fait que l'imprimante soit connectée à la HP-41 ralentit l'exécution des programmes, que le drapeau 21 soit ou non levé et que l'imprimante soit allumée ne change rien à l'affaire. Même les fonctions n'utilisant pas l'imprimante sont plus lentes.

Ce fait peut être atténué en baissant synthétiquement le drapeau 55, celui signalant la présence d'une imprimante. Les séquences suivantes font toutes l'affaire :

Avec une 41 seule

```
SF 07**
RCL d
CLA
STO M
ASTO M
-
X<> M
STO d
RDN
```

Avec un module X-Fonctions*

```
RCLFLAG
SIGN
STO d
X<> L
STOFLAG
RDN
```

*(Programme écrit par Steve Wandzura)

**tout drapeau de 00 à 07 convient

Avec un PPC ROM

```
55
FS? 55
RDN
FC? 55
XROM IF
```

Tant que votre programme ne rencontre pas de fonction de l'imprimante, le drapeau 55 reste baissé et l'exécution est accélérée. Si le drapeau 21 est baissé, la rencontre d'une instruction de l'imprimante ne lèvera pas non plus le drapeau 55. La fonction sera ignorée comme normalement.

Si le drapeau 21 est levé, le comportement dépend du type de l'imprimante connectée. Avec une imprimante HP 82143A, toutes les fonctions de l'imprimante sont inhibées jusqu'à ce que le programme s'arrête, auquel cas les deux drapeaux sont immédiatement levés, même si le drapeau 21 était baissé. Avec une imprimante HP-IL, l'état du drapeau 21 conditionnera l'exécution de la fonction et la levée du drapeau 55. Un arrêt de l'exécution ne lèvera pas le drapeau 55 comme pour l'imprimante 82143A, mais tester un drapeau, faire VIEW ou exécuter une fonction au clavier lèvera le drapeau 55.

Eviter la décompilation

Supposez que vous ayez enregistré un programme sur cartes magnétiques après l'avoir fait tourner une fois pour compiler tous les GTOs et XEQs. (Reportez-vous page 60 pour y trouver des explications sur la compilation). Lorsque vous relirez les cartes, les GTOs et XEQs seront toujours compilés, et la recherche des labels ne sera donc pas nécessaire. Néanmoins, les informations contenues dans les GTOs et XEQs seront perdues à la première tentative de GTO.. ou de PACK. Une technique synthétique simple découverte par Clifford Stern vous permet de PACKer sans perdre cette information.

Après avoir chargé le programme à partir du lecteur, passez en mode PRGM puis faites BST. Ceci vous place sur le .END., qui est la dernière ligne de votre programme. Assurez-vous qu'il y a au moins deux registres disponibles (.END. REG 02 ou plus). Pressez ENTER↑, STO IND 66, BST, BG, deux fois la flèche de correction, et PACK (pas GTO..). Le suffixe IND 66

se transforme en premier octet d'un END packé, ce qui empêche le processeur de remettre à zéro les informations de compilation. Aucun octet n'est perdu, car PACK enlève tous les nuls éliminables du programme. La présence du nouveau END élimine la décompilation qui suivrait normalement.

Cette méthode s'applique également aux programmes lus à partir de la cassette, de la mémoire étendue ou d'une autre source.

Distinctions entre ROM et RAM pour STO b

La plupart des pointeurs de RAM constituent des pointeurs de ROM valides (voir paragraphe 6A). La HP-41 doit se souvenir par un drapeau interne que le pointeur est en RAM ou en ROM. Ce drapeau ne peut pas être modifié par STO b.

Donc STO b ne peut être employé que pour sauter d'un endroit en ROM à un autre endroit en ROM, ou d'un endroit en RAM vers un autre endroit en RAM. Une erreur courante consiste à presser la touche assignée avec STO b avec un pointeur en ROM, espérant ainsi se placer à une position particulière en RAM. Ceci ne fonctionne pas. Vous devez d'abord exécuter CATalogue 1 (quitte à l'arrêter tout de suite avec R/S) pour revenir en RAM avant de presser STO b.

Raccourcis passant par le registre Q

Lorsque vous épelez un label ALPHA au clavier (en tapant un LBL, GTO ou XEQ par exemple), le nom est chargé dans le registre Q. Ce fait est utile lors de l'utilisation de eGOBEEP 77 au lieu de PRP. Par exemple, pour imprimer un programme contenant LBL "ABC", vous pouvez presser GTO ALPHA A B C ALPHA eGOBEEP 77. Vous pouvez même faire encore mieux en pressant eGOBEEP ALPHA A B C ALPHA, eGOBEEP 77. Ce dernier exemple utilise un fait obscur, découvert par Robert Edelen, qui conduit à ce que eGOBEEP nom a la même résultat que LBL nom.

Un autre raccourci utile, découvert par Clifford Stern, est de vider le registre Q en pressant ALPHA, flèche de correction. Vous pouvez alors obtenir une instruction TEXT 0 en pressant Q-LOAD (Données pour MK : 27, 0) et flèche de correction. Si vous pressez eGOBEEP après avoir vidé Q, vous imprimerez le programme courant, comme vous l'auriez fait en pressant PRP ALPHA ALPHA.

Utilisation de RA comme sous-programme

Si RA doit être appelé comme sous-programme, remplacez la ligne 38 (l'instruction OFF) par ALMNOW et RTN. ALMNOW relancera le compte à rebours interne pour l'alarme en attente.

Utilisation de EFT pour PCLPS

La très utile fonction PCLPS peut être exécutée par l'intermédiaire de EFT tant que EFT lui-même n'est pas effacé par PCLPS. PCLPS est la méthode la plus rapide pour effacer des programmes en mémoire principale.

Achevé d'imprimer par
la Société Pyrénéenne d'Impression S.A.
à Toulouse, 20 Avenue du Lauragais,
au cours du mois d'octobre 1984.
Dépôt légal, quatrième trimestre 1984.

"GONFLEZ" VOTRE HP-41
C'EST FACILE ET AMUSANT !

La programmation synthétique embrasse la création et l'usage d'instructions synthétiques, ces instructions qui ne peuvent pas être tapées de façon normale. L'application des instructions synthétiques inclu l'expansion des possibilités d'assignement (assigner SF 14 ou GTO IND X à une touche), 21 caractères supplémentaires pour l'affichage et la renumérotation des registres de données sous contrôle d'un programme.

Si vous avez entendu parler de la programmation synthétique et que vous voulez en savoir plus, ou si vous avez trouvé d'autres sources d'information sur la programmation synthétique confus ou difficiles à lire, essayez ce livre. HP-41 LA PROGRAMMATION SYNTHETIQUE, C'EST FACILE ! utilise les techniques et les programmes les plus récents et donne toutes les références sur les autres sources, qui seront toutes plus facile à lire aprés la lecture du présent livre. Les codes barre pour tous les programmes sont inclus pour les lecteurs qui ont accés à un lecteur de codes-barres.

Si vous aimez la HP-41, vous aimerez là programmation synthétique. Des milliers d'utilisateurs ont appris la programmation synthétique, et vous ?