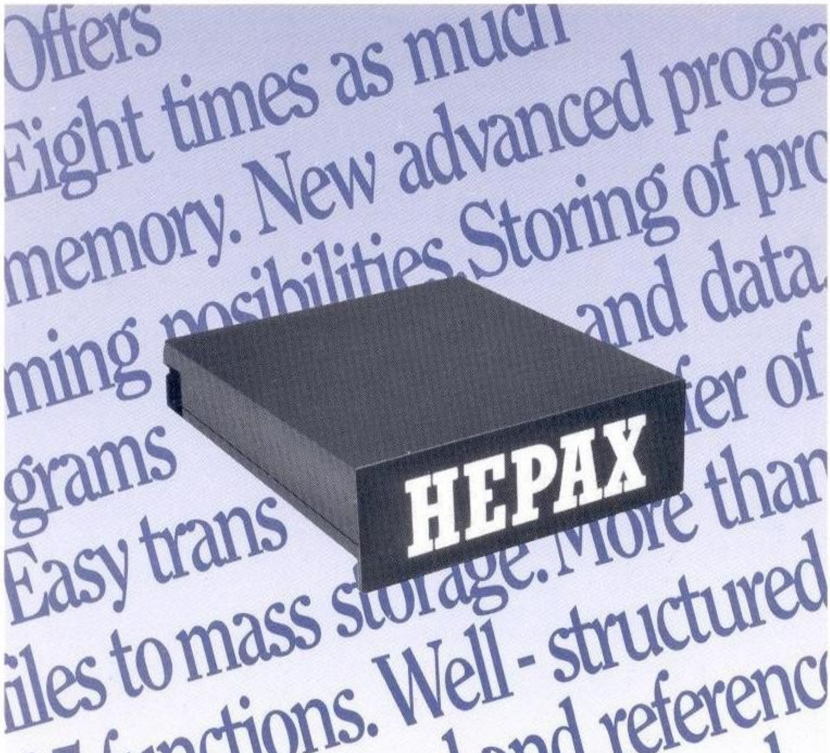
VM ELECTRONICS

HEPAX MODULE

Owner's Manual

Volume 2: M-code Programming


Offers
Eight times as much
memory. New advanced program-
ming posibilities. Storing of pro-
grams and data.
Easy transfer of
files to mass storage. More than
95 functions. Well - structured
manual and reference
card.
HEPAX

The XF multi-function subfunctions

Name	Number	Function
ALENG	000	Return length of string in ALPHA.
ANUM	001	Convert string in ALPHA to numerical value in X.
AROT	002	Rotate contents of ALPHA.
ATOX	003	Convert character in ALPHA to character code in X.
CLKEYS	004	Clear all key assignments.
CLRGX	005	Clear registers as specified by X.
GETKEY	006	Get keycode depending on key pressed.
GETKEYX	007	Get keycode within time specified by X.
PASN	006	Programmable assignment.
PCLPS	009	Programmable clear programs.
POSA	010	Find position of string or character in ALPHA.
PSIZE	011	Programmable SIZE.
RCLFLAG	012	Recall the status of user flags 00-43.
REGMOVE	013	Move a block of main memory data registers.
REGSWAP	011	Swap two blocks of main memory data registers.
ΣREG?	015	Return the location of the statistical registers.
SIZE?	016	Return the current SIZE.
STOFLAG	017	Restore the status of user flags 00-43.
x<>F	018	Exchange status of user flags 0-7 with X.
XTOA	019	Convert character code in X to character in ALPHA.
X=NN?	020	Compare X with indirect Y.
X≠NN?	021	Compare X with indirect Y.
X<NN?	022	Compare X with indirect Y.
X< =NN?	023	Compare X with indirect Y.
X>NN?	024	Compare X with indirect Y.
X> =NN?	025	Compare X with indirect Y.

The HEPAX multi-function subfunctions

Name	Number	Function
AND	001	Logical X AND Y.
BCAT	002	Block catalog.
BCD-BIN	003	Converts number in X from BCD to binary.
BIN-BCD	004	Converts number in X from binary to BCD.
CTRAST	005	Set display contrast ("Halfnut" calculators only).
DELETE	006	Works like DELETE of the hexadecimal editor.
INSERT	007	Works like INSERT of the hexadecimal editor.
NOT	008	Complement of X.
OR	009	Logical X OR Y.
ROTYX	010	Rotates Y register X nybbles.
SHIFTYX	011	Shift Y register X bits.
XOBR	012	Logical X exclusive-or Y.
X+Y	013	Bitwise addition.
X-$	014	Converts X register to alpha string.
Y-X	015	Bitwise subtraction.

electronics

The HEPAX Module

Volume 2
M-code Programming

August 2010

Contents

List of figures

List of tables

Part III:

The inner secrets of the HP-41

Section 7:

HP-41 internal structure

The major parts of the HP-41 itself are the Central Processing Unit (the CPU, the "brain" of the calculator), the user memory (RAM memory), the system memory (ROM memory) and the keyboard and the display. The relation between these parts is shown below.

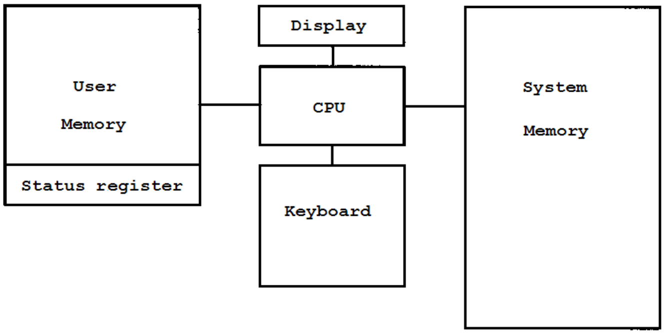


Fig. 10, Internal structure of the HP-41

Note that there are two different memory areas.

HP-41 memory

Unlike an "ordinary" computer, the memory of the HP-41 is divided into two distinct areas. These areas are known as the user memory area (main and extended memory) and the system memory area (used for operating system, plug-in modules, etc.).

User memory

The HP-41 user memory consists of the up to 319 registers of main memory and the up to 600 registers of extended memory. This is where your programs, data, key assignments, alarms, etc. are usually stored.

The user memory is RAM memory. This means that its contents may be changed by the user, and that the contents will be lost when power is removed from the calculator for an extended period of time.

User memory consists of registers, each register again divided into 7 bytes of 8 bits. Each user memory register has a unique address and each byte has a unique subaddress within the register.

To picture this, imagine a street of apartment buildings where the register address corresponds to the street no. and the byte subaddress corresponds to the floor.

Note that the CPU has no way of knowing if there is actually a memory chip at a given address. To find out, it tries to store some data and then reads the data back. If the CPU does read the same as it tried to write, there is actual memory at that address.

In our apartment building model, this corresponds to not knowing if anybody is home in a given apartment. To find out, we call up the apartment and give a message. We then ask to hear our message. If the message is repeated correctly, then there is someone home in that apartment.

There are 1024 register addresses, but some of these are used for housekeeping, stack, ALPHA register, etc. The user is left with 919 registers of main and extended memory. Since there are only 1024 addresses, there is no way of expanding user memory further.

The first 16 registers (the lowest addresses) hold special information about the status of the calculator. These registers are known as the status registers and are explained in detail later in this section.

System memory

The HP-41 system memory is normally used for operating system, peripherals like printer or card reader, and plug-in modules like the TIME module, the MATH modules etc.

The system memory is ROM memory. This means that its contents cannot be changed and that it will not be affected by power failure.

System memory consists of 10-bit words. There are 65536 addresses, divided into blocks of 4096 words. Remember that HEPAX memory is also divided in this way. There are 16 blocks, numbered 0 through F (hexadecimal).

Block	Addresses	Primary bank	Secondary bank
F	F000-FFFF	Port 4, upper	
E	E000-EFFF	Port 4, lower	
D	D000-DFFF	Port 3, upper	
C	C000-CFFF	Port 3, lower	
B	B000-BFFF	Port 2, upper	
A	A000-AFFF	Port 2, lower	
9	9000-9FFF	Port 1, upper	
8	8000-8FFF	Port 1, lower	
7	7000-7FFF	HP-IL module	
6	6F00-6FFF	Printer	IR printer
5	5000-5FFF	TIME	CX system
4	4000-4FFF	Take-over ROM	
3	3000-3FFF	Unused/CX	
2	2000-2FFF	System ROM 2	
1	1000-1FFF	System ROM 1	
0	0000-0FFF	System ROM 0	

Fig. 11, HP-41 system memory

The lower three blocks are always used for the operating system. Block 3 is used by the operating system of HP-41CX and is unused in HP-41C/CV.

Block 4 is used by a special type of ROMs known as "take-over ROMs". These special ROMs take over the control of the HP-41 from the operating system. Hewlett-Packard's DIAGNOSTIC ROM is of this type. Take-over ROMs will be discussed below.

Block 5 is used by the TIME module, block 6 for the printer (HP-82143A, HP-82161A and the IR printer module HP-82242A) and block 7 is used for the HP-IL module. Note that when you set the switch on the HP-IL module to "disable", the printer ROM is addressed to block 4.

The TIME module, printer and HP-IL modules will (if present) always answer to the addresses reserved for them. This means that although they may physically take up a port, they do not use the addresses reserved for the port. That's why they are called system addressed devices. HP-41C memory modules and Extended Memory modules are addressed to the user memory area, so they will not take up any space in the system area either. These modules are also known as system addressed devices.

Blocks 8 through F are used for plug-in modules and peripherals. Two blocks are reserved to each port, as shown in fig. 11 above. Normal modules - like the MATH module - only take up one block (usually the lower block). Some modules - like the plotter ROM - do, however, take up the full 8K space reserved.

The HEPAX BCAT (Block CATalog) function lists the contents of blocks 3/5 through F. Look through the block catalog to see what is addressed to each block of your HP-41 system.

User memory vs. system memory

The system, memory differs from user memory in one significant way: System memory will hold both "normal" programs and functions. "Normal" programs are known as FOCAL programs - Forty One CAlculator Language.

FOCAL programs in plug-in modules are identified by the "raised-T" symbol preceding their label in CATALOG 2 and appear just like FOCAL programs in main memory. They consist of normal program lines and can be viewed in PRGM mode. Since they are in ROM, they cannot be edited (you'll get the ROM message), but they can be copied to main memory using the built-in COPY function.

Functions are not preceded by the "raised-T" and cannot be listed. They are not written in ordinary PRGM-mode type instructions, but rather in HP- 41 M-code (see part IV: M-code programming).

Bank switching

To make the most of available system address space, Hewlett-Packard use bank switching. Bank switching means that there may be several "banks" of ROM at the same block address. Only one bank is enabled at any time.

The HP-41C/CV operating system does not use bank switching, but the HP-41CX has a second bank in block 5. The IR printer module residing in block 6 also has two banks. The Advantage module has a second bank in the upper block, and the ROM of the HEPAX module has four banks in the same block.

The method for bank switching is described in section 10: The M-code instructions.

The operating system

The HP-41 operating system tells the CPU how to read from the keyboard, how to access user memory, how to make calculations and how to output results on the display. The operating system is stored in ROM memory.

The operating system is actually a very long and complex program written in M-code. When you press a key on the keyboard, the HP-41 CPU "wakes up" and begins executing the operating system program.

Let us for a moment return to the take-over ROM's in block 4. When the CPU starts executing the operating system program, one of the very first things it does is to jump to the first address in block 4 (address 4000). If there is no module addressed to block 4, the CPU simply continues executing the operating system program. If there is a module addressed to port 4, the CPU will begin executing the program in this module, starting from address 4000.

Hewlett-Packard's DIAGNOSTIC ROM (used for diagnosing hardware errors) is of the take-over type. The printer ROM is written in such a way that it will immediately transfer control back to the operating system, even though it may be addressed to block 4 (HP-IL module set to "disable").

After checking for a take-over ROM, the operating system will then determine which key is down and perform the requested action (enter one digit into the X register, perform a calculation, store the key pressed as a program line, etc.)

We will consider one special case, namely that of running a FOCAL program. The HP-41 CPU doesn't understand FOCAL language - only M-code. It is the job of the operating system to read the FOCAL program one line at a time, interpret this line and then execute an appropriate M-code routine.

In this way, each FOCAL program line actually represents an M-code subroutine, typically consisting of hundreds of M-code instructions. Thus, a FOCAL program is written by stringing together references to M-code subroutines and when it is executed, the CPU actually executes the M-code subroutines specified by the program lines.

The HEPAX module

There are a few special things to note about the HEPAX module.

For one thing, the Advanced HEPAX and Double HEPAX Memory modules contain 16,000 bytes of HEPAX memory. Since this much memory cannot fit into the address space reserved for one port, it must use the address space of two ports. This is why these modules occupy the address space of a port and its neighbor, either port 1 and 2 (blocks 8 through B) or port 3 and 4 (blocks C through F).

The Advanced HEPAX and Double HEPAX memory modules only need the address space reserved for each port, they do not physically take up two ports. Therefore, modules and peripherals that do not address themselves to the port address space (system addressed devices) can be inserted next to Advanced and Double memory modules without problems.

All the functions in the HEPAX ROM occupy only one block. This is achieved by the use of bank switching between four banks. To make the HEPAX system as flexible as possible, the HEPAX ROM will scan the system address space each time the HP-41 is turned on. The HEPAX ROM will then automatically address itself to a vacant block. Therefore, the HEPAX ROM may be addressed to any block from 5 to F. This works all automatically, and you need not concern yourself with the location of the HEPAX ROM. If no free block exists in your HP-41 system, you will get the **ILL CONFIG** message.

The status registers

As mentioned above, the first 16 registers of user memory has a special significance. These registers are known as the status registers. Note that some information does not take up whole bytes, but rather a number of half bytes. A half byte is known as a nybble - 4 bits to 1 nybble, 2 nybbles to a byte.

This manual will only give a brief overview of the way the HP-41 uses the status registers. For a more detailed description, refer to William C. Wickes' "Synthetic programming on the HP-41" or another book on the subject of synthetic programming.

The structure of the status registers is shown in figure 12 below.

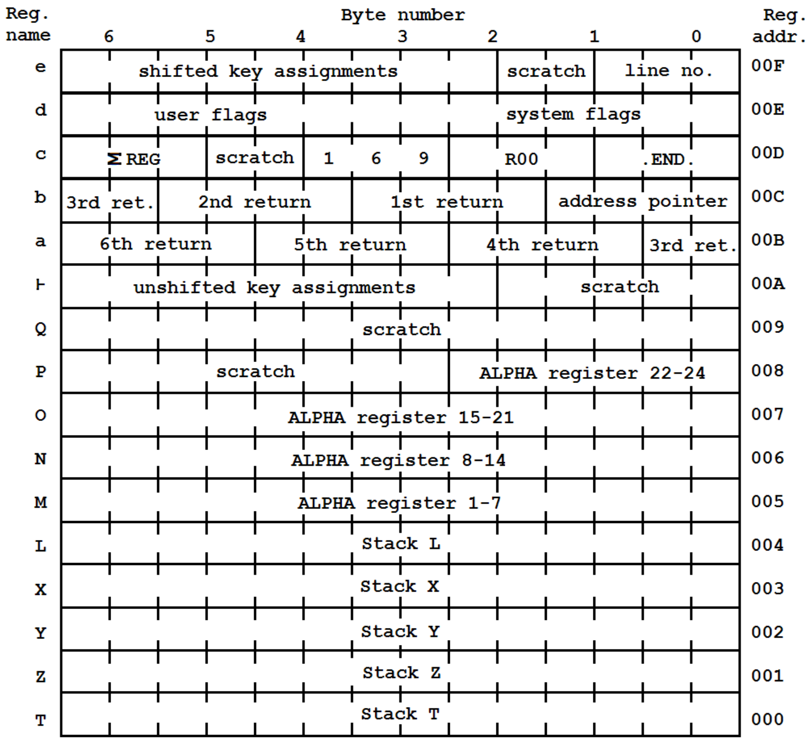


Fig. 12, The structure of status registers

The stack registers

The X, Y, Z, T and L registers are the stack registers. They may contain a number or an ALPHA string of up to 6 characters.

Numbers are always stored in scientific notation, i.e. as a 10-digit signed mantissa and a 2-digit signed exponent, as shown below. The display format (FIX, SCI or ENG) affects the displaying of numbers only.

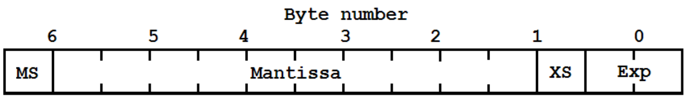


Fig. 13, Number register format

The MS and the XS nybble represent the sign of the number and the sign of the exponent, respectively. The HP-41 uses 0000b for "plus" and 1001b for "minus". Any other value will normally display as a minus.

The mantissa is written with one nybble for each digit. The most significant digit (MSD) is written to the left, i.e. in the lower half of the sixth byte. The exponent is also written with the MSD to the left. If the exponent is negative, it is written as "100 minus exponent".

For example, the number 1.2345 E-78 would be stored as 01234500000922 hexadecimal.

A stack register can also hold text. Text is stored as shown below.

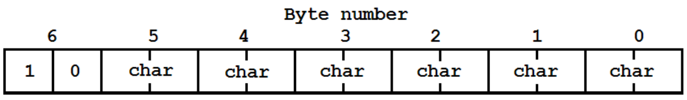


Fig. 14, Text register format

The first nybble is always a hexadecimal 0, binary 0000b. The rest of the register holds the up to 6 characters of ALPHA data. Character no. 1 is the leftmost one. If there is less than 6 characters in the register, it is filled up from the right end.

For example, the three letters "XYZ" would be stored in a register as 1000000058595A hexadecimal.

The ALPHA register

The registers M, N, O and P together make up the ALPHA register. Since the operating system knows that the contents of these registers is ALPHA data, the leading l0h byte is not needed. It is thus possible to store up to 7 characters in each register.

Characters are added to the ALPHA register from the right end of register M and all the characters in the register is pushed to the left. The leftmost character is pushed to the right end of the next register.

Other parts of the status registers

All the space marked as "scratch" is used by the CPU for temporary storage at some time.

Each bit of the 9 leftmost nybbles of registers ├ and e correspond to a key. When a key is pressed, the CPU first reads the bit corresponding to that key. If the bit is set, it starts looking for an assigned function or FOCAL program.

Registers a and b hold the return stack for FOCAL programs and the address pointer in FOCAL programs. The address pointer tells the CPU where to find the next byte of the current FOCAL program. The return stack and the address pointer are all 4 nybbles: 3 nybbles for register address and one nybble for the byte subaddress.

Register c contains the register address of the first of the statistical registers, the "cold start constant", the register address of main memory data register 00 and the register address of the permanent .END. The cold start constant is used to check if the contents of memory has been corrupted (e.g. due to power failure). Each time the CPU starts running, it checks if the contents of these three nybbles is 169 (hexadecimal). If this is not the case, the CPU assumes that memory has been corrupted, and clears the entire continuous memory. It gives the MEMORY LOST message and writes 169 in the three nybbles of register c.

Register d contains all the user and system flags (flags 00 through 55). The leftmost bit is flag 00.

Register e contains the flags for assignments to shifted keys as mentioned above, and the current line number in the current FOCAL program.

ROM block Structure

All system memory (ROM) blocks from 5 and up must have a certain structure, described in this paragraph.

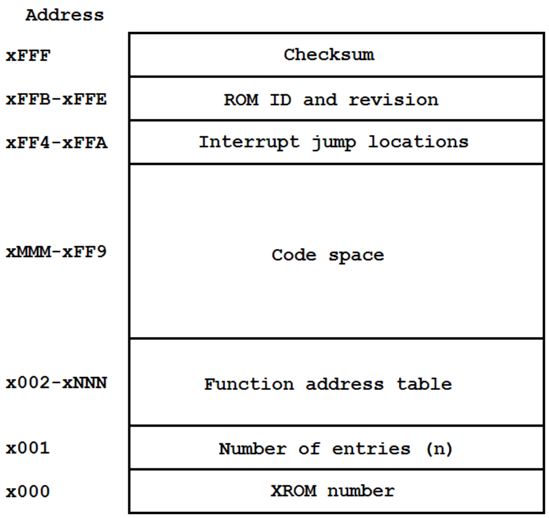


Fig. 15, ROM block structure

The addresses xNNN and xMMM are explained below. Refer to section 12: "Developing your own ROM" for a fully commented example of a user-developed ROM.

The very first word of the ROM is the XROM number. Possible XROM numbers range from 0 through 31 (decimal). The next word gives the number of functions and FOCAL programs in the ROM - the maximum number is 64 (decimal). No two blocks may have the same XROM number. The XROM numbers of most modules and peripherals available are listed in appendix E.

The next part of the ROM structure is the Function address table (FAT). The FAT is a look-up table that tells the CPU where to find the functions and/or FOCAL programs in that ROM block. Each function and each label in a FOCAL program occupies one entry in the FAT, and each entry takes up two words.

The FAT cannot hold more than 64 entries, but it can hold less. The end of the FAT is marked by a null entry, i.e. two words with the value 000.

The first word of a FAT entry is of the form t0a and the second is of the form 0bc. t is the type, where 0 means an M-code routine (function) and 2 means a FOCAL program. abc is the address of the first executable word.

Let's take an example. There is a FOCAL program starting at address x460 and an M-code function starting at address x807 in the ROM. The FAT would look like this:

Address	Word	Comment
x000	011	The XROM number of this ROM is 11h=17d.
x001	002	Two entries
x002	204	The first entry is a FOCAL program (t=2)
x003	060	It starts at address x460 (a=4, bc=60)
x004	008	The second entry is an M-code routine (t=0)
x005	007	It starts at address x807 (a=8, bc=07)
x006	000	Two null words at
x007	000	the end of the FAT

The length of the FAT varies according to how many FAT entries there are. Recall that the FAT starts at address x002, each entry takes up 2 words and the end of the FAT takes up two words. This means that the FAT takes up (n x 2 + 4) words.

In the above example, we find that xNNN (last word of the FAT) in the above figure is 2 x 2 + 3 = x007 and that xMMM (first word of code) is 2 x 2 + 4 = x008.

The code space is where the FOCAL programs and functions are actually stored.

The next part of the ROM structure is the interrupt jump locations. Each time a certain event occurs, the CPU checks the interrupt locations in all blocks. An interrupt location normally contains a null word or a jump instruction.

If, for example, an M-code routine in the block needs to react to MEMORY LOST, the interrupt location for MEMORY LOST would contain a jump instruction. On MEMORY LOST, the jump is executed, the routine runs and terminate with a jump to address 27F3. This returns control to the operating system.

The below table list the interrupt addresses and when they are checked.

Address	Checked
xFF4	During PSE. The pause timer is in A S&X. Called 92 times each pause.
xFF5	If system flag 53 or peripheral flag 13 is set. The timer stops the polling of this address, i.e. if the timer has business to perform, this address in other ROMs is never asked.
xFF6	On wakeup with no key pressed.
xFF7	When the calculator is turned off.
xFF8	Just before the CPU stops.
xFF9	On wakeup.
xFFA	On MEMORY LOST

Table 7, Interrupt addresses

The second last part of the ROM structure is the ROM ID and revision number in addresses xFFB-xFFE. The ROM ID is two letters in addresses xFFD and xFFE, and the revision is typically a letter and a number in addresses xFFB and xFFC. As an example we look at the TIME 2C module. It has ROM ID "TM" and version "2C". The contents of addresses 5FFB through 5FFE are:

```
5FFB C
5FFC 2
5FFD M
5FFE T
```

The very last word is the ROM checksum. This is calculated by adding up all other words in the block with wrap-around carry (i.e. each time the sum exceeds 1023, one extra is added), and then taking the 2's complement of the sum.

When using the HEPAX file system, there is no need for you to worry about XROM numbers, FAT entries, etc. The HEPAX file system will automatically take care of all these details.

Section 8:

HP-41 microprocessor

Introduction to the CPU

The Central Processing Unit (CPU) is the "brain" of the calculator. All information processing (calculation, copying contents of memory, etc.) goes via the CPU.

Within the CPU there are a number of registers used for temporary storage of the information the CPU is working on. There are three major groups of registers: The Arithmetic, Storage and Address registers. In addition to these registers there are some special registers and flags.

Name	Length	Use
C	56 bit	Accumulator
A	56 bit	Primary arithmetic register
B	56 bit	Secondary arithmetic register
M	56 bit	Storage register
N	56 bit	Storage register
G	8 bit	Storage register
PC	16 bit	Program counter
STK	16 bit	Bottom of the 4-level CPU return stack
KY	8 bit	Keyboard buffer register
ST	8 bit	Flag register
T	8 bit	Beeper output register

Table 8, CPU registers

The ST register contains the status of CPU flags 0-7. In addition to these, the CPU also contains 6 more flags that can only be accessed individually.

The STK registers is the CPU return stack. Over the bottom register there are three more 16-bit registers that cannot be accessed.

Information can be moved in different directions between registers as shown below. A double line indicates bidirectional transfer, a single line indicates only one-way transfer.

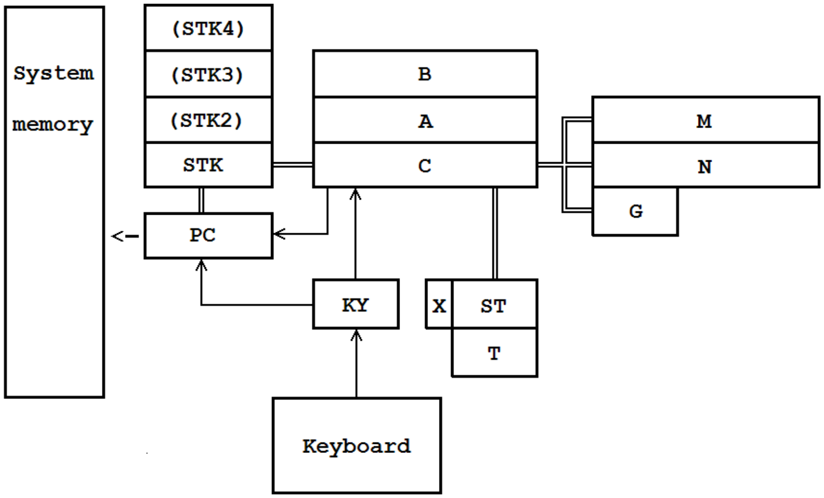


Fig. 16, CPU register connections

The CPU C register also connects directly to system memory, user memory, the display and peripheral units like the HP-IL module or card reader.

It is important that you do not confuse the CPU registers with the status registers described in section 7. Even though some of them have the same or similar names, the CPU registers have nothing to do with the status registers. The CPU also contains 14 flags - these are all different from the user and system flags accessed with the FOCAL instructions FS?, SF and CF.

More about the structure of registers

To the CPU, a 56-bit register consist of 14 nybbles or digits, numbered 0 to 13, starting at the right end of the register.

The CPU can access all or part of the 56-bit registers. The registers are divided into fields as shown below.

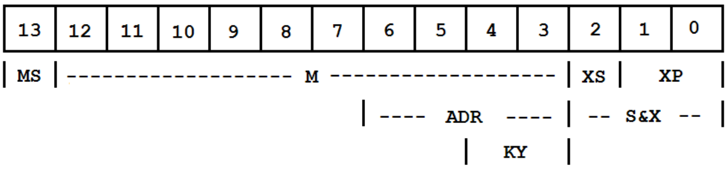


Fig. 17, Structure of registers

Abbreviation	Full name	Digits
MS	Sign of mantissa	13
M	Mantissa	3-12
XS	Sign of exponent	2
XP	Exponent	0-1
S&X	Sign of exponent and exponent	0-2
ADR	Address field	3-6
KY	Key buffer field	3-4

Table 9, Register fields

We will refer to any part of a register using square brackets. For example, C[6:3] means the 6th, 5th, 4th and 3rd digit of the C register. We could also refer to this part of the register as C ADR, meaning "C register, address field".

The CPU can also access one digit or any continuous range of digits. This is done by using pointers, as explained in section 9: "The M-code instructions".

The arithmetic registers (A, B, C)

There are three arithmetic registers named C, A and B. They are of different importance - the most important register is the C register, known as the accumulator.

All data transfer to and from user memory and peripheral units goes via the C register.
The A register is the primary arithmetic register and the B register is the secondary arithmetic register. The A register may be used for both operands and results, whereas the B register can only hold operands.

The storage registers (M, N, G)

The CPU contains two full-size storage registers, the M and the N registers. In addition to these, there is also an 8-bit storage register named G. The storage registers can only exchange data with the C register.

The M and N registers exchange data with the full C register. The G register only exchanges data with two digits of the C register as specified by the pointer. Refer to section 10: "The M-code instructions" for an explanation of the use of pointers.

The address registers (PC, STK)

Just like any other conventional type computer, the HP-41 CPU need to keep track of where to find the next instruction to be executed. The 16-bit PC register is used tor this purpose.

The PC always contain the address of the next instruction to be executed. Normally, the PC is simply incremented by one after each instruction.

The CPU also contains a 4-register stack for return addresses. Only the bottom register of the stack can be accessed. The instructions working with the return stack refer to this bottom register simply as STK. Whenever an address is put on the stack from the C register ("pushed") or taken from the stack to the C register ("popped"), the stack automatically moves, just like the normal RPN number stack.

In the case of a GO (go to address) instruction, the jump address is simply loaded into PC. In the case of an XQ (execute subroutine) instruction, the PC is copied to the return stack and then overwritten by the XQ address. When a "return" instruction is encountered, the PC is loaded with the return address from the stack.

The data paths around the PC register are a little complicated:
- The C register can only write to the PC register
- The KY register may be written to the lower 8 bits of PC.

You'll see that there is no way to read directly from the PC register to the C register.

Other registers and flags

The KY register

When a key is pressed, its keycode will be placed in the KY register (provided that another key is not still held down). The keycodes are shown in figure 18 below.

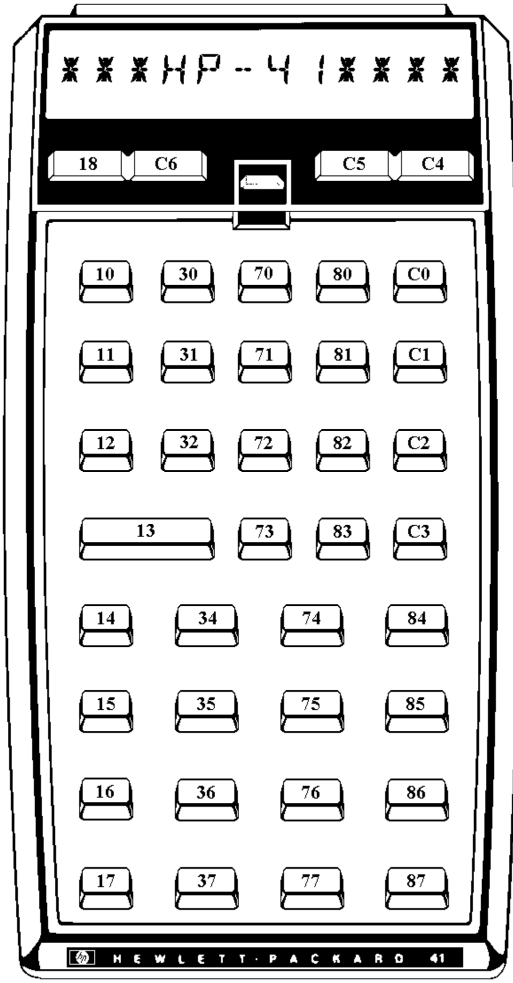


Fig. 18, The internal keycodes of the HP-41

Note that these codes are different from the codes you may be familiar with from Synthetic Programming.

There is also a keydown flag that is set when a key is down. It can be tested and cleared. If a key is down when the clear instruction is given, it is set again immediately.

The KY register may be copied to the lower 8 bits of the PC register - in effect creating a jump depending on the key pressed.

The ST register and the CPU flags

The most often used flag in the HP-41 CPU is the carry flag. It is used to control jumps and returns.

The carry flag is set if a test result is true or a calculation results in an over- or underflow. Unlike all other flags, the carry flag is cleared after each instruction that does not specifically set it. Thus, the carry flag remains set only for one instruction.

The HP-41 CPU has 14 flags in addition to the carry flag and the keydown flag. Of these, flags 10-13 have special meanings as shown below. Flag 0-7 may be accessed as the ST register in the same way as user flags 0-7 are exchanged with the user X register by the X< >F or XFA X< >F FOCAL instruction. Flags 8-13 can only be accessed individually.

CPU flag	Meaning
8	Occasional use
9	Occasional use
10	FOCAL program pointer in ROM
11	Stack lift enabled
12	FOCAL program pointer in PRIVATE program
13	FOCAL program running

Table 10, CPU flags

The HP-41 system also contains 14 peripheral (or interrupt) flags. They are set by various peripheral units, and may be read by the HP-41 CPU. If any peripheral flag (0 through 12) is set, flag 13 is also set.

The T register

The T register is the tone register. It is accessed via the ST register and is connected to the beeper. The greater the number in the T register, the louder the tone. See section 11: "M-code for peripheral units" for an explanation of how to make tones.

The pointers

There are two pointers P and Q that can take on values from 0 to 13. They are used to point to a specific part of a 56-bit register.

Some instructions use both pointers, and some use only one (the active pointer). One pointer (either P or Q) is active at any time. This pointer is referred to as PT.

Part IV:

M-code programming

Section 9:

Introduction to M-code

This part of the manual explains about HP-41 M-code. Programming in M-code is somewhat more difficult than FOCAL programming, but it also gives you a lot of new possibilities.

The native language of any CPU is called the machine language of that CPU. On the HP-41, machine language is also known as machine code, microcode or simply M-Code.

Machine language consist of simple instructions like "Increment A" or "Add A and C and put result in C". When you need an advanced FOCAL instruction like SIN or SDEV, the operating system reads your keystrokes and then performs a series of simple M-Code instructions that gives the result you asked for.

Why M-code?

As you know by now, the operating system also takes care of many housekeeping tasks, like keeping track of where in memory your data is stored, reading from and writing to peripheral units, error checking (**DATA ERROR, NONEXISTENT,** etc.)

Naturally, you pay a price for this convenience. Program execution is relatively slow and you can only access memory and peripherals in the way the operating system defines.

With M-code, none of these limitations exist. Here are a few examples of what you can do with M-code:

- Rewrite FOCAL programs to run up to 100 times faster,
- Use high precision arithmetic with 13 digits instead of 10,
- Create subroutines that do not disturb the stack,
- Fast and advanced HP-IL communication,
- Special use of card reader and wand,
- Very fast integer arithmetic,
- Special input routines,
- Create whole new data structures,
- Easy use of hexadecimal numbers,
- Create special tones, e.g. for dialing on your telephone.

How do I program in M-code?

The most important tools you need are pen and paper. Write down your M-code routine and "assemble" it, i.e. convert the mnemonics to hex codes using the tables in appendix C. Enter the hex codes using the HEXEDIT function, and disassemble the code using DISASM. This also allows you to check the addresses of all jumps.

Now run the routine with some test data and check the results.

Why doesn't my routine work?

In most cases, because a jump distance is wrong. Either you have miscalculated a jump, or you have inserted or deleted code without changing all jumps affected.

Also notice that the "port dependent jumps" (covered in section 12) overwrite the contents of the C register. Remember whether your calculations are in hexadecimal or decimal mode. Check that you have not mixed up some "jump if carry" with "jump if not carry". Check that you have given any system subroutines the correct input, and that you take the output from the correct place. And finally, check that you remember to deselect RAM and peripherals.

CPU "bugs"

The most annoying error you can find is an error in the HP-41 itself. The HP-41 CPU contains a number of errors or "bugs". The bugs found to date are:

> PT= 13, PT=PT-1, C=G @PT,+ does not copy G correctly to C. Insert a NOP before the C=G @PT,+ instruction to make it work as expected.
>
> Don't use CLRF, SETF, ?FSET, ?PT=, C< >ST XP, C=C OR A, C=C AND A, T< >ST, ST=T, T=ST immediately after a class 2 instruction. If you need to use any of the above instructions right after a class 2 instruction, insert a NOP after the class 2 instruction.

Not Manufacturer Supported

All information about M-code programming is "NOMAS". NOMAS stands for NOt MAnufacturer Supported – i.e. Hewlett-Packard does not support M-code in any way. Don't call HP if you have problems with your M-code programming.

Instead, you will probably benefit from joining one of the user groups listed in appendix G.

You should note that since there is no official source of information about M-code, some uncertainty prevail. Although we have taken great pains to compile the most accurate information about HP-41 M-code, we cannot guarantee that the information below is absolutely error-free.

"Crashes"

Since there is no error checking when programming in M-code, you are subjected to the full effect of M-code programming errors. This will most often result in the occurrence of a "crash". A crash is a condition where the calculator has a blank or unintelligible display and does not respond to any keys.

This is not in any way dangerous to the calculator. On newer HP-41's, press and hold the ENTER key and press the ON key a few times. Release both keys and press the backarrow key. This will usually return the HP-41 to life. If this doesn't help (and on older HP-41's), take out the batteries for a few seconds, insert the batteries again and press the backarrow key.

If the calculator is still "crashed", remove the batteries and short the rightmost and leftmost terminal in the calculator momentarily. This should clear memory and unlock your HP-41.

Section 10:

The M-code instructions

This section describes all the normal M-code instructions that the HP-41 CPU recognizes. There are some special codes that are used when control of the HP-41 is given temporarily to a peripheral. These codes will be described in the next section.

The HP-41 operating system contains many useful routines that you can call from your own M-code programs. A selection of the most commonly used entry points in the operating system is given in section 12.

The structure of M-code instructions

All instructions consist of one or two 10-bit words. They are divided into four classes according to the two least significant (rightmost) bits as follows:

Word	Class
xx xxxx xx00	0
xx xxxx xx01	1
xx xxxx xx10	2
xx xxxx xx11	3

Table 11, M-code instruction classes

All class 1 instructions are two-word absolute GO and XQ instructions. Class 2 contains all instructions dealing only with register C, A and B, class 3 contains all relative jumps and class 0 contains the remaining instructions.

All instructions have a 10-bit hexadecimal code. For ease of reading, each instruction is also assigned a mnemonic that tells what the instruction does. Example: Hex code 148h means "set CPU flag 6" and has the mnemonic SETF 6.

The mnemonics used in this manual and by the HEPAX disassembler were first created by Jacobs and DeArras and are the de facto standard for HP-41 M-code. Hewlett-Packard has their own mnemonics for all instructions, but have never officially published these.

The only differences between HEPAX mnemonics and Jacobs/DeArras are:

- The active pointer is referred to as PT instead of R in the original mnemonics,
- The exponent field of a registers is referred to as XP instead of X. This also avoids possible confusion with Hewlett-Packard mnemonics that use "X" for the sign and exponent field.
- The peripheral flag instructions (?FI n) have been replaced by descriptive names.

Note that this section only explains the instructions. Refer to section 12: "Creating your own ROM" for examples of M-code programming.

About jumps

The class 1 instructions are the "absolute go to" and "absolute execute" instructions. These instructions are used to jump to a specific address. The class 3 instructions are the "relative jump" instructions. They are used to jump up to 63 addresses forwards or 64 addresses backwards. There is a third kind of jumps called "port dependent jumps". They are used to jump to a specific address within the same block.

All jump types have their advantages and disadvantages, as shown below.

Jump type	Advantages	Disadvantages
Absolute	Can jump to any address in system memory	Routine is fixed to one address.
Relative	Relocatable	Limited range.
“Port Dependent”	Can jump to any address in the current block.	Routine is fixed to one address within the block.

Table 12, Advantages and disadvantages of jump types

Absolute jumps

Absolute jumps should only be used when calling the operating system or a system addressed device. If you use absolute jumps to call your own M-code routines, they must stay in exactly the same memory location. If you (or anyone else) later needs to use your routine in another block, the code must be rewritten.

This example is not as far-fetched as it may sound. For instance, if you decide to have your M-code routines programmed into a ROM module, this module may be plugged into either port and your code will therefore have to operate from a different block address.

Relative jumps

You should use relative jumps within your routines as much as possible. With relative jumps, your routine may be moved to another position within the block or to another block without any problems.

You might even want to create "stopover" jumps if you need to jump further than 63 or 64 addresses. The below example illustrates this:

```
xC32 ?KEY       If a key is down, you must jump 5Bh forward.
xC33 JC +3F     Jump 63d addresses forward
   .
   .
xC6F iii        Part of another routine
xC70 NOP        Clears carry (not needed if iii never sets carry)
xC71 JNC +02    Jump 2 forward, i.e. skip the next address.
xC72 JNC +1C    "Stopover" jump.
xC73 jjj        The other routine continues.
   .
   .
```

First, you jump 63 (=3Fh) addresses forward to address xC72, then you jump 28 (=1Ch) addresses forward (in this case to xC8Eh). In the other routine, the JNC +02 instruction simply skips over the 28-address jump.

Port dependent jumps

If you are creating a whole ROM, you might create subroutines that you wish to call from another part of that ROM. This is done by means of "port dependent jumps". A routine that is called with a port dependent jump must stay at the same address within the block, but the code for your ROM may be relocated to another block without problems.

A port dependant jump is actually a call to a subroutine in the operating system. There are four subroutine calls for jump instructions and four subroutine for execute instructions. They correspond to the first, second, third and last quarter of a 4K ROM block. There are also two subroutine calls for jump and execute within the same quarter block. The word following the subroutine call must contain the address within the quarter you wish to go to or execute.

Port dependent jumps are described in detail in the next section.

Class 0 instructions

Class 0 mainly contains instructions dealing with flags, pointers, data storage and basic peripheral handling. Don't despair - this is the most complicated class of instructions. You don't have to read and understand every instruction - just browse through when first reading this section.

Parameter instructions

The most commonly used instructions in class 0 are the instructions that use a parameter. The below table gives an overview of the parameter instructions. The actual hex codes are given in appendix C.

Mnemonic	Meaning	Parameter
CLRF p	Clear CPU flag p.	0 < = p < = 13
SETF p	Set CPU flag p.	0 < = p < = 13
?FSET p	Set carry if CPU flag p is set.	0 < = p < = 13
PT= p	Set pointer to digit p.	0 < = p < = 13
?PT= p	Set carry if pointer is at digit p.	0 < = p < = 13
LD@PT- p	Load C register digit at pointer with the value p and decrement pointer. Pointer "wraps around".	0 < = p < = Fh
RCR p	Rotate C register p digits right.	1 < = p < = 13
WRIT p	Write C register to selected user memory or peripheral register.	0 < = p < = 15
READ p	Read selected user memory or peripheral register to C register.	1 < = p < = 15
HPIL=C p	Copy C[1:0] to HP-IL, register p.	0 < = p < = 7
SELP p	Select peripheral to take control.	0 < = p < = 15

The LD@PT- p automatically decrements the pointer. If the pointer was at digit 0, it is set to digit 13.

If you need to rotate the C register n digits left, simply rotate it 14 minus n digits right.

Communication with peripheral units is described in the next section: "Using M-code with peripheral units".

Reading from and writing to user memory registers is described in detail later in this section.

Special instructions

The special instructions of class 0 are described below, along with their hexadecimal codes.

General

Mnemonic	Hex	Meaning
NOP	000	No operation - just clears the carry flag and takes time
LDI S&X	130	Load the 10-bit word in the next address into C[2:0].

Pointer instructions

Mnemonic	Hex	Meaning
PT=PT-1	3D4	Decrement pointer. If PT=0, then PT is set to 13.
PT=PT+1	3DC	Increment pointer. If PT=13, then PT is set to 0.
SLCT P	0A0	Select P as the active pointer (PT)
SLCT Q	0E0	Select Q as the active pointer (PT)
?P=Q	120	Set carry if P and Q have same value.

Storage register instructions

Mnemonic	Hex	Meaning
C=M ALL	198	Copy M register to C register.
M=C ALL	158	Copy C register to M register.
C< >M ALL	1D8	Exchange C and M register.
C=N ALL	0B0	Copy N register to C register.
N=C ALL	070	Copy C register to N register.
C< >N ALL	0F0	Exchange C and N register.
C=G@PT,+	098	Copy G register to C register digits at pointer and at pointer + 1.*
G=C@PT,+	058	Copy C register digits at pointer and at pointer + 1 to G register.*
C< >G@PT,+	0D8	Exchange C register digits at pointer and at pointer + 1 with G register.*

ST register instructions

Mnemonic	Hex	Meaning
C=ST XP	398	Copy ST register to C[1:0]
ST=C XP	358	Copy C[1:0] (eXPonent) to ST register.
C< >ST XP	3D8	Exchange C[1:0] and ST register.
ST=0	3C4	Clears the ST register, i.e. clears CPU flags 0-7.

Tone register instructions

Mnemonic	Hex	Meaning
ST=T	298	Copy T register to ST register.
T=ST	258	Copy ST register to T register.
ST< >T	2D8	Exchange ST and T register.

* If PT=13 then C[0] and C[13] is copied. Last digit of G is always last in C, even if PT=13.

Arithmetic and logic instructions

Mnemonic	Hex	Meaning
A=B=C=0	lA0	Clear A, B and C registers.
SETHEX	260	Set CPU to calculate in hexadecimal.
SETDEC	2A0	Set CPU to calculate in decimal.
C=C OR A	370	Perform logical OR on the A and C registers and store result in C.
C=C AND A	3B0	Perform logical AND on the A and C registers and store result in C.

Memory and peripheral handling instructions

Mnemonic	Hex	Meaning
READ DATA	038	Copy the active user memory register to the C register.
WRIT DATA	2F0	Copy C register to the active user memory register.
FETCH S&X	330	Fetches the word at system memory address given in C[6:3] to C[2:0]. Do not fetch from address 0002h, as this will cause a file system reset.
WRIT S&X	040	Writes the word in C[2:0] at system memory address given in C[6:3]. Only works if there is HEPAX RAM* at the address.
RAM SLCT	270	Select the user memory register specified in C[2:0].
PRPH SLCT	3F0	Select peripheral unit specified in C[2:0].

Jump related instructions

Mnemonic	Hex	Meaning
RTN	3E0	Return to address in STK.
?C RTN	360	Return to address in STK if carry is set.
?NC RTN	3A0	Return to address in STK if carry is clear.
POP ADR	1B0	Pop STK. Bottom STK register is copied to C[6:3] and STK drops.
PUSH ADR	170	Push STK up and store C[6:3] in the bottom STK register.
GOTO ADR	lE0	Jumps to the address in C[6:3].
XQ->GO	020	Pop the CPU return stack (loses one return address). This turns the latest XQ into a GO.

* Or other MLDL type RAM.

Display handling instructions

Mnemonic	Hex	Meaning
DSPOFF	2E0	Turns display off.
DSPTOG	320	Toggles display between on and off.

Keyboard handling instructions

Mnemonic	Hex	Meaning
CLRKEY	3C8	Clears the keydown flag. If a key is down, the flag is set again immediately.
?KEY	3CC	Sets carry if keydown flag is set.
C=KEY KY	220	Copy key code from KY to C[4:3].
GOTO KEY	230	The contents of the KEY register is written in the lowest byte of the program pointer PC.

Battery and power instructions

Mnemonic	Hex	Meaning
?LOWBAT	160	Set carry if the battery is low.
POWOFF	060	Must be followed by a NOP. If display is on: stop CPU. If display is off: turn HP-41 off.

I/O handling instructions

Mnemonic	Hex	Meaning
?PBSY	3AC	Set carry if HP-82143A printer busy.
?CRDR	32C	Used with card reader. See section 11, "M-Code for peripheral units".
?WNDB	22C	Set carry if there is data in the buffer of the optical wand.
?EDAV	0AC	Set carry if the emitting diode of the HP-82242 IR module is available.
?IFCR	16C	Set carry if the HP-IL interface is ready (InterFace Clear Received).
?SRQR	2AC	Set carry if the HP-IL interface needs service (Service ReQuest Received).
?FRAV	12C	Set carry if a frame is available in the HP-IL interface (FRAme aVailable).
?FRNS	26C	Set carry if the frame transmitted on HP-IL does not return as sent (Frame Return Not as Sent).
?ORAV	0EC	Set carry if the output register is available (Output Register AVailable).
?ALM	36C	Set carry if an alarm from the timer has occurred.
?SERV	2EC	Set carry if any peripheral unit needs service. The SERV flag is set if any other interrupt flags is set.

HEPAX instructions

Mnemonic	Hex	Meaning
ENBANK1	100	Enables primary bank. Only works in the same system memory block as the instruction, and only if supported by the ROM.
ENBANK2	180	Enables secondary bank. Only works in the same system memory block as the instruction, and only if supported by the ROM.
ENBANK3	140	Enables third bank. Only works in the same system memory block as the instruction, and only supported by the ROM.
ENBANK4	1C0	Enables fourth bank. Only works in the same system memory block as the instruction, and only supported by the ROM.
WPTOG	1F0	Toggles write protection status of HEPAX RAM in system memory block specified in C[0].
ROM BLK	030	Moves HEPAX ROM to system memory block specified in C[0].

Note that if the following instructions are used immediately after a class 2 instruction, you might get an unexpected result:

> CLRF, SETF, ?FSET, ?PT=, C< >ST XP, C=C OR A, C=C AND A, T< >ST, ST=T, T=ST.

If you need to use any of the above instructions right after a class 2 instruction, insert a NOP after the class 2 instruction.

Accessing user memory registers

User memory registers are physically grouped in blocks of 16 registers. One user memory register is active at any time, and the block that contains this register is the active block. You select the active user memory register with the RAM SLCT instruction.

The WRIT DATA instruction copies the CPU C register to the active user memory register. The READ DATA instruction copies the active register to C.

You could also write to any register in the active block of user memory using the WRIT 0 through WRIT 15 instructions. The corresponding READ instructions are, however, only valid for registers 1 through 15. This means that to read to register 0 of any block, you must select it directly using the RAM SLCT instruction and then use READ DATA.

When you select a peripheral unit (e.g. the display), you *must* deselect the user memory. This is done by selecting a non-existent RAM chip using the RAM SLCT instruction with 0l0h in C[2:0]. If you forget this, your HP-41 will almost surely crash.

Class 1 instructions

All class 1 instructions are two words long. The two words have the following structure:

First word:	ccccdddd01
Second word:	aaaabbbbtt

Where tt is the type of instruction and aaaabbbbccccdddd is the address.

The type is interpreted as follows:

t t	Mnemonic	Instruction type
00	?NC XQ	If carry clear then execute subroutine
01	?C XQ	If carry set then execute subroutine
10	?NC GO	If carry clear then go to address
11	?C GO	If carry set go to address

Table 13, Class 1 jump types

The below FOCAL program calculates the code of all four types of jumps, but let's first work out a jump by hand:

We need to execute the subroutine that disables user memory and enables the display (address 07EF) if carry is clear. The jump is calculated as follows:

Address 07EFh hexadecimal is 0000 0111 1110 1111 binary. Jump type is ?NC XQ, i.e. tt is 00. The code is:

First word:	1110111101	(binary)	3BD (hex)
Second word:	0000011100	(binary)	01C (hex)

To calculate a jump automatically, execute the "JUMP" FOCAL program shown below. Enter the jump type (0, 1, 2 or 3). Enter the address at the prompt. The jump type and the two words are displayed. To let the "JUMP" program calculate the above jump, do the following:

Keystrokes:	Display:	
XEQ JUMP	TYPE 0-3?	
0 R/S	?NC XQ _ _ _ _	Jump type 0, ?NC XQ
07EF R/S	?NC XQ 07EF:	
	3BD,O1C	The two words are 3BD and 01C.

And now for the promised FOCAL program:

```
01  LBL "JUMP"
02  "TYPE 0-3?"
03  0
04  PROMPT
05  4
06  MOD
07  STO 00
08  .003
09   +
10  "?C GO"
11  ISG X
12  "?NC GO"
13  ISG X
14  "?C XQ"
15  ISG X
16  "?NC XQ"
17  "├ (space)"
18  4
19  HPROMPT
20  4
21  DECODYX
22  "├:"
23  1
24  HEPAX     The BCD-BIN
25  3         function
26  X<>Y
27  -2
28  HEPAX     The SHIFTYX
29  11        function
30  HEPAX     The OR
31  9         function
32  1023
33  HEPAX     The BCD-BIN
34 3          function
35  X< >Y
36  HEPAX     The AND
37  1         function
38  LASTX
39  X < >Y
40  3
41  DECODYX
42  "├,"
43  RDN
44  10
45  HEPAX     The SHIFTYX
46  11        function
47  -2
48  HEPAX     The SHIFTYX
49  11        function
50  RCL 00
51  HEPAX     The BCD-BIN
52  3         function
53  HEPAX     The OR function
54  9
55  3
56  DECODYX
57  AVIEW
58  CLX
59  END
```

Program listing of the "JUMP" program

Class 2 instructions

All class two instructions operate on a specific part of the registers involved. The following possibilities exist:

ALL	The entire register.
M	The mantissa, digits [12:3].
S&X	Sign and exponent, digits [2:0].
MS	The sign of the mantissa.
XS	The sign of the exponent.
@PT	The digit at the active pointer.
PT<-	From digit 0 up to the digit at the active pointer, inclusive.
P-Q	From pointer P to pointer Q, from right to left.

Table 14, Fields used with class 2 instructions

When using class two instructions, one of the above fields must always be specified.

The class two instructions are:

Mnemonic	Meaning
A=0	Clear the A register.
B=0	Clear the B register.
C=0	Clear the C register.
A=C	Copy C register to A register.
C=B	Copy B register to C register.
B=A	Copy A register to B register.
A< >C	Exchange A and C registers.
C< >B	Exchange C and B registers.
A< >B	Exchange A and B registers.
C=C+A	Add C and A and put result in C register.
A=A+C	Add A and C and put result in A register.
A=A+B	Add A and B and put result in A register.
C=C+C	Double C = shift C one bit left.
C=A-C	Subtract C from A and put result in C register.
A=A-C	Subtract C from A and put result in A register.
A=A-B	Subtract B from A and put result in A register.
C=C+1	Increment C.
A=A+1	Increment A.
C=C-1	Decrement C.
A=A-1	Decrement A.
?C≠0	Set carry if C different from 0.
?A≠0	Set carry if A different from 0.
?B≠0	Set carry if B different from 0.
?A≠C	Set carry if A different from C.
?A<C	Set carry if A less than C.
?A<B	Set carry if A less than B.
RSHFC	Shift contents of C register one digit right.
RSHFA	Shift contents of A register one digit right.
RSHFB	Shift contents of B register one digit right.
LSHFA	Shift contents of A register one digit left.
C=0-C	Replace C with 1's complement of C.
C=-C-1	Replace C with 2's complement of C.

Let's take a few examples:

C=0 S&X	Clear the S&X field of the C register, i.e. C[2:0].
A=C MS	Copy the sign of the mantissa of C to the same field of A.
C=C+1 M	Increment the C register mantissa.
?A<C @PT	Set carry if the digit at the active pointer in the A register is less than the same field of the C register.
RSHFB ALL	Shift the entire B register one digit right.

If any class 2 operation results in the most significant digit becoming greater than 9 (in decimal mode) or Fh (in hexadecimal mode), then carry is set. Carry is also set if a subtraction results in a borrow.

Note that due to an error in the HP-41 CPU the C=-C-1 instruction sometimes sets carry. Therefore there should be at least one instruction (e.g. a NOP) between this instruction and the first following jump instruction.

Class 3 instructions

Class three instructions are relative jumps, i.e. of the type "jump nn instructions forwards or backwards". These jumps should be used whenever possible, because they are freely relocatable.

There are four types of relative jumps:

JNC +nn	Jump nn instructions forwards if carry clear.	0l < = nn < = 3Fh
JC +nn	Jump nn instructions forwards if carry set.	0l < = nn < = 3Fh
JNC -nn	Jump nn instructions backwards if carry clear.	0l < = nn < = 40h
JC -nn	Jump nn instructions backwards it carry set.	0l < = nn < = 40h

Table 15, Class 3 jump types

The structure of the class 3 instructions is:

d6 d5 d4 d3 d2 d1 d0 n 1 1

where ddddddd is the signed jump distance and n specifies if the instruction is a "jump if carry" or "jump if not carry".

Section 11:

M-code for peripheral units

This section describes how M-code allows you to communicate with the following peripherals units in special ways:

- Tone generator
- Display
- Printer
- Optical wand
- Card reader
- Timer
- HP-IL interface

Using the tone generator

The tone generator (the beeper) is accessed using the ST=T (298h), T=ST (258h) and ST<>T (2D8h) instructions.

The T register is connected to the beeper, and tones are created by repeatedly changing the value in the T register (usually exchanging 00h and FFh). Other values may be used, but will result in a weaker tone.

The frequency is determined by the swap rate. Usually, you would put FFh in the T register, wait a while, put 00h in the T register, etc. Each HP-41 M-code word takes about 158 µs to execute (one machine cycle), so the frequency is

$$f = \frac{1}{(no.\ of\ FFh\, cycles + no.\ of\ 00\text{h}\, cycles)\, x\ 158\,10\text{E-6}}$$

You can create odd-sounding tones by leaving the FFh and 00h in the T register for a different number of cycles. Note that if you have “speeded” your HP-41, the tone frequency will be increased.

ROM character codes

Each character is represented by a 9-bit ROM character code. The ROM character codes are used for ROM function names and when writing to the display. Note that the ROM character code is different from the user character code described in section 3: "The Extended Functions". User character code is used for FOCAL programs and by the XFA XTOA and XFA ATOX functions.

The ROM character code has the following structure:

Bit(s)	Meaning
8	Specifies "special character".
7-6	Specifies punctuation.
5-4	Specifies the row of the ROM character table.
3-0	Specifies the column of the ROM character table.

Table 15, ROM character code structure

The punctuation is determined as follows:

Bit 7	Bit 6	Punctuation
0	0	none
0	1	. (period)
1	0	: (colon)
1	1	, (comma)

Table 16, ROM character code punctuation

Bit 8 specifies if the character is a "special character". On older HP-41C/CV/CX calculators, only the first row of special characters existed, the remaining three rows simply displaying as spaces. However, the newer HP-41 calculators (known as "halfnut" calculators, and identified by a 1/16" black rim on the display) have four rows of special characters as shown below.

Normal characters

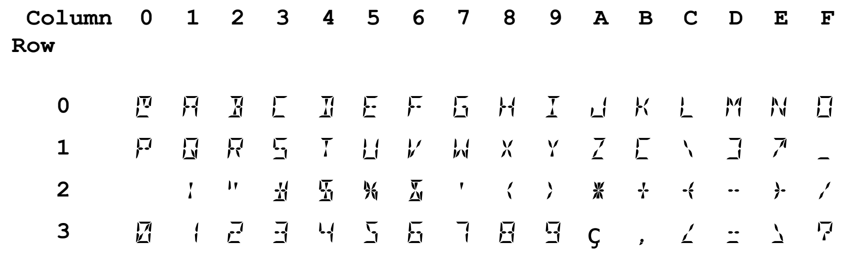

Column
Row
0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3

Fig. 19, Normal ROM characters

Special characters (older HP-41 calculators)

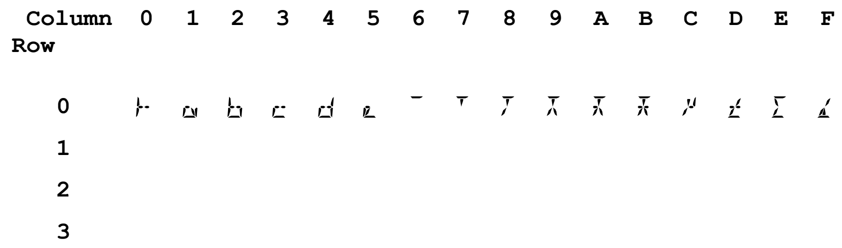

Column
Row
0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3

Fig. 20, Special ROM characters

Special characters ("halfnut" HP-41 calculators)

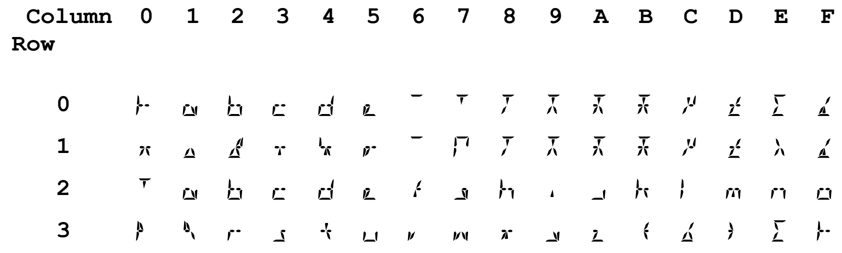

Column
Row
0 1 2 3 4 5 6 7 8 9 A B C D E F
0 1 2 3

Fig. 21, Halfnut special ROM characters

Using the display

The display is a peripheral unit and must be selected using the PRPH SLCT instruction. The procedure for this is as follows:
1. Issue 0l0h, RAM SLCT (270h) to de-select the user memory.
2. Issue 0FDh, PRPH SLCT (3F0h) to select the display.
System subroutine 07F6h (see section 12) performs this task.

Once the display has been selected, it is accessed with the WRIT and READ instructions.

The annunciators are set using the WRIT DATA (2F0h) instruction and may be read using the READ M (178h) instruction. The last 12 bits of the C register each corresponds to one display annunciator as follows:

Bit	11	10	9	8	7	6	5	4	3	2	1	0
Annunciator	BAT	USER	G	RAD	SHIFT	0	1	2	3	4	PRGM	ALPHA

Fig. 22, Display annunciators

All the remaining instructions that work when the display is selected have some common features:
- Field: The instruction affects a range of bits (8, 8-0, 7-0, 7-4 or 3-0).
- Number of characters: The instruction affects a number of characters in the display (1, 4, 6 or 12).
- Rotation: Data is always written to or read off one end of the display (right or left). When data is written, it is pushed onto the end, and the remaining data is shifted to make room. When data is read, it is pulled off the end and shifted back onto the other end of the display.
- Digits in C: Each character in the display occupies 1, 2 or 3 digits in the C register. Data is always taken off the right end of the C register.

All possible combinations are given in the table below.

Instruction	Hex	Bits	No. of chars	Rotation	Digits in C
READ DATA	038	3-0	12	left	1
WRIT 0(T)	028	3-0	12	right	1
READ 1(Z)	078	7-4	12	left	1
WRIT 1(Z)	068	7-4	12	right	1
READ 2(Y)	0B8	8	12	left	1
WRIT 2(Y)	0A8	8	12	right	1
READ 3(X)	0F8	7-0	6	left	2
WRIT 3(X)	0E8	7-0	6	right	2
READ 4(L)	138	8-0	4	left	3
WRIT 4(L)	128	8-0	4	right	3
WRIT 5(M)	168	7-0	6	left	2
READ 6(N)	1B8	8	1	left	1
WRIT 6(N)	lA8	8-0	4	left	3
READ 7(O)	1F8	3-0	1	right	1
WRIT 7(O)	lE8	3-0	1	right	1
READ 8(P)	238	7-4	1	right	1
WRIT 8(P)	228	7-4	1	right	1
READ 9(Q)	278	8	1	right	1
WRIT 9(Q)	268	8	1	right	1
READ 10(├)	2B8	3-0	1	left	1
WRIT 10(├)	2A8	3-0	1	left	1
READ 11(a)	2F8	7-4	1	left	1
WRIT 11(a)	2E8	7-4	1	left	1
READ 12(b)	338	7-0	1	right	2
WRIT 12(b)	328	7-0	1	right	2
READ 13(c)	378	7-0	1	left	2
WRIT 13(c)	368	7-0	1	left	2
READ 14(d)	3B8	8-0	1	right	3
WRIT 14(d)	3A8	8-0	1	right	3
READ 15(e)	3F8	8-0	1	left	3
WRIT 15(c)	3E8	8-0	1	left	3

Table 17, Display handling instructions

Using the HP-82143A printer

There are two ways you can communicate with the HP-82143A printer: With the ?PBSY instruction and with the SELP 9 instruction.

The ?PBSY instruction (hex 3AC) sets carry if the printer is busy.

The SELP 9 instruction (hex 264) transfers control of the HP-41 system to the printer. The printer has control until an instruction with the rightmost bit set is encountered.

While the printer has control, it understands the following instructions:

Mnemonic	Hex	Meaning
BUSY?	003	Set carry if the printer is busy (just like ?PBSY)
ERROR?	083	Set carry in case of a printer error.
POWON?	043	Set carry if the printer is on.
BUF=BUF+C	007	Copy the byte in C[1:0] to the printer buffer.
C=STATUS	03A	Copy the printer status word to C[1:0]. Note that the next word after this instruction must be 001h.

Table 18, Printer handling instructions

The structure of the printer status word is:

Bit	Meaning
15-14	Indicates the printer mode. Both clear means MAN mode, bit 15 set indicates TRACE mode and bit 14 set means NOR mode. Bit 14 and 15 can never be set at the same time.
13	The PRINT key on the printer is down.
12	The PAPER ADVANCE key on the printer is down.
11	The printer is OUT OF PAPER.
10	The printer battery is low.
9	The printer is idle (i.e. not printing).
8	The printer buffer is empty.
7	The printer is using lower case (user flag 13 set).
6	The printer is in graphics mode (column mode).
5	The printer is using double wide characters (user flag 12 set).
4	The printer is printing right justified.
3	The last byte sent was an End-Of-Line byte.
2	A print error is occurring.
1-0	Always set.

Table 19, Structure of printer status word

The optical wand

You can communicate with the HP-82153A optical wand using two instructions: ?WNDB (hex 22C) and READ DATA (hex 038).

?WNDB sets carry if there is data in the wand buffer.

To read data from the wand buffer, you must first deselect the user memory and select the wand: 0l0h, RAM SLCT (270h), 0FEh, PRPH SLCT (3F0h). READ DATA now reads one byte at a time from the buffer to C[1:0]. The contents of the rest of the C register is destroyed.

Magnetic card reader

To access the HP-82104A magnetic card reader, you must deselect the user memory and select the card reader: 0l0h, RAM SLCT (270h), 0FCh, PRPH SLCT (3F0h).

The card reader now responds to the below 13 instructions. Note that some of the instructions set the card reader interrupt flag. This flag can later be tested with the ?CRDR (32Ch) instruction that sets carry if the card reader interrupt flag is set.

The instructions that are used with the card reader are the following:

Mnemonic	Hex	Meaning
READ DATA	038	Read one record from the card reader buffer to C[13:7] and C[6:0].
WRIT DATA	2F0	Write one record from C[13:7] to the card reader buffer. If there is a card in the card reader and the motor is running, this instruction will write the data to the card. If the data is not written immediately, it must be written later using the WRIT 1(Z) instruction.
WRIT 0(T)	028	End write cycle.
WRIT 1(Z)	068	Used when the motor is running to start a write cycle.
WRIT 2(Y)	0A8	End read cycle.
WRIT 3(X)	0E8	Used to prepare for reading (Set read mode).
WRIT 5(M)	168	Set card reader interrupt flag if the inserted card is write protected. This instruction will only work immediately after the motor has started.
WRIT 7(O)	lE8	Set card reader interrupt flag if there is a card in the card reader and the motor is running.
WRIT 11(a)	2E8	Depends on the operation mode: - In read mode clears the card reader interrupt flag if a record can be read from the card reader buffer. - In write mode sets the card reader interrupt flag if a record can be written to the buffer.
WRIT 12(b)	328	Stop the card reader motor.
WRIT 13(c)	368	Start the motor. If the WRIT 1(Z) (start write cycle) instruction has been executed, the motor will begin running slowly, even without any card inserted. If the WRIT 1(Z) instruction has not been executed, the motor will not start before a card is inserted. After the card has passed the card reader, the motor will run slowly.
WRIT 15(e)	3E8	Set the card reader interrupt flag if the card reader external flag is set.

Table 20, Card reader handling instructions

The Timer

The timer chip found in the HP-82182A module and in the HP-41CX is a rather complicated device. Like most other peripheral units it has an interrupt flag, the user memory must be deselected and the timer must be selected before use. Use 0l0h, RAM SLCT (270h), 0FBh, PRPH SLCT (3F0h).

The timer contains a number of registers:
- Two clock registers (A and B).
- Two alarm registers (A and B).
- Two scratch registers (A and B).
- An A/B pointer.
- An accuracy factor register.
- An interval timer.
- A 13-bit status register.

One clock register, one alarm register and one scratch register will be active at any time. Which is active is determined by the A/B pointer.

The times in clock and alarm registers is written as "number of 1/100 seconds since start", decimally, right aligned. The time is given as time since January 1, 1900. This means that, as far as the timer is concerned, the end of the world occurs at 9:46:40 AM on the morning of December 20, 2330.

Clock register A will usually contain the current time and clock register B will contain the stopwatch time.

The alarm register A will usually contain the time of the next alarm. If no alarms are set, the alarm register A will be cleared. The alarm register B will usually contain the constant 09999999999000h. If the timer ROM at any time finds out that there is anything but this constant in alarm register B, it will assume that power has been disconnected, and all information in the timer will be cleared. This procedure is the same as with the 169h constant in the user memory status register c.

Scratch register A is used to hold the time when the clock was last corrected. This is used by the CORRECT function to calculate a new accuracy factor. Bit 5 of the scratch register B is set if the clock displays in 24-hour format (CLK24) and bit 6 of scratch register B is set if the clock function displays both time and date (CLKTD).

The following instructions are available:

Mnemonic	Hex	Meaning
WRIT 0(T)	028	Copy the C register to the active clock register of the timer.
READ DATA	038	Copy the content of the active clock register to the C register.
WRIT 1(Z)	068	As WRIT 0(T), used after READ 1(Z). Takes into account the time used since reading the time.
READ 1(Z)	078	As READ DATA, used when correcting the time using T+X.
WRIT 2(Y)	0A8	Copy the C register to the active alarm register.
READ 2(Y)	0B8	Copy the active alarm register to the C register.
WRIT 3(X)	0E8	If the A/B pointer is set to A: Copy bits 0-5 of the C register to the timer status register. Note that bits 0-5 of the timer status register may only be cleared, not set, by this instruction. If the A/B pointer is set to B: Write bits 4-16 of the C register to the timer accuracy factor register. Bits 4 through 15 can give a value of 0.0 through 99.9, bit 16 indicates the sign of the factor.
READ 3(X)	0F8	If the A/B pointer is set to A: Copy all 13 status bits to the 13 least significant (rightmost) bits of the C register. If the A/B pointer is set to B: Copy the accuracy factor register to bits 4 through 16 of the C register.
WRIT 4(L)	128	Copy C register to the active scratch register.
READ 4(L)	138	Copy the active scratch register to the C register.
WRIT 5(M)	168	Copy the 5 least significant digits of the C register to the interval timer and start the interval timer. The timer can assume values of 0.01 through 999.99 seconds. Each time the interval timer period has passed, the timer interrupt flag is set. This function is used by the CLOCK function that updates the display every second or every minute.
READ 5(M)	178	Copy the value of the interval timer to C[4:0].

WRIT 7(O)	lE8	Stop the interval timer.
WRIT 8 (P)	228	Clear test mode A or B, depending on the A/B pointer.
WRIT 9(Q)	268	Set test mode A or B, depending on the A/B pointer. The test instructions are used in connection with measurements on the timer chip.
WRIT 10(├)	2A8	Disable the active alarm (A or B), but does not clear them. When the calculator is turned off, alarm A is re-enabled. Timer alarms (negative stopwatch time) cannot be disabled.
WRIT 11(a)	2E8	Re-enable the disabled alarm.
WRIT 12(b)	328	Stop the clock in the active clock register. Clock register A will be re-started as soon as the CPU stops.
WRIT 13(c)	368	Start the clock in the active clock register.
WRIT 14(d)	3A8	Set the A/B pointer to B.
WRIT 15(e)	3E8	Set the A/B pointer to A.

Table 21, Timer handling instructions

The structure of the status register is:

Bit	Meaning
0	Set if the time in Alarm A is the same as in Clock A,
1	Set if an overflow has occurred in Clock A.
2	Set if the time in Alarm B is the same as in Clock B.
3	Set if an overflow has occurred in Clock B.
4	Set if the interval timer has counted a whole interval.
5	Set if the timer chip supply voltage has been lower than certain minimum.
6	Set if Clock A is counting forwards (may be cleared and set using WRIT b and WRIT c with the pointer set to A).
7	Set if Clock B is counting forwards (may be cleared and set using WRIT b and WRIT c with the pointer set to B).
8	Set if Alarm A is enabled. Since time alarms are usually enabled, this flag is usually set.
9	Set if Alarm B is enabled. Always clear since stopwatch alarms are not possible. Timer alarms occur as a result of overflow in the stopwatch register (bit 3).
10	Set if the interval timer is running.
11	Timer is in test A mode.
12	Timer is in test B mode.

Table 22, Structure of timer status registers

The timer is a system addressed device and will always address itself to ROM block 5. The timer ROM contains a number of routines that makes use of the timer somewhat easier than the above instructions suggest.

The HP-IL interface

The HP-82160A HP-IL interface module is the most complicated peripheral device used with the HP-41. To program the HP-IL loop, it is strongly recommended that you read the book "The HP-IL System" by Kane, Harper and Ushijima, or "Control the world with HP-IL" by Gary Friedman. These books describe how to program the HP-IL loop.

The HP-IL interface contains 7 registers, each of them one byte long. The HP-IL, registers are used as follows:

Register 0, Status Register.

Bit 0:	Master clear
Bit 1:	Clear IFC received
Bit 2:	When writing: Set Local Ready. When reading: RFC received
Bit 3:	Send Service Request
Bit 4:	Listener active
Bit 5:	Talker active
Bit 6:	Controller active
Bit 7:	System controller

Register 1, Control Interrupt Register.

When writing:

Bit 0:	Enable FI line
Bit 1-4:	Unused
Bit 5-7:	Output Control Bits

When reading:

Bit 0:	Output Register Available
Bit 1:	Frame Received Not as Sent
Bit 2:	Frame Available
Bit 3:	Service Request Received
Bit 4:	Interface Clear Received
Bit 5-7:	Input Control Bits

Register 2, Data Bits Register.

Bit 0-7:	When writing: Input Data Bits When reading: Output Data Bits

Register 3: Parallel Poll Register.

Bit 0-2:	Parallel Poll Response Bit Designation
Bit 3:	Parallel Poll Polarity
Bit 4:	Parallel Poll Enable
Bit 5:	Parallel Poll Individual Status
Bit 6:	Automatic IDY Sourcing in Idle Mode
Bit 7:	Oscillator Disable

Register 4: Loop Address Register.

Bit 0-4:	Address Bits
Bit 5-7:	Scratch Bits

Register 5, 6 and 7 are all scratch registers.

Table 23, HP-IL interface register structure

The HP-IL interface will respond to the following interrupt flag instructions:

Mnemonic	Hex	Meaning
?IFCR	16C	Set carry if interface ready
?SRQR	2AC	Set carry if the interface requests service
?FRAV	12C	Set carry if a frame is available from the loop
?FRNS	26C	Set carry if does not return as it was sent
?ORAV	0EC	Set carry if an output register is available

Table 24, HP-IL interface interrupt flag instructions

The HPIL=C r instruction copies C[1:0] to HP-IL register r, 0 < = r < = 7.

There are 8 different SELP instructions (one for each HP-IL register) that gives the HP-IL interface control of the HP-41 system. Control is given back to the HP-41 CPU when the least significant bit of an instruction is set.

The following possibilities exist:

SELP r, cccccccc01b	Place the binary constant cccccccc in HP-IL register r, 0 < = r < = 7.
SELP 0, 03Ah, 003h	Copy HP-IL register 0 to C[1:0].
SELP l, 07Ah, 043h	Copy HP-IL register 1 to C[1:0].
SELP 2, 0BAh, 083h	Copy HP-IL register 2 to C[1:0].
SELP 3, 0FAh, 0C3h	Copy HP-IL register 3 to C[1:0].
SELP 4, 13Ah, 103h	Copy HP-IL register 4 to C[1:0].
SELP 5, 17Ah, 143h	Copy HP-IL register 5 to C[1:0].
SELP 6, 1BAh, 183h	Copy HP-IL register 6 to C[1:0].
SELP 7, 1FAh, 1C3h	Copy HP-IL register 7 to C[1:0].

Table 25, HP-IL interface handling instructions

The 03Ah through 1FAh instructions are C=PREG r (0 < = r < = 7) instructions and copy the contents of HP-IL register (peripheral register) r to C[1:0].

The 003h through 1C3h instructions are ?PFSET r (0 < = r < = 7) instructions and set carry if peripheral flag r is set.

Note that these instructions are pairs, both must be used and they must have the same parameter r. E.g. a C=PREG 3 and ?PFSET 3 must be preceded by SELP 3, and not by any other instruction.

The HP-IL interface is a system addressed device and will always address itself to ROM block address 7.

Section 12:

Developing your own ROM

This section describes how to build your own ROM. It will explain about function names, how to use some of the most useful HP-41 system subroutines and finally give a commented example of a small user developed ROM.

Function and program names

Each time you specify a function or program to be executed, you specify it by name. The HP-41 first checks if this is the name of a FOCAL program in main memory, then if the name appears in a peripheral unit and finally if it is the name of a built-in function.

When the HP-41 is looking for a function or FOCAL program in system memory, it checks the Function Address Table (FAT) of each system address block. Recall that each FAT entry indicated whether it referred to an M-code routine or a FOCAL program, and it contained the address of the first executable word.

Other HEPAX file types (like data and text files) are also stored in system memory, but since the HP-41 never needs to execute them, they can be stored in a different format. Therefore, the other HEPAX file types don't take up any FAT entries.

If the FAT entry refers to a FOCAL program, the HP-41 knows that there is a LBL at the given address. It is simple for the HP-41 to look at that address and the following to find the name of the program.

The format for M-code routine names is a little more complicated. Since the FAT entry points to the first executable instruction, the HP-41 must start here. It then looks backwards, word by word, to find the characters that make up the function name.

The function name is written using ROM character codes, described in section 11. If any special ROM characters are used (bit 8 of the character code set), you must add 40h to the character code instead of setting bit 8. I.e. use the character "a", character code l0lh, clear bit 8 and add 40h - the result is 041h. Add 80h to the character code of the last character of the function name. Function names may be up to 11 characters long, but function names longer than 7 characters should not be used - you can't execute these functions! These functions are seen as ROM names by the HP-41CX. You might want to start you own ROM with a header - this is shown in the example at the end of this section.

Let's take an example of how to code the function name:
You have an M-code routine called "SORT", with the first executable M-code instruction at address x440. It is first in the FAT. The FAT entry would be:

x002	004	Specifies M-code routine, starting at
x003	040	address x440.

The start of the routine would be:

x43C	094	Character code for "T" + 080h = 094h
x43D	012	Character code for "R" = 012h
x43E	00F	Character code for "O" = 00Fh
x43F	013	Character code for "S" = 013h
x440	iii	First executable instruction

Prompting

You can make your own functions prompt in two different ways. This is done by adding a constant to the two first characters of the function name.

The possibilities are:

000,x00	No prompt
100,100	Prompt for three digits (4 if the EEX key is pressed)
100,200	Prompt for ALPHA input (null input accepted)

Table 26, Function prompting

I.e. to make the SORT function above prompt for ALPHA input, the code should be:

x43C	194	Character code for "T" + 080h = 194h
x43D	012	Character code for "R" = 012h
x43E	20F	Character code for "O" + 200h = 20Fh
x43F	113	Character code for "S" + 100h = 113h
x440	iii	First executable instruction

Note that some literature en HP-41 M-code programming lists a long range of other prompting possibilities. There are more prompting possibilities, but they only work correctly when used in blocks 0 through 2. The above three prompts are the only prompts that can be used in the rest of system memory.

Non-programmable functions

A function can be made non-programmable, and directly executing (not **NULL**able). If you place a NOP as the first executable instruction, the function is non-programmable. If the first two executable words are NOPs, the function will be executed as soon as you press the key (you can't hold the key to **NULL** the function). Exit by executing 0098 and then jumping to 00F0.

Selected HP-41 system subroutines

The HP-41 operating system contains many useful subroutines that handle some of the more trivial housekeeping tasks. A number of these subroutines are given below. To use them, place any input data as specified and use an absolute XQ or GO to the address.

Display handling routines

07F6	Disable RAM and enable display.
0899	Makes the display blink once. The Operating system uses this to indicate an illegal keystroke (like if you press XEQ ALPHA ALPHA).
0952	Disable peripheral units, enable RAM (status registers).
10E0	Clears the display, identical with CLD.
2BF7	Flush the contents of the display left.
2C5D	Send an ASCII character to the display. The character code must be in and the display must be enabled.
2CF0	Enables the display and clears it.

Keyboard handling routines

0098 This routine resets the keyboard, i.e. it waits until the key that is down is released, and then waits a short while longer to make sure the key is released. This is called debounce, and ensures that the calling routine will only see one keystroke.
This routine is useful if your routine needs direct key entry, or you could use it at the end of your routine so that any key pressed during your routine will not be interpreted again.

0E50 Alternative key input routine. Will place the calculator in stand-by mode until a key is pressed, then it returns to the calling address with the keycode in N[2:1]. The key codes are shown below.

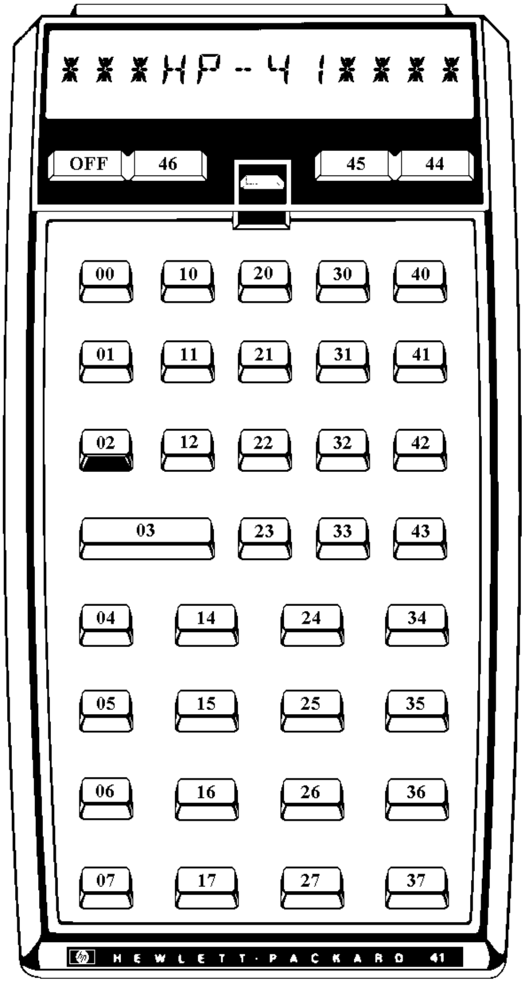


Fig. 23, Keycodes returned by 0E50 subroutine

Message routines

07EF Message routine. When calling, the display must be enabled and the desired message must be placed as constants in the words immediately after the XQ 07EF instruction. The constants must be ROM character codes, and the last character is indicated by adding 200h to the character code. A maximum of 12 characters is allowed.

When returning to the operating system, the message will be cleared, unless user flag 50 is set.

Example:

```
?NC XQ       Enable display and clear it.
->2CF0
?NC XQ       The following message will be displayed.
 ->07EF
008h "H"
005h "E"
00Ch "L"
00Ch "L"
20Fh "O"     200h is added to the last character.
?NC XQ       Flush the message left.
->2BF7
?NC XQ       Disable display and enable status registers.
->0952
```

1C0F Start of error message table.

22F5 This routine gives an error message as indicated by the constant following the call. The following combinations are available:

018	ALPHA DATA	(14E2)
022	DATA ERROR	(282D)
02D	MEMORY LOST	
038	NONEXISTENT	(02E0)
03C	NULL	
043	PRIVATE	(2184)
04F	OUT OF RANGE	(00A2)
056	PACKING	
05F	TRY AGAIN	(2F17)
062	YES	
064	NO	
067	RAM	(2172)
06A	ROM	(21F0)

Some messages are available directly with their own entry point, shown to the right of the message above. A call directly to an entry point takes up only two words, whereas a call via the 22F5 routine takes three words.

After this routine, the CPU returns to the operating system and checks the error ignore flags (user flag 25). The CPU does not return to the calling routine.

Example:
To get the MEMORY LOST message, use the following code:

```
?NC XQ
- > 22F5
02Dh                          The MEMORY LOST constant.
```

ALPHA register handling routines

10Dl	Clears the ALPHA register, identical with the CLA function.
2D0E	Appends one character to the ALPHA register. The character code must be in the G register. A warning tone will sound if the ALPHA register is now full.
2D14	As above, but does not give any warning if the ALPHA register is full.

Main memory handling routines

0232	The start of the MEMORY LOST routine!
05A1	Number of free registers in main memory is returned to C[2:0].
2000	Pack main memory, key assignments and i/o area.

Return points

0000 The CPU always starts from this address, with carry set if it starts from calculator off.

00F0 This routine updates the display, checks all ROMs (e.g. checks timer for alarms) and places the calculator in stand-by mode. This address is placed on the return stack before any call to external ROMs. If your routine ends with a RTN instruction and hasn't changed the return stack, your routine will automatically return to this address.

27F3 When using the interrupt jump locations, always return to this address to continue checking the interrupt locations of the following ROMs. When returning via this routine, the contents of C[10:3] must be restored, i.e. the interrupt routine should save C[10:3] before doing anything, and restore this data before calling 27F3.

WARNING: If you are not quite certain how to use the interrupt jump locations, don't use them at all. They will very often result in MEMORY LOST.

Miscellaneous routines

00D7 Calling address is placed in C[6:3].

02E3 Takes the scientific notation number in the C register and convert it to a hexadecimal number in C[2:0]. If the number is larger than 999, the message NONEXISTENT is given, if the contents of C is alpha data, the message ALPHA DATA is given.

16DE Start of the TONE function. A tone number must be in the ST register.

1EF5 Toggle the shift flag. The display is not updated.

Using port dependent jumps

A port dependent jump is actually a call to a system subroutine, followed by a constant. The system subroutine called tells the HP-41 if you want a GO or an XQ instruction, and which quarter of the block you wish to GO to or XQ. There are also two system subroutines for port dependent jumps within the same quarter of the current block.

Note that the CPU must be in hexadecimal mode (SETHEX) and that all the system subroutines for port dependent jump and execute overwrite the previous contents of the C register.

All ten system subroutines for port dependent jump and execute instructions are of the "if not carry" type. If you need to jump or execute if carry, you should use a relative jump to skip the subroutine call.

Example: If flag 10 is set, you need to GO to address xDF7 in the same block. Use the following code:

?FSET 10	Set carry if CPU flag set
JNC +04	Jump four addresses forward if carry not set
?NC XQ	Call subroutine for port dependent GO to last quarter.
-> 23EB	
1F7h	Data for the subroutine.
iii	Following instructions.

Which subroutine call to use is shown in the below table.

	GO	XQ
1. quarter (x000-x3FF)	23D0	23D2
2. quarter (x400-x7FF)	23D9	23DB
3. quarter (x800-xB99)	23E2	23E4
4. quarter (xC00-xFFF)	23EB	23ED
Same quarter	0FD9	0FDD

Table 27, Subroutine addresses for port dependent jumps

Example of a user-developed ROM

Now that we know all about HP-41 M-code programming, it's about time we start writing some of our own functions in M-code.

Our first ROM will contain a ROM name and two simple functions.

The first function will be called "Y<> Z" and will swap the contents of stack registers Y and Z. The second function will be called "X-ROM" and will write a word anywhere in HEPAX memory. Input for "X-ROM" will be a word of the form aaaaccc right justified in the X register. aaaa is the address and ccc is the code to be written.

Before we start writing our code, we must take a block out of the HEPAX file system. We'll refer to this block as "x". Remember that this must be the last block of HEPAX memory.

Keystrokes:	Display:	
XEQ HEXEDIT	ADR: _ _ _ _	Start the editor
xFF3	xFF3 100 _ _ _	The block is in the file system.
300	xFF4 000 _ _ _	Place 300h to take the block out of the file system.
<-	ADR: _ _ _ _	
xFE7	xFE7 00E _ _ _	Clear xFE7 and xFE8.
000	xFE8 000 _ _ _	
000	xFE9 091 _ _ _	
<-	ADR: _ _ _ _	
<-	0.0000	Leave the editor.

Now press the ON key twice to turn the calculator off and back on. Block 'x' is no longer part of the file system.

x000	x000 00D _ _ _	The first word of the block.

Now enter the hexadecimal code shown in the second column. When all the code has been entered, use the DISASM function to produce a disassembled listing. The listing should be the same as the text in the third column below.

```
x001   003  3 FUNCTIONS        Three functions.
x002   000  FCT:MY OWN ROM     FAT entry for the ROM name.
x003   08D  ADR: x08D          Address of the ROM name.
x004   000  FCT:Y<>Z           The FAT entry for our "Y
                               exchange with Z" function.
x005   092  ADR: x092          The start address of the
                               Y<>Z function.
x006   000  FCT: X-ROM         The FAT entry far the "X to
                               ROM" function.
x007   09E  ADR: x09E          The start address of the
                               X-ROM function.
x008   000  NOP                Two NOP words to mark
x009   000  NOP                the end of the FAT.
.
.
```

```
x082  000 NOP
x083  08D "M"              The ROM name written in
x084  00F "O"              reverse order. Note that
x085  012 "R"              080h has been added to the
x086  020 " "              character code of the last
x087  00E "N"              character in the name. A ROM
x088  017 "W"              name must be longer than
x089  00F "O"              7 characters - add spaces if
x08A  020 " "              needed.
x08B  019 "Y"              A ROM name cannot be
x08C  00D "M"              executed - it returns
x08D  3E0 RTN              immediately.
x08E  09A "Z"              Name of next function
x08F  03E ">"              written in reverse.
x090  03C "<"              Note that 080h is added
x091  019 "Y"              to last character.
x092  0B8 READ 2(Y)        Read stack Y register to C.
x093  10E A=C ALL          Save in A register.
x094  078 READ 1(Z)        Read stack Z register to C.
x095  0A8 WRIT 2(Y)        Write in stack register Y.
x096  0AE A<>C ALL         Get previous Y contents.
x097  068 WRIT 1(Z)        Write in stack register Z.
x098  3E0 RTN              Return to operating system.
x099  08D "M"              Name of next function.
x09A  00F "O"
x09B  012 "R"
x09C  02D "-"
x09D  018 "X"
x09E  0F8 READ 3(X)        Read stack X register to C.
x09F  040 WRIT S&X         Write C to HEPAX memory.
x0A0  3E0 RTN              Return to operating system.
.
.
```

```
xFF4  000 NOP                  Don't change the interrupt
xFF5  000 NOP                  locations.
xFF6  000 NOP
xFF7  000 NOP
xFF8  000 NOP
xFF9  000 NOP
xFFA  000 NOP
.
.
xFFB  001 "A"
xFFC  031 "1"                  Revision 1A.
xFFD  012 "R"
xFFE  00D "M"                  ROM ID is "MR"
xFFF  000 CHKSUM=000 HEX       No checksum calculated.
```

Appendices

Appendix A:

Messages from the HEPAX module

This appendix lists all the messages given by the HEPAX module, and some that are related to the use of the HEPAX module.

Some messages are error messages and indicate that a function has not been completed due to an error. Other messages are status messages and are simply for your information. Status messages are marked with an *.

Message	Functions	Meaning
DATA ERROR	HEPDIRX	No entry has number 0.
	XFA X< >F XFA XTOA	Input > 255.
DUP FL NAME	HWRTFL	File name already in use.
FL NOT FOUND	HREADFL	No such file on mass storage.
FL TYPE ERR	WRTROM	File name already in use.
GTO xx SHORT*	HSAVEP	Cannot compile GTO jump.
H:DIR EMPTY*	HEPDIR	No files in the HEPAX file system.
H:DUP FL	HCRFLAS HCRFLD HREADFL HSAVEA HSAVEK HSAVEP	File name already in use.
H:DUP FL NAME	HRENAME	File name already in use.
H:END OF FL	HAPPCHR HAPPREC HARCLRC HDELREC HGETREC HGETRX HGETX HINSCHR HINSREC	You attempted to read, write or insert past the end of the file.

H:END OF FL (continued)	HSAVER HSAVERX HSAVEX HSEKPT HSEKPTA	You attempted to read, write or insert past the end of the file.
H:END OF REC	HSEKPT HSEKPTA	You attempted to place the pointer after the end of the record.
H:FAT FULL	HSAVEP	All entries in a block is used. Create a dummy data file and try again.
H:FL NOT FND	"HRESZFL" HAPPCHR HAPPREC HARCLRC HASROOM HCLFL HDELCHR HDELREC HFLSIZE HGETA HGETK HGETR HGETREC HGETRX HGETX HINSCHR HINSREC HPOSFL HPURFL HRCLPT HRCLPTA HRENAME HSAVER HSAVERX HSAVEX HSEC HSEKPT HSEKPTA HUNSEC HWRTFL PRIVATE	The specified file is not found, or there is no current file

H:FL SECURED	HAPPCHR	You have tried to change a secured file.
	HAPPREC	If you want to change it, you must
	HCLFL	first unsecure it with HUNSEC.
	HDELCHR	
	HDELREC	
	HINSCHR	
	HINSREC	
	HPURFL	
	HSAVEA	
	HSAVEK	
	HSAVEP	
	HSAVER	
	HSAVERX	
	HSAVEX	
	HWRTFL	
H:FL SIZE ERR	"HRESZFL"	Data would be lost if you resized the file to the specified size. Use a negative size in X to resize anyway.
H:FL TYPE ERR	"HRESZFL"	You have tried to use a file of the
	HAPPCHR	wrong type.
	HAPPREC	
	HARCLRC	
	HASROOM	
	HCLFL	
	HDELCHR	
	HDELREC	
	HGETA	
	HGETK	
	HGETR	
	HGETREC	
	HGETRX	
	HGETX	
	HINSCHR	
	HINSREC	
	HPOSFL	
	HRCLPT	
	HRCLPTA	
	HSAVER	
	HSAVERX	

H:FL TYPE ERR (continued)	HSAVEX HSEKPT HSEKPTA	You have tried to use a file of the wrong type.
H:KEYCODE ERR	XFA PASN	No key with the specified keycode exist.
H:NAME ERR	HCLFL HCRFLAS HCRFLD HGETA HGETK HRENAME HPURFL HREADFL HSAVEA HSAVEK HSAVEP HWRTFL	No filename is specified.
H:NO FILESYS	All HEPAX file system functions	There is no file system in any HEPAX module. See page 61.
H:NO HPIL	HREADFL HWRTFL READROM WRTROM	No HP-IL module is plugged in.
H:NO KEYS	HSAVEK	There are no key assignments to save.
H:NO ROOM	HCRFLAS HCRFLD HSAVEA HSAVEK HSAVEP	There is not room in the HEPAX file system for a file of the specified size.
NO LBL xx*	HSAVEP	There is no LBL xx in the saved program.

NONEXISTENT	HSAVEP	The specified program does not exist.
	XFA PCLPS	
	XFA PASN	The specified function does not exist.
	XFA CLRGX XFA REGMOVE XFA REGSWAP XFA X=NN? XFA X≠NN? XFA X < NN? XFA X< =NN? XFA X > NN? XFA X> =NN?	Some of the specified registers do not exist.
PACKING TRY AGAIN	XFA PSIZE	There is not room for the specified size.
NO DRIVE	HREADFL HWRTFL READROM WRTROM	No mass storage device is connected to the HP-IL
HEPAX ROM	HEXEDIT DISASM	You attempted to edit or disassemble the HEPAX ROM
ILL CONFIG	ON	You turned the calculator on with an illegal configuration.
H:REC TOO LNG	HAPPCHR HINSCHR	You attempted to create a record longer than 254 characters.
H:CHKSUM ERR	READROM	An error occurred when reading a ROM image from mass storage.
x:WRT PRTCTED*	RAMTOG	Block x is write protected.
x:NOT PRTCTED*	RAMTOG	Block x is not write protected.
x:NOT RAM	RAMTOG	Block x is not RAM.
x:RAM ERROR	RAMTOG	Block x is not HEPAX RAM.

Appendix B:

Function overview

This appendix gives an overview of all the file system functions and XFA functions in the HEPAX module. The necessary input parameters are given. To obtain more information about a given function, look it up at the page reference given in the function index inside the back cover.

If a function has several different possible inputs, the possibilities are shown on separate lines.

Function	Inputs:	
HAPPCHR	ALPHA: alpha characters	
HAPPREC	ALPHA: alpha data	
HARCLRC	(none)	
HASROOM	(none)	
HCLFL	ALPHA: file name	
HCRFLAS	X: file size	ALPHA: file name
HCRFLD	X: file size	ALPHA: file name
HDELCHR	(none)	
HDELREC	(none)	
HEPDIR	(none)	
HEPDIRX	X: file no.	
HEPROOM	(none)	
HFLSIZE	ALPHA: (empty)	
	ALPHA: file name	
HGETA	ALPHA: file name	
HGETK	ALPHA: file name	
HGETR	ALPHA: (empty)	
	ALPHA: data file name	
HGETREC	(none)	
HGETRX	X: bbb.eee control number	
HGETX	(none)	
HINSCHR	ALPHA: alpha characters	
HINSREC	ALPHA: alpha data	
HPOSFL	ALPHA: search string	
HPURFL	ALPHA: file name	
HRCLPT	(none)	

HRCLPTA	ALPHA: (empty)	
	ALPHA: file name	
HREADFL	ALPHA: file name	
	ALPHA: Mass Storage file name,HEPAX file name	
HRENAME	ALPHA: old file name,new file name	
"HRFSZFL"	X: new file size	ALPHA: file name
HSAVEA	ALPHA: file name	
HSAVEK	ALPHA: file name	
HSAVEP	ALPHA: file name	
	ALPHA:,file name	
	ALPHA: program name,file name	
HSAVER	ALPHA: data file name	
HSAVERX	X: bbb.eee control number	
HSAVEX	X: data value	
HSEC	ALPHA: file name	
HSEKPT	X: pointer value	
HSEKPTA	X: pointer value	ALPHA: file name
HUNSEC	ALPHA: file name	
HWRTFL	ALPHA: file name	
	ALPHA: HEPAX file name,Mass Storage file name	
PRIVATE	ALPHA: file name	
CLRAM	X: block no.	ALPHA: "OK"
CODE	ALPHA: String of hexadecimal characters	
COPYROM	X: destination block	Y: source block
DECODE	X: code to be decoded	
DECODYX	X: no. of digits	Y: code
DISASM	Input from keyboard	
HEPAX	Input from keyboard	
HEPAXA	Input from keyboard	
HEXEDIT	Input from keyboard	
HPROMPT	X: No. of digits	ALPHA: prompt string
RAMTOG	X: HEPAX RAM block no.	
READROM	X: bb.ee	ALPHA: file name
WRTROM	X: bb.ee	ALPHA: file name
XF	Input from keyboard	
XFA	Input from keyboard	
HEPAXA AND	X: code	Y: code
HEPAXA BCAT	(none)	
HEPAXA BCD-BIN	X: number	
HEPAXA BIN-BCD	X: code	
HEPAXA CTRAST	X: contrast value	

HEPAXA DELETE	X: 00bbbbeeeellll	
HEPAXA INSERT	X: 00bbbbeeeellll	
HEPAXA NOT	X: code	
HEPAXA OR	X: code	Y: code
HEPAXA ROTYX	X: number to rotate	Y: code
HEPAXA SHIFTYX	X: number to shift	Y: code
HEPAXA XOR	X: code	Y: code
HEPAXA X+Y	X: code	Y: code
HEPAXA X-$	X: code	
HEPAXA Y-X	X: code	Y: code
XFA ALENG	(none)	
XFA ANUM	ALPHA: string	
XFA AROT	no. of characters to rotate	
XFA ATOX	ALPHA: text	
XFA CLKEYS	(none)	
XFA CLRGX	X: bbb.eee	
XFA GETKEY	(none)	
XFA GETKEYX	X: tt.t wait time	
XFA PASN	X: keycode	ALPHA: program/function name
	X: keycode	ALPHA: (empty)
XFA PCLPS	(none)	
	ALPHA: program name	
XFA POSA	X: char. code/string	ALPHA: string
XFA PSIZE	X: new size	
XFA RCLFLAG	(none)	
XFA REGMOVE	sss.dddnnn	
XFA REGSWAP	sss.dddnnn	
XFA ΣREG?	(none)	
XFA SIZE?	(none)	
XFA STOFLAG	X: flag status	
	X: bb.ee flag numbers	Y: flag status
XFA X < > F	X: flag value	
XFA XTOA	X: character code	
XFA X = NN?	X: test data	Y: register number
XFA X ≠ NN?	X: test data	Y: register number
XFA X < NN?	X: test data	Y: register number
XFA X < = NN?	X: test data	Y: register number
XFA X > NN?	X: test data	Y: register number
XFA X > = NN?	X: test data	Y: register number

Appendix C:

Reference tables for M-code programming

This appendix gives the actual hexadecimal codes for the M-code instructions described in section 10.

When writing M-code programs, write them in assembly language using the mnemonics in section 10. Then translate the mnemonics into hex codes using the tables in this section.

Class 0 parameter instructions

Parameter	0	1	2	3	4	5	6	7
	(T)	(Z)	(Y)	(X)	(L)	(M)	(N)	(O)
Mnemonic								
CLRF	384	304	204	004	044	084	144	284
SETF	388	308	208	008	048	088	148	288
?FSET	38C	30C	20C	00C	04C	08C	14C	28C
PT=	39C	31C	21C	01C	05C	09C	15C	29C
?PT=	394	314	214	014	054	094	154	294
LD@PT-	010	050	090	0D0	110	150	190	1D0
RCR	***	33C	23C	03C	07C	0BC	17C	2BC
WRIT	028	068	0A8	0E8	128	168	1A8	1E8
READ	***	078	0B8	0F8	138	178	1B8	1F8
HPIL=C	200	240	280	2C0	300	340	380	3C0
SELP	024	064	0A4	0E4	124	164	1A4	1E4

Class 0 parameter instructions, continued

Parameter	8	9	10	11	12	13	14	15
	(P)	(Q)	(├)	(a)	(b)	(c)	(d)	(e)
Mnemonic								
CLRF	104	244	0C4	184	344	2C4	***	***
SETF	108	248	0C8	188	348	2C8	***	***
?FSET	10C	24C	0CC	18C	34C	2CC	***	***
PT=	11C	25C	0DC	19C	35C	2DC	***	***
?PT=	114	254	0D4	194	354	2D4	***	***
LD@PT-	210	250	290	2D0	310	350	390	3D0
RCR	13C	27C	0FC	1BC	37C	2FC	***	***
WRIT	228	268	2A8	2E8	328	368	3A8	3E8
READ	238	278	2B8	2F8	338	378	3B8	3F8
HPIL=C	***	***	***	***	***	***	***	***
SELP	224	264	2A4	2E4	324	364	3A4	3E4

Class 0 special instructions

?ALM	36C
?C RTN	360
?CRDR	32C
?EDAV	0AC
?FRAV	12C
?FRNS	26C
?IFCR	16C
?KEY	3CC
?LOWBAT	160
?NC RTN	3A0
?ORAV	0EC
?P=Q	120
?PBSY	3AC
?SERV	2EC
?SRQR	2AC
?WNDB	22C

Class 0 special instructions, continued

A=B=C=0	1A0	N=C ALL	070
C<>G @PT,+	0D8	NOP	000
C<>M ALL	1D8	POP ADR	1B0
C<>N ALL	0F0	POWOFF	060
C<>ST XP	3D8	PRPH SLCT	3F0
C=C AND A	3B0	PT=PT+1	3DC
C=C OR A	370	PT=PT-1	3D4
C=G @PT,+	098	PUSH ADR	170
C=KEY KY	220	RAM SLCT	270
C=M ALL	198	READ DATA	038
C=N ALL	0B0	ROM BLK	030
C=ST XP	398	RTN	3E0
CLRKEY	3C8	SETDEC	2A0
DSPOFF	2E0	SETHEX	260
DSPTOG	320	SLCT P	0A0
ENBANK1	100	SLCT Q	0E0
ENBANK2	180	ST<>T	2D8
ENBANK3	140	ST=0	3C4
ENBANK4	1C0	ST=C XP	358
FETCH S&X	330	ST=T	298
G=C @PT,+	058	T=ST	258
GOTO ADR	1E0	WPTOG	1F0
GOTO KEY	230	WRIT DATA	2F0
LDI S&X	130	WRIT S&X	040
M=C ALL	158	XQ->GO	020

Class 1 instructions

Refer to the “JUMP” program on page 127. This FOCAL program calculates all types of class 1 instructions.

Class 2 instructions

Field	ALL	M	S&X	MS	XS	@PT	PT<-	P-Q
Instruction								
A=0	00E	01A	006	01E	016	002	00A	012
B=0	02E	03A	026	03E	036	022	02A	032
C=0	04E	05A	046	05E	056	042	04A	052
A=C	10E	11A	106	11E	116	102	10A	112
C=B	0CE	0DA	0C6	0DE	0D6	0C2	0CA	0D2
B=A	08E	09A	086	09E	096	082	08A	092
A<>C	0AE	0BA	0A6	0BE	0B6	0A2	0AA	0B2
C<>B	0EE	0FA	0E6	0FE	0F6	0E2	0EA	0F2
A<>B	06E	07A	066	07E	076	062	06A	072
C=C+A	20E	21A	206	21E	216	202	20A	212
A=A+C	14E	15A	146	15E	156	142	14A	152
A=A+B	12E	13A	126	13E	136	122	12A	132
C=C+C	1EE	1FA	1E6	1FE	1F6	1E2	1EA	1F2
C=A-C	24E	25A	246	25E	256	242	24A	252
A=A-C	1CE	1DA	1C6	1DE	1D6	1C2	1CA	1D2
A=A-B	18E	19A	186	19E	196	182	18A	192
C=C+1	22E	23A	226	23E	236	222	22A	232
A=A+1	16E	17A	166	17E	176	162	16A	172
C=C-1	26E	27A	266	27E	276	262	26A	272
A=A-1	1AE	1BA	1A6	1BE	1B6	1A2	1AA	1B2
?C≠0	2EE	2FA	2E6	2FE	2F6	2E2	2EA	2F2
?A≠0	34E	35A	346	35E	356	342	34A	352
?B≠0	2CE	2DA	2C6	2DE	2D6	2C2	2CA	2D2
?A≠C	36E	37A	366	37E	376	362	36A	372
?A<C	30E	31A	306	31E	316	302	30A	312
?A<B	32E	33A	326	33E	336	322	32A	332
RSHFC	3CE	3DA	3C6	3DE	3D6	3C2	3CA	3D2
RSHFA	38E	39A	386	39E	396	382	38A	392
RSHFB	3AE	3BA	3A6	3BE	3B6	3A2	3AA	3B2
LSHFA	3EE	3FA	3E6	3FE	3F6	3E2	3EA	3F2
C=0-C	28E	29A	286	29E	296	282	28A	292
C=-C-1	2AE	2BA	2A6	2BE	2B6	2A2	2AA	2B2

Class 3 instructions

Type	JNC+	JC+	JNC-	JC-
Distance				
01	00B	00F	3FB	3FF
02	013	017	3F3	3F7
03	01B	01F	3EB	3EF
04	023	027	3E3	3E7
05	02B	02F	3DB	3DF
06	033	037	3D3	3D7
07	03B	03F	3CB	3CF
08	043	047	3C3	3C7
09	04B	04F	3BB	3BF
0A	053	057	3B3	3B7
0B	05B	05F	3AB	3AF
0C	063	067	3A3	3A7
0D	06B	06F	39B	39F
0E	073	077	393	397
0F	07B	07F	38B	38F
10	083	087	383	387
11	08B	08F	37B	37F
12	093	097	373	377
13	09B	09F	36B	36F
14	0A3	0A7	363	367
15	0AB	0AF	35B	35F
16	0B3	0B7	353	357
17	0BB	0BF	34B	34F
18	0C3	0C7	343	347
19	0CB	0CF	33B	33F
1A	0D3	097	333	337
1B	0DB	0DF	32B	32F
1C	0E3	0E7	323	327
1D	0EB	0EF	31B	31F
1E	0F3	0F7	313	317
1F	0FB	0FF	30B	30F

Class 3 instructions, continued

Type	JNC+	JC+	JNC-	JC-
Distance				
20	103	107	303	307
21	10B	10F	2FB	2FF
22	113	117	2F3	2F7
23	11B	11F	2EB	2EF
24	123	127	2E3	2E7
25	12B	12F	2DB	2DF
26	133	137	2D3	2D7
27	13B	13F	2CB	2CF
28	143	147	2C3	2C7
29	14B	14F	2BB	2BF
2A	153	157	2B3	2B7
2B	15B	15F	2AB	2AF
2C	163	167	2A3	2A7
2D	16B	16F	29B	29F
2E	173	177	293	297
2F	17B	17F	28B	28F
30	183	187	283	287
31	18B	18F	27B	27F
32	193	197	273	277
33	19B	19F	26B	26F
34	1A3	1A7	263	267
35	1AB	1AF	25B	25F
36	1B3	1B7	253	257
37	1BB	1BF	24B	24F
38	1C3	1C7	243	247
39	1CB	1CF	23B	23F
3A	1D3	197	233	237
3B	1DB	1DF	22B	22F
3C	1E3	1E7	223	227
3D	1EB	1EF	21B	21F
3E	1F3	1F7	213	217
3F	1FB	1FF	20B	20F
40	***	***	203	207

Appendix D:

Hexadecimal and Binary numbers

This appendix gives a short explanation about hexadecimal and binary numbers. For a more complete explanation, consult a textbook on computer programming.

In both decimal, binary and hexadecimal number systems, each digit has a value and a "weight". In the decimal system, the rightmost (least significant) digit has the weight 1, the next digit has the weight 10, the next 100 and so on.

In the hexadecimal number system we have 16 digits. The first 10 are the same as in the decimal system, but then we have to start on the letters. Thus, the hexadecimal digits are 0-9, A, B, C, D, E and F. The least significant digit also has the weight 1, but the next digit has the weight 16, the next 256, and so on - multiplying by 16 for each position.

In the binary system there are only two digits: 0 and 1. The least significant digit has the weight 1, the next has the weight 2, the next has the weight 4 and so on - we multiply by 2 for each position.

We can write the values up to 16 with one or two decimal digits, one hexadecimal digit or four binary digits (bit).

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Table 28, Decimal, Hexadecimal and Binary numbers

To convert a hexadecimal number to decimal, we take it digit by digit. We multiply the weight of the digit with the value, and sum al1 these products. For example, the hexadecimal number 3AE is 3 x 256 + 10 x 76 + 14 = 942.

To convert a decimal number to hexadecimal, we divide by decreasing multiples of 16. For example, the decimal number 3572 divided by 256 = 13,95. This means that the third hexadecimal digit (with the weight 256) is decimal 13, the hex digit "D". We calculate the remainder 3572 - 13 x 256 = 244. 244 / 16 = 15.25, thus the second hex digit (with the weight 16) is decimal 15, hex "F'. The remainder is decimal 4, the same as the hexadecimal digit "4". Thus, the hexadecimal equivalent of 3572 decimal is the hexadecimal number DF4.

To convert a binary number to decimal, we also take it digit by digit. We multiply the weight of the digit with the value, and sum all these products.
For example, the binary number 101101 is 1 x 32 + 0 x 16 + 1 x 8 + 1 x 4 + 0 x 2 + 1 = 45.

To convert a decimal number to binary, we simply subtract decreasing multiples of two. To convert the decimal number 145 to binary, we subtract 128, giving the remainder 17. Since 128 is 2 x 2 x 2 x 2 x 2 x 2 x 2, we have the 7th binary digit is 1. We can now subtract 16, and since 16 is 2 x 2 x 2 x 2, the 4th bit is also 1. The remainder is 1, giving the 1st bit to be 1. Altogether, the binary equivalent of 145 is 10010001.

To convert between hexadecimal and binary numbers, simply convert one hexadecimal digit to 4 bits or vice versa, using table 28 above. For example, to convert 4EB7 to binary, simply look up digit by digit to find 0100 1110 1011 0111. To convert 1011010110 to hexadecimal, group the bits in fours from the right end like this: 10 1101 0110. Each group is then one hex digit, in this example, we find 1011010110 to be 2D6 hex.

Appendix E:
XROM numbers

This appendix gives the XROM number of all the functions in the HEPAX module. It also gives the XROM ID no. of all external ROMs available at the time of printing of this manual. Note that the HEPAX file system automatically allocates an unused XROM ID no. to each block of HEPAX memory, starting with 11d.

Some XROM ID numbers are used by two modules or more modules. Only one of these may be plugged into the HP-41 at a time. Modules with two XROM numbers are normally of the "8K" type.

XROM no.	Module
01	Math Pac
02	Statistics Pac
03	Surveying Pac
04	Financial Analysis Pac
05	Standard Pac
06	Circuit Analysis
07	Structural (A)
07	HEPAX module
08	Stress Analysis
09	Home Management
10	Games
10	PPC ROM
10	Auto/Duplication ROM
11	Real Estate
12	Machine Design
13	Thermal and Trans.
14	Navigation Pac
15	Petroleum Fluids
16	Petroleum Fluids
17	Plotter ROM
18	Plotter ROM
19	Aviation
19	Clinical Lab.
19	Securities
19	Structural(B)
20	PPC ROM
21	Custom 8K

21	Assembler 3
22	HP-IL Development ROM
23	Extended I/O
24	HP-IL Development ROM
25	Extended Functions
26	Time module
27	Optical Wand
28	HP-IL Control and Mass Storage
29	Printer
30	Card Reader
31	Custom 4K and 8K

The XROM numbers of the functions in the HEPAX module are:

07,00	-HEPAX ID	07,20	HINSCHR	07,40	CLRAM
07,01	HAPPCHR	07,21	HINSREC	07,41	CODE
07,02	HAPPREC	07,22	HPOSFL	07,42	COPYROM
07,03	HARCLRC	07,23	HPURFL	07,43	DECODE
07,04	HASROOM	07,24	HRCLPT	07,44	DECODYX
07,05	HCLFL	07,25	HRCLPTA	07,45	DISASM
07,06	HCRFLAS	07,26	HREADFL	07,46	HEPAX
07,07	HCRFLD	07,27	HRENAME	07,47	HEPAXA
07,08	HDELCHR	07,28	HSAVEA	07,48	HEXEDIT
07,09	HDELREC	07,29	HSAVEK	07,49	HPROMPT
07,10	HEPDIR	07,30	HSAVEP	07,50	RAMTOG
07,11	HEPDIRX	07,31	HSAVER	07,51	READROM
07,12	HEPROOM	07,32	HSAVERX	07,52	WRTROM
07,13	HFLSIZE	07,33	HSAVEX	07,53	XF
07,14	HGETA	07,34	HSEC	07,54	XFA
07,15	HGETK	07,35	HSEKPT		
07,16	HGETR	07,36	HSEKPTA		
07,17	HGETREC	07,37	HUNSEC		
07,18	HGETRX	07,38	HWRTFL		
07,19	HGETX	07,39	PRIVATE		

Subject index

Function index