INSIDE THE HP-41

by
Jean-Daniel Dodin

UNLOCK THE SECRETS OF YOUR HP-41!

Jean-Daniel Dodin

INSIDE THE HP-41C

Translated from French by:
Mary-Denise Dodin
and
John Vandenabbeele

Revised and Edited by:
Wilson W. Holes

Published by:
Keith Jarett
SYNTHETIX
P.O. Box 1080
Berkeley, California 94701
U.S.A.

ISBN 0-9612174-4-8
Library of Congress #84-51921

Also available from SYNTHETIX:

HP-41 Extended Functions Made Easy, by Keith Jarett
HP-41 Synthetic Programming Made Easy, by Keith Jarett
HP-41 Quick Reference Card for Synthetic Programming
ENTER (A Book for HP Series 1Ø Calculator Users), by Jean-Daniel Dodin, revised and expanded by Keith Jarett

French language publications on Hewlett-Packard calculators are available from:

Editions du Cagire
77 rue du Cagire
31100 Toulouse FRANCE

FOREWORD

This book is especially for owners of the HP-41C, HP-41CV and HP-41CX calculators. It has been written to help you better understand the operation of your machine. It contains few programs, and most of you will never have the opportunity to use some of the chapters. But after all, I'll never have the opportunity to climb Mount Everest, and yet I've enjoyed reading the account.

ABOUT THE AUTHOR

The author is French. He teaches drafting and engineering calculations at a high school in Toulouse, France.

He was introduced to Reverse Polish Notation in 1975, with the non-programmable HP-21 calculator. In 1979, when the HP-41C first became available, it was natural for him to order one. He learned about the PPC in early 1981 through Bill Wickes's book "Synthetic Programming on the HP-41C," and in September of the same year founded a PPC chapter in Toulouse. This chapter, with 400 members, was at the time the largest French speaking chapter. The author is also the editor of the French chapter newsletter, PPC-T.

DISCLAIMER

The material in this book is supplied without representation or warranty of any kind. The publisher, the editor, the translators, nor the author shall have any liability, consequential or otherwise, arising from the use of any material contained in this book.

TABLE OF CONTENTS

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

INTRODUCTION

ACKNOWLEDGEMENTS:

This book would not have been possible without the existence of PPC (Personal Programming Center) which gathers what surely must be the best Hewlett-Packard hand-held calculator experts. The amount of knowledge that the club has accumulated on the HP-41 is bewildering. Out of the roughly 6,ØØØ members of the club, many are those who have brought their stone to the building. The main workers themselves are too numerous for me to be able to quote all of them. I'll have to limit myself to the architects; the others must excuse me.

First, I am very grateful to Richard J. Nelson, the founder of PPC and publisher of the PPC Journal for many years. Without him, this work would not have been possible. Others are Bill Wickes, whose book on synthetic programming has opened my eyes on a new world; and finally John McGechie, the Australian Chapter Coordinator of PPC who was first to spread general knowledge on microcode.

And so many others...John Dearing, Paul Lind, Cary Reinstein, Robert Groom, Keith Jarett, Lynn Wilkins...Thanks to all of them.

Thanks also from the editor to all of those who helped produce this English version of "Au Fond de la 41C." Thanks to Mary-Denise Dodin and John Vandenabbeele for the translation of the French text and to Bobbi Stevenson, who spent many hours deciphering my scribbles while keying this book into the Wang word processor.

WHAT ARE WE GOING TO TALK ABOUT?

Chapter 2, "Geography," will give you a description of the structure of your HP-41, and of the various areas sharing the memory: like so many drawers being able to hold treasures.

Chapter 3, "Meaning of the Digits," will explain the number bases the HP-41 uses in its operating system and allow you to better understand the structure of the programs.

Chapter 4, "A Special Area," analyzes the main memory of the HP-41, the one which defines all the rest. The main memory is intended for the calculator's own use, but here you will be shown how to open and access it.

Chapter 5, "Thief!," will show you a particular method of building artificial statements which will allow you to solve most synthetic programming problems and will give you some examples of applications.

Chapter 6, "Microcode," will unveil the "Holy of the Holies," microcode. You will see the user statements and their operating mode.

Chapter 7, "Using Microcode," at last will give you some examples of programming in Microcode, taken from the calculator or written by the author.

Finally, the Appendix will try to answer some remaining questions.

WARNINGS:

The manipulations on software, whether it is normal, synthetic, or in microcode, present no risk to the HP-41. The worst which may happen to you is a one night unavailability for your calculator (Chapter 5).

It isn't the same thing with hardware modifications, which can lead to destruction of the central processing unit with a repair cost of about $80. That's why this kind of modification is not described here.

The content of this book is by no means guaranteed by Hewlett-Packard. Synthetic programming and microcode are not supported by Hewlett-Packard.

This is NOMAS (NOt MAnufacturer Supported).

This book won't teach you everything (far from it) about HP-41 programming, be it normal, synthetic, or microcode. Moreover, this isn't its purpose. It is a basic working tool from which everyone is free to elaborate one's own applications.

The best means to progress is to exchange one's store of knowledge with other people. This is the aim of the PPC club. Indeed, many users working singly rediscover every day facts that are already known, for the sources are often foreign or inaccessible. If you appreciate this book, PPC fits you.

CHAPTER 2

GEOGRAPHY

GEOGRAPHY

Our favorite calculator has many capabilities, consequently its internal structure is complicated and it is thus worth lingering over it.

We are all familiar with the HP-41C, the HP-41CV (C5), and now the HP-41CX (C1Ø). Although similar, each successive model has additional capabilities its predecessor did not. All that hasn't simplified anything.

2.1 Geography of the Hardware

The HP-41C is essentially made up of six independent parts (Figure 1).

1. The Display: comprised of liquid crystal cells and their control circuits including a control timer.

2. The Keyboard: made of a special printed circuit, it serves not only its obvious function but also as the link between the display, logic board and interface (port) connectors.

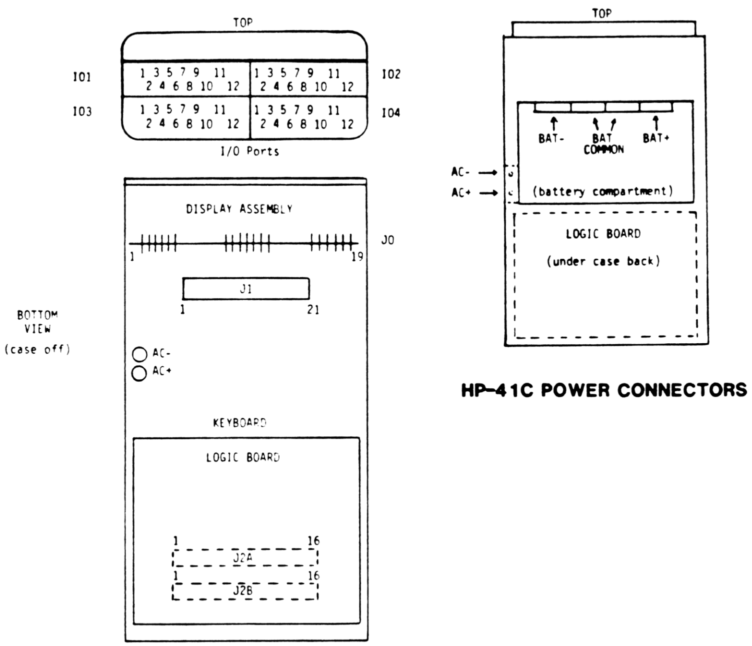
3. The Logic Board: the brain and memory of the HP-41, it is composed of a special microprocessor, internal Read Only Memory (ROM) and standard Random Access Memory (RAM), as well as a control timer and supplementary circuits. The new calculators, HP-41CV and HP-41CX, are expanded at this level.

4. Interface (Port) Connectors: These connecting circuits are visible in the ports at the upper part of the HP-41, and are mechanically independent and therefore easily interchangeable. In fact, they are made up of a flexible printed circuit; this circuit isn't soldered, but simply pressed against the printed circuit of the keyboard. This also serves as the connection to the power supply.

5. Power Supply: made up of either "N" size dry cells or "N" size rechargeable Ni-Cads recharged outside the calculator, or a battery pack of HP rechargeable Ni-Cads recharged either inside or outside the calculator.

6. The Case: made in three parts of sturdy plastic assembled with screws hidden under the rubber feet. The screws are threaded into the plastic, and you mustn't force them too much.

Taking account of the necessity of adding some components, the latest HP circuit boards use the so-called "piggy back" method consisting of setting two integrated circuits one upon the other.



HP-41C MAIN INTERCONNECTS

HP-41C POWER CONNECTORS

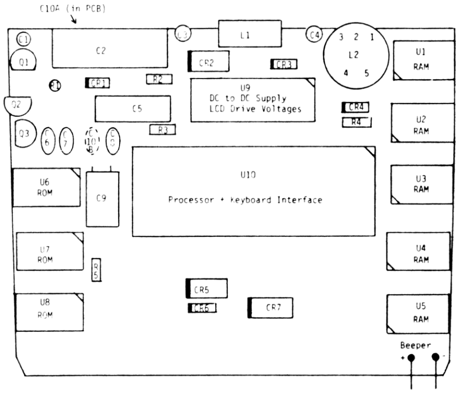
Structure of the HP-41C

FIGURE 1

The logic board reproduced here in Figure 2 was copied from a drawing made in October of 1979 and that of Figure 3 in October of 1982. Within three years, it has lost many of its main components. The transistors have all but disappeared. Is this the result of the new RESET mode (see Chapter 5) or of more advanced integration? The fact remains that there is much room for future components, some of which has been consumed in the new HP-41CX.

C = Condenser
R = Resistor
CR = Diode
L = Coil
U = Integrated Circuit

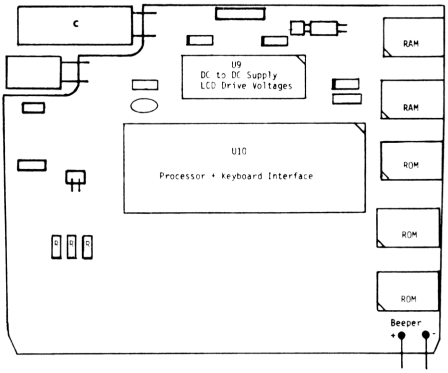
LOGIC BOARD COMPONENT LOCATIONS



1979 Logic Board

FIGURE 2

LOGIC BOARD COMPONENT LOCATIONS



1983 Logic Board

Taking account of the necessity of adding some components, the latest HP circuit boards use the so-called "piggy back" method consisting of setting two integrated circuits one upon the other.

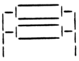


FIGURE 3

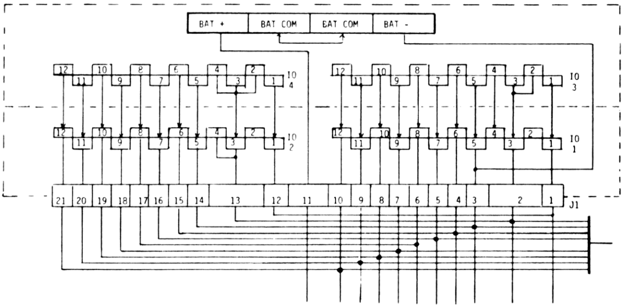
2.2 Electrical Structure

First, let's examine the port connectors. Notice particularly (Figures 4 and 5) the use of ports 2 and 4. These are not connected directly inside the HP-41 but only, possibly, at port 3. This is how the HP-41 recognizes the setting of the modules and numbers ports 1, 2, 3 and 4 (see the picture on the back of the HP-41).

Notice as well (Figures 1 and 5) the configuration of the power supply allowing the connection of external batteries or a 6V power supply. Beware, it is possible that on the future models this configuration may be modified.

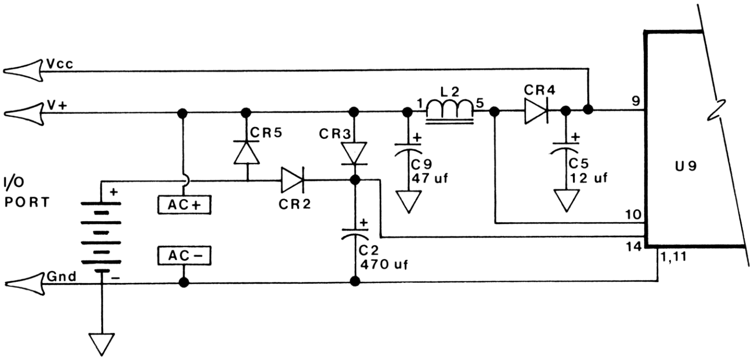
The early HP-41s were planned to receive a 6V power supply by the lateral port used by the battery charger. If you took off the small plastic overlay on older models, two yellow gold connectors were visible protruding from the plastic.

Unfortunately, Hewlett-Packard gave up the idea of making a 6V adaptor when its "battery pack" came out and since has suppressed those gold connectors as well as their plastic guide and the little contact springs. For some time now, the keyboard circuit has not been modified and still maintains the connections. The tinkerers need only to rebuild the gold connectors to be able to connect an external power supply. It is still possible on the latest models, but for how long?



Port Connector
Electrical Diagram

FIGURE 4



Power Supply Circuit Diagram

Component Functions

C2 Constant Memory (without batteries) Capacitor
C5 DC to DC Converter Output Filter
C9 DC to DC Converter Input Filter
CR2 Path from battery into C2, also blocks current from the AC adapter into the battery, protects against reversal.
CR3 Path from V+ to C2, keeps C2 from discharging to plug-ins or AC adapter.
CR4 Rectifies pulses from L2 to charge C5 for Vcc voltage.
CR5 Path from battery to plug-ins and to DC to DC converter, also blocks the current from the adapter into the batteries and protects the plug-ins from battery reversal.

FIGURE 5

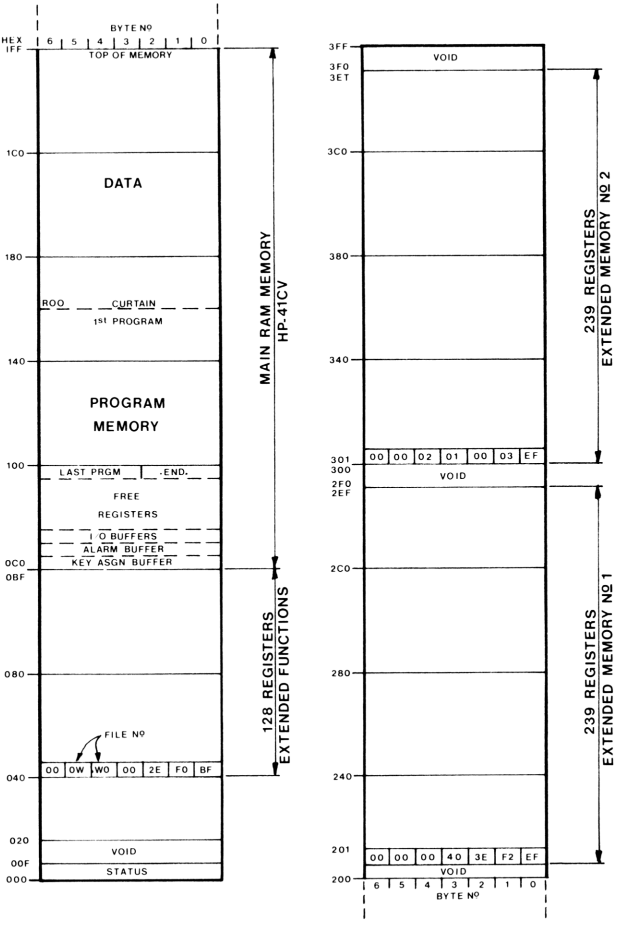
2.3 Geography of Random Access Memory (RAM)

All RAM memory of the HP-41 is now accessible to the shrewd user. It is no more a matter of hardware but a matter of software, but it is still geography.

The basic measure of the RAM unit in the HP-41 is a register. A register has as submultiples the byte, the nibble and the bit, (Appendix II). Unless explicitly mentioned, we will compute in hexadecimal (base 16).

The first register of RAM is R(ØØØ). This numbering has nothing to do with the one which you use with RCL and STO (see Chapter 4). The hexadecimal digit (hhh) will be called the "absolute address" of the register. The last register which is possible (in 1984...) is R(3FF). As 3FF = 1Ø23 decimal, there are therefore from Ø to 1Ø23 = 1Ø24 possible registers, that is to say exactly 1K registers and therefore with 7 bytes a register, 7K bytes. Alas, not all these registers exist.

A RAM map is included as Figure 6 which shows all defined areas of RAM.



RAM Memory Map

FIGURE 6

The following areas are part of RAM:

1 - Status Registers
2 - Empty Registers
3 - X-Memory
4 - Assignments
5 - Alarms
6 - Buffers
7 - Programs
8 - Data

2.3.1 Status Registers

From R(ØØØ) to R(ØØF) is the internal operating system status memory of the HP-41. Normally, you shouldn't have access to it, but we aren't normal people. These registers are so important that a whole chapter is devoted to them.

2.3.2 Empty Registers

Some locations have an address but not any content! On a standard HP-41, this is the case from R(Ø1Ø) through R(Ø1F) and from R(2ØØ) through R(3FF). The RAM of the extended functions/memory module occupies from R(Ø4Ø) to R(ØBF) (standard on the HP-41CX) and the X-memories above R(1FF). However, some empty registers still remain. They are not all useless, especially registers R(Ø1Ø) through R(Ø1F) which must stay empty so that the HP-41 can function as it does (see Chapter 7).

2.3.3 X-Memories

These are ordinary registers, but normally are accessed only by the special functions of the X-Functions Module. They are located from R(Ø4Ø) to R(ØBF) for memories being placed side by side with the X-Functions, from R(2Ø1) to R(2EF) and from R(3Ø1) to R(3EF) for all the possible modules of X-Memories.

If you observe a few limitations, you can extend their use and even make programs execute inside these registers. Yet this isn't their main use. Their complete study exceeds this work. We mustn't confuse them with the other RAM registers.

2.3.4 Assignments

These are registers holding the references to assigned keys. These registers start at R(ØCØ) and extend upward according to the number of assigned keys. They are organized as follows:

```
:F:Ø: : : : : : : : : : : : :
```

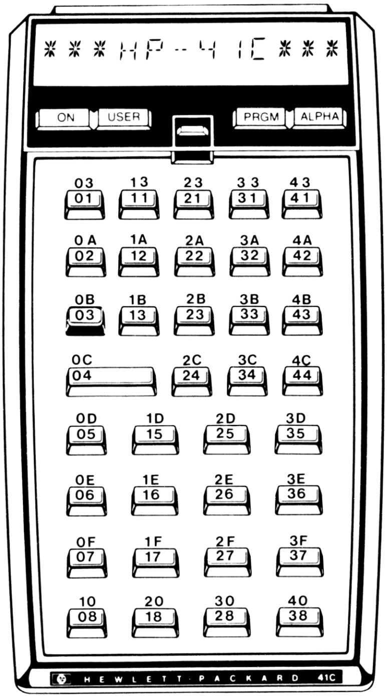
FØ is followed by a function code of 4 digits, + key code of 2 digits + function code + key code, to fill the entire 14 digits. There are therefore two assignments per register. These registers are filled from right to left and from bottom to top. The function code is the one described in Chapter 3. If the function uses only one byte, the 2 nibbles on the left of the function code are Ø4.

For example, assigning LN function to the key A gives the following (assuming no other assignment exists):

R(ØCØ) = :F:Ø:Ø:Ø:Ø:Ø:Ø:Ø:Ø:4:5:Ø:Ø:1:

If we assign another function, the HP-41 will first check to see if the left side of R(ØCØ) is free. If it is, it uses the latter as an available place. If not, the HP-41 pushes the assignments registers upwards and moves the content of R(ØCØ) to R(ØC1). R(ØCØ) then becomes available.

Figure 7 gives the code used in the assignment registers to represent the keys. This code is different from the one visible on the display.



Keycodes (hexadecimal) found in the assignment registers. Above the keys, the keycode of the shifted keys. Note that if one knows how to do it, one can assign a function to the SHIFT key (Ø3) and even to the shifted SHIFT key (ØB). To use this last one, we must get out of the User mode, press SHIFT, USER, and a second SHIFT. The "keyboard" code is the one expected (31 and -31). Try it after seeing KA (Chapter 4).

Key Assignment Keycodes

graphic assist by RJS

FIGURE 7

When you delete an assignment, the 3 bytes are not all suppressed at once, only the byte giving the key code is set to zero (and the key index in the registers ⊢ or e). This location is reused by a new assignment, but it isn't cleared by a PACK. However, it is cleared by PK from PPC ROM.

2.3.5 Alarms

For those who have the time module or a HP-41CX, they already know that the HP-41 uses the registers located above the assignment registers to store alarms and their messages. The registers used in this way are enclosed between an upper status register and a lower status register (see Appendix VIII).

2.3.6 The Buffer

Another type of memory is used by HP-41, mainly in a module called HP-IL development. A buffer is an intermediate memory intended to temporarily receive data, for example, during a transfer on the HP-IL loop between mass memory and the printer. The buffer normally is located above the assignment registers and above (or below, by chronological order) the alarms.

It is composed of a lower status register, an upper status register and between them the buffer itself. The lower status register looks like:

B B F 5 6 4 Ø

BB is (always in hex, don't forget) the reference mark of the register, as FØ is for the assignments and AA is for the alarms, F5 is the whole number of the registers, status included, here, 245 registers. 64Ø is the position of the pointer in the buffer. Here, the pointer points out the byte 64Ø (decimal 16ØØ). As the upper status register, it has for content:

:1:Ø:4:D:4:F:4:E:4:9:5:4:5:2:

This is an inside joke by Hewlett-Packard. The meaning can be easily deciphered by those of you familiar with the internal workings of the HP-41. The beginner will find the solution in Chapter 3.

2.3.7 Programs

The programs are located in an area above the three types of memories previously described. This area is located between two limits defined by addresses stored in the status registers of the HP-41 (see Chapter 4).

The lower limit is the address of the register containing the permanent .END. The statement .END. is the end of the last program in memory in the order of the CAT 1. It is located somewhere above the assignment-alarms-buffer area. Between the .END. and these specialized registers, there are available registers which are allocated according to need.

This allocation is automatic. In the case of an insufficient number of registers, this is usually shown by a message "PACKING" followed by "TRY AGAIN".

The upper limit of the program area is in fact the lower limit of the data area.

2.3.8 Data

The data registers are located between the program registers and the top point of the standard memory, varying with the number of RAM memory modules, with a maximum of 1FF.

There is no upper limit recorded. This is due to the ability to remove RAM memory modules without the knowledge of the calculator.

The lower limit is the absolute address of RØØ (notice: no parentheses around the digits), the register to which you have access by STO ØØ or RCL ØØ.

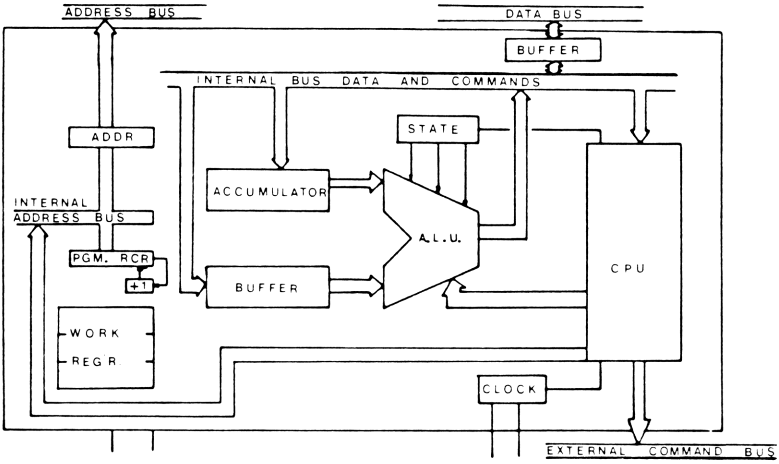
2.4 Central Processing Unit (CPU) Geography

How can you get your bearings in all of these memory areas? It all seems quite confusing. Yet the HP-41 does. It also knows how to perform the functions described in Chapter 3.

But what is that "IC"? It is an integrated electronic circuit whose principle is now well known and which is called the microprocessor.

This microprocessor is by itself a small calculator with RAM and ROM and special statements. The ROMs of the microprocessor are photo engraved, and we can't read them. So, we must be satisfied with observing the effects of the instructions, giving them a name and using them as best we can.

We will see in Chapter 4 how the RAM of the microprocessor is organized. Figure 8 gives you an idea of the standard scheme of the microprocessor. The HP-41 CPU considers the display and the RAM as peripherals, as well as the printer and the card reader. It takes its statements from ROM which is described next.



Geography of the Microprocessor

FIGURE 8

2.5 Organization of Read Only Memory (ROM)

If the RAM has the register as a unit, the ROM has the word as a unit. This is due to historical reasons rather than some grand design. The HP-41 inherited its microprocessor from earlier HP calculators. The instruction word of the HP-41's microprocessor is 1Ø bits, a byte with two more bits.

In hexadecimal, a byte is represented by two digits (for example: 3E). To represent the two supplementary bits of a ROM word, you must therefore use one more digit. The problem is to know if these two supplementary bits are taken on the left or on the right. Let's explain an example:

> The statement RTN, for the microprocessor must be coded:
>
> 11111ØØØØØ
>
> Given 1Ø bits, to represent them in hex, we can use 2 bits on the left, then 4 bits, then the last 4 bits, obtaining the 244 code. In this code, RTN is written:

```
11  111Ø  ØØØØ
 3    E    Ø        = 3EØ
```

Or we can start with the 4 leftmost bits, then the next 4 bits and let the two last ones from the right alone. We now have the 442 code:

```
1111  1ØØØ  ØØ
 F      8    Ø      = F8Ø
```

These words are counted one by one, there is no structure similar to the register. The 244 code is most commonly used, but you may encounter the 442 format as well. Just to make things more complicated, HP's annotated listings of the operating system use octal notation! The format is 1333.

To sum it all up, the processing unit of ROM is the word. The ROM is organized in columns of 16 words, in pages of 16 columns and 256 words and in modules of 16 pages or 4Ø96 words (4K). The microprocessor is able to recognize 16 different modules existing simultaneously in memory, numbered from Ø to F, or 65,536 words (64K). Since the introduction of the HP-41CX it is even possible to have "hidden" modules, that is, more than one module at a single address.

2.6 ROM Pages

Pages Ø, 1 and 2 are reserved for the functions built into the HP-41, it is the internal operating system. It's the content of these modules that makes the HP-41 what it is.

Page 3 is used only on the HP-41CX, and holds the X-Functions. This page is unused on the HP-41C and HP-41CV.

Page 4 is reserved for the service module. In its workshop, Hewlett-Packard uses a service module assigned to checking the operation of the HP-41 and is set to this address. In fact, the HP-41 tests for the presence of this module each time it is turned on. This module seizes control of the calculator and overrides the internal operating system.

ROM page 4 could be used by a computer expert to completely modify the operation of the HP-41. This address is also used by the HP-IL module. The address of the printing portion of this module is in binary Ø11Ø (6). When you use a non-IL printer, you must DISABLE the IL module with the switch located under the case of the module.

This switch simply changes (to Ø) one of the address lines. Transform Ø11Ø (6) into Ø1ØØ (4) and the address goes from 6 to 4. This works because the first words of this module, when read as instructions, lead to a return. The exact instructions are:

```
4ØØØ Ø1D                  Carry is not set
4ØØ1 Ø1B ?C GO Ø6Ø7       therefore doesn't jump
4ØØ2 ØØ7 JC +ØØ           doesn't jump
4ØØ3 1BB JNC +37          jumps and arrives on
.... ...
.... ...
4Ø3A 3EØ RTN              RTN
```

You will understand this better after reading Chapter 7.

Page 5 is the time module. The presence of this module is sometimes tested for by the HP-IL. Using this address should be avoided if the HP-IL is in memory unless also using the time module. The time module is present internally on the HP-41CX. On the HP-41CX this same address is used for the supplementary functions. There is a "hidden" module in the HP-41CX that can be selected in place of the time module.

Page 6 is the printer module (IL and non-IL); ports 3 and 4 of the HP-41 are often tested by the internal operating system for this module. If we remember this, we can use page 6, particularly for a video extension that could take into account these calls to the printer. This isn't possible now. Page 7 is the HP-IL, mass memory and controls. If the HP-IL is missing, you can use page 7 without any problem.

Pages (8 through F) are free except for Page E if the card reader is present, because the card reader always takes this page.

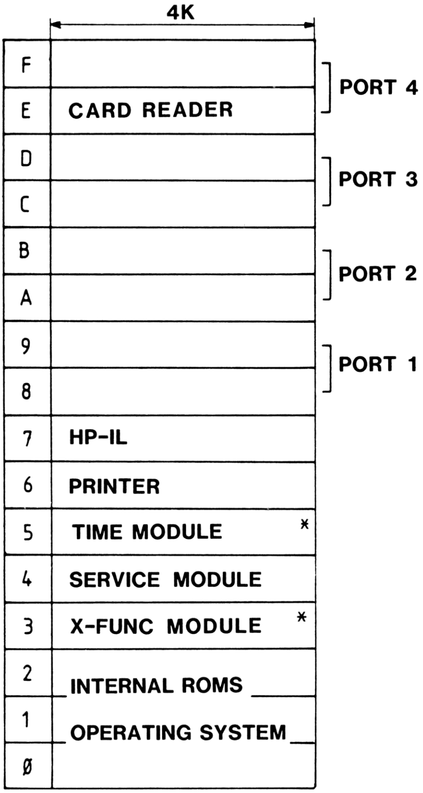
It's important to note this: The accessories listed above have their programs located at the addresses listed regardless of their physical location. The accessories may be plugged into any port, a port-extender, or (as in the case of the HP-41CX) built into the calculator.

All the other accessories have an address corresponding to that of the port in which they are physically located. If two accessories have the same address, nothing works. Most of the time, the HP-41 crashes when turned on. Therefore, it's important to note the ROM map of your HP-41 if you use a port-extender or an Eprom box.

The Eprom box, which we will describe further, is a special case since its address is internally switchable (refer to its owner's manual).

If you use a "lower" 4K module, its address will be 8, A, C or E depending on its physical location (port 1, 2, 3 or 4). Some modules are hard-wired to the "upper" addresses, that is 9, B, D, or F, i.e., the Auto Start/Duplication ROM and ZENROM-3B.

If it's an 8K module, it holds the addresses 8 and 9, A and B, C and D, or E and F, depending again on its physical port location.



* These pages consist of internal ROMs in the HP-41CX. Page 5 of the CX consists of two (2) 4K blocks, one being the hidden page referred to previously.

HP-41C ROM Pages

FIGURE 9

CHAPTER 3

MEANING OF THE DIGITS

MEANING OF THE DIGITS

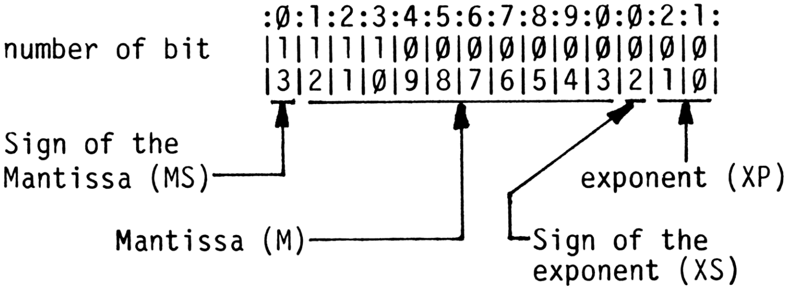
The HP-41C, like all other calculators, deals only with electronic signals. Whatever the memory is, these signals are in the form of digits Ø or 1. These digits are brought together into registers, bytes, or set of bytes.

3.1 NUMBERS OR LETTERS, NNN, NORMALIZATION

Some statements of the HP-41 act on a complete register. The most familiar are STO or RCL, but we have as well ASTO, ARCL, VIEW and AVIEW. The contents of these registers can be considered either as numbers or as letters.

a. An HP-41 register contains 7 bytes or 14 digits or 56 bits. These three ways of speaking designate the same contents.

b. We agree, for convenience, on giving some areas of a register a name connected with the representation of numbers and numbering the digits from 13 to Ø (from left to right). The number +1.23456789Ø 1Ø$^{+21}$, for example, is represented by:



```
                  :Ø:1:2:3:4:5:6:7:8:9:Ø:Ø:2:1:
number of bit     |1|1|1|1|Ø|Ø|Ø|Ø|Ø|Ø|Ø|Ø|Ø|Ø|
                  |3|2|1|Ø|9|8|7|6|5|4|3|2|1|Ø|
```

c. A register can contain:
-- a positive or negative number, with a positive or negative exponent
-- alphanumerical characters
-- another thing that we will call Non-Normalized Numbers (NNN)

d. If the digits in positions Ø and 1 and 3 to 12 are between Ø and 9, and if MS=Ø or 9 and XS=Ø or 9, the register contains a decimal number. In the MS and XS positions, the sign + is represented by Ø, the sign - by 9. Moreover, if the exponent is negative, it is represented by its complement. For example, to find the internal representation of E-21 (E for exponent of 1Ø) do 1ØØØ-21=979. The result (979) is the 3 rightmost digits (the sign is implicitly included):

+1.23456789Ø - 21 =

:Ø:1:2:3:4:5:6:7:8:9:Ø:9:7:9:

Let's notice as well that the number is always stored in SCI 9 format and that neither the decimal point nor the E of the exponent are coded. The FIX or ENG modes apply only to the display of a number.

e. If the first digit (position 13) is 1, the HP-41 knows it is not a number but rather an alpha string. The next six bytes are characters and are displayed as such. The null bytes present on the left and the right or in the middle of a string of characters are ignored when displayed, and the non-null characters are left justified.

f. Any other value of digits, for example a value above 9, yields a non-normalized number (NNN). A NNN can be stored and recalled easily in a status register. A NNN can be stored in a RAM register, but it is modified by the statements RCL, VIEW, etc. Beware, this modification intervenes not only on the recalled value, but also on the register itself. This modification is called normalization.

g. If the sign of the Mantissa is Ø or 9, the NNN is transformed into an ordinary decimal number, holding a digit above 9. A digit above 9, for example B, is then interpreted as its decimal counter value 11 or that of a Carry added to the digit on the left.

```
    1B ===> 21
```

If only the exponent sign (XS) is abnormal, the normalization of the digit will cause a carry. If so, a Ø remains in XS.

If the mantissa sign is not Ø or 9, the "normalization" is the same as replacing the value of the mantissa sign (MS) by 1. The register contents then are considered as alpha data.

Synthetic programming now permits us to avoid the inconveniences of normalization, and microcode functions have been created by the PPC Club members which also avoid normalization. (To use them, See Chapter 6.)

h. The NNN are in fact composed of hex digits. Under some conditions (FIX 9...) the HP-41 is able to display these hex digits. They are the available characters after digits Ø-9 in the hex table. It's called "natural notation."

NATURAL NOTATION

Hex	Display
A	⊠
B	;
C	<
D	=
E	>
F	?

3.2 CHARACTERS

a. We have seen that the HP-41 deals only with numbers, but these numbers are sometimes interpreted as characters, letters, flags or digits.

Why should we do in a simple way that which we can do in a complex way? Depending on the situation, the same character can be represented by one code or another.

For the moment, three cases are listed:

-- the display
-- the printers
-- microcode

The first two cases that any programmer may meet are indicated in the byte table. The third case is given in Figure 1Ø.

Let's notice that in the three cases these codes will be considered as characters only if the HP-41 has been "warned" that it must be so. That is to say:

-- When you use the function AON, the HP-41 then considers all that is found in the Alpha register represents characters.

-- When a normal register is called with the ARCL function, the register contents are transformed into the character equivalents and loaded into alpha.

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
2		⧗	"	#	$	%	&	'	(	)	*	+	←	—	→	/
3	Ø	1	2	3	4	5	6	7	8	9	⊠	,	∠	=	⦣	?
4	⊦	a	b	c	d	e	‾	ᵀ	Ʈ	Ϫ	Ѫ	₮	μ	Ⱡ	Σ	∡

DISPLAY

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
2		!	"	#	$	%	&	'	(	)	*	+	,	–	.	/
3	Ø	1	2	3	4	5	6	7	8	9	:	;	<		>	?
4	⊦	a	b	c	d	e	◆	ᵀ	Γ	α	β	×	μ	≠	Σ	∡

PRINTER

Microcode Character Set

REFERENCE: PPC-TN BY ROBERT GROOM & JOHN MC GECHIE

FIGURE 10

Do you understand now that, in any register:

1Ø 4D 4F 4E 49 54 52

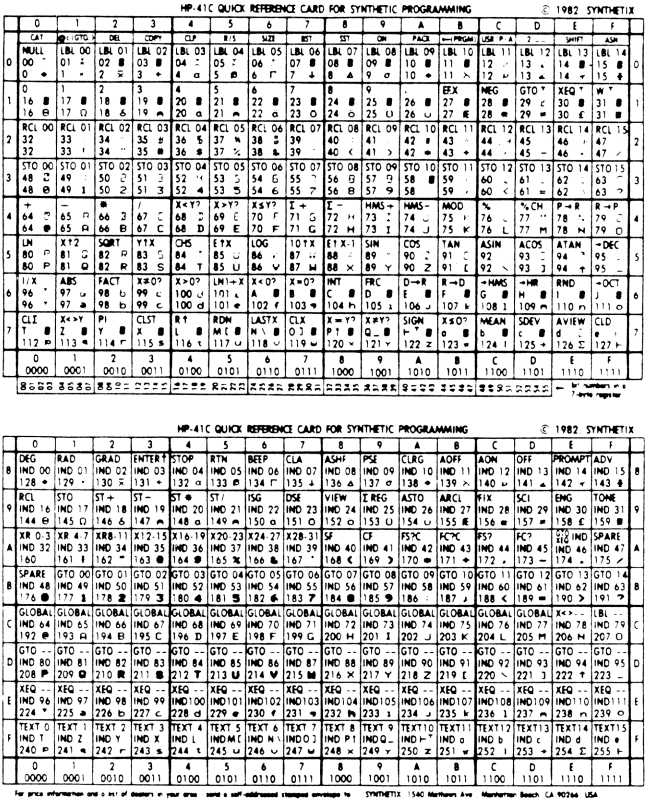
means the following:

1Ø: characters are all alphanumerical
4D: M If we could recall this regis-
4F: O ter we would have MONITR in
4E: N the display. In fact, if you
49: I have the means of interacting
54: T with the buffer, you probably
52: R also have access to the microcode functions of the PPC-ROM and monitor EPROMs.

We must notice that the initial version of the HP-IL development, released on EPROM, was called the HP-IL MONITOR.

In a master cleared calculator (MEMORY LOST) do: 5, BSIZEX (SETBUFX with the MONITOR EPROM), followed by either 194 CHS NRCL (Microcode) or 194 RX (PPC ROM), and you will see "MONITR" in the display.

b. The byte table (Figure 11) is a summary of the most current meanings of the HP-41 codes. It is organized so that every possible value able to be assumed by a byte is represented.



HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING © 1982 SYNTHETIX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	CAT	●:(GTO.)	DEL	COPY	CLP	R/S	SIZE	BST	SST	ON	PACK	←(PRGM)	USR P·A	? _ _	SHIFT	ASN	
0	NULL 00 ¯ 0 ◆	LBL 00 01 ẋ 1 ·	LBL 01 02 ▮ 2 x̄	LBL 02 03 ▮ 3 ←	LBL 03 04 ⁚ 4 α	LBL 04 05 ⁚ 5 β	LBL 05 06 ˙ 6 Γ	LBL 06 07 ▮ 7 ↓	LBL 07 08 ▮ 8 Δ	LBL 08 09 ▮ 9 σ	LBL 09 10 ▮ 10 ◆	LBL 10 11 ▮ 11 λ	LBL 11 12 ⁄ 12 μ	LBL 12 13 . 13 ∡	LBL 13 14 ▮ 14 τ	LBL 14 15 ▮ 15 Φ	0
1	0 16 ▮ 16 Θ	1 17 ▮ 17 Ω	2 18 ▮ 18 δ	3 19 ▮ 19 Å	4 20 ▮ 20 a	5 21 ▮ 21 Ä	6 22 ▮ 22 ä	7 23 ▮ 23 Ö	8 24 ▮ 24 ö	9 25 ▮ 25 Ü	. 26 ▮ 26 ü	EEX 27 ▮ 27 Æ	NEG 28 ▮ 28 œ	GTO ˙ 29 ≠ 29 ≠	XEQ ˙ 30 ▮ 30 £	W ˙ 31 ▮ 31 ▒	1
2	RCL 00 32 32	RCL 01 33 33 !	RCL 02 34 " 34 "	RCL 03 35 # 35 #	RCL 04 36 $ 36 $	RCL 05 37 % 37 %	RCL 06 38 & 38 &	RCL 07 39 39 '	RCL 08 40 : 40 (	RCL 09 41 41)	RCL 10 42 * 42 *	RCL 11 43 ⁘ 43 +	RCL 12 44 ˏ 44 ,	RCL 13 45 45 −	RCL 14 46 ⁄ 46 .	RCL 15 47 47 /	2
3	STO 00 48 0 48 0	STO 01 49 : 49 1	STO 02 50 2 50 2	STO 03 51 3 51 3	STO 04 52 4 52 4	STO 05 53 5 53 5	STO 06 54 6 54 6	STO 07 55 7 55 7	STO 08 56 8 56 8	STO 09 57 9 57 9	STO 10 58 ▮ 58	STO 11 59 ; 59 ;	STO 12 60 ⌊ 60 <	STO 13 61 _ 61 =	STO 14 62 ⁻ 62 >	STO 15 63 ⌝ 63 ?	3
4	+ 64 @ 64 @	− 65 A 65 A	* 66 B 66 B	/ 67 C 67 C	X<Y? 68 D 68 D	X>Y? 69 E 69 E	X≤Y? 70 F 70 F	Σ+ 71 G 71 G	Σ− 72 H 72 H	HMS+ 73 I 73 I	HMS− 74 J 74 J	MOD 75 K 75 K	% 76 L 76 L	%CH 77 M 77 M	P→R 78 N 78 N	R→P 79 O 79 O	4
5	LN 80 P 80 P	X↑2 81 Q 81 Q	SQRT 82 R 82 R	Y↑X 83 S 83 S	CHS 84 T 84 T	E↑X 85 U 85 U	LOG 86 V 86 V	10↑X 87 W 87 W	E↑X-1 88 X 88 X	SIN 89 Y 89 Y	COS 90 Z 90 Z	TAN 91 [91 [	ASIN 92 92 \	ACOS 93] 93]	ATAN 94 ↑ 94 ↑	→DEC 95 _ 95 _	5
6	1/X 96 ˙ 96 ˙	ABS 97 a 97 a	FACT 98 b 98 b	X≠0? 99 c 99 c	X>0? 100 d 100 d	LN1+X 101 e 101 e	X<0? A ▮ 102 f	X=0? B ▮ 103 g	INT C ▮ 104 h	FRC D ▮ 105 i	D→R E ▮ 106 j	R→D F ▮ 107 k	→HMS G ▮ 108 l	→HR H ▮ 109 m	RND I ▮ 110 n	→OCT J ▮ 111 o	6
7	CLΣ T ▮ 112 p	X<>Y Z ▮ 113 q	PI Y ▮ 114 r	CLST X ▮ 115 s	R↑ L ▮ 116 t	RDN M [▮ 117 u	LASTX N \ ▮ 118 v	CLX O] ▮ 119 w	X=Y? P ↑ ▮ 120 x	X≠Y? Q _ ▮ 121 y	SIGN ⊢ ˙ ▮ 122 z	X≤0? a ▮ 123 π	MEAN b ▮ 124 |	SDEV c ▮ 125 →	AVIEW d 126 Σ	CLD e › 127 ⊢	7
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	
	00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31	32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	48 49 50 51	52 53 54 55	← bit numbers in a 7-byte register		

HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING © 1982 SYNTHETIX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	DEG IND 00 128 ◆	RAD IND 01 129 ·	GRAD IND 02 130 x̄	ENTER↑ IND 03 131 ←	STOP IND 04 132 α	RTN IND 05 133 β	BEEP IND 06 134 Γ	CLA IND 07 135 ↓	ASHF IND 08 136 Δ	PSE IND 09 137 σ	CLRG IND 10 138 ◆	AOFF IND 11 139 λ	AON IND 12 140 μ	OFF IND 13 141 ∡	PROMPT IND 14 142 τ	ADV IND 15 143 Φ	8
9	RCL IND 16 144 Θ	STO IND 17 145 Ω	ST+ IND 18 146 δ	ST− IND 19 147 Å	ST* IND 20 148 a	ST/ IND 21 149 Ä	ISG IND 22 150 ä	DSE IND 23 151 Ö	VIEW IND 24 152 ö	ΣREG IND 25 153 Ü	ASTO IND 26 154 ü	ARCL IND 27 155 Æ	FIX IND 28 156 œ	SCI IND 29 157 ≠	ENG IND 30 158 £	TONE IND 31 159 ▒	9
A	XR 0-3 IND 32 160	XR 4-7 IND 33 161 !	XR8-11 IND 34 162 "	X12-15 IND 35 163 #	X16-19 IND 36 164 $	X20-23 IND 37 165 %	X24-27 IND 38 166 &	X28-31 IND 39 167 '	SF IND 40 168 (	CF IND 41 169)	FS?C IND 42 170 *	FC?C IND 43 171 +	FS? IND 44 172 ,	FC? IND 45 173 −	GTO XEQ IND IND 46 174 .	SPARE IND 47 175 /	A
B	SPARE IND 48 176 0	GTO 00 IND 49 177 1	GTO 01 IND 50 178 2	GTO 02 IND 51 179 3	GTO 03 IND 52 180 4	GTO 04 IND 53 181 5	GTO 05 IND 54 182 6	GTO 06 IND 55 183 7	GTO 07 IND 56 184 8	GTO 08 IND 57 185 9	GTO 09 IND 58 186 :	GTO 10 IND 59 187 ;	GTO 11 IND 60 188 <	GTO 12 IND 61 189 =	GTO 13 IND 62 190 >	GTO 14 IND 63 191 ?	B
C	GLOBAL IND 64 192 @	GLOBAL IND 65 193 A	GLOBAL IND 66 194 B	GLOBAL IND 67 195 C	GLOBAL IND 68 196 D	GLOBAL IND 69 197 E	GLOBAL IND 70 198 F	GLOBAL IND 71 199 G	GLOBAL IND 72 200 H	GLOBAL IND 73 201 I	GLOBAL IND 74 202 J	GLOBAL IND 75 203 K	GLOBAL IND 76 204 L	GLOBAL IND 77 205 M	X<>-- IND 78 206 N	LBL -- IND 79 207 O	C
D	GTO -- IND 80 208 P	GTO -- IND 81 209 Q	GTO -- IND 82 210 R	GTO -- IND 83 211 S	GTO -- IND 84 212 T	GTO -- IND 85 213 U	GTO -- IND 86 214 V	GTO -- IND 87 215 W	GTO -- IND 88 216 X	GTO -- IND 89 217 Y	GTO -- IND 90 218 Z	GTO -- IND 91 219 [	GTO -- IND 92 220 \	GTO -- IND 93 221]	GTO -- IND 94 222 ↑	GTO -- IND 95 223 _	D
E	XEQ -- IND 96 224 ˙	XEQ -- IND 97 225 a	XEQ -- IND 98 226 b	XEQ -- IND 99 227 c	XEQ -- IND100 228 d	XEQ -- IND101 229 e	XEQ -- IND102 230 f	XEQ -- IND103 231 g	XEQ -- IND104 232 h	XEQ -- IND105 233 i	XEQ -- IND106 234 j	XEQ -- IND107 235 k	XEQ -- IND108 236 l	XEQ -- IND109 237 m	XEQ -- IND110 238 n	XEQ -- IND111 239 o	E
F	TEXT 0 IND T 240 p	TEXT 1 IND Z 241 q	TEXT 2 IND Y 242 r	TEXT 3 IND X 243 s	TEXT 4 IND L 244 t	TEXT 5 INDM[245 u	TEXT 6 IND N\ 246 v	TEXT 7 INDO] 247 w	TEXT 8 IND P↑ 248 x	TEXT 9 INDQ_ 249 y	TEXT10 IND⊢˙ 250 z	TEXT11 IND a 251 π	TEXT12 IND b 252 |	TEXT13 IND c 253 →	TEXT14 IND d 254 Σ	TEXT15 IND e 255 ⊢	F
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	

For price information and a list of dealers in your area, send a self-addressed stamped envelope to SYNTHETIX 1540 Matthews Ave Manhattan Beach CA 90266 USA

HP-41C Byte Table

REFERENCE: BY KEITH JARETT, SYNTHETIX

AVAILABLE IN CARD FORM FROM SYNTHETIX

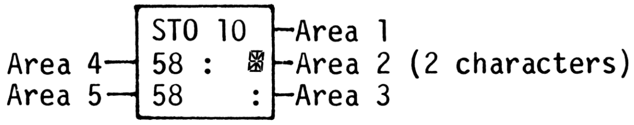
SEE APPENDIX IV.

FIGURE 11

A byte can be represented by two hexadecimal characters. For example, 3A = character 3 and nibble A. The nibble 3 indicates the fourth line (row) from the top of the table, the one beginning by STO ØØ.

The right nibble is the number of the column. It can be read from above or below the table. The character A indicates then the column beginning by LBL Ø9.

At the intersection of row 3 and column A is the diagram of the byte 3A.



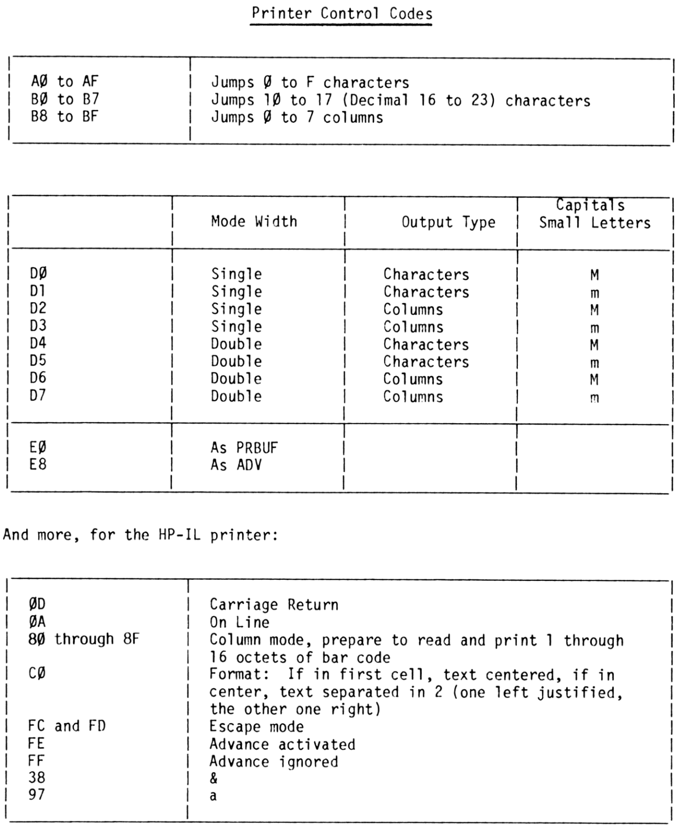
We can distinguish five areas in this diagram which will be analyzed below. Area 5 simply represents the decimal value of the byte considered (hexadecimal 3A = decimal 5B).

c. Area 2 of the diagram is the representation of the code as shown as an alpha character in the display. Never forget that the display is an HP-41 peripheral as well as the card reader and the printer. The HP-41 sends codes, but the display does as it likes. See Appendix X for its exact behavior. The bytes 2C, 2E and 3A (that of this example) have two characters shown in the area 2. The second character is visible only in very unusual situations.

The display shows the starburst ✳ for each character of the second half of the table. This area has therefore been omitted in the second half of the table.

d. Area 3 of the diagram represents the code printed by most of the HP thermal printers. The new Thinkjet printer does not print all of these characters. The HP-IL printer can also, in some cases, react differently (refer to the owner's manual). It is important to note that the printer reacts normally only with characters from the first half of the table.

   The codes of the second half of the table are sometimes ignored, sometimes printed. Certain codes from the second half of the table, when they are transmitted to the printer (for example, as part of a text line in a program listing), can cause the execution of various printer functions. We can provoke a skip of characters or of columns (SKPCHR or SKPCOL), print the printer buffer or get the same effect as the flags 12 and 13 (double width or lower cases, see Figure 12). The HP-IL printer goes even further.



Printer Control Codes

AØ to AF	Jumps Ø to F characters
BØ to B7	Jumps 1Ø to 17 (Decimal 16 to 23) characters
B8 to BF	Jumps Ø to 7 columns

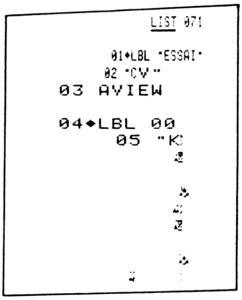
	Mode Width	Output Type	Capitals Small Letters
DØ	Single	Characters	M
D1	Single	Characters	m
D2	Single	Columns	M
D3	Single	Columns	m
D4	Double	Characters	M
D5	Double	Characters	m
D6	Double	Columns	M
D7	Double	Columns	m
EØ	As PRBUF		
E8	As ADV		

And more, for the HP-IL printer:

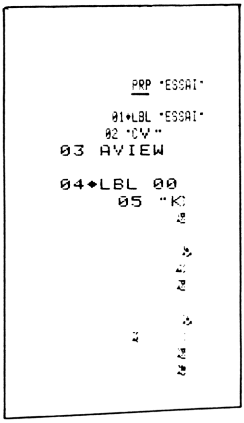
ØD	Carriage Return
ØA	On Line
8Ø through 8F	Column mode, prepare to read and print 1 through 16 octets of bar code
CØ	Format: If in first cell, text centered, if in center, text separated in 2 (one left justified, the other one right)
FC and FD	Escape mode
FE	Advance activated
FF	Advance ignored
38	&
97	a

Printer Control Codes

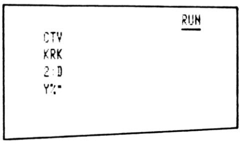
FIGURE 12



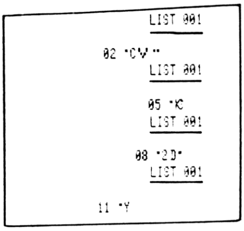
```
               LIST 071

          01♦LBL "ESSAI"
          02 "CV"
03 AVIEW

04♦LBL 00
     05 "K
```



```
               PRP "ESSAI"

          01♦LBL "ESSAI"
          02 "CV"
03 AVIEW

04♦LBL 00
     05 "K
```



```
                    RUN
CTV
KRK
2 D
Y%"
```



```
           LIST 001

     02 "CV"
           LIST 001

           05 "K
           LIST 001

          08 "2D"
           LIST 001

     11 "Y
```

Certain programs are very difficult to list on the printer. For example: in a program you may see in the display these four (4) text lines:

```
02 ᵀ C ▨ V
05 ᵀ K ▨ K
08 ᵀ 2 ▨ D
11 ᵀ Y ▨ ▨
```

Draw your own conclusions.

RUN gives the solution: The control code is given by the second printed character under the condition of looking it up under the second half of the table. After the % any character of the second half of the table, not being a control character, is suitable (here character A2).

Solution

Printer Demonstration Program

FIGURE 13

Notice that certain codes in a program listing give quite curious results. You can see this with the demonstration program, Figure 13.

3.3 INSTRUCTIONS

In hexadecimal code, there are 256 possible combinations, or bytes. This number is not high enough to account for all possible HP-41 instructions, and certain instructions must use several bytes.

3.3.1 One-Byte Functions

One-byte functions appear in area 1 of the table. They use the codes Ø1 to 8F except the codes 1D, 1E and 1F. When the HP-41 encounters these codes in a program, it executes the corresponding function. There is no particular problem to printout, except...

The byte ØØ (null) when found in a program is just ignored and generally will be deleted during the next PACKING. During the modification of a program, an erased (backarrowed) function is replaced by as many bytes as it used. These replacement bytes are ØØ (nulls). An inserted instruction takes the place of the available nulls wherever you want to put it. If there are not enough, the HP-41 liberates a whole register by pushing the following instructions down. This register is filled with nulls.

If two numbers are present in program memory without a separator (press 1 alpha, alpha 2), they will be separated by a null which cannot be removed. Nulls can also exist in multi-byte functions. The byte FØ at the bottom left of the table is also a one-byte instruction.

Bytes 1B and 1C represent EEX and NEG, respectively. EEX is the entry of an exponent and appears under the form of E (1 E2 for example).

NEG is the effect of CHS acting directly on a numerical entry (negation 9 as the - of 1E-2). Curiously, 25 CHS in a program is executed more swiftly than -25 executes.

3.3.2 Two-Byte Functions

The first byte is called the Prefix byte and the second one the Postfix byte. The prefixes are shown in the byte table 9Ø through BF plus CE and CF. Area 1 represents the function Prefix. All the bytes from ØØ through FF can be used as Postfix bytes.

RCL 79 is coded as 9Ø 4F
RCL IND 32 is coded as 9Ø AØ

For the codes ØØ to 63, the value of the Postfix byte is the same as the decimal value of the byte.

It is possible to assign any two-byte functions to any key, but the result is not always useful. This will be described later in Chapter 5.

Area 4 gives the way the postfix byte is shown by the HP-41 display, and where it differs, Area 3 shows the way it is printed by the printer.

M = [	P = ↑
N = \	Q = _
O =]	F = ⊤

Let's notice as well that the postfixes of the second half of the table correspond to the indirect functions. The indirection is indicated by the postfix byte and not by the prefix byte.

Byte AE has a unique double function. It is GTO IND if the postfix comes from the bytes ØØ through 7F, XEQ IND if the postfix comes from the bytes 8Ø through FF.

AE Ø8 is GTO IND Ø8
AE 88 is XEQ IND Ø8

The bytes from AØ through A7 are called XROM. These bytes are prefix bytes, but not exactly with the same meaning as the others. Each function of a peripheral is associated with a two-byte code. XROM numbers appear when the peripheral is not there or if the program, previously recorded on a card or cassette tape, is read into a calculator devoid of these functions. This is the case with such functions as WDTA (write data, card reader data recording), or ACSPEC (accumulate special character of the printer). This also includes the non-programmable functions such as the names of the modules such as CARD RDR IF, -PRINTER 2E, etc. even if the module is plugged into the calculator or included in the calculator such as the time functions of the HP-41CX. We will have to go down to the bit level in order to understand the structure of these XROM instructions. Two bytes are equivalent to 16 bits. The first five bits (on the left) are used as the prefix (1Ø1ØØ) leaving 11 bits for the postfix. These 11 bits are divided into five bits for the peripheral number and six bits for the function number within the peripheral.

With six bits we can count in decimal up to 64 (in fact, from Ø to 63), with five bits, we can count up to 32. There are thus 32 possible numbers for the peripherals and 64 possible functions in each peripheral.

Let's look at an example:

WDTA has the number XROM 3Ø, Ø7. That is to say in binary:

```
XROM = 1Ø1ØØ (5 bits = prefix)
3Ø = 1111Ø (peripheral number 3Ø)
Ø7 = ØØØ111 (function number Ø7)
```

Putting all 16 bits together in groups of 4 bits each gives us:

```
1Ø1Ø Ø111 1ØØØ Ø111
1Ø1Ø = A
Ø111 = 7
1ØØØ = 8
Ø111 = 7
```

Therefore, XROM 3Ø, Ø7 consists of codes A7 87.

Notice also that the prefix byte contains all but the two rightmost bits of the peripheral number. This explains why each entry of the table corresponds to four peripherals (two bits = four possibilities). For additional information on XROMS, see Figure 14.

Consider boxes 9Ø and 91 of the byte table. In order to save room in program memory, Hewlett-Packard allowed for STO and RCL to be one-byte functions for the most used registers (rows 2 and 3 of the table). It is possible, but not profitable to build RCL and STO two-byte functions with prefix 9Ø and 91 and postfixes from ØØ to ØF. In fact, before executing these in microcode, the HP transforms them from two-byte functions to one-byte functions.

Supplementary Information about XROM Numbers:

People wonder why the XROM number prefix consists of five bits. This is due to the fact that the XROM uses only half of line A, the fifth bit is always null (Ø). However, we do have to underline here a particular phenomenon which is that of synthetic XROM displays.

Later you will see a program (KA) which allows you to assign to a key any pair of codes. The only functions which can legally hold two bytes of the assignment registers are the XROM functions. When the HP-41 reads a function assigned to a key, the program which displays the name of the function only checks to see that the first nibble of the function is not Ø and then displays an XROM number calculated from the three following nibbles. The HP-41 assumes that if the first nibble is not zero, it must be A. The message XROM __, __ corresponds to 15 functions (first nibble between 1 and F). This kind of XROM may appear up to XROM 63, 63.

If x is the decimal value of the first byte of the function and y the value of the second byte, XROM i, j is calculated by:

i = 4 (x MOD 16) + INT (y/64)
j = y MOD 64

We can deduce from this formula a method to find without calculators the XROM number from the byte table (a method described by Keith Jarett).

To find the XROM number of an instruction (ST + IND M for example), notice that ST + (92 - the first byte) is in the same column as XR 8-11 (A-2). The column number of the first byte x is, in fact, x mod 16. This pins down i to four possible values, which are shown in row A of the byte table, at least for columns Ø through 7. For example, ST+ is in column 2. Checking column 2 of row A we see the notation XR 8-11, indicating that the first of the two XROM numbers displayed will be 8, 9, 1Ø or 11.

The exact value of i is determined by which block of four rows the second byte y is in. The heavier horizontal lines on the byte table help you to visualize the block boundaries. Rows Ø to 3 correspond to the first value of i, rows 4 through 7 to the second, rows 8 through B to the third, and rows C through F to the fourth. If you then visually move the second byte up to a corresponding box in rows Ø to 3 (this is equivalent to taking y mod 64), you can read off the value of j from the Area 4 or 5 of the box.

In the same way, if you assign ADV IND e (decimal 143, 255), using the KA program, you will have XROM 63, 63 and in program mode will display simply ADV. Of course, ADV IND e isn't very useful, and using the byte table isn't very convenient for any XROM above 31.

FIGURE 14

A similar circumstance arises with the numerical tables. A supplementary remark is essential here when you assign to a label (prefix CF) a postfix byte 66 through 6F. The table shows that you obtain LBL A to LBL J which are the "local" labels A-J. We can see that the local labels are actually numerical labels (1Ø2 through 111).

In Row B, except for the case of BØ which isn't used as a prefix, we deal with a particular case of two-byte functions. The prefix is by itself the function GTO Ø1 or GTO 13. The second byte is null when keying the program. At the first execution of GTO __ (Ø1 through 13), the HP-41 calculates the absolute jump distance to the LBL and stores it in this free byte. This operation is called compilation.

If the second byte is null, it indicates that a compilation hasn't been performed. After compiling, this second byte is organized such that if you number the bits 7 6 5 4 3 2 1 Ø you have:

> Bit 7 indicates the direction of the jump. 1 if the jump is forward to a program line of a greater number, and Ø if the jump is backwards to a program line of a lesser number.

Bits 6, 5 and 4 indicate the number of remaining bytes.

Bits 3, 2, 1 and Ø indicate the number of registers.

The distance is counted such that the number of bytes located between the end of the GTO and the beginning of the LBL in a forward jump are the only bytes counted, whereas in a backward jump the two bytes of GTO and the byte of the LBL are counted. The maximum distance of the label is F registers and F bytes or 112 bytes. The minimum distance is zero. If the label is too far, the second byte of the GTO is left at Ø and the HP-41 performs all this work each time it is encountered in a program.

3.3.3 Three-Byte Functions

These are rows D (GTO __) and E (XEQ __). These are similar to the two-byte GTOs but with a greater jumping distance which justifies the extra byte required. Note that this is the only possible form for XEQ:

3 bytes = 24 bits, therefore:

2 bits are used as the prefix: 11
2 bits give the type GTO=Ø1, XEQ=1Ø
3 bits give the number of remaining bytes (from Ø to 7).

9 bits give the number of registers from Ø to 511
1 bit gives the direction.
1 = Forward, Ø = Backwards.
7 bits are reserved for the description of the label

For example:

```
11Ø1:  1ØØ      Ø:ØØØØ:ØØ1Ø: 1 ØØØ:Ø1ØØ
GTO   4 bytes   2 registers  + LBL 04
```

This example has been selected to show that it is quite possible to use a three-byte GTO to seek a one-byte label. Here the codes are D8 Ø2 84.

The byte count is done between the end of the first byte of GTO or XEQ and the beginning of the LBL. Therefore, in a forward jump the two compilation bytes of the GTO or XEQ are counted, but not the LBL. In a backward jump, the first byte of the GTO or XEQ is counted as well as the 1 or 2 bytes of the LBL.

3.3.4 Variable Byte Instructions

Character Strings

The bytes of row F are prefix bytes announcing strings of characters in a program. We have seen how to identify the characters in a register. In a program, this identification is done by a byte Fn where n is the number of bytes to be read after the prefix. These are considered as alpha characters and placed in the alpha register.

The maximum is therefore FF, or 15 characters, and the minimum is FØ or Ø characters. When FØ is used, no characters are placed in alpha and effectively alpha is not disturbed.

In addition, this code (FØ) is used to identify the assignment registers.

When incorporated into a program by means of synthetic programming, the FØ byte turns out to be a non-operating instruction (NOP) and hence useful as a filler in some cases, after ISG or DSE for example.

Multi-byte example:

^{T}DODIN = F5 44 4F 44 49 4E

The character ⊦ (7F) placed just after the prefix warns the HP-41 that the alpha register mustn't be deleted before adding the new string. This is the APPEND function.

3.3.5 Global labels - END

The bytes CØ through CD "Global," perform a double role: they identify both the END and alphanumeric labels.

The END

If the third byte of a line starting with the byte C2 (with "a" being between Ø and D) is a Fn text byte, the line is a label. Otherwise, it is a three-byte end.

In both cases, the second, third and fourth nibble (from the left) give the distance to the preceding END or LBL in Catalog 1. This distance is coded just as for a three-byte GTO. It is counted from the label beginning to END beginning. A label (or END) begins by Cx in binary:

    11ØØ abc d

11ØØ = C
abc = remaining bytes counting the distance
d = added to the following byte = number of registers

WARNING: CE = X<>__ and CF = numerical LBL, therefore, it is impossible to have abc = 111. If so, we would have CE = CF. This isn't too serious, except that when the

the number of registers is maximum, we can add only six bytes. With the three-byte GTO and XEQ, it is possible to add seven bytes. This capability is not needed in normal programming.

This storage of the distance between elements of Catalog 1 provides for the "chaining" of labels in memory. A GTO or XEQ alpha begins to seek a label from the end of memory (the permanent .END.) and proceeds backwards through Catalog 1 up to the first label which is then identified by the two first bytes, CØ ØØ.

The fifth and sixth nibbles of the end (from the left) are used to provide status information. The fifth nibble is a 2 (ØØ1Ø) for a permanent .END., 4 (Ø1ØØ) for a private END, and 6 (Ø11Ø) if it's the .END. and it's private.

The sixth nibble is Ø (ØØØØ) for the .END. after a Memory Lost, 9 (1ØØ1) if the program isn't packed, and D (11Ø1) if the program is packed.

The Labels

Let's talk about alphabetical labels. The first two bytes are then composed of C followed by 3 distance nibbles. The third byte is a text prefix, the fourth byte is a null character. Invisible to the user, the null character is used to record a possible eventual key assignment (the key code). The following bytes spell the name. Because of the null, the n of Fn is 1 greater than the number of letters of the name.

A LBL "DE" alone in memory assigned to the key (A) will be coded as CØ ØØ F3 Ø1 44 45.

The codes 1D and 1E are codes of GTO and XEQ alpha, they are followed by an ordinary string of characters. For example:

GTO "DODIN" = 1D F5 44 4F 44 49 4E

The byte 1F displays W^T. This byte is inactive as a prefix. At the end of 1983, different uses of this code were discussed by the PPC-T club without any convenient applications being found.

The XEQ "ALPHA" are substituted by XROM "ALPHA" if the so-called program "ALPHA" is a program in user language and not in microcode located within a ROM. This is the case for the math module and its programs.

3.4 ORGANIZATION OF PROGRAMS IN ROM

This organization is mostly required due to the possibility of using the COPY function which allows one to copy a program from ROM into RAM.

3.4.1 The Control Words

A program in ROM is preceded by a code word indicating the number of registers required to copy the program into RAM.

The second word determines (Code 244) if the program is private. The middle nibble gives the number of bytes used in an incomplete register.

The following words are the program bytes. The two extra bits (since the words in ROM are 1Ø bits versus the 8 in RAM) are on the left (244 Code). They are equal to Ø1 if the byte is the first byte of an instruction and ØØ otherwise.

At the end of a program, the last byte of END (and therefore of the program) is 22F (244 code).

3.4.2 The Links

Once the corresponding GTO's and XEQs are compiled, labels are no longer useful (except for GTO IND, XEQ IND, and copying the program from ROM into RAM). The distance is calculated with the address of the last included byte, less the address of the first byte included plus 1.

The linking of alphabetical labels is done the same as in RAM. For the first label, a remark is required.

During a copy into RAM, the HP-41 frees the required number of registers below the .END. . This .END. is then transformed into a normal END. The block of registers so liberated is at the bottom of program memory. The .END. is therefore the new END and must be located on the right of its register. The whole program copied is therefore compressed down and the first register is incomplete. We have:

```
x x x x E N D    program residing
x x x L A B E    in RAM
L x x x x x x
```

The number of bytes of this first register is the number shown in the second header word. We must take this into account to calculate the distance between the first label of the program and the following END. The empty bytes of the first register must be counted.

CHAPTER 4

A SPECIAL AREA

A SPECIAL AREA

The Status Registers

The status registers are the 16 registers whose address is included between absolute addresses ØØØ and ØØF.

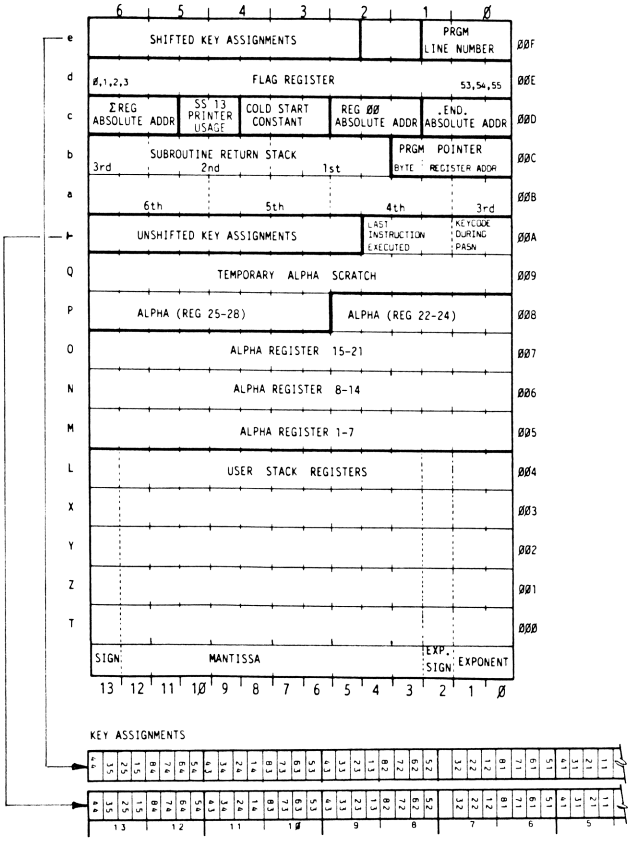
A detailed map of these registers is shown in Figure 15.

4.1 The Stack

The famous automatic stack of the HP-41 is a set of five registers: X, Y, Z, T, and L (LASTX). These registers are are available to any user through normal means. They are located at the bottom of RAM memory. T is the register R(ØØØ), Z is R (ØØ1), Y is R(ØØ2) and X is R(ØØ3). Register L (LASTX) is R(ØØ4).

These registers don't have any other special feature. Their manipulation by the microcode functions of the HP-41 is what makes them special.

Yet, there is another detail. We have just seen that the stack is just another address in memory but at a constant location (ØØØ-ØØ4). On the other hand, the user registers (RØØ..R99..R319) are at variable locations and at times do not even exist. The HP-41 must check the existence of these registers, then calculate their position each time it uses them.



The Status Register Map

FIGURE 15

In contrast, the stack registers are always there. That is why an ISG or a DSE statement is far faster in the stack than in a user register.

Well, have you noticed that the statements STO L (STO.L) or RCL L (RCL.L) exist initially? Although the first one is sometimes useful, the second does the same thing as LASTx but uses one more byte.

4.2 The Alpha Register

Here it becomes more interesting. The "alpha" register is known to hold 24 characters, right? Well, in fact, this register is not one register but, in fact, four normal 7 byte registers called M, N, O and P which are the registers R(ØØ5), R(ØØ6), R(ØØ7) and R(ØØ8), respectively.

Still more interesting is that these registers can be used as any other ordinary registers. Remember that we have said that any given prefix can be used with any given postfix. Well, from Ø to 99 (decimal) we have the normal registers, from 1ØØ to 111 we have the same normal registers, and from 112 to 127 we obtain status registers! The code 9Ø 75 or RCL 117 displays in fact RCL M. This is the status register M, the rightmost register of Alpha.

When you enter a character into the alpha register, it is always right justified in the M register. That is, the rightmost byte always contains the rightmost character, and the next byte contains the second to the last character, and so on. If the alpha register contains 7 or fewer characters, only the M register is used.

As you enter characters into alpha, the first character is pushed toward the left in N, into O, then into P. An even more extraordinary thing is that the character is pushed up to the far left side of P before completely disappearing. We then have 28 characters in alpha. But this won't last very long, for the four bytes on the left of P are often used as scratch by the HP-41. Moreover, the characters which are located there are not visible on the display. Although not readily apparent, this property is sometimes useful.

Registers M, N, O, and P can therefore be used:

-- normally with the alpha functions
-- Directly with STO, RCL, VIEW, ISG...and so on, as storing and counting registers, with the same advantages of speed as the stack.

The mixture of these two ways of working allows increased control of the content of these registers.

The other status registers which are normally invisible to the user but not for us are Q, ⊢, a, b, c, d, and e. They aren't generally used completely by the HP-41, but several areas of certain registers have a specific use.

4.3 Register P (Address ØØ8)

As mentioned previously, the four leftmost bytes of the P register are used as scratch by the HP-41. For example, during a "catalog" the nibble located at the far left (MS) of P contains the catalog number (1, 2, or 3), and the other nibbles contain the number of functions which have already been displayed during the execution of the catalog.

It would be tedious and unnecessary to list all the different cases of using the P register by the HP-41. A test is better if you intend to use it. However, be careful when using peripherals, for they also sometimes use the four leftmost bytes of the P register.

4.4 Register Q (Address ØØ9)

This register is also a scratch register, but it is a very important one, since the HP-41 uses this register for the name of the functions that you spell after an XEQ. This name is loaded into Q, written from right to left. The non-used bytes are null (ØØ). This register is also used in many other cases, particularly by the printer. In fact, it is used so often that it is near impossible to use it for storage with any sense of security.

4.5 Register ⊦ (Address ØØA)

The append register is named for its display by the HP-41. The five right nibbles are used as scratch, but the others are far more interesting. They represent the key map of assigned unshifted keys.

You know that each key of the HP-41 can be assigned. It would take far too much time for the HP-41 to look for an occasional assignment in the entire memory when you press each key. Consequently, the calculator keeps an up-to-date index of the keys which have been assigned and starts the research only if this index points out the existence of an assignment.

There are 35 keys on the keyboard. Therefore, it is necessary to have 35 index positions, therefore, 35 bits. Four bytes yield 4 x 8 = 32 bits, so we must have one more nibble. That is to say 36 bits for the key map. There is one unused bit, but that is not a problem. The correspondence between bits and keys is shown in Figure 15.

4.6 Registers a and b (Addresses ØØB and ØØC)

These registers are generally pointer registers.The main pointer is the program pointer, the one which tells us which program byte is performing. This pointer is composed of the two rightmost bytes of register b.

a. ROM:
When the pointer is in a ROM module, for example in the PPC ROM, the program pointer represents the address of the

byte expressed in four nibbles from ØØØØ to FFFF, which allows 65,536 bytes. These bytes have a number increasing in the same sense as the program lines.

b. RAM:
When the pointer is in RAM, for example running your favorite program, it has a different form. The three rightmost nibbles of register b give the address of the register in which the byte is located and nibble number 3 gives the number of the byte in the register. The register and byte numbers decrease as the line numbers increase.

c. The Subroutine Stack:
When you use the XEQ statement in a program, the HP-41 must note the position of the XEQ to be able to come back after executing the sub-routine. Where is this address stored?

It is next to the program pointer in bytes number 2 and number 3 of register b. Register b contains the first and second return pointers, and half of the third return pointer. Register a contains the other half of the third return pointer and the fourth, fifth and sixth return pointers. Each pointer consists of two bytes. The pointers are pushed to the left upon each return from an XEQ (subroutine call). In fact, the entire register b is pushed to the left, and that which goes out on the left of b goes in on the right of a. Bytes pushed to the left of a are lost. This is why, if a seventh call occurs without a RTN, the first call is lost.

Bytes 1 and Ø of register b contain the current program pointer. When an XEQ instruction is encountered, this pointer is pushed onto the return stack; that is, into bytes 3 and 2 of register b. If another XEQ is encountered before the RTN from the first one, the program pointer and the first return are pushed leftward two more bytes. In this way, the return stack in registers a and b can accommodate up to six pending return addresses.

When a RTN instruction is encountered, the first return address in bytes 3 and 2 of register b is checked. If its value is zero, the current program pointer is retained and control returns to the keyboard. Otherwise, the return stack is shifted right two bytes, with the former first return address being moved into the program pointer slot. Execution continues from that location in program memory, one step past the XEQ instruction that caused the return address to be pushed onto the return stack.

Now for a little technical detail on program pointers. The four hexadecimal digits of the program pointer are interpreted one way for RAM and another way for ROM pointers (those from a plug-in Read Only Memory). For RAM, the first four bits denote the byte number within the register, while the other 12 bits denote the register's absolute address from the bottom of memory. The format is:

Øbbb,ØØØr,rrrr,rrrr

where bbb denotes the byte number (expressible in three bits since the maximum value is 6 = Ø11Ø base 2) and where r,rrrr,rrrr denotes the register number (expressible in 9 bits since the maximum value is 511 = ØØØ1,1111,1111 base 2). For example, Ø1Ø1,ØØØ1,1Ø1Ø,111Ø = hex 51AE denotes byte 5 of register 1AE (= 43Ø decimal). Byte numbers range from 6 to Ø as the program pointer moves downward through one register of a program. Thus, 61AE is above 41AE in a program, and 41AE is above 61AD.

RAM return address pointers are the same as ordinary RAM pointers, except that the three bits that designate the byte number within the register are shifted to the right. These bits, normally the second, third and fourth from the left of the 16-bit pointer, are shifted three positions over, to the fifth, sixth and seventh bit positions. The RAM return pointer format is:

ØØØØ,bbbr,rrrr,rrrr

ROM pointers consist of a port address in the first four bits plus a 12-bit byte number within that port:

```
   pppp,bbbb,bbbb,bbbb
```

The port address part of a ROM pointer is not the same as the physical port number. The correspondence is:

Port Address	Physical Port or Device
Ø	internal ROM Ø
1	internal ROM 1
2	internal ROM 2
3	X Functions (CX only)
4	Service module
5	Time module
6	Printer
7	Tape Drive (IL monitor)
8	Port 1, Lower 4K
9	Port 1, Upper 4K
A	Port 2, Lower 4K
B	Port 2, Upper 4K
C	Port 3, Lower 4K
D	Port 3, Upper 4K
E	Port 4, Lower 4K
F	Port 4, Upper 4K

Each port address can accommodate a 4 kilobyte ROM (4Ø96 = hex FFF + 1 bytes). The 12-bit byte number starts at zero and increases toward FFF as sequential ROM program instructions are executed.

Another important detail: When you RCL b in RUN mode at a specific line of program memory, the pointer value is usually one byte above the location where the instruction resides. Thus, if a RCL M instruction is located in bytes 6 and 5 of register 1AE, and you RCL b at this line of program memory, the resulting pointer value will be Ø1AF hex, one byte above the actual location of the RCL M instruction. Where nulls are present, the pointer will be farther above the instruction. In fact, it will be one byte above the group of nulls preceding the instruction.

4.7 Register c (Address ØØD)

This is one of the more dangerous and interesting registers among the status registers. This register holds, from right to left:

The right 3 nibbles give the address of the register in which the permanent .END. is located. As the .END. is always located in the 3 rightmost bytes of its register, 3 nibbles are enough to define this address.

Next is the address of the first data register, the one which we call RØØ. This also consists of 3 nibbles. This is the lower limit of the data registers and the upper limit of the program storage area. We call this the "curtain" between programs and data registers.

Next to this is the "Cold Start Constant" which is the number 169. It is a value which must not be disturbed unless caution is used. The HP-41 checks it whenever a program is not running, and even sometimes in a running program. If it finds 169 in this area everything goes right. Otherwise, it runs a "cold start", and you again are unlucky enough to see the foreboding "Memory Lost."

However, while a program is running, this checking is infrequent and you can play a bit, but don't forget to replace the 169 prior to stopping the program.

The 3 leftmost nibbles point to the absolute address of the first statistic register (ΣREG).

The two remaining nibbles (9 and 1Ø) are used by the printer.

4.8 Register d (Address ØØE)

In this register are the 56 flags of the HP-41 (see Figure 16). Each flag is represented by one bit. The capability of accessing this register allows you total control of all flags. In fact, all 56 flags can be manipulated by one single statement. As the flags are tested by the HP-41 only from time to time, we can, most of the time, manipulate them without any problem. When manipulated within a program, it is good practice to restore the original value of the register before stopping the program. Beware of specialized flags like flag 45 (data input) or flag 21. This register is one of the most used, so much so that Hewlett-Packard finally recognized it by including functions in the X-function module for working on this register.

4.9 Register e (Address ØØF)

The three rightmost nibbles hold the number of the program line in the case of a running program. The two following nibbles are used as scratch on several occasions by the HP-41. The other nibbles, from 8 through 13, are the key map of the shifted key assignments.

6	13	00	
		01	
		02	
		03	
	12	04	
		05	
		06	
		07	
5	11	08	
		09	
		10	
		11	AUTO EXECUTE
	10	12	DOUBLEWIDE
		13	LOWERCASE
		14	OVERWRITE
		15	IL
4	9	16	PRINTER
		17	INCOMPLETE RECORD
		18	GENERAL USE
		19	- -
	8	20	
		21	PRINTER ENABLE
		22	NUMBER ENTRY
		23	ALPHA ENTRY
3	7	24	RANGE IGNORE
		25	ERROR IGNORE
		26	AUDIO ENABLE
		27	USER MODE
	6	28	DECIMAL / COMMA
		29	DIGIT GROUPING
		30	CATALOG
		31	TIME DMY/MDY
2	5	32	IL MANUAL I/O
		33	IL ABSOLUTE MANUAL
		34	IL ADDRESS ON/OFF
		35	AUTOSTART ROM
	4	36	NUMBER OF DIGITS
		37	
		38	
		39	
1	3	40	DISPLAY MODE
		41	
		42	TRIG MODE
		43	
	2	44	CONTINUOUS ON
		45	DATA ENTRY
		46	PARTIAL KEY
		47	SHIFT
0	1	48	ALPHA
		49	LOW BATTERY
		50	MESSAGE
		51	SST
	0	52	PGRM
		53	I/O
		54	PSE
		55	PRINTER EXISTENCE

00	MANUAL
01	NORMAL
10	TRACE
11	STK TRACE

SCI	00	00	DEG
ENG	01	01	RAD
FIX	10	10	GRAD
FIX/ENG	11	11	RAD

The HP-41C Flag Map

FIGURE 16

CHAPTER 5

THIEF!

THIEF!

Since the discovery of the status registers and the commands with abnormal postfixes, many applications have been created. The best application is the module created by the PPC club and named the PPC ROM. This module contains, amongst many programs of general interest, a certain number of small programs utilizing a lot of synthetic programming. These programs have minor flaws but bring to the programmer an evident improvement. Believe me, the advertising that I am doing for PPC ROM is free! It is available to all by mail in the United States, even to non-members of the club, at a price of about $100.

But the PPC ROM is not a requirement, luckily! A very simple method has been found to create a byte grabber.

5.1 The Byte Grabber

Follow to the letter the instructions below (one bare HP-41 is the only requirement; however, a card reader is a welcome addition).

This procedure works with all machines. Certain accessories might sometimes disturb this operation. Therefore, use a bare (no module) machine (HP-41C, CV or CX). Be patient on all steps!

DO	SEE (In the Display)
1. Master Clear	MEMORY LOST
2. Assign + to the LN Key	ASN +15
3. Assign DEL to the LOG Key	ASN DEL 14
4. Put the calculator in User mode	Ø.ØØØØ
5. Put the calculator in PRGM mode	ØØ REG 45
a. Shift LBL Alpha T Alpha	Ø1 LBL ^{T}T
b. Shift CAT 1 and immediately R/S	LBL^{T}T
c. LOG Σ+	DEL ØØ1 4Ø94 .END. Reg 44
d. BST	4Ø93 4Ø93 DEC
e. Shift GTO. LN	GTO.ØØ5 Ø5 LBL Ø3
f. LOG Σ+	DEL ØØ3 Ø4 STO Ø1
g. Alpha ?AAAAAA Alpha (on the CX or if you have X Functions you will see ^{T}?AAAAAA).	^{T}?A-----

Get out of Program Mode, do GTO.. and it's finished.

Press <u>and hold</u> LN; see XROM 28,63. If you don't see this, start over from step 1.

Explanation

Before going further, if you have a card reader, record a status card (WSTS) on both track 1 and 2.

Steps a, b, and c utilize a peculiarity of the machine called "BUG 9" discovered by the PPC Club.

The line number 4Ø94 appearing on the display then the number 4Ø93 preceding DEC don't have any meaning, except to show you that you are now below the .END., in fact in the (Ø1Ø) register, the first register of the shifted key assignment registers (Status Register e).

Do again the operations from 1 to 5d, then, instead of doing a GTO.005, do many BST (be patient because each BST is slow). You will see:

```
                     Code

4Ø86 T               FØ
4Ø87 LBL Ø3          Ø4
4Ø88 LBL Ø1          Ø2 Equivalent of DEL
4Ø89 STO Ø1          31 Key Code for LOG
4Ø9Ø LBL Ø3          Ø4
4Ø91 +               4Ø Equivalent of +
4Ø92 -               41 Key Code for LN

Here 176 empty registers ---------------

4Ø93 DEC             Now you are in the
                     status registers
```

When you do GTO.ØØ5, the HP-41 shakes a little and retakes a "normal" count of the lines starting from FØ, the Ø5 LBL Ø3 appearing is thus the one preceding the +.

DEL.ØØ3 erases the LBL Ø3, + and -, leaving three nulls.

When you introduce the chain of characters (?AAAAAA), the HP-41 displays nulls: the first byte is the Fn byte where n will vary.

The n increases from 1 to 7 with the gradual introduction of letters, first ?, byte 3F, then A, byte 41 (look here, the same code as -!), and the three nulls opened by DELØØ3 are used. But now you are to the right of R(ØCØ) and there is nothing below, so the following letters fall into emptiness and cannot be recorded. However, the n of Fn increases each time by one unit and the display shows as nulls (----) the part of "emptiness" included in the chain.

NOTE: If X Functions are present, you will see A's instead of nulls because R(ØBF) is present.

You end up with a chain of seven characters (?, A and 5 nulls), the first character being ? and the second A. This A is taken by the machine as key code. Try another key: TAN has for assignment code 42, corresponding to the letter B. Try ? and six times B.

Well, it doesn't work? Did you think it would? Yes, it is now the TAN key that has to be assigned in Step 2.

Let's go back to the first example (for instance by reading the card just recorded). Pressing and holding the LN key in User mode:

XROM 28,63

This is one of the many possible byte grabbers.

5.2 What is the Byte Grabber (BG)?

First, it is principally useful in program mode and must never be used before an .END. or in empty program memory (no kidding, this is really important).

The BG creates in program memory a text line of seven characters. In order to put it in place, the HP-41 frees a register of seven bytes. But before the chain there is a F7 byte and thus this chain of seven bytes requires eight bytes of memory. We have thus created a chain:

F7 ØØ 3F ØØ ØØ ØØ ØØ = 7 bytes, but the HP-41 will by any means find the seventh character that is missing. The HP-41 purely and simply appends the first byte, null or not null that will come forward (for details see Figure 17).

Example:

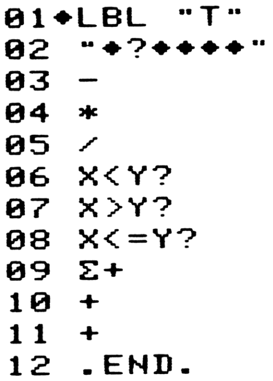
```
    Ø1 LBL^TT
    Ø2 +
    Ø3 +
    Ø4 +
```

Then do GTO.ØØ1 and BG (press LN in user program mode) you will see:

02^{T}-?----@. You see the character with code 4Ø, the same code as +, stolen and introduced into the chain.

Backarrow this line and type:

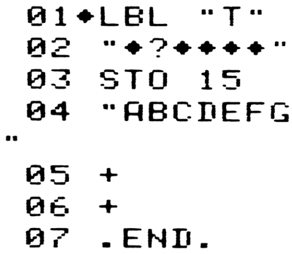
Ø2^{T}ABCDEFG
BST and BG



```
01◆LBL "T"
02 "◆?◆◆◆◆"
03 -
04 *
05 /
06 X<Y?
07 X>Y?
08 X<=Y?
09 Σ+
10 +
11 +
12 .END.
```

Where are these program lines coming from? As you can see from the code tables, these are simply the instructions having the same code as the letters previously entered. It is the masking of the F7 byte at the beginning of the "ABCEDFG" text line by the BG that keeps the HP-41 from knowing they are letters. You can SST or BST in this "program" without worries.

GTO.ØØ1, BG and you will see



```
 01◆LBL "T"
 02 "◆?◆◆◆◆"
 03 STO 15
 04 "ABCDEFG
"
 05 +
 06 +
 07 .END.
```

What has happened? The new BG has stolen the F7 byte from the first BG. The thief has been robbed! The nulls are not visible, STO 15 has the same code as ?, the F7 of the chain is freed and the "ABCDEFG" text line takes back its normal existence. You can backarrow lines 2 and 3, pack, and you are back at the start. We will call this operation "to suppress the BG." Do it. And let's see the use of this new function.

Again BG at line Ø1, you will be back to the configuration shown in the first figure. Backarrow line Ø6 X<>Y? and replace it by LBL ØØ then GTO.ØØ1 and suppress the BG. You will see:

ᵀABC웃EFG

You have replaced the D, in the text line by a symbol called "the complete man." If you make a mistake during the operation, simply XEQ PACK. Enjoy yourself by trying other characters.

Introduce a null: the nulls necessarily must be introduced last, because one can no longer XEQ PACK (PACK removes all nulls). Thus, you cannot make an error. By starting from the situation above, let's do GTO.ØØ1, BG.

NOTE: If, after a BG, there are no characters at the right of the BG text, it is because there were some null bytes lagging behind. Don't worry; simply do BG many times and leave the cleaning for later. In this case, it can happen that the BG steals more than one byte. If this bothers you, suppress the BG by the usual way and XEQ PACK before restarting. Otherwise, continue as if nothing had happened.

Backarrow LBL ØØ and, without doing PACK, BST up to the line containing the beginning of your character chain, suppress the BG, and you will see:

ᵀABC‾EFG

A little bit of housekeeping and a PACK wouldn't do any harm, would it?

NOTE: Before the execution of BG, you must see on the display the line preceding the command that you want to steal.

These explanations may seem complex, but with practice they become so practical that I always have a BG assigned to a key of my HP-41.

You probably asked yourself why one can see a line 4Ø94 appear during the creation of the BG. In fact, during the program, the HP-41 does not calculate the program line numbers, it does not need to. But if one stops the program to look at it, it has to recalculate everything. So that it would be aware of the necessity of this calculation, the HP-41 places in register e the line number FFF, which is clearly not a valid line number.

Now FFF equals 4095 in decimal. In the procedure of creating the BG, at the moment we erase the LBL "T", the HP-41 goes back one line from the FFF without noticing that this number is invalid, from where 4Ø95 -1 = 4Ø94.

The Operation of the Byte Grabber Unveiled

After assigning the byte grabber (XROM 28,63), GTO.. enter the following program:

```
    Ø1        ENTER↑
    Ø2        ENTER↑
    Ø3        ENTER↑
    Ø4        ENTER↑
    Ø5        ENTER↑
    Ø6        ENTER↑
    Ø7        ENTER↑
    Ø8        "ABCDEFGHIJ"  the FA byte followed by ten characters = 11 bytes
```

Accounting for the three bytes of the .END., you have thus filled 7 + 11 + 3 = 21 bytes or 3 registers.

An XEQ "Pack" will confirm it for you. Place yourself at Line Ø7 ENTER and backarrow 3 lines you see:

```
    Ø4 ENTER↑
    BG (press and release the LN key in user mode) you will see:
    Ø5 ⁻⁻▩▩ABCD
```

What Happens? When you want to introduce a function in memory, the HP-41 knows (thinks it knows!) that this function occupies at the most three bytes. (There is a function of three bytes that can be assigned, it's the function END.) In fact only the byte CØ figures in the assignment register, and the following bytes are built at the moment of the introduction of the function in the memory. If there is not enough room available to introduce the desired function, the HP-41 frees an entire register, i.e. 7 bytes. That appears more than sufficient to the calculator to hold a command that can use only three bytes.

But the byte grabber has two characteristics: it is a three byte function which puts into memory one F7 byte. The following chain of characters thus contains seven characters. Let's investigate the different cases:

1. No null bytes are present. The HP-41 frees one register and places the BG to the left of this register. The chain so created includes the F7 byte, the six following bytes from the register and one byte from the following register, the stolen byte. This is the usual case.

2. A single byte is free. The F7 puts itself in place of the free byte, and the chain occupies the following register, freed for that purpose, nothing is stolen, but all the null bytes have been covered, and we are back to the first case.

3. Two bytes are free. The F7 puts itself in the free byte, and the chain occupies the following register, but one null byte is left. This takes us back to the second case.

4. Three bytes are free. That's enough for the byte grabber (don't forget that the byte grabber is a function that copies again in memory the three bytes contained in the corresponding assignment register). With that move no register is free and the BG steals five characters. This is the case experienced above.

If one increases the number of nulls, one can choose between Ø and 5 the number of stolen characters.

FIGURE 17

5.3 The Synthetic Postfixes

The byte grabber can be used to fabricate in program memory all the synthetic functions that you want. Consider the two byte functions, for instance RCL, ISG, VIEW, FIX or TONE. The prefix codes of these functions are from Row 9. We can build in memory a byte chain that will put this prefix and one byte that we will transform into a postfix. We will try to use the byte 75 as a postfix.

Using the following bytes:

```
90 98 75
```

90 = RCL, after RCL, the HP-41 seeks a postfix and takes the 98 as the postfix. 98 is IND 24, there thus remains one 75 byte all alone, which is RDN and the HP writes:

```
RCL IND 24
RDN
```

Are you capable of creating these two program lines? Yes? Then you can do everything. Did you understand? We will steal (BG) the 90 and then simply erase the chain of characters given by the BG. Remaining will be the two bytes 98 and 75.

DO

```
Ø1 LBL Ø1
Ø2 RCL IND 24
Ø3 RDN
BST
BST
BG
Backarrow
SST
```

SEE

```
Ø2 VIEW M
```

Try this also with:

```
RCL IND 16 & RDN (RCL M)
RCL IND 22 & RDN (ISG M)
RCL IND 28 & RDN (FIX M)
RCL IND 31 & RDN (TONE M)
```

Enjoy yourself...there are still many things to explore...

5.4 Key Assignments

The following program (Figure 18) is an alternate of a very common program from the PPC Club traditionally called KA (Key Assignment), modified to be absolutely without risk of memory lost. Watch out, however, you can stop the program, and SST, but it is imperative that you complete the two assignments if you want to find your machine back in its starting position.

Before using "KA", each time you must manually assign (using ASN) any function to two keys. When you execute "KA", these dummy assignments will be replaced by the synthetic functions that you specify.

Program Description: This program fabricates in Alpha the contents of a synthetic assignment register and places this contents in R(ØCØ) that was previously occupied when we assigned the two keys. This assignment serves also to update the index in the registers ⊦ or e.

The building of this dummy assignment register is started on line Ø5 where the character FØ is placed in alpha.

When ??? appears in the display, enter in decimal the prefix of the function to be assigned, ENTER↑, next the postfix; ENTER↑, then the row/column key code, then R/S.

The subprogram Ø2 converts the prefix and then the postfix to hexadecimal and places these in alpha, the key code is then put in the correct form and placed in alpha. Then a second ??? asks for the second key assignment. You absolutely must answer this demand, if needed simply enter zeros or repeat the first assignment.

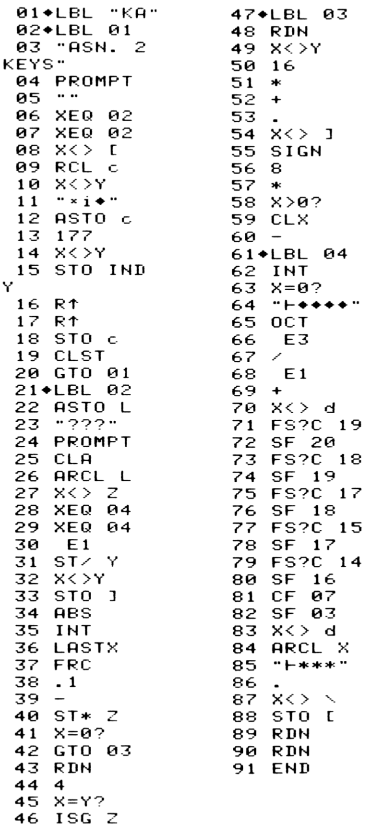
Then the machine places the curtain at R(ØØF) (this is the effect of line 12). The register R(ØCØ) to be filled has then the relative address R177, from where the recourse to indirect storing, then we reconstitute the initial state.

It is this part of the program that constitutes its originality. Indeed, in general, one satisfies himself by placing the curtain in R(ØCØ), a simple STO ØØ will then suffice for filling the register. But if at this moment the program is interrupted, or SST, there results Memory Lost! Indeed the HP-41 tests for the existence of the register immediately under the curtain: there is always at least one program register at this place, the one of .END.

If the curtain is in R(ØCØ) there is nothing below and the HP-41 reacts brutally.

One can execute this program indefinitely and thus enter as many assignments as desired (within the limits of the calculator). Each time you need to make two assignments manually before executing "KA".

Watch out: If you have some keys assigned beforehand and you have deassigned some of them before using KA, the program will probably not work. It is much better, at least at the beginning, to use KA on a machine without any key assignments other than the two used by KA to make the assignments.



```
 01♦LBL "KA"
 02♦LBL 01
 03 "ASN. 2
KEYS"
 04 PROMPT
 05 ""
 06 XEQ 02
 07 XEQ 02
 08 X<> [
 09 RCL c
 10 X<>Y
 11 "×i♦"
 12 ASTO c
 13 177
 14 X<>Y
 15 STO IND
Y
 16 R↑
 17 R↑
 18 STO c
 19 CLST
 20 GTO 01
 21♦LBL 02
 22 ASTO L
 23 "???"
 24 PROMPT
 25 CLA
 26 ARCL L
 27 X<> Z
 28 XEQ 04
 29 XEQ 04
 30  E1
 31 ST/ Y
 32 X<>Y
 33 STO ]
 34 ABS
 35 INT
 36 LASTX
 37 FRC
 38 .1
 39 -
 40 ST* Z
 41 X=0?
 42 GTO 03
 43 RDN
 44 4
 45 X=Y?
 46 ISG Z
 47♦LBL 03
 48 RDN
 49 X<>Y
 50 16
 51 *
 52 +
 53 .
 54 X<> ]
 55 SIGN
 56 8
 57 *
 58 X>0?
 59 CLX
 60 -
 61♦LBL 04
 62 INT
 63 X=0?
 64 "⊢♦♦♦♦"
 65 OCT
 66  E3
 67 /
 68  E1
 69 +
 70 X<> d
 71 FS?C 19
 72 SF 20
 73 FS?C 18
 74 SF 19
 75 FS?C 17
 76 SF 18
 77 FS?C 15
 78 SF 17
 79 FS?C 14
 80 SF 16
 81 CF 07
 82 SF 03
 83 X<> d
 84 ARCL X
 85 "⊢***"
 86 .
 87 X<> \
 88 STO [
 89 RDN
 90 RDN
 91 END
```

KA Program Listing

FOR BAR CODE SEE APPENDIX XIV

FIGURE 18

COMMENTARY OF THE KA PROGRAM

The line Ø3 is "ASN. 2 KEYS". You will have to assign any function to the two keys that you are planning to assign with the program. This results in updating the key indexes in register e and reserving a space in the register Ø1Ø that KA will replace by the synthetic assignment.

Two interesting assignments: The BG and eGØBEEP.

The BG that we have created places in the assignment register the bytes F7 and 3F (decimals 247/63), but try to assign F7 8Ø (247/128). With the same XROM, same display, here with the card reader you have the appearance of CARD READER, an easy identification (F7 8Ø = XROM 3Ø,ØØ).

eGØBEEP is the assignment XX/167, XX being between Ø and F (Ø to 15 decimal). It is an "unexpected function" like the byte grabber. This function has the property of creating (and eventually executing) the functions XROM 28 and 29, the functions of the HP-IL cassette and of the printers.

This function displays eGØBEEP __. If you answer the prompts by a number from Ø to 41, you create an XROM 28, which corresponds to the HP-IL. For instance, the number 12 gives XROM 28, 12, which is function RENAME. If the IL is present as well as the cassette, this function executes itself (or writes rename in a program). Otherwise, we have a nonexistent (or XROM 28,12 in the program).

Between 42 and 63 there are no functions for eGØBEEP's prompts. From 64 to 89, one obtains XROM 29 (number 64) which are the functions of the printer.

Watch out! 89 is FMT (format) and exists only on the HP-IL; furthermore, we can also assign and "execute" the titles mass storage or printer....with varying fortune, usually bad.

FIGURE 19

5.5 The Heavy Artillery

When there are many synthetic program lines that need to be created, the BG becomes tedious. On top of that, it is not possible to obtain certain codes for rows C through E of the table. The following program written by Lionel Ancele and published in the "Journal" of the PPC-Toulouse, creates in memory all the necessary bytes. It is a version of a well-known program in the Club with the classic name LB (Load Bytes). See Figure 20.

5.5.1 Use of "LB"

After having introduced LB into program memory, do GTO.. Enter in program mode LBL"++" followed by more than 12 +'s. You must have 12 +'s over the number of synthetic bytes you wish to introduce, then XEQ"LB", END. Be sure not to have the .END. too close or you might overwrite it accidentally. BST to the XEQ "LB" line and go out of PRGM Mode. Then press R/S. After a few seconds, a number in the form 1,nnn is displayed. That means that you are asked to give the first byte of a program that might contain nnn bytes. If you did not enter enough +'s, an alpha chain or an odd number is displayed. Enter then the decimal value of the byte you wish to introduce, R/S and so on.

```
01◆LBL "LB"
02 FS? 48
03 GTO 00
04 AON
05 CLST
06  E
07 RDN
08 GTO "++"
09◆LBL 00
10 AOFF
11 RCL b
12 STO [
13 "⊢◆◆◆"
14 RCL [
15 X<> d
16 CF 16
17 FS?C 07
18 SF 16
19 FS?C 06
20 SF 07
21 FS?C 05
22 SF 06
23 FS?C 04
24 SF 05
25 X<> d
26 FRC
27 STO [
28 LASTX
29 INT
30 R↑
31 +
32  E
33 -
34 STO 00
35 7
36 MOD
37 ST- 00
38  E3
39 ST/ 00
40 RCL [
41 X<> d
42 FS? 12
43 SF 03
44 X<> d
45 "θ◆×i"
46 X<> [
47 STO \
48 "⊢◆◆◆"
49 RCL \
50 STO 01
51 FIX 3
52 SF 22
53 CLA
54 GTO 00
55◆LBL 01
56 SF 22
57  E
58 RCL 00
59 INT
60 X=Y?
61 GTO 02
62 7
63 MOD
64 X<>Y
65 X=Y?
66 LASTX
67 ST- 00
68  E1
69 ARCL X
70 .
71 X<> \
72 STO [
73◆LBL 02
74 RCL 00
75 FS?C 22
76 STOP
77 FC? 22
78 .
79 256
80 MOD
81 LASTX
82 +
83 OCT
84 X<> d
85 FS?C 11
86 SF 12
87 FS?C 10
88 SF 11
89 FS?C 09
90 SF 10
91 FS? 07
92 SF 09
93 FS? 06
94 SF 08
95 X<> d
96 X<> [
97 "⊢◆"
98 STO \
99 "⊢◆"
100 RCL \
101 CLA
102 STO [
103 RCL 00
104 INT
105 7
106 MOD
107 X≠0?
108 GTO 00
109 RCL 00
110 FRC
111  E3
112 *
113 RCL 00
114 INT
115 -
116 7
117 /
118 INT
119  E
120 +
121 RCL 01
122 X<> c
123 RCL [
124 STO IND
Z
125 X<>Y
126 STO c
127 CLA
128◆LBL 00
129 ISG 00
130 GTO 02
131 END
```

LB Program Listing

FOR BAR CODE SEE APPENDIX XV

FIGURE 20

If you don't want to enter any bytes, do R/S without introducing anything. The program then enters the terminal phase.

When "LB" stops, a SST brings you back to the LBL "++". The only thing left to do is to check your program and backarrow the remaining unnecessary commands.

There is a correction procedure. If you made an error, do XEQ Ø1 instead of the R/S and restart on the number of the requested byte, 1 step or 1 register of 7 bytes backwards depending on the case.

5.5.2 Description of LB

When you XEQ"LB", the machine is not in the alpha mode, thus the flag 48 is clear. Thus the execution jumps to step **Ø4**. AON sets the flag 48, CLST, E, RDN, erases the stack and places 1 in register T. The execution is then transferred to LBL"++"; the remaining +'s assure the counting of the bytes, and we come to step XEQ"LB". This pushes into the b register the return address of the XEQ in compacted form and the execution is given back to the LB program. Since this time the flag 48 is set, the execution jumps to the LBL ØØ. AOFF clears flag 48, and the commands 11 to 39 calculate the available bytes. The commands 4Ø through 5Ø decode the return address, the location of the XEQ "LB" instruction and use it to build a temporary c register, to store the address of a RØØ temporary register. The synthetic bytes will be grouped by sevens and stored register by register.

The main loop of the program is located on the lines 73 through 13Ø: register RØØ contains the byte pointer; its contents are displayed at line 76 when the program stops to ask for a byte. If R/S has been punched without entering a byte, the value Ø is used (lines 77 and 78). Lines 79 and 8Ø make sure that the number entered is in the range Ø through 255, and the lines 81 through 1Ø2 build in alpha the character corresponding to the decimal code entered. (For those who have the X-Function module, these lines can be replaced by XTOA.)

Then the program looks to see if one has completed enough to constitute a full register of 7 bytes. If this is the case, the address relative to the temporary RØØ register is calculated on lines 1Ø9 through 12Ø, then the contents of c are modified so as to place RØØ in the right location (lines 121 and 122). Lines 123 and 124 store the register constituted of the 7 last bytes which were entered at the address previously calculated, then the contents of c is restored to its initial value. Alpha is then cleared and if it is still possible to enter bytes, we continue.

Steps 55 to 72 contain the correction procedure. Register RØØ is decremented, and the last character of the alpha register is erased.

Finally, the hexadecimal code of line 45 is:

```
F4 1Ø ØØ Ø1 69
```

The sequence LBL "++" + +....XEQ "LB" is tedious to key in. Think about copying it on a card; it is easy then to work on the last program (the end is the .END.) and to enter the sequence by doing XEQ"MRG". Maybe by doing this you will finally discover a use for the card reader function MRG.

5.6 In Case of Disaster

It might happen (more often than you may desire) that, because of a harmless manipulation the HP-41 does not react or reacts in an abnormal way. One can either reset or cause memory lost to return to normal functioning.

Reset is a function of the microprocessor originally provided by Hewlett-Packard to be accessible to the user. On an older model, one must pull out the batteries and put them back. Make sure the printer is disconnected as it will supply power to the HP-41. (In fact, except for the display, the HP-41 works very well without batteries when the printer is connected.) In some very serious cases, the HP-41 must remain without batteries for many days.

On recent models, one must press the back-arrow key and give one or two presses to the ON key. If CLX appears, it is unlocked.

Memory lost is obtained by pressing and holding the backarrow key then pressing and releasing the ON key.

CHAPTER 6

MICROCODE

MICROCODE

Now we will explore the very depths of the calculator. As already explained, you may be frustrated by reading this chapter, because you may not own the necessary hardware required for using microcode.

Don't forget that Hewlett-Packard will not provide any help or material concerning microcode. All that follows is the work of the PPC Club and accounted for by the PPC Club. For further information and "NOMAS" Hewlett-Packard publications on microcode, the interested user must contact PPC. On a closer view, these conditions are not really as harsh as they may seem.

6.1 HOW TO USE IT

It might seem paradoxical to discuss how to use microcode before having seen it, but this point is very important.

Let us first say that it is materially impossible to program in microcode in the live RAM of the HP-41; one must do it then in the depth of memory. However, two types of hardware allow for this type of program:

- An EPROM Reader
- A ROM Simulator

The first hardware, EPROM Reader, is available from several retail sources (see addresses for manufacturers in Appendix IV). The price is slightly higher than a magnetic card reader, the size normally approximates that of the HP-41 to which it is connected via a standard Hewlett-Packard printer cable; this is the only Hewlett-Packard original part of most EPROM readers. One EPROM unit available from retail sources is self-contained within a standard Hewlett-Packard magnetic card reader case. This unit, of course, plugs into port 4 but may be internally addressed to any other desired port via internal switches.

EPROM readers allow for the reading of commands previously recorded on the EPROMs which are located in the box. EPROMs are readily interchanged in all EPROM readers.

The memories used are standard commercial integrated circuits, called EPROMs (Erasable Programmable Read Only Memories). These memories can be erased by ultraviolet light and reprogrammed many times over. They are programmed by means of special hardware called EPROM programmers. There are some very cheap EPROM programmers on the market generally to be assembled by amateurs, but most are not very easy to use. Professional hardware is very expensive and generally must be connected to a computer. There are also a few in-between models of reasonable prices for the amateur, but their use is limited. It is in the interest of the amateur to generally go to a club or a professional offering EPROM programming services (Appendix V). PPC-T has an EPROM programmer as do several local PPC chapters throughout the world.

These EPROMs once programmed are placed in the EPROM reader. The EPROM reader then works exactly like a standard Hewlett-Packard plug-in module (Math, PPC-ROM, Statistics, etc.).

In an EPROM, one can also record programs programmed in our regular "user" language. These programs are then available at will and free the main RAM registers of the HP-41 for other uses.

One can also enter microcode programs and use them exactly as if they were standard internal functions, LOG, SDEV, +, etc.

An EPROM holds up to 32K bytes, the equivalent of eight full modules, depending on the model...just think about it.

The second type of hardware which is also available through several retail sources is the ROM Simulator (see addresses for manufacturers in Appendix IV).

It consists of a live RAM memory that can be programmed directly from the HP-41, and appears to the HP-41 as a ROM plug-in module. It is therefore a very interesting system. There are, however, some drawbacks. The major one is that it is a cumbersome, hard-to-carry unit, especially if made by an amateur. Also, it normally has only a 4K memory, equivalent to one module. Some 8K memory units are available, and one unit in particular may be stacked to provide the user 32K of memory. This configuration, however, would be quite cumbersome and very expensive. All these units have one of the same drawbacks of the internal RAM memory: the possibility of erasure in case of prolonged power failure. It is, however, possible to record some short programs on magnetic cards or the entire memory contents on a casette. Furthermore, this type of memory is immune to MEMORY LOST. This is, therefore, the ideal hardware for developing programs in microcode.

6.2 MICROCODE: WHAT IS IT?

You may have seen the Russian dolls containing another doll, and in turn containing another...your HP-41 is a little bit like that.

For instance, using the LB program, you are in dialogue with a program, not with the HP-41. The messages have been provided by the programmer, not by the HP-41. You have to get used to it; you are in dialogue with a program, not with the machine. If you do CLX, LN you will see data error. You are in dialogue with a program. If you use the math module or the PPC-ROM, you call for the programs names and wait for the outcome. When you use the function LN, + or ENTER, you do the same thing.

The commands themselves for the microcode programs are similar to LN or +. The difference is that these commands are understood by the HP-41 thanks to a particular arrangement of the transistors reacting to electrical impulses.

The different levels are as follows:

- The program "wired" internally within the microprocessor of the HP-41, which, when it receives electrical impulses on certain wires, reacts accordingly.
- A coded definition of these electrical impulses forming what is called microcode.
- A grouping of internal modules containing microcode programs reacting to the touch of the keyboard to obey these commands forming the functions that we use every day.
- The programs that we write in "User" language are simply our organized linkage of the microcode programs contained within these modules.

6.3 THE FUNCTIONING PRINCIPLE

Our machine is like a marmot. It goes to sleep each time a function is finished. This is called a "light sleep."

Do: 2, ENTER↑, 3, +, Output = 5. A few microseconds later, the microprocessor of the HP-41 is stopped. Only the display remains alive, but not for long. After a few minutes, the display will also go off; it is then in "deep sleep." But the slightest touch of the key awakes the monster! The HP-41 then starts a race to put all the memory in order before returning to the keyboard to see what is asked of it.

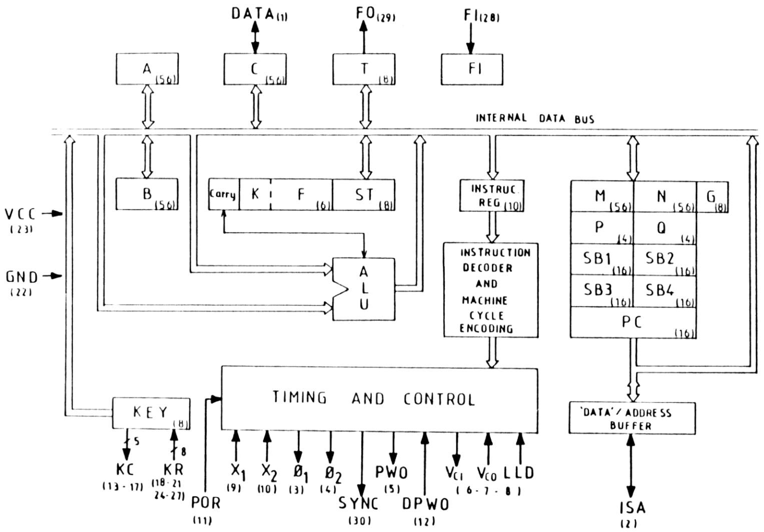
Depending on the key punched, a function is executed. When the work is complete, a new purging is done and the HP-41 goes back to sleep.

It should be noted that microcode does not know, for instance, the LN function, but it will notice that the key pressed was the fifth of the top row, that the user flag is not set, and that no program is being exe cuted. In that case, it will go ahead with the execution of the LN internal microcode program from a particular point of the memory, then back to sleep.

If a program is being executed, there is a continuous functioning, with, however, a purging being performed after each function is completed. This precautionary purging is the major part of the execution time of the function. It is this time economy that produces the quickness of execution of microcode (up to 1ØØØ times faster).

6.4 GEOGRAPHY OF THE MICROPROCESSOR

The microprocessor (CPU: Central Processing Unit) has, for its own needs, special memory registers. I will use here the notations (mnemonics) given to us by the American pioneers of the PPC club who have deciphered the whole thing.



The Microcode Registers and Instruction Flow

FIGURE 21

6.4.1 The Internal Registers

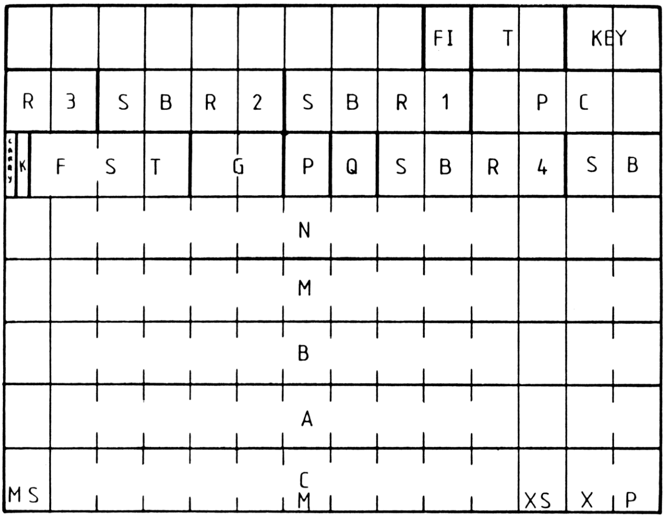
a. The "C" Register

This register consisting of 56 bits is the main register of the microprocessor memories, the forced passage point for all entries and exits. It plays, but more strongly, the same role that the X register of the stack usually plays. Most of the arithmetic calculations begin and finish with the C register. It is the principal accumulator.

b. The "A" and "B" Registers

These are also main registers consisting of 56 bits each working in relation with the C register or with themselves. The 56 bits of the C, A and B registers are divided into 14 digits of 4 bits each (4 x 14 = 56). Each digit can be a binary number from Ø through F Hex. The A and B registers are "almost" the equivalent of the Y and Z registers of the stack.

NOTE: There is no automatic register stack except SBR in the microprocessor; everything has to be foreseen and provided for by the programmer.



Microcode Register Map

FIGURE 22

The C, A and B registers are each 56 bits long, as are any of the user registers of the HP-41. As in the user registers, digit 13 is named MS (mantissa sign), digits 12 to 3 are named M (Mantissa), digit 2 is named XS (Exponent Sign), digits 2 through Ø are named X (Exponent). In addition to this, two supplementary fields can also be distinguished: the digits 4 and 3 named KY (Key Buffer) and a zone consisting of digits 6 to 3 named ADR (Address). Later, we will see the use of these fields as well as the complete microcode commands to use these only internal registers of the HP-41 in which arithmetic can be performed. All arithmetic, logical and I/O operations are performed by Registers C, A and B.

c. The M and N Registers

Registers M and N are used for temporary storage. These are both main registers of 56 bits each, but they cannot do arithmetic or logical operations. They can only input storage (STO) from C, recall (RCL) or exchange (EX) their contents with the contents of C and this only for the entirety of the register.

d. The G Register

The G Register consists of 8 bits. This register allows the user to store, recall, or exchange one single byte chosen at random from the C Register. The G Register is used for temporary storage of one selected byte.

e. The P&Q Registers

The P and Q Registers each consist of 4 bits. Peculiar registers, these are the pointers. This means that a particular digit of Registers C, A or B can be pointed to by a number placed in P or Q. Since registers P and Q each have 4 bits, they can contain a binary number from Ø through F Hex.

The HP-41 cannot work simultaneously with both P and Q, the user must first choose between P or Q. The chosen pointer is then designated by the letter R (one can thus have R = P or R = Q).

The contents of R can be used to designate a particular digit of Register C, A or B. For example, a command could be: Is the digit whose number is in R null?

Both registers P and Q can serve as an iterative adder, similar to the I register of certain other Hewlett-Packard calculators. It is possible to increment (or decrement) the selected R register by 1 and test the contents for "Do if: else" conditionals.

f. <u>The PC Register and SBR Registers</u>

The PC and SBR registers each consist of 16 bits and thus can address 65,536 locations. We have seen that the HP-41 at the user code level has a program pointer and six registers (levels) of returns for subprograms. Likewise, the microprocessor has a subroutine register (SBR) and a Program Counter (PC) register. The PC register contains the line number of the ROM microcode being executed. It is normally incremented after each instruction is executed, but may be altered by several of the microcode instructions. There are four SBR registers, each holding a subroutine address. Subroutine return addresses can either be pushed onto or popped from the SBR registers on a LIFO (Last In, First Out) basis.

g. The ST Register

ST is called the Status Register, not to be confused with the "User" status registers. It is simply a flag register of 14 bits. Fourteen flags are each represented by one bit, where 1 = Set and Ø = Clear.

As the "User" function (from the X-Function Module) X<>F allows an exchange between X and a part of the user flags register (d), it is also possible to exchange the ST register's 8 rightmost bits (7 to Ø) with the C register's rightmost 8 bits (7 to Ø). Therefore, it is possible to set (1) and clear (Ø) all 14 system flags. This exchange with C can also serve to test the flags from the "User" d register.

The 6 flags 13 through 8 have, in the internal registers, the precise following use:

Flag 13 Set = A program is being executed.

Flag 12 Set = The program is private.

Flag 11 Set = The stack lift (XYZT) is enabled.

Flag 1Ø Set = The program pointer is in ROM.

Flag 9 Set = General Use.

Flag 8 Set = General Use.

Note especially the use of flag 1Ø allowing the splitting of memory between RAM and ROM as we have seen in Chapter 2. Most of the time, the ST Flags 7 to Ø hold what Hewlett-Packard calls the "Byte Ø of the Flags," that is to say flags 48 to 55 of status register d (in positions 7 to Ø), which control important aspects of the machine's state.

h. The T Register

The T Register consists of 8 bits. All transfers to the T Register must be accomplished through the ST Register. The T register seems to have the following behavior:

Inputting - ØØ Hex the acoustical bender (beeper) is silent. Inputting FF Hex, an impulse is sent to the beeper. Tones are created by alternating ØØ Hex and FF Hex into the T Register.

The frequency is obtained by varying the time between two exchanges and the duration of the tone by varying the number of exchanges performed.

i. The Key Register

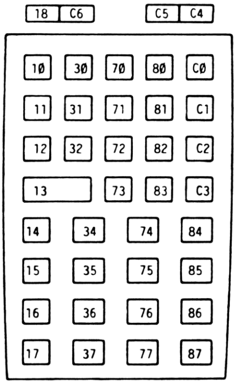
The Key Register consists of 8 bits. By punching a key, a code is placed in that register. The number placed in the Key Register is shown in Figure 23.

j. The Carry Register

The Carry Register is one (1) bit long. It is a special flag register that is set (1) by a carry or borrow, and cleared (Ø) otherwise.

Only the field concerned by the operation is considered (maybe a digit of the pointer, at XS or MS, or maybe at M, XP...). The carry bit is set if an addition gives an outcome higher than the number formed by putting a 1 in all the bits of the field (1111, if there is only

	1	3	7	8	C
Ø	A	B	C	D	E
1	F	G	H	I	J
2	sh.	K	L	M	SST
3	N	☒	O	P	←
4	Q	R	S	T	Alpha
5	U	V	W	X	Prgm
6	Y	Z	=	?	User
7	:	space	,	R/s	☒
8	on				



Value found in
KEY KY by
pressing a
column-row
keycode.

KEY Register Map

FIGURE 23

one digit; 1111 1111, if SP is in question, etc.). The carry bit is also set if a subtraction gives an outcome below zero. The Carry Flag can be set in response to a test (set carry if...).

NOTE: In microcode, the HP-41 does additions as well in hexadecimal as in decimal. By working in decimal, Carry is set by addition when the largest number able to be written in the field is exceeded.

Most important, the carry flag is cleared by any instruction that does not set it. It can only be tested by the instruction immediately following the one that set it.

k. There are other registers, required by the internal functioning. They are not accessible and, therefore, not necessary to describe.

6.5 MICROCODE COMMANDS

We have said already that microcode commands are all coded in 1Ø bits.

The two rightmost bits (1 and Ø) of the commands determine the type of command. There are thus only four possible types of commands: Ø, 1, 2 and 3.

In the tables, the commands are shown using the 244 codes. In the program listings of this book, both coding ways are indicated using the format 244-442.

6.5.1 Type Ø Commands

This command type consists of: parameter (4 bits), instruction (4 bits), type (2 bits). The parameter serves to define on what the command is acting. See Figure 24.

Example:

ØØ1Ø ØØØ1 ØØ means:
Parameter 2, Instruction 1, Type Ø and is translated as:
Clear Flag 5

This command might be written as follows:
442 Code: 21Ø
244 Code: Ø84

Note that the commands of classes 6/Ø (13/Ø), 8/Ø and C/Ø combine with their parameter to make distinct commands. They are given in Figures 25 and 26.

These commands 6/Ø (identical to the 13/Ø commands) are the ones giving access to the registers G, M, T and ST.

The byte in the G register is the one made up by the digit located at the position in C designated by the pointer and the preceding digit.

The 8/Ø instructions are varied. Note the command XQ-GO which lowers the return stack of the subprogram one notch, bypassing the closest return address and placing Ø in SBR 4.

GTO ADR replaces the usual GTO IND and sends the program to a calculated address previously placed in the ADR zone of G.

DSP ON must be done by the sequence DSP OFF, DSPTOG.

There are many important commands in the last table (Figure 26).

PUSH ADR places the content of the ADR field of register C in the return stack (SBR Registers). This return stack acts automatically just as the "User" stack XYZT. When using PUSH, the contents of the stack raises one notch to make room for the new address. The content of SBR 4 is lost.

POP ADR. When this command is executed, the contents of the first register of the stack (SBR 1) is placed in the ADR field of Register C and the SBR stack lowers one notch. Contrary to the "User" stack, XYZT there is no duplication of SBR 4 which is set to zero.

Sample:

```
Before:  SBR 4 : Ø123   After:  SBR 4 : ØØØØ
         SBR 3 : 6742           SBR 3 : Ø123
         SBR 2 : 234Ø           SBR 2 : 6742
         SBR 1 : 1572           SBR 1 : 234Ø
         C ADR : 6ØØ1           C ADR : 1572
         The 6ØØ1 is lost.
```

Notes on the SBR Registers (SBR Stack): This stack is of the LIFO (last-in-first-out) type. When given the instruction XQ, the microprocessor puts the return address in the stack. In other terms, the address placed in the stack is the address of the word following the XQ instruction. In so doing, it is possible to pass parameters to a subprogram (for instance, a number of loops to be performed, the code of the character to be displayed, or the code of an error message). It is easy to place after the call to the subprogram.

This parameter can be retrieved by the FETCH S&X (to fetch) instruction used after a POP of the SBR registers.

RAM SLCT allows the programmer to select any register of RAM. This selection is accomplished from a three-digit address: it, therefore, appears possible to select 4Ø96 (FFF) registers. In fact, we know by experience, only the 1Ø rightmost bytes respond, limiting to 1Ø24 the actual number of accessible registers.

PRPH SLCT. Select the peripheral whose number is indicated in C(XP). We are now in an "uncertain zone." It looks like the digit XS has no influence on the chosen PRPH. But...this is not certain.

RAM is a special peripheral. It must be invalidated before selection of any other peripheral by selecting an empty address, in general Ø1Ø (sometimes 2FD).

The display is peripheral Number FD.
The card reader is peripheral Number FC.
The wand is peripheral Number FE.
The time module is peripheral Number FB.

The commands WRITE and READ send or receive data to or from a peripheral.

In the case of RAM, the parameter indicates the location of the first register in the group of 16 registers selected.

In the case of the display, refer to Appendix X.

The other peripherals (IL, 82143 PRINTER, etc.) use contact commands with the registers of the selected peripheral.

Reading: If n is the register number of the called peripheral, the HP-41 uses three commands:

n9Ø SELP n
nE2 These instructions read the nØ3 register and put it in C S&X (the exponent sign and exponent field).

Writing of the Variables: n being the number of the destination register on the HP-IL module coded in 3 bits, the HP-41 uses a command which is a NOP of the microprocessor but which is accepted by the peripheral: in binary, 1nnnØØØØØØ.... This instruction writes the C S&X contents to register n of the peripheral. For example, to write the contents of the S&X field of register C to register 2, use

AØØ

Writing of the Constants: First word: n9Ø (Code 442) where n is the peripheral or register number. Second word: Consists of the byte to be written followed by the 2 bits ØX. If X is Ø, the peripheral retains control; if X is 1, the HP-41 retains control.

Sample: Read Register 3 of the HP-IL module

```
39Ø
3E2
3Ø3
```

Write E2 to register Ø:

```
Ø9Ø
E21
```

Flag tests: The instruction DØ3 tests flag 3 of the selected peripheral and sets the HP-41's carry flag if the peripheral flag was set.

The ROM listings distributed by the PPC club are mostly incorrect on these points.

For more details on the peripheral registers, see the HP Journal of January 1983 (official Hewlett-Packard International newsletter, available from all major Hewlett-Packard dealerships or the PPC club) entirely dedicated to the HP-IL loop.

6.5.2 Type 1 Commands

These commands consist of two successive words in one program. They consist of skips to an address (GOTO) or to a subprogram (XEQ) given by the 16-bit address of the first line (there is no LBL instruction in microcode).

Since there are 16 bits for the address, one can skip to any point of memory (Figure 27).

6.5.3 Type 2 Commands

These are the arithmetic and logic operations. They can be executed in hexadecimal or decimal mode and are only executed within the selected field. (Figure 28).

6.5.4 Type 3 Commands

Relative Jumps: the value of the jumps is added to the number of the starting line to obtain the number of the last line. By hand, one counts starting from Ø on the starting line (JNC or JC); don't forget to count in hexadecimal (good intellectual exercise). The negative jumps are counted by their complements (Figure 27).

Taking into account the method of notation of the negative jumps by complements and of the 7 bits giving the length of the jump it is possible to jump up to 64 words. In fact +63 (hex) and -64 (hex) words.

Type Ø Commands

*Cannot be executed immediately after an arithmetic command.

```
Word:  P3 P2 P1 PØ   I3 I2 I1 IØ   Ø Ø
         Parameter      Command    Type
```

n	Bits	Mnemonic	Remarks
Ø	ØØØØ	NOP	No Operation - Parameter Not Utilized
1	ØØØ1	*CLRF f	Clear Flag f (1)
2	ØØ1Ø	*SETF f	Set Flag (2)
3	ØØ11	*?FSET f	Clear carry if flag f is cleared (3)
4	Ø1ØØ	LD@R-d	Put d in C at R pointer (4)
5	Ø1Ø1	*?R=f	Clear carry if pointer is at f (5)
6	Ø11Ø	MISC	See Table 6/Ø
7	Ø111	R=f	Place the selected pointer at f
8	1ØØØ	MISC	See Table 8/Ø
9	1ØØ1	SELP d	Select the peripheral d (7)
1Ø	1Ø1Ø	WRIT r	Write C to the selected peripheral
11	1Ø11	?FI f	Test the flag peripheral
12	11ØØ	MISC	See Table C/Ø
13	11Ø1	MISC	See Table 6/Ø
14	111Ø	Read r	Read a peripheral and input in C (8)
15	1111	RCR f	Rotate C f hexadecimal digits to the right (9)

NOTES:

1 [f = F(15)] except 3C4= ST=Ø clears the ST register.
2 [f = F(15)] except 3C8= CLRKEY clears the keyboard K flag.
3 [f = F(15)] except 3CC= ?KEY clears the CARRY bit if flag K is clear.
4 [f = F(15)] subtract 1 from the pointer value after execution.
5 [f = F(15)] except 3D4= R= R+1 decrement the selected pointer.
6 [f = F(15)] except 3DC= R= R+1 increment the selected pointer.
7 The CPU in inactive during the execution of the special commands.
8 Read Ø(T) is also written READ DATA.
9 RCR 15 = Reset display (display immobilized 24 cycles).

n	bits	d	f	r
Ø	ØØØØ	Ø	3	Ø(T)
1	ØØØ1	1	4	1(Z)
2	ØØ1Ø	2	5	2(Y)
3	ØØ11	2	A(1Ø)	3(X)
4	Ø1ØØ	4	8	4(L)
5	Ø1Ø1	6	6	5(M)
6	Ø11Ø	6	B(11)	6(N)
7	Ø111	7		7(O)
8	1ØØØ	8	2	8(P)
9	1ØØ1	9	9	9(Q)
1Ø	1Ø1Ø	A(1Ø)	7	1Ø(+)
11	1Ø11	B(11)	D(13)	11(a)
12	11ØØ	C(12)	1	12(b)
13	11Ø1	D(13)	C(12)	13(c)
14	111Ø	E(14)	Ø	14(d)
15	1111	F(15)		15(e)

The decoding of all these commands is the work of Paul Lind, Jim De Arras, and Steven Jacobs, all members of PPC.

Microcode Type Ø Commands

FIGURE 24

Type Ø, Under Classes 6/Ø (and 13/Ø) and 8/Ø

Code 244	Mnemonic	Remarks
Ø18		Not Used
Ø58	G=C @R, +	Puts the C's, R and R+1 digits in G
Ø98	C=G @R, +	Puts G in C R and R+1
ØD8	C<>G @R, +	Exchanges G and two C digits
118		Not Used
158	M=C All	Puts C into M
198	C=M All	Puts M into C
1D8	C<>M All	Exchange C with M
218		Not Used
258	T=ST	Puts in T the contents of ST
298	ST=T	Puts in ST the contents of T
2D8	ST<>T	Exchange contents of T and ST
318		Not Used
358	ST=C XP	Puts in ST the C exponent
398	S=ST XP	Puts in C (XP) the ST register
3D8	*C<> ST XP	Exchange contents of C (XP) and ST

Note that ST consists of the flags 7 through Ø. All other flags must be cleared or set individually.

Code 244	Mnemonic	Remarks
Ø2Ø	XQ-GO	Transform SX in GO while POP the SBR stack
Ø6Ø	POWOFF	Stops the CPU (no effect on the display) (1)
ØAØ	SLCT P	P as active pointer
ØEØ	SLCT Q	Q as active pointer
12Ø	?P=Q	Clears CARRY flag if P=Q
16Ø	?LOWBAT	Test battery (clear CARRY flag if stack is insufficient)
1AØ	A=B=C=Ø	Erases A, B and C
1EØ	GOTO ADR	Puts in PC the C's ADR digits
22Ø	C=Key KY	Puts in C's KY the contents of Key KY
26Ø	SETHEX	Puts the CPU in the hexamode
2AØ	SETDEC	Puts the CPU in decimal mode
2EØ	DSPOFF	Display Off
32Ø	DSPTOG	Toggle the status of the Display (On<>Off)
36Ø	?C RTN	Return if CARRY is clear
3AØ	?NC RTN	Return if CARRY is set
3EØ	RTN	Imperative return

Note 1: POWOFF is a 2 byte command, the second byte always being ØØØ NOP.

Microcode Type Ø, Class 6/Ø and 8/Ø

FIGURE 25

Type Ø Under Class C/Ø

Code 244	Mnemonic	Remarks
Ø3Ø		Not Used
Ø7Ø	N=C ALL	Put C in N
ØBØ	C=N ALL	Put N in C
ØFØ	C<>N ALL	Exchange of C and N
13Ø	LDI S&X	Load immediate: Put in C S&X the word immediately following.
17Ø	PUSH ADR	Puts C's ADR in SBR1
1BØ	POP ADR	Puts SBR1 in C's ADR
1FØ		Not Used
23Ø	GOTO KEY	Puts in PC the Key content
27Ø	RAM SLCT	Select the RAM Address at C (S&X)
2BØ	???	Clear data registers
2FØ	WRITE DATA	Write C to the selected A Peripheral
33Ø	FETCH S&X	Puts in C (S&X) the word located at the ADR address
37Ø	*C=C OR A	Or Logic
3BØ	*C=C AND A	And Logic
3FØ	PRPH SLCT	Select the peripheral which N° is in C(S&X)

LOGIC TABLE OF FUNCTIONS

OR

C	A	C or A
1	1	1
1	Ø	1
Ø	1	1
Ø	Ø	Ø

AND

C	A	C or A
1	1	1
1	Ø	Ø
Ø	1	Ø
Ø	Ø	Ø

Microcode Type Ø, Class C/Ø

FIGURE 26

Type 1: GO/XQ Absolute

First Word: A7 A6 A5 A4 A3 A2 A1 Ø 1
Second Word: A15 A14 A13 A12 A11 A1Ø A9 A8 G C

G= GO or XQ C= Condition

G	C	Mnemonics	Remarks
Ø	Ø	?NC XQ	Executes if CARRY is set
Ø	1	?C XQ	Executes if CARRY is clear
1	Ø	?NC GO	Goes if CARRY is set
1	1	?C GO	Goes if CARRY is clear

Type 3: Relative Skips

S6 S5 S4 S3 S2 S1 SØ C 1 1
C = Condition

C	Mnemonics	Remarks
Ø	JNC Xss	Skips if CARRY is set
1	JC Xss	Skips if CARRY is clear

x = sign
ss = length of skip (HEXA) (+63/-64)

EXAMPLES:

Code 244	Mnemonic	Remark
3F3	JNC -Ø2	Skips backward two words
ØEF	JC +1D	Skips forward 1D (29) words

For the complements, do it in Hexadecimal 8Ø - Skip (complement of 2).

Microcode Type 1 and 3 Commands

FIGURE 27

```
Type 2:  Arithmetic and Logic Operations
Word:  14 13 12 11 1Ø       C2 C1 CØ      1 Ø
            Command           Field       Type
```

n	Bits	Mnemonic	Remarks
Ø	ØØØØØ	A=Ø	Clears the A register
1	ØØØØ1	B=Ø	Clears the B register
2	ØØØ1Ø	C=Ø	Clears the C register
3	ØØØ11	A<>B	Exchange A with B
4	ØØ1ØØ	B=A	Puts A in B
5	ØØ1Ø1	A<>B	Exchange A with C
6	ØØ11Ø	C=B	Puts B in C
7	ØØ111	C<>B	Exchange C&B
8	Ø1ØØØ	A=C	Put C in A
9			
1Ø			
11	Ø1Ø11	A=A+1	Increments A
12			
13	Ø11Ø1	A=A-1	Decrements A
14			
15	Ø1111	C=C+C	Multiply the C register by 2. It is also a binary shift of 1 bit to the left.
16			
17	1ØØØ1	C=C+1	Increments C
18			
19	1ØØ11	C=C-1	Decrements C
2Ø	1Ø1ØØ	C=Ø-C	Arithmetic Complement
21	1Ø1Ø1	C=-C-1	Ø-C-1 Complement
22	1Ø11Ø	?B≠Ø	Clears CARRY if B is not equal to Ø
23	1Ø111	?C≠Ø	Clears CARRY if C is not equal to Ø
24	11ØØØ	?A<C	Clears CARRY if A is less than C
25	11ØØ1	?A<B	Clears CARRY if A is less than B
26	11Ø1Ø	?A≠Ø	Clears CARRY if A is not equal to Ø
27	11Ø11	?A≠C	Clears CARRY if A is not equal to C
28	111ØØ	RSHFA	Shift A right 1 digit
29	111Ø1	RSHFB	Shift B right 1 digit
3Ø	1111Ø	RSHFC	Shift C right 1 digit
31	11111	LSHFA	Shift A one digit to the left

Definition of the Field

n	bits	Mnemonic	Remarks
Ø	ØØØ	@R	On the digit specified by R
1	ØØ1	S&X	Sign and exponent of C
2	Ø1Ø	R<	Right of the pointer
3	Ø11	All	All 14 digits
4	1ØØ	P-Q	Between the pointers
5	1Ø1	XS	The sign of the exponent
6	11Ø	M	Mantissa alone (digits 12-3)
7	111	MS	The sign of the mantissa (13)

Microcode Type 2 Commands **FIGURE 28**

CHAPTER 7

USING MICROCODE

USING MICROCODE

Thanks to the ingeniousness of the PPC members, all the Hewlett-Packard ROM modules including the internal modules have been deciphered (disassembled), we could say "taken to pieces." However, knowing the name of the command is not enough; one must know what its purpose is.

There are over 5ØØ pages of disassembled listings. We could not finish this study without analyzing some of these elements, chosen somewhat arbitrarily as illustrations.

7.1 LOGIC GEOGRAPHY OF A MODULE

As with the HP-41, a module has its own internal logic geography, or programs. The following is intended to describe the external plug-in modules. The internal modules are similar in operation but physically different in size and shape. As mentioned before, each 4K module consists of 4Ø96 1Ø bit words. Each 4K module begins at Address XØØØ and ends at Address XFFF, X being a hex digit from Ø through F depending on the module's page assignment.

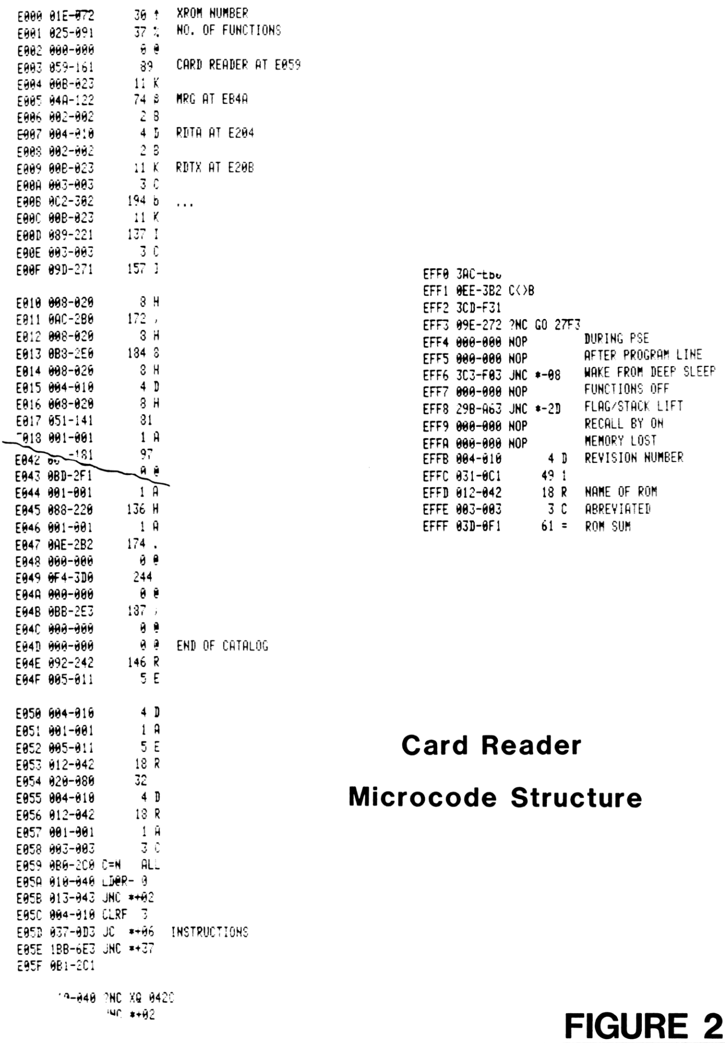
a. Address XØØØ contains the XROM number of the module. Address XØØ1 contains the number of functions existing in the module. Remember all numbers are hex. Each 4K module may contain 64 functions.

b. From the Address XØØ2 continuing to, and ending at two (2) NOPS (ØØØ - ØØØ) is the catalog of the functions contained within the module. Each cat- alog entry consists of two words per function, giving both the program type and the function's entry address within the ROM module.

The first two bits of the first word indicate the program type. They are ØØ if it is a program is a microcode function, and 1Ø if the program is in standard "user" language.

The four last bits of the first word and the eight last bits of the second word give the three hex digit address of the program residing inside the module.

By executing a CAT 2, the addresses of the functions are read in the order in which they appear in the module catalog. It is thus possible to list the functions in alphabetic order if one has organized the catalog properly. Whatever the disposition of the programs in the module are, the first function listed is generally the name of the module. This is not necessary, but it does provide a means of identification. The module name is, however, counted as one of the 64 maximum functions. As such, this name has an XROM number. It normally is a RTN which only displays the name during a Catalog 2 execution. The name generally does



```
E000 01E-072      30 ↑  XROM NUMBER
E001 025-091      37 %  NO. OF FUNCTIONS
E002 000-000       0 @
E003 059-161      89    CARD READER AT E059
E004 00B-023      11 K
E005 04A-122      74 8  MRG AT EB4A
E006 002-002       2 B
E007 004-010       4 D  RDTA AT E204
E008 002-002       2 B
E009 00B-023      11 K  RDTX AT E20B
E00A 003-003       3 C
E00B 0C2-302     194 b  ...
E00C 00B-023      11 K
E00D 089-221     137 I
E00E 003-003       3 C
E00F 09D-271     157 ]

E010 008-020       8 H
E011 0AC-2B0     172 ,
E012 008-020       8 H
E013 0B3-2E0     184 8
E014 008-020       8 H
E015 004-010       4 D
E016 008-020       8 H
E017 051-141      81
E018 001-001       1 A
E042 00 -181      97
E043 0BD-2F1       0 @
E044 001-001       1 A
E045 088-220     136 H
E046 001-001       1 A
E047 0AE-2B2     174 .
E048 000-000       0 @
E049 0F4-3D0     244
E04A 000-000       0 @
E04B 0BB-2E3     187 ,
E04C 000-000       0 @
E04D 000-000       0 @  END OF CATALOG
E04E 092-242     146 R
E04F 005-011       5 E

E050 004-010       4 D
E051 001-001       1 A
E052 005-011       5 E
E053 012-042      18 R
E054 020-080      32
E055 004-010       4 D
E056 012-042      18 R
E057 001-001       1 A
E058 003-003       3 C
E059 0B0-2C0 C=N   ALL
E05A 010-040 LD@R- 0
E05B 013-043 JNC **02
E05C 004-010 CLRF  3
E05D 037-0D3 JC   *+06  INSTRUCTIONS
E05E 1BB-6E3 JNC *+37
E95F 0B1-2C1

        10-040 ?NC XQ 042C
               JNC *+02
```

```
EFF0 3AC-EB0
EFF1 0EE-3B2 C<>B
EFF2 3CD-F31
EFF3 09E-272 ?NC GO 27F3
EFF4 000-000 NOP              DURING PSE
EFF5 000-000 NOP              AFTER PROGRAM LINE
EFF6 3C3-F03 JNC *-08         WAKE FROM DEEP SLEEP
EFF7 000-000 NOP              FUNCTIONS OFF
EFF8 29B-A63 JNC *-2D         FLAG/STACK LIFT
EFF9 000-000 NOP              RECALL BY ON
EFFA 000-000 NOP              MEMORY LOST
EFFB 004-010          4 D     REVISION NUMBER
EFFC 031-0C1         49 1
EFFD 012-042         18 R     NAME OF ROM
EFFE 003-003          3 C     ABREVIATED
EFFF 03D-0F1         61 =     ROM SUM
```

Card Reader

Microcode Structure

FIGURE 29

not respond to an XEQ instruction. This is not to say, though, that it cannot also be a function if so desired.

The running of a module's portion of catalog 2 is terminated when two NOPS (ØØØ - ØØØ) are encountered (Figure 29).

c. At the very end of a 4K module, at Addresses XFFB, XFFC, XFFD and XFFE is the ROM abridged name, first letter in FFE, last in FFB (Figure 29).

d. The last module word is the ROM checksum (optional).

e. Prior to the ROM name are seven special and very important addresses called polling points. These addresses are questioned by a fetch. If they contain Ø, the questioning program continues without paying any attention to it; otherwise, it branches to that address.

The address:	is questioned when:
XFF4	Pause loop
XFF5	Main running loop
XFF6	Deep sleep wake up, no key down
XFF7	Off
XFF8	I/O Service
XFF9	Deep sleep wake up
XFFA	Cold Start (memory lost)

f. Between the two NOPS which are at the end of all catalogs and Address XFF4 are the codes that comprise the following:

-- Programs in User language

-- Programs in microcode (functions)

-- Data

A program in User language is in the following form:

-- Two words "copy information." The first word gives the number of required registers to copy the program in RAM. I don't know exactly the meaning of the second word that is always (mode 244) 22Ø, 23Ø, 24Ø or 26Ø.

-- The point of entry of the program is the first byte of the alphabetic label designating the program name. It is imperative that this label be in the front of the program.

-- After the entry point is the list of program bytes, including the GTO and XEQ compilation bytes.

The two left bits of the word, useless to the program, are set at Ø1 to indicate the first byte of a multi-byte command or the sole byte of single byte command, the rest of the time is set to zero (ØØ).

An exception to this rule is that the last byte of the program (and third byte from the END) is equal to 22F.

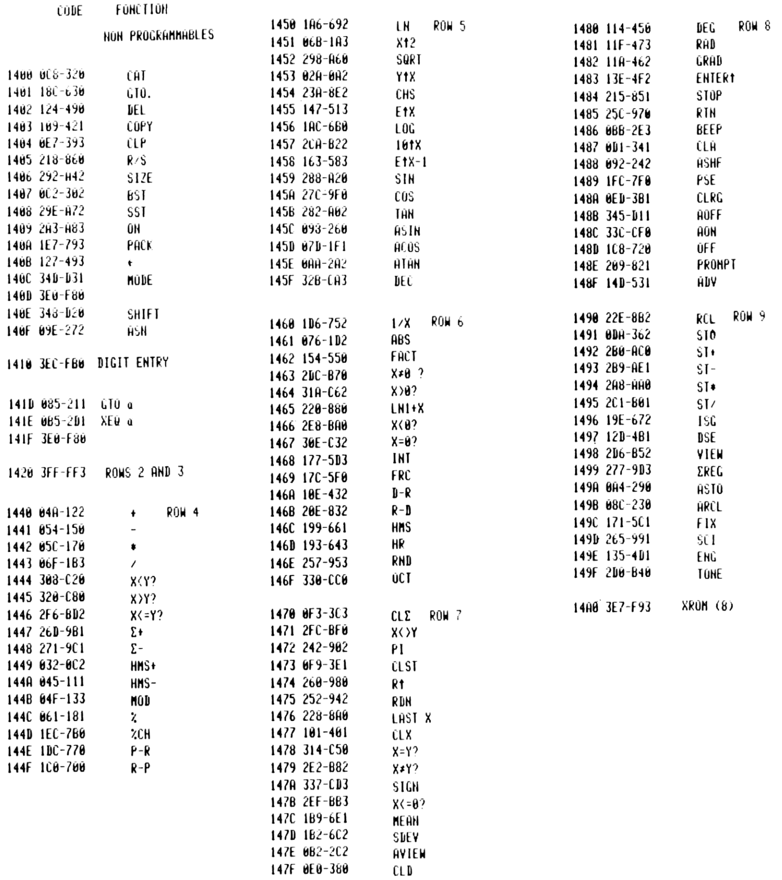
Microcode commands and data are not distinct. Only the way of using the code makes the difference as we will see in the examples.

The internal functions of the HP-41 are accessed by the machine's decoding of the key pressed. But the ROM functions are executed from the entry point.

This function entry point gives access to two types of information:

- Reading from bottom to top, the codes preceding the entry point represent the characters composing the name of the function, coded in microcode (Figure 1Ø).

  The HP-41 knows that the last character is reached when that code is increased by Ø8Ø.



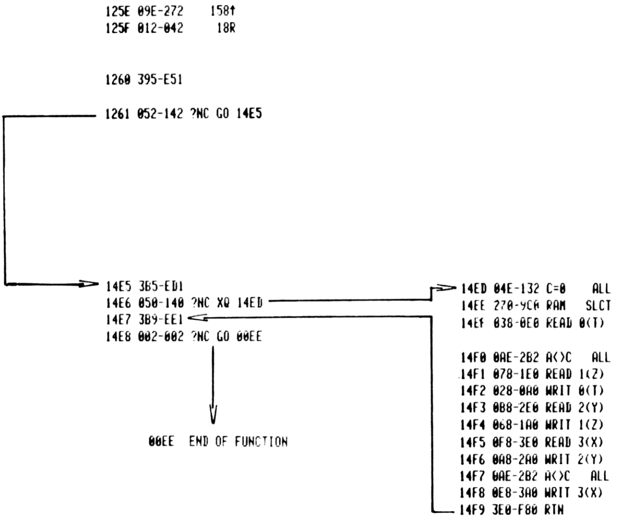
```
      CODE     FUNCTION

               NON PROGRAMMABLES

1400 0C8-320      CAT
1401 18C-630      GTO.
1402 124-490      DEL
1403 109-421      COPY
1404 0E7-393      CLP
1405 218-860      R/S
1406 292-A42      SIZE
1407 0C2-302      BST
1408 29E-A72      SST
1409 2A3-A83      ON
140A 1E7-793      PACK
140B 127-493      +
140C 34D-D31      MODE
140D 3E0-F80
140E 348-D20      SHIFT
140F 09E-272      ASN

1410 3EC-FB0  DIGIT ENTRY


141D 085-211  GTO a
141E 0B5-2D1  XEQ a
141F 3E0-F80

1420 3FF-FF3   ROWS 2 AND 3

1440 04A-122       +     ROW 4
1441 054-150       -
1442 05C-170       *
1443 06F-1B3       /
1444 308-C20       X<Y?
1445 320-C80       X>Y?
1446 2F6-BD2       X<=Y?
1447 26D-9B1       Σ+
1448 271-9C1       Σ-
1449 032-0C2       HMS+
144A 045-111       HMS-
144B 04F-133       MOD
144C 061-181       %
144D 1EC-7B0       %CH
144E 1DC-770       P-R
144F 1C0-700       R-P
```

```
1450 1A6-692        LN    ROW 5
1451 06B-1A3        X↑2
1452 298-A60        SQRT
1453 02A-0A2        Y↑X
1454 23A-8E2        CHS
1455 147-513        E↑X
1456 1AC-6B0        LOG
1457 2CA-B22        10↑X
1458 163-583        E↑X-1
1459 288-A20        SIN
145A 27C-9F0        COS
145B 282-A02        TAN
145C 093-260        ASIN
145D 07D-1F1        ACOS
145E 0AA-2A2        ATAN
145F 32B-CA3        DEC


1460 1D6-752        1/X   ROW 6
1461 076-1D2        ABS
1462 154-550        FACT
1463 2DC-B70        X≠0 ?
1464 31A-C62        X>0?
1465 220-880        LN1+X
1466 2E8-BA0        X<0?
1467 30E-C32        X=0?
1468 177-5D3        INT
1469 17C-5F0        FRC
146A 10E-432        D-R
146B 20E-832        R-D
146C 199-661        HMS
146D 193-643        HR
146E 257-953        RND
146F 330-CC0        OCT

1470 0F3-3C3        CLΣ   ROW 7
1471 2FC-BF0        X<>Y
1472 242-902        PI
1473 0F9-3E1        CLST
1474 260-980        R↑
1475 252-942        RDN
1476 228-8A0        LAST X
1477 101-401        CLX
1478 314-C50        X=Y?
1479 2E2-B82        X≠Y?
147A 337-CD3        SIGN
147B 2EF-BB3        X<=0?
147C 1B9-6E1        MEAN
147D 1B2-6C2        SDEV
147E 0B2-2C2        AVIEW
147F 0E0-380        CLD
```

```
1480 114-450       DEG    ROW 8
1481 11F-473       RAD
1482 11A-462       GRAD
1483 13E-4F2       ENTER↑
1484 215-851       STOP
1485 25C-970       RTN
1486 0BB-2E3       BEEP
1487 0D1-341       CLA
1488 092-242       ASHF
1489 1FC-7F0       PSE
148A 0ED-3B1       CLRG
148B 345-D11       AOFF
148C 33C-CF0       AON
148D 1C8-720       OFF
148E 209-821       PROMPT
148F 14D-531       ADV

1490 22E-8B2       RCL   ROW 9
1491 0DA-362       STO
1492 2B0-AC0       ST+
1493 2B9-AE1       ST-
1494 2A8-AA0       ST*
1495 2C1-B01       ST/
1496 19E-672       ISG
1497 12D-4B1       DSE
1498 2D6-B52       VIEW
1499 277-9D3       ΣREG
149A 0A4-290       ASTO
149B 08C-230       ARCL
149C 171-5C1       FIX
149D 265-991       SCI
149E 135-4D1       ENG
149F 2D0-B40       TONE

14A0 3E7-F93     XROM (8)
```

HP-41C Function Entry Addresses

FIGURE 30



```
125E 09E-272    158↑
125F 012-042     18R


1260 395-E51

1261 052-142 ?NC GO 14E5





14E5 3B5-ED1                         14ED 04E-132 C=0    ALL
14E6 050-140 ?NC XQ 14ED             14EE 270-9C0 RAM   SLCT
14E7 3B9-EE1                         14EF 038-0E0 READ 0(T)
14E8 002-002 ?NC GO 00EE
                                     14F0 0AE-2B2 A<>C   ALL
                                     14F1 078-1E0 READ 1(Z)
                                     14F2 028-0A0 WRIT 0(T)
                                     14F3 0B8-2E0 READ 2(Y)
                                     14F4 068-1A0 WRIT 1(Z)
     00EE  END OF FUNCTION           14F5 0F8-3E0 READ 3(X)
                                     14F6 0A8-2A0 WRIT 2(Y)
                                     14F7 0AE-2B2 A<>C   ALL
                                     14F8 0E8-3A0 WRIT 3(X)
                                     14F9 3E0-F80 RTN
```

R↓ Microcode Example

FIGURE 31

- Reading from top to bottom following the entry point are the words of the program, the program is in User Code or microcode.

7.2 FIRST EXAMPLE: R↑

This function can be found in the operating system ROM Number 1 in a table located in 14xx (Figure 31). This table gives the addresses of all the internal functions. In Figure 3Ø, you can see from left to right the line number from (14xx), followed by the code in both 244 and 442 notation.

You will find R↑ at line 1474 with a 26Ø code. Add a 1 in front of the code and you will have the address of the R↑ function, or 126Ø. Now go back to Figure 31.

The 126Ø line is just below the name of the function, which you can read from bottom to top. The blank space is there because of the passage of 125F to 126Ø, corresponding to a paragraph change.

The ↑ code is 1E, since it is the name's last character it is increased by Ø8Ø which gives Ø9E (Ø1E + Ø8Ø = Ø9E).

The execution thus starts at line 126Ø.

You may be frustrated to find out that a new address is found where the execution has to go, that is 14E5.

This "subroutine" itself is to execute a subprogram before ending in ØØEE, one of the starting points of the routines to terminate all functions.

The control is thus the subprogram starting at 14ED. If you program in microcode and you want to execute R↑ it is enough to call that subprogram located at 14ED.

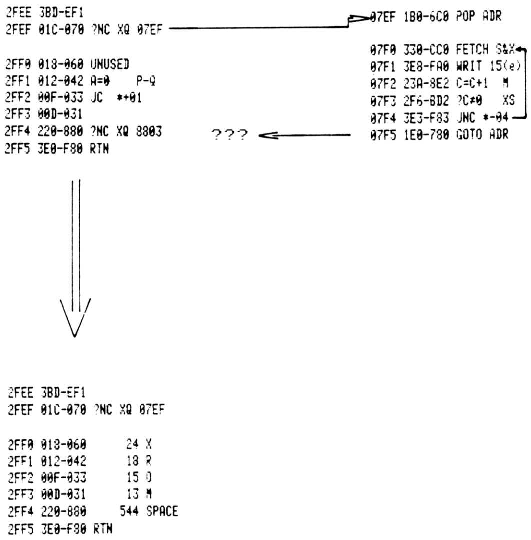
This subprogram starts by selecting the R(ØØØ) register. Then it temporarily stores the contents of the T register in the A register. Then we see the stack raise: Z goes into T, Y into Z, X into Y and finally recovering T to put it into X. Simple, is it not?

7.3 SECOND EXAMPLE: OPERATION OF THE DISPLAY

The display is one of the several peripherals of the HP-41. To activate it and use it, the following is required:

```
Ø7F6 13Ø-4CØ LDI S&X
Ø7F7 Ø1Ø-Ø4Ø Ø1Ø 16P
Ø7F8 27Ø-9CØ RAM SLCT
Ø7F9 13Ø-4CØ LDI S&X
Ø7FA ØFD-3F1 ØFD 253
Ø7FB 3FØ-FCØ PRPH SLCT
Ø7FC 3EØ-F8Ø RTN
Ø7FD ØØØ-ØØØ NOP
```

As you can see, RAM memory must first be invalidated by selecting a now existing register, Ø1Ø, the one located just over the e register, in the empty register area. After that, the display address



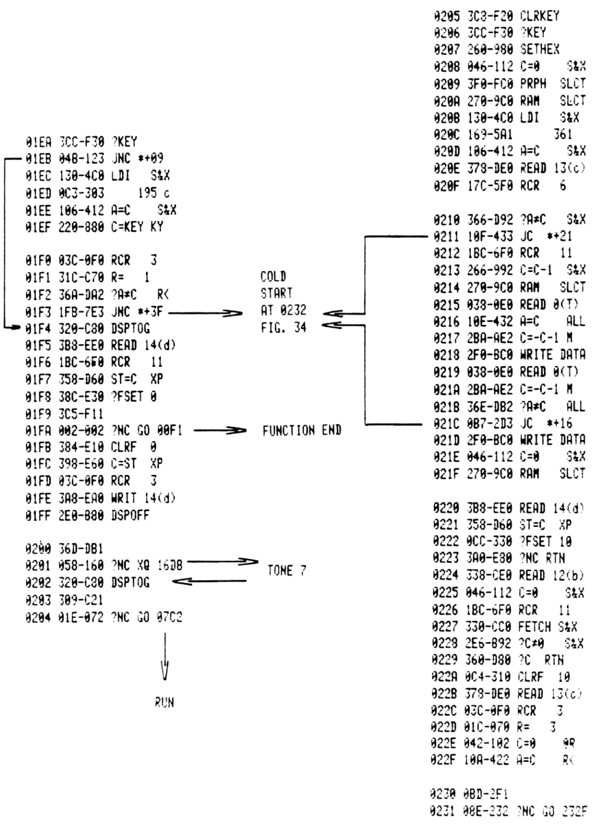
```
2FEE 3BD-EF1
2FEF 01C-070 ?NC XQ 07EF

2FF0 018-060 UNUSED
2FF1 012-042 A=0    P-Q
2FF2 00F-033 JC  *+01
2FF3 00D-031
2FF4 220-880 ?NC XQ 8803
2FF5 3E0-F80 RTN
```

```
07EF 1B0-6C0 POP ADR

07F0 330-CC0 FETCH S&X
07F1 3E8-FA0 WRIT 15(e)
07F2 23A-8E2 C=C+1   M
07F3 2F6-BD2 ?C≠0    XS
07F4 3E3-F83 JNC *-04
07F5 1E0-780 GOTO ADR
```

```
2FEE 3BD-EF1
2FEF 01C-070 ?NC XQ 07EF

2FF0 018-060      24 X
2FF1 012-042      18 R
2FF2 00F-033      15 O
2FF3 00D-031      13 M
2FF4 220-880     544 SPACE
2FF5 3E0-F80 RTN
```

Message Codes

FIGURE 32



```
01EA 3CC-F30 ?KEY
01EB 048-123 JNC *+09
01EC 130-4C0 LDI    S&X
01ED 0C3-303      195 c
01EE 106-412 A=C     S&X
01EF 220-880 C=KEY KY

01F0 03C-0F0 RCR    3
01F1 31C-C70 R=    1
01F2 36A-DA2 ?A≠C    R<
01F3 1FB-7E3 JNC *+3F
01F4 320-C80 DSPTOG
01F5 3B8-EE0 READ 14(d)
01F6 1BC-6F0 RCR    11
01F7 358-D60 ST=C   XP
01F8 38C-E30 ?FSET 0
01F9 3C5-F11
01FA 002-002 ?NC GO 00F1
01FB 384-E10 CLRF   0
01FC 398-E60 C=ST   XP
01FD 03C-0F0 RCR    3
01FE 3A8-EA0 WRIT 14(d)
01FF 2E0-880 DSPOFF

0200 36D-DB1
0201 058-160 ?NC XQ 16D8
0202 320-C80 DSPTOG
0203 309-C21
0204 01E-072 ?NC GO 07C2
```

```
0205 3C8-F20 CLRKEY
0206 3CC-F30 ?KEY
0207 260-980 SETHEX
0208 046-112 C=0     S&X
0209 3F0-FC0 PRPH   SLCT
020A 270-9C0 RAM    SLCT
020B 130-4C0 LDI    S&X
020C 169-5A1        361
020D 106-412 A=C     S&X
020E 378-DE0 READ 13(c)
020F 17C-5F0 RCR    6

0210 366-D92 ?A≠C    S&X
0211 10F-433 JC   *+21
0212 1BC-6F0 RCR    11
0213 266-992 C=C-1  S&X
0214 270-9C0 RAM    SLCT
0215 038-0E0 READ 0(T)
0216 10E-432 A=C     ALL
0217 2BA-AE2 C=-C-1 M
0218 2F0-BC0 WRITE DATA
0219 038-0E0 READ 0(T)
021A 2BA-AE2 C=-C-1 M
021B 36E-DB2 ?A≠C    ALL
021C 0B7-2D3 JC   *+16
021D 2F0-BC0 WRITE DATA
021E 046-112 C=0     S&X
021F 270-9C0 RAM    SLCT

0220 3B8-EE0 READ 14(d)
0221 358-D60 ST=C   XP
0222 0CC-330 ?FSET 10
0223 3A0-E80 ?NC RTN
0224 338-CE0 READ 12(b)
0225 046-112 C=0     S&X
0226 1BC-6F0 RCR    11
0227 330-CC0 FETCH S&X
0228 2E6-892 ?C≠0    S&X
0229 360-D80 ?C   RTN
022A 0C4-310 CLRF   10
022B 378-DE0 READ 13(c)
022C 03C-0F0 RCR    3
022D 01C-070 R=    3
022E 042-102 C=0     @R
022F 10A-422 A=C     R<

0230 0BD-2F1
0231 08E-232 ?NC GO 232F
```

Message Routine

FIGURE 33

(ØFD) is loaded into the S&X field of the C register, and the display is then selected. From that moment on, writing to the display is possible. You will find in the Appendix the detail of the commands required.

7.4 MESSAGES

Sometimes we find in memory codes like the one in Figure 32 on top left. This group of commands starts with XQ Ø7EF; let's see what that subprogram does (on the right side of the figure).

A POP brings into C's ADR field the first return address. We know that address; it's the one that is just below the XQ Ø7EF, thus 2FFØ. The HP-41 fetches the contents of this address and puts it in the S&X field of the C register.(That is, XS becomes Ø and X becomes 18.) This code appears on the display (previously selected) as characters, thus it was not a command!

The command: C=C+1 adds 1 to the address which becomes 2FF1 and the HP tests the XS: is it different from zero? If yes, load CARRY. It is not the case here, thus the HP-41 will go back to the FETCH instruction.

Thus, the codes Ø18, Ø12, ØØF, ØØD are sent in sequence to the display then finally 22Ø. Then there is a non-zero value (2) in the XS field, thus we are out of the loop. The GTO ADR will cause execution to continue the execution. AT 2FF5 (this value is in the ADR field of the C register).

Address 2FF5 contains a RTN and ends the subprogram by return to the principal program. But what are those codes doing in the display? We can tell by correcting the listing to have the characters appear. See the bottom of Figure 32.

7.5 HOT OR COLD START?

We will now investigate a particular portion of the HP-41 starting procedure (Figure 33). At Ø1EA, the machine is just awakened from "deep sleep". After performing all the housekeeping chores, it again returns to see if any key is pressed (other than ON). Otherwise, it skips to Ø1F4.

If a key was pressed, it tests to see which key it is. It loads ØC3 in the S&X field of register C then in the S&X field of register A, then takes the key register contents and moves them to the S&X field of the C register where it can compare it to the ØC3 now in A, this comparison being done on digits 1 and Ø.

If the two elements are not different, it is because the punched key has the C3 code, the one...guess or see Figure 23. In that case, go to the cold start (Memory Lost) routine. If it is another key, we come back in the case where no key was punched located at Ø1F4.

If no key or a key other than backarrow was pressed at turn-on, the code at Ø1F4 is executed. It is then time to light the display and to verify quickly if Flag 11 (autostart) is set. To check that case, read the d status register, rotate Flag 11 to the right place to be put into the ST register and test it.

If this flag is not set, the calculator can back to light sleep. If the flag is set, start executing the current user program.

We are then at location Ø2Ø4. It is logical to continue through Ø2Ø5. However, in the machine chronology this passage was executed before the one that we talk about above. Nothing is simple.

First, in that passage (called CHECKRAM, test of the RAM) it did an updating of the keyboard and the CPU calculating methods. Next was a deactivation of the peripherals and the selection of RAM Ø, the status registers, in order to be able to read status register c and verify if cold start constant (that 169 that was just put in register A) is present in status register c. If not, a cold start is performed.

RCR 11 rotates into the S&X field of the C register the address of RAM register ØØ and subtracts 1 to get the location of the beginning of the programs. If the HP-41 is functioning normally, there must be at least one register containing the .END., the following commands will verify that.

When there is nothing at the beginning of the programs, a reading of the registers will always give a series of 1's or a series of Ø's in register C. But this situation might also exist in a normal register, how can you tell the difference?

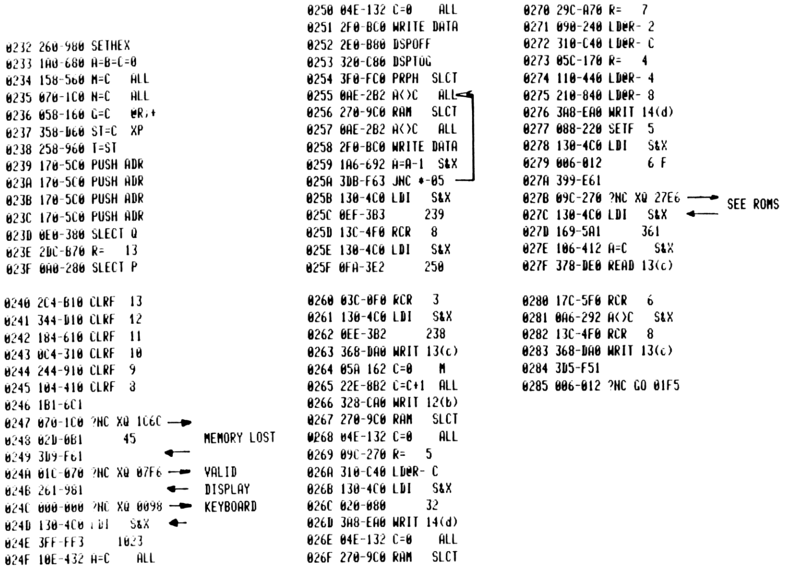
The HP-41 takes the complement of a part of the C register and writes the outcome in the selected register (taking the complement is to replace all the Ø's by 1's and all the 1's by Ø's).

A new reading will give the same contents if the register is good, there will be thus at the same time Ø's and 1's in the C register. If the register is bad we will find the same thing returned as in the first read.

The second taking of the complement will thus establish the content of a correct register and undo that of a bad register, controlled by the following word. If the register is bad, the HP-41 will perform a cold start, otherwise it will write data to reestablish the original contents in the register.

Put back in ST the system flags (48 to 55), and if the program is not in ROM, the routine is complete.

Otherwise, the HP-41 proceeds with a peculiar operation: it verifies that during deep sleep someone has not removed the module in which the program pointer is located.



```
0232 260-980 SETHEX
0233 1A0-680 A=B=C=0
0234 158-560 M=C    ALL
0235 070-1C0 N=C    ALL
0236 058-160 G=C    @R,+
0237 358-D60 ST=C   XP
0238 258-960 T=ST
0239 170-5C0 PUSH ADR
023A 170-5C0 PUSH ADR
023B 170-5C0 PUSH ADR
023C 170-5C0 PUSH ADR
023D 0E0-380 SLECT Q
023E 2DC-B70 R=    13
023F 0A0-280 SLECT P

0240 2C4-B10 CLRF  13
0241 344-D10 CLRF  12
0242 184-610 CLRF  11
0243 0C4-310 CLRF  10
0244 244-910 CLRF  9
0245 104-410 CLRF  8
0246 1B1-6C1
0247 070-1C0 ?NC XQ 1C6C  →
0248 02D-0B1        45             MEMORY LOST
0249 3D9-F61              ←
024A 01C-070 ?NC XQ 07F6  →        VALID
024B 261-981              ←        DISPLAY
024C 000-000 ?NC XQ 0098  →        KEYBOARD
024D 130-4C0 LDI    S&X   ←
024E 3FF-FF3        1023
024F 10E-432 A=C     ALL
```

```
0250 04E-132 C=0     ALL
0251 2F0-BC0 WRITE DATA
0252 2E0-B80 DSPOFF
0253 320-C80 DSPTOG
0254 3F0-FC0 PRPH  SLCT
0255 0AE-2B2 A<>C    ALL
0256 270-9C0 RAM   SLCT
0257 0AE-2B2 A<>C    ALL
0258 2F0-BC0 WRITE DATA
0259 1A6-692 A=A-1  S&X
025A 3DB-F63 JNC *-05
025B 130-4C0 LDI    S&X
025C 0EF-3B3        239
025D 13C-4F0 RCR    8
025E 130-4C0 LDI    S&X
025F 0FA-3E2        250

0260 03C-0F0 RCR    3
0261 130-4C0 LDI    S&X
0262 0EE-3B2        238
0263 368-DA0 WRIT 13(c)
0264 05A 162 C=0     M
0265 22E-8B2 C=C+1  ALL
0266 328-CA0 WRIT 12(b)
0267 270-9C0 RAM    SLCT
0268 04E-132 C=0     ALL
0269 09C-270 R=    5
026A 310-C40 LD@R- C
026B 130-4C0 LDI    S&X
026C 020-080        32
026D 3A8-EA0 WRIT 14(d)
026E 04E-132 C=0     ALL
026F 270-9C0 RAM    SLCT
```

```
0270 29C-A70 R=    7
0271 090-240 LD@R- 2
0272 310-C40 LD@R- C
0273 05C-170 R=    4
0274 110-440 LD@R- 4
0275 210-840 LD@R- 8
0276 3A8-EA0 WRIT 14(d)
0277 088-220 SETF  5
0278 130-4C0 LDI    S&X
0279 006-012        6 F
027A 399-E61
027B 09C-270 ?NC XQ 27E6  →
                                 SEE ROMS
027C 130-4C0 LDI    S&X   ←
027D 169-5A1       361
027E 106-412 A=C     S&X
027F 378-DE0 READ 13(c)

0280 17C-5F0 RCR    6
0281 0A6-292 A<>C    S&X
0282 13C-4F0 RCR    8
0283 368-DA0 WRIT 13(c)
0284 3D5-F51
0285 006-012 ?NC GO 01F5
```

"MEMORY LOST" Routine

FIGURE 34

So if we read the pointer in status register b and load Ø into the three rightmost digits, only the port number remains. The address XØØØ, the first ROM word, is put into ADR and READ. If there is anything else than Ø, the HP-41 assumes that the ROM is still there and ends the control. If there is Ø, it sends the pointer back to the beginning of the RAM program.

This analysis suggests a felony to me. The HP-41 verifies the existence of a ROM, but which one? Here is an occasion to get into microcode.

Therefore, a ROM having a program in user language and another ROM having at the same address a microcode program are needed. I tested it with the PPC ROM and the X-Function ROM.

Place in one port the PPC ROM. Do GTO "MK", shut off the machine, and replace the PPC ROM with the X-Function ROM.

Turn the HP-41 on, turn PRGM ON, you will see Ø1 STO Ø3. There are funny things in that module! SST will continue to display the same Ø1 STO Ø3, but BST gives 01 P-R, try LIST...Have fun.

Let's come back to microcode. We now approach the routine of the HP-41 cold start. You may get a chill (Figure 34).

The 7 first lines set everything at zero! Then Ø's are loaded into all four levels of the SBR stack. Place the pointer Q at R = 13, but leave pointer P in place where it is.

All the flags are then erased (set to Ø), and the error message routine is called at 1C6C. The next word used routinely indicates the dreadful "MEMORY LOST."

Note that in microcode the memory lost can be displayed without erasing anything simply XQ 1C6C...a trap.

Ø7F6 activates the display, ØØ98 clears the keyboard register.

Now we come to a crucial point. The HP-41 loads 3FF. It is the address of the highest register to be erased. Hewlett-Packard was foreseeing the X memories already! If the Hewlett-Packard engineers had chosen 1FF instead, that could have saved the X memories from "MEMORY LOST." Advantage or inconvenience? Either way, it must be accepted.

The C=Ø WRITE DATA acts on the display by erasing the annunciators (BAT, USER...) After that the HP-41 erases the 3FF possible registers after having deactivated the display (C=Ø PRPH SLCT).

Then the value ØEF for RAM register RØØ, and ØEE for .END. have to be reestablished by default (no program) and ØFA for ΣREG.

The HP-41 sets the flags to their default status. Flag 50, the message flag (for the MEMORY LOST display) must be set. The HP-41 then checks the peripherals (subprogram 27E6). Then it loads the constant 169 for hot start indicating that all the work has been completed.

To conclude, it will check in Ø1F5 to see if a peripheral has not cleared the automatic execution flag (case of reading a magnetic card). The little HP-41 does quite a job!

7.6 A LITTLE BIT OF MUSIC

We will now see how the "tones" function."

You will see that the synthetic tones (postfix greater than 9) have a strange origin! (Figure 35).

The entry point for TONE is line 12DØ (after 149F). The HP-41 has already loaded into register ST the number of the tone to be executed. This number is saved in ST, while the HP-41 checks to see if it has the right to make noise (Flag 26).

Following this, it sets the T register to Ø to prepare the bender (beeper). Indeed it is the T register that commands the beeper by its variations. The tone number is once again saved in the ST register and FF placed in the XP field of the C register then swapped into the ST register, the tone number is brought into the M field of the C

```
12CC 085-211      133 E
12CD 00E-032       14 N
12CE 30F-C33      783 O
12CF 114-450      276 T

12D0 379-DE1
12D1 05A-162 ?NC GO 16DE
```

```
16DE 3B8-EE0 READ 14(d)
16DF 2BC-AF0 RCR   7

16E0 3D8-F60 C<>ST XP
16E1 30C-C30 ?FSET 1
16E2 3A0-E80 ?NC RTN
16E3 3C4-F10 ST=0
16E4 2D8-B60 ST<>T
16E5 358-D60 ST=C  XP
16E6 04E-132 C=0    ALL
16E7 130-4C0 LDI   S&X
16E8 0FF-3F3        255
16E9 3D8-F60 C<>ST XP
16EA 1BC-6F0 RCR   11
16EB 10E-432 A=C    ALL
16EC 07C-1F0 RCR   4
16ED 2A0-A80 SETDEC
16EE 2BE-AF2 C=-C-1 MS
16EF 260-980 SETHEX

16F0 3F1-FC1
16F1 058-160 ?NC XQ 16FC
16F2 05B-163         91
16F3 065-191        101
16F4 073-1C3        115
16F5 087-213       135 G
16F6 0A1-281       161 !
16F7 0C9-321       201 a
16F8 10B-423       267 K
16F9 13F-4F3       319 ?
16FA 18F-633       399 O
16FB 215-851       533 U
16FC 1B0-6C0 POP ADR
16FD 21A-862 C=C+A  M
16FE 330-CC0 FETCH S&X
16FF 0A6-292 A<>C   S&X
```

```
1700 27E-9F2 C=C-1  MS
1701 067-193 JC  *+0C
1702 27E-9F2 C=C-1  MS
1703 077-1D3 JC  *+0E
1704 27E-9F2 C=C-1  MS
1705 08F-233 JC  *+11
1706 11E-472 A=C    MS
1707 2D8-B60 ST<>T
1708 1BE-6F2 A=A-1  MS
1709 3FB-FE3 JNC *-01
170A 1A6-692 A=A-1  S&X
170B 3DB-F63 JNC *-05
170C 3E0-F80 RTN
170D 2D8-B60 ST<>T
170E 1A6-692 A=A-1  S&X
170F 3F3-FC3 JNC *-02

1710 3E0-F80 RTN
1711 2D8-B60 ST<>T
1712 000-000 NOP
1713 1A6-692 A=A-1  S&X
1714 3EB-FA3 JNC *-03
1715 3E0-F80 RTN
1716 2D8-B60 ST<>T
1717 000-000 NOP
1718 000-000 NOP
1719 1A6-692 A=A-1  S&X
171A 3E3-F83 JNC *-04
171B 3E0-F80 RTN
```

"TONE" Routine

FIGURE 35

register, then into the M field of the A register. A peculiar operation takes place now: don't forget that the number of the normal tone is a decimal from Ø to 9. The HP-41 takes a decimal complement of the tone number 9->Ø, 8->1, etc.

The tones all have the same duration but a variable frequency. A tone cycle of high frequency does not last as long as a tone cycle of low frequency. Thus, the number of tone cycles to be executed for a constant duration varies. The tone number allows us to choose that number (duration). That operation is realized in the MS field of the C register, in a single digit.

We will then choose from the table the required value (number of tone cycles). Using a very common procedure in the HP-41, we skip a zone of data to arrive at a POP ADR taking the 16F2 address and add the TONE number (from Ø to 9). You can now see TONE Ø uses Ø5B cycles, tone 1: Ø65, etc. This duration constant is loaded into the S&X (exponent and sign) field of the A register, which serves as the loop index.

Remember that we have in the MS field of the C register the tone frequency. This will have various ramifications.

For TONE 9, the MS field of the C register contains Ø. Subtracting 1, will clear the CARRY flag and we skip to 17ØD where we produce the number of exchanges between the ST register and the T register (thus values FF and ØØ) indicated by register A.

Here there are three lines to execute, the minimum possible, thus the highest frequency possible.

For TONE 8, the MS field of register C contains 1, we do not skip on the first subtraction, but on the second, and then go to 1711. We have the same principle, but a NOP is interpolated, thus a lower frequency. Two NOPS are used for TONE 7. For higher cycle durations, a double loop is used.

The synthetic tones (greater than 9) work as follows:

- The duration constant is taken in order of the tone numbers. TONE 1Ø (decimal) takes this as the constant to the POP ADR code...and so on, the tone duration depends on codes that were intended to be instructions.

- The value of the tone in the XP field may be greater than 9. I don't know how the microprocessor reacts when asked to take the decimal complement of a number greater than 9? I did not judge it useful to try it in microcode, listening is enough.

7.7 USING MICROCODE: REP AND XCAT

7.7.1 First a very simple example: REP

This function, available on an EPROM baptized as TOULROM-1B (ROM 1B made in Toulouse) has the objective of copying X into Y, Z and T in one command.

This operation is usually performed by ENTER↑, ENTER↑, ENTER↑. For example, if you do 1 REP +++... you will get a computer adding 1 at each +. It is the system used in LB to calculate the number of free bytes.

There are two ways to write a function (program) in microcode. You can build your function from the instructions of microcode, and this is perfectly possible but it requires a perfect knowledge of microcode and most of all a perfect analysis of the problem. Don't forget that with microcode we work without a safety net! In normal programming it is always possible to do R/S if the program refuses to get out of the cycle. In microcode, if you do not test the R/S key, the HP will not recognize it as being pressed.

The second method consists of extensive use of the internal operating system routines, kindly provided by Hewlett-Packard. The only nuisance is that you have to know them. This can be difficult considering the volumes of material on them that generally can only be done by collaboration with a club.

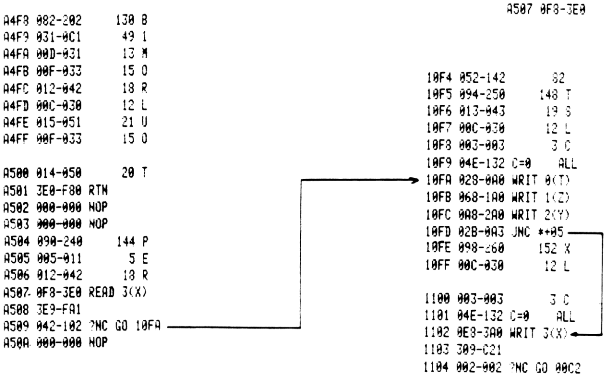
It is this last method that I use here, so that you will see how efficient it is.

The function REP, at least for the part that figures in the EPROM, is limited to its name, a read 3 (X) and a ?NC GO 1ØFA (Figure 36).

Difficult to make it simpler, no? The only thing done is to read the contents of X and load it into the C register.

But what is the ?NC GO 1ØFA? As you see in Figure 36, it is an entry point (not the usual one) into the CLST routine. This routine erases the stack by setting the C register to Ø and recopying C in the stack. REP uses only the recopy part of this routine.

Note that the CLST routine also uses part of the CLX routine.



```
A4F8 082-202      130 B
A4F9 031-0C1       49 I
A4FA 00D-031       13 M
A4FB 00F-033       15 O
A4FC 012-042       18 R
A4FD 00C-030       12 L
A4FE 015-051       21 U
A4FF 00F-033       15 O

A500 014-050       20 T
A501 3E0-F80 RTN
A502 000-000 NOP
A503 000-000 NOP
A504 090-240      144 P
A505 005-011        5 E
A506 012-042       18 R
A507 0F8-3E0 READ 3(X)
A508 3E9-FA1
A509 042-102 ?NC GO 10FA
A50A 000-000 NOP
```

```
A507 0F8-3E0

10F4 052-142        82
10F5 094-250      148 T
10F6 013-043       19 S
10F7 00C-030       12 L
10F8 003-003        3 C
10F9 04E-132 C=0     ALL
10FA 028-0A0 WRIT 0(T)
10FB 068-1A0 WRIT 1(Z)
10FC 0A8-2A0 WRIT 2(Y)
10FD 02B-0A3 JNC *+05
10FE 098-260      152 X
10FF 00C-030       12 L

1100 003-003        3 C
1101 04E-132 C=0     ALL
1102 0E8-3A0 WRIT 3(X)
1103 309-C21
1104 002-002 ?NC GO 00C2
```

REP Program Listing

FIGURE 36

7.7.2 A More Serious Application: XCAT

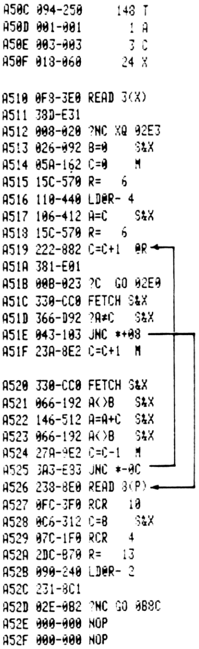
All of you who own many modules have been, like me, exasperated many times by the difficulty of listing the functions (programs) by means of CAT 2. One of my friends measured approximately 2-1/2 minutes for the total duration of his catalog 2.

XCAT has the feature of starting the CAT 2 on the module that you designate by its XROM number. For instance 3Ø XCAT starts the catalog with the card reader. Then the catalog proceeds normally. It is also possible to SST and BST at will.

This program (see Figure 37) starts with its name (read in reverse) as usual. It reads from the X register the number of ROM whose catalog is desired, and translates it into binary by means of the internal routine starting in Ø2E3.

This routine takes a decimal number of three digits in the C register and translates it into binary in C's S&X field.

XCAT then initializes registers C and B of the microprocessor. It loads the page number of the first possible module, less 1, in digit 6



```
A50C 094-250     148 T
A50D 001-001       1 A
A50E 003-003       3 C
A50F 018-060      24 X

A510 0F8-3E0 READ 3(X)
A511 38D-E31
A512 008-020 ?NC XQ 02E3
A513 026-092 B=0    S&X
A514 05A-162 C=0    M
A515 15C-570 R=   6
A516 110-440 LD@R- 4
A517 106-412 A=C    S&X
A518 15C-570 R=   6
A519 222-882 C=C+1  @R ←┐
A51A 381-E01            │
A51B 00B-023 ?C  GO 02E0│
A51C 330-CC0 FETCH S&X  │
A51D 366-D92 ?A≠C   S&X │
A51E 043-103 JNC *+08 ──┼─┐
A51F 23A-8E2 C=C+1  M   │ │
                        │ │
A520 330-CC0 FETCH S&X  │ │
A521 066-192 A<>B   S&X │ │
A522 146-512 A=A+C  S&X │ │
A523 066-192 A<>B   S&X │ │
A524 27A-9E2 C=C-1  M   │ │
A525 3A3-E33 JNC *-0C ──┘ │
A526 238-8E0 READ 8(P) ←──┘
A527 0FC-3F0 RCR   10
A528 0C6-312 C=B    S&X
A529 07C-1F0 RCR   4
A52A 2DC-870 R=   13
A52B 090-240 LD@R- 2
A52C 231-8C1
A52D 02E-0B2 ?NC GO 0B8C
A52E 000-000 NOP
A52F 000-000 NOP
```

XCAT Program Listing

FIGURE 37

of C. This page number is 4, because the first page normally used is 5. XCAT loads in the A register the XROM code, resets the pointer to 6 and starts the search for the requested ROM.

Thereafter, it increments C at the pointer (digit 6 of C). If C exceeds F (hexadecimal) the carry flag is clear and the following command is executed: if F is exceeded without having found the requested ROM, that means the ROM is not resident in memory, and we go to Ø2EØ which causes "NONEXISTENT" to be displayed.

If the value at the pointer is less than F, FETCH into the S&X field of the C register the first tested word of the ROM. This word, remember, is the XROM number. Compare it to the code put in reserve in the A register. If it is different, take the following word as being the number of functions it tested. Accumulate the number of these functions in B, then test the following ROM.

If it is the right ROM, jump to A526 and initiate the CAT 2 by loading into status register P the number of skipped functions and the catalog number (2), then branch to standard catalog 2 routine starting at the first function of the requested ROM (in general its name).

Since you are in the standard CAT 2, you have all the advantages of it: go forward or backward, step by step, slowing down the flow by pressing one key or faster on the HP-41CX...all things you could do otherwise.

7.8 A Proof of Microcode Power: Charge (By Stephane Barizien)

This program was created in France by a young member of PPC-T, Stephane Barizien, who quickly became a microcode virtuoso. It illustrates clearly what I have tried to demonstrate with REP, that is the efficiency that one can get from the proper use of the internal modules of the HP-41. This function is not programmable, and thus can be executed even if the machine is in program mode. It prompts Charge___ and waits for a decimal number of three digits. When given that number, it introduces into program memory the corresponding byte. The only flaw of this simple program is that it does not show the line numbers, and it places the byte on top of the displayed line. This feature could have been corrected at the expense of lengthening the program. The operating mode is as follows:

Assign to a key the function CHARGE.
Switch to program mode.
Press ENTER↑.
Press the assigned key (in mode user)
CHARGE___.

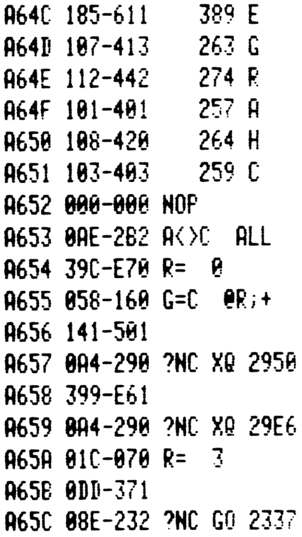
Fill the prompts with with decimal number of the byte.
Start all over as many times as there are codes to be furnished.
When finished, backarrow the ENTER↑.
In the procedure that follows, the HP-41 renumbers the lines.

Example: ENTER↑, Charge 242, Charge Ø79, backarrow, place "DØ" in memory. This program is the microcode equivalent of LB. The program's length, excluding the name, is 11 words! What could better illustrate the power of microcode programs that use the internal operating system routines.

In the program, the NOP indicates a null (no operation) function. The left bits of the "C" at the beginning of the name cause the triple underline prompt that gets the decimal input from the keyboard. This number is transformed into binary by the operating system and stored in A. The program extracts the decimal input from A and moves it into C (and then G). Since G only has 8 bits, the number is taken module 256. For example, if you do CHARGE 3ØØ, the result is CHARGE 44. The number is taken up again in C and stored in G, which, on the passage, limits it to 8 bits and thus at 255. If you answer 3ØØ the program takes 3ØØ-255.

Next CHARGE recalls the program pointer from status register b, using routine 295Ø, "GETPC".

The Routine 29E6 ("INBYT") inserts the byte contained in the G Register into the program after having incremented the PC. Routine 2337 ("PUTPC") reinstates the program counter in Status Register b, it requires R=3. Very simple.



```
A64C 185-611    389 E
A64D 107-413    263 G
A64E 112-442    274 R
A64F 101-401    257 A
A650 108-420    264 H
A651 103-403    259 C
A652 000-000 NOP
A653 0AE-2B2 A<>C  ALL
A654 39C-E70 R=  0
A655 058-160 G=C  @R,+
A656 141-501
A657 0A4-290 ?NC XQ 2950
A658 399-E61
A659 0A4-290 ?NC XQ 29E6
A65A 01C-070 R=  3
A65B 0DD-371
A65C 08E-232 ?NC GO 2337
```

APPENDIX I

NUMBER SYSTEMS

APPENDIX I
NUMBER SYSTEMS

We know by looking whether a switch is open or closed (off or on). A binary digit is the same: Ø or 1.

But we are never satisfied, we have the need to do some mathematics. To represent a switch, we need to designate two states, and thus count in base 2, in binary. We thus count:

Ø	Zero
1	One
1Ø	Two
11	Three
Etc.	

We call each of these numbers Ø or 1 a bit (binary digit).

But we are used to counting in decimal (base 1Ø), thus we have some conversions to do. Alas it comes out that the translation from base 2 to base 1Ø does not work out simply. There is no correspondence between the digits obtained in Base 1Ø and the bits in base 2, but only correspondence between the entire numbers. For example:

> 1 in base 2 or base 1Ø is written the same.
> 2 in base 1Ø gives 1Ø in base 2 (!!!).
> 12 in base 1Ø does not give 1 followed by 1Ø in base 2, but 11ØØ. The numbers must be translated in blocks, and this is not easy.

It is easy to translate into binary numbers in any base that is a power of 2:

2↑1 = 2 No problem
2↑2 = 4 Too close to 2, is not used
2↑3 = 8 Octal base, often used (OCT and DEC of the 41C)
2↑4 = 16 Hexadecimal base that we will constantly use here. A hexadecimal number is a number translated exactly by 4 bits.

4 = Ø1ØØ
6 = Ø11Ø
46 = Ø1ØØ Ø11Ø without any problems.

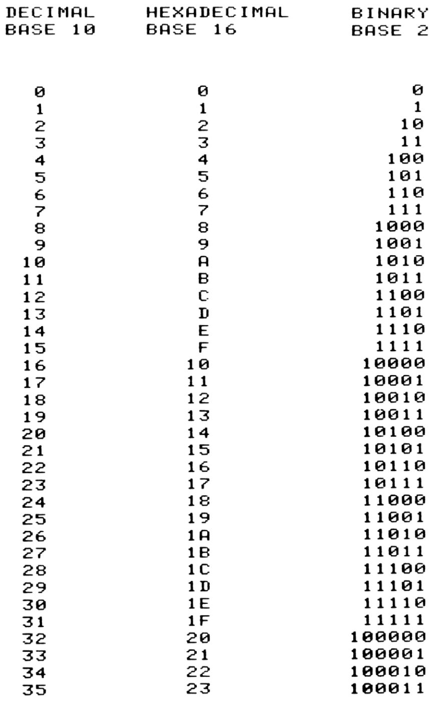
For the numbers higher than 9 hex we use the alphabet letters from A to F. The bottom line of the byte table gives the correspondence between hexadecimal and binary.

We will call a nibble a single hexadecimal digit. A byte (in reference to the binary) is a group of two hexadecimal digits.

We will mix the decimal and the other bases by using the decimal to number the elements.

We will say, "There are 162 registers," but, "the register A2." This is hard, but you will get used to it. If you have money to purchase yourself a HP-16C, it can do the work for you.

The table in Figure 38 will help somewhat.



DECIMAL BASE 10	HEXADECIMAL BASE 16	BINARY BASE 2
0	0	0
1	1	1
2	2	10
3	3	11
4	4	100
5	5	101
6	6	110
7	7	111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111
16	10	10000
17	11	10001
18	12	10010
19	13	10011
20	14	10100
21	15	10101
22	16	10110
23	17	10111
24	18	11000
25	19	11001
26	1A	11010
27	1B	11011
28	1C	11100
29	1D	11101
30	1E	11110
31	1F	11111
32	20	100000
33	21	100001
34	22	100010
35	23	100011

Decimal/Hex/Binary Number Systems

FIGURE 38

APPENDIX II

DEFINITIONS

APPENDIX II
DEFINITIONS

BIT — Binary Digit - An element of a binary number; can be either Ø or 1.

BYTE — Consisting of eight bits or two digits.

DIGIT — A grouping of four (4) bits to represent Ø to F Hex. Digits are often referred to as nibbles.

E — Exponent - Part of a digit string; e.g., 2E6 = 2,ØØØ,ØØØ.

EPROM — Erasable Programmable Read Only Memory - Sometimes also referred to as Electrically Programmable Read Only Memory. EPROMs maintain the programmed memories without a continuous power supply. When used with an EPROM reader, they appear to the HP-41 to be a standard Hewlett-Packard application module.

Generally EPROM readers require the use of two EPROMS to simulate an applications module.

K — Kilo - In general, 2 to the 1Øth power or 1Ø24 elements. A standard 4K Hewlett-Packard applications module is 4Ø96 (4 x 1Ø24) bytes.

MICROCODE — A term coined to describe the lowest level programming language of the HP-41. This language is at times called "M-Code" for short or "Assembly Language." Microcode, in the true sense of the word, refers to the "hard wired" internal language of a computer microprocessor.

RAM — Random Access Memory - A semiconductor memory which can be erased and reused. The RAM memory of the HP-41 consists of the status registers and all registers which may contain programs and/or data. The CPU of the HP-41 also contains RAM memory which is used by all microcode functions. The RAM of the HP-41 requires continuous power to maintain its contents, although this is only a few microamperes.

REGISTER — The HP-41's registers consist of 56 bits, therefore seven bytes. A register of the HP-41 may contain program functions or data (alpha or numerical). The largest or smallest decimal number that an HP register can contain is 1Ø raised to the +99th power.

ROM — Read Only Memory - A semiconductor memory which cannot be erased. All application modules and the HP-41's internal operating system are "ROMs."

APPENDIX III

USER CLUBS AND PUBLICATIONS

APPENDIX III
USER CLUBS AND PUBLICATIONS

International:

PPC
P.O. Box 9599
Fountain Valley, California   92728-9599
U.S.A.

Publications: PPC Journal

U.S.A.

CHHU
2545 West Camden Place
Santa Ana, California   92704

Publication: CHHU Chronicle

France:

PPC-Toulouse
77 Rue du Cagire
31100 Toulouse
France

Publication: Micro Revue

PPC-Paris
56 Rue J.J. Rousseau
F-75001 Paris
France

Publication: PPC-PC

Australia:

PPC Melbourne
John McGechie
P.O. Box 512
Ringwood Victoria
3134 Australia

Publication: PPC-Technical Notes

United Kingdom:

PPC-UK
Astage
Rectory Lane, GB
Windlesham Surrey
GU20 GBW
England

Publication: Datafile

Germany:

CCD
Computerclub Deutchland
Postfach 2129
D-6242 Kronberg 2
West Germany

Publication: PRISMA

Austria:

CCA
Computerclub Austria
P.O. Box 50
A-1111 Wien, Austria

Publication: CCA Journal

Denmark:

PPC-Danmark
Postboks 2
DK-3500 Yaerloese
Denmark

Publication: USER

APPENDIX IV

FURTHER READING AND REFERENCE

APPENDIX IV
FURTHER READING AND REFERENCE

"Synthetic Programming on the HP41C," by William C. Wickes.

Larken Publications
4516 N.W. Queens Avenue
Corvallis, Oregon 97330
U.S.A.

"HP-41 Synthetic Programming Made Easy," by Keith Jarett.

Synthetix
P.O. Box 1080
Berkeley, California 94701-1080
U.S.A.

"HP-41 Extended Functions Made Easy," by Keith Jarett.

Synthetix
P.O. Box 1080
Berkeley, California 94701-1080
U.S.A.

"The PPC ROM Users Manual," by the Members of PPC

PPC
P.O. Box 9599
Fountain Valley, California 92728-9599
U.S.A.

"The Zenrom-3B Users Manual," by Zengrange, Ltd.

Zengrange, Ltd.
Greenfield Road, GB
Leeds
LS9 8DB
England

"The HP41 Synthetic Quick Reference Guide," by Jeremy Smith.

Codesmith
2056 Maple Avenue
Costa Mesa, California   92627
U.S.A.

"Exploring Extended Functions on the HP-41," by Frank Wales.

PPC-UK
Astage
Recotry Lane
Windlesham, Surrey
GU20 6BW
England

"Calculator Tips and Routines," by John S. Dearing.

Corvallis Software Inc.
P.O. Box 1412
Corvallis, Oregon   97339-1412
U.S.A.

"HP-41/HP-IL System Dictionary," by
Cary E. Reinstein.

PPC
P.O. Box 9599
Fountain Valley, California 92728-9599
U.S.A.

"The Protosystem Users Manual," by
Nelson F. Crowle.

Prototech, Inc.
P.O. Box 12104
Boulder, Colorado 80203
U.S.A.

"The Synthetic Quick Reference Card," by
Keith Jarett.

Synthetix
P.O. Box 1080
Berkeley, California 94701-1080
U.S.A.

"Curve Fitting on the HP-41," by William M. Kolb.

IMTEC
P.O. Box 1402
Bowie, Maryland 20716
U.S.A.

"HP-41 VASM Listings," by Employees of Hewlett-Packard. "NOMAS" Publications.

PPC
P.O. Box 9599
Fountain Valley, California 92728-9599
U.S.A.

"HP-41 IC Specifications," by Employees of Hewlett-Packard. "NOMAS" Publication.

PPC
P.O. Box 9599
Fountain Valley, California 92728-9599
U.S.A.

Many other HP-41 books are available from various other sources. One of the most complete selections of HP-41 books and supplies can be found in the EduCalc catalog available from:

EduCalc Mail Store
27953 Cabot Road
Laguna Niguel, California 92677
U.S.A.

APPENDIX V

MICROCODE STORAGE AND DEVELOPMENT EQUIPMENT

APPENDIX V
MICROCODE STORAGE AND DEVELOPMENT EQUIPMENT

"ProtoCODER" and "ProtoEPROM"
Prototech, Inc.
Nelson F. Crowle, President
P.O. Box 12104
Boulder, Colorado 80303

"HHP EPROM Reader"
F. M. Weaver Associates
6201 Fair Valley Drive
Charlotte, North Carolina 28211
U.S.A.

"ERAMCO MLDL"
ERAMCO Systems
Valentynkade 27-11
NL-1094 SR Amsterdam
The Netherlands

"HP-41 EPROM ROM Simulator"
Dallas Development Systems
7410 Stillwater
Garland, Texas 75042
U.S.A.

"Redshift EPROM Programming"
Wilson W. Holes
7614 Lakecliff Way
Parker, Colorado 80134

APPENDIX VI

PROGRAM ASSEMBLER

APPENDIX VI
PROGRAM ASSEMBLER

Included herein is a program "ASM" that will allow you to find the 244 code from the commands (Figure 39).

This program allows you to write in microcode without having to worry about the 244 codes. The program calculates the codes.

The execution of the program does not require the entry of mnemonics, but instead the order number of the function as shown in the first column of the table, which is the command or parameter in decimal.

The particular tables (6/Ø, 8/Ø, C/Ø) giving the code directly are disregarded by the assembler.

For commands of type Ø (TØ) and type 2 (T2), enter the number of the command. ENTER, the field number, XEQ A for TØ or XEQ C for T2, you will see the hex code in the display.

For the commands of Type 1, load alpha with the address of the function and do XEQ D for ?NCXQ, XEQ E for ?CXQ, XEQ F for "NC GO and XEQ G for "C GO.

For Type 3 commands, load X with the decimal value of xxx of the jump and XEQ H for JNC or XEQ I for JC.

The installation of a keyboard overlay as indicated in Figure 39 will allow the assignments by default of the upper keys of the keyboard for quick and easy use.

This program uses two routines of PPC ROM: QR (quotient and remainder) giving MOD and quotient of a division and BD (base to decimal) to translate the address of the jumps.

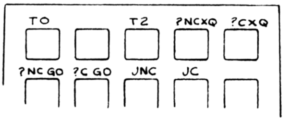
For those who do not have the PPC ROM, I have reproduced these programs (QR and BD). BD uses R06 to store the base and a required Size 007. Barcodes are included in Appendix XVI.

Line 12 is (Hex) F3 F7 ØØ Ø8 (synthetic). You must be able to do everything!

```
01♦LBL "ASM"
02♦LBL I
03 SF 00
04♦LBL H
05 CLA
06 X<0?
07 SF 01
08 ABS
09 63
10 X<Y?
11 GTO 19
12 RDN
13 128
14 X<>Y
15 FS?C 01
16 -
17 32
18 XROM "QR"
19 XEQ IND Y
20 2
21 XROM "QR"
22 XEQ IND Y
23 2
24 *
25 1
26 X<>Y
27 FS?C 00
28 +
29 4
30 *
31 3
32 GTO 20
33♦LBL D
34 XEQ 17
35 .
36 GTO 20
37♦LBL E
38 XEQ 17
39 1
40 GTO 20
41♦LBL F
42 XEQ 17
43 2
44 GTO 20
45♦LBL G
46 XEQ 17
47 3
48 GTO 20
49♦LBL 17
50 16
51 STO 06
52 XROM "BD"
53 256
54 XROM "QR"
55 XEQ 18
56 1
57 +
58 XEQ IND X
59 "⊦/"
60 RDN
61 RDN
62 XEQ 18
63 RTN
64♦LBL C
65 CLA
66 X<>Y
67 8
68 XROM "QR"
69 XEQ IND Y
70 X<>Y
71 RDN
72 8
73 *
74 +
75 4
76 XROM "QR"
77 XEQ IND Y
78 4
79 *
80 2
81 GTO 20
82♦LBL A
83 CLA
84 XEQ 16
85 .
86♦LBL 20
87 +
88 XEQ IND X
89 AVIEW
90 STOP
91♦LBL 18
92 16
93 XROM "QR"
94 X<>Y
95♦LBL 16
96 4
97 XROM "QR"
98 XEQ IND Y
99 X<>Y
100 RDN
101 16
102 *
103 +
104 4
105 XROM "QR"
106 XEQ IND Y
107 4
108 *
109 RTN
110♦LBL 00
111 "⊦0"
112 RTN
113♦LBL 01
114 "⊦1"
115 RTN
116♦LBL 02
117 "⊦2"
118 RTN
119♦LBL 03
120 "⊦3"
121 RTN
122♦LBL 04
123 "⊦4"
124 RTN
125♦LBL 05
126 "⊦5"
127 RTN
128♦LBL 06
129 "⊦6"
130 RTN
131♦LBL 07
132 "⊦7"
133 RTN
134♦LBL 08
135 "⊦8"
136 RTN
137♦LBL 09
138 "⊦9"
139 RTN
140♦LBL 10
141 "⊦A"
142 RTN
143♦LBL 11
144 "⊦B"
145 RTN
146♦LBL 12
147 "⊦C"
148 RTN
149♦LBL 13
150 "⊦D"
151 RTN
152♦LBL 14
153 "⊦E"
154 RTN
155♦LBL 15
156 "⊦F"
157 END
```

```
01♦LBL "BD"
02♦LBL A
03 CLST
04♦LBL 01
05 "⊦ "
06 X<> ]
07 X=0?
08 GTO 01
09 X<> [
10 R↑
11 X<> \
12 "⊦♦ℓ"
13 X<> \
14 RDN
15 X<> [
16  E
17 *
18 39
19 -
20 X>0?
21 DSE X
22 9
23 +
24 X<0?
25 GTO 02
26 X<>Y
27 RCL 06
28 *
29 +
30 .
31 GTO 01
32♦LBL 02
33 RDN
34 CLA
35 RTN
```

```
01♦LBL "QR"
02 X<>Y
03 STO ]
04 X<>Y
05 MOD
06 ST- ]
07 LASTX
08 ST/ ]
09 CLX
10 X<> ]
11 X<>Y
12 RTN
13 END
```



ASM Listing

Program Assembler

SEE APPENDIX
XVI FOR BAR
CODES

FIGURE 39

APPENDIX VII

ADDRESSING ROM AND RAM, BY DIDIER JEHL

APPENDIX VII
ADDRESSING ROM AND RAM BY DIDIER JEHL

ROM Principle

1) The HP-41 sends an address (A15...AØ) on line

ISA

2) Each ROM module compares this address with the ones assigned to it:

   If there is no match, the module disconnects itself.

   If there is a match, it prepares the data (DØ to D9) corresponding to the specified address (A15...AØ) between phases 31 to 46, and sends them in series starting from the rising edge of the sync signal.

The two types of HP Modules:

1. Module with fixed address (MFA): This type of module will have an address which is totally independent of the port it is plugged in to; i.e., the time module will be addressed to page 5 regardless of the port (1, 2, 3 or 4) that it is plugged in to.

2. Module with variable address (MVA): This type of module will have an address determined by the port in which it is located; i.e., the Math ROM will

be addressed in either page 8, A, C or E depending on whether it is resident in port 1, 2, 3 or 4, respectively.

Internal Address:

T&C (Timing and Control) connects the ISA line at the write interface during phases 16 to 31. This allows for the comparison of the address bits (A15...A12) with R2, R3, R4 and R5. If there is a match between R2-R5 and A15-A12, one of the memory blocks is selected and delivers ten bits of data DØ-D9.

T&C detects the sync rising edge then connects the ISA line with the read interface and sends the data DØ-D9 until the sync falling edge.

RAM Principle

1. The HP-41 sends on the ISA line the command 27Ø (RAM SLCT) in order to warn the registers that it will select one of them.

2. At the beginning of the next cycle, the data line sends ten address bits. The register with the corresponding address is then selected and remains selected until another is selected.

3. To write to the selected register, the HP sends on the ISA line the 27Ø command and at the beginning of next cycle the 56 bits present on the data line is written in the selected register. These 56 bits are the ones contained in the C register of the microprocessor.

4. To read the selected register, the HP sends on the ISA line the command Ø38H (Read Ø/T), and at the beginning of the following cycle, the 56 bits of the selected register are sent on the data line, the C register of the microprocessor.

Internal Addressing:

For the simple modules B3 and B4 (which value depends only on the port) they are part of the addresses comparison.

For the quad memory, B3 and B4 = Ø (inverted).

A counter connected to the ISA line allows the microprocessor to detect the different commands 27Ø, 2FØ, Ø38. When the 27Ø command is detected, the T&C (Timing and Control) positions the buffer (in entry and commands) at the beginning of the following cycle, the latch circuit will latch as soon as 1Ø address bits are present on the data line. A comparer compare (!) the six most significant bits with the addresses assigned to each chip; the chip with the corresponding address is selected and in turn selects a register (amongst the 16 that she contains) by means of the four least significant bits.

T&C waits for the command 2FØ (or Ø38) to be detected in order to command the writing (or reading) of the selected register and send to (or from) the register (or on the DATA line) the 56 data bits from the cycle following the 2FØ (or Ø38) command.

APPENDIX VIII

ALARMS

APPENDIX VIII
ALARMS
(Module HP82182A)

Let's start the study from the bottom of the alarm portion of memory and move upward; that's the way the HP-41 works. Let's follow an example: Suppose we desire an alarm to activate on October 25, 1983 at 12 noon and display "AU FOND DE LA HP-41C." We introduce this text in Alpha, Ø, ENTER↑, 1Ø.251983, ENTER↑, 12, XEQ "XYZALM".

We have created a status register starting with the hexadecimal digits AA, followed by a byte giving the total numbers of alarm registers. The following digits are not used. In our example we have, AA Ø6 ØØ ØØ ØØ ØØ ØØ.

The register following (above) this register contains:

- In the 11 leftmost digits, the time between the alarm date and January 1, 19ØØ, expressed in binary as the number of tenths of seconds.

- In the following digit (number 2), the indication of the existence of a repeating alarm (if digit is 1).

- In the following digit (number 1), the indication of "past due" alarm. A value of F in digit 1 means that the alarm has past without having been acknowledged.

- The last digit (number Ø) indicates the number of registers occupied by the alphanumeric message (from Ø to 4).

- In our example we have:

  :2:6:4:4:9:2:Ø:Ø:Ø:Ø:Ø:Ø:Ø:3:

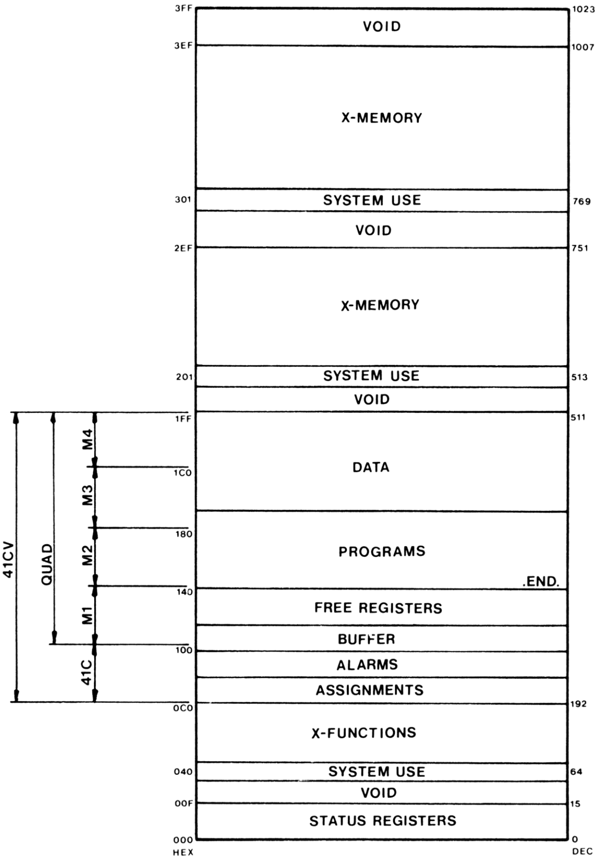
The message is located in the following registers, read from left to right and from bottom to top:

```
ØØ  41  55  2Ø  46  4E  4F
    A   U       F   O   N
44  2Ø  44  45  2Ø  4C  41
D       D   E       L   A
2Ø  48  5Ø  2D  34  31  43
    H   P   -   4   1   C
```

For repeating alarms, the repeating interval is stored between the register giving the alarm date and the message. The repeating interval in seconds occupies digits 5 to 11 of this register, which is not present in our example.

A new alarm is immediately located above the previous.

On top of the last alarm, there is a register starting with the code hex FØ. The rest of this upper register is empty: FØ ØØ ØØ....



HP-41C Memory Map

FIGURE 40

APPENDIX IX

COMMENTARY ON THE SCHEMATIC OF THE HP-41C

APPENDIX IX

Commentary on the Schematic of the HP-41C microprocessor.

TERM	SENSE
KC	Key Column: Number of the Column of the Keyboard
KR	Key Row: Number of the Row of the Keyboard
POR	Detection-On Key Used (put in or out of deep sleep)
X1, X2	Connection to the oscillating circuit
Ø1	Clock signal for reading data
Ø2	Clock signal for sending data and calculations
SYNC	Synchronization
PWO	Power on: microprocessor active
DPWO	(Display power on) Commands the Display
VC1, VCO, LLD	Command of different modes (deep sleep, active, light sleep)
ISA	Commands and ROM Addresses
FO	Flag output to the bender (beeper)
FI	Flag in: detection of a nonsynchronized peripheral
DATA	Transfer data, RAM addresses

APPENDIX X

THE DISPLAY OF THE HP-41C

APPENDIX X
THE DISPLAY OF THE HP-41C

These commands have been detailed by Paul Lind and published for the first time in the Australian PPC publication, PPC-TN.

Each character is sent to the display under the form of a chain of nine bits:

:8:7:6:5:4:3:2:1:Ø:

Bit number 8 when set to 1 indicates row 4 of the character table (Figure 1Ø) if the bits 5-Ø give a number from Ø to F. For all other values of the bits 5-Ø the display shows a space when bit 8 is 1. When bit 8 is zero, bits 5-Ø specify a character from rows Ø-3 of the character table. Bit 8 has no effect on the punctuation position of each display character.

Bits 6 and 7 indicate punctuation: no punctuation: ØØ, decimal point (.) Ø, colon (:) 1Ø, or the comma (,) 11.

The information can be sent to the display by 1, 4, 8 or 9 bits, character by character or in blocks. Here are the results of the commands transferred by 9 bits:

When a character is pushed into the display, a character is lost from the other side. A read causes the display to be scrolled.

Writ 14(d) Take a character from the C registers S&X field and push it onto the left side of the display.

Read 14(d) Read the character from the right side of the display into the C registers S&X field.

Writ 15(e) Take a character from the C registers S&X field and push it onto the right side of the display.

Read 15(e) Read a character from the left side of the display.

Writ 4(L) Push 4 characters from the C register onto the left side of the display.

Read 4(L) Read 4 characters from the left side of the display.

Writ 6(N) Push 6 characters onto the right of the display.

Read 6(N) Read 6 characters from the right of the display.

The following commands handle eight bits only. Only the considered elements are modified on the display.

Writ 12(b) Push a XP character from the C register to the left side of the display.

Read 12(b) Read the 8 lower bits of the left most character from the display.

Writ 13(c) Push the character to the right.

Read 13(c)   Read the character from the right.

Writ 3(x)   Push 6 characters to the left.

Read 5(M)   Push 6 characters to the left.

Writ 5(M)   Push 6 characters to the right.

Read 5(M)   Function not well known.

The following commands handle four bits only; high (bits 4 to 7) or low (bits Ø to 3), either 1 or 12 characters:

Write, push, and read

Command	High	Low	1	12	Left	Right
Writ 7(O)		x	x		x	
Read 7(O)		x	x			x
Writ 10(⊦)		x	x			x
Read 10(⊦)		x	x		x	
Writ 0(T)		x		x	x	
Read 0(T)		x		x		x
Writ 8(P)	x		x		x	
Read 8(p)	x		x			x
Writ 11(a)	x		x			x
Read 11(a)	x		x		x	
Writ 1(Z)	x		x			
Read 1(Z)	x					x

The following commands concern only the left bit (bit 8), and work analogously to the commands on 4 bits:

Writ 9(Q)	Like Writ 7(O) (but for Bit 8)
Read 9(Q)	Like Read 7(O)
Writ 2(Y)	Like Writ Ø(T)
Read 2(Y)	Like Read Ø(T)

The two read commands have an amusing result.

The Annunciators:
The Effect of Write Data, Read 5(M) in C S&X

BIT	ANNUNCIATOR
Ø	ALPHA
1	PRGM
2	Flag 4
3	Flag 3
4	Flag 2
5	Flag 1
6	Flag Ø
7	SHIFT
8	RAD
9	G (GRAD)
1Ø	USER
11	BAT

APPENDIX XI

THE HP-41C OPERATING SYSTEM, BY BILL REGGUSY

APPENDIX XI
Commentary on the HP-41C Operating System

Applications of Unusual Instructions:

1FØh GO->XQ

Complementing XQ->GO, this function allows the last-execution GOTO to be treated as an XEQ instead. I have found this instruction very useful where the size of the subroutine return stack has been too small for my needs.

```
1D4h ENF14
1DCh DISF14
1C4h CF 14
1C8h SF 14
1CCH ?FS 14
```

It appears that Jake Schwartz was correct in his assumption that there were more than fourteen CPU flags available to the M-coder. The reason this fifteenth flag has only just been discovered is that you need to enable the flag (ENF14) before it can be used. You may well ask yourself why this flag is "protected" in such a manner -- HP reveals all! This is a flag you want to be extremely careful with; it's the carry flag! Whenever the CPU wakes up, this flag is disabled, but once enabled, remains so until explicitly disabled (DISF14), or the CPU stops running.

```
118h ?OFF
Ø18h POWON
218h ?WKUP
```

POWON complements POWOF and starts the processor running, while ?OFF is a simple conditional test which sets the condition bit if the HP-41 is off (i.e., Deep Sleep). ?WKUP sets the condition bit if the 41 has been asked to wake up. These three instructions are used in the CPU Master Control Program (MCP) -- see below.

Ø3Øh KEY=C

This instruction causes the digits C(4:3) to be placed into the keycode register by a rather convoluted method -- the key corresponding to this keycode is pulled for one instruction cycle (156 microseconds), which then causes the keycode to be read in. Because the key is pulled for one instruction cycle, the instruction must be followed by a NOP (ØØØh), as with POWOFF, which has the effect of disabling the SYNC signal for one cycle. If not followed by a NOP, the key may stay down, and in extreme circumstances may even null itself.

I would like to thank my friends at HP for the information they provided -- this can only confirm HP's new policy of "open machines."

I have listed below the Master Control Program which the HP-41 CPU runs while in Deep Sleep Mode. This code is resident in the CPU, not in the internal ROMs. Again, this is courtesy of HP.

HP Master Control Program

```
ØØ 218 ?WKUP        is the 41 being asked to
                    wake up?
Ø1 3FB JNC -1       no, so try again
Ø2 2EC ?PF 13       yes, then is it a periph-
                    eral?
Ø3 Ø33 JNC +6       no, keyboard wake-up
Ø4 13Ø LDI          yes, ON key must be
                    pulled to effect wake-up
```

```
Ø5 Ø18 CON Ø18        load keycode for ON key
                      into C(1:Ø)
Ø6 1BC RCR 11         place it into C(4:3)
Ø7 Ø3Ø KEY=C          pull the key
Ø8 ØØØ NOP            ensure the key doesn't
                      stick
Ø9 1D4 ENF14          enable carry flag access
1Ø 1C4 CF 14          get it to a known state
11 118 ?OFF           is it from Deep Sleep?
12 Ø13 JNC +2         no, so leave carry clear
13 1C8 SF 14          yes, set carry
14 Ø18 POWON          switch us on
15 ØF3 checksum       Used by Service ROM to
                      verify CPU OK)
```

The above routine explains why the carry flag is set on Deep Sleep wake-up, but not Light Sleep wake-up.

APPENDIX XII

EPROM STRUCTURES, BY JIM DE ARRAS

APPENDIX XII
EPROM STRUCTURES
By Jim DeArras

An EPROM, ready for use in an EPROM reader (whatever model it might be) consists of two EPROMs at least, named U2 (for upper two) and L8 (for lower eight). L8 contains simply the 8 least significant bits (the rightmost 8) of the word. U2 contains thus the 2 most significant bits (the leftmost 2). These two bits, considered as a pair and read in order of increasing addresses, are lined up in U2 from right to left and from top to bottom. Example:

Words	Bits from Right
Øxx	ØØ
Øxx	ØØ
3xx	11
Øxx	ØØ

are lined up: ØØ 11 ØØ ØØ to form a byte of U2, here the byte 3Ø.

In the same manner:

ØØØ3 gives 11 ØØ ØØ ØØ = CØ
Ø3ØØ gives ØØ ØØ 11 ØØ = ØC
3ØØØ gives ØØ ØØ ØØ 11 = Ø3

Address ØØØ of U2 corresponds to the address ØØØ, ØØ1, ØØ2, and ØØ3 of L8. We thus obtain the address of the preceding byte reconstituted in U2 by dividing the address of L8 by 4 and using the integer part. Watch out, you must work in hexadecimal. If you have a HP-16C (lucky duck!), no problem, otherwise, use a base changing program. BD and TB of the PPC ROM will do very well.

In an EPROM set of 4K, with a 2732 (4K bytes) and a 2716 (2K bytes) only the first or the second half (your choice) of the 2716 is used.

I know that all this is not simple, but if you really need it, you will certainly have a "Club" EPROM set or EPROM set furnished by the fabricator of your EPROM reader which will allow you to experiment. You will see that it is fascinating.

APPENDIX XIII

REGISTER SELECTION BY RAM SELECT

APPENDIX XIII
Register Selection by RAM SELECT

RAM Select (RAMSLCT) selects the 16 register block of the given number. The address is thus taken modulo 16: 3=Ø 18=16,... For example, 3 RAM Select selects the status registers, but Ø(T) is always the register Ø, not the register 3! You will find in the code of the internal modules of the HP-41C, starting at the address Ø3F5, the following instructions:

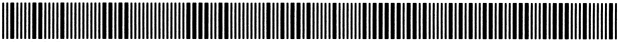
```
LDI
2FD
RAMSLCT
PRPHSLCT
READ(Y)
?NCGO Ø2Ø5 (MEMCHEK)
```

These codes don't have much meaning; it is a modification which occurred during development of the machine to remedy the synchronization problem between the conducting circuits of the display. But HP uses a trick to save space, using the same instruction to select the display and to deselect RAM memory. One must know, in fact, that during a read or write operation, the 41C microprocessor always sends the C contents to RAM memories, even if another peripheral has been selected. If when using the display, you do not watch that carefully, every message destined to the display sends the C contents to the previous selected memory. The problem is solved by selecting a portion of empty memory. But by doing so, 16 register places are rendered useless. Usually, the registers Ø1Ø to Ø1F are the ones playing this role, but the program example above makes the registers 2FØ to 2FF play the same role. I don't know if it is the only reason, but when HP built the X-memory modules, it was forced to leave those 16 register addresses empty! All that to save two steps, LDI Ø1Ø before RAMSLCT and LDI 2FØ before PRPHSLCT.

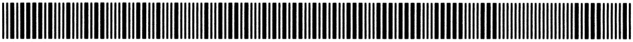
APPENDIX XIV

BAR CODE FOR PROGRAM "KA"

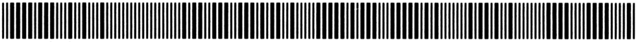
ROW 1 (1 : 3)



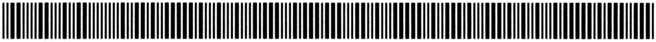
ROW 2 (3 : 7)



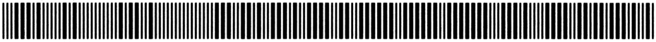
ROW 3 (7 : 11)



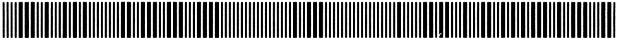
ROW 4 (12 : 19)



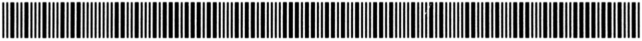
ROW 5 (20 : 26)



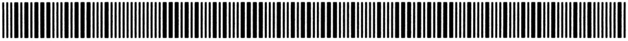
ROW 6 (27 : 32)



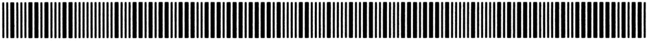
ROW 7 (33 : 42)



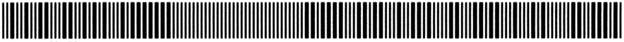
ROW 8 (42 : 52)



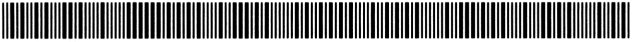
ROW 9 (53 : 64)



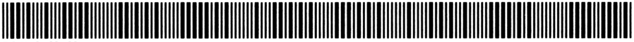
ROW 10 (64 : 70)



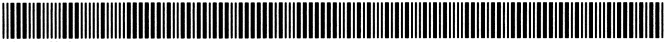
ROW 11 (70 : 76)



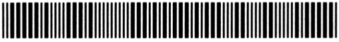
ROW 12 (77 : 83)



ROW 13 (83 : 88)



ROW 14 (89 : 91)



KA

PROGRAM REGISTERS NEEDED: 25

APPENDIX XV

BAR CODE FOR PROGRAM "LB"

ROW 1 (1 : 6)

ROW 2 (7 : 13)

ROW 3 (13 : 18)

ROW 4 (19 : 25)

ROW 5 (25 : 36)

ROW 6 (37 : 43)

ROW 7 (43 : 48)

ROW 8 (48 : 54)

ROW 9 (54 : 64)

ROW 10 (65 : 72)

ROW 11 (73 : 80)

ROW 12 (81 : 88)

ROW 13 (89 : 95)

ROW 14 (95 : 100)

ROW 15 (101 : 111)

ROW 16 (111 : 122)

ROW 17 (123 : 130)

ROW 18 (131 : 131)

LB

PROGRAM REGISTERS NEEDED: 32

APPENDIX XVI

Bar Codes for Programs "ASM", "BD" and "QR"

ROW 1 (1 : 4)

ROW 2 (5 : 13)

ROW 3 (13 : 20)

ROW 4 (21 : 30)

ROW 5 (31 : 36)

ROW 6 (37 : 42)

ROW 7 (42 : 48)

ROW 8 (48 : 54)

ROW 9 (55 : 62)

ROW 10 (62 : 70)

ROW 11 (71 : 81)

ROW 12 (81 : 88)

ROW 13 (88 : 96)

ROW 14 (97 : 105)

ROW 15 (106 : 114)

ROW 16 (114 : 122)

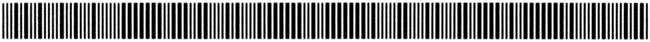
ROW 17 (123 : 129)

ROW 18 (130 : 138)

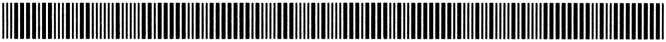
ASM

PROGRAM REGISTERS NEEDED: 39

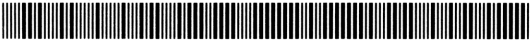
ROW 19 (138 : 145)



ROW 20 (146 : 153)



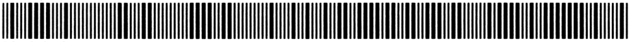
ROW 21 (153 : 157)



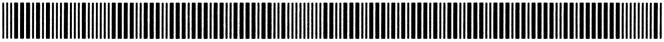
ASM

PROGRAM REGISTERS NEEDED: 39

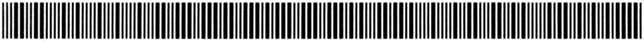
ROW 1 (1 : 5)



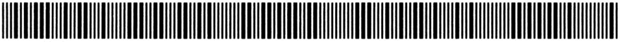
ROW 2 (6 : 12)



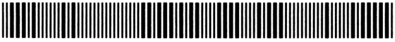
ROW 3 (12 : 21)



ROW 4 (21 : 31)



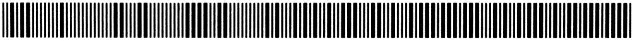
ROW 5 (32 : 36)



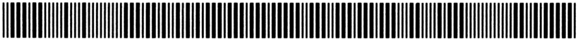
BD

PROGRAM REGISTERS NEEDED: 9

ROW 1 (1 : 6)



ROW 2 (7 : 13)



QR

PROGRAM REGISTERS NEEDED: 4

INDEX OF FIGURES

INDEX OF FIGURES

List of PPC members whose cooperation was important for this book:

Richard J. Nelson (1)
Gary Reinstein (2046)
John Dearing (2791)
John McGechie (3324)
Bill Wickes (3735)
Jim DeArras (4706)
Valentin Albillo (4747)
Steven Jacobs (5358)
Paul Lind (6157)
Didier Jehl (8116)
Lionel Ancelet (Paris Chapter)
Lynn Wilkins (?)
Stephane Barisien (?)

The Author:
J. D. Dodin (7726)

The Publisher:
K. Jarett (4360)

The Editor:
W. W. Holes (9026)

ORDER BLANK

	Quantity	Amount
HP—41 Extended Functions Made Easy By Keith Jarett $16.95 per copy	______	$______________
HP—41 Synthetic Programming Made Easy By Keith Jarett $16.95 per copy (QRC included)	______	$______________
Quick Reference Card (QRC) $3.00 each	______	$______________
ENTER (Reverse Polish Notation Made Easy) By Jean-Daniel Dodin $12.95 per copy	______	$______________
Inside the HP—41 By Jean-Daniel Dodin $18.95 per copy	______	$______________
HP—71 Basic Made Easy By Joseph Horn $18.95 per copy	______	$______________
Sales Tax (California orders only, 6 or 6.5%)		$______________
Shipping, per book		
within USA, book rate (4th class)	$1.50	
USA 48 states, United Parcel Service	$2.50	
USA, Canada, air mail	$3.00	
elsewhere, air mail	$6.05	
Shipping for QRC plastic cards (any number)		
Free with book order or stamped envelope		
Otherwise	$1.00	
Enter shipping total here		$______________
Total Enclosed		$______________

Checks must be payable through a U.S. bank

Name____

Address______________________________________

City_____________________State__________ZIP_______

Country______________________________________

Mail to:
SYNTHETIX
P.O. Box 1080
Berkeley, CA
94701-1080 U.S.A.
(415) 339-0601

ORDER BLANK

	Quantity	Amount
HP—41 Extended Functions Made Easy By Keith Jarett $16.95 per copy	______	$______________
HP—41 Synthetic Programming Made Easy By Keith Jarett $16.95 per copy (QRC included)	______	$______________
Quick Reference Card (QRC) $3.00 each	______	$______________
ENTER (Reverse Polish Notation Made Easy) By Jean-Daniel Dodin $12.95 per copy	______	$______________
Inside the HP—41 By Jean-Daniel Dodin $18.95 per copy	______	$______________
HP—71 Basic Made Easy By Joseph Horn $18.95 per copy	______	$______________
Sales Tax (California orders only, 6 or 6.5%)		$______________
Shipping, per book		
within USA, book rate (4th class)	$1.50	
USA 48 states, United Parcel Service	$2.50	
USA, Canada, air mail	$3.00	
elsewhere, air mail	$6.05	
Shipping for QRC plastic cards (any number)		
Free with book order or stamped envelope		
Otherwise	$1.00	
Enter shipping total here		$______________
Total Enclosed		$______________

Checks must be payable through a U.S. bank

Name____

Address______________________________________

City____________________State__________ZIP_______

Country______________________________________

Mail to:
SYNTHETIX
P.O. Box 1080
Berkeley, CA
94701-1080 U.S.A.
(415) 339-0601

Jean-Daniel Dodin

INSIDE THE HP-41C

UNLOCK THE SECRETS OF YOUR HP-41

Discover your calculator's internal workings. Synthetic programming, the status registers, the byte grabber, microcode - all are described here.

See how the byte grabber ''steals'' bytes to create *synthetic* instructions, an extension of the standard HP-41 program instruction set. Meet the *status registers* which hold fascinating secrets and danger for the unwary. Try several synthetic programs (bar code included). Learn about *microcode*, the language used by the machine's own microprocessor. Take a walk through the intricate built-in programming that makes the HP-41 work.

Your HP-41 has many hidden secrets. Isn't it time you started discovering them?

ISBN 0-9612174-4-8 $18.95