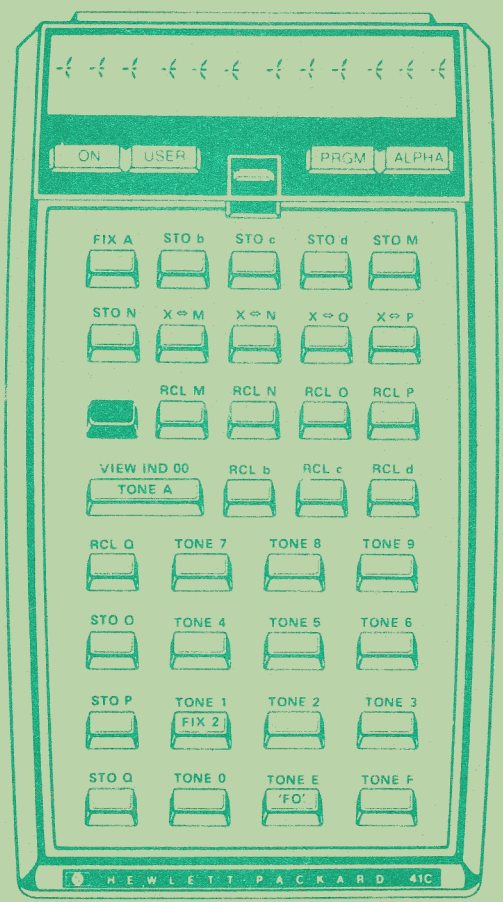
LA PROGRAMMATION SYNTHETIQUE DE LA HP-41

(Compatible HP-41C, CV et CX)

par W.C. Wickes



Editions du Cagire
Toulouse

EDITIONS DU CAGIRE 77 RUE DU CAGIRE 31100 TOULOUSE
FRANCE - TEL 61 44 03 06

La programmation synthétique de la HP-41

ERRATUM

Page 72, dans le programme 6F-6, ajouter une ligne manquante :

03 x<>y

Page 77, ajouter les indications suivantes :

Codes de la ligne 16 : F2 9F 7F
Codes de la ligne 24 : F2 7F 9F

LA PROGRAMMATION SYNTHETIQUE DE LA HP-41

(Compatible HP-41C, CV et CX)

par W.C. Wickes
Editions du Cagire
Toulouse

Avertissement de l'éditeur

Cet ouvrage a été traduit par Luc Mathieu, que nous devons remercier pour ce travail.

Il a également été tapé à la machine par le traducteur. Nous vous demandons donc votre indulgence pour quelques défauts de composition et fautes de frappe éventuelles.

Merci

© 1984 Editions du Cagire pour la traduction.
dépot légal 2ème trimestre 1984

Reproduction interdite

ISBN 2 86811 004 5

Remerciements

C'est un plaisir de remercier l'importante contribution de beaucoup de gens qui m'ont aidé dans la préparation de ce livre.
La couverture est de William Kolb, qui fit aussi une édition du manuscrit, prodigua encouragements, et appuya le projet d'un bout à l'autre. Tom Cadwallader, Keith Jarett, et John McGehie, tous des pionniers de la programmation synthétique, ont aussi apporté des articles et des suggestions inappréciables, avec Tom James et Lee Vogel.
Charles Close a contribué de façon importante à notre savoir approfondi sur le codage des lignes de programme et autres subtilités de la HP 41C. La decouverte du merveilleux 'sauteur d'octet ' par Roger Hill a été une avance de premiére importance pour faire de la programmation synthétique un processus agréable. Jacob Schwartz a developpé la table des caractères codes-barres de l'appendice 3. Richard Nelson, á travers ses louables efforts pour fonder et diriger le PPC, a developpé la communication internationnale nécessaire auprogrés commun de la programmation synthetique. Le professeur Carroll Alley de l'universite de Maryland a été le principal soutient de mes efforts en programmation synthetique depuis l'arrivée de notre HP 41C vierge.
Ce livre est dedicacé à ma femme, Susanne. Elle a été l'indispensable collègue pour la dactylo et les travaux de relecture. Je suis aussi reconnaissant à mes deux enfants, Kenny et Lara, qui m'ont permis d'utiliser leur HP 41C quand j'avais envie de m'amuser avec.

PROGRAMMATION SYNTHETIQUE DE LA HP 41CV
(appendice à la quatième édition)

Toutes les fonctions et techniques synthétiques decrites dans ce livre sont correctement opérantes sur HP 41CV; qui a été commercialisée aprés que ce livre ait été imprimé pour la première fois. Toutefois, le 'truc' d'enlever le module décrit au chapitre 3 pour implanter le 'sauteur d'octet' n'est pas possible sur HP 41CV.
La procedure suivante peut y etre substituée (comme d'ailleurs sur HP 41C) :

1 executer les pas 2 et 4 de la page 25
2 mettre en mode PRGM, taper la ligne 01 Lbl "abc"
3 executer Cat 1 (toujours en mode PRGM), appuyer sur R/S immediatement pour que 01 Lbl "ABC" apparaisse à l'affichage. presser XEQ ALPHA 'DEL ' ALPHA 001 ('4094' apparaitra un court instant, puis le .END.), appuyer sur BST poue voir'4093' DEC'. BST de nouveau (attendre) pour voir '4092 X<>Ø6 '. C'est la meme ligne que la ligne Ø6 décrite au pas 6 page 25.
4 continuer les instructions 7 à10 de la page 25, en remarquant que les lignes à effacer sont maintenant '4089' et '4088 '. Le Lbl Ø1 au pas 09 est absent.

Les possesseurs de HP 41CV devront sauter l'exemple du 'creepy man' de la page 1.
Ce 'truc' dépend d'une erreur dans le système des HP 41C et HP 41CV, qui peut disparaître dans les versions futures. Dans cette eventualité, les possesseurs de HP 41CV devront se servir du lecteur de codes barres pour pouvoir commencer la programmation synthetique.

Crash et autres désastres

Les crash de la 41C,i.e. quand l'affichage disparait ou se fige, et que le clavier devient inopérant, sont un risque de la programmation synthetique.. Aucun programme ou technique de ce livre ne causera de crash si les instructions sont suivies exactement. Mais les erreurs sont inévitables. Si vous plantez votre machine, ne vous inquietez pas, aucun dommage n'en résultera.. Essayer d'enlever et de remettre les piles. Si ça n'arrange rien (essayer plus ieurs fois), retirez tous les périphériques et les modules enfichables avant de retirer les piles. En dernier ressort, (je n'en suis jamais arrivé là)retirer les piles toute la nuit.
Le contenu de ce livre est livré sans aucune garantie d'aucune sorte; l'éditeur, Larken Publications, et l'auteur n'assument aucune responsabilité. Il en est de même du traducteur et de l'éditeur de la traduction.

Copyright William C.Wickes
Publié par:
Larken Publications,4517 NW Queens Avenue,Corvallis,OREGON 97330, USA
Sixieme édition, Mars 1982

CHAPITRE 1

LES POURQUOIS ET LES COMMENTS

1A Programmation synthetique ?

Personne,de l'étudiant sérieux en sciences de l'ordinateur à l'utilisateur occasionnel d'une machine 4 opérations, ne peut manquer d'être impressionné par le calculateur HP 41 . Cette machine combine un pouvoir de programmation étonnant avec une compléte portabilité. L'acheteur éventuel est attiré par la longue liste des fonctions pré-programmées du calculateur. Le possesseur expérimenté remarque que sa HP 41C tient une place toujours plus importante dans la résolution des problêmes techniques, alors qu'il maîtrise la programmation et intègre ses propres inventions aux fonctions pré-programmées.
Et maintenant, quant un utilisateur a appris tout ce que le 'guide de l'utilisateur' peut lui apprendre, voici autre chose: la liste des fonctions et des capacités de programmation de la HP 41 n'est pas limitée aux propriétés cataloguées dans le guide. Il existe, en fait, une classe compléte de fonctions et de programmes d'application qui peuvent être utilisés pour rehausser grandement la puissance du calculateur, bien que ces fonctions ne puissent, à priori, pas être exécutées ou programmées par les touches normales. Les nouvelles fonctions, qui sont 'synthetisées' à partir de nouvelles combinaisons d'octets normaux,sont appelées 'fonctions synthétiques', d'où le titre de ce livre.
Pour vous mettre en appetît, voici un aperçu de quelques applications typiques de programmation synthétique qui sont impossibles ou irréalisables sans les techniques décrites ci-aprés

*** addition de 31 'nouveaux' caractères d'affichage

*** transformation du registre alpha en 4 registres additionnels de données. Ces registres peuvent devenir une zone de réserve pour un programme sans affecter le contenu des autres registres de données (utilisés par d'autres programmes). En plus, le contenu de ces registres peut être lu et écrit par le lecteur de cartes (fonction WSTS).

*** accés au contrôle des 56 drapeaux utilisateur et système. Exemple: deux appuis de touches peuvent effacer les 56 drapeaux simultanément.

*** détection automatique de la taille mémoire (SIZE) en moins de 2 secondes.

*** classement alphabétique rapide de données alphanumériques.

*** travail en chaines alphanumériques.

*** addition de six nouvelles fréquencees plus une variation de la durée pour TONE.

*** interchangeabilité des lignes programmes et des données.

*** amélioration du contrôle des assignationns , incluant les assignations de fonctions de deux octets (ex: STO 56), effacement automatique de toutes les assignations,et compactage du registre d'assignation.

Un exercice simple vous initiera au monde de la programmation synthétique, et vous motivera sans doute pour réaliser l'effort de lire le reste de ce livre. Essayer le tour de passe-passe suivant:

1 insérer un module dans la HP 41C
2 executer un MEMORY LOST
3 faites SIZE 063 (si le module est double densité, SIZE 127)
4 mettre en mode PRGM
5entrez les lignes programmes :
    01 12345
    02 STO IND 17
    03 RDN
6éteindre la HP 41C
7 retirer le module .attendre 1 minute. le remettre
8 allumer la HP 41C
9 appuyer sur return (RTN)
10 entrer 1.435245455 E 59
11 appuyer sur SST
12 mettre en mode ALPHA

d'où vient ce petit 'creepy man'? Mettez en mode PRGM, pressez BST et vous verrez la ligne 01 STO M. Cette ligne synthétique est la combinaison de l'octet IND 17 et de l'octet RDN qui résulte du fait que vous éliminez l'octet STO de STO IND 17 en retirant le module.
Pas d'allusion à l'existence de la fonction STO M dans le 'guide de l'utilisateur', mais vous allez connaître et aimer STO M et ses amis lorsque vous maîtriserez la programmation synthétique.

1B BUT ET ORGANISATION

Ce livre est fait pour communiquer les joies et révéler les mystères de la programmation synthétique à tout utilisateur de HP 41 , depuis le programmeur novice jusqu'à l'expert. C'est un résumé de la théorie des opérations du calculateur qui rendent possible la programmation synthétique, les procédures d'implantation, les lignes synthétiques de programme, et un éventail de programmes d'exploitation qui servent aux applications pratiques et illustrent aussi l'utilité des techniques exotiques de programmation.

* Le chapître 2 décrit le fonctionnement interne de la HP 41 avec un point de vue conceptuel qui attirera sans doute le respect des ingénieurs informaticiens. Vous acquiérerez là une méthode pour programmer le calculateur à un niveau plus approfondi que celui donné par le 'guide de l'utilisateur ' seul. A côté de ce défrichement de la programmation synthétique, le châpitre 2 vous donnera une image du fonctionnementde la HP 41 qui vous aidera à optimiser votre travail de programmation.
* Le chapître 3 introduit la première et la plus importante des fonctions synthétiques, le 'sauteur d'octet'. Cette fonction ne nécessite qu'une pression de touche et ouvre la porte à des procédures simples pour créer l'éventail complet des lignes synthétiques de programme. Nous créerons le 'sauteur d'octet' en utilisant la substitution de module de la même manière que nous l'avons fait pour STO M; une fois fait, nous n'aurons plus jamais besoin de retirer le module.
* Dans le chapître 4, on introduit de nouveaux registres de la HP 41 , les 'registres d'etat'. De l'accés à ces registres, qui incluent le registre alpha, les 56 drapeaux,la répartition mémoire, le pointeur programme et la pile de retour des sous-programmes, résulte une foule d'applications pratiques comme l'exemple donné en 1A .
* 'DES programmes pour programmer',une enveloppe des techniques et programmes pour HP 41 , sont décris dans le chapître 5. la principale utilité de ces programmes est de permettre l'ecriture et le déchiffrage des autres programmes.
* Le chapître 6 est un chapître d'applications standards , dans lequel vous trouverez un choix de programmes synthétiques qui sont en eux mêmes une justification suffisante de l'étude du contenu des chapîtres suivants. Mais en plus, les programmes illustrent les techniques de programmation synthétiques qui ont leur application dans une large gamme de sujets, limités seulement par la motivation et l'ingéniosité de l'utilisateur éclairé.
* Finallement, dans le chapître 7, nous apprendrons quelques trucs amusants qui n'ont pas particuliérement d'aspect pratique, mais qui réjouiront le coeur du programmeur confirmé. Avec, bien sur, comme suprême exemple d'un enorme effort de recherche pour découvrir un résultat tout à fait inutile : comment faire faire demi-tour à ce sacré canard et le faire voler dans l'autre sens !

Trois appendices sont inclus : l'appendice 1 est un bref aperçu des systèmes de numération binaire, décimal, octal et héxadécimal. Si vous n'êtes pas familier avec, ou avez oublié ces notations, vous trouverez utile d'étudier l'appendice 1 avant d'attaquer le chapître 2. L'appendice 2 contient les codes-barres des programmes importants du chapître V : "code"; "reg"; "ka" et "decode", plus le long programme "HANGMAN" du chapitre 6. L'appendice 3 contient des codes-barres spéciaux pour des lignes synthétiques spéciales.

1C L'ORIGINE DE LA PROGRAMMATION SYNTHETIQUE

Tout est venu par accident ! Les premiers modèles de la HP 41 avaient un défaut involontaire ou 'bogue', dans leur codage interne, qui permettait l'exécution de l'opération STO IND 01,par exemple, avec des valeurs de 719 à 999 dans le registre de données R01 . Cette opération provoquait le stockage du registre X dans la mémoire programme. Je me demandais ce qui se passerait si j'utilisais cette caractéristique pour synthétiser de nouvelles lignes de programme en stockant des nombres dans le programme qui formerait ensemble des combinaisons impossibles d'octet normaux. En bref, ça à marché. Une fois les nouvelles fonctions implantées dans ma mémoire, elles pouvaient être enregistrées sur carte magétique et elles émergeaient dans n'importe quel programme. . Les applications pratiques arrivèrent en foule .

Suivant la découverte des fonctions synthétiques, les progrés majeurs en programmation synthétique furent 1) le développement des assignations synthétiques, qui permirent d'exécuter de nouvelles fonctions par appui sur une seule touche, et permirent de se passer du bogue hardware; 2) la découverte du sauteur d'octet, qui est sans doute la fonction synthétique fondamentale. Puisque le sauteur d'octet peut être généré sur n'importe quelle HP 41 et qu'il peut être utilisé pour créer à peu prés toutes les autres lignes programme, nous pouvons donner à la HP 41 une gamme compléte de programmes synthétiques.

1D PAS DE RISQUES POUR LA HP 41C

Les fonctions synthétiques, quand elles sont entrées dans la HP 41 avec les méthodes décrites dans ce livre, sont des opérations 'propres' au calculateur. De ce fait , elles ne constituent pas de traitement physique pour la HP 41 . Le seul risque, qui devrait être réellement considéré comme un ennui potentiel plutôt que comme un danger, est que certaines fonctions synthétiques peuvent provoquer un 'MEMORY LOST' ou un crash, c'est à dire un état où l'affichage se fige, se vide et ou le clavier devient inopérant. Le premier désastre cause beaucoup de grincements de dents et de tremblements de mains, mais n'endommage certainement pas la HP 41 . Le second problème peut pratiquement toujours être rattrappé (99.9% des cas en enlevant les piles et en les replaçant immédiatement, et avec un ou deux ON/OFF .Je n'ai entendu parler que d'un cas ou un crash a demandé de retirer les piles toutes la nuit, mais la cause en est inconnue. Je ne peut rien garantir, bien sûr, mais en développant la programmation synthétique, , j'ai accidentellement effacé l mémoire et planté la machine des douzaines de fois et maintenant la HP 41 continue à fonctionner.
A cause du risque d'éffacement accidentel de la mémoire, toutefois, ce n'est pas à Hewlett-Packard de 'supporter' l'usage des fonctions synthétiques. Vous ne devez donc pas soumettre à la Librairie des Utilisateurs des programmes contenant des lignes synthétiques.

1E QUELQUES CONVENTIONS

Voici une liste de conventions spéciales que j'ai adoptées pour ce livre, pour simplifier la description des programmes, caractères, nombres, instructions ... du calculateur.
1.Vous avez peut-être remarqué l'emploi des guillemets simples à la place des guillemets doubles, comme pour 'exemple' au lieu de "exemple". Les guillemets doubles sont réservés pour indiquer les caractères alphanumériques et les textes de la HP 41 , comme l'imprimante identifie les caractères sur les listings. Cette convention est aussi utilisée dans les listings des programmes de ce livre, dont la plupart sont des listings d'imprimante. Vous devez aussi faire attention au fait que les lignes programme entre guillemets seront affichées sur la HP 41 précédées du symbole "►".
En plus, je viole délibérement les règles de la ponctuation concernant les virgules et les espaces entre guillemets quand ils sont adjacents,i.e. "abcd", au lieu de "abcd," . En plus d'être illogique, cette règle conduirait à des ambiguîtés inacceptables dans ce livre, puisque la virgule et l'espace sont des caractères standards de la HP 41 . Leur inclusion dans les guillemets pourrait suggérer qu'ils font partie de la chaine alphanumérique adjacente.
2.Autant que possible, pour donner un texte clair, les symboles standards de machine à écrire seront utilisés pour représenter les caractères affichables de la HP 41 . L'identification est sans doute évidente, avec une exception possible pour le point virgule ";" représantant le symbole HP 41 "▼" .
3.Les chiffres à l'affichage seront représentés de la même façon que s'ils étaient entrés dans le registre alpha par 'ARCL' . Les nombres au format SCI ou ENG seront listés avec le "E" pour indiquer l'exposant, donc '1.23 E10' plutôt que '1.23 10' pour éliminer la possibilité de confondre les espaces.
4.Les séquences du type 'STO mn', 'DEL lmn', ou 'SF mn' sont considérées comme des opérations ou les lettres l,m,n etc ... représentent les chiffres qui peuvent prendre n'importe quelle valeur associée à la fonction. Pour 'STO mn', m et n peuvent chacun prendre les valeurs de 0 à 9.
5.Pour le listing de longs nombres hexadécimaux ou binaires, il est souvent commode de grouper les chiffres pour l'explication. Le groupement est indiqué par des espaces ou des barres"/" placées dans les nombres qui ne sont pas contenues, bien sûr, dans le codage réel de la HP 41 .
6.Une unité de base pour la mémoire utilisateur de la HP 41 est le registre, un block de 7 octets qui peut être utilisé pour le stockage d'un nombre ou pour 7 octets de programme. Les registres seront identifiés comme suit : a) 'Registre lmn ' indique le registre à l'adresse 'lmn', où lmn est un nombre héxa-décimal. b) 'Registre a' ,ou 'a' est un caractère alphanumérique, indique l'un des 16 'registres d'état' détaillés au chapître quatre. Les registres X,Y,Z,T etL sont les registres habituels de la pile RPN .Les 11 registres restant : M,N,O,P,Q,►,a,b,c,d, et e sont ainsi nommés à cause de la façon dont affichées les 'fonctions synthétiques d'accés aux registres d'état', par exemple : STO M, RCL►, ISG d, etc ...c) Rmn indique le registre de données numéro'mn', où 'mn' est un nombre décimal à 2,éventuellement 3, chiffres.

7 Pour éviter les erreurs de transcription, les listings de programme de ce livre proviennent directement de l'imprimante 82143 . Malheureusement, l'imprimante liste les les fonctions d'accés aux registres d'état M,N,O,P,Q et ⊦ en utilisant des caractères différents de ceux de l'affichage de la HP 41 . Le lecteur devra se familiariser avec les tables suivantes :

TABLE 1-1

Symboles des registres d'état

Symbole de l'affichage	Symbole de l'imprimante
M	⊏
N	/
O	⊐
P	↑
Q	⊤
⊦	⊤

En plus, dans les lignes synthétiques de texte, l'imprimante utilise le symbole "◆" à la fois pour l'octet nul '00' (affiché par la HP 41 par"¯") et pour l'octet '0A' . Cependant, il n'y a aucune ligne contenant l'octet '0A' dans ce livre, donc le " " indiquera toujours l'octet '00'.

8 Les courts sous-programmes sans labels alphanumériques seront identifiés par un nombre entre parenthèses à droite de la routine. La règle en est : '(référence dans le chapître - numero de la routine)'.

9 Pour simplifier les instructions relatives à l'entrée des lignes de programme, ou à l'exploitation des programmes, ces instructions seront présentées la plupart du temps en trois colonnes. Dans la colonne de gauche se trouvent les codes à entrer en mémoire :nombres à mettre dans le regisre X, caractères alphanumériques à entrer dans le registre alpha, ou lignes de programme (présentées avec le numéro de ligne) à entrer dans le programme. La colonne du centre liste les séquences de touches, comme ' GTO .123' ou ' DEL 005', qui ne sont pas enregistrées par le programme. La colonne de droite montrera, quand il y aura lieu, l'affichage de la HP 41 résultant de chaque instruction de la colonne centrale. Cet affichage sera compris entre crochets . Exemple :

Au clavier	Opérations	Affichage
	GTO .000	00 REG 123
01 STO 01		
	SST	02 X<>Y

indique que vous devez presser 'GTO .000' (pour voir 00 REG 123);entrer la ligne '01 STO 01' puis faire SST une fois pour voir la ligne ' 02 X<>Y '.

1F NECESSAIRE

De façon à toucher la plus large majorité d'utilisateurs possible, ce livre est conçu autour d'un 'système HP 41 ' minimum. Tout ce dont vous aurez besoin , c'est 1) une HP 41 2) temporairement, un module mémoire, 3) et quelques heures pour vous servir de ce matériel. Le lecteur de cartes, l'imprimante et le lecteur de codes-barres ne sont pas nécessaires, bien qu'ils soient des accessoires intéressants de programmation synthétique, tout comme pour n'importe quel autre usage de la HP 41 . Le lecteur de cartes, par exemple, donne un moyen très sur pour rentrer vos nouveaux programmes et assignations sans provoqure d'effacement accidentel de la mémoire . L'imprimante est inappréciable pour lister les programmes au fur et à mesure que vous les entrez, et pour garder une trace de tout ce que vous faîtes. Si vous avez la chance d'avoir un lecteur de codes-barres disponible, vous pouvez vous dispenser de beaucoup d'entrées au clavier en utilisant les codes fournis aux annexes 2 et 3. Le chapître 2 est la pierre d'achoppement de la programmation synthétique, puisqu'il contient de nombreuses descriptions détaillées sans la joie de presser une seule touche du clavier. Je suggère de lire le chapître 2 relativement vite une première fois, avec juste assez d'attention pour saisir sa portée de façon à vous préparer aux plus interessantes entrées au clavier qui débutent avec le chapître 3.

Et, alors que vous poursuivrez votre étude, vous pourez vous référer aux différents détails du chapître 2.

1G REFERENCES

La plupart des découvertes et des techniques décrites dans ce livre ont été publiées une première fois dans différents numéros du PPC CALCULATEUR JOURNAL. Le PPC (les initiales n'ont pas de signification particulière) est un club indépendant internationnal de fanatiques de l'ordinateur de poche et qui ont en commun le désir d'étudier des applications pour les calculateurs programmables de Hewlett-Packard. Le journal est le principal moyen d'echanger des informations pour ceux qui développent la programmation synthétique, et qui sont dispersés dans le monde entier. Tout utilisateur sérieux de HP 41C, particulièrement s'il est interessé par la continuation de ce qu'il apprendra dans ce livre, trouvera du plus haut interêt de rejoindre le club et de s'abonner au journal. D'avoir des contact avec des centaines d'autres programmeurs peut vous épargner beaucoup de travail ! Les demandes sont à adresser à:

PPC PoB 9599 Fountain Valley, California, 92728-9599 USA
En France PPC-T 77 rue du Cagire 31100 Toulouse

Voici une liste d'articles se rapportant à la programmation synthétique (Volume, numéro, page):

Caldwallader T.'Improved synthetic key assignments' V7N3P3
Close C. 4Bug 2 : a practical application' V7N3P8
Hewlett-Packard 'HP 41C Function Table ' V6N4P11
'HP 41C Postfix Table' V6N5P11
'HP 41C Data & Program structure' V6N6P19
Istok G. 'Pseudo XROM's on the HP 41C' V7N2P32
Kennedy J. 'The HP 41C Combined Hex Table' V6N5P27
McGechie J. 'HP 41C Synthetic key assignments' V7N2P34
Nelson R. 'Bugs in the box' V6N5P27
Wickes W. 'Direct status register access on the HP 41C' V6N7P31
'Through the HP 41C with gun and camera' V6N8P27
' HP 41C black box programs' V6N2P35
'Freedom from bugs' V7N2P35
'Synthetic key assignments' V7N2P30
'Improved black box programs' V7N2P35
'HP 41C synthetic function routines ' V7N4P26
'Byte jumping, or the poor man's black box' V7N4P26
'Direct addressing of ROM routines' V7N5P55
'Understanding BLDSPEC ' V7N5P56

Il faut insister sur le fait que le developpement de la programmation synthétique est un procédé continu. Même pendant que ce livre est écrit, des programmeurs curieux mettent à jour de nouveaux trucs et approffondissent notre compréhension de la HP 41C. Même écrire ce livre a accru ma propre compréhension de la HP 41 , et a conduit à de nouvelles découvertes comme 'l'autorisateur de textes' décrit au chapître 5.

CHAPITRE 2

A l'intérieur de la HP 41

Ce chapître sera, en quelque sorte, une excursion dans le monde du fantastique . De facon à vous donner une conception utile du tavail de la HP 41 , je vais introduire certains systèmes fictifs, presque personnifiés, pour représenter les opérations importantes du calculateur .Ces systèmes peuvent ou non avoir leur correspondant électronique à l'intérieur de la HP 41 ; ces détails sont du ressort des ingénieurs électroniciens et n'entrent pas dans le cadre de ce livre .Ce qui est important est que la HP 41 se comporte comme si ces systèmes existaient. Tout d'abord il faut que vous conceviez le 'cerveau' du calculateur comme un dispositif appelé 'processeur' . Ce processeur est responsable de la lecture des données et des programmes stockés en mémoire, puis des ordres à donner aux autres systèmes du calculateur pour leur dire quoi faire avec ce qu'il vient de lire . Selon votre imagination, vous pouvez imaginer que le processeur est un oppresseur humain lisant fébrilement une partie d'instruction donnée par l'utilisateur, puis donnant ses propres instructions aux différents ' travailleurs' qui composent la HP 41 .

2A. LANGAGE DU CALCULATEUR : BITS , NYBBLES ET OCTETS

Énigme : qu'ont en commun le nombre '1.435245455 E59 , la chaîne alphanumérique "XCREEPY" et le programme:

```
01 LBL 00
02 /
03 SQRT
04 X>Y?
05 X>Y?
06 LN
07 SIN
```

Réponse: tous les trois sont stockés de façon identique dans la mémoire utilisateur de la HP 41 !! Comprendre ce concept apparemment obscur, c'est maîtriser la base de l'organisation et du codage de la mémoire toute entière . Par 'mémoire utilisateur',nous désignons la partie de la mémoire de la HP 41 qui est sous le contrôle de l'utilisateur : les registres de données et de programmes, d'assignation, de la pile RPN, alphanumériques etc ...

A un niveau élémentaire, un calculateur est vraiment un dispositif simple .Il peut stocker et rappeler des nombres, les additionner éventuellement, et voilà.Pour pouvoir mener à bien des instructions qui peuvent sembler élémentaires à l'utilisateur, comme '+' ou 'LN', le processeur doit mettre en route des séquences de douzaines de pas élémentaires . Le véritable pouvoir du calculateur repose en sa capacité de permettre à l'utilisateur d'initialiser ce processus interne par le simple moyen d'une pression de touche .Un calculateur programmable est celui qui peut aussi coder la séquence de touches, et stocker le code pour une exécution automatiquement répétée .

La HP 41 représente une avance décisive sur les calculettes de poche précédentes dans ce sens que les codes des programmes utilisateurs sont affichés en mots et caractères alphanumériques directement lisibles .Nous pouvons imaginer la HP 41 comme contenant un invisible traducteur qui prend une partie du code stocké et le transforme en un nombre ou un mot affichable . Mais, en fait, il faut aussi un autre traducteur pour donner au calculateur la séquence d'instructions élémentaires, appelés microcodes, à éxécuter .Il y a, en effet, trois niveaux d'interprétation pour un même code stocké, comme le montre la figure 2-1 .

Le premier niveau est la traduction des codes en une forme directement visualisable par l'utilisateur . L'entrée de l'utilisateur consiste à générer des codes en pressant des touches; les codes résultants sont relus par le 'traducteur utilisateur' sous forme de nombres, caractères, lignes programme visualisées à l'affichage .Au niveau 2, les codes sont transformés pour être stockés en mémoire, ou pour être lusou écrits sur un autre dispositif comme le lecteur de carte ou le lecteur de codes-barres . Finallement, un 'traducteur machine' est nécessaire pour traduire les codes au niveau 3, i.e. la séquence d'instructions machine élémentaires requises pour mener une opération.

Dans le principe, nous sommes intéressés par les niveaux 1 et 2 . 'La programmation synthétique ' est le processus pour créer de nouveaux codes de niveau 2 en court-circuitant la logique du clavierqui réduit les entrées aux instructions listées dans le guide de l'utilisateur . Les codes synthétiques résultants peuvent être interprétés par tous les traducteurs , produisant souvent des résultats pratiques comme des nouveaux caractères affichables et fonctions programmables . Pour terminer, l'utilisateur doit apprendre à parler le niveau deux pour qu'il puisse intervenir directement sur les codes stockés sans dépendre du 'traducteur utilisateur' .

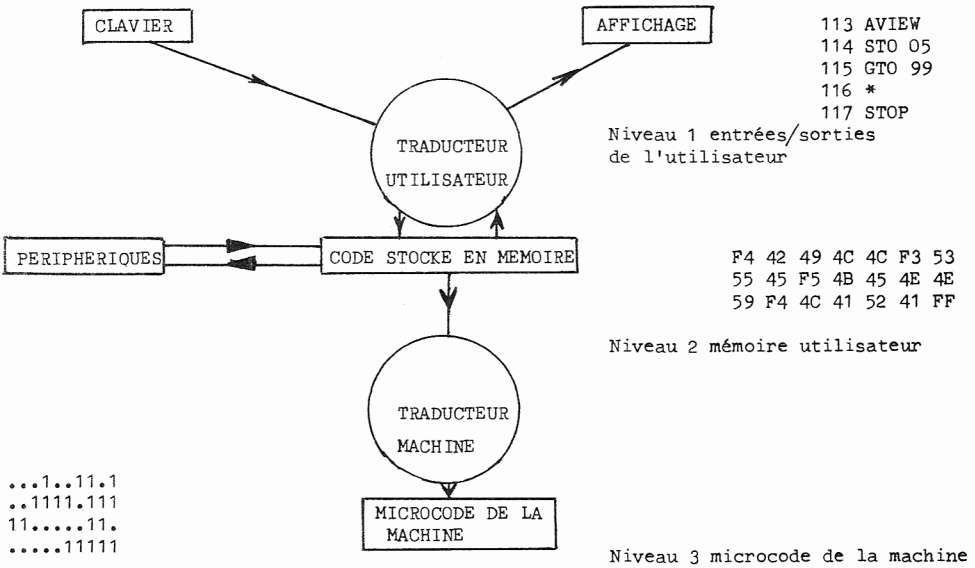


FIGURE 2-1 LES TROIS NIVEAUX DE CODAGE DE LA HP 41

La nécessité pour les codes mémoires d'être stockés dans la mémoire de la HP 41 et aussi de pouvoir être lus ou écrits depuis un périphérique dicte la forme que ces codes doivent prendre . Le lecteur de cartes, par exemple, utilise des cartes magnétiques pour le stockage des codes; la carte par elle-même ne peut contenir que des informations stockées comme des parties ordonnées d'éléments magnétiques dans le film d'oxyde de la carte Pour avoir un stockage reproductible et digne de confiance, l'ordonnancement doit être aussi simple que possible : le code est figuré par une petite partie de carte sur laquelle figure des régions, magnétisées ou non, en forme de barres. Ce concept est le même que pour les code-barres utilisés par le lecteur de code-barres. Alors que le lecteur est promené le long de la ligne des barres, il 'voit' des barres blanches ou noires. Une série de barres peut être considérée comme un long nombre binaire avec les barres larges représentant '1' et les barres étroites représentant '0'. La figure 2-2 est un échantillon de code-barres, qui peut être utilisé pour visualiser comment les codes sont stockés sur une carte magnétique, et, bien sûr, dans la HP 41 elle même .

A l'intérieur du calculateur, les 1 et les 0 sont représentés par les états de microscopiques transistors, mais l'idée directrice est la même que pour le lecteur de cartes et le lecteur de code-barres : les codes utilisateur sont stockés en sections d'une longue chaîne de bits binaires . Pour donner un sens au code, le processeur, doit savoir comment briser la chaîne en sections compréhensibles .
Considérons de nouveau le nombre '1.435245455 E59' .Un simple compte montre qu'il faut 14 informations pour représenter ce nombre sous forme décimale : 10 digits de mantisse, 2 d'exposant, 1 pour le signe de l'exposant et 1 pour le signe de la mantisse .

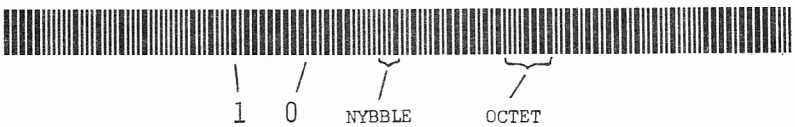


FIGURE 2-2 CODE BARRE DE LA HP 41

L'unité de base ou block du code stocké doit pouvoir représenter une de ces informations, i.e , elle doit pouvoir prendre au moins dix valeurs pour représenter un chiffre décimal . Les chiffres décimaux 0 à 9 sont représentés en binaire par les valeurs de 0000 à 1001, d'où nous concluons que l'unité de base doit comprendre 4 bits consécutifs . Cette unité est appelée 'nybble' - c'est à dire la moitié d'un octet, comme nous verrons- . Nous nous référerons aussi au nybble comme à un digit, conséquence de son rôle de stockage d'un nombre . Le nombre décimal précité est codé ainsi sur 14 nybbles :

0000 0001 0100 0011 0101 0010 0100 0101 0100 0101 0101 0000 0101 1001
\+ 1 4 3 5 2 4 5 4 5 5 \+ 5 9

Les espaces sont donnés pour plus de clarté . Le 'E' et le '.' n'ont pas besoin de code explicite puisque leur position et leur 'valeur' ne change pas.Pour les digits de signe, le premier et le douzième nybble à partir de la gauche, la HP 41 met '0000' pour '+' et '1001' pour '-' .
Vous avez peut-être déja vu que les quatre bits nécessaires pour représenter un digit décimal peuvent prendre jusqu'à la valeur 1111, 15 en décimal. Le code peut représenter de l'hexadécimal aussi bien que du décimal. Cette possibilité serait gâchée si la HP 41 travaillait uniquement avec des nombres décimaux. Cependant, même un nybble de 4 bits est inadéquat pour coder les lignes programme .
La HP 41 utilise 2 nybbles consécutifs, 8 bits, comme unité élémentaire pour les codes programme, appelé 'octet'. 11111111 (FF en héxadécimal) en binaire est la même chose que 255 en décimal, il y a donc 256 possibilités élémentaires de code programme, ce qui est suffisant même pour un calculateur de la puissance de la HP 41 .Quand la HP 41 fait tourner un programme, nous pouvons l'imaginer 'octetant' successivement par 8 bits de code pour opérer. La signification de l'énigme du début de ce chapître doit maintenant commencer à devenir claire.Le nombre '+1.435245455 E+59' est stocké sur 7 octets, '01 43 52 45 45 50 59'. Quand les mêmes octets sont dans la mémoire programme, ils représentent les lignes programme 'LBL 00', '/', 'SQRT', 'X>Y?', 'X>Y?', 'LN', et 'SIN'. Le traducteur utilisateur de la HP 41 est bien plus sophistiqué que nous avons pu le croire . La traduction pour l'affichage ne dépend pas seulement du code lu par le traducteur, mais aussi du mode (PRGM, ALPHA, etc..) du calculateur. L'énigme suggère maintenant une troisième façon de traduire le même code: si le code de l'exemple était dans le registre alpha, il serait affiché comme les 7 caractères alphanumériques "ΣCREEPY".
La sortie du traducteur utilisateur est l'affichage . Tout ce que vous voyez à l'affichage de la HP 41 est la traduction du contenu d'un registre en un affichage alphanumérique. Quand vous mettez le calculateur pour la première fois sous tension, il est en mode 'par défaut', dans lequel le contenu du registre X est copié dans l'affichage comme un nombre, avec chaque caractère numérique représentant un digit du registre X. Si la HP 41 est mise en mode ALPHA, le registre alpha est copié, avec un caractère affiché par octet de registre alphanumérique. Si un 'VIEW nn' est exécuté, le drapeau 50 est mis pour indiquer un affi-

chage non par défaut, avec le contenu du registre Rmn affiché. Ce système d'affichage permet la visualisation sans affecter le registre X. De la même façon, nous pouvons voir le contenu du registre alpha en utilisant 'AVIEW' . Un 'CLD' baisse le drapeau 50, restaurant le mode par défaut. En mode programme, l'affichage montre une ligne programme faite d'octets de programme. Pendant qu'un programme tourne, le 'canard' est l'affichage par défaut, qui peut être remplacé par le biais d'une instruction 'VIEW' ou 'AVIEW'. La figure 2-3 montre la logique impliquée par le procédé d'affichage.
Nous avons vu que les programmes utilisateur et les données stockés dans la HP 41 sont codés par une longue chaîne de '0' et de '1' appelés bits . Dans la mémoire utilisateur, chaque groupe successif de 4 bits, appelé nybble, peut représenter un digit décimal, ou le signe pour la mantisse ou l'exposant. Comme un nombre demande 14 nybbles, 14 nybbles successifs sont stockés et rappelés ensemble depuis une unité de stockage appelée 'registre'. L'opération 'RCL 01', par exemple, demande au calculateur de copier les 56 bits trouvés dans la section de mémoire appelée R01 dans une autre section appelée registre X. Dans la mémoire programme, le code est rappelé et stocké un ou plusieurs octets à la fois. Comme indiqué dans le chapitre suivant, chacun des 256 octets possibles représente une série unique d'instructions de programme. La division de la mémoire en registres est plus apparente dans la mémoire programme que dans la mémoire de données, mais le mode d'adressage décrit au chapître 2-C, néanmoins, est organisé par registres de 7 octets, de sorte que les registres sont interchangeables entre le stockage des données et des programmes.

2B. LA TABLE D'OCTETS

Avant de continuer avec une discussion sur le mode d'addressage utilisé par la HP 41 , voyons plus en détails le codage des lignes de programme. L'élément du code programme est l'octet ; chaque octet a 256 valeurs possibles, de l'héxa 00 à FF. Cependant, il y a bien plus que 256 lignes programme différentes, cette possibilité est offerte en donnant aux lignes de programme un ou plusieurs octets, jusqu'à un maximum de 16. De ce fait, bien que l'affichage montre ce qui pourraît passer pour une seule instruction, une ligne de programme, cette seule ligne peut en fait être constituée de plusieurs octets ou codes stockés.
La table 2-1, 'table d'octets de la HP 41 ', montre les 256 octets possibles dans une grille 16X16. Cette table est un outil puissant, indispensable pour la programmation synthétique, aussi il est important pour le nouveau programmeur de comprendre son maniement. Elle est, en effet, le dictionnaire utilisé par le traducteur utilisateur. Les nombres 0-F sur l'en-tête horizontale de la table représentent le second nybble ou digit d'un code d'octet. Les nombres de la colonne verticale gauche donnent le premier digit de l'octet. L'intersection d'une ligne et d'une colonne liste un certain nombre de figures ,i.e. les différentes façons dont l'octet correspondant peut être interprété, en fonction de sa position dans la mémoire. La figure 2-4 montre un exemple, en utilisant des entrées fictives pour illustrer toutes les possibilités.
Le premier nombre dans la case, dans le coin supérieur gauche, est l'équivalent décimal de la valeur de l'octet à deux digits héxa. Ce nombre est aussi la valeur utilisée par la fonction 'ACCHR' de l'imprimante pour obtenir le caractère dessiné dans la case à la droite du nombre décimal. (Les équivalents décimaux seront aussi utilisés comme entrées pour le programme d'assignations de touches "KA" décrit au chapître 5.) Par exemple dans la case 34(ligne 3, colonne 4), nous voyons le nombre décimal 52 (3X16+4=52), et le caractère correspondant de l'imprimante "4".
L'entrée suivante dans chaque case est une fonction de la HP 41C. Pour les octets des lignes 0 à 8 (excepté pour les octets 1D et 1E) , chaque octet constitue une ligne programme à lui seul. L'octet 34 est affiché et executé comme 'sto 04', l'octet 5C comme 'ASIN', etc. Ces octets peuvent être appelés fonctions à un seul octet, ou octets seuls, puisqu'ils demandent des opérations qui sont indépendantes des octets suivants du programme.
La HP 41C se départage de ses prédecesseurs Hewlett-Packard en offrant des 'lignes' programme à plusieurs octets plutôt que des 'pas' à un seul octet. Les octets de la ligne 9, les octets A8 à AE, et les octets CE et CF sont des octets de préfixe pour les lignes programme à 2 octets. Quand le processeur rencontre un de ces octets, il doit aussi regarder l'octet suivant pour compléter l'instruction programme. Par exemple, l'octet 90 est le préfixe 'RCL', qui demande un second octet, ou 'postfixe', pour identifier le registre à rappeler. La valeur du postfixe de chaque octet est donné par le nombre ou la lettre

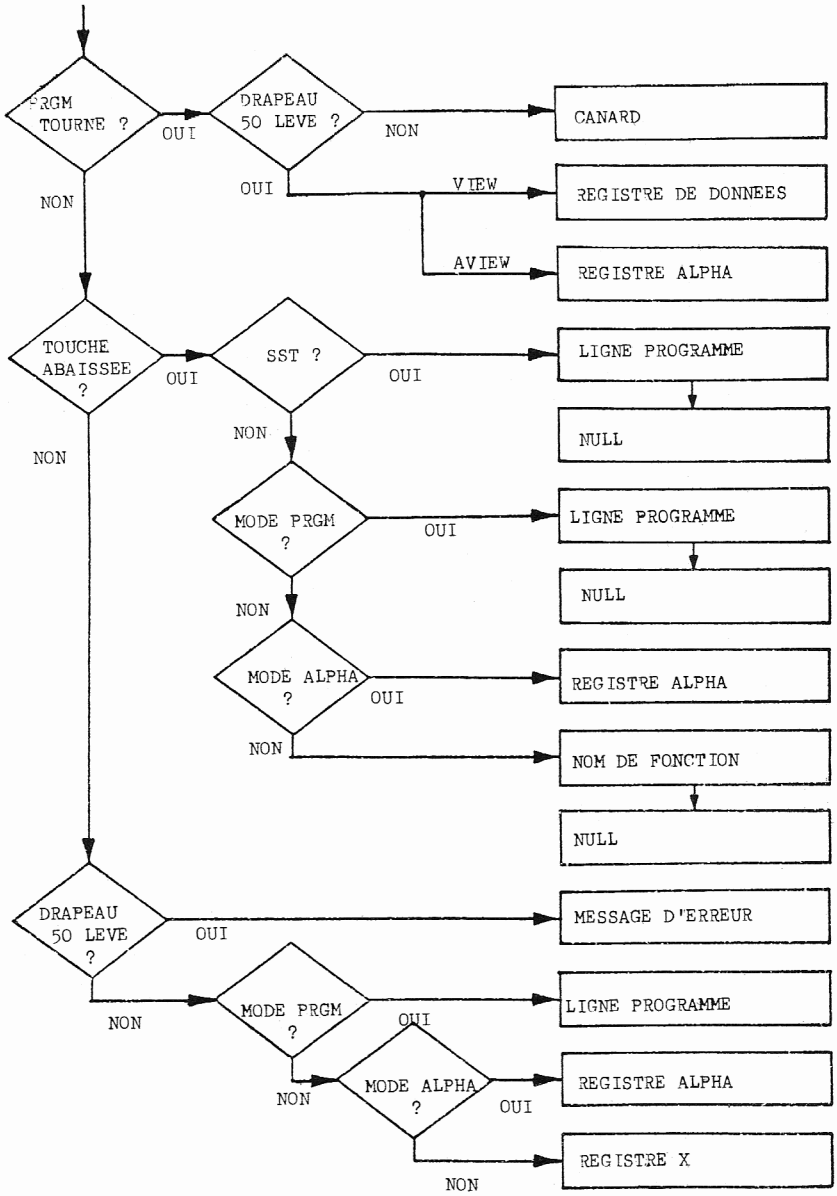


FIGURE 2-3 LOGIQUE D'AFFICHAGE

TABLE 2-1. LA TABLE D'OCTETS DE LA HP 41C.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0 NULL 00 _	1 × LBL 00 01	2 x̄ LBL 01 02	3 ← LBL 02 03	4 α LBL 03 04	5 β LBL 04 05	6 Γ LBL 05 06	7 ↓ LBL 06 07	8 Δ LBL 07 08	9 σ LBL 08 09	10 ◆ LBL 09 10	11 √ LBL 10 11	12 μ LBL 11 12	13 ∠ LBL 12 13	14 ┐ LBL 13 14	15 ▼ LBL 14 15	0
1	16 ▶ 0 16	17 Ω 1 17	18 δ 2 18	19 À 3 19	20 á 4 20	21 Ä 5 21	22 ä 6 22	23 Ö 7 23	24 ö 8 24	25 Ü 9 25	26 ü . 26	27 æ EEX 27	28 æ CHS 28	29 ≠ GTO α 29	30 £ XEQ α 30	31 ▒ SPARE 31	1
2	32 RCL 00 32 (space)	33 ! RCL 01 33	34 " RCL 02 34	35 # RCL 03 35	36 $ RCL 04 36	37 % RCL 05 37	38 & RCL 06 38	39 ' RCL 07 39	40 (RCL 08 40	41) RCL 09 41	42 * RCL 10 42	43 + RCL 11 43	44 , RCL 12 44	45 − RCL 13 45	46 . RCL 14 46	47 / RCL 15 47	2
3	48 0 STO 00 48	49 1 STO 01 49	50 2 STO 02 50	51 3 STO 03 51	52 4 STO 04 52	53 5 STO 05 53	54 6 STO 06 54	55 7 STO 07 55	56 8 STO 08 56	57 9 STO 09 57	58 : STO 10 58	59 ; STO 11 59	60 < STO 12 60	61 = STO 13 61	62 > STO 14 62	63 ? STO 15 63	3
4	64 @ + 64	65 A - 65	66 B * 66	67 C / 67	68 D X<Y? 68	69 E X>Y? 69	70 F X≤Y? 70	71 G Σ+ 71	72 H Σ- 72	73 I HMS+ 73	74 J HMS- 74	75 K MOD 75	76 L % 76	77 M %CH 77	78 N P-R 78	79 O R-P 79	4
5	80 P LN 80	81 Q y^2 81	82 R SQRT 82	83 S y^x 83	84 T CHS 84	85 U e^x 85	86 V LOG 86	87 W 10^x 87	88 X e^x-1 88	89 Y SIN 89	90 Z COS 9	91 [TAN 91	92 \ ASIN 92	93] ACOS 93	94 ↑ ATAN 94	95 _ DEC 95	5
6	96 ‾ 1/X 96	97 a ABS 97	98 b FACT 98	99 c X≠0? 99	100 d X>0? 00 100	101 e LN1+X 01 101	102 f X<0? A 102	103 g X=0? B 103	104 h INT C 104	105 i FRC D 105	106 j D-R E 106	107 k R-D F 107	108 l HMS G 108	109 m HR H 109	110 n RND I 110	111 o OCT J 111	6
7	112 p CL Σ T	113 q X<>Y Z	114 r PI Y	115 s CLST X	116 t R↑ L	117 u RDN M	118 v LASTX N	119 w CLX O	120 x X=Y? P	121 y X≠Y? Q	122 z SIGN	123 { X≤0? a	124 | MEAN b	125 → SDEV c	126 Σ AVIEW d	127 ⊢ CLD e	7
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	128 ← DEG 00	129 × RAD 01	130 ÷ GRAD 02	131 ← ENTER 03	132 α STOP 04	133 β RTN 05	134 Γ BEEP 06	135 ↓ CLA 07	136 Δ ASHF 08	137 σ PSE 09	138 ← CLRG 10	139 λ AOFF 11	140 μ AON 12	141 ∡ OFF 13	142 ▽ PROMPT 14	143 ▼ ADV 15	8
9	144 θ RCL 16	145 Ω STO 17	146 δ STO+ 18	147 á STO- 19	148 à STO* 20	149 ä STO/ 21	150 â ISG 22	151 ë DSE 23	152 ö VIEW 24	153 ô ΣREG 25	154 ü ASTO 26	155 ê ARCL 27	156 æ FIX 28	157 ≠ SCI 29	158 £ ENG 30	159 ¥ TONE 31	9
A	160 XROM 32	161 ! XROM 33	162 " XROM 34	163 # XROM 35	164 $ XROM 36	165 % XROM 37	166 & XROM 38	167 ' XROM 39	168 (SF 40	169) CF 41	170 * FS?C 42	171 + FC?C 43	172 , FS? 44	173 - FC? 45	174 . GTO IND XEQ IND 46	175 / SPARE 47	A
B	176 0 SPARE 48	177 1 GTO 00 49	178 2 GTO 01 50	179 3 GTO 02 51	180 4 GTO 03 52	181 5 GTO 04 53	182 6 GTO 05 54	183 7 GTO 06 55	184 8 GTO 07 56	185 9 GTO 08 57	186 : GTO 09 58	187 ; GTO 10 59	188 < GTO 11 60	189 = GTO 12 61	190 > GTO 13 62	191 ? GTO 14 63	B
C	192 @ GLOBAL 64	193 A GLOBAL 65	194 B GLOBAL 66	195 C GLOBAL 67	196 D GLOBAL 68	197 E GLOBAL 69	198 F GLOBAL 70	199 G GLOBAL 71	200 H GLOBAL 72	201 I GLOBAL 73	202 J GLOBAL 74	203 K GLOBAL 75	204 L GLOBAL 76	205 M GLOBAL 77	206 N X < > 78	207 O LBL 79	C
D	208 P GTO 80	209 Q GTO 81	210 R GTO 82	211 S GTO 83	212 T GTO 84	213 U GTO 85	214 V GTO 86	215 W GTO 87	216 X GTO 88	217 Y GTO 89	218 Z GTO 90	219 [GTO 91	220 \ GTO 92	221] GTO 93	222 ↑ GTO 94	223 _ GTO 95	D
E	224 ` XEQ 96	225 a XEQ 97	226 b XEQ 98	227 c XEQ 99	228 d XEQ 00 100	229 e XEQ 01 100	230 f XEQ A 102	231 g XEQ B 103	232 h XEQ C 104	233 i XEQ D 105	234 j XEQ E 106	235 k XEQ F 107	236 l XEQ G 108	237 m XEQ H 109	238 n XEQ I 110	239 o XEQ J 111	E
F	240 p TEXT 0 T	241 q TEXT 1 Z	242 r TEXT 2 Y	243 s TEXT 3 X	244 t TEXT 4 L	245 u TEXT 5 M [	246 v TEXT 6 N \	247 w TEXT 7 O]	248 x TEXT 8 P ↑	249 y TEXT 9 Q _	250 z TEXT 10 ← →	251 π TEXT 11 a	252 | TEXT 12 b	253 → TEXT 13 c	254 Σ TEXT 14 d	255 ← TEXT 15 e	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

listé immediatement sous le nom de la fonction dans la case de la table d'octet.Pour déchiffrer les octets '904C', par exemple, nous voyons dans la table que le premier octet, 90 est le préfixe 'RCL', donc nous devons regarder le second octet, 4C, comme un postfixe, spécifiquement 76. Donc les octets '904C' constituent la ligne 'RCL 76'. Similairement, 921D est 'STO 29', A803 est 'SF 03', etc.Notez que pour les octets 00à63, la valeur du postfixe pour l'octet est la même que l'équivalent décimal de la valeur de l'octet. Les cinq premiers octets de la ligne 7, quand ils sont utilisés comme postfixes, donnent accés aux registres de la pile X,Y,Z,T,L (Last X), donc 9170 est 'STO T', 9373 est 'VIEW X', et ainsi de suite. Certains des octets de la ligne 6 et de la ligne 7 sont montrés avec 2 valeurs de postfixe , l'une ou les deux étant 2 fois sous-lignée . Le sous-lignage indique une valeur de postfixe qui n'est accessible que par programmation synthétique. Ces valeurs alternatives seront expliquées au chapître 4.Les valeurs de postfixe des lignes 0 à 7 sont dupliquées dans les lignes 8 à F, nous changeant apparemment peu de 127 postfixes. Cependant, ce n'est pas une réelle duplication : les postfixes de la demi-table inférieure permettent l'exécution indirecte des fonctions préfixées. Par exemple, 9152 est 'STO 82', mais 91D2 est 'STO IND 82'. Cette configuration permet l'utilisation de n'importe quel registre de donnée (de R00 à R99) à partir de l'addressage indirect. Autres exemples : AAAA est 'FS?C IND 42'; 9D8F est 'SCI IND 15' ; 9F86 est 'TONE IND 06'.
L'octet 'AE' a un rôle double quand il est utilisé comme préfixe. Si le postfixe est de la demi-table supérieure, AE est exécuté comme 'GTO IND'; si le postfixe est de la demi-table inférieure, AE devient 'XEQ IND'. Par exemple, AE2A est 'GTO IND 42', alors que AEAA est 'XEQ IND 42'.
Les octets A0 à A7 ont 'XROM' à la place de leur nom de fonction. Ces octets sont des préfixes, mais pas tout à fait dans le sens décrit précédemment.Chaque fonction de périphérique, comme 'WDTA' ou 'ACSPEC', y compris les fonctions non programmables 'WALL' et 'LIST' , a un code unique de 2 octets associé, décrit dans la table entre A000 et A7FF.

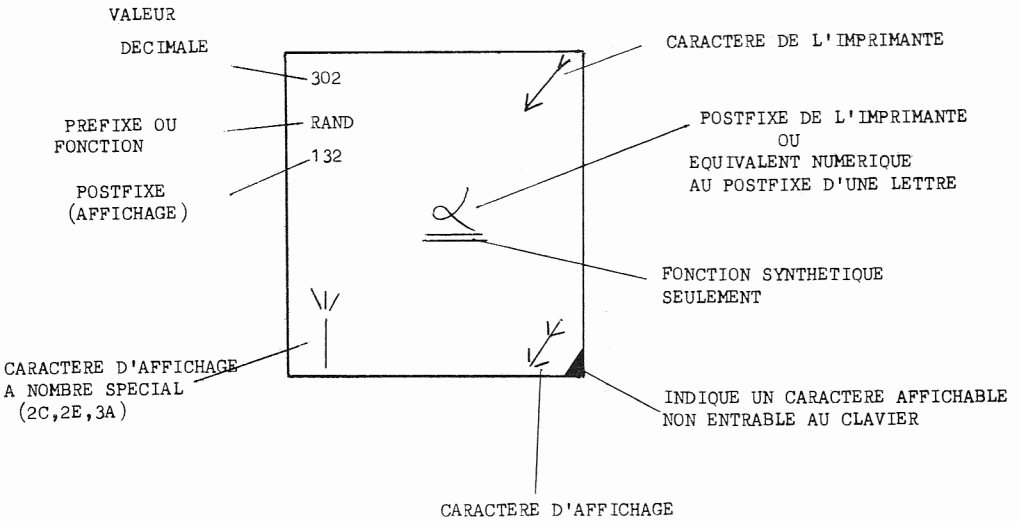


FIGURE 2-4. EXEMPLE DE CASE DE LA TABLE D'OCTETS

D'une façon plus préçise, nous devons considérer le nybble principal 'A' comme un préfixe et les trois autres nybbles comme des postfixes identifiant une fonction de périphérique. Les codes 'XROM' qui sont affichés quand un périphérique est absent dérivent directement de la valeur des octets pour les fonctions des périphériques. Les trois nybbles qui suivent un 'A' sont séparés en groupes de six bits. Les 2 numéros affichés avec 'XROM' sont les équivalents décimaux des deux groupes. Par exemple :

'PRX' : hex A754 : 1010/0111 01/01 0100 : XROM 29,20
29 20

'WSTS' : hex A78A : 1010/0111 10/00 1010 : XROM 30,10
30 10

Il peut y avoir confusion à propos des fonctions à un seul octet des lignes 0,2, et 3. Pour pouvoir permettre le stockage et le rappel directs depuis 100 registres, les 'STO' et les 'RCL' doivent être à 2 octets; ou sinon une trop grande partie de la table hexa serait utilisée pour des fonctions explicites comme STO 99 et RCL 50. D'un autre côté, trop de fonctions à deux octets dans un programme utilisent beaucoup de place. Comme compromis, la HP 41Cpermet l'accés aux registres Roo à R15 avec un seul octet en réservant des codes à un octet pour STO et RCL 00-15. Pour R16-R99, une combinaison de deux octets préfixe/postfixe est nécessaire .De la même façon, il pourait y avoir accés direct aux registres R100-R255, en assignant à chaque octet de la table différents nombres de postfixes numériques, mais ça n'aurait plus laissé de place pour l'adressage indirect souple existant.
Le même choix entre souplesse des fonctions et conservation des programmes apparaît dans la possibilité de labels 'courts', à un octet, aussi bien qu'à deux octets.Les labels 00 à 14 sont explicitement codés dans la ligne 0 de la table, alors que les labels 15 à 99 demandent chacun deux octets: le préfixe CF et un postfixe de la demi-table supérieure. Les labels à deux octets peuvent aussi utiliser les postfixes 66-6F et 7B-7F, générant les 'labels alpha-numériques locaux', 'LBL A' à'LBL J' et 'LBL a' à 'LBL e'.
Tout commence à devenir un peu plus mystérieux alors que nous continuons à descendre dans la table, entrant dans la région commençant avec la ligne B. RegardeZ, par exemple, à la ligne E. Pourquoi y a-t il 16 différents préfixes 'XEQ' ? Dans cette région de la table, la fonction commencant avec chaque octet comprend 2 octets ou plus, ce qui fait que la table, comme elle est dressée, devient impropre à montrer chaque détail. Considérons d'abord les 'GTO' à deux octets dans la ligne B par comparaison avec les 'GTO' à trois octets de la ligne D. De nouveau, nous avons le compromis entre la souplesse et la place en mémoire, comme pour les STO et les RCL à un et deux octets.
Quand 'GTO 05', par exemple, est entré au clavier dans un programme, ces deux octets sont stockés en mémoire :

1011 0110/ 0000 0000

Le premier octet est 'B6', ce qui correspond dans la table à 'GTO 05'. Alors, à quoi sert le second octet ? Nous rencontrons là un des traits,invisible mais merveilleux,de la HP 41C :le branchement rapide. La première fois qu'un programme rencontre 'GTO 05', le processeur doit chercher dans tout le programme jusqu'à ce qu'il trouve le 'LBL 05', un procédé (relativement) lent. Une fois trouvée sa destination, il enregistre la distance entre le 'GTO 05' et le 'LBL 05' dans le second octet du 'GTO 05', de façon que lors de toutes les exécutions ultérieures, il puisse directement sauter au 'LBL 05'.Nous verrons en détails comment cette information est stocké dans le prochain chapître. Il est suffisant de dire que l'octet réservé pour l'enregistrement du saut permet de sauter jusqu'à 16 registres, en avant ou en arrière. Les GTO à trois octets de la ligne D ont un supplément de 5 bits pour l'enregistrement de la distance, permettant des sauts jusqu'à 512 registres. Le choix du programmeur entre les GTO à deux et à trois octets est donc un choix entre la rapidité du programme et sa longueur: si le saut est plus petit que 16 registres, le GTO à deux octets et son LBL correspondant à un octet épargnent deux octets sans perte de rapidité. Pour des sauts plus importants, cependant, la forme courte prendra un temps d'exécution plus long.
Les XEQ de la ligne E de la table sont structurés de la même manière que les GTO à trois octets de la ligne D. Ils sont exécutés aussi de la même façon, avec en plus le fait que l'adresse de la ligne du XEQ est enregistrée tout comme la longueur jusqu'au LBL.
l'adresse de retour est stockée dans des registres spéciaux appelés 'pile de retour'. Voir chapître 4F.
Courage, nous en avons pratiquement fini avec la table !! Ligne F: ces octets identifient des lignes programme alphanumériques. Quand le processeur rencontre un F (1111 en binaire) comme premier nybble d'un octet, il est averti que la ligne programme contient du texte. le nombre de caractères, de 1 à 15, est indiqué par le second nybble de l'octet de texte. En mode programme, un octet 'Fn' aboutit à l'affichage avec le symbole texte "⊤" suivi par n caractères déduits des n octets suivants dans la mémoire programme. Lors de l'exécution d'un programme, ou lors d'un SST, le processeur copie simplement les n octets suivants dans le registre alpha, puis reprend l'exécution à l'octet suivant le dernier des n.
Exemples :

⊤A: F1 41
⊤BIG: F3 42 49 47
⊤THRILL: F6 54 48 52 49 4C 4C

Les lignes de texte demandent l'explication de la dernière entrée dans chaque case de la table. Le caractère dans le coin inférieur droit montre le caractère alphanumérique affiché si l'octet correspondant est dans le registre alpha, dans une ligne programme, ou dans une ligne alphanumérique de programme. Le mécanisme d'affichage est susceptible de générer 83 caractères différents. Parmi eux, 59 constituent les caractères normaux entrables directement au clavier. Deux autres, le symbole texte "┬" et le symbole append "├" peuvent être 'entrés', mais pas dans n'importe quelle circonstance.
Dix-neuf caractères, identifiés par le triangle noir dans le coin inférieur droit de la case, ne peuvent pas être entrés directement. Il apparaissent à l'affichage suite à la fonction 'BLDSPEC'. Le canard ">" est vu menant inlassablement sa ronde pendant la marche d'un programme, mais son inverse "<" demande d'extraordinaires efforts pour surgir de son nid puisque le code 2C s'affiche normalement comme le caractère virgule. Les octets 2C, 2E et 3A sont montrés avec deux caractères. Le caractère normal est montré à la droite de l'endroit où ces 3 caractères, ",", ".", et ":", sont affichés en utilisant les LCD point/virgule spéciaux entre les pavés normaux de segments. Le caractère de gauche peut être contrôlé par l'utilisateur dans certains affichages spéciaux, comme décrit dans le chapître 7C. Enfin, si un octet n'est pas assigné à un des 82 caractères mentionnés ci-dessus, il est, par défaut, le caractère plein "■".A L'exeption de l'octet 3A, les caractères pleins ne sont pas apparents dans la table.
Deux remarques: d'abord, l'opération 'append' est codée par l'octet 7F. Si l'octet apparait seul, c'est 'CLD', mais s'il est le second octet d'une chaine de caractères il provoque la concaténation des autres octets au contenu du registre alphanumérique. Le second nybble 'n' de l'octet 'Fn' de texte aura une valeur incrémentée de 1 par rapport au nombre de caractères réellement concaténés. "LEG" est 'F3 4C 45 47' mais "├LEG" est 'F4 7F 4C 45 47'. Deuxièmement, l'octet 'F0', ou 'texte 0', n'apparait normalement pas dans les programmes utilisateurs sauf pour les suffixes 'IND T', mais joue un rôle dans les registres d'assignation de touches(chapître 2E)
Nous en arrivons au 'END'. Les octets C0-CD, 'GLOBAL', jouent un rôle double; ils identifient les lignes 'END' et les labels alphanumériques globaux. Si le troisième octet d'une ligne commençant par 'Cn' (0<n<E) est un octet de texte 'Fn', alors la ligne est un label alphanumérique global. Autrement, c'est un 'END' à trois octets. Pour tous ces types de lignes, le second, le troisième et le quatrième nybbles donnent la distance entre cette ligne et le 'end' suivant ou le label alpha précédant en mémoire. La distance est codée comme pour un GTO à trois octets (chapître 2C). Ainsi, toutes les lignes globales sont liées ensemble. Un GTO-alphanumérique ou un XEQ-alphanumérique commence à chercher la chaîne globale depuis la fin de la mémoire programme, le 'END' permanent, en reculant jusqu'à la première ligne globale en mémoire, qui est définie par ses deux premiers octets 'C0 00'. 'CAT 1' montre les labels et les END en ordre depuis la première ligne globale.
Dans les lignes END, le premier octet est utilisé pour donner une information sur le programme courant - si il a été compacté et si c'est le dernier en mémoire, i.e., si le END est le permanent .END. Dans le troisième octet, un premier nybble '0' indique un END normal; un '2' indique le .END. permanent.Pour le second nybble, '9' veut dire que le fichier programme est compacté; 'D' indique qu'il a besoin d'un packing.
Les labels alphanumériques globaux sont les lignes programme de la HP 41C les plus compliquées. Le troisième octet est un 'Fn', ou 'n' est le nombre héxa plus un de caractères dans le label alpha. Le quatrième octet de la ligne, l'octet supplémentaire réservé par le Fn, contient un code pour l'assignation à une touche du label. '00' indique qu'il n'y a pas d'assignation. Les octets restant (n-1) dans la ligne contiennent le nom du label. Exemple: LBL "ABC" : Cl mn F4 ab 41 42 43, où lmn est la distance au prochain label, et 'ab' identifie l'assignation.
Il y a encore quelques trucs à expliquer dans la table. Les octets 1D et 1E, préfixes étrangers dans cette région de fonctions à un seul octet, sont les préfixes pour GTO (alpha) et XEQ (alpha),respectivement. Quand un de ces octets commence une ligne, il est suivi par un octet 'Fn' réservant n octets pour le nom du label appelé. Par exemple:

GTO "BLAZES" : 1D F6 42 4C 41 5A 55 53
XEQ "SPY" : 1E F3 53 50 59

Maintenant, voici l'homme invisible, l'octet 00, la fonction nulle. Ces octets sont normalement invisibles pour le programmeur, mais ils sont utilisés par la HP 41C pour faciliter l'édition et pour réserver la place de futurs codes. Comme exemple d'utilisation, un nul est

automatiquement inséré devant le premier digit d'un nombre enregistré. Le nul sert à isoler cette ligne de la précédente au cas ou celle-ci serait un nombre aussi; le nul est équivalent d'un invisible 'ENTER' dans ce contexte. Pendant l'éxecution d'un PACKING, un tel nul est effacé si nécessaire, de même que les autres nuls superflus du programme.
Finalement, les octets 1F, AF, et BO sont des codes de fonction de rechange; c'est à dire qu'ils n'ont ni préfixes ni utilisation seule, il ne sont là que comme postfixes.

2C. REGISTRE S.V.P.

Nous avons vu que la mémoire utilisateur de la HP 41C, et sa réplique sur carte magnétique ou en code-barres, peut être vue comme une longue chaîne de bits binaires, comme une cartouchière ou il manquerait des balles. Pour donner un sens à la chaîne, alors que le processeur examine, il groupe les bits en nybbles et en octets pour les décoder, et en registres de 7 octets pour le stockage et le rappel des données. Mais, pour que le processeur sache quels bits grouper, il doit y avoir un moyen d'identifier chaque section de mèmoire. Ce moyen doit à la fois permettre l'adressage 'absolu' pour permettre au processeur de retrouver les informations stockés en des endroits fixes comme la pile de registres, et aussi l'adressage relatif pour s'assurer que les sauts des programmes provoqués par les GTO et les XEQ seront inchangés en dépit de l'opération SIZE.
Puisque le plus petit élément de stockage de programme est l'octet, et puisque les registres de données sont un nombre entier d'octets, il est suffisant d'avoir des adresses individuelles jusqu'au niveau de l'octet, plutôt qu'au niveau du nybble ou même du bit. Il devrait aussi y avoir une adresse pour chaque registre, pour faciliter la manipulation des données, et pour accelérer le processus de recherche d'adresse - ce que nous voulons ressemble à une adresse dans une rue, avec le registre et l'octet respectivement analogues à la rue et à la maison. Ces simples idées nous conduisent tout droit au système d'adressage de la HP 41C.
Chaque octet en mémoire utilisateur a une adresse de la forme :

(n a b c).

'abc' est un nombre héxa à trois nybbles désignant un registre particulier. Nous devons faire une distinction entre l'adressage absolu d'un registre de données et son numéro de registre de données, qui est son adresse relative. L'emplacement mémoire du nombre stocké en registre Roo, par exemple, n'est pas fixe. Quand une nouvelle SIZE est exécutée, le contenu de la mémoire bouge pour changer la répartition entre les données et les programmes. Pour aider l'utilisateur, le contenu original de Roo sera accessible par RCL 00, etc., même si l'emplacement, ou adresse absolue, du contenu a changé (voir chapître 4G).
Le digit restant de l'adresse à 4 digits, 'n', est le numero d'octet . Chaque registre a 7 octets, donc n peut prendre les valeurs de o à 6. Nous étendons maintenant notre conception du processeur en incluant un pointeur d'adresse, qui contient toujours l'adresse courante en 4 digits de l'octet de programme en cours de traitement. La convention utilisée par la HP 41C est que 'en avant' dans la memoire programme dans la direction des numéros de lignes croissants correspond à décrémenter l'adresse (voir figure 2-5). Lors de l'éxecution de SST, le pointeur décrémente le numéro d'octet du nombre d'octets de la ligne programme, avec l'octet 6 comme premier octet du registre, et l'octet 0 pour le dernier. Quand la frontière d'un registre est franchie, 'n' est initialisé à 6 et 'abc' est décrémenté de 1.Les registres de données sont numérotés dans le sens opposé, de sorte que R1o, par exemple, est le registre (absolu) '123', et R11 est '124', R12 est '125', etc. Si nous pouvions placer le pointeur dans les registres de données et exécuter un pas en mode programme, nous pourrions voir 7 octets pour chaque registre, commençant par un octet consistant en un nybble du signe de la mantisse et un digit de la mantisse, et finissant par les deux digits de l'exposant.
Il a été établi que le branchement causé par GTO et XEQ enregistre la longueur des sauts, plutôt que l'adresse absolue des labels, dans les lignes programme. Ce qui fait qu'un changement du contenu des registres comme celui causé par 'SIZE' ou par l'insertion d'un nouveau fichier programme à une adresse supérieure ne nécessite pas de changer les sauts stockés.
La distance d'un saut est exprimée par un nombre stocké en registres de 7 octets complets, plus des octets supplémentaires. La distance est mesurée depuis l'octet contenant la première partie du code de la distance du saut, jusqu'à l'octet précédant le label en question. Pour clarifier ce codage, voyons quelques exemples. D'abord, regardons cette routine :

```
01 GTO 05          B6 22
02 "ABCDEFGHIJKLMNO"FF 41 42 ... 4E 4F
03 LBL 05          06
04 "ABCDEFGHIJ"    FA 41 42...  49 4A
05 GTO 05          B6 82
```

Les nombres à la droite des lignes programme sont les codes des octets de programme. A la prelière exécution de la routine, les codes des lignes 01 et 05 seraient 'B6 00' . Le 'B6' identifie 'GTO 05'; le '00' indique que la distance du saut est inconnue . Exécution suivante, les codes sont comme montré ci-dessus, avec chaque '00' remplacé par le code de la distance.
En écrivant les octets en binaire, nous pouvons voir commant les octets sont interprétés:

	Direction	Octets	Registres
(ligne 01) 22	0	010	0010
(ligne 05) 82	1	000	0010

Si le premier bit est 0, le saut est en avant(à une adresse inférieure); si c'est un 1, le saut est en arrière. Pour les GTO à 2 octets, l'information de saut est entiérement dans le second octet de chaque 'GTO 05', donc nous comptons la distance de saut à partir de là. Depuis le '22' de la ligne 01, nous comptons 2 registres plus deux octets : 16 octets, commençant avec le FF de la ligne 2. L'instruction suivante est le 'LBL 05'. Pour le GTO de la ligne 05, nous comptons à reculons 2 registres plus 0 octets : 14 octets, commençant avec le 'B6'. De nouveau le pointeur va au "O". La longueur maximale de cette sorte de sauts est de F registres plus 7 octets : 112 octets ou 16 registres. Les GTO et les XEQ à trois octets sont similaires aux GTO à deux octets, mais avec une différence dans le rangement de l'information de saut. Si nous substituons les formes longues dans la routine 2C-1:

```
01 GTO 45                          D8 02 2D
02 "ABCDEFGHIJKLMNO"               FF 41 42 .... 4E 4F
03 LBL 45                          CF 2D
04 "ABCDEFGHIJ"                    FA 41 42 ..... 49 4A
05 XEQ 45                          E0 02 AD
```

De nouveau, nous cassons les codes en bits, et nous groupons:

	Type	Octets	Registres	Direction	Label
(ligne 01) D8 02 2D	1101	100	000000010	0	0101101
(ligne 02) E0 02 AD	1110	000	000000010	1	0101101

Seulement 7 bits sont requis pour les postfixes des labels jusqu'à 99; trois autres bits sont nécessaires pour le numéro d'octets 0-6. Donc avec 4 bits pour le type de ligne (1101 pour GTO, 1110 pour XEQ), et un bit pour la direction, il reste 9 bits pour le nombre de registres. De ce fait, le saut peut atteindre 2^9 : 512 registres, ce qui est plus grand que la mémoire.
Le premier octet de la ligne GTO ou XEQ commence le codage du saut, donc nous décomptons le saut depuis ce premier octet. Depuis 'GTO 45' de la ligne 01, nous comptons 4 octets plus 2 registres soit 18 octets depuis l'octet D8, qui place le pointeur sur le "O" comme avant. La ligne '05 XEQ 45' recule le pointeur depuis l'octet E0, 0 octets plus 2 registres : 14 octets. Exactement comme il est préférable pour un registre de données d'avoir une adresse relative et non absolue comme numéro pour faciliter les changements de contenu mémoire, il n'y a pas de numéro absolu de ligne programme associé à une place particulière en mémoire. Le numéro de ligne est une quantité qui est recalculée chaque fois que nécesaire, i.e., pour chaque pas de programme affiché en mode PRGM ou pour un SST ou un BST. Vous avez peut-être déjà remarqué que la première fois que vous mettez en mode PRGM aprés qu'un programme ait tourné, ou pressez BST à la fin d'un long programme,il y a une pause remarquable avant que la ligne courante ne soit affichée. Ce temps mort est utilisé par le processeur pour calculer le numéro de ligne ce qu'il ne peut faire qu'en se repérant et en allant au début du programme, et redescendant tout le programme en incrémentant le compteur de lignes (stocké dans un registre spécial) pour chaque ligne programme compléte. Il serait superflu et retardateur pour le processeur de garder trace des numéros de lignes pendant le déroulement d'un programme, donc il doit faire le calcul complet des numéros de ligne quand l'utilisateur met ensuite le mode PRGM. Subséquemment, les SST sont plus rapides, mais un BST est lent parce que le processeur n'a aucun moyen de savoir si l'octet précédent est un octet seul ou une fonction multi- octets. Il doit de nouveau retourner au début du fichier et recompter ligne par ligne jusqu'à ce qu'il atteigne le nombre immédiatement inférieur à celui de la ligne de départ.

2D. REPARTITION DE LA MEMOIRE

La figure 2-5 est une représentation figurative de la mémoire utilisateur de la HP 41C, où nous visualisons tous les registres de mémoire les uns au dessus des autres. Le dessin montre la configuration de base plus 4 modules de mémoire possibles. Le haut du dessin est 'le haut de la mémoire', c'est à dire le registre de données disponible de plus grand numéro. Descendre sur le dessin correspond à décroître pour les numéros de registres et pour les adresses absolues, ou accroitre pour les lignes de programme. La direction horizontale représente le numéro d'octet, avec le premier octet'6', de chaque registre à la gauche, et le dernier, '0', à la droite. L'exécution pas à pas fait bouger le pointeur d'adresses vers la droite à travers les octets d'un registre, puis vers le '6' du registre du dessous.
Le premier registre de données, Roo, et la première ligne de programme du premier programme utilisateur sont immédiatement adjacents en mémoire, sans frontière physique entre eux. L'adresse absolue courante de Roo est stockée par la HP 41C, pour que le processeur sache toujours quels registres sont affectés aux données (au dessus de Roo) et lesquels sont réservés aux programmes (en dessous de Roo). Quand un module mémoire est ajouté, ses 64 registres sont ajoutés au sommet de la mémoire, pour que la SIZE automatiquement augmente de 64 (hex 40). Quand 'SIZE abc' est éxécuté, le contenu des registres de données et de programme est déplacé vers le haut ou vers le bas jusqu'à ce que le contenu original de Roo soit dans le registre mno (mno est un nombre héxa à trois digits), à 'abc' registres du sommet.
Le registre 'mno-1' est le premier registre de la mémoire programme. Si nous commençons sans programme dans la mémoire, les trois derniers octets du registre 'mno-1' contiennent automatiquement le .END. permanent. Ce .END. est toujours présent en mémoire, nécessairement parce qu'il est le premier maillon de la chaine d'adresses globales qui regroupe les labels globaux et les END de la mémoire. Quand nous commençons à entrer un programme, les quatre premiers octets remplacent les octets nuls restant dans le registre 'mno-1'. Si de nouveaux octets sont ajoutés, le .END. est automatiquement déplacé vers les trois derniers octets du registre suivant, donnant 7 octets supplémentaires pour le programme. Ce procédé est répété jusqu'à ce que le programme soit complet, ou jusqu'à ce que tous les registres libres soient occupés. Si nous entrons un END quelque part, nous érigeons une barrière dans la mémoire, servant à diviser les lignes programme précedemment entrées en un programme dans le programme. La ligne END elle-même est la barrière, quand elle est rencontrée durant un SST, ou pendant la recherche d'un label local pendant un programme, elle fait retourner le pointeur d'adresses au END supérieur de la chaine de labels, ou à l'octet '0mno', si le programme courant est le premier en mémoire. Si nous avons rentrés un total de 'def' registres de programme (y compris le END), l'adresse du registre contenant le END sera (pqr = mno-def). Souvenons nous qu'une telle arithmétique sur les adresses de registres est faite en héxadécimal. Dans la HP 41C, 'pqr' ne peut jamais être inférieur à héxa 0C0 (décimal 192). Le choix de 0C0 pour le bas de la mémoire programmable donne pour adresses de la configuration de base : 0C0 à 0FF. Si le premier digit du numéro d'un registre est '1', le registre est dans un module mémoire: module 1: regitres 100 à 13F, module 2: 140 à 17F, module 3: 180 à 1BF, module 4: 1C0 à 1FF.
A n'importe quel moment, il y a (pqr-0C0) registres disponibles pour les programmes, moins ceux utilisés pour les assignations de touches. Les assignations de fonctions utilisateurs sont codées dans un block de registres commençant à 0C0 et montant dans la mémoire (les détails sur le codage sont donnés au chapitre 2E). Si 'jkl' registres sont utilisés pour les assignations de touches, il y a (ghi = mno-def-jkl-0C0) registres encore disponibles pour de nouvelles lignes de programme ou pour des assignations.
En tout , nous avons:

$$(N\ 1)*40 = abc + def + ghi + jkl$$

registres dans le système, ou 'N' est le nombre de modules de mémoire insérés.
Sous le registre 0C0, il y a un espace dans le dessin, représentant un trou dans le shéma d'adresses , puisque il n'existe pas de registre qui corresponde à ces adresses. Entre les adresses 000 et 00F, cependant, il y a un block extrêmement interessant de 16 registres. Nous appelons ces registres les registres d'état, puisque leur contenu est enregistré par le lecteur de cartes sur la piste 1 des cartes par la fonction 'WSTS' (write status). L'accés à ces registres est la base de la programmation synthétique; leur étude mérite un chapitre entier, le chapitre 4.

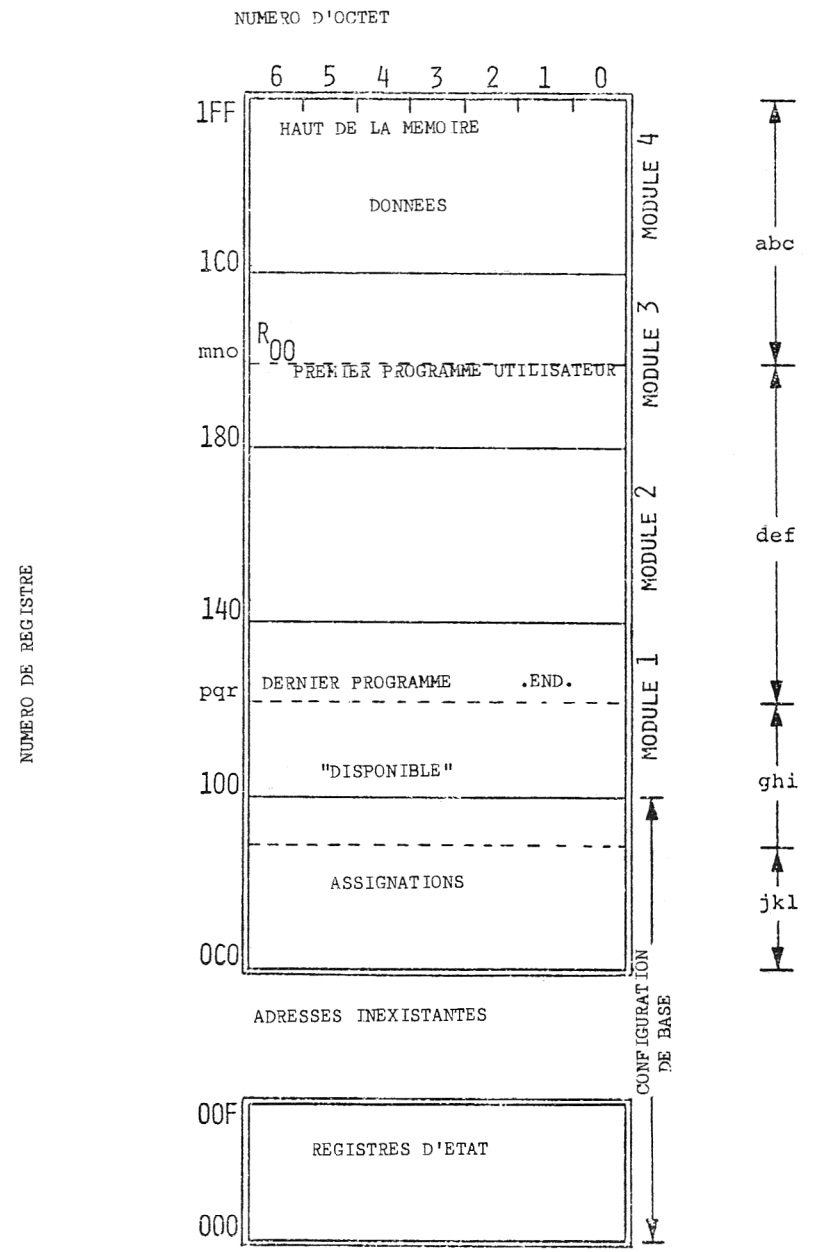


FIGURE 2-5. REPARTITION MEMOIRE DE LA HP 41C

2E. LES REGISTRES D'ASSIGNATION DE TOUCHES

Les registres d'assignation de touches, commençent au registre OCO jusqu'au registre contenant le .END. non compris. Les registres contiennent les codes qui disent au processeur quelles fonctions sont assignées à quelles touches (rappelons que l'assignation des labels globaux utilisateurs sont enregistrées dans le label lui-même). Considérons la séquence de touches suivante:

ASN ALPHA "LN" ALPHA 8 (Assigne 'LN' à la touche 8)

Si nous étions capable par quelque tour de passe-passe de placer le pointeur d'adresses dans le registre OCO en mode PRGM et de lister son contenu, nous verrions ceci (avec des numéros de ligne arbitraires, et les codes d'octets listés sur la droite):

```
01 ""          F0
02 LBL 03      04
03 LN          50
04 RCL 05      25
```

(La ligne 01, l'octet F0 ou 'TEXT 0', apparait à l'affichage comme '01 ⊤ '.) Quatre octets ne sont pas suffisants pour remplir un registre ; il y a trois invisibles nuls entre les lignes 01 et 02. Les nuls disparaissent si nous faisons une deuxième assignation:

ASN ALPHA "LOG" ALPHA SHIFT 8 (Assigne 'log' à la touche shiftée 8)

Maintenant le registre OCO contient:

```
01 ""          F0
02 LBL 03      04
03 LOG         56
04 RCL 13      2D
05 LBL 03      04
06 LN          50
07 RCL 05      25
```

Comme programme, ces lignes ne veulent rien dire, bien que nous reconnaissions le 'LOG' et le 'LN' que nous avons assignés. Plutôt, les octets sont un code spécial. Le premier octet, 'F0', identifie le registre comme un registre d'assignation et le sépare des registres d'assignation adjacents. Les trois octets suivants sont un code pour l'assignation de 'LOG', la deuxième assignation faite. Les trois derniers octets codent l'assignation de 'LN'. Dans chaque groupe de trois octets, les deux premiers octets identifient la fonction assignée et le troisième identifie la touche. Pour les assignations des fonctions de la HP 41C, seulement un octet est requis pour identifier la fonction, de sorte que le '04' (LBL 03) est mis là comme remplissage. Si une fonction de périphérique est assignée, tous les octets de fonction sont requis pour représenter la fonction. Par exemple, si nous avions assignés 'PRP' et 'WSTS' au lieu de 'LN' et 'LOG', le registre OCO aurait contenu:

```
01 ""          F0
02 WSTS        A7 8A
03 RCL 13      2D
04 PRP         A7 4D
05 RCL 05      25
```

Le code pour une touche désignée est comme suit: supposons que l'on assigne la touche 'MN', i.e., la touche de la colonne N et de la ligne M du clavier. Alors l'octet représentant cette touche sera héxa 'XY', avec (X: N-1), et (Y: M). La touche '8' assignée ci-dessus est la touche 53, donc (X :2) et (Y :5), produisant le 'RCL 05' dans le registre d'assignation. Autres exemples: la touche 'COS', touche 24, est représentée par l'octet 32, ou 'STO 02'; 'R/S', touche 84, est codé par l'octet 38 : 'STO 08'.

Le code pour la touche shiftée '-MN' est obtenu à partir de (X : N-1) , (Y : M+8). Ainsi l'assignation de la touche '8' shiftée, touche 53, est codée par l'octet 2D : 'RCL 13'. Pour les touches shiftées de la ligne 8, ou N 8, nous reportons le '1' résultant de (8+8=10) dans X. Par exemple, la touche 'VIEW", touche -84, est codée par l'octet 40: '+'.

Les touches ordinairement numérotées 42,43 et44, i.e., 'CHS', 'EEX', et ⌫, et leur contreparties shiftées, sont physiquement dans les colonnes 3,4, et 5 respectivement, et doivent être ainsi numérotées pour leur assignation. Tout se passe comme si 'ENTER' couvrait une touche imaginaire 42. La figure 2-6 montre les codes d'assignation sur un dessin du clavier pour plus de facilité dans les références.

Le nombre sur chaque touche est l'octet d'assignation de touche de cette touche. Les codes pour les touches shiftées sont écrits au dessus de chaque touche.

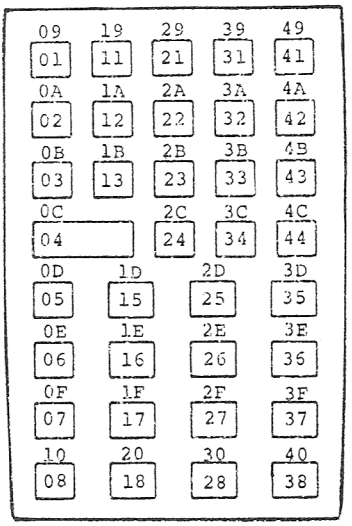


Figure 2-6 OCTETS D'ASSIGNATION DE TOUCHES

Si la fonction assignée est un préfixe, comme 'STO', 'ISG', 'GTO', etc., le listing du registre d'assignation montrera l'octet de la fonction et l'octet de désignation de la touche regroupé dans une seule ligne programme. L'octet de désignation de la touche agit comme un postfixe pour le préfixe assigné.
Quand une fonction non-programmable de la HP 41C est assignée, l'octet de fonction est trouvé dans la ligne 0 de la table d'octet, de sorte que la ligne programme correspondante est une des formes courtes de labels ou un nul. La table 2-2 montre la correspondance.
Bien que la plupart des enregistrements de la table 2-2 correspondent à des assignations normales, les octets 01, 05, 0B, 0C, 0D et 0E représentent des 'fonctions' qui ne sont pas normalement assignables. En utilisant le programme d'assignation du chapitre 5, cependant, nous pouvons placer ces octets dans les registres d'assignation avec des résultats amusants. Les fonctions '∂c' et '2--' sont ainsi nommées parce que presser sur la touche à laquelle l'une d'elle est assignée fait que l'affichage montre ce nom. L'exécution de '∂c', par moments, ne produit rien; d'autres fois, un GTO.. est éxécuté. '2--', dans l'enregistrement d'un nombre à deux digits, provoque le verrouillage temporaire de la HP 41C. L'utilisation la plus pratique des octets 05, 0B et 0E nous permet de réassigner le 'R/S', '←', et le SHIFT, respectivement. Presser sur la touche de correction réassignée efface toujours la ligne courante de programme, que la HP 41C soit en mode PRGM ou non. Finalement , l'octet 0C réassigne les 'fonctions à bascule', 'ALPHA', 'PRGM', et 'USER'. Le choix de la fonction dépend de la touche désignée (!): si la touche est dans la ligne 1 ou la ligne 5, 'ALPHA' est assigné; les touches des lignes 2 et 6 seront assignées à 'PRGM'; 'USER' reste pour les assignations des touches des lignes restantes 3,4,7 et 8.

TABLE 2-2

Fonction	Octet	Ligne programme
CAT	00	nul
℘c	01	LBL 00
DEL	02	LBL 01
COPY	03	LBL 02
CLP	04	LBL 03
R/S	05	LBL 04
SIZE	06	LBL 05
BST	07	LBL 06
SST	08	LBL 07
ON	09	LBL 08
PACK	0A	LBL 09
⏎	0B	LBL 10
ALPHA/PRGM/USER	0C	LBL 11
2--	0D	LBL 12
SHIFT	0E	LBL 13
ASN	0F	LBL 14

ASSIGNATION DE FONCTIONS NON PROGRAMMABLES DE LA HP 41C

CHAPITRE 3

EDITION EXOTIQUE AVEC LE SAUTEUR D'OCTETS

3A. EDITION NORMALE

Tout programmeur de HP 41C connait les règles simples qui gouvernent l'édition normale : (1) en mode PRGM, une ligne programme entrée est insérée immédiatement aprés la ligne montrée à l'affichage. Toutes les lignes suivantes ont leur numéro incrémenté de 1. (2) Quand la touche de correction est pressée, la ligne programme affichée est effacée, et les lignes suivantes ont leur numéro décrémenté de 1. L'affichage montrera la ligne précédant la ligne effacée. (3) l'éxécution de 'DEL lmn' fait que 'lmn' lignes de programme sont effacées, y compris la ligne affichée. (4) PACK fait en quelque sorte le ménage, en effaçant les nuls invisibles pour maximiser l'espace mémoire disponible.
Ces opérations produisent une capacité simple et rapide d'édition pour la HP 41C. Mais pour nos visées, les descriptions de (1) à (4) ci-dessus sont inadéquates; nous avons besoin de savoir exactement ce qui se passe dans la mémoire au niveau de l'octet, non au niveau de la ligne de programme. Donc réécrivons les régles comme suit:

(1) La ligne programme montrée en mode PRGM est la ligne programme commençant avec le premier octet non nul suivant l'octet où le pointeur d'adresses est situé. Quand une nouvelle ligne de programme est entrée, les octets constituant la nouvelle ligne sont placés immédiatement aprés le dernier octet de la ligne initialement affichée, en remplaçant les octets nuls. Si aucun octet nul n'est disponible, i.e., si l'octet à la place de l'insertion n'est pas un nul, ('00'), le processeur insère automatiquement 7 nuls ou un multiple de 7 si nécessaire, avant d'entrer les nouveaux octets de programme. La nouvelle ligne recouvre alors autant de nouveaux nuls que nécessaire, laissant les autres (invisiblement) dans le programme. L'insertion d'exactement 7 nuls fait que le processus pour bouger les lignes suivantes en mémoire est simple -- chaque registre contenant des programmes utilisateurs est seulement copié dans le registre inférieur, depuis le .END. et ainsi de suite jusqu'au registre où a lieu l'insertion.
Un 'RTN' manuel, 'GTO.000', ou 'GTO.001' met le pointeur d'adresses sur le dernier octet du précédent programme. Les deux premières opérations, en ayant '00' comme numéro de ligne, font apparaitre '00 REG lmn' à l'affichage au lieu d'une ligne de programme. Quand le numéro de ligne est '00', les octets de programme sont entrés immédiatement aprés l'octet courant du pointeur plutôt qu'aprés la ligne programme courante.
(2) Quand la touche de correction est pressée en mode PRGM, les octets de la ligne affichée sont remplacés par un même nombre de nuls. Le pointeur programme recule d'une ligne.
(3) L'opération 'DEL lmn' remplace tous les octets des 'lmn' lignes programme suivantes par des nuls. Vous pouvez observer qu'un SST suivant un effacement de nombreuses lignes demande une pause appréciable, qui est en fait le temps requis par le processeur pour parcourir tous les octets nuls résultant de l'effacement jusqu'à ce qu'il trouve un octet non nul pour l'affichage.
(4) Une séquence d'édition peut produire un nombre appréciable de nuls superflus dans un fichier programme. L'opération PACK enlève tous les nuls inutiles en faisant monter en mémoire les octets de programme non nuls. Les nuls trouvés à l'intérieur de fonctions multi-octets ne sont pas effacés.

Quand les octets de programme bougent dans la mémoire, soit par édition (insertion ou effacement d'octets) ou par packing, des codes variés de distance de saut peuvent devenir caducs. C'est pourquoi, aprés chacune de ces opérations, les octets de distances de sauts de tous les GTO et XEQ locaux du fichier qui vient d'être édité sont remis à 0, pour qu'ils aient à être recalculés quand le programme tournera de nouveau. Bien plus, les adresses relatives dans la chaine des labels globaux doivent être réactualisées. Enfin, le END terminant le fichier édité est recodé pour indiquer que le fichier doit être compacté.
Un exemple d'édition de programme clarifiera sans doute les règles ci-dessus remaniées. Commençant d'un 'MEMORY LOST', nous entrons ce simple programme:

```
01  LBL  00
    .END.
```

Si nous écrivons tous les octets du fichier, le programme ressemble à ceci:

Adresses	NUméro de ligne	Ligne	Code d'octet
60EE	01	LBL 00	01
50EE			00
40EE			00
30EE			00
20EE		.END.	C0
10EE			00
00EE			29

Les adresses viennent du fait que la HP 41C 'se réveille' avec 47 (hex 2F) registres d'espace programmable, commençant au registre 0C0: (0C0+02F-1=0EE). Donc le registre 0EE est le plus haut registre de programme. Dans la ligne du .END., les nybbles '000' indiquent qu'il s'agit du label le plus haut en mémoire; le '29' indique un fichier compacté avec le .END. permanent. Supposons maintenant que nous insérions trois lignes '+' aprés la ligne 01:

60EE	01	LBL 00	01
50EE	02	+	40
40EE	03	+	40
30EE	04	+	40
20EE		.END.	C0
10EE			00
00EE			29

Les nuls ont étés remplacés par des octets '40'. Effaçons maitenant la ligne 03:

60EE	01	LBL 00	01
50EE	02	+	40
40EE			00
30EE	03	+	40
20EE		.END.	C0
10EE			00
00EE			2D

Le '40' de l'adresse 40EE a été remplacé par un nul; le dernier octet du .END. a changé pour un '2D' pour indiquer un fichier non compacté. Si nous compactons à ce moment, le '40' de 30EE montera à 40EE, mais un autre '00' sera inséré à 30EE pour garder le .END. dans les 3 derniers octets du registre. Si nous insérions une ligne mono-octet aprés la ligne 02, elle se mettrait simplement à la place du nul à 40EE. Mais si nous insérons une ligne à 2 octets, 'STO 65' par exemple, nous obtenons:

60EE	01	LBL 00	01
50EE	02	+	40
40EE	03	STO 65	91
30EE			41
20EE			00
10EE			00
00EE			00
60ED			00
50ED			00
40ED			00
30ED	04	+	40
20ED		.END.	C0
10ED			00
00ED			2D

Puisqu'il y avait seulement un octet nul disponible entre les lignes 02 et 03, 7 autres nuls ont été insérés. Deux d'entre eux ont ensuite été recouverts par les octets de 'STO 65'. Le '40' à 30EE est descendu à 30ED. Enfin, le .END. a été réinséré, à la fin du registre 0ED. aprés un 'PACK', le programme devient:

60EE	01	LBL 00	01
50EE	02	+	40
40EE	03	STO 65	91
30EE			41
20EE	04	+	40
10EE			
00EE			
60ED			
50ED			
40ED			
30ED			
20ED		.END.	C0
10ED			

Les lignes programme utilisateur ont été mises ensemble, mais puisqu'il n'y a pas de place pour le .END. dans le registre 0EE, il reste dans le registre 0ED.

3B. LE SAUTEUR D'OCTET

Maintenant armé d'un savoir suffisant sur les opérations normales de la HP 41C, nous pouvons hardiment nous aventurer sur un territoire nouveau et brûlant. Cela semblerait une répétition de dire que, aussi étranges que puissent paraitre les procédures, il n'y a pas de risques pour la HP 41C. Suivez moi tout au long de cette procédure: (les possesseurs de HP 41CV se référerons à la page 1 .)

1. Insérer un module de mémoire dans la HP 41C.
2. Effacement général. (HP 41C off; tenir la touche de correction; HP 41C on.) Une coupure radicale avec le passé!
3. Exécuter 'SIZE 000'. Ceci place le .END. dans le module.
4. ASN "X<>" +; ASN "Σ+"Σ+. Ceci remplit le registre 0C0 avec deux assignations.
5. HP 41C off; enlever le module mémoire; attendre 60 secondes à peu prés; remettre le module; HP 41C on. Si vous avez un deuxième module disponible, vous pouvez vous épargner d'attendre 60 secondes en enfichant le module vierge à la place de celui qui est enlevé. Maintenant le .END., que nous avions mis dans le module, s'est évaporé. Si vous aviez rallumé le calculateur avant de remettre un module, 'MEMORY LOST' serait apparu. Evidemment, le processeur vérifie le registre où le .END. est sensé exister mais non si les octets du .END. sont réellement présents dans ce registre.
6. Mettre en mode programme; vous verez '00 REG 126' (190 si votre module est double densité). Maintenant pressez SST une fois. Aprés quelques secondes d'attente, vous verrez '01 ⁻'! Le pointeur d'adresses est maintenant dans le registre 0C0, le premier registre d'assignation! Avec le .END. absent, il n'y avait rien pour arréter le pointeur de continuer gaiement à travers la mémoire vide jusqu'à ce qu'il rencontre un octet non nul, qui est dans ce cas le 'F0' de l'assignation de touches que nous avons mis au 4. Avec 5 SST, la séquence suivante doit apparaitre:

```
02 LBL 03
03 Σ+
04 LBL 00
05 LBL 03
06 X<>06
```

(Si vous faites SST encore une fois, le pointeur arrivera dans les registres d'état.) Vous reconnaitrez ce groupe de lignes comme les codes des assignations que nous avons faites au 4.

7. Utilisez BST pour retourner à la ligne 3. Ne soyez pas inquiets si quelques uns de ces SST et BST demandent quelques secondes. Maintenant, pressez la touche de correction deux fois pour effacer les lignes 03 et 02.
8. Maintenant, entrez ALPHA "A" ALPHA, donnant la ligne '02⁻A'. (En fait, n'importe quel caractère seul fonctionnerait aussi bien que "A".)
9. Pressez 'GTO..'; l'affichage 'GTO..' persistera quelques secondes, suivi par un rapide PACKING. SST une fois, et effacer la ligne '01 LBL 01'. Pour la deuxième et la dernière fois dans ce livre, vous avez mené à bien une opération synthétique par le moyen de l'ablation de module. A partir de maintenant, nous serons capables d'atteindre tous nos buts sans avoir recours à cette pratique déplaisante.
10. Pressez et tenez la touche 'Σ+' en mode USER . Vous devez voir l'affichage de 'XROM 05,01'.

Si cela ne se produit pas, vous devez avoir fait une erreur, donc répétez les étapes 1 à 9.
Par édition directe d'un registre d'assignation, vous avez créé une assignation toute nouvelle, appelée le 'sauteur d'octet'.
La meilleure explication pour l'opération du sauteur d'octets est qu'il est, effectivement, une ligne programme normalement exécutée. Pour comprendre ceci, reprenez à la section 2B ce qui se passe quand il y a éxécution automatique d'une ligne de texte: le processeur regarde le second nybble de l'octet de programme courant, i.e., l'octet 'Fn' qui signale l'instruction de texte,copie les n octets suivants dans le registre alpha, et avance le pointeur de n octets. Le sauteur d'octets est l'équivalent manuel de cette opération, à ne pas confondre avec l'éxécution pas à pas d'une ligne de texte. Le nybble 'F' qui commence le processus est 'produit' en pressant la touche USER à laquelle nous avons assigné "A" ('F1 41'). Pour voir pourquoi cette opération est intéressante, entrez ces lignes:

```
01 STO 04          34
02 "ABCDEFG"       F7 41 42 43 44 45 46 47
```

(3B-1)

Avec la ligne encore à l'affichage, éteindre le mode PRGM, mettre en mode USER, et presser le sauteur d'octets ($\Sigma+$). Remettre en mode programme, et vous verrez pas à pas:

```
02 X<Y?            44
03 X>Y?            45
04 X=Y?            46
05 Σ+              47
```

D'où viennent ces lignes programme ? Comme vous pouvez voir en regardant la valeur des octets des 'nouvelles' lignes, elles sont simplement les fonctions mono-octet correspondant aux caractères "D", "E", "F", et "G" de la ligne originale "ABCDEFG". Nous avons commencé avec l'affichage montrant la ligne '02 "ABCDEFG"', i.e., le pointeur d'adresses était positionné sur l'octet '34', ligne 01. Puis nous avons éxécuté le sauteur d'octets, qui a fait que le processeur a cru éxécuter une ligne de texte (ne pas confondre cette ligne imaginaire avec la vraie ligne 02). Il a donc regardé au second nybble de l'octet courant,'34', copié les 4 octets suivants dans le registre alpha, et avancé le pointeur de 4 octets jusqu'à l'octet '43'. Avec le pointeur à cet endroit, l'affichage montrera la prochaine ligne programme, qui est la ligne mono-octet '02 X<Y?' correspondant à l'octet 44. Si vous mettez maintenant le mode PRGM off, puis en mode alpha, vous verrez les quatres carctères "■ABC", qui sont les 4 octets copiés depuis le programme. Le caractère plein est l'octet 'F7'. Puisque le sauteur d'octets est une fonction éxécutée manuellement, il ne change pas le numéro de la ligne programme courante, même si le pointeur change, ce qui fait que '02 X<Y?' a le même numéro de ligne que '02 "ABCDEFG"' d'où le saut a été effectué.
Pendant que le pointeur est à l'intérieur de la ligne texte, 'SST' opère normalement, mais un 'BST' depuis n'importe quel octet ramène le pointeur à la ligne 'STO 04'. Rappelons que 'BST' renvoie le pointeur au début du programme, d'où il compte ligne par ligne, refusant naturellement de sauter dans une ligne multi-octets.
Voyons quel utilité peut avoir le sauteur d'octets. Pour résumer des instructions futures, je vais introduire une nouvelle instruction:'JUMP .lmn', qui veut dire 'saut d'octet à la ligne lmn'. C'est à dire:

'JUMP .lmn' veut dire:

1. GTO.lmn (même si la ligne affichée est déjà lmn)
2. PRGM off
3. Presser la touche du saut d'octets
4. PRGM on

'JUMP' sans numéro de ligne veut dire 'saut d'octet' depuis la ligne courante, on omet l'étape 1.
Maintenant, en utilisant la routine 3B-1, essayez 'JUMP.002'. Aprés l'étape 4 du JUMP, vous verrez affiché '02 X<Y?'. Appuyez une fois sur la touche de correction, puis sur 'SST'.Vous devez maintenant voir '02 "ABC⁻EFG". Le '⁻' est le caractère affiché pour un nul, dans ce cas le nul par lequel l'octet "D" a été remplacé lors de la correction. Vous venez de modifier une ligne de texte programme sans avoir à effacer la ligne entière ! Et ce n'est que le début..
Maintenant changez la ligne 01 de la routine pour 'STO 03'. (Si vous faites des erreurs pendant l'édition, cela peut introduire des nuls invisibles, éliminez les avec PACK. Si, par exemple, un nul précédait la ligne 02, ce que l'affichage ne nous aurait pas dit, 'JUMP .oo2'

n'aurait rien fait puisque le second nybble du nul est '0'.) Puis 'JUMP.002' pour voir la ligne '02 /', l'octet de "C". Entrez '03 LBL 00', puis faites 'GTO .002'. l'affichage montrera '02 "ABC[illegible]EFG"'. Vous avez remplacé le nul suivant le "C" par l'octet '01', avec 'LBL 00', qui s'affiche comme le caractère 'homme', "[illegible]", quand la ligne complète est affichée. La séquence suivante produira les pas de programme écrits à sa droite:

```
JUMP .002                    |  01 STO 03
Entrez:                      |  02 "ABC[illegible]"
     03 LBL 11               |  03 LBL 00
     04 DEC          ------> |  04 X>Y?
     05 RCL 09               |  05 X≤Y?
     06 ACOS                 |  06 Σ+
                             |     .END.
```

C'est votre premier exemple de 'programmation synthétique', où des combinaisons d'octets non entrables au clavier sont synthétisés d'une façon extraordinaire. Selon les règles d'édition au niveau de l'octet de la section 3A, quand nous avons essayé d'insérer le 'LBL 11', le processeur a du insérer 7 nuls pour créer de la place pour la nouvelle ligne. Ce qui a aussi procuré de la place à 'DEC', 'RCL 09', et 'ACOS', mais dans le même temps poussé les octets "DEFG" en bas de la mémoire, bien loin de la portée de l'octet 'F7'. De ce fait, ces quatre octets deviennent les lignes mono-octet 03 à 06.
N'importe quelle ligne synthétique de texte de la même sorte créée par le sauteur d'octets sera éxécutée normalement, comme vous pouvez le vérifier en pas à pas pour la ligne 02 (PRGM off), puis en controlant le registre alpha pour voir les caractères résultants. Nous pouvons utiliser cette technique pour placer dans une ligne de texte programme l'un des 19 caractères non entrables au clavier, plus les symboles texte et append (mais pas le canard). Chacun de ces caractères est dans la partie supérieure de la table d'octets, de façon que chacun puisse être édité comme ci-dessus en utilisant une ligne programme mono-octet directement entrable au clavier. Bien plus, tous les 128 caractères de l'imprimante peuvent être placés dans une ligne de texte (certains sans équivalents à l'affichage seront remplacés par des pleins), pour être transférés dans le buffer par 'ACA' (Voir section 6E).
Les lignes programmes qui concatènent des nuls à la chaine alpha existante sont fréquemment utilisés dans les programmes synthétiques. Voici un exemple de création d'une de ces lignes, dans ce cas pour concaténer 5 nuls:

```
Entrez:
     01 ASTO 02
     02 "[illegible]ABCDE"
JUMP
DEL 005
DEL 001
```

La ligne '02 "[illegible]ABCDE" est choisie pour avoir le même nombre de caractères que le nombre de nuls à concaténer. Le 'DEL 005' change les caractères en nuls. Le 'DEL 001' enlève le 'ASTO 02' utilisé pour controler le sauteur d'octets.
L'édition par sauteur d'octets décrite plus haut est limitée par le fait que le premier caractère d'une ligne de texte ne peut pas être changé, sauf en l'effaçant pour un nul. Dans "ABCDEFG" par exemple, le "A" ne peut pas être altéré, puique entrer un nouvel octet à la position du "A" demanderait que l'octet précédent soit affiché avant le remplacement de l'octet, mais puisque l'octet précédant le "A" est 'F7', l'affichage persiste à afficher la ligne complète.
Cependant, des lignes de texte arbitraires, avec des caractères non entrables directement dans n'importe quelle(s) position(s), peuvent être créées par une élaboration de la technique du sauteur d'octets. Supposons que nous voulions créer la ligne de texte "([illegible])", i.e., les octets 'F3 28 23 29'. Dans le précédent exemple, les octets de texte étaient créés normalement par l'intermédiaire de caractères temporaires, qui sont ensuite éjectés de la ligne par les caractères désirés. Cette fois, ce sera l'octet de texte lui-même qui sera créé à l'intérieur d'une autre ligne de texte. les caractères de texte désirés seront d'abord entrés comme des instructions mono octet normales. Commençons par cette routine:

```
01 STO 01        31
02 "BJ"          F2 42 4A
```

La ligne '01 STO 01' est utilisée pour procurer le nybble '1' nécessaire pour un saut d'octets de un octet; nous parlerons de cette ligne comme du 'controleur', puisqu'elle controle la longueur du saut d'octets. De même la ligne '02 "BJ"' est une ligne de texte temporaire que nous appellerons le 'générateur'. Le 'controleur' et le 'générateur' sont à effacer une fois que l'édition par le sauteur d'octets est complète. Exécutez maintenant 'JUMP .002' et entrez la ligne '03 "ABC"'. Le programme complet est maintenant:

```
01 STO 01        31
02 "B■"          F2 42 F3
03 -             41
04 *             42
05 /             43
06 HMS-          00 00 00 4A
```

L'entrée de la ligne '03 "ABC"' place un 'F3' aussitôt aprés le '42', où il devient le dernier octet de la ligne génératrice. Le '41 42 43' ("ABC") n'est pas dans le générateur, donc ces octets seront montrés comme trois lignes programme indépendantes, 03 à 05. Enfin, nous avons l'octet '4A', le "J" de départ, qui a été expulsé du générateur par l'insertion, il est donc devenu la ligne '06 HMS-'. Les trois nuls laissés par l'insertion sont, comme d'habitude, invisibles.

Ensuite, pressez 'GTO .002' et entrez :

```
02 RCL 08
03 RCL 03
04 RCL 09
```

pour placer les octets '28 23 29' immédiatement aprés l'octet 'F3'. Puis 'JUMP .002', entrez '03 HMS-'. Le programme est maintenant:

```
01 STO 01        31
02 "BJ"          F2 42 4A
03 "(#)"         00 00 00 00 00 00 F3 28 23 29
04 -             00 00 00 00 41
05 *             42
06 /             43
07 HMS-          00 00 00 4A
```

L'insertion de 'HMS-' pousse le 'F3' dehors du générateur, où il réassume son rôle en temps que 'text 3', agrippant les octets '28 23 29' pour compléter la ligne ce texte en tant que caractères "(", "#", et ")" respectivement. Les nuls variés sont les restes des groupes de 7 nuls placés dans le programme à chaque insertion. Pour 'nettoyer', nous effaçons les lignes 01, 02 et 04-07, puis 'PACK'. Avec un peu de pratique , vous trouverez que toute la procédure va trés vite. L'édition par sauteur d'octets n'est en aucune façon restreinte aux lignes de texte, particulièrement en utilisant la méthode du controleur-générateur décrite en dernier. Comme exemple amusant, essayez ceci: commencez de nouveau avec la routine 3B-2 (reprendre les instructions de la section 1E):

```
                 JUMP .002       (02 *          )
03 TONE 1
                 GTO .002        (02 "B■"       )
03 LBL 09
                 JUMP .002       (02 *          )
03 HMS-
                 SST             (04 TONE 0     )
```

Le 'TONE 0' a l'air normal, mais éxécutez le en pas à pas (PRGM off). Vous devez entendre une nouvelle basse fréquence durant plus de 2 secondes!
Avec la méthode du controleur-générateur, nous pouvons créer à peu prés n'importe quelle combinaison de postfixe et de préfixe que nous voulons, pour faire des fonctions synthétiques à deux octets. Les plus importantes de ces fonctions étant celles qui donnent accés aux registres d'état, que nous allons examiner au chapitre 4.

CHAPITRE 4

LES REGISTRES D'ETAT

4A ETRANGES POSTFIXES

Vers la fin de la section 2b, nous avons éxaminé en détails la table d'octets, tenant compte de presque chaque entrée de la table. Mais il y a encore un aspect important à explorer: Remarquez que les octets 64-65 ont leurs postfixes deux fois sous-lignés, et que chaque octet 66-6F a deux valeurs de postfixes, dont la seconde est sous-lignée. Les postfixes sous-lignés n'apparaissent jamais dans les programmes 'normaux' de la HP 41C. Considérez les postfixes 66 à 6F, décrits comme les lettres "A" à "J". Ces lettres-postfixes ne se rencontrent que dans les ligne de programme incluant des labels alpha locaux, comme 'LBL C' ou 'XEQ F'. La logique du clavier nous empêche de mettre ces postfixes aprés d'autres préfixes. Par exemple quand nous pressons 'STO', la touche ALPHA est neutralisée pour que seuls les postfixes numériques puissent être entrés. Mais le sauteur d'octets nous a affranchis des contraintes du clavier, essayons donc un 'STO A'.

```
01 STO 01
02 "BJ"
                JUMP .002          (02 *           )
03 STO 22 (postfixe arbitraire)
                GTO .002           (02 "B"         )
03 X<0?
                JUMP. 002          (02 *           )
```

A cette étape, nous sommes prêts à expulser le préfixe 'STO' du générateur. Mais pendant que nous y sommes, mettons aussi un 'RCL A':

```
03 RCL 22 (expulse le 'STO', mets le 'RCL' dans le générateur)
                GTO .002           (02 "B"         )
03 X<0?
                JUMP .002          (02 *           )
03 HMS-
```

Le programme est maintenant :

```
01 STO 01
02 "BJ"
03 RCL A
04 6
05 STO A
06 6
07 HMS-
```

Les lignes '6' (04 et 06) sont les mono-octet équivalents aux octets de postfixe '22'. Maintenant , essayez:

```
                SIZE 103
                PRGM off
    12345
                GTO .005
                SST
                CLX                (0.0000         )
```

A cette étape, le nombre '12345' a disparu de l'affichage. Pour le retrouver, pressez 'GTO .003',SST.Le nombre réapparait, montrant bien qu'il était stocké dans le registre A. Pour voir où le nombre est allé en réalité, stockez 102 dans Roo, puis pressez 'VIEW IND 00'. Comme vous pouvez le constater, 'STO A' est équivalent à 'STO 102'. Si les postfixes numériques continuaient plus loin que le nombre décimal 99, le postfixe 'A' serait 102. La programmation synthétique nous permet ainsi d'étendre l'accés aux registres de données jusqu'à R111 (postfixe "J"), nous conduisant aux postfixes sous-lignés des octets 66-6F de la table d'octets.

Mais pourquoi s'arréter à 111? Il y a une autre ligne de postfixes disponible. Ne pouvons nous pas les utiliser pour accéder jusqu'aux registres 127? Notez d'abord que les postfixes 70, 71, 72 73, et 74 sont déjà utilisés pour accéder aux registres de la pile T ,Z ,Y ,X et L,

respectivement. Mais les octets restants nous font signe: ils semblent être les touches de la 'boite de Pandore' de la programmation synthétique! Essayons un exemple dramatique: utilisant la routine 4A-1:

```
                        JUMP .002     (02 *          )
03 STO 22
                        GTO .002      (02 "B■"       )
03 AVIEW
                        JUMP .002     (02 *          )
03 HMS-
                        GTO .003      (03 STO d      )
```

Maintenant, mettez le mode programme; pressez 'SF 00, SF 01, SF 02, SF 03, SF 04, FIX 9, SF 28, GRAD, USER, CLX, ALPHA', ce qui allume différents indicateurs. Pressez 'SST' une fois. Comme l'affichage résultant est complétement vide, la simple opération 'STO d', avec 0 dans le registre X, baisse tous les 56 drapeaux de la HP 41C en deux coups de cuillère à pot! L'implication est évidente -- le code bi-octet '91 7E', ou 'STO d', que nous avons fait avec le sauteur d'octets, nous permet de stocker directement dans un registre spécial, que nous devrons appeler le 'regsitre d', et qui contient les 56 drapeaux.

Les registres d'état (registres 000-00F) mentionnés à la fin de la section 2D sont enregistrés par le lecteur de cartes au moyen de l'opération 'WSTS'. Les états de tous les drapeaux utilisateurs et système est une partie de l'information stockée sur la carte d'état, d'où nous concluons que le registre d est un des registres (16) d'état. Le fait que les registres de la pile , également accessibles par les postfixes de la ligne 7 de la table d'octets, sont aussi des registres d'état, nous conduit, d'un saut hardi, à dire que tous les postfixes de la ligne 7 se réfèrent aux registres d'état-- 16 registres, 16 postfixes . IL ne reste plus qu'à identifier le rôle de chacun de ces registres. Par référence aux opérations 'WSTS' et 'WALL', nous attendrions des registres d'état de contenir, en plus, le registre alpha, le pointeur d'adresses, la pile de retour des sous routines, la SIZE courante,et la position des registres de données. Comme décrit dans la table d'octets, les postfixes 75 à 7F sont affichés comme 'M', 'N', 'O', 'P', 'Q'; '⊢', 'a', 'b', 'c', 'd', et 'e' respectivement. La deuxième valeur de postfixe indiquée pour les octets 75 à 7A sont les postfixes montrés par l'imprimante pour les fonctions utilisant les registres d'état. Ainsi la ligne 'STO M' apparue telle à l'affichage sera imprimée comme 'STO [' .La correspondance est indiquée dans la table 1-1.

De façon à pouvoir étudier les propriétés et les utilisations de chacun des registres d'état, nous synthétiserons les lignes programme qui nous permettrons de visualiser ou de changer le contenu de ces registres. La fonction 'X<>'sert ce dessein d'une façon admirable, puisqu'elle peut agir à la fois comme STO et comme RCL. Etant donné que le stockage dans les registres a,b, c peut produire occasionnellement des résultats déplaisants ('0 STO c'est l'exemple le plus déplaisant: il provoque un 'MEMORY LOST'), nous nous limiterons pour l'instant à la description verbale de ces registres. A titre d'exercice, vous devriez essayer de créer la routine 4A-2 en utilisant le sauteur d'octets sans aide. Référez vous aux instructions du paragraphe suivant si vous êtes perdu. Comme nous avons fait pour la création de 'STO A' et de 'RCL A', nous limiterons les séquences de touches en utilisant chaque 'X<>' successif pour expulser le précédent du générateur. Si vous avez effacé l'assignation de 'X<>' que nous avions faite au chapitre 3, vous devez la refaire maintenant.

Commencez par la routine 3B-2. Comme d'habitude, le choix d'un postfixe temporaire pour 'X<>' est arbitraire (simplement, n'utilisez pas 'X<>29, 30 ou 31', car les postfixes correspondants sont eux-même des préfixes). Le meilleur choix est 'X<>00', puisque le '00' est hex '00', le nul Tous les postfixes restants sont invisibles et seront éliminés par 'PACKING'. Maintenant, essayez:

```
                        JUMP .002     (02 *          )
03 X<>00
                        GTO .002      (02 "B■"       )
03 CLD    (e)
                        JUMP .002     (02 *          )
03 X<>00
                        GTO .002      (02 "B■"       )
03 AVIEW  (d)
                        JUMP .002     (02 *          )
03 X<>00
                        GTO .002      (02 "B■"       )
03 SIGN   (⊢)
                        JUMP .002     (02 *          )
```

```
03 X<>00
                    GTO .002      (02 "B■"        )
03 X≠Y?  (Q)
                    JUMP .002     (02 *           )
03 X<>00
                    GTO .002      (02 "B■"        )
03 X=Y?  (P)
                    JUMP .002     (02 *           )
03 X<>00
                    GTO .002      (02 "B■"        )
03 CLX   (O)
                    JUMP .002     (02 *           )
03 X<>00
                    GTO.002       (02 "B■"        )
03 LAST X  (N)
                    JUMP .002     (02 *           )
03 X<>00
                    GTO.002       (02 "B■"        )
03 RDN   (M)
                    JUMP .002     (02 *           )
03 HMS-
```

Suivant cette séquence, nous arrivons à la routine 4A-2 (effacez la ligne '11 HMS-'):

```
01 STO 01        06 X<>P
02 "BJ"          07 X<>Q
03 X<>M          08 X<>F
04 X<>N          09 X<>d
05 X<>O          10 X<>e
```

4A-2

Maintenant, si nous voulons éxécuter un 'x<>M', par exemple, en mode prgm off, nous pressons 'GTO .003', 'SST'. Dans le reste de ce chapitre, nous utiliserons les ligne de programme de la routine ci dessus pour explorer les registres d'état. Les applications pratiques à la programmation des propriétés que nous venons de découvrir seront discutées au chapitre 6. La figure 4-1 est un diagramme résumant les utilités des différentes parties des registres d'état, dans une présentation similaire à celle de la figure 2-5.

4B. LE REGISTRE ALPHA

Un registre programmable de la HP 41C est long de 7 octets. Le registre alpha apparait être une exeption à cette règle, puisque nous pouvons stocker jusqu'à 24 octets (caractères) en alpha. En fait, le registre alpha réside dans les 4 registres d'état M, N, O et P. Quatre registres nous donnent un total de 28 octets, mais seulement 24 d'entre eux peuvent être affichés, ou accessibles par 'ASTO' et 'ARCL'.
Pour illustrer la structure du registre alpha, nous pouvons utiliser la routine 4A-2. Commençons par entrer 24 caractères dans le registre alpha, comme "ABCDEFGHIJKLMNOPQRSTUVWX". Si le drapeau 26 est levé, vous entendrez le bip d'alarme en entrant le "X", pour vous informer que le registre alpha est plein. Pressez maintenant 'GTO .003', 'CLX', 'ALPHA(on)', 'SST', pour voir "ABCDEFGHIJKLMNOPQ⁻⁻⁻⁻⁻⁻⁻". Le surlignage "⁻" est le caractère correspondant au nul. Le 'CLX' remplit le registre X de nuls. Le 'SST' éxécute un 'X<>M', déplaçant les nuls dans le registre M, qui se révèle être les 7 caractères les plus à droite du registre alpha. Si vous éteignez le mode alpha, puis mettez la HP 41C en Fix 9, vous verrez '-2.5354555 E-42' dans X. Le code héxa pour la chaine alpha "RSTUVWX" est '52 53 54 55 56 57 58'. Ces octets sont maintenant dans le registre X, où le processeur essaie vaillamment d'afficher un nombre décimal. Il y a un '5' dans le digit de signe de la mantisse, et un '7' pour le signe de l'exposant, résultants dans les signes moins (voir section 5A). Les digits de la mantisse sont tous des digits décimaux normaux, donc la mantisse apparait comme '2.535455565', les deux derniers digits étant supprimés pour laisser de la place à l'exposant. L'octet de code pour un exposant négatif est le complément de l'exposant, '100-xy'(décimal); dans ce cas nous voyons un exposant (100-58=42).
Pour continuer votre exploration, essayez 'ALPHA(on)', 'SST' pour voir "ABCDEFGHIJRSTUVWX⁻⁻⁻⁻⁻⁻⁻"
Le 'SST' a éxécuté la ligne '04 X<>N', et donc le contenu original du registre M, "RSTUVWX", a été dans le registre N, les 7 octets suivants du registre alpha. Pour compléter l'exercice, entrez 'ALPHA(off)', 'GTO .003', 'SST', 'ALPHA'; l'affichage est "ABCDEFGHIJRSTUVWXKLMNOPQ".. les contenus originaux des registres M et N sont interchangés. Finalemant, les 24 caractères originaux sont divisés comme suit:

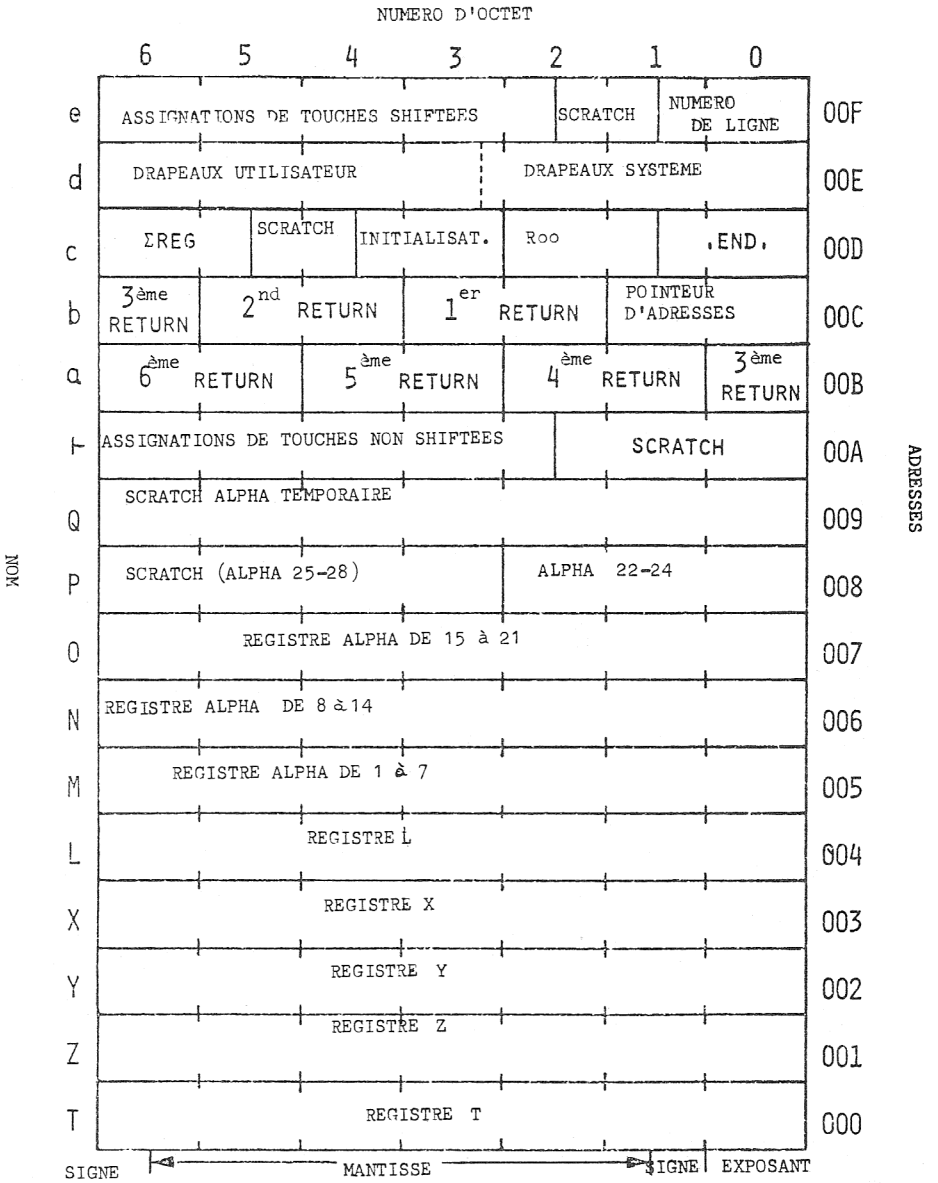


FIGURE 1-4 LES REGISTRES D'ETAT

```
                    xxxABC/DEFGHIJ/KLMNOPQ/RSTUVWX
Registres:          P       O       N       M
```

Quand une nouvelle chaine est entrée dans le registre alpha, le premier caractère entré pénétre dans le dernier octet (l'octet d'exposant) du registre M, adresse 0005. Le caractère suivant va aussi en 0005, poussant le précédent vers la gauche dans le registre alpha, i.e., vers l'avant-dernier octet du registre M, l'octet 1005. Les caractères suivants continuent à progresser suivant le même principe; lorsque M est plein, le caractère suivant pousse le premier dans le dernier octet du registre N, l'octet 0006, et ainsi de suite dans les registres O et P. Quand un caractère entre dans le troisième octet avant la fin du registreP, 2008, le bip d'alarme tone. Les caractères suivants poussent les caractères de tête dans les quatre premiers octets de P,où ils disparaissent de l'affichage.
Pressez maintenant 'GTO .006', mettez en mode alpha, et concaténez encore 4 caractères "YZ=?" aux 24 originaux. "ABCD" disparait, mais étonnement: il est encore présent dans le registre P. Pressez 'SST' une fois, pour éxécuter 'X<>P', puis 'ALPHA(off)'. Vous verrez '-1.4243444 E-53'. Pour traduire ceci en caractères, essayez 'GTO .003', 'ALPHA', 'CLA', 'SST'('X<>M'). Les caractères "ABCDEFG" réapparaissent.. ceux qui étaient le contenu original de P, et que nous avons mis dans le registre M via le registre X. Cepndant, si nous répétons tout le processus commençant avec l'entrée des 28 caractères, mais, cette fois, éteignant le mode alpha avant le 'SST' qui éxécute le 'X<>P', nous nous retrouvons avec "▮▮CDEFG". Le processeur utilise occasionnellement les 4 premiers octets de P comme scratch, faisant disparaitre tout ou partie de son contenu original. Evidemment, presser la touche 'ALPHA' demande au processeur d'utiliser le registre P. Une étude de Charles Close a révélé que, pendant l'éxécution d'un programme, seuls 'VIEW' ,'AVIEW' et les lignes de programme qui font entrer des nombres provoquent la perte des octets de tête du registre P. Si ces pas sont évités, nous pouvons utiliser un registre alpha plein de 28 caractères pour les traitements de chaine alpha. Comme exemple d'utilisation par le processeur du registre P, les octets 1, 2 et 3 sont utilisés pour enregistrer la size courante et l'emplacement des registres statistiques sur une carte magnétique par l'opération 'WSTS'.
Il y a deux importantes zones d'applications à l'accés direct aux registres M, N, O, et P. D'abord, comme démontré dans les exemples précédents, nous avons obtenu une nouvelle sorte de manipulation de chaine alpha qui, ajoutée aux conventionnels 'ASTO', 'ARCL', 'APPEND' et 'ASHF', produit un tri de caractères efficient et rapide, utile pour l'affichage, les jeux, le tri de mots,etc. La deuxième application est l'utilisation du registre alpha comme trois (ou 4, si on inclue le registre P) registres de données additionnels, avec les même capacités que les registres de données normaux, mais avec l'avantage d'un emplacement fixe en mémoire et un rappel non-normalisant (section 5B). Ces applications seront étudiées en détails aux chapitres 5 et 6.

4C. LE REGISTRE Q

Le registre Q est en principe un registre de scratch pour le processeur. Il est utilisé tellement fréquemment qu'il est virtuellement inutile comme registre additionnel de données. De première importance pour la programmation synthétique est l'utilisation du registre Q pour le stockage temporaire des chaines alpha qui n'entrent pas directement dans le registre alpha. De telles chaines sont obtenues pendant l'éxécution de fonctions ou de programmes qui sont 'épelés' par l'utilisateur, ou pendant l'enregistrement de lignes programme de texte. Par exemple, en utilisant la routine 4A-2, essayez 'XEQ' 'ALPHA' "GTO" 'ALPHA' '.007', 'SST', 'GTO.003', 'ALPHA' 'CLA', 'SST'. Le "OTG" maintenant dans le registre alpha est l'inverse des lettres "G", "T","O" que vous avez utilisées pour épeler "GTO". Cette configuration peut être utilisée pour simplifier la création de lignes programme non entrables directement au clavier (section 51).

4D. LE REGISTRE DES DRAPEAUX

Nous avons découvert au début de ce chapitre que le registre d 'contient' tous les 56 drapeaux utilisateurs et système de la HP 41C. Si nous nous souvenons que chaque registre consiste en exactement 56 bits, il devient évident que chacun des drapeaux est juste un des bits du registre d. Le premier (ou le plus à gauche, ou le plus haut, ou le plus significatif, selon votre façon de visualiser un registre) bit est la drapeau 00, le second est le drapeau 01, et ainsi de suite jusqu'au 56e bit, drapeau 55.
Comme exemple du comportement du registre d, configurez votre HP 41C comme suit: SF 04, SF 09, SF 17, SF 18, SF 26, USER(on), SF 28, FIX 9, RAD,; tous les autres drapeaux baissés. Maintenant en utilisant la routine 4A-2, rappelez le contenu du registre d en pressant: 'GTO .009', 'SST', 'ENTER', 'BST', 'SST', 'rdn', 'ALPHA', 'CLA', 'ARCL X'. . Cette séquence nous permet de voir le

contenu du registre d sans le changer. Nous utilisons le 'ARCL X' pour voir tous les 10 chiffres de la mantisse aussi bien que l'exposant. Rappelons que la mantisse et l'exposant positifs correspondent à 0 dans les digits de signe; nous en concluons depuis l'affichage alpha que les octets de d sont '08 40 60 38 09 90 10', ce qui est, en l'écrivant sur 56 bits (groupés en nybble)

```
Drapeaux:        4    9        17 18      26 27 28      36 39 40 43       51
Bits:      0000 1000 0100 0000 0110 0000 0011 1000 0000 1001 1001 0000 0001 0000
```

Chaque '1' dans la chaine correspond à un drapeau levé. Le premier un à partir de la gauche, dans le second nybble, est le drapeau 4, par exemple. Le suivant est le drapeau 9, et ainsi de suite jusqu'au dernier '1', dans le second nybble à partir de la droite, qui est le drapeau 51, le drapeau du 'SST', qui était levé momentanément à cause de l'utilisation de 'SST' pour éxécuter 'X<>d'.
Pour vous donner un avant goût de ce qui peut être fait à travers l'utilisation du registre des drapeaux, multipliez le nombre existant '8.406038099 E10' dans le registre X par '1 E30'. Puis éxécutez 'GTO .009', 'SST'. Non, vous n'avez pas de batterie défaillante, vous avez juste levé le drapeau 49, le drapeau de batterie défaillante. (pour le baisser, éteindre puis rallumer la HP 41C). Ceci est un exemple de l'utilité des octets controlés par l'utilisateur,i.e., le nombre que vous avez placé dans X, pour controler les drapeaux système à travers l'échange avec le registre d. Considérons maintenant le processus inverse -- utiliser le controle sur les drapeaux pour créer des octets arbitraires dans le registre d, d'où ils peuvent être transférés dans le registre X et ailleurs avec les fonctions d'accés aux registres d'état (voir chapitre 5). C'est l'utilisation de ce concept qui a ,au départ, mené à un développement sérieux de la programmation synthétique.

4E. LES DRAPEAUX D'ASSIGNATION

Quand une touche est pressée en mode USER, si cette touche est assignée à une autre fonction que sa fonction par défaut, le processeur doit compiler les labels globaux utilisateurs et les registres d'assignations pour découvrir quel programme ou fonction cette touche est sensée éxécuter. Pour s'épargner un monceau de recherches inutiles, la HP 41C garde un groupe de 72 'drapeaux d'assignation de touches', un pour chaque touche et pour chaque touche shiftée (en comptant la touche imaginaire sous la touche 'enter'). Quand une touche utilisateur est pressée , le processeur commence par vérifier le drapeau d'assignation correspondant. La recherche ne commence que si le drapeau est levé.
Comme dans le registre d, un bit de mémoire est utilisé pour chaque drapeau d'assignation. Comme 72 bits sont trop pour un seul registre, les drapeaux d'assignation sont divisés entre le registre ⊢ et le registre e. Les 36 drapeaux non shiftés sont les 36 premiers bits du registre ⊢ ; les drapeaux des touches shiftées sont de même situés dans le registre e. La figure 4-2 montre la correspondance entre numéro de bit et emplacement de la touche.
Pour voir un de ces registres 'en action', nous utiliserons le registre ⊢ comme illustration. La seule assignation de touche que nous avons faite jusqu' ici a été faite au chapitre 3, i.e., l'assignation du sauteur d'octets à la touche 'Σ+', et l'assignation de la fonction 'X<>' à la touche '+'. Si vous avez fait d'autres assignations entre temps, vos résultats différeront de ce qui va être montré, donc vous pouvez souhaiter effacer ces assignations supplémentaires.
Pour visualiser le contenu du registre ⊢ en utilisant la routine 4A-2, pressez 'GTO.008', 'CLX' 'SST', 'FIX 7'. Vous devez voir '0.0000021'. Les 6 zéros plus le signe positif montrent que les premiers (7*4=28) bits du nombre (qui est le contenu original du registre ⊢) sont des o. Le chiffre '2', qui est '0010' en binaire, indique que le drapeau d'assignation 31 est levé; le '1', 0001 en binaire, provient du drapeau d'assignation 36. En se référant à la figure 4-2, nous voyons que ces bits correspondent aux touches '+' et 'Σ+' respectivement, qui sont justement les touches que nous avons assignées.
L'inspection du contenu des registres ⊢ et e nous donne ainsi une façon rapide de retrouver que les touches ont été assignées sans avoir recours à l'imprimante. Nous avons utilisé 'FIX 7' parce que nous n'étions intéressés que par les 9 premiers nybbles du registre. Ce moyen n'est pas vraiment général dans son application. Si trop de touches ont été assignées, le nombre provenant d'un 'RCL ⊢' ou d'un 'RCL e' peut contenir des chiffres héxa A-F, qui peuvent être difficiles à déchiffrer à l'affichage (voir section 5A). Vous devez être sûr de restaurer le contenu original du registre d'assignation; sinon vous perdrez les assignations de touches , incluant la portion de mémoire utilisateur utilisée pour coder ces assignations. Si vous pressez maintenant la touche '+' en mode USER, la HP 41C éxécutera '+' au lieu de 'X<Y', car le '0' stocké dans le registre avec 'X<>⊢' a éffacé tous les drapeaux d'assignation de touches.

Pour les retrouver, pressez ('LAST X', si vous avez éxécuté le '+') 'GTO .008', 'SST'. Si vous perdez accidentellement le contenu du registre ⊢ ou e en jouant à ces jeux-là, l'éxécution du 'WSTS' du lecteur de cartes suivi par une relecture des états de la carte restaurera les assignations de touches originales.
Les trois derniers nybbles du registre e sont l'emplacement du numéro de la ligne programme (codée en héxa). Si les nybbles du numéro de ligne sont '000', comme aprés un 'GTO .000', un 'RTN' manuel, ou l'éxécution d'un programme se terminant par 'END', l'affichage programme suivant sera '00 REG lmn'. Quand l'éxécution programme s'arrête ailleurs que sur un 'END', le numéro de ligne est forcé à 'FFF'. Quand le processeur voit ce numéro de ligne mytique, il sait qu'il doit recalculer le numéro de ligne avant de montrer la ligne de programme courante lors de l'activation suivante du mode PRGM ou d'un 'SST'.

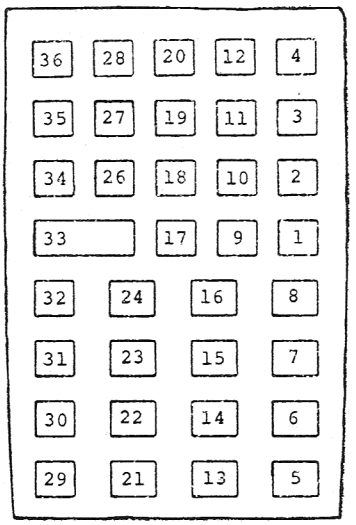


FIGURE 4-2. LES BITS/DRAPEAUX DES ASSIGNATIONS DE TOUCHE

4F. LE POINTEUR D'ADRESSES ET LA PILE DE RETOUR

Dans le chapitre 2, nous avons appris que le pointeur d'adresses utilise une adresse sur 4 digits consistant en le numéro d'octet plus trois digits pour le numéro du registre. Le pointeur d'adresses lui-même est les quatre derniers nybbles du registre b. C'est à dire, si le pointeur d'adresses est positionné sur l'octet 'n' du registre 'abc', alors le registre b contiendra le nombre '0000000000nabc'. Si une sous routine est appelée, l'adresse de retour est enregistrée dans les quatre nybbles suivants à la gauche du pointeur d'adresses, avec la nouvelle adresse courante, suivant le saut à la sous-routine, mis dans les nybbles du pointeur d'adresses. Quand une routine suivante est appelée, les adresses de retour précédentes sont pousseés sur la gauche. Quand le registre b est plein, les adresses continuent dans le registre a, de sorte qu'il y a suffisamment de place dans les registres a et b pour l'adresse courante et 6 adresses de retour. A n'importe quel moment, le contenu des registres a at b sont ainsi:

```
/yzab uvwx qrst mn/op ijkl efgh abcd/
/ registre a      / registre b     /
```

où 'efgh' est la première adresse de retour, 'ijkl' la seconde, et ainsi de suite. Quand un 'RTN' ou un 'END' est éxécuté, la pile des retours est décalée vers la droite, pour que la première adresse de retour devienne le nouveau pointeur d'adresses, la seconde adresse de retour devienne la première, etc. La seule complication dans ce shéma merveilleux est que les adresses de retour sont écrites d'une manière un peu différente de celle du pointeur d'adresses. Par exemple, si le retour à adresser est '3160', l'adresse sera stockée dans la pile de retour par '0760'. Le '0760' est réellement une forme comprimée de '3160'. Si nous écrivons tous les bits de '3160', nous obtenons:

3160 = 0011 0001 0110 0000

Pour les adresses de programme utilisateur (i.e., emplacements dans la HP 41C ou dans les modules de mémoire) les premier , cinquième, sixième et septième bits sont toujours nuls , puisque le premier nybble ne prend ses valeurs que jusqu'à 6 (0110) et que le second est toujours 0000 ou 0001. Donc dans le pointeur d'adresses, les trois bits utilisés du premier nybble sont déplacés à la place des trois bits inutilisés du second. Le '3160' devient:

0760 = 0000 0111 0110 0000

Cette compression du code adresse permet au premier nybble de contenir d'autres informations. De fait, si le premier nybble est zéro, le retour est à une adresse de programme utilisateur. Alors que si ce n'est pas zéro, le processeur sait devoir retourner à une adresse contenue dans un système de périphérique. Par exemple, toutes les adresses dans la mémoire de l'imprimante commencent par le nybble '0110'.
L'application la plus évidente de la connaissance de la structure de la pile de retour et de l'accés par les fonctions synthétiques de cette pile de retour, comme 'STO b' ou 'RCL a' est de permettre à l'utilisateur de bouger le pointeur en des positions arbitraires de la mémoire, y compris dans les registres d'assignation de touches et même dans les registres de données. De même, utilisant un 'STO b', nous pouvons amener le pointeur directement dans n'importe quel octet d'une ligne programme multi-octets pour une édition, comme nous l'avons fait avec le sauteur d'octets. Pour un controle complet de ces opérations, nous avons besoin tout d'abord de développer un moyen aisé nous permettant de générer des codes héxa à 7 octets, et de déchiffrer ces codes. Ce moyen sera développé dans le chapitre 5.

4G. LE REGISTRE c ET LA REPARTITION DE LA MEMOIRE

La dernière partie de la mémoire de la HP 41C non encore explorée est le registre c, qui est plein de nybbles et d'octets interessants. Les 14 bits héxa du registre c sont comme suit:

stu/vw/169/mno/pqr

où les lettres représentants les digits sont regroupées comme les digits sont utilisés par le processeur. Les lettres sont choisies pour correspondre à la répartition mémoire montrée à la figure 2-5. Les trois premiers digits,'stu', sont l'adresse absolue du premier registre statistique parmi les 6 spécifiés par la fonction 'ΣREG'. Cette adresse est changée à chaque fois que 'ΣREG' ou 'SIZE' est éxécutée. Quand 'Σ+', 'Σ-', 'CLΣ', 'SDEV', ou 'MEAN' est éxécutée, le processeur se réfère aux digits 'stu' pour trouver quels registres sont désignés comme registres statistiques.
Les deux digits suivants,'vw' sont utilisés comme scratch par l'imprimante. Les digits 6, 7 et 8, montrés avec la valeur explicite '169', sont intéressants d'une façon perverse, puisqu'ils sont ainsi nommés 'constante de départ à froid'. A différents moments durant les opérations, particulièrement quand la HP 41C est mise sous tension, le processeur vérifie la valeur de ces 3 digits et les compare à la valeur fixe '169'. Si les digits ont une autre valeur que 169, le processeur en conclut que quelque chose de dramatique est arrivé à la mémoire, et prend donc la décision irréversible d'effacer la mémoire et d'afficher MEMORY LOST. Ceci explique pourquoi '0 STO c' cause MEMORY LOST: cette opération efface le 169.
Les digits 'mno' et 'pqr' sont des numéros de registres à 3 digits, correspondant aux registres 'mno' et 'pqr' écrits sur la figure 2-5. 'mno' est l'adresse absolue du plus bas des registres de données, Roo. Le processeur se réfère à 'mno' pour chaque usage des registres de données. Le registre de données 'Rab' est le registre de mémoire à l'adresse absolue (mno+ab), avec 'ab' converti en héxa.
Les digits 'pqr' indiquent l'emplacement courant du .END. permanent. Comme la chaine des labels globaux et des END commence avec le .END., un 'GTO (alpha)' ou un 'XEQ (alpha)' commence la recherche du label par le registre 'pqr'.
En dépit du risque de perte de la mémoire inhérent au registre c, il y a beaucoup d'applications importantes de la manipulation de son contenu. En premier, il y a le controle de la ligne séparant les mémoires programme et données, et la possibilité de stocker des données dans les registres de programme, comme sera dit au chapitre 5 .

CHAPITRE 5

DES PROGRAMMES POUR LA PROGRAMMATION

Le thème central de ce chapitre est de développer une série de programmes pour la HP 41C qui nous permettrons de créer, de déchiffrer, stocker, rappeler arbitrairement des codes héxa à 7 octets, nous donnant un grand pouvoir de programmation synthétique. Les programmes 'pour programmer' et leur techniques sont fait pour éxécuter des tâches spécifiques de programmation synthétiques, tout en se comportant comme des exemples d'utilisation des fonctions synthétiques. Même si vous n'utilisez pas tous les programmes de ce chapitre, vous trouverez interessant d'étudier les techniques de programmation et la philosophie contenues dans les routines. Nous supposons ici que vous maîtrisez suffisamment les techniques du sauteur d'octets décrites aux chapitres 3 et 4 pour être capable d'entrer n'importe quelle fonction synthétique à 2 octets, comme 'RCL M' ou 'X<>d', à chaque fois que nécessaire dans un programme. Comme nous l'avons fait en créant la routine 4A-2, vous trouverez commode, en sautant les octets de plusieurs fonctions, de commencer par la fin du programme et de remonter, utilisant chaque préfixe entré dans le générateur du sauteur d'octets pour éjecter le précédent. Aprés que toutes les fonctions synthétiques sont entrées, vous pouvez insérer les fonctions normales par des séquences de touches ordinaires.

Trois techniques synthétiques de programmation sont traitées dans ces programmes:

1. Usages multiples des drapeaux: L'accés direct au registre d permet à un groupe de drapeaux utilisateurs d'être utilisés pour plusieurs buts simultannément, en sauvant simplement un état des drapeaux avec 'RCL d' ou 'X<>d', utilisant les drapeaux pour un autre motif, puis réstaurant l'état initial de tous les drapeaux avec 'RCL d' ou 'X<>d'. Ceci permet ,par exemple, à un programme d'employer différents états trigonométriques et d'affichage en remettant toujours la HP 41C à la fin du programme dans l'état de drapeaux favorit à l'utilisateur.

2. Utilisation des drapeaux comme bits: Le contrôle et le test par l'utilisateur des drapeaux 0-29 permet une conversion programmable de nombres binaires sur 30 bits vers ou depuis le décimal ou l'octal. Cette technique est le moyen indispensable pour créer et déchiffrer des octets de mémoire et des codes multi-octets de façon automatique.
3. Manipulation de chaines alphanumériques: Les puissantes fonctions synthétiques d'accés au registre alpha, utilisées dans ce chapitre pour assembler des octets individuels en code à 7 octets, et vice-versa, sont rencontrées sans cesse dans la programmation synthétique pratique, comme nous verrons dans le chapitre 6.

La HP 41C est dessinée pour ne procéder que sur des données programme ou utilisateur étant des nombres décimaux normaux. Nous pouvons, de ce fait, dire que demander au prcesseur d'opérer avec des nombres contenant des digits 'A' à 'F' causera au moins des attitudes inattendues. Avant d'explorer les programmes promis, nous devons d'abord apprendre comment les codes à 7 octets sont interprétés par l'affichage, et aussi étudier comment les codes sont affectés par l'échange de registres comme 'RCL' ou 'STO'.

5A. AFFICHAGES INCONVENANTS

Quand le processeur se trouve avec une séquence d'octets à afficher,(PRGM off),il doit d'abord décider d'afficher les octets comme un nombre ou une chaine de caractères alpha. Si le mode alpha est mis, ou si 'AVIEW' est éxécuté, il n'y pas de choix: tous les octets du registre alpha sont affichés comme caractères. Mais si le mode alpha est off, donc que l'affichage montre un registre de données, le choix est déterminé par la valeur du digit de signe de la mantisse. Nous savons déjà que si ce digit est '0' ou '9' (1001), le contenu du registre est affiché respectivement comme un nombre positif ou négatif. Le seul autre digit 'normal' est '1', pour lequel le registre est supposé contenir une chaine alpha de six caractères, comme celle résultant des opérations 'ASTO'. Chacun des six caractères restants du registre est affiché comme un caractère. Le premier octet n'est pas montré. Tout autre digit de signe que '0', '1' ou '9' fait que le contenu du registre est affiché comme un nombre négatif, mais traité comme une donnée alpha dans de nombreuses opérations arithmétiques.
Quand le registre affiché contient des données alpha, comme indiqué par le digit de signe '1', l'affichage est simplifié par la suppression des octets nuls. Tous les octets nuls, au lieu de conduire à des nuls comme dans les affichages alpha, sont non seulement effcés mais aussi éliminés par la montée des caractères adjacents non nuls dans leur positions. Par exemple, la donnée alpha codée '10 00 41 00 00 42 00', qui serait montrée comme "A⁻⁻B⁻" si elle était visualisée dans un registre alpha par un 'ARCL', sera affichée comme "AB" dans le registre X.

Les nombres affichés ont la particularité supplémentaire du choix de l'utilisateur quant au nombre de chiffres qui seront montrés, avec l'arrondi approprié, utilisant les fonctions de format 'FIX', 'ENG' et 'SCI'. Nous devons nous souvenir que ce choix ne porte que sur l'affichage, le nombre est toujours stocké comme 7 octets pleins.
Dans beaucoup de cas les nombres contenant les digits 'A' à 'F' seront affichés avec les caractères décimaux ordinaires '0' à '9'. Comme A-F est plus grand que 9, chacun de ces digits de la mantisse d'un nombre amène un '1' dans le digit de gauche suivant. Par exemple, les octets '01 03 B0 0F 00 00 00', qui devraient être le nombre '1.03B00F', seront affichés en FIX 6 sous le format '1.041015'. Le digit 'F' montré comme '15' occupe deux positions. Le 'B' est devenu un '11', mais comme il y avait déjà un 3 dans le digit suivant, la combinaison '3B' a donné '41'.
Une exeption à cette règle générale apparait quand le nombre affiché a un exposant non négatif, et est affiché dans un format qui montre tous les 10 chiffres de la mantisse. Ceci ne peut être réalisé que par le format FIX affichant des nombres dont l'exposant est entre 00 et 09. Si nous mettons en format FIX 9, le nombre '1.03B00F' sera affiché comme '1.03;00?000'. Les digits B et F sont respectivement représentés par les caractères seuls ";" et "?". Ces caractères sont dans la ligne 3 de la table d'octets, de le même façon que les caractères des nombres décimaux "0" à "9". De même, les digits 'C', 'D', et 'E' sont affichés comme les caractères "<", "=" et ">", respectivement. Un digit 'A' est représenté par le caractère plein dessiné dans le coin inférieur gauche de la case 3A. Si un nombre est copié dans le registre alpha par 'ARCL', les caractères spéciaux sont conservés, sauf le caractère plein qui change pour le caractère alpha ":".
Les nombres avec un exposant nul comme '1.03B00F' sont affichés avec dix chiffres pour la mantisse en format FIX 9 uniquement. Si le nombre était '10.3B00F' ('1.03B00F E01'), l'affichage spécial aboutirait en FIX 8 comme en FIX 9. En général, nous obtenons les caractères spéciaux en format FIX 'n' à FIX 9, où 'n' est 9 moins l'exposant. Cependant, quand l'exposant lui-même contient un des digits A-F, l'exposant sera affiché en utilisant les caractères spéciaux (mais non la mantisse, puisque 10 chiffres de mantisse ne peuvent être affichés avec un exposant).

5B. ECHANGES DE REGISTRES ET NORMALISATION

En nous préparant à travailler avec des registres de données contenant des codes arbitraires à 7 octets, introduisons la classification de codes suivante: premiérement, une 'chaine de donnée alpha' est un code de 7 octets dont le premier nybble est '0001'. Un nombre est un code pour lequel le premier et le douzième nybble sont '0000' ou '1001', et les autres digits sont un des chiffres décimaux de '0' à '9'. Tout autre code sera appelé un 'nombre non normalisé' ou plus simplement un 'NNN'.
Que cette classification soit utile découle de la considération du traitement des codes à 7 octets par les fonctions d'échange de registres 'STO', 'RCL', 'X<>' et 'VIEW'. 'STO' est le plus direct. STO mn, où 'mn' se réfère à n'importe quel registre adressé directement ou indirectement copie le contenu du registre X dans le registre désigné. Malheureusement pour la programmation synthétique, les trois autres fonctions ne sont pas si simples. RCL pq, X<>pq ou VIEW pq, si 'pq' se réfère à un registre de données numériques, provoque la normalisation du nombre avant sa copie. Les échanges entre les registres d'état ne provoquent pas de normalisation. Par 'normalisation', nous voulons dire que les NNN sont changés, soit en nombres décimaux normaux ne contenant pas de digits hérétiques plus grands que 9 si le signe original de la mantisse était 0 ou 9, ou en une donnée alpha avec un digit de signe de 1 si le digit de signe original était autre que 0 et 9. Par exemple, un NNN codé par les octets '01 0C 00 D0 0E 00 FF', qui s'affiche comme '1.1201301 E??', est normalisé à la suite de 'STO 01' et 'RCL 01' en '1.120130140 E-35'. Les octets se sont en fait changés en '01 12 01 30 14 09 65'. Le NNN '21 0C 42 34 7E 40 DD', qui a un digit de signe anormal, s'affiche comme '-1.1242348 E==', S'il est normalisé, il se changera en l'alpha "µB4Σ†", i.e., les octets '11 0C 42 34 7E 40 DD'.
'ASTO' et 'ARCL' éxécutent différentes sortes d'échanges de registres. 'ASTO' prend les 6 premiers octets dans le registre alpha, commençant au premier octet non nul, ajoute un identificateur alpha '10' devant pour faire un total de 7 octets, puis stocke les octets résultants dans le registre adressé. 'ARCL' renverse le processus si le registre adressé contient une donnée alpha, jetant l'octet '10' et concaténant les caractères alpha à la droite de la chaine existant dans le registre alpha. Les nuls de tête dans la donnée alpha sont perdus; les nuls de queue ou intermédiaires sont conservés. Par exemple, commencez avec "ABC" en alpha. Puis 'ARCL X', où X contient '10 44 45 46 47 48 49' ("DEFGHI"), donnera "ABCDEFGHI". Si le registre Y contient '10 00 4A 00 00 4B 00' (affiché comme "JK"), ARCL Y produit la chaine "ABCDEFGHIJ⁻⁻K⁻" dans le registre alpha. Enfin, si le registre adressé par 'ARCL' contient un nombre ou un NNN, l'opération ne copie pas les octets du nombre en alpha, mais change chaque digit, comme montré dans l'affichage d'un nombre, en son correspondant de la ligne 3 pour stockage dans le registre alpha. 'ARCL' normalise aussi le contenu du registre adressé.

5C. DEMARONS :"CODE"

Le sauteur d'octets est jusqu'ici le seul outil que nous avons pour la création de lignes de programme non standard et de NNN (depuis 'RCL M', etc). Cependant, le sauteur d'octets est strictement une opération manuelle, et a quelques limitations dans les codes d'octets qu'il peut générer (les octets de la moitié inférieure de la table d'octets sont difficiles à manier). Nous allons maintenant écrire un programme appelé "CODE" qui générera automatiquement **n'importe quel** code à 7 octets spécifié par l'utilisateur, plaçant le code à la fois dans les registres X et M. Ce programme, avec l'aide d'une routine "REG" (qui permet le stockage dans n'importe quel registre du code en question), va permettre la génération de n'i mporte quelle séquence programme, d'un NNN, ou d'une chaine alpha non standard. Bien plus, nous pourrons faire des assignations de touches synthétiques, qui assignent des fonctions synthétiques aux touches utilisateurs, ainsi que les fonctions à deux octets. Cette capacité placera finalement les fonctions synthétiques sur le même pied que les fonctions normales de la HP 41C ou de ses périphériques: elles seront donc capables d'être éxécutées manuellement ou insérées dans un programme avec une seule séquence de touches.

"CODE" est fait pour satisfaire à de nombreuses utilisations. Pendant l'éxécution, le programme doit s'arrêter et demander à l'utilisateur d'entrer un code spécifiant un nombre de 14 digits héxadécimaux. Aprés cela, le programme doit tourner sans autre intervention de l'utilisateur, donnant le nombre désiré codé avec les octets appropriés. Bien que le registre des drapeaux soit utilisé pour la création des octets, le programme doit retourner l'état original des drapeaux à la fin de l'éxécution. Enfin, pour des raisons qui seront éclaircies plus tard,(section 6G), "CODE" ne doit pas utiliser de registres de données. Cet ensemble de contraintes est assez difficile à mettre en oeuvre ; la version de "CODE" qui est décrite ici est celle maintes fois révisée depuis l'originale (voir 'HP 41C BLACK BOX PROGRAMS', PPC calculator journal, V6 N8 P27).

```
01*LBL "CODE"      24 CHS      46 SF 15      68 RCL \
02 "CODE=?"        25 FS? 06   47 FS? 15     69 RCL [
03 AON             26 CHS      48 CHS        70 STO \
04 STOP            27 SF 06    49 FS? 14     71 R↑
05 AOFF            28 X<0?     50 CHS        72 STO [
06*LBL "CO"        29 CF 06    51 SF 14      73 "⊢*"
07 "⊢ABCDEFG"      30 X<0?     52 X<0?       74 X<> \
08 .066            31 SF 05    53 CF 14      75 R↑
09 STO L           32*LBL 01   54 X<0?       76 STO ]
10*LBL "HB"        33 ST+ X    55 SF 13      77 R↑
11 1               34 FS?C 04  56*LBL 01     78 STO \
12 RCL ]           35 SF 00    57 FS?  12    79 "⊢**"
13 X<> d           36 FS?C 05  58 SF 04      80 RCL Z
14 X<>Y            37 SF 01    59 FS?  13    81 STO [
15 CF 00           38 FS?C 06  60 SF 05      82 ISG L
16 CF 01           39 SF 02    61 FS?  14    83 GTO "HB"
17 CF 02           40 FS?C 07  62 SF 06      84 RCL [
18 FS?C 03         41 SF 03    63 FS?  15    85 CLA
19 GTO 01          42 FS?C 11  64 SF 07      86 STO [
20 SF 04           43 GTO 01   65 RDN         87 AVIEW
21 FC?C 07         44 SF 12    66 X<> d      88 TONE 9
22 SF 07           45 FC?C 15  67 RCL ]      89 END
23 FS? 07
```

"CODE"
191 octets

Instructions pour utiliser "CODE":

1.XEQ "CODE"
2.Au prompt "CODE=?", entrez 14 caractères alpha pour représenter le code désiré en utilisant les caractères "0" à "9" et "A" à "F" pour les nybbles '0' à 'F'.
3. R/S
4.A la tonalité, le code demandé est à la fois dans les registres X et M (montré avec 'AVIEW').

Pour voir si votre version de "CODE" est bonne, essayez ces exemples:

ENTREES	SORTIES (Aview)
"41 42 43 44 45 46 47"	"ABCEDFG"
"00 01 28 29 00 7F 7E"	"⅄()⁻⊢Σ"

(Le reste de cette section est une discussion détaillée sur l'opération de "CODE", et peut être shuntée en première lecture).

La tâche de "CODE" est de convertir 4 caractères, entrés par l'utilisateur aprés la halte de la ligne 04, en octets correspondants. (Les labesl globaux "C0" et "HB" sont là pour être utilisés par d'autres programmes qui utilisent des parties de "CODE" comme sous-routines.) Pour optimiser la longueur de programme, cette tâche est dirigée par une routine qui convertit une paire des caractères entrés en un octet sorti. Cette routine est appelée 7 fois, utilisant le registre alpha pour concaténer les octets de sortie ensemble. Le problème est compliqué par le fait que nous ne devons pas utiliser de registres de données, ce qui ne laisse que les registres alpha et la pile pour jongler avec le code entré, le code de sortie, le contenu original du registre des drapeaux, et toute l'arithmétique qui peut être nécessaire pour les conversions.
La conversion de base est dans les lignes 10-64. Pour comprendre comment ça marche, commençons par supposer qu'une paire de caractères entrés a été déplacée dans les deux premiers octets de d. Par exemple, prenons les caractères "49", qui sont à changer pour le seul octet '49'. Les caractères initiaux sont en fait les octets '34 39'. Ce que nous devons faire est d'ignorer les '3' et d'amener le '4' dans le premier nybble, et le '9' dans le second:

34 39 ⟶ 49 39

aprés quoi nous nous occuperons seulement du premier des deux octets. Les mouvements des bits sont opérés par d'ordinaires opérations sur les drapeaux. Rappelons que dans le registre d les quatre premiers bits sont les drapeaux 0-3, les quatre suivants sont les drapeaux 4-7, etc. Les lignes programme pour éxécuter la copie sont comme suit (à la droite de chaque ligne est montré l'effet de l'éxécution de la ligne sur les 16 premiers bits du registre d):

Ligne programme	Drapeaux 00-15
(valeur initiale'34 39')	0011 0100 0011 1001
15 CF 00	"
16 CF 01	"
17 CF 02	0001 0100 0011 1001
18 FS?C 03	0000 0100 0011 1001
19 GTO 01	
32 LBL 01	
34 FS?C 04	"
35 SF 00	"
36 FS?C 05	0000 0000 0011 1001
37 SF 01	0100 0000 0011 1001
38 FS?C 06	"
39 SF 02	"
40 FS?C 07	"
41 SF 03	"
42 FS? 11	"
43 GTO 01	"
56 LBL 01	
57 FS? 12	"
58 SF 04	0100 1000 0011 1001
59 FS? 13	"
60 SF 05	"
61 FS?14	"
62 SF 06	"
63 FS?15	"
64 SF 07	0100 1000 0011 1001 = 49 39

La simplicité de la précédente série de lignes provient du fait que le premier nybble de chaque caractère "0" à "9" est le même que l'équivalent numérique du caractère. Malheureusement, ce n'est pas vrai pour les caractères "A" à "F", qui demandent un processus de conversion plus compliqué. Le programme doit tester chaque caractère entré pour déterminer s'il est 'plus grand' que "9", et donc nécessite un processus supplémentaire. Le test est réalisé pour le premier de la paire des caractères entrés à la ligne 18. La conversion additionnelle est réalisée aux lignes 20-31 (et utilise la ligne 11). Le second caractère de la paire est testé à la ligne 42; les lignes 33 et 44-55 provoquent la conversion correspondante.
Le reste de "CODE" est centré sur la routine de base décrite ci-aprés: le programme doit placer deux octets des caractères entrés dans les deux premiers octets du registre d, lancer la rou-

tine, puis extrait un octet de sortie depuis le registre d. Les registres N et O sont utilisés pour stocker les caractères d'entrée; le registre M contient le code en cours d'expansion. Vous pouvez trouver intéressant d'éxécuter les lignes 66-81 pas à pas pour voir comment l'octet de tête du registre d est concaténé au dernier caractère du registre M, pendant que le code d'entrée de l'utilisateur est décalé de deux positions vers la gauche dans les registres N et O. Aprés 7 itérations (le registre L sert de compteur) M contient les octets finaux de sortie. Quand les caractères d'entrée sont stockés dans le registre d (ligne 13), de nombreux drapeaux système sont levés ou baissés (la moitié de l'aspect humoristique de "CODE" est de pouvoir regarder les différents drapeaux clignoter), y compris le drapeau 52, le drapeau du mode programme. Si ce drapeau est levé pendant qu'un programme tourne, certaines opérations feront que la HP 41C se programmera elle-même! (voir section 7B). Parmi ces opérations sont les lignes d'entrée de nombres ordinaires, donc pour éviter de pareilles catastrophes, l'entrée d'un '1' à la ligne 11 précédant la ligne '13 X<>d', nécessite l'inclusion inutile de la ligne '14 X<>Y'.

5D. ACCES DIRECT AUX REGISTRES PROGRAMME

Le programme "CODE" développé dans la dernière section est un outil puissant de programmation synthétique, mais jusqu'ici nous ne pouvons pas faire grand chose avec,au delà de créer des caractères non entrables au clavier, puisque les codes de sortie ne pouvaient qu'être transférés dans d'autres registres de données. Mais considérons ceci: la division de la mémoire de la HP 41C en registres de données et de programme est entiérement controlée par un seul nombre, qui est l'adresse absolue de Roo, stockée dans le registre c. Changer cette adresse est fait normalement par la fonction 'SIZE', qui déplace aussi le contenu des registres de mémoire pour que les programmes restent les programmes et que les données restent les données. Comme des programmeurs ambitieux, nous n'allons pas laisser ce détail mineur nous arrêter. Les fonctions synthétiques nous donnent accés au registre c, et maintenant "CODE" nous permet de créer n'importe quel octet(s) que nous pouvons vouloir y mettre. En plaçant le code approprié dans le registre c, nous pouvons déplacer le 'rideau' séparant les registres de programme et de données à l'endroit que nous voulons sans utiliser 'SIZE', et de là transformer des lignes programme en registres de données et vice-versa!
Bien plus, pour maintenir le niveau, nous écrirons un programme pour assurer tout le processus de stockage automatique dans les registres de programme. L'approche générale est la suivante: nous utilisons "CODE" deux fois, une fois pour créer une valeur temporaire à stocker dans le registre c qui spécifiera comme Roo le registre programme dans lequel nous voulons stocker; et une nouvelle fois, pour créer le code spécial à stocker. Puis nous échangeons la nouvelle valeur de c avec l'ancienne, en utilisant 'X<>c', nous éxécutons 'STO 00' avec le code spécial en X, puis nous réstaurons le contenu original de c pour que l'accés aux programmes et aux données existants soit préservé.
La valeur spécifique que nous voulons stocker dans le registre c est '10 00 01 69 xy z1 00', où 'xyz' est l'adresse absolue sur 3 digits du registre de programme auquel nous voulons accéder. Le '169' est la constante de départ à froid nécessaire pour prévenir un MEMORY LOST. '100' est choisi pour simplifier à la fois les adresses de 'ΣREG' et du .END.Puisque cette valeur du registre c est seulement temporaire, les valeurs que nous utilisons ne sont pas tellement importantes Il est commode d'avoir l'octet '10' pour commencer le code puisqu'il rend la chaine 'donnée alpha', qui peut donc être stockée et rappelée sans normalisation. La seule variable dans la nouvelle valeur du registre c sont les digits 'xyz', nous pouvons donc minimiser le temps d'éxécution en n'utilisant "CODE" que pour créer deux octets ('xy z1' dans ce cas) plutôt que 7. Le programme suivant, 'REG', est la mise en œuvre de toutes ces idées.
Instructions pour "REG":

1. XEQ "REG". (si vous ne voulez que rappeler le contenu d'un registre, XEQ "RREG".)
2. Au prompt "REG?", entrez trois caractères alpha pour identifier l'adresse absolue d'un registre, puis R/S.
3. Au prompt "CODE=?", entrez 14 caractères alpha pour spécifier le code à stocker, puis R/S.
4. L'affichage "REG-abc" (abc est l'adresse du registre) indique que l'éxécution du programme est terminée.

```
01*LBL "RREG"        14 STO \             26 X<> c
02 SF 10             15 "┣┼┼┼┼┼┼8┼┼┫"    27 RCL [
03 GTO 01            16 .001              28 X<> 00
04*LBL "REG"         17 STO L             29 FS?C 10
05 CF 10             18 XEQ "HB"          30 STO 00
06*LBL 01            19 "┣┼"              31 X<>Y
07 "REG?"            20 RCL [             32 STO c
08 AON               21 STO 03            33 X<>Y
09 STOP              22 CLD               34 "REG-"
10 AOFF              23 FC? 10            35 ARCL 01
11 ASTO 01           24 XEQ "CODE"        36 AVIEW
12 "┣┫"              25 RCL 00            37 END
13 RCL [
```

"REG"

101 OCTETS

SIZE 002

La ligne 15 de "REG" est une ligne synthétique de programme, codée 'FB 7F 00 00 00 00 00 00 10 00 01 69'. Elle peut être créée par le sauteur d'octets comme suit:

```
15 STO 07
16 STO 02
17 "┣ABCDEFGHIJ"
                                JUMP .016          (16 X>Y?       )
17 0
18 /
19 LBL 00
20 FRC
                                JUMP .016          (X>Y?          )
                                SST,SST            (18 /          )
                                DEL 001            (17 STO 02     )
                                JUMP .017          (17 -          )
                                DEL 005            (16 STO 02     )
                                GTO .018           (18 X<=Y?      )
                                DEL 005            (17 "----------▯天▯ ")
                                GTO .015           (15 STO 07     )
                                DEL 002            (14 STO N      )
```

Le label "RREG" est donné au cas où vous ne voudriez que rappeler le contenu d'un registre. Rappelez vous, toutefois, que le registre rappelé sera normalisé par le rappel.
A titre d'exemple pour l'utilisation de "REG", éxécutez 'SIZE 010' (si aucun module n'est connecté, éxécutez 'SIZE 074' avec un module, 'SIZE 138' pour deux, 'SIZE 202' pour trois, 'SIZE 266' pour quatre). Maintenant 'CAT 1' pour placer le pointeur sur le premier programme en mémoire, puis pressez 'RTN' pour mettre le pointeur au début du programme. L'adresse du premier registre de programme pour cette configuration est '0F5'. En mode PRGM, entrez sept lignes 'ENTER', qui remplissent juste le registre 0F5, renvoyant les programmes existants plus bas dans la mémoire. Pour créer la ligne de texte synthétique "######":

```
                                XEQ "REG"          ("REG?"        )
"0F5"
                                R/S                ("CODE ?"      )
"F6232323232323"
                                R/S                ("REG-0F5"     )
```

GTO le premier programme (utilisez CAT 1)
GTO .001
PRGM (on) (01 "######")

Si vous répétez la séquence précédente en remplaçant "F6232323232323" par "01 91 75 9F 0A 96 76" les lignes programme suivantes seront placées en haut de la mémoire programme:

```
01 LBL 00
02 STO M
03 TONE 0
04 ISG N
```

Le TONE 0 de la ligne 03 est en fait 'TONE 10', voir section 7A.
N'importe quelle ligne programme peut être créée de cette manière. Pour faire une ligne demandant plus de 7 octets, nous utiliserons simplement "REG" deux fois (ou même trois), en stockant 7 octets du code chaque fois dans des registres adjacents.

5E. ASSIGNATIONS DE TOUCHES SYNTHETIQUES

Une puissant application de "REG" est la génération des assignations synthétiques de touches, i.e., assigner des fonctions synthétiques à deux octets à des touches utilisateur pour qu'elles puissent être éxécutées manuellement ou entrées directement dans un programme. Pour mener à bien ces assignations, nous utilisons simplement "REG" pour stocker des codes spéciaux dans les registres d'assignation de touches. Chaque utilisation de ce type de "REG" peut assigner deux touches utilisateur. Le procédé est bien illustré par un exemple: nous allons assigner les fonctions 'RCL M' et 'STO M' aux touches 'TAN' (25) et 'ATAN' (-25), respectivement.

1. Effacer toutes les assignations éxistantes sauf le sauteur d'octets.(Ce pas draconien n'est pas toujours nécessaire, mais vous devez le faire cette fois ci). 'PACK', puis assignez n'importe quelle fonction à n'importe quelle touche.

2. Assigner n'importe quelle fonction aux deux touches qui doivent être assignées (pour cet exemple, assignez les touches 'TAN' et 'ATAN'). Ceci met un 'code de paille' dans le registre 0C0, le plus bas des registres d'assignation, et lève les drapeaux appropriés dans les registres e et F.

3. Déterminez le code nécessaire pour prendre la place du code de paille dans le registre 0C0. Ce code suit le format décrit dans la section 2E. Les octets de code des fonctions à assigner peuvent être trouvés dans la table d'octets; les octets d'assignation de touche sont montrés dans la figure 2-6. Dans ce cas, les codes sont:

```
F0          commence le registre d'assignation
90 75       'RCL M'
42          assignation de la touche 'TAN'
91 75       'STO M'
4A          assignation de la touche 'ATAN'
```

4. Utilisez "CODE" pour stocker le code du registre d'assignation dans le registre 0C0. Pour cet exemple, nous entrons "0C0" au prompt "REG?", et "F090754291754A" au prompt "CODE=?". Aprés l'affichage "REG-0C0", nous verrons que presser sur la touche 'TAN' éxécute la fonction 'RCL M'; la touche 'ATAN' éxécute la fonction 'STO M'.

Cette technique d'assignation synthétique n'est en aucune façon limité aux fonctions synthétiques. Le principe des assignations de touche est de permettre à l'utilisateur de remplacer des séquences de touches souvent utilisées par une seule pression de touche. Ordinairement, une seule instruction comme 'ST+IND X' demande 5 pressions de touche; le mieux que nous pouvons faire normalement est de réduire ce chiffre à 4 en assignant 'ST+' à une touche utilisateur. Mais avec les techniques d'assignation synthétique, il n'y a pas de raison de ne pas assigner toute la fonction 'ST+IND X' à une touche (le code de fonction sera dans ce cas '92 F3').

En plus, les assignations synthétiques ne sont pas limitées aux fonctions de la HP 41C; nous pouvons aussi assigner des fonctions de périphériques à des touches. Nous obtenons simplement le code 'XROM' des fonctions désirées depuis le manuel du périphérique, en convertissant le code XROM en héxa en utilisant la conversion décrite dans la section 2B, puis en stockant les octets de code résultants dans un registre d'assignation. Cette méthode permet à un utilisateur d'entrer des programmes contenant des fonctions de périphériques en mémoire même quand le périphérique n'est pas disponible. Nous pouvons aussi entrer des 'fonctions non programmables' dans des programmes (comme 'LIST' ou 'WALL'). Les codes pour les fonctions non programmables du lecteur de cartes et de l'imprimante sont montrés dans la table 5-1.

Quand une fonction normale de la HP 41C est assignée à une touche, un seul des deux octets de fonction dans un demi registre d'assignation est utilisé pour identifier la fonction. Le premier de ces deux octets est toujours '04' (LBL 03). Si le premier octet est autre que '04', le processeur sait que l'assignation correspond à une fonction de périphérique à deux octets. Si le périphérique est absent, presser sur la touche produit le code XROM approprié. Dans le cas d'assignation synthétique à deux octets, le premier octet de fonction est différent de '04', avec de nouveau le résultat que presser et tenir la touche affiche le code XROM. Dans notre exemple d'assigner 'RCL M' à la touche 'TAN', presser la touche TAN affiche XROM 01,53. Relacher la touche avant que le 'NULL' n'apparaisse fait éxécuter le RCL M

Les codes synthétiques XROM peuvent être déchiffrés de la même manière que les XROM normaux, comme décrit dans la section 2B. Par exemple, pour déterminer le code XROM correspondant à 'RCL M', nous écrivons le code héxa de RCL M en binaire, puis nous groupons les 12 derniers bits en deux nombres de 6 bits, que nous convertissons en décimal:

```
héxa          9     0      7     5
binaire    1001 / 0000   01/11  0101/
décimal              01       53
```

De ce fait RCL M=XROM 01,53. De la même manière STO M=XROM 05,53.
Pour faciliter la construction d'une large série d'assignations de touches, il est commode d'automatiser la procédure d'assignation synthétique plus que cela n'est possible avec l'utilisation de "REG". L'utilisation du programme d'assignation de touches, 'KA', listé ci-aprés, élimine les assignations manuelles préliminaires, l'utilisation de la figure 2-6 pour déterminer les codes de touches, et tout ce qui nécessite de s'inquiéter du contenu courant des registres d'assignation. "KA" refusera de recouvrir une assignation déjà existante sauf si l'utilisateur efface cette assignation quand il est demandé.
Trois routines utilitaires sont listées avec "KA"."KP" compacte les registres d'assignation. Bien que deux assignations de touches ne prennent qu'un registre d'assignation, effacer deux assignations ne fera pas récupérer un registre pour utilisation ultérieure sauf si les deux assignations ont été faites dans le même registre. Il est donc possible d'avoir un certain nombre de registres d'assignations à demi remplis. "KP" compresse les codes dans les registres d'assignation, laissant au plus un demi registre vide quand il y a un nombre impair d'assignations actives. Le registre à demi rempli sera le registre OCO, pour qu'une nouvelle assignation remplisse ce registre.
Le programme d'effacement d'assignations "CA" efface automatiquement toutes les assignations de fonctions et de programmes utilisateurs. Cepndant, "CA" n'efface pas les octets d'assignation dans les labels globaux, de sorte que, si un programme est assigné à une touche qui était assignée avant à un autre programme plus bas en mémoire puis effacée par "CA", ce premier état sera réactivé (la première assignation).
"EF" ou detecteur de fin, est une routine utilisée par "KA", "CA", et "KP" pour localiser le .END. Si 'xyz' est le nombre de registres séparant le registre OCO et le .END., alors "EF" place le nombre '0.xyz' dans le registre X, pour contrôler les itérations comprenant le rappel de chacun des registres d'assignations à tour de rôle. La ligne 31 de "EF" place le code 'F0 00 00 00 0C 0C 0C' dans le registre M. Ce NNN est stocké par les autres programmes dans le registre pour que le registre OCO devienne temporairement le registre R00. Quand cette valeur est dans l le registre c, un arrêt de programme causera un MEMORY LOST. Pour éviter ceci, ne stoppez pas l'éxécution de "KA", "CA" ou "KP" quand ils tournent. Bien plus, assurez vous toujours que aucun de ces programmes n'est le premier fichier en mémoire. Il doit y avoir au moins un END précédant les routines pour que leurs GTO sautant en arrière fonctionnent correctement. Quand un GTO provoque un saut vers un label plus haut en mémoire, la recherche du label est faite en descendant jusqu'à la fin du fichier, puis reprend au début jusqu'à ce que le label soit trouvé. Quand le rideau entre les données et les programmes est déplacé sur le registre OCO, la recherche commencera dans les registres de données sauf s'il y a un END dans un fichier programme plus haut dans la mémoire.

TABLE 5-1

Fonctions non programmables de périphériques

Fonction	Numéro XROM	Code octet	Exécution ?
CARD READER	30,00	A7 80	Efface les drapeaux de bits
VER	30,05	A7 85	Normal
WALL	30,06	A7 86	Normal
WPRV	30,09	A7 89	Normal
-PRINTER-	29,00	A7,40	Crash
LIST	29,07	A7 47	Liste depuis la ligne suivante
PRP	29,13	A7 AD	NONEXISTENT

Instructions pour l'utilisation de "KA" (ne doit pas être le premier programme en mémoire). Pour faire 1 ou deux assignations:
1. XEQ "KA". L'affichage montrera 'NONEXISTENT' si aucun registre n'est disponible pour les assignations. NE PAS ESSAYER DE STOPPER L'EXECUTION. UNE FOIS COMMENCEE, FAIRE AU MOINS UNE ASSIGNATION.
2. Au prompt "PRE↑POST↑KEY", entrez trois nombres:

'préfixe', 'enter↑', 'postfixe', 'ENTER↑', 'code de touche', R/S

Le 'préfixe' et le 'postfixe' sont les valeurs décimales des octets de préfixe et de postfixe de la fonction à assigner, qui peuvent être déterminés à partir de la table d'octets. Par exemple, la fonction 'STO N' (91 76) sera entrée avec le préfixe '145', et le postfixe 118. Le code de touche est le même code ligne-colonne qui est affiché pendant les assignations ordinaires. Ainsi la touche 1/X a comme code de touche '12', la touche FS?, -54, etc. Les touches CHS, EEX

```
01♦LBL "KA"
02 XEQ "EF"
03 CF 00
04 CF 01
05 SF 03
06 RCL [
07 ENTER↑
08 X<> c
09 X<>Y
10 X<> 00
11♦LBL 00
12 ""
13 X<> [
14 "├♦"
15 X<> \
16 X=0?
17 GTO 01
18 "├-------"
19 X<> \
20 ISG Z
21 GTO 03
22 STO 00
23 RDN
24 STO c
25 GTO 15
26♦LBL 03
27 X<> IND Z
28 GTO 00
29♦LBL 01
30 RDN
31 STO c
32 CLA
33♦LBL 05
34 CF 22
35 ASTO L
36 "PRE↑POST↑KEY"
37 TONE 8
38 PROMPT
39 CLA
40 ARCL L
41 FC? 22
42 GTO 06
43 X<> Z
44 XEQ 02
45 XEQ 02
46 36
47 STO a
48 RDN
49 ENTER↑
50 X<0?
51 SF 00
52 ABS
53 .1
54 *
55 LASTX
56 -
57 INT
58 ST- a
59 LASTX
60 FRC
61 00
62 *
63 ST- a
64 2
65 *
66 +
67 8
68 FC? 00
69 CLX
70 +
71 X<> a
72 24
73 X<=Y?
74 SF 01
75 FC? 01
76 CLX
77 -
78 FS? 00
79 RCL e
80 FC? 00
81 RCL '
82 ASTO L
83 STO [
84 FS? 01
85 "├♦♦♦"
86 X<> [
87 X<> d
88 FC? IND Y
89 GTO 01
90 X<> d
91 RCL \
92 FIX 0
93 "CLEAR "
94 ARCL T
95 TONE 4
96 PROMPT
97 STO \
98 RDN
99 X<> d
100♦LBL 01
101 SF IND Y
102 X<> d
103 STO [
104 "├♦♦♦♦"
105 FC?C 01
106 "├♦♦♦"
107 RCL \
108 FS? 00
109 STO e
110 FC?C 00
111 STO '
112 CLA
113 ARCL L
114 RCL a
115 XEQ 02
116 FS?C 03
117 GTO 05
118♦LBL 06
119 ASTO X
120 ""
121 FC?C 22
122 "├♦♦♦"
123 ARCL X
124 RCL [
125 "µµµ"
126 RCL [
127 X<> c
128 X<>Y
129 STO 00
130 X<>Y
131 X<> c
132 "DONE"
133 BEEP
134 PROMPT
135♦LBL 02
136 X=0?
137 "├♦♦♦♦"
138 OCT
139  E3
140 /
141 10
142 +
143 X<> d
144 FS?C 19
145 SF 20
146 FS?C 18
147 SF 19
148 FS?C 17
149 SF 18
150 FS?C 15
151 SF 17
152 FS?C 14
153 SF 16
154 CF 07
155 SF 03
156 X<> d
157 ARCL X
158 "├ABC"
159 0
160 X<> \
161 STO [
162 RDN
163 RDN
164 END
```

"KA"

334 OCTETS

```
01♦LBL "EF"
02 RCL c
03 STO [
04 "├♦♦♦♦X"
05 RCL [
06 X<> d
07 CF 00
08 CF 01
09 CF 02
10 CF 03
11 FS?C 07
12 SF 05
13 FS?C 08
14 SF 06
15 FS?C 09
16 SF 07
17 FS?C 10
18 SF 09
19 FS?C 11
20 SF 10
21 FS?C 12
22 SF 11
23 X<> d
24 DEC
25 193
26 -
27 X<0?
28 GTO 15
29 1 E3
30 /
31 "♦♦♦µµµ"
32 END
```

"EF"

79 OCTETS

et '⌨' (ainsi que leurs correspondantes shiftées) doivent être entrées avec les codes de touche (négatifs pour les touches shiftées) '43', '44', '45', respectivement, comme si la touche 'ENTER↑' couvrait une touche imaginaire '42'.
3. Si la touche désignée dans l'étape 2 est déjà assignée, le TONE 4 retentira, et l'affichage montrera "CLEAR nm", où nm est le code de touche. L'utilisateur doit manuellement effacer l'assignation puis appuyer sur R/S pour continuer.
4. L'étape 2 sera répétée automatiquement pour la seconde assignation. Si une seule assignation est désirée, pressez R/S au prompt. A la fin de "KA", le BEEP tonera et "DONE" sera affiché. Pour d'autres assignations, reprendre l'étape 1.

Instructions pour "CA":
1. XEQ "CA". NE PAS ARRETER L'EXECUTION. 'NONEXISTENT" sera affiché si aucun registre de programme n'est disponible pour les assignations de touches.

```
01*LBL "CA"
02 0
03 STO e
04 STO '
05 XEQ "EF"
06 X<> [
07 X<> c
08*LBL 00
09 0
10 ENTER↑
11 X<> IND [
12 X=Y?
13 GTO 01
14 RCL Z
15 ISG [
16 GTO 00
17 RDN
18*LBL 01
19 RCL Z
20 STO c
21 END
```

"CA"

42 OCTETS

Instructions pour "KP":
1. XEQ "KP". "NONEXISTENT" sera affiché si les registres d'assignations de touches sont pleins. NE PAS ARRETER L'EXECUTION.

```
01*LBL "KP"
02 XEQ "EF"
03 STO Y
04 RCL [
05 X<> c
06 ENTER↑
07 CLA
08 CF 03
09*LBL 14
10 FS?C 03
11 GTO 07
12 CLX
13 X<> IND Z
14 SF 25
15 X=0?
16 FS?C 25
17 GTO 07
18 ASTO L
19 CF 01
20 STO [
21 ASHF
22 X<> [
23 X=0?
24 SF 01
25 CLX
26 "├*"
27 STO \
28 "├**"
29 X<> \
30 "├*"
31 X<> \
32 X=0?
33 GTO 01
34 X<> \
35 ASTO [
36 "├*"
37 STO \
38 "├*"
39 X<> \
40*LBL 01
41 STO \
42 FC?C 01
43 "├***"
44 X<> \
45 CLA
46 STO [
47*LBL 09
48 ASTO X
49 CLA
50 ARCL L
51 ARCL X
52 SF 03
53 ISG Z
54 CF 03
55 GTO 05
56*LBL 07
57 X<> [
58 XEQ 00
59 X<> 00
60 SIGN
61 X≠0?
62 GTO 03
63 CLX
64 LASTX
65 XEQ 00
66 STO IND T
67*LBL 03
68 RDN
69 STO c
70 TONE 9
71 RTN
72*LBL 05
73 X<> \
74 STO \
75 X=0?
76 GTO 14
77 ASTO X
78 ASHF
79 ASTO L
80 ""
81 ARCL X
82 CLX
83 X<> [
84 STO IND T
85 ARCL L
86 ISG T
87 GTO 14
88 RTN
89*LBL 00
90 ""
91 X<> [
92 "├↑"
93 X<> \
94 "├↑↑↑↑↑↑"
95 X<> \
96 END
```

"KP"

190 OCTETS

Certaines lignes dans ces programmes d'assignation demandent des saut d'octets minutieux . Les procédures pas à pas pour entrer ces lignes sont listées ci-aprés.
Les lignes 12 et 120 de "KA", et les lignes 80 et 90 de "KP" ont toutes comme code octet 'F1 F0'. Pour entrer l'une d'elles, utilisez cette procédure:

```
12 STO 01
13 "BJ"
                    JUMP            (13*          )
14 GTO 07
15 STO IND T
                    PACK            (14 STO IND T )
                    JUMP .013       (13 *         )
14 "A"
                    GTO .014        (14 -         )
                    DEL 002         (13 "B■"      )
                    PACK
                    JUMP            (13 *         )
14 X  Y
                    GTO .015        (15 HMS-      )
                    DEL 001         (14 "■"       )
                    GTO .012        (12 STO 01    )
                    DEL 002         (11 LBL 00    )
```

Le code de la ligne 125 de "KA" est 'F3 0C 0C 0C':

```
125 STO 01
126 "BJ"
                    JUMP            (126 *          )
127 "ABC"
                    GTO .127        (127 -          )
                    DEL 004         (126 "B■"       )
127 LBL 11
128 LBL 11
129 LBL 11
                    JUMP .126       (126 *          )
127 X<>Y
                    GTO .125        (125 STO 01     )
                    DEL 002         (124 RCL M      )
```

Le code de la ligne 04 de "EF" est 'F6 7F 00 00 00 00 02':

```
04 STO 02
05 "⊢ABCDE"
                    JUMP            (05 -           )
                    SST 3 fois      (08 X<Y?        )
09 LBL 01
                    JUMP .005       (05 -           )
                    DEL 004         (04 STO 02      )
                    DEL 001         (03 STO M       )
                    GTO .005        (05 X>Y?        )
                    DEL 001         (04 "⊢----■"    )
```

Enfin, la ligne 31 de "EF" est 'F7 F0 00 00 00 0C 0C 0C':

```
31 STO 01
32 "BJ"
                    JUMP            (32 *           )
33 GTO 10
34 STO IND T
                    PACK            (33 STO IND T   )
                    JUMP .032       (32 *           )
33 "ABCDEFG"
                    GTO .033        (33 -           )
                    DEL 008         (32 "B■"        )
                    PACK
                    GTO .033        (33 ""          )
34 +
35 +
36 +
37 LBL 11
38 LBL 11
39 LBL 11
                    GTO .034        (34             )
                    DEL 003         (33 ""          )
                    JUMP .032       (32 *           )
33 X<>Y
                    GTO .031        (31 STO 01      )
                    DEL 002         (30 /           )
                    GTO .032        (32 HMS-        )
                    DEL 001         (31 "■---µµµ-" ))
```

A titre d'exemple de l'utilisation de "KA", utilisez le pour assigner les fonctions 'RCL b' (préfixe 144/ postfixe 124) et 'STO b' (145/124) à des touches. Ces fonctions nous permettent de bouger le pointeur d'adresses vers des emplacements en mémoire usuellement inaccessibles. Essayez ceci (suivez exactement ; entrer d'autres lignes pendant que le pointeur est dans les registres d'état causera un MEMORY LOST.) :

```
                    XEQ "CODE"          (CODE:?          )
"00000000000006"
                    R/S                 ("J"             )
                    ALPHA               ("J"             )
"ABCDEFGHIJ"
                    ALPHA               (0,000,000.000 )
                    STO b (utilisez la touche assignée)
                    PRGM, SST           (01 X<Y?         )
                    DEL 003             (00 REG ---      )
01 RCL 08
02 STO 15
03 RCL 09
                    PRGM (off)
                    ALPHA               (ABC(?)GHIJ      )
```

Vous avez en fait édité le registre alpha. Le STO b a envoyé le pointeur d'adresses à l'adresse 0006, le dernier octet du registre N. Le DEL 003 a effacé les trois premiers octets du registre M, que vous avez ensuite remplacés par les caractères "(?)" en insérant les lignes programme correspondantes. De la même manière, si vous changez les caractères dans le registre alpha, vous verrez de nouvelles lignes programme en mode PRGM.

5F. CREATION DE LIGNES PROGRAMME SYNTHETIQUES

Les lignes synthétiques de programme peuvent être groupées en quatre grandes catégories: (1) fonctions synthétiques à deux octets, généralement une combinaison d'un postfixe d'un registre d'état avec un préfixe normal; (2) les lignes de texte synthétiques, contenant au moins un caractère non entrable au clavier; (3) autres lignes multi-octets non standards, principalement les labels globaux, les GTO et les XEQ, où le nom du label contient des caractères non entrables; (4) lignes 'En', où une ligne normale '1 En' d'une puissance de dix comme '1 E3' est raccourcie à 'En' en enlevant l'octet '1' superflu. De ces quatre types, le 1 et le 2 sont les plus fréquents. Les lignes de type 3, qui donnent une abondance de labels normaux globaux, sont avant tout une curiosité qui n'intéressera que les programmeurs confirmés. La génération des lignes de type 4 plaira aux puristes pour qui un seul octet de trop est un scandale. Nous discuterons en détails les méthodes pour faire les types 1 et 2, et nous apprendrons en passant comment faire les lignes de type 4. Nous n'accorderons que peu d'attention aux lignes de type 3; en général , elles peuvent être crées de la même façon que les lignes du type 2.

Nous avons étudié trois approches de la création des lignes synthétiques de programme. La plus puissante de celles-ci est l'utilisation de la routine de création d'octet "CODE" pour des séquences arbitraires d'octets qui peuvent être stockés en mémoire programme par "REG". Cette méthode de programmation par programme n'est pas limité dans ses applications. Avec elle, nous pouvons créer les quatre types de ligne, avec n'importe quelle combinaison d'octets. Le prix de cette puissance est l'en-tête de programme de presque 300 octets requis par "CODE" et "REG". Bien plus, nous avons besoin d'une méthode pour déterminer l'adresse du registre où les nouvelles lignes sont à stocker, ce qui demande encore un autre programme (voir section 5J).

La seconde technique de génération de lignes programme est d'utiliser "KA" pour assigner des fonctions synthétiques pour qu'elles puissent être entrées dans les programmes à la demande. Cette méthode est limité aux lignes de type 1. Comme "KA" est un long programme,il est optimisé dans son utilisation en créant de nombreuses assignations d'un coup puis en étant effacé de la mémoire.

La disponibilité de certaines assignations spéciales, qui dépend des finesses dans les opérations de la HP 41C, produit une troisième approche de la génération des lignes synthétiques. Le de ces assignations est le sauteur d'octets, que nous avons utilisé pour commencer tout le processus de la programmation synthétique. Comme nous verrons dans la section suivante, l'utilisation des fonctions assignées 'RCL e' et 'STO e' donne une amélioration importante du processus du sauteur d'octets. Les sections 5H et 5I décrivent deux nouvelles assignations spéciales pour l'édition exotique, l'autorisateur de texte et le chargeur Q. L'autorisateur de texte nous permet de convertir des lignes programme arbitraires existantes en lignes de texte et vice-versa. Le chargeur Q est utilisé avec "CODE" pour stocker des codes arbitraires de 7 octets dans les programmes en tant que lignes de texte. Le sauteur d'octets et l'autorisateur de texte demandent une petite en-tête de programme, mais sont quelque peu limités dans les combinaisons d'octets qu'ils peuvent produire séparément (utilisés ensemble, le sauteur d'octets et l'autorisateur de texte peuvent faire n'importe quelle combinaison d'octets sauf celles contenant les octets E4 à EF). N'importe quelle combinaison de 7, ou moins, octets peut être faite en utilisant l'auto-

risateur de texte avec "CODE".
De ces considérations,il découle qu'une approche recommandée de la programmation synthétique est de se faire 'un clavier standard de programmation synthétique', comme celui montré à la figure 5-1, avec des assignations de touches pour les fonctions synthétiques les plus utilisées. Quand le besoin se fait sentir d'autres lignes synthétiques de programme, utilisez le sauteur d'octets et/ou l'autorisateur de texte. Quand ceux-ci sont insuffisants, charger "CODE" et utilisez le chargeur-Q. Enfin, si nécessaire, utilisez "REG" pour toute combinaison étrange d'octets qui défierait les autres méthodes.
Une étude des différents programmes de ce livre révélerait que les fonctions synthétiques suivantes sont utilisées suffisamment souvent pour justifier une assignation permanente: STO M,RCL M, X<>M, STO N, RCL N, et X<>N pour un accés rapide au registre alpha; STO b et RCL b pour bouger le pointeur programme; X<>c et RCL c pour manipuler les adresses des registres de données; STO d, RCL d et X<>d pour l'accés au registre des drapeaux; STO e et RCL e pour améliorer le sauteur d'octets (voir section 5G); STO Q et le chargeur Q(voir section 5I); le sauteur d'octets. Si l'espace mémoire le permet, il est commode d'avoir "CODE" , "REG", "CS" et "CR"(voir section 5K), et "AD"(et même "DECODE", voir section 5J) en mémoire et assignés à des touches.
La figure montre une façon commode d'assigner les touches utilisateur pour faire un clavier de programmation synthétique. Notez que toutes les fonctions 'STO' et 'X<>' sont assignées à des touches shiftées pour réduire le risque de stockage accidentel dans un registre sensible.
"KA" peut être utilisé pour faire les assignations de fonctions requises pour la clavier utilisateur montré à la figure 5-1. La table 5-2 liste les préfixes, les postfixes et les codes de touches requis, comme les codes XROM seront affichés quand une des touches assignées est pressée et tenue.

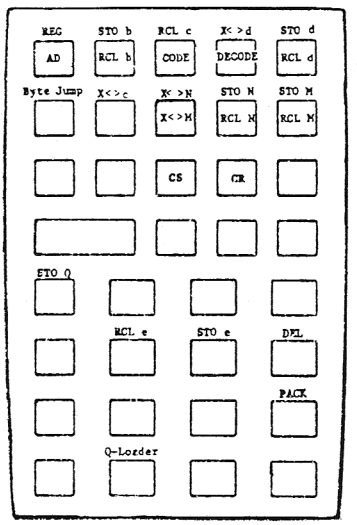


FIGURE 5-1 Un clavier de programmation synthétique

TABLE 5-2

Entrées de "KA" pour le clavier de la figure 5-1

Fonction	Préfixe	Postfixe	Code de touche	Code XROM
STO b	145	124	-12	05,60
RCL b	144	124	12	01,60
RCL c	144	125	-13	01,61
X<>d	206	126	-14	57,62
STO d	145	126	-15	05,62
RCL d	144	126	15	01,62
X<>c	206	125	-22	57,61
X<>N	206	118	-23	57,54

Table 5-2 (suite)

Fonction	Préfixe	Postfixe	Code de touche	Code XROM
X<>M	206	117	23	57,53
RCL N	144	118	24	01,54
STO N	145	118	-24	05,54
RCL M	144	117	25	01,53
STO M	145	117	-25	05,53
STO Q	145	121	-51	05,57
RCL e	144	127	-62	01,63
STO e	145	127	-63	05,63
Chargeur Q	4	25	-82	"OD"
Sauteur d'octets	241	65	-21	05,01

La séquence suivante assignera aussi les fonctions de la table 5-2:

1. Assignez n'importe quelle fonction de la HP 41C aux touches suivantes:

$1/X$	y^x	X Y?
10^x	x^2	ASIN
LN	e^x	ACOS
%	TAN	ATAN
SIN	CL	P-R
COS	BEEP	π

2. Exécutez "REG" 9 fois, en entrant les codes listés aux prompt correspondants:

RUN	entrée "REG?"	entrée "CODE=?"
1	0C0	F091763A91754A
2	0C1	F0F1410ACE762A
3	0C2	F0CE7E39CE7522
4	0C3	F0917C19907C11
5	0C4	F0CE7D1A907D29
6	0C5	F0917E49907E41
7	0C6	F0907542907632
8	0C7	F091790D041920
9	0C8	F0907F1E917F2E

5G. AMELIORATION DU SAUTEUR D'OCTETS (eJUMP)

Le principal défaut de l'édition par sauteur d'octets est son incapacité à changer directement le deuxième octet d'une ligne programme multi-octets. De ce fait, nous devons utiliser le 'générateur', une ligne de texte de paille utilisée pour cacher les octets de préfixe pour que les postfixes puissent être insérés dans le programme puis attachés au préfixe quand il est poussé dehors du générateur. En plus de demander un double usage du sauteur d'octets, cette méthode laisse beaucoup d'octets rémanants qui doivent être effacés, y compris le générateur.
Cette incapacité à changer les seconds octets vient du fait que les octets sont normalement insérés dans le programme en suivant le dernier octet de la ligne courante affichée. Mais comme il a été dit pour la première fois par Roger Hill, cette règle ne s'applique pas si le numéro de la ligne courante de programme est '00'. Dans ce cas, l' octet inséré entre dans le programme immédiatement aprés l'octet d'adresse le pointeur d'adresses. L'amélioration du sauteur d'octets tire parti de ce trait pour éliminer le besoin d'une ligne de générateur.
La méthode la plus aisée pour amener la ligne de numéro '0' est de presser 'RTN', quoique cela ramène le pointeur au sommet du fichier programme courant. Mais la simple séquence (PRGM off) RTN, RCL e, GTO .lmn, STO e, change le numéro de n'importe quelle ligne de programme de sa valeur normale à '00'. Le registre e contient le numéro de ligne; à n'importe quel moment, remettre dans le registre e la valeur qu'il contenait aprés un RTN réinitialise un numéro de ligne arbitraire à 0. Mettre en mode PRGM aprés cette séquence donne toujours l'affichage '00 REG lmn'. Vous pouvez aussi arriver au même résultat avec GTO .lmn, RCL b, RTN, STO b.
Comme nous l'avons fait pour la séquence originale du sauteur d'octets, il est commode pour la simplification des instructions de programme d'introduire une nouvelle instruction 'eJUMP .lmn' qui demande à l'utilisateur de faire la séquence de touches suivante:

'eJUMP .lmn' veut dire

1. PRGM off	1. PRGM off
2. RTN	2. GTO.lmn
3. RCL e	3. Saut d'octets
4. GTO. lmn	4. RCL b
5. Saut d'octets	5. RTN
6. STO e	6. STO b
7. PRGM on	7. PRGM on

2 versions possibles

Pour une séance prolongée d'édition de programme, la première méthode est préférable. Si les pas 2 et 3 sont éxécutés au début de la séance, aussi longtemps que le contenu du registre e est laissé intact dans le registre X(et aussi longtemps qu'aucune assignation de touche shiftée n' est changée), ces deux étapes peuvent être omisesde la séquence.
Pour illustrer l'utilisation du sauteur d'octets amélioré, créons un 'RCL M':

```
01 STO 01 (contrôleur)
02 RCL 99 (n'importe quel registre donnée de numéro supérieur à 15)
                  eJUMP .002                    (00 REG lmn )
                  DEL 001*                      (00 REG lmn )

01 RDN
                  GTO .001                      (01 STO 01  )
                  DEL 001
                  SST                           (01 RCL M   )
```

* Dans ce cas, DEL 001 n'est pas équivalent à . Vous pouvez y substituer SST, .

Les fonctions à deux octets, autres que 'STO', 'RCL', et 'LBL' (qui ont des formes à un octet pour 'STO 00', 'RCL 00' ,'LBL 00'), sont plus faciles à éditer si elles sont initialement entrées avec '00' comme postfixe, puisque le 'DEL 001' suivant le 'eJUMP' n'est plus nécessaire:

```
01 STO 01
02 ISG 00
                  eJUMP .002                    (00 REG lmn )
01 LAST X
                  GTO .001                      (01 STO 01  )
                  DEL 001
                  SST                           (01 ISG N   )
```

Cette méthode n'est pas limitée aux fonctions à deux octets. Dans la séquence suivante, nous créons la ligne de texte à 9 caractères de la ligne 90 du programme 'HANGMAN' de la section 6C. Le code est 'F9 40 40 40 40 40 40 43 4C 5F'.

```
90 STO 01
91 "ABCDEFGHI"
                  eJUMP .091                    (00 REG lmn )
                  DEL 009                       (00 REG lmn )
01 +
02 +
03 +
04 +
05 +
06 +
07 /
08 %
09 DEC
                  GTO .090                      (01 STO 01  )
                  DEL 001
                  SST                           ("@ÉÉÉÉÉÉCL-")
```

5H. L'AUTORISATEUR DE TEXTE

La création de lignes de texte programme est un processus intéressant. Quand un utilisateur presse la touche du caractère initial, en mode PRGM-ALPHA, le processeur écrit un octet 'F1' de texte, suivi par l'octet du caractère, en mémoire. Pour chaque caractère suivant ajouté à la chai-

ne, le processeur doit mettre à jour l'octet de texte aussi bien qu'ajouter un nouvel octet de caractère. L'information nécessaire au processeur pour garder trace de cette opération est enregistrée dans le registre Q pendant l'entrée de la ligne de texte. Le premier nybble de Q est le nombre courant de caractères dans la chaine; les quatre derniers nybbles enregistrent l'adresse du dernier octet entré. Le drapeau 45, le drapeau d'entrée de donnée, est levé durant l'entrée de la ligne. Une fois que le drapeau 45 est baissé par une séquence de touche terminant l'entrée des caractères, il n'est normalement plus possible d'ajouter d'autres caractères à la ligne. De ce fait, il ne devrait pas y avoir moyen de changer des caractères dans une ligne de texte existante sauf en l'effaçant et en la recommençant.
Mais de nouveau, la programmation synthétique nous conduit dans un terrain noir et inexploré. Nous pouvons manipuler le drapeau que nous voulons (voir section 6F) en stockant le NNN approprié dans le registre d. En particulier, si nous utilisons 'STO d' pour lever simultanément les drapeaux 45 (entrée de données), 48 (ALPHA), et 52 (PRGM), nous pouvons en fait ajouter des caractères à une ligne programme de texte existante. Bien plus, nous verrons que nous pouvons changer n'importe quelle séquence d'octets de programme en ligne de texte, et vice-versa!
Mais un mot de prudence. Il y a des gouffres sans fond, ce qui fait que les instructions suivantes doivent être suivies exactement, pour éviter des 'crash' de la HP 41C qui demanderaient de retirer la batterie pour retrouver le contrôle. De plus, toutes les HP 41C ne sont pas semblables. Soyez donc prêts pour une expérimentation.
Le fait de stocker le NNN magique dans le registre d sera appelé 'l'autorisateur de texte', et le NNN lui même sera appelé le TEN (initiales de 'nombre autorisateur de texte' en anglais). Un NNN convenable est un NNN qui a '0' dans son premier nybble et '488' dans ses trois derniers nybbles. Le '488' vient des trois bits du registre d qui correspondent aux drapeaux 45, 48 et 52. Comme le stockage dans le registre d affecte les 56 drapeaux, nous pouvons choisir à volonté les nybbles non spécifiés du TEN pour avoir un format d'affichage agréable. Le NNN '00 00 02 3C 04 84 88' utilisé dans les exemples qui suivent comme TEN lève les drapeaux 26 (son), 27 (USER), 28 et 29 (format FIX 4), et met en mode DEG.

Utilisez "CODE" pour générer le TEN:

```
                XEQ "CODE"                  (CODE=?      )
"00000023C048488"
                R/S                         (■ﾉ天■■      )
                ASTO 00
```

Le 'ASTO 00' sauvegarde le TEN pour des utilisations futures. Si vous voulez rappeler le TEN, toutefois, faites CLA, ARCL 00, RCL M, plutôt que RCL 00, pour éliminer l'octet d'identification alpha que le ASTO 00 a ajouté au NNN.
Maintenant, éxécutez GTO.. pour commencer un nouveau fichier programme. Mettez en mode PRGM et entrez la ligne de texte "ABC". Si vous éteignez le mode ALPHA aprés l'entrée du "C", la ligne est terminée. Mais si vous éteignez ensuite le mode PRGM, et pressez 'STO d' avec le TEN dans le registre X, vous verrez la ligne '01 "ABC" en mode PRGM et ALPHA; presser une touche ajoute le caractère correspondant à la ligne de texte (plus le "_" pour les caractères uivants). Presser la touche de correction efface des caractères dans la chaine, y compris les caractères originaux "ABC" si nécessaire. Cependant, ce truc ne marche que si le contenu du registre Q est demeuré intact depuis que l'entrée de la ligne originale est terminée. En général, revenir à une ligne de texte par le moyen de l'autorisateur de texte aprés que des instructions intermédiaires aient été éxécutées ne permettra pas d'ajouter des nouveaux caractères mais provoquera certainement un crash. Le processeur a besoin de corriger des instructions dans le registre Q pour continuer la construction de la ligne de texte, sinon il est perdu sans espoir.
Nous voilà donc avec un étrange processus. Plutôt que d'essayer d'ajouter à une ligne de texte existante depuis la fin de la ligne, nous mettons délibérément le premier nybble de Q à zéro en stockant le TEN dans Q immédiatement avant de le mettre dans d. Remarquablement, avec le TEN dans Q , le processeur fera des lignes de texte des octets qui sont déjà présents en mémoire. Si le premier octet se trouve ne pas être un 'Fn', le processeur le remplace simplement par un 'Fn', changeant 'n' à chaque fois que des caractères sont ajoutés ou effacés de la ligne de texte. Tout ceci est bien mieux expliqué par un exemple. Une nouvelle notation raccourcie est nécessaire:

'TE .lmn' veut dire

1. PRGM off
2. Placer le TEN dans le registre X (CLA, ARCL 00, RCL M)
3. GTO. lmn
4. STO Q
5. STO d

Maintenant effacez le programme existant et entrez:

```
01 "ABC"
02 X<Y?
03 X>Y?
04 X<=Y?
```

```
                PACK
                TE .001          (01 "ABC"    )

Pressez la touche "A"            (01 "A-"     )
Pressez la touche "A"            (01 "AB-"    )
Pressez la touche "A"            (01 "ABC-"   )
Pressez la touche "A"            (01 "ABCD-"  )
Pressez la touche "A"            (01 "ABCDE-")
Pressez la touche "A"            (01"ABCDEF-")

                SST              (.END.       )
```

Chaque pression de la touche "A" prend un octet existant du programme et l'incorpore dans la ligne de texte. N'importe quelle touche de caractère aurait agit de la même façon que la touche "A", i.e., n'importe quelle touche autre que 'SST', 'BST', 'SHIFT', 'R/S' et la touche de correction. Si nous avions continué aprés les six "A", le .END. lui-même aurait été absorbé dans le texte. Cependant, certaines HP 41C ne continuerait pas aprés le 7ème caractère de la ligne; presser une huitième touche causerait un crash. Malheureusement, le seul moyen pour déterminer quel type de HP 41C vous avez est d'essayer de faire un huitième caractère avec l'autorisateur de texte.
La touche de correction ne travaille que d'une façon avec l'autorisateur de texte. Si elle est pressée à n'importe quel moment aprés que la ligne de texte ait été commencée avec au moins un caractère, le caractère le plus à droite sera effacé de la ligne de texte mais restera en mémoire comme une ligne mono-octet (ou comme préfixe). Vous avez actuellement en memoire la ligne '01 "ABCDEF"' . Essayez:

```
                TE .001          (01 "ABCDEF" )
Pressez la touche "A" six fois   (01 "ABCDEF-")
                ←                (01 "ABCDE-" )
                SST              (02 X<=Y?    )
```

Comme vous voyez, l'octet "F" n'a pas été effacé, seulement éjecté de la ligne de texte pour reprendre son rôle dans la ligne '02 X<=Y?'. Si vous n'aviez pas pressé la touche 'SST', les utilisations ultérieures de la touche de correction auraient déplacé les caractères "E", "D", "C", et "B" de la ligne, laissant leurs octets en mémoire. Mais un effacement de plus, avec '01 "A-" affiché, efface le "A" et l'octet de texte 'F1' de la mémoire.
Si la touche de correction est la première pressée aprés le 'STO d' de l'autorisateur de texte, le premier octet de la ligne affichée est remplacé par un 'FF', et les 15 octets suivants de programme sont incorporés dans une ligne de texte. Les pressions ultérieures de la touche de correction 'remonte' le processus, redonnant à chaque carcatère le plus à droite son rôle original et rétrécissant la ligne de texte.
L'autorisateur de texte fait qu'il est simple de générer des lignes de texte contenant des caractères non entrables au clavier. Chaque caractère désiré est entré dans le programme comme une ligne de programme mono octet. La chaine doit être précédée par une ligne mono octet de paille, qui sera convertie en octet de texte. Quand nous sommes satisfait de la séquence de caractères, nous utilisons simplement l'autorisateur de texte pour transformer les lignes mono octet en une ligne de texte, comme nous enfilons des perles:

```
01 X<>Y (de paille)
02 LBL 11
03 X<=Y?
04 RCL 08
05 E↑X-1
06 RCL 09
                TE .001
Pressez cinq touches de caractères
                ALPHA            (01 "μF(X)"  )
```

Si vous voulez une ligne de plus de 7 caractères mais que votre HP 41C ne coopère pas, vous pou-

vez presser la touche de correction aprés l'autorisateur de texte, puis l'utiliser pour effacer des caractères jusqu'à ce qu'il ne vous reste que la ligne désirée.
L'autorisateur de texte peut être utilisé pour générer des fonctions synthétiques, en s'en servant pour déplacer un préfixe existant dans une ligne de texte, éditer un nouveau préfixe, puis en utilisant de nouveau l'autorisateur de texte pour éjecter le préfixe. Cette procédure est généralement plus compliquée pour les fonctions à deux octets que l'utilisation du sauteur d'octets amélioré. Mais nous verrons dans la prochaine section qu'une combinaison de l'autorisateur de texte et du chargeur Q produit une façon soignée d'assembler des lignes synthétiques directement avec un NNN dans le registre X.
Faire des lignes de programme de type 4 est un exercice plaisant avec l'autorisateur de texte:

```
01 1 E25
                    TE .001                    (01 1 E25     )
Pressez une touche de caractère                (01 "■"       )
Pressez la touche de correction, SST           (01 E25       )
```

Ceci marche si bien parce que le nul automatiquement inséré devant le '1' sert d'octet de paille pour être transformé en octet 'F1', puis effacé avec le '1'. Si vous voulez effacer le '1' d'une ligne déjà éxistante, dans un fichier programme compacté, vous devez d'abord éditer un octet de paille puis PACK avant d'utiliser l'autorisateur de texte.

5I. LE CHARGEUR Q

Dans la section 4C il a été montré comment le registre Q est utilisé temporairement pour le stockage de chaines alpha qui ne sont pas stockées en programme ou dans le registre alpha. A travers une autre particularité dans les opérations de la HP 41C, nous pouvons exploiter ce comportement pour assister la création de lignes de texte jusqu'à 7 caractères de long. Nous avons besoin des assignations de touches de 'STO Q' et du chargeur Q (préfixe 4, postfixe 24) que nous avons faites à la section 5F.
Le chargeur Q est à priori l'assignation à une touche utilisateur du nombre '9', i.e., l'octet '19'. En fait, l'assignation de n'importe quel octet entre '10' et '1C' fonctionnera comme un chargeur Q, mais l'assignation '19' est facile à utiliser, et donne un affichage aisément reconnaissable "0D" quand la touche est pressée et tenue. Si elle est pressée en PRGM off, le chargeur Q entre un '9' dans le registre X. Mais en mode PRGM, le chargeur Q entre deux lignes programme: la première est juste une ligne '9', mais la seconde est une ligne de texte, contenant les caractères présents dans le registre Q. Pour illustrer ceci, éxécutez PACK (PRGM on) en épelant 'XEQ "PACK"', puis en pressant le chargeur Q. Vous verrez la ligne '9' suivie par la ligne de texte "PACK". Les lettres "P-A-C-K" ont été placées dans le registre Q par 'XEQ "PACK"', puis transférées dans la ligne de texte par le chargeur Q.
En utilisant 'STO Q', nous ne sommes pas limités aux séquences que nous pouvons épeler; n'importe quel NNN peut être placé dans Q. Pour une souplesse complète, nous pouvons utiliser "CODE" pour assembler le NNN. Donc, pour faire une ligne de 7 octets arbitraires:

1. Utiliser "CODE" pour créer une chaine de 7 octets correspondant aux caractères que vous voulez dans la ligne de texte, entrés dans l'ordre inverse. Si vous ne voulez que 'n' caractères, où 'n' est plus petit que 7, alors les (7-n) caractères de tête à entrer dans "CODE" doivent être des nuls (00). Notez que des lignes de texte finissant par un ou plusieurs nuls ne peuvent pas être créées par cette seule métode directement.
2. 'GTO' la ligne programme précédant l'endroit où vous voulez insérer la nouvelle ligne de texte.
3. En PRGM off, pressez STO Q.
4. Mettez en mode PRGM, et pressez la touche du chargeur Q. Effacez la ligne '9' résultant, puis SST pour voir la nouvelle ligne de texte.

A titre d'exemple, utilisons le chargeur Q pour faire la ligne 81 du programme 'HANGMAN' de la section 6C. Le code de cette ligne est 'F5 60 06 04 05 01':

```
                    XEQ "CODE"                 (CODE ?       )
"00000105040660"
                    R/S                        ("ÆÆÆJ"       )
                    GTO .000*
                    STO Q
```

* Dans la réelle construction de 'HANGMAN', ce serait 'GTO .080'.

```
                    PRGM (on)
                    Chargeur Q                  (01 9        )
                    DEL 001
                    SST                         (01 "ƒƒƒƒ"   )
```

Le chargeur Q insère cette ligne de texte n'importe où dans un programme, sans considération pour les frontières de registres ou leurs adresses requises pour l'utilisation de "REG". En utilisant l'autorisateur de texte por transformer les lignes de texte générées par le chargeur Q en ligne programme séparées, nous pouvons faire des lignes synthétiques de programme arbitraires jusqu'à une longueur de 7 octets, ou plusieurs lignes plus courtes simultanément. Faisons la séquence '01 "(#)"','02 ASTO M':

```
                    XEQ "CODE"                  (CODE=?      )
"0000759A292328"
                    R/S                         ("▮▮)#("     )
                    GTO ..
                    STO Q
                    PRGM on
                    Chargeur Q                  (01 9        )
                    DEL 001
                    PACK
                    PRGM Off
                    TE .001
Pressez trois touches de caractères             (01 "(#)"    )
                    SST                         (02 ASTO M   )
```

Si vous voulez qu'aucun caractère de la chaine ne reste à l'état de caractère de texte, vous pouvez utiliser l'autorisateur de texte pour enlever l'octet de texte de la chaine, comme nous avons éliminé le '1' des lignes '1 En' dans la section 5H.
Il y a d'autres types d'assignations du chargeur Q disponibles, correspondant aux autres utilisations des chaines de texte programme. Les assignations suivantes ont été développées par Tom Cadwallader:

Code octet	Préfixe/postfixe "KA"	Charge le contenu de Q dans
04 1D	4/29	GTO (alpha)
04 1E	4/30	XEQ (alpha)
CD 00	205/00	LBL (alpha)

Un inconvénient mineur à l'utilisation du chargeur Q est la nécessité de coder la séquence d'octets à l'envers, ce qui est facile à oublier. Cependant, si la séqunece désirée est de 6 octets ou moins, la routine suivante inversera automatiquement la séquence:

```
01 LBL "REV"
02 ASTO M
03 SF 25
04 GTO IND M                "REV"
05 RCL Q                    20 OCTETS
06 STO M
07 END
```

"REV" est faite pour inverser une chaine de 6 caractères maximum dans le registre alpha. Pour inverser une chaine dans le registre X, éxécutez un STO M avant de faire 'XEQ REV'. Notez que "REV" ne fonctionnera pas correctement s'il y a un label global du même nom que dans la chaine alpha, mais ce fait est rare. De même, "REV" échouera si l'imprimante est connectée.

5J. PARLER A LA HP 41 C

A ce point, nous pouvons 'dire' à la HP 41C tout ce que nous voulons, en utilisant "CODE" pour traduire les caractères lisibles par l'utilisateur en code interne HP 41C. Mais pour un dialogue complet avec la machine, nous avons aussi besoin d'un programme pour prendre des codes éxistants et les déchiffrer en caractères lisibles. Le programme "DECODE", listé ci-aprés, renverse exactement l'opération de "CODE", prenant un code arbitraire de 7 octets dans X et le traduisant en 14 caractères dans le registre alpha. Comme adjuvant, les caractères de sortie sont groupés par 2 (octet) en utilisant les deux points comme séparateur. L'opération de base de

"DECODE" est pratiquement identique à celle de "CODE". "DECODE" sert d'illustration à l'utilisation du registre P comme une extension de 7 octets complets au registre alpha, comme décrit à la section 4B.

```
01*LBL "DECODE"     27 CF 09     53 RCL ↑     79 FC? 14
02 CLA              28 SF 10     54 STO J     80 RTN
03 .006             29 SF 11     55 R↑        81 CF 13
04 STO L            30 FS?C 04   56 ISG L      82 RTN
05 X<>Y             31 XEQ 01    57 GTO 13    83*LBL 02
06 GTO 14           32 FS? 03    58 0         84 FS? 05
07*LBL 13           33 SF 07     59 STO ↑     85 CF 02
08 STO ↑            34 FS? 02    60 TONE 9    86 FS? 06
09 ":"              35 SF 06     61 AVIEW     87 CF 02
10 RCL ↑            36 FS?C 01   62 RTN       88 FS? 02
11*LBL 14           37 SF 05     63*LBL 01    89 RTN
12 ENTER↑           38 FS?C 00   64 FS? 13     90 CF 04
13 X<> d            39 SF 04     65 CF 10     91 CF 03
14 CF 12            40 SF 02     66 FS? 14    92 SF 01
15 CF 13            41 SF 03     67 CF 10     93 FC?C 07
16 CF 14            42 FS? 04    68 FS? 10    94 SF 07
17 CF 15            43 XEQ 02    69 RTN       95 FC? 07
18 FS?C 07          44 FIX 5     70 CF 12     96 RTN
19 SF 15            45 ARCL L    71 CF 11     97 FC?C 06
20 FS?C 06          46 X<> d     72 SF 09     98 SF 06
21 SF 14            47 STO [     73 FC?C 15   99 FC? 06
22 FS?C 05          48 "+12"     74 SF 15     100 RTN
23 SF 13            49 X<> \     75 FC? 15    101 CF 05
24 FS? 04           50 STO [     76 RTN       102 END
25 SF 12            51 RCL J     77 FC?C 14
26 CF 08            52 STO \     78 SF 14
```

"DECODE"
203 OCTETS

Instructions pour "DECODE":

1. Placer le code à déchiffrer dans le registre X.
2. XEQ "DECODE"
3. Les caractères de sortie sont placés en alpha, montrés avec 'aview'.

Exemples:

ALPHA "ABCDEFG" ALPHA, RCL M , XEQ "DECODE" "41:42:43:44:45:46:47"
-1.234567891 E-56, XEQ "DECODE" "91:23:45:67:89:19:44"

"DECODE" est fait pour manipuler des codes de 7 octets complets, ce qui est vraiment trop pour une application particulière, c'est à dire pour déterminer l'adresse du pointeur programme. Si nous avons 'RCL b' assignée à une touche, alors 'RCL b', 'XEQ "DECODE"' déchiffrera certainement le registre b pour nous. Usuellement, cependant, nous ne sommes pas intéressés par les 5 premiers octets du registre b (adresses de retour des sous routines). "AD" (pour 'Adresses') est une routine rapide qui décodera une adresse sur deux octets, en sacrifiant la puissance et élégance de "DECODE" à la rapidité d'éxécution.
"AD" est rapide parce qu'il est court, mais il est 'maudit' parce qu'il utilise les propriétés de l'affichage en FIX 9 pour une conversion rapide des codes héxa en caractères. Notez que puisque la sortie est visualisée par le registre alpha, un digit 'A' sera représenté par ":" plutôt que par le caractère plein. Voir la section 5A.
"AD" laisse l'adresse originale dans le registre X, pour qu'un STO b éxécuté aprés "AD" retourne le pointeur à cette adresse. Cette particularité requiert la ligne superflue autrement '15 STOP', pour être sur qu'une mise en mode PRGM aprés un STOb donnera le numéro de ligne correct (si le .END. terminait l'éxécution de "AD", nous finirions avec le numéro de ligne '00').

```
01♦LBL "AD"
02 STO [
03 "⊢♦♦♦"
04 RCL d
05 FIX 9
06 ARCL [
07 STO d
08 X<> [
09 " "
10 X<> [
11 STO \
12 ASTO L
13 RDN
14 VIEW L
15 STOP
16 END
```

"AD"
39 OCTETS

Instructions pour "AD":

1. Pressez RCL b (PRGM off)
2. XEQ "AD"
3. La sortie est quatre caractères alpha (montrés avec AVIEW) correspondant aux quatre digits du pointeur d'adresses obtenu avec RCL b. Les digits héxa plus grands que 9 sont représentés comme suit: 'A' = ":", 'B' = ";", 'C' = "<", 'D' = "=", 'E' = ">", et 'F' = "?".
4. Pour retourner à l'octet original (où le RCL b a été éxécuté), pressez 'STO b'.

5K. CODE DE STOCKAGE

Une paire de petites routines terminera notre 'librairie' de programmes pour programmer. Il y a beaucoup d'occasions où il est souhaitable de stocker un NNN dans un registre de données pour un rappel futur, mais la normalisation du NNN pendant un rappel depuis un registre de donnée est un obstacle majeur.

Une manière de rappeler un NNN sans normalisation est d'utiliser le sauteur d'octets pour transférer le NNN dans le registre alpha. Par exemple, supposons que le NNN en question soit dans le registre Roo, que nous déterminons être le registre 123 en décodant le registre c. Alors nous stockons '1 E7' dans le registre 124, i.e., Ro1. Puis nous utilisons "CODE" pour faire l'adresse '00 00 00 00 00 01 24', suivi par STO b. Presser la sauteur d'octets copiera le NNN du registre Roo dans le registre M. Le '1 E7' dans Ro1 a mis l'octet '07' à l'adresse '0124', pour un saut de sept octets.

Cette méthode est assez maladroite, et ne peut être utilisée que manuellement. Pour des rappels et des stockages automatiques, nous pouvons utiliser les routines "CS" et "CR". "CS" prend un NNN et le coupe en deux morceaux, dont chaque est converti en donnée alpha pour un stockage ordinaire. Ceci demande deux registres de données pour stocker le code entier du NNN. "CR" renverse le processus, rappelant les deux chaines alpha et les réassemblant en un seul NNN de 7 octets. Par commodité, les routines sont désignées pour être éxécutées manuellement comme les ordinaires 'RCL' et 'STO': assignez "CS" et "CR" à des touches utilisateurs, et les éxécutez chacune en pressant la touche appropriée suivie (pendant la pause, qui commence à peu prés immédiatement) par le numéro de registre de donnée désiré. Chaque routine utlise le registre de donnée désigné plus celui de numéro immédiatement supérieur.

```
01♦LBL "CS"
02 CLA
03 STO [
04 STO L
05 RDN
06 PSE
07 X<> L
08 "⊢♦♦"
09 .9
10 ST+ L
11 X<> \
12 ASHF
13 ASTO IND L
14 ISG L
15 CLA
16 STO [
17 ASTO IND L
18 CLA
19 RDN
20 END
```

"CS"
40 OCTETS
SIZE 002

```
01♦LBL "CR"
02 PSE
03 STO L
04 CLA
05 ARCL IND L
06 "⊢♦♦"
07 ISG L
08 CLD
09 CLX
10 RCL IND L
11 STO \
12 "⊢♦♦♦♦"
13 X<> \
14 CLA
15 END
```

"CR"
37 OCTETS
SIZE 002

Instructions pour "CS":

1. Utilisation manuelle: XEQ "CS"; pendant la pause, entrer un numéro de registre de donnée 'mn'. Le NNN sera stocké en R_{mn} et R_{mn+1}. Les contenus des registres X, Y, et Z sont préservés.
2. Utilisation comme sous routine:le programme d'appel doit avoir le NNN en X, et 'mn' en Y. Aprés l'éxécution, les contenus des registres T et Z sauteront en Z et Y, respectivement.

Instructions pour "CR":

1. Utilisation manuelle: XEQ "CR"; pendant la pause, entrer un numéro de registre de donnée 'mn'. Le NNN en R_{mn} et R_{mn+1} sera placé en X, remplaçant 'mn' et, en effet, élevant les contenus de la pile avant l'éxécution de "CR".
2. Utilisation comme sous routine: le programme d'appel doit placer 'mn' en X avant d'appeler "CR". Aprés l'éxécution, le NNN rappelé remplacera 'mn' en X.

CHAPITRE 6

APPLICATIONS

Ce chapitre doit servir de 'CARNET D'APPLICATIONS STANDARDS'pour la programmation synthétique. Inclues sont de nombreuses routines de la HP 41C, qui, comme les programmes du chapitre 5, illustrent l'usage créatif des fonctions synthétiques tout en ayant des applications pratiques puissantes. Le but et la raison d'être de la programmation synthétique sont inclus dans ces routines. Premièrement, l'utilisation des fonctions synthétiques permet à la HP 41C de faire diverses opérations importantes plus vite et avec moins de place en mémoire que ça n'est possible avec seulement les fonctions standards. Les exemples sont le 'détecteur de SIZE' (section 6B) et la manipulation des chaines alpha (section 6C). Ensuite, la programmation synthétique produit une nouvelle classe d'opérations qui ne peuvent absolument pas être réalisées avec les fonctions standards seules. De telles opérations incluent l'identification de caractère alpha et leur comparaison (section 6D) et l'accés direct aux programmes de l'Application Pac' (section 6H).
L'ensemble des routines décrites dans ce chapitre ne constituent en aucun cas une liste complète des utilisations de la programmation synthétique. Aucune liste de programme ne peut faire le tour des capacités de la HP 41C, spécialement si améliorées par les fonctions synthétiques. Le développement des techniques et des applications de la programmation synthétique est un processus continu. (A titre d'exemple, la découverte de l'autorisateur de texte décrit dans la section 5H vient d'une erreur de frappe dans une édition préliminaire de ce livre!). Quand vous aurez complétement étudié le contenu de ce livre, vous serez préts à utiliser les fonctions synthétiques dans vos programmes de tous les jours, aussi facilement et avec peu d'effort en plus que quand vous utilisez les fonctions standards de la HP 41C. Vous pouvez même découvrir quelque nouveau truc vous même. A cet égard, la règle est 'ne rien prendre comme admis'. Si vous avez une idée, essayez pour voir si ça marche, aussi farfelue puisse-t-elle être. Il a fallu des mois de programmation synthétique avant que quelqu'un ne remarque que le registre P, par exemple, était une continuation complète de 7 octets du registre alpha. Puisque l'affichage montrait un maximum de 24 caractères, il était admis que les caractères perdus à la gauche du registre alpha était perdus pour toujours. Maintenant, ils sont là, se cachant dans le registre P

6A. ARRIVER AU .END.

Bien souvent, un programme en cours de réalisation est le dernier programme en mémoire, i.e., le fichier contenant le .END. Si le pointeur d'adresses est mis dans un autre fichier, il n'y a que deux moyens pour le faire revenir au dernier fichier: utiliser 'GTO' et épeler un label global qui est dans le programme, ou utiliser 'CAT 1' jusqu'à la fin du catalogue. Si le nouveau programme n'a pas de label global, la première méthode est éliminée. Si il y a plusieurs modules de mémoire et de nombreux programmes dans la HP 41C, la seconde méthode peut être ennuyeusement longue. Le programme "EN" ajoute une troisième méthode, que vous trouverez d'une grande commodité durant de nombreuses sessions d'édition, particulièrement si vous programmez en utilisant les routines du chapitre 5.

```
01*LBL "EN"          10 SF 03
02 RCL c             11 X<> d
03 STO [             12 CLA
04 "┣♦♦♦♦♦"          13 STO [
05 X<> [             14 "┣♦♦"
06 X<> d             15 X<> \
07 CF 00             16 STO b
08 CF 01             17 END
09 SF 02
```

"EN"

45 OCTETS

Instructions pour "EN":

1. XEQ "EN".
2. A la fin, le pointeur programme sera en haut du fichier programme contenant le .END.

Le '.END.' est situé dans les octets 2, 1 et 0 du registre identifié par l'adresse enregistrée dans les trois derniers nybbles du registre c. "EN" prend cette adresse 'lmn' (ligne 02), la met dans les trois premiers nybbles du registre d (lignes 03-06), et fabrique le code '3lmn'

en baissant les drapeaux 0 & 1, et en levant les drapeaux 2 & 3 (lignes 07-10). Ce code est ensuite mis dans les deux derniers octets du registre N (lignes 11-15). Quand le code est finalement transféré au registre b à la ligne 16, le pointeur d'adresses saute immédiatement à l'octet précédant le .END. Le programme continue à tourner, pour que le .END. lui-même soit éxécuté ce qui stoppe l'éxécution avec le pointeur au sommet du fichier.

6B. DETECTION DE LA SIZE ET AUTRES TRUCS

Une élégante démonstration de la manière dont les fonctions synthétiques améliorent les performances de la HP41C est la routine suivante de détection de la SIZE, écrite par Keith Jarett (PPC Calculator Journal, V7 N5 P57):

```
01♦LBL "S"      14 FS?C 13      27 CHS
02 "AB"         15 SF 11        28 64
03 RCL c        16 FS?C 14      29 MOD
04 X<> [        17 SF 13        30 SF 25
05 STO \        18 FS?C 15      31♦LBL 14
06 ASHF         19 SF 14        32 VIEW IND X
07 "┣♦♦♦"       20 FS?C 16      33 FC? 25
08 X<> [        21 SF 15        34 RTN
09 X<> d        22 X<> d        35 64
10 FS?C 11      23 1 E3         36 +
11 SF 09        24 *            37 GTO 14
12 FS?C 12      25 INT          38 END
13 SF 10        26 DEC
```

"S"

75 OCTETS

Instructions pour "S":

1. XEQ "S".
2. A la fin, la 'size' courante est affichée dans le registre X.

Il n'y a malheureusement pas de moyen direct pour déterminer la size programmes/données courante dans la HP 41C. Aucune adresse de sommet de mémoire n'est gardée en mémoire, puisqu'elle change avec chaque insertion ou ablation de module de mémoire. Le seul moyen de déterminer la SIZE est d'essayer d'avoir accés à des registres de données de numéros croissants jusqu'à ce qu'un message 'NONEXISTENT' indique que le dernier registre éxistant de données a été dépassé. Cette méthode peut être mise en programme, en se servant du drapeau d'erreur 25, mais si le nombre de registres de données est grand, le processus peut être très long. Même le plus performant de ces programmes prend 4 secondes pour aboutir. La routine "S" prend un maximum de 1.5 seconde. L'amélioration vient d'un décodage partiel du registre c, qui produit une valeur de départ pour la SIZE qui doit être incrémentée de 64 fois le nombre de modules présents. Seul un max de 4 registres doit être testé pour déterminer le nombre de modules.
Le 'coeur' de la routine de détection de la size est les lignes 09-26, qui constituent une conversion héxa/décimal sur trois digits développée par Roger Hill. Les trois digits en question sont les digits 9, 10, et 11 du registre c, l'adresse absolue de Roo. Les lignes 01-09 de "S" placent ces digits dans le registre d comme digits 3, 4 et 5, i.e., comme les drapeaux 8-19. Considérons une adresse de Roo typique, mettons '12A', qui, en décimal, est (256+32+10=298); utilisez la fonction 'OCT' pour voir que 298_{10} équivaut à 452_8. Ecrivons les deux nombres '12A' et '452' comme ils sont codés par la HP 41C:

héxadécimal	=12A	0001 0010 1010	binaire
octal	=452	0100 0101 0010	binaire

Notez que les deux nombres ont le même nombre de bits à valeur '1'. La différence entre les deux représentations est que le premier bit de chaque digit octal est toujours '0', puisque les digits octal ont une valeur maximale de 7 (0111). Pour convertir chaque bit héxadécimal en octal, il nous suffit de bouger les valeurs de certains bits vers la gauche pour faire de la place pour les bits '0' supplémentaires. Voici un second exemple, qui montre le mouvement des bits du modèle héxa vers l'octal:

héxadécimal	=1BC	0001 1011 1100
		↙↙ ↓↓ ↙ ↓↓↓
octal	=674	0110 0111 0100

Ce mouvement de bits est aisément réalisé par des opérations explicites sur les drapeaux utilisateurs, comme on peut voir dans les lignes 10-21 de "S". Les lignes 22-26 complètent la conversion héxa à décimal, prenant les trois digits du registre d et les convertissant en un entier décimal dans le registre X.
Le résultat 's' dans X est l'adresse absolue de Roo, maintenant exprimée en décimal. S'il n'y a pas de modules de mémoire dans le calculateur, la 'size' est (256-s) où '256' est l'adresse décimale du sommet de la mémoire. Cependant, comme (256-s) est toujours inférieur à 256 (sans module), et comme 256 est un multiple de 64, (-s mod 64) (qui, comme '-s' est négatif, est le plus petit nombre positif obtenu en ajoutant un multiple de 64 à '-s') est la même chose que (256+N*64 -s)mod 64,où 'N' est le nombre de modules. De ce fait, les lignes 27-29 donnent la distance, en registres, de Roo jusqu'à la frontière du module supérieur adjacent. La size est ce nombre plus un multiple inconnu de 64. Les lignes 30-37 sont une méthode de tests pour déterminer 'N', en incrémentant 's' par pas de 64 jusqu'à ce que 'VIEW IND X' cause une erreur qui baisse le drapeau 25.
Le shéma de la conversion héxa/décimal de "S" peut être utilisé dans une variété de programmes. Seule une petite modification de "S" serait requise, par exemple, pour donner l'emplacement du bloc des registres statistiques depuis les trois premiers digits du registre c. Un exemple différent est donné par la routine suivante, "BYTE", qui est faite pour donner l'emplacement du pointeur d'adresses courant sous forme d'un nombre décimal d'octets, compté depuis le bas de la mémoire programme utilisateur. Dans ce contexte, l'octet '1' est l'octet 'OCO'.

```
01♦LBL "BYTE"      12 SF 15        23 LASTX
02 CLA             13 FS?C 18      24 FRC
03 STO [           14 SF 17        25 1 E3
04 "┣♦♦♦♦"         15 FS?C 19      26 *
05 X<> [           16 SF 18        27 DEC
06 X<> d           17 FS?C 20      28 7
07 FS?C 15         18 SF 19        29 *
08 SF 13           19 X<> d        30 +
09 FS?C 16         20 10           31 1343
10 SF 14           21 *            32 -
11 FS?C 17         22 INT          33 END
```

"BYTE"

69 OCTETS

Instructions pour "BYTE":

1. Pressez 'RCL b'.
2. XEQ "BYTE".
3. Le nombre de sortie est le numéro de l'octet en décimal.

Après le 'RCL b' éxécuté par l'utilisateur, qui place le pointeur d'adresses courant dans les deux derniers octets du registre X, les lignes 01-06 de "BYTE" envoient les deux octets dans le second et le troisième octets du registre d (drapeaux 08-23). Le premier digit de l'adresse est un numéro d'octet, qui ne dépasse jamais 6. Les trois digits restants numérotent les registres de 7 octets, avec une valeur maximale de 1FF héxa, ou 777 octal. Donc le drapeau 12 sera toujours zéro. Les lignes 07-21 éxécutent la conversion héxa/octal, plaçant en X le nombre 'n.abc', où 'n' est le numéro d'octet, et 'abc' le numéro du registre sur trois digits octaux. La ligne 22 isole 'n', aprés quoi les lignes 23-27 convertissent 'abc' en un entier décimal. Les lignes 28-30 calculent tous les octets (n+7*abc), mesurés depuis '0000'. Les lignes 31-32 soustraient 1343, pour que le numéro d'octet de sortie soit mesuré depuis le bas de la mémoire programme normale, l'octet '00C0'.
Il y a deux applications évidentes pour "BYTE", dont chacune requiert deux éxécutions. Plus utile que le numéro d'octet de n'importe quelle adresse est la distance en octets entre deux emplacements mémoire. "BYTE" sauvegarde la valeur originale du registre X avant l'éxécution manuelle du 'RCL b', la plaçant dans le registre Y au dessus de la valeur de sortie à la fin de l'éxécution du programme. Donc la séquence

```
GTO 'point A'
RCL b
GTO 'point B'
RCL b
```

```
XEQ "BYTE"
X<>Y
XEQ "BYTE"
-
```

qui peut être éxécutée manuellement ou par programme, donnera la distance en octets entre 'point A' et 'point B' en mémoire. La première application de cette procédure est d'avoir 'point B' comme première ligne d'un programme, et 'point A' comme première ligne d'un autre programme plus bas en mémoire. Alors la différence d'octets est la longueur du programme, comme aussi donné par 'Cat 1' en utilisant l'imprimante. La deuxième utilisation est d'avoir comme 'point B' une ligne programme 'GTO' à deux octets et comme 'point A' le label correspondant, pour déterminer si le saut entre le 'GTO' et le label est plus petit que 112 octets. La seule autre manière de déterminer ce résultat est de compter les octets de programme, ligne par ligne.

6C. JEUX ET HUMOUR DANS LE REGISTRE ALPHA

Sans doute le groupe des fonctions synthétiques les plus utilisées est celui des fonctions d'accés au registre alpha, comme 'STO M', 'RCL N' ou 'X<>IND O'. Le fait que les postfixes 'M', 'N' 'O' et 'P' puissent être attachés à n'importe quel préfixe de fonction normale de registre de données signifie que le registre alpha peut être utilisé comme quatre (avec quelques limitations pour les utilisations de P) registres de données supplémentaires. Ceci est évidemment avantageux quand l'espace mémoire est limité. L'utilisation du registre alpha libère quatre registres ordinaires pour des programmes ou des données additionnelles. La meilleure utilisation pour ces registres 'en extra' sont les buts de 'scratch'(comme une extension de la pile RPN) qui peuvent être intégrés dans les utilisations normales du registre alpha pour les messages.. (c'est à dire entre ces usages.) Les exemples sont les fonctions indirectes indicées ('ISG M', 'DSE O', 'STO IND N'), les additions ('STO+M', 'STO-N', etc.), et le stockage temporaire de résultats numériques intermédiaires. Notez que les quatre registres alpha sont effacés simultanément par une fonction mono-octet 'CLA'. Les registres alpha (et les registres de la pile) ne peuvent être adressés indirectement que par le biais extraordinaire d'ajuster le contenu du registre c pour que l'un des registres d'état (n'importe lequel ira sauf le registre T) devienne le Roo, mais cela est rarement pratique.
Les fonctions d'accés au registre alpha combinées avec les fonctions standards alpha 'APPEND', 'ASTO', 'ARCL', 'ASHF' et 'CLA' donnent des manipulations de chaines de caractères alpha d'une vitesse et d'une fléxibilité beaucoup améliorées par rapport à ce qui est possible avec les fonctions standards seules. Considerez, comme premier exemple, le problème d'isoler un caractère particulier dans une chaine alpha, comme il peut être nécessaire dans une grande variété de jeux de recherches de mots. Voici une routine qui isolera (i.e., le laissera par lui même dans le registre alpha) le nième caractère (compté depuis la gauche) dans une chaine de 6 caractères max, en utilisant seulement des fonctions standards.

```
01*LBL A          10*LBL 01
02 7              11 "*"
03 -              12 ARCL Y
04 CHS            13 ASTO Y
05 1 E3           14 ASHF
06 /              15 ISG X
07 1              16 GTO 01
08 +              17 AVIEW
09 ASTO Y         18 .END.
```

(6C-1)

Pour utiliser la routine, le programme ou l'utilisateur place 'n' dans le registre X. Puis 'XEQ "A"' isole le nième caractère dans le registre alpha. Il y a deux problèmes avec cette routine, ce qui la rend moins que satisfaisante: d'abord, elle est relativement lente, demandant de 0.9 à 2.1 secondes pour aboutir, dépendant de 'n'; deuxièmement, elle n'est pas directement extensible à une chaine de plus de 6 caractères. Si les chaines à traiter peuvent être de plus de 6 caractères, le programme n'a aucun moyen de savoir où le 'premier' caractère est situé dans le registre alpha. Ce dernier problème peut être résolu dans une certaine mesure en numérotant les caractères de la droite vers la gauche pour que 'n=1' corresponde au dernier (le plus à droite) caractère dans la chaine alpha. Alors la chaine peut être cassée jusqu'à quatre chaines de 6 caractères, avec la chaine appropriée, dépendant de 'n', étant cherchée par la routine 6C-1

pour le caractère désiré. Mais, comme il avait été annoncé, l'utilisation des fonctions synthétiques donne une meilleure méthode.
les frontières invisibles entre les registres M,N, O et P simplifient le travail de découpe des chaines alpha. Tout ce que nous avons à faire est de trouver une procédure automatique pour bouger les chaines de façon que le caractère désiré soit à une de ces frontières. Considérons la séquence 'CLX', 'FIX 4', 'ARCL X'. Aprés l'éxécution de ces pas, le registre alpha contiendra son contenu original, maintenant déplacé vers la gauche par la concaténation des 6 caractères "0.0000". Si nous avions utilisé 'FIX 6' au lieu de 'FIX 4', la chaine originale aurait été déplacée de huit caractères. Ceci démontre une méthode non itérative (et de ce fait plus rapide) pour déplacer les chaines alpha d'un nombre variable de positions, qui est utilisé dans la version suivante de "ISO". Si vous éxécutez la routine en pas à pas, avec la HP 41C en mode ALPHA, vous verrez les caractères déplacés et effacés de manière sélective pour ne laisser qu'un seul caractère.

```
01*LBL "ISO"        08 X<> J
02 10               09 "↑"
03 -                10 X<> J
04 CHS              11 CLA
05 SCI IND X        12 STO [
06 ARCL X           13 AVIEW
07 CLX              14 END
```

"ISO"

30 OCTETS

Instructions pour "ISO":

1. Commencez avec une chaine de 10 caractères max dans le registre alpha.
2. Placez un nombre 'n' compris entre 1 et 10 dans le registre X.
3. XEQ "ISO".
4. A la fin, seul le nième caractère de la chaine originale reste. 'n ' est compté depuis la droite.

Cette routine est à la fois plus rapide et plus courte que la routine 6C-1, damandant seulement 0.8 secondes pour l'éxécution, indépendemment de 'n'. 'SCI IND X' est utilisé à la ligne 05 plutôt que 'FIX IND X', pour donner des mouvements de 4 (pour n=10) à 13 (pour n=1) caractères. "ISO" a le désavantage de changer le mode d'affichage de la HP 41C, mais ceci peut être corrigé moyennant quatre octets de programme supplémentaires en remplaçant les pas 05 et 06 par

```
05 X<>d
06 SCI IND d
07 ARCL d
08 X<>d
```

Des opérations similaires sont trouvées dans la routine suivante, "SUB", qui est utilisée pour remplacer un caractère dans un chaine alpha, laissant le reste de la chaine intact:

```
01*LBL "SUB"        12 "↑↑"             23 X<> J            34 ARCL X
02 10               13 X<> T            24 LASTX            35 R↑
03 -                14 X<> J            25 X<> ↑            36 STO d
04 CHS              15 "↑======"        26 X<> T            37 CLX
05 RCL d            16 CLX              27 9                38 X<> J
06 SCI IND Y        17 X<> \            28 -                39 STO [
07 ARCL Y           18 STO [            29 CHS              40 CLX
08 RCL ↑            19 CLX              30 FIX 0            41 X<> ↑
09 STO L            20 X<> J            31 RND              42 STO \
10 CLX              21 STO \            32 CF 29            43 AVIEW
11 X<> J            22 X<> T            33 10↑X             44 END
```

"SUB"

86 OCTETS

Instructions pour "SUB":

1. Commencez avec une chaine de 10 caractères max dans le registre alpha.
2. Placez un caractère alpha dans le registre Y.
3. Placez un nombre 'n' compris entre 1 et 10 dans X.
4. XEQ "SUB".
5. Aprés l'éxécution, le caractère en Y remplacera le nième caractère dans la chaine alpha. 'n' est compté à partir de la droite.

Dans les lignes 30-34 de "SUB", nous voyons un autre type de déplacement variable de caractères utilisant la fonction '10↑x' pour produire un nombre de 'x+1' caractères de long dans le registre X.
Le jeu du pendu (HANGMAN) listé ci-aprés démontre une application pratique de la manipulation de chaines rendue possible par les routines "ISO" et "SUB". Des versions de ces rouines sont aux lignes 169-183 et 114-168 respectivement.

```
01*LBL "HM"
02 0
03 STO d
04 .009
05 STO 07
06 FIX 0
07 SF 26
08 "WORD?"
09 AON
10 STOP
*11 "├
12 ASTO 00
13 ASHF
14 ASTO X
15 CLA
16 ARCL X
17 "├↑↑↑↑"
18 RCL \
19 CLA
20 STO [
21 ASTO 01
22 "---"
23 ASTO 03
24 ARCL 03
25 ASTO 02
26 CLA
27 ASTO 04
28 1.009
29 STO 08
30 STO 06
31 SF 19
32 " "
33 ASTO 05
34 GTO 02
35*LBL 01
36 1.009
37 STO 06
38 CLA
39 ARCL 02
40 ARCL 03
41 "├ "
42 ARCL 04
43 TONE 9
44 CLD
45 STOP
46 ASTO 05
47 RCL d
48 AVIEW
49 STO d
50*LBL 02
51 FS? IND 06
52 GTO 04
53 RCL 05
54 CLA
55 ARCL 00
56 ARCL 01
57 RCL 06
58 INT
59 XEQ 08
60 ASTO X
61 X=Y?
62 XEQ 03
63*LBL 04
64 ISG 06
65 GTO 02
66 FS?C 19
67 GTO 01
68 ISG 07
69 GTO 06
70 "ARRRRGGH..."
71 AVIEW
72 TONE 0
73 TONE 0
74 PSE
75 "WORD IS: "
76 ARCL 00
77 ARCL 01
78 AOFF
79 PROMPT
80*LBL 06
81 "'Γαβ*"
82 10
83 RCL 07
84 INT
85 -
86 XEQ 08
87 ASTO X
88 RCL 07
89 INT
90 "@@@@@@CL_"
91 XEQ 08
92 ARCL Y
93 ASTO 04
94 GTO 01
95*LBL 03
96 ISG 08
97 GTO 05
98 " **DONE**"
99 AVIEW
100 TONE 3
101 TONE 4
102 TONE 5
103 TONE 8
104 TONE 7
105 TONE 8
106 CLA
107 PSE
108 RCL 07
109 INT
110 ARCL X
111 "├ WRONG."
112 AOFF
113 PROMPT
114*LBL 05
115 SF IND 06
116 SF 19
117 RCL 06
118 INT
119 CLA
120 ARCL 02
121 ARCL 03
122 19
123 -
124 CHS
125 RCL d
126 SCI IND Y
127 ARCL Y
128 RCL ↑
129 STO L
130 CLX
131 X<> ]
132 "├↑"
133 X<> T
134 X<> ]
135 CLX
136 FIX 4
137 ARCL X
138 CLX
139 X<> \
140 STO [
141 CLX
142 X<> ]
143 STO \
144 X<> T
145 X<> ]
146 LASTX
147 X<> ↑
148 X<> T
149 9
150 -
151 CHS
152 FIX 0
153 RND
154 CF 29
155 10↑X
156 ARCL X
157 R↑
158 STO d
159 CLX
160 X<> ]
161 STO [
162 CLX
163 X<> ↑
164 STO \
165 ASTO 02
166 ASHF
167 ASTO 03
168 RTN
169*LBL 08
170 10
171 -
172 CHS
173 X<> d
174 SCI IND d
175 ARCL d
176 X<> d
177 CLX
178 X<> ]
179 "├↑"
180 X<> ]
181 CLA
182 STO [
183 END
```

"HM"
386 OCTETS
SIZE 009

* APPEND 11 SPACES

Instructions pour "HM":

1. XEQ "HM".
2. Le premier joueur entre un mot de 9 caractères max; R/S.
3. Au bip, l'affichage montrera autant de blancs "-" qu'il y a de lettres dans le mot inconnu. Le second joueur pense à une lettre en pressant la touche correspondante; R/S.
4. Au bip suivant, toutes les places de la lettre pensée seront montrées à l'affichage. Si la lettre n'est pas présente, un bout sera ajouté à la potence '@' ou à l'homme "Ŧ" à la droite de l'affichage. Le jeu reprend au pas 3.
5. Si le mot complet est trouvé en moins de 10 erreurs, "**DONE**" est affiché, suivi par le total d'erreurs.
6. A la dixième erreur, l'homme est pendu et le mot inconnu est affiché.

'HANG MAN' travaille avec des mots de 9 lettres au maximum. Si le premier joueur entre moins que neuf lettres, le programme remplit le mot avec des espaces (lignes 11-21), puis 'pense' le caractère espace, de la même façon que le ferait l'utilisateur, de façon à afficher le nombre correct de lettres inconnues pour le second joueur.
Quelques notes de programmation synthétique pour "HM": les lignes 72 et 73 sont "TONE 10", '9F 0A' en héxa, qui peuvent être crées avec le sauteur d'octets. La construction de la ligne 81, 'F5 60 06 04 05 01', a été décrite dans la section 5I; la ligne 90, 'F9 40 40 40 40 40 40 43 4C 5F', a été faite à titre d'exemple pour le sauteur d'octets amélioré dans la section 5G. Le truc utilisé pour faire faire des sauts de canard aux lettres supposées dans l'affichage (ligne 47-49) est décrit à la section 7B.

Le stockage en mémoire est organisé par "HM" comme suit:

Roo & Ro1	mot mystérieux
Ro2 & Ro3	mot supposé
Ro4	'pendu'
Ro5	lettre supposée
Ro6	compteur de boucle
Ro7	compteur d'erreurs
Ro8	compteur de bonnes suppositions

6D. RECONNAISSANCE DE CARACTERES

Bien que l'utilisateur ne puisse que regarder simplement un affichage alphanumérique pour lire son contenu, la HP 41C elle même n'a aucun moyen de savoir quel caractère, s'il y en a, est présent dans le registre alpha, sauf par une laborieuse comparaison pas par pas avec les caractères connus. De ce fait, par exemple, grouper par ordre alphabétique une chaine de données alpha est un processus prohibitivement lent et coûteux en mémoire. Cependant, les fonctions synthétiques peuvent étendre la capacité de la HP 41C dans le domaine du traitement des mots, en fournissant des conversions de caractères en chiffres et vice-versa.
Supposons que nous voulions identifier ou donner un équivalent numérique à un caractère alpha. Comme il y a 256 caractères (pas tous différenciés à l'affichage, bien sur), l'identification devra consister en un nombre décimal compris entre 0 et 255; i.e., l'équivalent décimal du code octet du caractère. La même conversion héxa/octal/décimal utilisée dans la section 6B peut être utilisée pour ce but, comme montré dans ce 'Caractère en Décimal' ("CD");

```
01♦LBL "CD"        10 SF 09
02 "┣♦♦♦♦Ē"        11 FS?C 11
03 X<> [           12 SF 10
04 X<> d           13 FS?C 12
05 FS?C 08         14 SF 11
06 SF 06           15 X<> d
07 FS?C 09         16 DEC
08 SF 07           17 END
09 FS?C 10
```

"CD"

43 OCTETS

(La ligne 02 est 'F6 7F 00 00 00 00 02', la même que dans le programme de la section 5E.)
Exemples: "A", XEQ "CD" donne '65'; "$", XEQ "CD" donne '36'.

Le procédé inverse, 'Décimal en Caractère' ("DC"), est un peu plus compliqué. Les lignes 03-06 s'assurent que les trois digits octaux du nombre entré vont toujours dans le même groupe de drapeaux de d, même si le nombre est un entier à un ou deux digits.

```
01♦LBL "DC"      11 SF 19          20 "⊢♦♦"
02 OCT           12 FS?C 17        21 CLX
03  E3           13 SF 18          22 STO \
04 /             14 FS?C 15        23 "⊢A"
05 10            15 SF 17          24 X<> \
06 +             16 FS?C 14        25 CLA
07 X<> d         17 SF 16          26 X<> [
08 FS?C 19       18 X<> d          27 AVIEW
09 SF 20         19 STO [          28 END
10 FS?C 18
```

"DC"

58 OCTETS

Exemples: '37', XEQ "DC" donne "%"; '64', XEQ "DC" donne "@".

Le problème de classer par ordre alphabétique un groupe de chaines de données alpha requiert un shéma de reconnaissance de caractères plus compliqué que celui donné par "CD". Puisque la seule comparaison alpha que peut faire la HP 41C est X=Y?, nous avons besoin d'équivalents numériques pour des chaines alpha entières de façon à faire la comparaison 'X<Y?' nécessaire au classement par ordre alphabétique. Une fois qu'une telle comparaison est faite, les techniques de tri des nombres standards peuvent être utilisées pour classer une liste de chaines alpha. Une manière directe de générer une telle équivalence serait d'utiliser "CD" sur chaque caractère de la chaine de donnée alpha et de combiner les résultats en un seul nombre. Notez que comme l'équivalent décimal de "Z" est 90, la valeur maximale d'une chaine de 6 caractères est $90^6 = 5.3\ E11$, qui est plus grand que le plus important entier que la HP 41C peut manipuler. De ce fait, cette conversion devra soustraire 64 de la valeur décimale de chaque caractère (pour avoir "A"=1, "B"=2, etc.) avant de faire la combinaison des six valeurs dans un seul nombre. La programmation synthétique offre une méthode de générer des équivalents numériques pour les chaines alpha qui est plus rapide et plus courte que la conversion caractère par caractère. la routine suivante "AL" classe par ordre alphabétique une simple paire de chaines alpha. Elle doit être combinée avec des routines de tri de nombres ordinaires, au choix de l'utilisateur, pour classer un groupe de chaines alpha.

```
01♦LBL "AL"      12 RTN                 22 RCL [
02 XEQ 01        13 X<> IND T           23 RTN
03 XEQ 01        14 X<> IND Z           24♦LBL 01
04 X≠Y?          15 X<> IND T           25 "×××"  (F2 01 01)
05 GTO 03        16 RTN                 26 ARCL IND Y
06 RDN           17♦LBL 02              27 "⊢♦♦♦"
07 RDN           18 "×××"  (F3 01 01 01)  28 ASTO [
08 XEQ 02        19 ARCL IND Y          29 "⊢♦♦"
09 XEQ 02        20 ASHF                30 RCL [
10♦LBL 03        21 "⊢♦♦"               31 END
11 X>Y?
```

"AL"

70 OCTETS

SIZE 002

Instructions pour "AL":

1. Les deux chaines alpha à classer doivent être dans des registres de données. Les chaines peuvent être de 1 à 6 caractères.
2. PLacez le numéro d'un registre de donnée en X, le numéro de l'autre en Y.
3. XEQ "AL".
4. "AL" place la première chaine dans l'ordre alphabétique dans le registre originalement désigné en Y; l'autre chaine va dans le registre originalement désigné en X.

"AL" utilise d'abord la sousroutine 01 (lignes 24-31) pour changer les quatre premiers caractères des deux chaines pour comparaison. Un 'nombre' est caractérisé par un premier nybble de '0' ou de '9'; bien plus, la comparaison numérique de chaines alpha n'a de sens que si les nombres ont même exposant. Ces deux considérations réstreignent la comparaison alpha à quatre caractères à la fois, puisque nous avons besoin de mettre un identificateur numérique ("AL" utilise '01') à la gauche de la chaine, et deux octets '00 00' à la droite pour standardiser les exposants. Ceci ne laisse que quatre octets libres dans un registre de 7 octets. Quatre caractères sont généralement suffisants pour distinguer deux chaines; si les deux octets restants des chaines de départ sont requis, la sousroutine 02 (lignes 17-23) fait la comparaison additionnelle.

Le truc de changer une chaine de donnée alpha en un nombre peut être renversé. La routine suivante, "MANT", montre comment un nombre peut être changé en caractères alpha pour utiliser des instructions alpha comme 'APPEND' ou 'ASTO' pour changer le nombre. Dans ce cas, nous voulons remplacer un nombre avec sa mantisse en retranchant son exposant. Nous pourrions faire cela en utilisant les fonctions 'LOG' et '10^X', mais cela occasionne des erreurs dans le dernier digit de la mantisse. "MANT" donne toujours le bon résultat. Aprés l'éxécution de "MANT", le nombre est remplacé par sa mantisse dans le registre X (y compris le signe); Y et Z ne sont pas affectés; T, L et le registre alpha sont perdus.

```
01♦LBL "MANT"      09 1 E50
02 STO [           10 *
03 CLX             11 LASTX
04 FIX 4           12 X>Y?
05 ARCL X          13 1/X
06 X<> [           14 /
07 "+"             15 FIX 9
08 X<> \           16 END
```

"MANT"

36 OCTETS

6E. LES LIGNES DE TEXTE SYNTHETIQUES ET L'IMPRIMANTE

Les lignes de texte synthétiques, créées par une des méthodes décrites dans les chapitres 3 & 5, sont particulièrement utiles pour les applications de l'imprimante. N'importe lequel des 128 caractères standards de l'imprimante peut être inclu dans une ligne de texte en plaçant l'octet correspondant (trouvé dans la table d'octets) dans la ligne. Ceci élimine la nécessité d'utiliser la fonction de l'imprimante 'ACCHR'. Par exemple, essayez d'écrire une routine qui imprimerait "Big Deal #7". En utilisant le drapeau 13 et 'ACCHR', vous aurez besoin d'un total de 40 octets. Mais le résultat attendu peut être obtenu en seulement 14 octets en écrivant une ligne de texte synthétique qui contienne explicitement les lettres des cases inférieures et le symbole "#":

```
01 "B■■ Dea■ #7
02 PRA
```

La ligne 01 est codée 'FB 42 69 67 20 44 65 61 6C 20 23 37'. Elle peut aisément être créée avec le sauteur d'octets ou l'autorisateur de texte:

```
01 +
02 *          ("B")
03 FRC        ("i")
04 X=0?       ("g")
05 RCL 00     (" ")
06 X<Y?       ("D")
07 LN1+X      ("e")
08 ABS        ("a")
09 HMS        ("l")
10 RCL 00     (" ")
11 RCL 03     ("#")
12 STO 07     ("7")
13 PRA
```

```
TE .001     (01"B■■ Dea■ #7")
Pressez la touche de correction 5 fois.
```

De la même manière, nous pouvons remplacer la fonction de l'imprimante 'BLDSPEC'. Considérons le caractère spécial graphique de la figure 6-1, où nous avons montré les points de l'exemple et les 'valeurs' et les nombres de colonnes d'impression comme ils sont appelés dans le Livre de l'utilisateur de l'imprimante 82143A, pp.64-66 pour l'instruction 'BLDSPEC'.

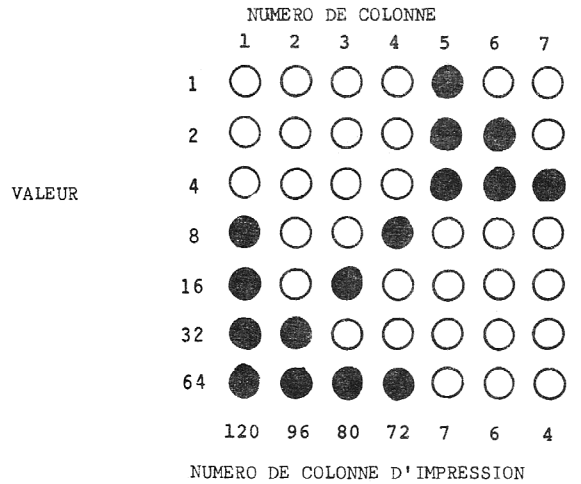


FIGURE 6-1 UN CARACTERE GRAPHIQUE SPECIAL

Les nombres de colonnes d'impression sont juste les équivalents décimaux des nombres faits en traitant chaque colonne comme un nombre binaire à 7 bits, avec les points noirs comptant comme des 1 et les points blancs comme des 0. La première fois que 'BLDSPEC' est éxécutée, avec zéro dans le registre Y et le premier nombre d'impression de colonne (120) en X, une chaine de donnée alpha est créée en X qui utilise les 7 octets de la première colonne à imprimer (1111000) comme les sept derniers bits de la chaine:

X= 0001000000000000000000000000000000000000000000000000/1111000

Quand 'BLDSPEC' est éxécutée pour le nombre d'impression de colonne suivant (96), ses sept bits sont copiés dans les derniers bits de X avec les précédents poussés vers la gauche:

X= 0001000000000000000000000000000000000000000000/1111000/1100000

Aprés que le dernier nombre d'impression de colonne soit entré, le registre X contient:

/ 120 / 96 / 80 / 72 / 7 / 6 / 4

X= 0001 000/1 1110 00/11 0000 0/101 0000/1001 000/0 0001 11/00 0011 0/000 0100

(hex) 1 1 E 3 0 5 0 9 0 1 C 3 0 4

Les nombres de colonne d'impression successifs sont montrés au dessus de la représentation binaire de X. Sous X, nous montrons les codes octets correspondant aux bits. Ce code octet est entré dans le programme comme une ligne de texte à 7 caractères 'F7 11 E3 05 09 01 C3 04' qui placera le code octet dans le registre alpha, d'où il pourra être transferré dans le registre X par 'RCL M'. Puis la fonction de l'imprimante 'BLDSPEC' peut être éxécutée normalement.
En pratique l'étape de calcul des nombres d'impression de colonnes n'est pas nécessaire. Tout ce que vous avez à faire est d'écrire un nombre binaire de 49 bits pour représenter la grille de points 7*7, en utilisant 1 pour les points noircis et 0 pour les blancs. Commencez avec le coin inférieur gauche de la grille comme le bit le plus à gauche, puis continuez jusqu'en haut de la première colonne, puis de bas en haut pour la seconde colonne, etc. Quand tous les points sont codés, ajoutez les 7 bits '0001000' à la gauche du nombre pour faire un total de 56 bits.

Groupez les 56 bits en digits de 4 bits, puis faites une ligne de texte de 7 caractères à partir des équivalents décimaux. Comme ces octets peuvent venir de n'importe où dans la table d'octets, l'utilisation de "CODE" plus le chargeur Q est une méthode idéale pour créer la ligne de texte désirée.
Cette procédure peut sembler compliquée, mais avec un peu de pratique, elle est à peine plus difficile que la méthode normale avec 'BLDSPEC'. Le gain par rapport à un programme normal est évident sur l'exemple suivant:

Programme 'normal'

```
01 0          10 BLDSPEC
02 ENTER      11 7
03 120        12 BLDSPEC
04 BLDSPEC    13 6
05 96         14 BLDSPEC
06 BLDSPEC    15 4
07 80         16 BLDSPEC
08 BLDSPEC    17 ACSPEC
09 72

     30 OCTETS
```

Programme synthétique

```
01 "▒▒▒▒▒▒▒"
02 RCL M
03 ACSPEC

   12 OCTETS
```

Bien sur, vous pouvez éxécuter la séquence 'BLDSPEC' normale manuellement et stocker le résultat, la chaine alpha de caractères spéciaux, dans un registre de donnée pour l'usage d'un programme, avec une utilisation finale de la mémoire (y compris le registre de donnée) de seulement 10 octets. Mais si le programme est lu depuis une carte magnétique, une carte de données doit aussi être lue; bien plus, le registre de donnée utilisé doit être gardé contre les utilisations par les autres programmes aussi longtemps que vous voulez utiliser le programme de caractères spéciaux.
Si vous vous attaquez à cette section, vous utilisez probablement une imprimante pour lister les programmes. Sur la sortie de l'imprimante, la ligne 01 du dernier programme synthétique décrit sera imprimé comme suit:

Il n'y a que cinq caractères montrés, parce que le listing d'une ligne de texte programme ne fera voir que les caractères de la moitié supérieure de la table d'octets. Les caractères correspondant aux octets de la moitié inférieure de la table sont invisibles. Bien plus, le tampon de l'imprimante utilise les octets des lignes A, B, D et E pour des opérations internes relatives aux impressions spéciales de caractères, instructions de simple et double largeur, etc..
De ce fait, les lignes de texte contenant des caractères de ces quatre lignes peuvent être imprimées d'étranges façons. Par exemple, si une ligne de texte contient le caractère correspondant à l'octet 'D5', un listing de programme contenant cette ligne aura toutes ses impressions qui suivent ce caractère en double largeur et en minuscules.

6F. NOMBRES NON NORMALISES ET CONTROLE DES DRAPEAUX

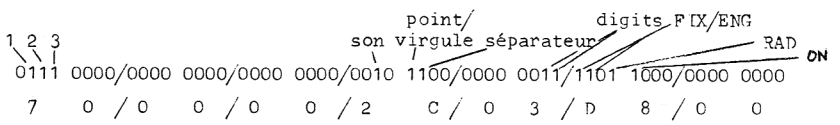
L'utilisation des lignes de texte synthétiques n'est en aucune façon restreinte à la génération programmée de chaines à caractères non standards dans le registre alpha. Une ligne de texte synthétique de 7 caractères max suivie par 'RCL M' placera un NNN dans le registre X. Une utilisation importante des NNN ainsi créés est le contrôle de masse des drapeaux par le stockage de NNN dans le registre d. Nous avons déjà vu une application du controle de masse des drapeaux dans l'utilisation de l'autorisateur de texte.
Les programmes décrits dans les sections précédentes contiennent de nombreux exemples de l'utilisation d'instructions comme 'X<>d' pour réstaurer le contenu initial d'un registre d'état (ex. le registre des drapeaux) aprés que le registre des drapeaux ait été utilisé comme codeur binaire. La capacité de créer n'importe quel NNN nous permet de lever ou de baisser les 56 drapeaux en une seule opération. La séquence de base est:

```
01 "xxxxxxx"
02 RCL M
03 STO d
```

(6F-1)

où "xxxxxxx" représente la ligne de texte synthétique utilisée pour générer le NNN. La routi-

ne 6F-1 prend 12 octets de programme, les mêmes qui aurait été nécessaires pour six lignes de programme 'SF mn' ou 'CF mn'. En général, de ce fait, il est plus performant d'utiliser la routine 6F-1 que des lignes 'SF' ou 'CF' individuelles quand plus de six drapeaux doivent être levés ou baissés. De telles occasions arrivent fréquemment dans les routines d'initialisation de programme, où des drapeaux divers sont levés ou baissés, et où des formats d'affichage et trigonométriques sont définis.
Comme illustration, écrivons une routine pour mettre les drapeaux de la HP 41C comme suit: Drapeaux 1, 2, 3, 26 (autorisation des sons), 28 (point ou virgule) et 29 (séparateur) sont levés; 'format FIX/ENG 3' (les drapeaux 38, 39, 40 et 41 sont levés); mode 'RAD' (drapeau 43 levé) mode continu ON (Drapeau 44); tous les autres drapeaux sont baissés. L'affichage 'FIX/ENG' est choisi particulièrement parce que c'est un format qui n'est pas disponible sans programmation synthétique. En format FIX ordinaire (drapeau 40 levé, drapeau 41 baissé), les nombres qui sont trop longs ou trop courts pour être affichés correctement font que l'affichage se met par défaut au format 'SCI'. Dans le format 'FIX/ENG', cependant, l'affichage par défaut est le mode 'ENG' Pour déterminer la ligne de texte synthétique requise pour générer l'état désiré des drapeaux, nous écrivons les états de tous les drapeaux comme un nombre binaire de 56 bits, avec des 1 pour les drapeaux levés et 0 pour les drapeaux baissés, puis nous groupons les bits en octets héxa à huit bits:



Nous voyons que la ligne de texte requise est 'F7 70 00 00 2C 03 D8 00'. Ce code octet particulier est un défi à chaque méthode de génération de lignes de texte synthétique que nous avons vue . Par exemple, parce que la ligne est longue de huit octets, elle ne peut pas être générée par une simple opération de "REG". Cependant, nous pouvons entrer dans un programme une ligne de texte de paille de 7 caractères et la faire tourner en mémoire en ajoutant ou en effaçant des octets plus haut en mémoire jusqu'à ce que les 7 octets des caractères soient positionnés tous dans le même registre (i.e., avec l'affichage programme montrant la ligne de paille, 'RCL b', 'XEQ "AD" doit donner une adresse commençant avec un octet numéro '1'). Puis nous pouvons utiliser "REG" pour stocker le code '70 00 00 2C 03 D8 00' dans le registre contenant les caractères de paille, en laissant l'octet 'F7' intact.
Il est inutile d'utiliser la sauteur d'octets ou l'autorisateur de texte pour éditer la ligne de texte à cause de l'octet 'D8' qui ne peut pas être rentré sous forme d'une ligne seule. De même, le chargeur Q ne fonctionnera pas à cause de l'octet '00' à la fin. Si nous utilisons le chargeur Q sur ce code, nous obtiendrons une ligne de texte à 6 caractères, puisque le chargeur Q ignore les nuls de tête dans le code d'entrée. Mais une combinaison du sauteur d'octets et du chargeur Q aboutira. Nous utilisons d'abord le chargeur Q avec le code '01 D8 03 2C 00 00 70', où nous avons remplacé l'octet '00' par un '01'. Puis nous sautons les octets jusqu'au '01' et nous l"effaçons.

	XEQ "CODE"	(CODE=?
"01 D8 03 2C 00 00 70"	R/S	("▯▯▯,▯▯▯")
	GTO le nouveau programme	
01 STO 07	PRGM off	
	STO Q	
	PRGM on	
	Pressez le chargeur Q	(02 9)
	DEL 001	(01 STO 07)
	PACK, SST	(02"▯▯▯,▯▯▯")
	JUMP .002	(02 LBL 00)
	DEL 001	(01 STO 07)
	DEL 001	(00 REG abc)

Nous finissons par ajouter les lignes '02 RCL M', '03 STO d' pour arriver à la routine 6F-1. Stocker des NNN dans le registre d est le seul moyen que nous avons pour lever beaucoup de drapeaux système, i.e., les drapeaux 30-35, 45-47 et 49-55. Bien que le contrôle de ces drapeaux conduise à des effets amusants mais non pratiques (voir section 7b), un exemple d'application pratique implique de lever le drapeau système d'entrée de donnée 45 (déjà rencontré à travers l'utilisation de l'autorisateur de texte).

Souvent, spécialement pendant les calculs qui utilisent les accumulations statistiques, nous devons entrer une longue chaine de nombres qui ne différent que par les deux derniers digits, comme '123456', '123457', '12360', etc. Pour ne pas avoir à entrer le '1234' chaque fois, nous pouvons ajouter quelques pas de programme qui ajoutent '1234' à notre entrée pour que nous n'ayons qu'à entrer les deux digits finaux de chaque nombre. Mais nous pouvons faire un processus encore plus 'amical' en demandant à la HP 41C d'entrer et d'afficher '1234' de telle manière que, quand nous entrons les deux digits finaux, nous voyons '1234' en même temps. Lever le drapeau 45 permet cette opération:

```
01 "■-"  (F2 84 00)
02 RCL M
03 X<>d
04 1234
05 STOP
06 X<>Y
07 STO d
```

(6F-2)

Le premier caractère de la ligne 01, l'octet '84', lève les drapeaux 40 et 45 quand a lieu le stockage dans le registre d. Quand le programme stoppe à la ligne 05, avec le drapeau 45 levé, le processeur pense qu'il est encore en entrée de données. Au stop , l'affichage montrera '1234' (registre X).Si nous pressons une touche de nombre, '5'par exemple, l'affichage deviendra '1234 5-' avec le souslignage indiquant qu'une entrée ultérieure est possible. Les lignes 06 et 07 sont en option; elles servent à réstaurer le contenu initial du registre des drapeaux, laissant le nombre nouvellement entré dans le registre X.
Le même truc utilisé dans la routine 6F-2 peut marcher pour une entrée de donnée alpha, d'où nous pouvons ajouter des caractères alpha à une chaine courante en affichant la chaine entière. Ou , pour mieux faire, nous pouvons ajouter des caractères à un message affiché, en ayant seulement les caractères nouvellement entrés restant en alpha pour la suite. Pour illustrer, remplacez les cinq premières lignes de "CODE" par cette séquence:

```
01 LBL "CODE"
02 "X■"      (F2 04 80)
04 RCL  M
05 "CODE="
06 AVIEW
07 CLA
08 STOP
09 X<>d
```

(6F-3)

Les lignes 02-04 lèvent les drapeaux 45 et 48 (alpha ON). Les lignes 05-07 font que, quand le programme s'arrête à la ligne 08, l'affichage montre "CODE=", même si le registre alpha a été effacé. Quand nous entrons des caractères alpha pour le code, ils entrent dans le registre alpha de façon normale mais ils apparaissent aussi à l'affichage concaténés à "CODE=". Si ,pendant la halte, nous effaçons l'affichage en mettant ALPHA off, puis on, le fantôme disparaitra, laissant seulement les caractères entrés à l'affichage. Ce n'est pas une grande victoire, mais cela rend la HP 41C plus accessible. Malheureusement, il ne semble pas y avoir (pour l'instant) de moyen d'ajouter des nombres à des prompt ALPHA.
Une note d'avertissement concernant l'utilisation des NNN dans la HP 41C. Les routines arithmétiques du calculateur sont destinées au maniement exclusif des nombres décimaux normaux. Leur utilisation avec des NNN peut amener des résultats surprenants et parfois désagréables. Par exemple, utilisez "CODE" pour générer le NNN '00 00 01 00 00 00 00' qui sera affiché en format SCI 5 comme '0.00010 E00'. Exécutez maintenant '1/X', et voyez ce qu'il se passe. L'affichage est blanc pendant environ 5 secondes durant lesquelles le clavier est 'verrouillé', i.e., la HP 41C ne répondra pas aux pressions de touches, y compris l'interrupteur 'ON'. Le dénominateur NNN a causé l'attente: l'opération de la division (ou 1/X) suppose que le dénominateur et le numérateur sont tous les deux écrits dans le format scientifique adéquat. La division est éxécutée par une série de soustractions; les deux exposants sont soustraits, puis le dénominateur est soustrait à répétition du numérateur; en effet, c'est l'inverse de la multiplication par additions successives. Le procédé ne demande pas longtemps si les mantisses du numérateur et du dénominateur sont dans le même ordre d'unités, comme ils devraient être, mais dans notre exemple, la mantisse du dénominateur n'est que de 0.0001, ce qui fait que 10^4 soustractions sont nécessaires pour achever la division. Cinq secondes, ce n'est pas long, mais un autre NNN pourrait facilement demander 5000 secondes ou plus pour la division. D'autres fonctions peuvent être plus longues: 'LOG (0.0001 E0)' demande 45 secondes, à comparer aux 5 secondes pour 1/X.

La 'normalisation' du contenu d'un registre qui se passe quand les fonctions de rappel des registres sont éxécutées est spécialement destinée à éliminer les dangers du blocage du calculateur à cause des NNN qui peuvent être introduits dans les registres de données quand un module de mémoire est inséré. Donc, soyez prudents. Comme pour les autres crash, enlever la batterie et la remettre débloquera la machine.

6G. LEVÉ DE RIDEAU

A cause de l'économie d'octets de programme associé aux utilisations des fonctions mono-octet 'STO' et 'RCL', il est souhaitable pour les programmes d'utiliser les registres de données Roo-R15 à chaque fois que possible. Ce ce fait, il est commun d'avoir beaucoup de programmes dans la mémoire de la HP 41C qui utilisent tous le même bloc de registres de données, et donc l'éxécution de l'un de ces programmes détruit les données utilisées par un autre. Une solution est d'écrire un programme de transfert de données qui bouge le contenu du bloc des registres de données dans un autre bloc, libérant le premier bloc de registres pour l'usage d'un autre programme. Si le nombre de registres utilisés est grand, ceci sera un procédé lent. Le programme "CU" offre une alternative plus rapide.

```
01*LBL "CU"      13 CLX          25*LBL 12       37 DSE [
02 STO L         14 LASTX        26 FC?C IND Y   38 GTO 11
03 CLX           15 INT          27 SF IND Y     39*LBL 14
04 RCL c         16 X=0?         28 FC? IND Y    40 X<> ]
05 STO [         17 GTO 14       29 CHS          41 X<> d
06 "┣++++"       18 2            30 X>0?         42 STO [
07 11            19 /            31 GTO 13       43 "┣ABC"
08 X<> [         20 RCL [        32 FC? IND Y    44 X<> \
09 X<> d         21 X<>Y         33 CHS          45 STO c
10 STO ]         22 FRC          34 DSE Y        46 RDN
11*LBL 11        23 X=0?         35 GTO 12       47 END
12 RDN           24 GTO 13       36*LBL 13
```

"CU"

87 OCTETS

Instructions

1. Entrer un entier dans le registre X.('n')
2. XEQ "CU".
3. Si n>0, Rn deviendra le nouveau Roo. Si n<0, R-n deviendra le nouveau Roo. Tous les autres registres de données bougeront derrière Roo.

"CU" prend un nombre entier dans le registre X,'n' ,(entré manuellement ou par un autre programme) et l'ajoute à l'adresse de Roo contenue dans le registre c. Si 'n' est positif, les registres de données Roo à Rn-1 seront 'transformés' en registres de programme, en levant le rideau imaginaire qui sépare les mémoires programme et de donnée de sa position initiale devant Roo jusqu'à sa nouvelle position devant Rn; Rn devient Roo. Si 'n' est négatif, le rideau est abaissé pour que 'n' registres de mémoire programme soient transformés en registres de données. Tout ceci se passe sans altération ou modification du contenu des registres qui bougent. Supposons que le 'programme 1' soit éxécuté, laissant des données dans les registres Roo-R5o qui sont requises pour une utilisation future. Mais entre temps, nous voulons utiliser le programme '2', qui utilise les registres Roo-R5o pour ses propres calculs. Dans ce cas, nous entrons '51' dans le registre X et nous éxécutons "CU" (la size doit être de 77 ou plus). Aprés l'éxécution du programme 2, nous nous préparons à une deuxième utilisation du programme 1 en pressant '-51', XEQ "CU".

** ATTENTION :Lever le rideau au dessus de la mémoire,i.e., éxécuter "CU" pour 'n' plus grand que la size courante, ou l'abaisser entre les adresses (hexa) '010' à 'OCO', ou à '000' causera un 'MEMORY LOST'.

"CU" travaille en effectuant une addition binaire du nombre 'n' aux digits héxadécimaux 9-11 de c qui constituent l'adresse du rideau. Les drapeaux correspondants 32-43 dans le registre d ne peuvent pas être contrôlés individuellement, donc le contenu du registre c est transféré dans le registre M et déplacé vers la gauche en ajoutant des nuls (ligne 06). Puis les lignes 08-09 placent l'adresse du rideau dans le registre d à la place des drapeaux 00-11.
L'addition binaire est un procédé trés simple. Pour ajouter 1 à un nombre binaire, nous changeons simplement la valeur du dernier bit, de 1à 0 ou vice-versa. Si le dernier bit devient un1 nous nous arrêtons. S'il devient un 0, nous échangeons le bit suivant à gauche. Si le bit suivant devient un 1, nous stoppons; s'il devient un 0, nous continuons au suivant à gauche, et ainsi de suite jusqu'à ce que nous stoppions à un bit qui change de 0 à 1. Soustraire 1 est à peu prés la même chose. Nous suivons la même procédure, commençant avec le bit le plus à droite

et continuant vers la gauche jusqu'à ce que nous rencontrions un bit qui change de 1 à 0. Ajouter 2 (10 en binaire) se fait de la même façon, mais nous commençons par l'avant dernier bit. En général, pour ajouter 2^m, nous commençons par le bit 'm+1', en comptant à partir de la droite. L'addition binaire est réalisée aux lignes 11-35 de "CU". Le nombre entré 'n' est cassé en bits binaires par divisions successives par 2 (lignes 18-19). Les bits successifs sont ajoutés ou soustraits des bits d'adresse selon le test de la ligne 30. Une fois que l'addition est complète, les trois octets du code du registre c qui sont dans le registre d sont joints aux quatre premiers octets qui attendent dans le registre N (lignes 41-42). Puis le code complet est poussé dans N (ligne 43), et finalement réstauré dans le registre c à la ligne 45. Quand "CU" est fini, les contenus des registres X et Y avant l'entrée de 'n' sont réstaurés.

La HP 41C opérera normalement pendant que le rideau est levé ou baissé depuis sa position établie en dernier par 'SIZE'. Cependant, si le rideau est levé, changeant les données en programmes, la mémoire ne doit pas être 'PACKée', puisque cela changerait de manière irréversible les données stockées auparavant en enlevant tous les octets nuls dans les données. Cette difficulté peut être évitée si un 'END' est placé au sommet de la mémoire programme, suivi par l'éxécution de 'PACK'. Si le rideau est levé par la suite, les registres de données transformées en mémoire programme ne seront affectées par aucun 'PACK'. Elles sont protégées par le 'END', qui a été codé pour indiquer un fichier compacté.

Une deuxième application de "CU" est de changer les données en programme de façon permanente, nous donnant un autre moyen de générer des lignes synthétiques de programme. Cette méthode est trés utile quand de nombreux registres consécutifs de programme, ou peut-être un programme entier, contiennent suffisamment de lignes synthétiques pour justifier d'être écrits entièrement par "CODE". Dans ce cas, nous utilisons "CODE" pour générer les codes octets pour chaque groupe de 7 octets de programme, stockant les codes successifs dans des registres adjacents de données. Les sept derniers octets vont dans R00, les sept avant derniers dans R01, etc. (Ceci, incidemment, est une justification majeure d'écrire "CODE" de façon à ce qu'il n'utilise pas de registres de données numérotés). Quand le codage est complet, nous utilisons "CU" pour lever le rideau au dessus du registre de données le plus haut qui contient le programme. Les codes synthétiques apparaitront alors comme des lignes de programme, commençant en haut de la mémoire programme. Pour accéder à ces nouvelles lignes, nous utilisons "CAT 1", en nous arrêtant au premier label global ou 'END', suivi par un 'RTN' manuel. Voici un exemple:

```
                        XEQ "CODE"              (CODE=?        )
"C000F400600401"
                        R/S                     ("▯▯▯▯▯▯▯"     )
                        STO 01
                        XEQ "CODE"              (CODE=?        )
"F32801297E8685"
                        R/S                     ("▯▯▯▯▯▯▯"     )
                        STO 00
2
                        XEQ "CU"
                        CAT 1,stopper au premier label ou END
                        RTN
                        SST pour voir le nouveau programme:

                        01 LBL " ▯▯▯"
                        02 "(▯)"
                        03 AVIEW
                        04 BEEP
                        05 RTN
```

A ce point, le label " ▯▯▯" n'apparaitra pas dans le catalogue utilisateur puisqu'il ne fait pas partie de la chaine globale. Ceci peut-être fait en insérant puis en effaçant une ligne temporaire de programme n'importe où parmi les nouvelles lignes, suivi par un PACK.

6H. LES PACS D'APPLICATION : PAR LA PORTE DE DERRIERE

Les modules d'application 'Read Only Memory' (ROM) de la HP 41C sont des moyens importants pour étendre la capacité mémoire de la machine en incluant une librairie extensible de routines pré-programmées. Malheureusement, beaucoup des routines souffrent de la limitation du fait qu'elles ne peuvent pas être appelées comme sousroutines automatiques par les programmes utilisateurs à cause des arrêts et des prompts pour les entrées/sorties inclus dans ces routines. Souvent,cette limitation peut être résolue en utilisant cette routine qui permet de se brancher à un point quelconque de la ROM:

```
01♦LBL "ROM"      11 X<> [          21 X<> ]
02 SF 01          12 X<> \          22 X<> a
03 X<> a          13 X<> [          23 X<> \
04 X<> \          14 ARCL 00        24 STO b
05 CLX            15 "⊢-----"       25♦LBL 00
06 RCL b          16 X<> ↑          26 X<> \
07 FC?C 01        17 X<> ]          27 CLA
08 GTO 00         18 "⊢++"          28 END
09 STO [          19 STO [
10 "⊢↑↑↑↑↑"       20 "⊢**"
```

"ROM"

73 OCTETS

SIZE 001

Instructions pour "ROM":
Avant l'éxécution de "ROM", stockez dans Roo l'adresse absolue du point dans la routine de la ROM où vous voulez que l'éxécution commence. Puis, votre programme doit appeler "ROM" comme une sous routine plutôt que d'appeler la routine de la ROM directement. La pile RPN et les registres de données doivent être configurés comme pour l'entrée dans la routine de la ROM au point choisi. "ROM" transfère l'éxécution à l'adresse spécifiée dans la ROM, suivi par la routine éxécutée normalement, retournant au programme principal original aprés la rencontre du 'RTN' ou 'END' final de la routine de la ROM. Bien qu'ordinairement un programme ROM appelé par son label global puisse être une routine de sixième niveau (i.e., pendant son éxécution, il peut y avoir jusqu'à six adresses dans la pile des retours), "ROM" ne peut être appelée que comme routine de 5^{e} niveau.
La perte d'un niveau de sousroutine vient de la manière dont "ROM" travaille. Aprés l'éxécution de la ligne 09, le registre alpha contiendra une copie de la pile des retours contenue dans les registres a et b:

R6 R5 R4 R3 R2 R1 A6

où 'A6' est l'adresse absolue du second octet de la ligne 06 (où le RCL b a été éxécuté); 'R1' est l'adresse de retour de la ligne de programme depuis laquelle "ROM" a été appelée comme sous routine; 'R2' est l'adresse de retour de la seconde routine, etc. Les lignes 10-20 remuent les caractères dans le registre alpha jusqu'à ce que les registres O et N contiennent une nouvelle pile de retour de la forme:

R5 R4 R3 R2 R1 ER A6

où 'ER' est l'adresse du point d'entrée dans la ROM, rappelé depuis Roo. Notez que l'adresse R6 a été perdue, comptant pour la perte d'un niveau de sous routine lors de l'éxécution de "ROM". La nouvelle pile des retours est stockée dans les registres a et b aux lignes 21-24. Lors de l'éxécution de la ligne 24 'STO b', 'A6' devient le pointeur d'adresses, et donc l'éxécution reprendra à la ligne 07. Cette fois ci, le drapeau 01 est baissé, donc le programme saute à la ligne '25 LBL 00'. Les lignes 26-27 complètent le 'ménage', réstaurant la pile RPN à son état lors de l'appel de "ROM". Le 'END' fait jouer la pile de retours, pour que 'ER', l'adresse dans la ROM, devienne le pointeur d'adresses,à partir duquel l'éxécution se transfère au programme de la ROM. Au 'RTN' ou 'END' rencontré là-bas, l'éxécution retourne au programme utilisateur à l'adresse d'appel 'R1'. Souvenez vous que la routine de la ROM peut elle même ajouter des niveaux de sous-routines, perdant ainsi 'R5', 'R4',etc.
La procédure pour déterminer l'adresse correcte dans la ROM du point d'entrée est assez simple. D'abord, éxécutez un 'GTO' n'importe quel label du programme de la ROM en question. Puis pres-

sez 'GTO .lmn' où 'lmn' est le numéro de ligne de la ligne où vous voulez que l'éxécution du programme soit transférée. Puis pressez 'RCL b', 'CLA', 'STO M', 'ASTO 00', qui place l'adresse en Roo, dans la forme correcte pour être utilisée par "ROM". Le choix de Roo est arbitraire; si ce registre est requis pour un autre but, n'importe quel registre de donnée peut être utilisé à condition de modifier la ligne 14 en conséquence.
Comme exemple de l'utilisation de "ROM", supposez que nous voulions utiliser le routine 'SSS' du Math Pac. Cette routine éxécute des prompts pour les entrées manuelles des longueurs des trois côtés d'un triangle, puis affiche les angles, les côtés, et l'aire du triangle. Si l'imprimante n'est pas connectée à la HP 41C, les sorties nécessitent des 'R/S' manuels pour poursuivre jusqu'à la 7ième donnée, empêchant l'utilisation de "SSS" comme routine automatique.
L'utilisation de "ROM" peut éliminer cette difficulté en appelant "SSS" en un point situé aprés les haltes d'entrées, et aussi aprés l'instruction 'SF 21' (ligne 65) qui provoque les haltes de sorties en l'absence de l'imprimante. Un point d'entrée correct est la ligne '06 LBL 05'. En ce point, le programme considère que les longueurs 'S1', 'S2', et 'S3' sont déjà dans Roo, Ro2 et Ro4 respectivement, donc nous devons nous arranger pour que le programme d'appel accomplisse ce stockage:

```
01♦LBL "MAIN"
02 25
03 STO 00
04 35
05 STO 02
06 45
07 STO 04
08 XEQ "ROM"
09 "DONE"
10 AVIEW
11 END
```

Dans ce programme d'exemple, "MAIN", nous incluons les longueurs explicites 25, 35 et 45. Notez que comme "SSS" utilise Roo, nous ne pouvons pas utiliser ce registre pour l'adresse de la ROM. Pour lancer le programme:

1. Changez la ligne 14 de "ROM" pour '14 ARCL 10'.
2. GTO ."SSS"
3. CLA
4. GTO 05 (ou GTO .006)
5. RCL b
6. STO M
7. ASTO 10
8. XEQ "MAIN"

Qaund "DONE" apparait à l'affichage, nous trouverons les résultats:

A1	95.74 dans Ro1
A2	33.56 dans Ro3
A3	50.70 dans Ro5
AIRE	435.31 dans X.

Pour les utilisations futures de "MAIN", les pas 1 à 7 peuvent être omis, pourvu que l'adresse de la ROM stockée dans R1o ne soit pas changée.

CHAPITRE 7

ANOMALIES AMUSANTES

Le but premier de la programmation synthétique est d'étendre la puissance de programmation de la HP 41C. Les programmes d'application du chapitre 6 sont les résultats d'une utilisation directe des fonctions synthétiques, plus une grande somme d'expériences, d'idées extravagantes, de recherches rapides, etc. Il ne devrait pas être surprenant que cette exploration du travail de la HP 41C ait aussi mis à jour un certain nombre de curiosités, qui n'ont pas d'utilisations pratiques importantes, mais qui sont amusantes à faire. Ce chapitre contient les descriptions de beaucoup de ces curiosités.

7A. 128 TONES ?

En opération normale, la HP 41C peut éxécuter les dix TONE de TONE 0 à TONE 9, correspondant aux codes octets '9F 00' à '9F 09'. Nous avons vu à la fin du chapitre 3, cependant, que le code '9F 0A' éxécute un nouveau TONE, plus long et plus bas en fréquence que les tons standards. '9F 0A' s'affiche en mode programme comme 'TONE 0', comme nous avons vu dans le programme 'HANGMAN' de la section 6C. Nous pouvons utiliser les techniques synthétiques de programmation pour attacher au préfixe '9F' n'importe lequel des 128 postfixes de la ligne 0 à la ligne 7 de la table d'octets. Cela révèle que presque chaque combinaison donne 1 ton différent. Il y a 16 fréquences, correspondant aux 16 valeurs possibles du second nybble du préfixe de la ligne de 'tone'. Les codes de TONE ne différant que par le premier nybble du postfixe produisent des tons de la même fréquence, mais généralement de durées variables.
En mode programme, une ligne 'TONE' avec un postfixe inférieur à 65 héxa (101 en décimal) sera affichée sous la forme 'TONE n',où 'n' est le second digit de l'équivalent décimal de l'octet de postfixe. Pour des postfixes supérieurs , les lignes seront affichées sous la forme 'TONE a' où 'a' est un seul caractère alpha de postfixe, comme 'TONE D' pour le code '9F 69' ou 'TONE P' pour le code '9F 78'. La table 7-1 montre les fréquences et les durées pour chacun des 128 postfixes possibles. Les données pour cette table ont été assemblées par Richard Nelson (PPC Calculator Journal, V7 N1 P21, 1980). Il y a quelques cas de duplication,ce qui fait qu'il n'y a en vérité que 114 sons différents.
De façon à expérimenter ces tons, vous pouvez éxécuter le programme "TON ", qui créera automatiquement 127 lignes programme de tons (toutes sauf 'TONE 0', '9F 00', qui peut être créé normalement). Aprés l'éxécution de "TON ", les 127 premières lignes du premier programme en mémoire seront des lignes de TONE, avec le numéro de ligne de chaque ligne étant le même que le numéro du TONE, de 1 à 127. "TON " appelle "DC" (section 6D) et "CU" (section 6C). Avant l'éxécution, faites 'SIZE 046' ou plus. Aprés l'éxécution, la size sera réduite de 43 registres.
Les tones synthétiques ne sont pas plus musicaux que les tons standards. Néanmoins, les fréquences additionnelles et la variation de la durée permettent des sorties sonores plus intéressantes pour la HP 41C.

```
01*LBL "TON          13 STO IND 44        24 "⊢"
02 .13               14 DSE 44            25 ASTO X
03 STO 43            15 GTO 01            26 ISG 43
04 42                16 "⊢"               27 RCL 43
05 STO 44            17 RCL [             28 INT
06*LBL 01            18 STO 00            29 XEQ "DC"
07 CLA               19 43                30 ASTO X
08 XEQ 03            20 XEQ "CU"          31 CLA
09 XEQ 03            21 BEEP              32 ARCL Y
10 XEQ 03            22 RTN               33 ARCL X
11 "⊢+"              23*LBL 03            34 END
12 RCL [
```

"TON "

78 OCTETS
SIZE 045

TABLE 7-1. FREQUENCES, NUMEROS ET DUREES (Sec) DES TONES

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
Freq. (Hz)	175	197	225	263	315	394	525	629	788	1051	105	113	121	131	143	158	
0	TONE 0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	0
	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	2.20	2.20	2.70	3.50	0.80	2.30	
1	TONE 6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	1
	2.00	0.34	1.50	0.33	0.50	1.00	0.45	0.84	0.30	0.55	5.00	3.50	2.00	4.10	0.30	2.40	
2	TONE 2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	2
	0.025	1.13	2.35	2.00	1.35	0.023	0.023	0.35	0.70	0.52	0.85	0.45	3.20	0.18	1.36	0.13	
3	TONE 8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	3
	0.54	0.37	2.10	1.95	0.28	0.15	0.80	0.77	0.65	0.058	0.42	0.41	3.30	0.39	0.97	0.30	
4	TONE 4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	4
	1.88	2.35	0.40	0.24	1.05	0.29	0.032	0.24	0.14	0.15	3.70	0.30	3.76	3.40	0.89	0.90	
5	TONE 0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	5
	0.085	0.22	1.75	0.74	0.28	1.25	0150	0.14	0.58	0.050	2.70	0.42	3.21	2.95	0.30	2.40	
6	TONE 6	7	8	9	0	1	A	B	C	D	E	F	G	H	I	J	6
	0.65	2.32	0.43	1.25	0.12	1.00	0.99	0.84	0.70	0.52	0.23	0.45	3.62	0.33	2.10	0.35	
7	TONE T	Z	Y	X	L	M	N	O	P	Q	⊢	a	b	c	d	e	7
	1.70	0.65	1.45	0.52	1.25	1.30	0.24	0.84	0.14	0.33	0.25	4.60	0.76	4.00	3.50	2.90	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

PREMIER DIGIT DE POSTFIXE

SECOND DIGIT DE POSTFIXE

7B. DES TRUCS AVEC LES DRAPEAUX SYSTEME

Dans notre développement de la programmation synthétique, nous avons rencontré de nombreux exemples de drapeaux système normalement inaccessibles volontairement levés ou baissés pour produire des résultats surprenants (comme le levage du drapeau de batterie défaillante,49 , dans la section 4D) ou utiles (l'autorisateur de texte, section 5H). D'autres effets amusants peuvent être produits en manoeuvrant les drapeaux stème. Voici un groupe de routines pour l'exploration du registre des drapeaux:

"SAVE"

16 OCTETS

SIZE 002

"RE"

14 OCTETS

SIZE 002

```
01♦LBL "SAVE"      10 RTN          19 SF IND X
02 0               11♦LBL "FL"     20 RCL d
03 RCL d           12 24           21 STO [
04 XEQ "CS"        13 -            22 "┣ABCD"
05 RTN             14 X<> d        23 X<> \
06♦LBL "RE"        15 STO [        24 STO d
07 0               16 "┣♦♦♦"       25 END
08 XEQ "CR"        17 RCL [
09 STO d           18 X<> d
```

"FL"

41 OCTETS

Avant d'expérimenter les drapeaux système, vous devez éxécuter "SAVE", qui rappelle le contenu courant du registre d et le stocke (en utilisant "CS") dans R00 et R01. Puis, n'importe quand, vous pouvez remettre votre calculateur dans son état initial de drapeaux en éxécutant "RE" (qui appelle "CR").

"FL" donne un moyen d'agir sur n'importe quel drapeau système (jusqu'au drapeau 53) en décalant le contenu du registre d vers la gauche, où le contrôle des drapeaux utilisateurs nous permet de positionner le bit dans la position 'décalée' du drapeau en question. Puis les octets du registre d sont décalés vers leur position de départ; quand le programme s'arrête, le drapeau choisi est levé. Par exemple, '49 XEQ "FL"' lève le drapeau de BAT; '47 XEQ "FL"' arrête l'éxécution avec le mode SHIFT actif (et la touche suivante pressée sera éxécutée comme une fonction shiftée). Lever le drapeau 30 produira un affichage inhabituel de 'catalog'. Ces catalogues fantômes n'ont pas d'application paticulière, mais il est intéressant de voir les 'entrées' variées de ces catalogues quand ils défilent. Selon Thomas Cadwallader, les différents catalogues sont accessibles en choisissant des nombres variés de format d'affichage avant de lever le drapeau de catalogue. Pour voir un de ces catalogues, essayez 'FIX 9, 30, XEQ "FL", R/S'. Notez que ces catalogues peuvent être arrétés et éxécutés en pas à pas comme n'importe quel catalogue normal.

Pour quelque but que ce soit, nous pouvons nous mettre en mode programme avec '52 XEQ "FL"'. Vous pouvez vous être demandé comment, dans les programmes comme "CODE" et "DECODE", nous pouvions être assez cavaliers pour stocker n'importe quoi dans le registre d; pourquoi, par exemple, la HP 41C ne se met-elle pas en mode programme quand le drapeau 52 est levé pendant qu'un programme tourne? La réponse réside dans le fait que le processeur ne vérifie les états des différents drapeaux qu' des moments particuliers, non continuellement, donc aussi longtemps que les drapeaux potentiellement dangereux sont baissés avant d'être vérifiés, rien de fâcheux ne se produit. Mais il y a certainement là des pièges; pour voir ce qui peut se passer, modifiez "FL" en insérant la ligne '25 1' aprés la ligne '24 STO d', puis pressez '52 XEQ "FL"'. Le processeur passe bien en mode PRGM, mais comme le programme tourne encore, la HP 41C commence à se programmer elle même, remplissant les espaces disponibles avec des lignes '1' jusqu'à ce que la mémoire soit pleine, à ce moment là l'affichage montre 'PACKING, TRY AGAIN'! De façon évidente, une des fois où le drapeau est vérifié est lors d'une ligne de programme d'entrée de nombre. Le drapeau 50, le drapeau des messages, est peut-être le plus intéressant des drapeaux système. Quand le drapeau est levé, tout ce qui se trouve dans l'affichage est 'enfermé dans les glaces' Pour voir quatre possibilités différentes, entrez (aprés avoir effacé la ligne '1' que nous avions entrée pour notre dernière expérience) une ligne '24 STOP' aprés la ligne '23 X<>N'. Puis essayez '50 XEQ "FL"', et observez l'affichage quand le programme s'arrête. Pressez et relachez rapidement 'SST'. Le nombre affiché (en format 'SCI 0') reste comme il est, bien que des indicateurs variés puissent changer lors du 'SST'. Presser la touche de correction retournera l'affichage à son format pré-"FL". Essayez maintenant '50 XEQ "FL"', 'SST' de nouveau, mais cette fois en pressant et en tenant la touche 'SST' pour voir '25 STO d', puis relachez; le '25 STO d' reste à l'affichage. Ensuite, faites '50 XEQ "FL"' et éxécutez 'STO d' manuellement (vous devrez mettre en mode USER pour accéder à l'assignation de touche de 'STO d'). Cette fois l'af-

fichage se figera sur 'XROM 05,62'. Enfin, faites encore '50 XEQ "FL"', puis pressez 'R/S'. Le canard reste sur son perchoir! Pour mettre le volatile en une autre position, insérez quelques lignes 'LBL 01' dans "FL" et effacez 'STOP', et éxécutez un autre '50 XEQ "FL"'.
Un abaissement judicieux du drapeau 50 peut aussi produire un résultat intéressant. Pendant qu'un programme tourne, si le drapeau 50 est baissé, l'affichage montre le canard volant. Cependant, si un 'VIEW nn' est éxécuté, ou un 'AVIEW', le drapeau 50 est levé et le contenu du registre spécifié est affiché. Un 'CLD' abaisse le drapeau 50, et renvoie le canard à l'affichage. Mais, si pendant un 'VIEW', nous abaissons le drapeau 50 sans avoir recours à 'CLD', le processeur reprend l'affichage par défaut, mais promène l'affichage 'VIEW' au lieu du canard. En effet, nous pouvons remplacer le canard par n'importe quel caractère ou chaine de 12 caractères maximum. Le moyen le plus simple pour accomplir ce truc est de faire éxécuter 'RCL d' au programme juste avant d'éxécuter la ligne 'VIEW', i.e., quand le drapeau 50 est baissé. Puis, immédiatement aprés que le 'VIEW' est éxécuté, réstaurer l'état des drapeaux antérieur au 'VIEW'.

```
01 "ABCD"    05 0
02 RCL d     06 LBL 01
03 AVIEW     07 SIN
04 STO d     08 GTO 01
```

Quand vous démarrez cette routine, vous voyez "ABCD" progresser pas à pas dans l'affichage. Les lignes 05-08 donnent une boucle infinie pour faire voler notre ersatz de canard. Les lignes 01-04 peuvent être incluses dans n'importe quel programme, utilisant n'importe quel affichage de 12 caractères pour personnaliser le programme qui tourne. Ce truc est utilisé dans le jeu du pendu 'HANGMAN' (programme de la section 6C) pour afficher la lettre supposée d'une nouvelle manière.
Voici une question test pour tester votre ingéniosité HP 41C. Il y a 216 segments LED indépendants dans l'affichage de la HP 41C; 12 caractères pleins fois 14 segments = 168 segments; plus 12 caractères point et virgule fois 3 segemnts = 36 segments; plus 12 segments de mots et de nombres dans les indicateurs; (168 + 36 + 12 = 216). La question: combien de segments peuvent être 'ON' simultanément, avec la HP 41C en mode attente, c'est à dire sans programme tournant mais 'on' ? Quand vous pensez avoir la réponse, et que vous pouvez l'étayer par un affichage réel, essayez la routine 'DI' . Si vous pouvez allumer plus de segments que "DI", vous avez appris plus que ce livre ne peut vous en dire.

```
01*LBL "DI"
02 "**8*"
.
03 RCL [
04 ":::"
05 ASTO Y
06 ARCL Y
07 ARCL Y
08 ARCL Y
09 AVIEW
10 XEQ d
11 END
```

(F7 F8 00 00 10 00 21 E8) Ligne 02

(F6 80 3A 80 3A 80 3A)

"DI"

37 OCTETS

7C. FAIRE VOLER LE CANARD A L'ENVERS

Pour envelopper l'exposé de la programmation synthétique, je voudrais vous donner encore un exemple de 'ils ont dit que ce n"était pas possible, mais nous l'avons fait (avec la programmation synthétique, bien sur!)'. En ce qui concernait la communauté des utilisateurs, le secret de la HP 41C le mieux gardé était l'existence du canard volant dans l'autre sens et le moyen de l'afficher. (Bien sur, il peut y avoir des secrets mieux gardés, mais ils sont encore secrets.
Pour amadouer cette créature timide dans l'affichage, nous avons besoin d'une dernière fonction synthétique, 'FIX 10'.
Les affichages numériques de la HP 41C sont contrôlés par les drapeaux 36-41. Le format 'FIX', par exemple, est établi quand le drapeau 40 est levé et quand le drapeau 41 est baissé. Les drapeaux 36-39 contrôlent le nombre de chiffres qui sont affichés aprés le point décimal. Les 4 drapeaux constituent un chiffre héxadécimal; le nombre de digits affichés est égal à la valeur de ce chiffre. Normalement, le format 'FIX' est choisi entre 'FIX 0' et 'FIX 9', d'où les valeurs correspondantes des quatre digits des drapeaux vont de '0000' à '1001'.
Avec les techniques de programmation synthétique, nous pavons placer n'importe quelle valeur dans les digits de drapeaux, étendant la série des formats 'FIX' de 'FIX 10' à 'FIX 15'. L'affichage ne peut montrer plus de 10 chiffres, bien sur;

en fait, les formats FIX 11 à FIX 15 produisent des affichages semblables au 'FIX 0'. Mais 'FIX 10' produit un nouveau format d'affichage. Pour des nombres à exposant positif, seuls les dix chiffres de la mantisse sont montrés avec l'exposant supprimé. De ce fait, par exemple, '1.234 567891 E56' sera affiché en 'FIX 10' sous la forme '1.234567891', alors qu'en 'FIX 9' il serait affiché sous la forme '1.2345678 E56'.
Il y a beaucoup de moyens pour placer la HP 41C en format 'FIX 10'. Comme suggéré ci dessus, nous pouvons avoir la valeur '10' (1010, ou 'A' héxa) pour les digits de drapeaux en stockant le NNN approprié dans le registre d. Un autre moyen facile est de mettre le format 'FIX 8', qui lève le drapeau 36 et baisse les drapeaux 37-39, puis utiliser '38 XEQ "FL"'. Ou encore la fonction synthétique 'FIX 10' (code 9C 0A) peut être utilisée, soit comme une ligne de programme (elle s'affiche comme 'FIX 0') ou en l'assignant à une touche utilisateur (les préfixe/postfixe pour "KA" sont 156/10). Enfin, nous pouvons placer le NNN '0A 00 00 00 00 00 00' dans le registre X et éxécuter 'FIX IND X'.
'FIX 10' n'a qu'une application pratique modérée comme moyen d'afficher seulement la mantisse d'un nombre avec un exposant positif. Malheureusement, la méthode n'est pas tout à fait parfaite :si l'exposant est 10, 11, 12 ou 13, pris en modulo 14, certains digits de la mantisse seront représentés à l'affichage par le caractère de la ligne 2 de la table plutôt que par le nombre approprié de la ligne 3. L'exemple le plus dramatique est quand l'exposant est 13 (ou 27, ou 41, etc.) et les drapeaux 28 et 29 baissés. Dans ce cas, seul le premier digit de la mantisse sera affiché normalement. '1.234567891 E13' sera affiché dans ce format comme '1"#*%&'(): ' ce qui est encore déchiffrable, si vous avez votre table d'octets à la main, mais pas vraiment commode.
L'affichage des caractères de la ligne 2 n'est en aucune façon restreint aux digits normaux '0' à '9'. En vous référant à la table d'octets, vous observerez que les octets '2C', '2E' et '3A' ont chacun deux caractères associés. En affichage alpha, ces octets sont toujours affichés comme les caractères ",", "." et ":", respectivement. Mais en affichage de nombres, les digits individuels du nombre sont chacun représenté par un caractère; voyez et contemplez, en un nombre avec l'exposant approprié, les digits de mantisse 'C' et 'E' sont représentés par les canards "≺" et "≻", respectivement. Le caractère du nombre '3A" est la caractère plein, comme nous avons vu à la section 5A.
Pour attraper un couple de canards, mettez en format 'FIX 10' par n'importe quelle méthode, et effacez les drapeaux 28 et 29. Puis:

"0100EC0C000013"	XEQ "CODE"	(CODE=?)
	R/S	(" ")
	Pressez ←	(1 ≻ ≺)

Un maximum de 9 caractères de canards peut être fait sur un simple affichage. Le premier caractère dans l'affichage du nombre sera toujours de la ligne 3; sans doute le plus innocent de tous est le caractère point-virgule (octet '3B'). Par exemple, le NNN '0B CC CC CC CC CC 13' s'affichera comme :

(≺≺≺≺≺≺≺≺≺)

Au risque d'en faire trop, retournons une fois encore à "CODE", le programme essentiel de la programmation synthétique. que pourrait-il y avoir de plus approprié que de faire voler le canard à l'envers pendant que "CODE" tourne? La ligne 07 de "CODE" (ligne 11 si vous avez modifié "CODE" comme suggéré dans la section 6F) est '07 " ABCDEFG"'. les sept caractères ajoutés peuvent être n'importe quels caractères; ils peuvent être aussi bien un NNN de 7 octets contenant un canard à l'envers. remplacez la ligne 07 par:

07 " "	F8 7F 0B C0 00 00 00 00 13
08 RCL d	
09 FIX 0	9C 0A
10 CF 28	
11 CF 29	
12 CF 21	(nécessaire seulement avec l'imprimante)
13 VIEW M	
14 STO d	

Alors que le canard vole à l'envers dans l'affichage, il sera poursuivi par un "≻". Peut-être est-ce une crotte de canard ??

* * *

C'est sur cette charmante note que nous arrivons à la fin de ce livre. Vous avez beaucoup tra-

vaillé, et beaucoup appris sur la HP 41C. De ce fait, la programmation synthétique devrait maintenant être une programmation normale pour vous. Vous pouvez maintenant vous appeler un UNN, un Utilisateur Non Normalisé !!!!!

APPENDICE 1

LES SYSTEMES DE NUMERATION

Chaque utilisateur de HP 41C est familiarisé avec le système de numération décimale, dans lequel la quantité fondamentale, ou 'base du système', est le nombre dix. Considérez le groupe suivant de lettres:

A B C D E F G H I J K L M

Si nous comptons les lettres, nous disons que nous avons 'treize' lettres; commme notation courte pour le nombre treize, nous écrirons, en décimal, :

13

La courte notation possible avec le système décimal de numération vient de l'usage répété d'un groupe limité de symboles, i.e., les ,nombres 0,1 ,2 ,3 ,4, 5, 6, 7, 8, 9, plutôt que d'avoir un symbole différent pour chaque nombre possible.Ouand nous écrivons le symbole double '13', la valeur représentée par chaque numéro dépend de sa position dans '13'. C'est à dire, '13' veut dire 'une fois dix plus trois'. Chaque nombre est multiplié par le nombre de base élevé à une puissance entière:

$$13 = (1*10^1) + (3*10^0)$$

Chaque numéro dans un nombre est appelé un chiffre (digit); nous disons que '13' est un nombre à deux chiffres. Pour un nombre à N chiffres:

$$abc....de = (a*10^{N-1}) + (b*10^{N-2}) + ... + (d*10^1) + (e*10^0)$$

les chiffres a, b,..., peuvent prendre des valeurs entre 0 et 9, i.e., jusqu'à un de moins que la base du système de numération, dix.
Il n'y a rien de sacré pour le mathématicien, cependant, dans le nombre dix. Nous pouvons aussi bien choisir un autre nombre comme base de numération. Par exemple, avec huit: en base huit, usuellement appelée 'base octale de numération' ou plus souvent 'octal', la valeur maximale d'un chiffre est sept. Le nombre treize est représenté par :

$$15_8 = (1*8^1) + (5*8^0) = 13_{10}$$

Ouand plus d'un système de numération est utilisé, les nombres de deux chiffres ou plus doivent être écrits avec des indices pour identifier la base dans laquelle ils sont écrits. Nous pouvons écrire des égalités comme:

$$15_8 = 13_{10}$$

$$1295_{10} = 2417_8$$

Les fonctions 'DEC' et 'OCT' de la HP 41C donnent une manière aisée de convertir des nombres dans les deux sens entre les bases décimale et octale. Il est important de se rappeler que des conversions de ce type ne changent pas le nombre, mais seulement les symboles utilisés pour le représenter. 7654_8 pommes restent la même quantité de pommes même si nous écrivons 4012_{10} pommes.
Deux autres systèmes de numération sont intéressants pour notre étude de la HP 41C. Le premier est le système de numération binaire dont la base est deux. Deux symboles seulement sont nécessaires, '1' et '0'. Notre nombre treize est représenté en binaire par

$$1101_2 = (1*2^3) + (1*2^2) + (0*2^1) + (1*2^0) = 13_{10}$$

Le système binaire est idéal pour l'utilisation dans les ordinateurs, car chaque digit ne peut prendre que deux valeurs, ce qui est aisé à réaliser mécaniquement ou électronniquement. Chaque digit, ou 'bit' comme 'binary digit' (chiffre binaire en anglais) comme ils sont appelés d'une façon générale, peut être représenté par l'état de n'importe quel interrupteur, où 'on' signifie '1', et 'Off', '0'. Tous les calculs de l'ordinateur sont effectués en binaire. La conversion en un affichage décimal n'est faite que pour le confort de l'utilisateur. En fait, même les nombres décimaux dans la HP 41C sont représentés à l'intérieur en 'Décimal Codé Binaire' ou 'DCB', où chaque digit décimal est codé avec quatre bits binaires. Par exemple, '13' est stocké sous la forme:

13 ⟶ 0001 0011

Chaque groupe de quatre bits peut représenter un nombre jusqu'à $15_{10} = 1111_2$, ce qui nous conduit au dernier système de numération que nous avons besoin de considérer, le système 'héxadécimal', de base seize. En notation héxadécimale, chaque chiffre peut prendre une valeur de zéro à 15, ce qui rend les symboles "0" à "9" insuffisants. Nous y ajoutons les symboles 'A' à 'F':

A = dix
B = onze
C = douze
D = treize
E = quatorze
F = quinze

Donc, par exemple,

$$8A_{16} = (8*16_{10}) + (10_{10}) = 138_{10}$$
$$1FF_{16} = 511_{10}$$

Notez que les valeurs des nombres à un chiffre ne sont jamais ambigües. Ce n'est que lorsque deux ou plus symboles sont combinés dans un nombre multi chiffres que nous avons besoin d'un indice pour spécifier le système de numération.

APPENDICE 2. CODES-BARRE DES PROGRAMMES

PROGRAM REGISTERS NEEDED: 28

ROW 1 (1 : 2)



ROW 2 (2 : 7)



ROW 3 (7 : 9)



ROW 4 (10 : 15)



ROW 5 (15 : 21)



ROW 6 (22 : 29)



ROW 7 (30 : 37)



ROW 8 (37 : 43)



ROW 9 (44 : 51)



ROW 10 (51 : 59)



ROW 11 (59 : 66)



ROW 12 (66 : 73)



ROW 13 (73 : 79)



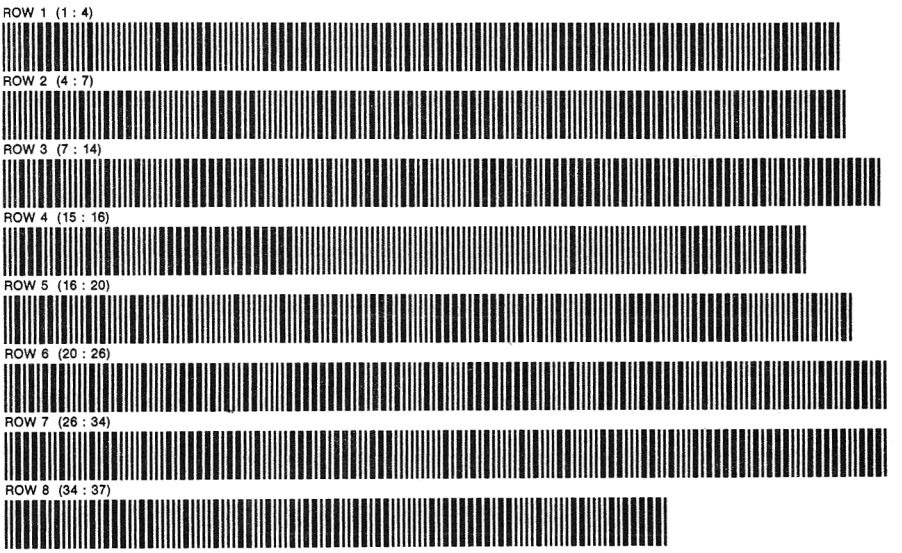
ROW 14 (79 : 84)



ROW 15 (85 : 89)



REGISTRES DE PROGRAMME NECESSAIRES: 15



Due à une erreur dans l'adaptation du programme de l'auteur pour les codes-barres, la ligne 34 de "REG" est '34 "B2-"' au lieu de '34 "REG"' comme décrit dans le texte. Le "B2" fait référence à la désignation du PPC du bogue hardware ('Bug 2') dans la HP 41C qui a amené la programmation synthétique. Vous pouvez, bien sur, changer la ligne de programme quand le programme est chargé.

KEY ASSIGNMENT PROGRAM

REGISTRES DE PROGRAMMES NECESSAIRES : 59

"KA" AND "EF"

ROW 1 (1 : 4)



ROW 2 (4 : 12)



ROW 3 (12 : 18)



ROW 4 (18 : 22)



ROW 5 (23 : 30)



ROW 6 (31 : 36)



ROW 7 (36 : 40)



ROW 8 (40 : 45)



ROW 9 (46 : 54)



ROW 10 (55 : 64)



ROW 11 (65 : 74)



ROW 12 (74 : 81)



ROW 13 (82 : 86)



ROW 14 (87 : 93)



ROW 15 (93 : 97)



ROW 16 (98 : 104)



ROW 17 (104 : 108)



ROW 18 (108 : 115)



ROW 19 (115 : 121)



ROW 20 (122 : 125)



ROW 21 (126 : 132)



ROW 22 (132 : 139)



ROW 23 (139 : 146)



ROW 24 (147 : 153)



ROW 25 (153 : 158)



ROW 26 (158 : 165)



ROW 27 (165 : 168)



ROW 28 (169 : 175)



ROW 29 (175 : 181)



ROW 30 (182 : 188)



ROW 31 (189 : 195)



ROW 32 (195 : 196)



REGISTRES DE PROGRAMME NECESSAIRES : 29

ROW 1 (1 : 3)



ROW 2 (3 : 9)



ROW 3 (10 : 17)



ROW 4 (17 : 23)



ROW 5 (24 : 30)



ROW 6 (30 : 36)



ROW 7 (36 : 42)



ROW 8 (43 : 48)



ROW 9 (48 : 54)



ROW 10 (54 : 62)



ROW 11 (63 : 70)



ROW 12 (70 : 77)



ROW 13 (77 : 85)



ROW 14 (85 : 92)



ROW 15 (92 : 99)



ROW 16 (99 : 103)



REGISTRES DE PROGRAMME NECESSAIRES: 56

ROW 1 (1 : 4)



ROW 2 (5 : 10)



ROW 3 (11 : 11)



ROW 4 (12 : 17)



ROW 5 (17 : 23)



ROW 6 (23 : 28)



ROW 7 (29 : 36)



ROW 8 (36 : 42)



ROW 9 (42 : 50)



ROW 10 (51 : 59)



ROW 11 (59 : 65)



ROW 12 (66 : 70)



ROW 13 (70 : 74)



ROW 14 (75 : 77)



ROW 15 (77 : 83)



ROW 16 (84 : 90)



ROW 17 (90 : 93)



ROW 18 (94 : 98)



ROW 19 (98 : 103)



ROW 20 (103 : 111)



ROW 21 (111 : 116)



ROW 22 (116 : 125)



ROW 23 (125 : 132)



ROW 24 (132 : 139)



ROW 25 (139 : 146)



ROW 26 (147 : 155)



ROW 27 (156 : 163)



ROW 28 (164 : 172)



ROW 29 (173 : 179)



ROW 30 (179 : 183)



APPENDICE 3

LA TABLE DE CODE BARRES POUR LES CARACTERES

Ce livre représente les résultats d'une année d'expérimentation avec la programmation synthétique pour la HP 41C. Maintenant, il est clair que le lecteur de code-barres sera un outil puissant pour la programmation synthétique, largement dû au travail de Jacob Schartz (un membre du PPC qui est aussi responsable du clavier de papier). La plupart des techniques du lecteur de code barres en sont encore à leur début, mais quelques exemples vous convaincront des promesses de leur projet .

** Utilisez toujours une feuille protectrice au dessus des codes quand vous procédez à des entrées avec le lecteur de code-barres**

Premièrement, vous pouvez ajouter à votre clavier de papier les codes barres pour le sauteur d'octets et le chargeur Q:

SAUTEUR D'OCTETS:



CHARGEUR Q:



La table de codes barres pour les caractères sur la page suivante a été fournie par Jacob Schwartz. N'importe quel caractère alpha non standard de la moitié supérieure de la table d'octets peut être directement ajouté à une chaine, soit dans le registre alpha, ou dans une ligne de programme, par entrée simple avec le lecteur de code barres à l'endroit approprié dans la table.

LA TABLE DES CODES-BARRE DES CARACTERES

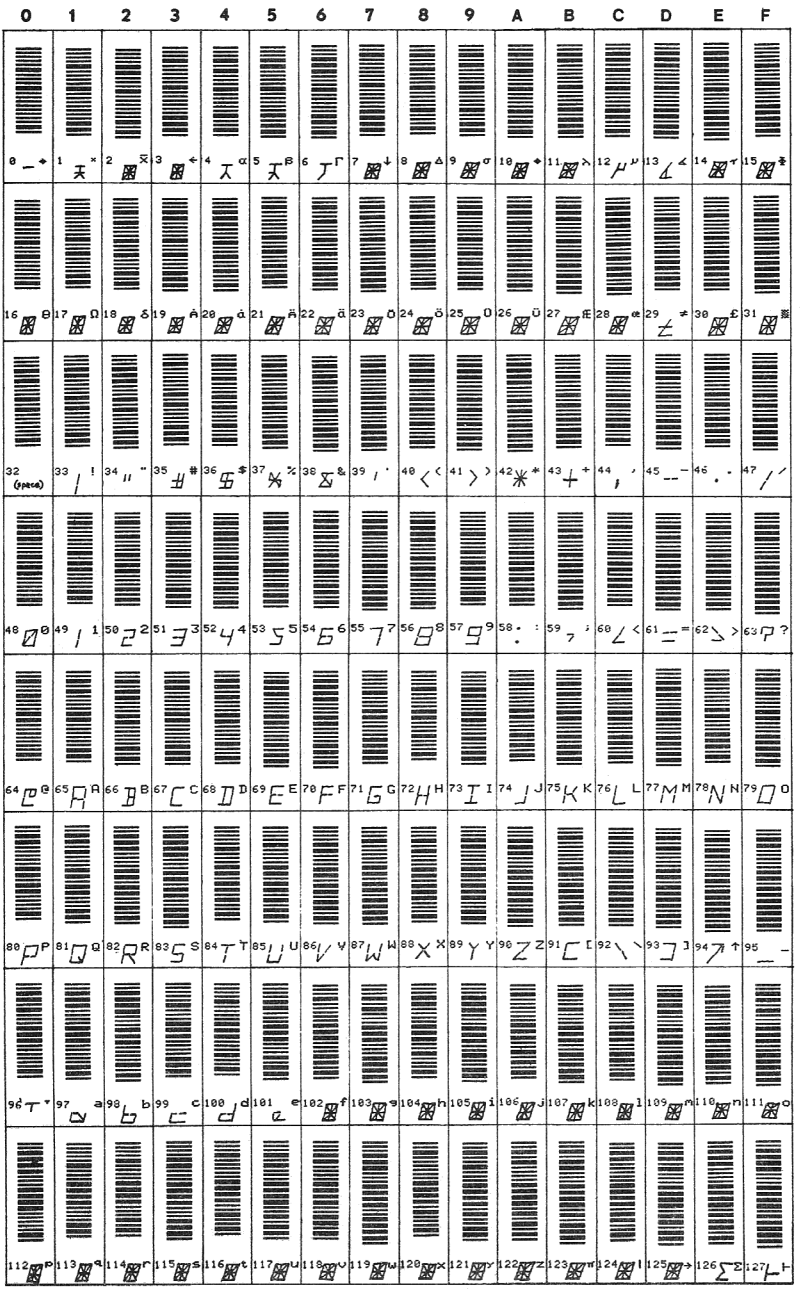


Table des matières

Page	Contenu
2	**Chapitre 1** : Les pourquois et les comments - 1A Programmation synthétique ? - 1B But et organisation - 1C L'origine de la programmation synthétique - 1D Pas de risque pour la HP-41 - 1E Quelques conventions - 1F Nécessaire - 1G Références.
7	**Chapitre 2** : A l'intérieur de la HP-41 - 2A Langage du calculateur - 2B La table d'octets - 2C Registre S.V.P. - 2D Répartition de la mémoire - 2E Les registres d'assignation de touches.
24	**Chapitre 3** : Edition exotique avec le sauteur d'octets - 3A Edition normale - 3B Le sauteur d'octet.
30	**Chapitre 4** : Les registres d'état - 4A Etranges postfixes - 4B Le registre alpha - 4C Le registre Q - 4D Le registre des drapeaux - 4E Les drapeaux d'assignation - 4F Le pointeur d'adresses et la pile de retour - 4G Le registre c et la répartition de la mémoire.
38	**Chapitre 5** : Des programmes pour la programmation - 5A Affichages inconvenants - 5B Echanges de registres et normalisation - 5C Démarons : "CODE" - 5D Accés direct aux registres programme - 5E Assignations de touches synthétiques - 5F Création de lignes de programme synthétiques - 5G Amélioration du sauteur d'octets (eJUMP) - 5H L'autorisateur de texte - 5I Le chargeur Q - 5J Parler à la HP-41 - 5K Code de stockage.
60	**Chapitre 6** : Applications - 6A Arriver au .END. - 6B Détection de la SIZE et autres trucs - 6C Jeux et humour dans le registre alpha - 6D Reconnaissance des caractères - 6E Les lignes de texte synthétiques et l'imprimante - 6F Nombres non normalisés et contrôle des drapeaux - 6G Levé de rideau - 6H Les pacs d'application : par la porte de derrière.
77	**Chapitre 7** : Anomalies amusantes - 7A 128 Tones ? - 7B Des trucs avec les drapeaux système - 7C Faire voler le canard à l'envers.
83	**Appendice I** : Les systèmes de numération.
85	Codes barre

Il était Octal, et les codes synthétiques
Furent balayés sans une perte.
Entrant et sortant du mode PRGM,
Les sauteurs d'octets grignotèrent le CMOS.

«Crains le STO c, mon fils,
Le MEMORY LOST, le blocage du clavier.
Crains les NNN, et évite
La curieuse horloge Phase 1.»

Il prit en main ses codes de boite noire,
Pensa longtemps au canard inversé;
La bête secrète du pays Achepé--
Toutes les recherches aboutissaient au néant.

Il fit un effort dément, et alors :
Le canard, sur l'écran posé,
Un saut pour chaque LBL 10,
Vint en couinant de gauche à droite!

STO b! STO d!, et RCL P!
Son clavier se mit à cliqueter.
Avec le bon code en mode numérique
Le canard vint, battant arrière.

«As tu trouvé la volaille fantôme?
Viens dans mes bras, mon fils binaire.
Laisse Corvallis écouter nos cris
Alors que nous gloussons de joie!»

Il était Octal, et les codes synthétiques
Furent balayés sans une perte.
Entrant et sortant du mode programme,
Les sauteurs d'octets grignotèrent le CMOS.

traduit de Bill Wickes (Keyboardlocky).
Ceci pour aider les non-anglicistes,
mais la moitié de la saveur de l'original est perdue.
(cf page finale de Synthétic...) J.D. Dodin.