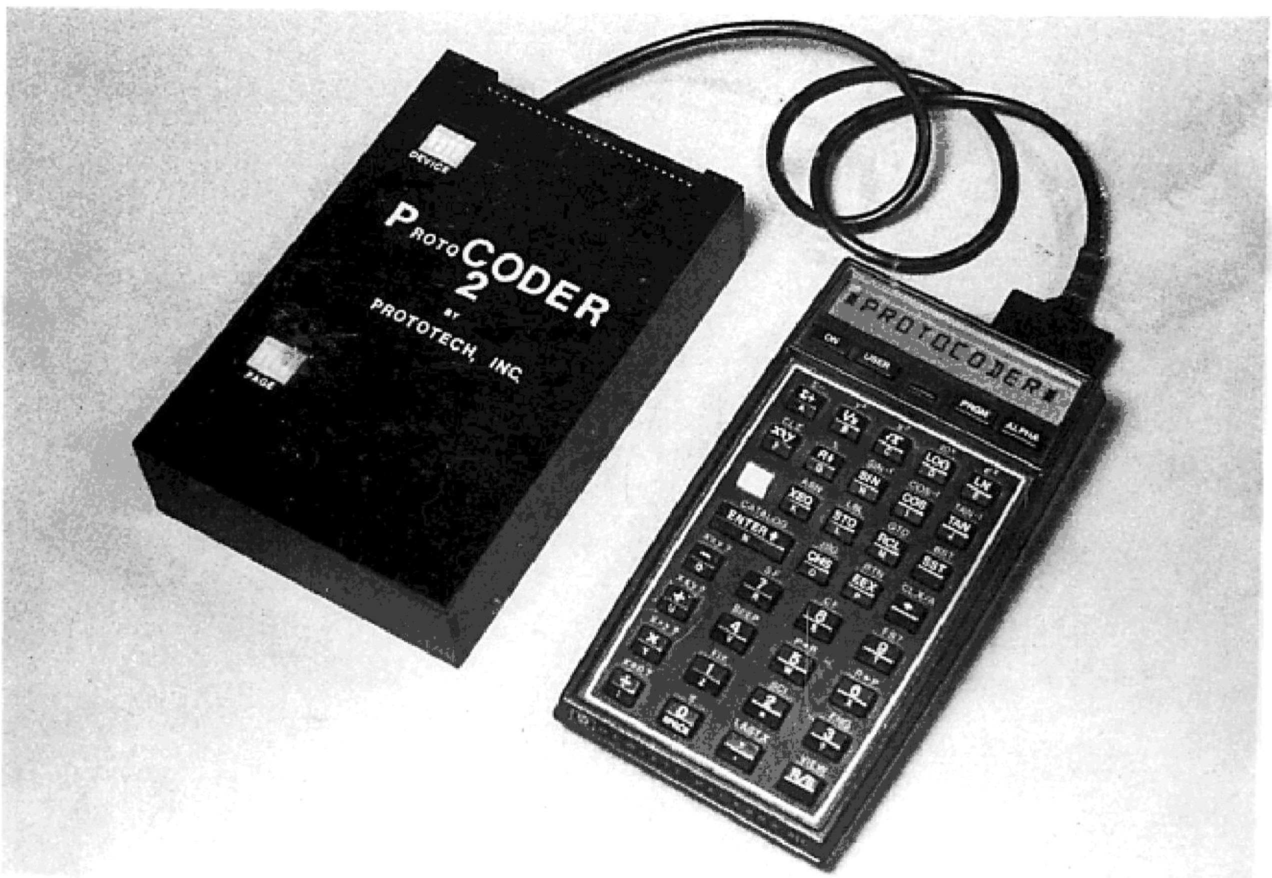
PROTOCODER2

OWNER'S MANUAL



PROTOTECH, INC.

P. O. BOX 12104

BOULDER, CO. 80303 USA

(303) 499-5541

ProtoCODER2 OWNERS MANUAL

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

ProtoSYSTEM OVERVIEW

The Prototech, Inc. ProtoSYSTEM is a flexible and expandable interface between the HP 41C calculator and various peripheral and memory devices. The modular design of the ProtoSYSTEM allows the user to expand his system as his needs increase.

The ProtoCODER2 is the initial device required to allow the user to add on any of the peripheral boards. The ProtoCODER2 provides the user with 4096 words (4K) of user-alterable memory which is addressed as an HP 41C ROM, thus allowing the user to write programs in the calculator's assembly language ("Microcode"), as well as RPN programs that are too large to fit in user memory. It plugs directly into any of the ports of the calculator. The user can obtain additional blocks of 4K of user-alterable memory by connecting additional ProtoCODER boards, or:
----> Plug in HP Application modules and switch them on or off from the calculator under program control (ProtoROM),
----> Plug in HP-format EPROM sets containing custom programs - like having your own custom ROM made but much less expensive (ProtoEPROM), or
----> Interface to any 5-volt level device such as a full-size keyboard, a light or power controller, another calculator or a computer (ProtoPARIO).

The ProtoCODER2 provides control, address, and data signals for all peripheral boards. It also contains a battery to maintain the contents of the internal memory. ProtoSYSTEM boards are programmed from the keyboard or under program control by a sequence of two operations:
1) Create a Non-Normalized Number (NNN) in X which contains the appropriate programming code, then
2) Execute ABS.
The ProtoCODER2 will examine the contents of the X register and use parts of it to determine which ProtoSYSTEM board to program and the data to be programmed.

CONNECTING THE ProtoCODER2

The ProtoCODER2 cable plugs directly into any of the four ports of the HP 41C. Before connecting the ProtoCODER2 to the HP 41C, turn off the calculator. If you do not, you may damage the calculator or the ProtoCODER2. Gently insert the plug with the flat surface upwards. To remove, pull the plug straight away from the calculator.

When the ProtoCODER2 is connected to the HP 41C, it uses the battery in the calculator. To retain the contents of the internal ProtoCODER memory, make sure that the battery is installed in the ProtoCODER2 when you remove it from the calculator.

CONNECTING PERIPHERAL BOARDS

The ProtoCODER2 has a 25-pin bus connector that passes signals between the ProtoSYSTEM peripheral boards. To plug a board onto the ProtoCODER2, check that all pins are straight then line up the pins into the socket. Press down on the left side of the 25-pin connector until some of the pins go into the lower socket. Then press the remaining pins into the socket.

POWER SUPPLY

The ProtoCODER2 contains a 3-volt battery (Duracell PX-30 or Eveready EPX-30) to maintain internal memory. To install a new battery, remove the screw in the side of the ProtoCODER2 box and open the box. Slide the battery out from between the clips and replace with a new one (with the + upwards). To insure that you do not lose data in the internal memory, it is a good idea to have the ProtoCODER2 plugged in to the calculator as you change the battery. The battery should be replaced when it reaches 2 volts. The battery will last several months if the ProtoCODER2 is attached to the calculator most of the time.

CHAPTER 2: ProtoCODER2 PROGRAMMING

DEVICE SELECTING

The ProtoCODER2 has its own addressing system so that several boards of the same type can be connected simultaneously and used independently. For example, two (or more) ProtoROM boards can be used at the same time by giving them different device select addresses which are set by switches on each board. Up to 16 devices (but only 4 ProtoROMs) can be addressed directly using hexadecimal addresses 0-F. Each address specifies a different board. The device select information is specified in the X register by the user during board programming.

CREATING THE PROGRAMMING DATA

The user must set up data in the X register to program the ProtoSYSTEM peripheral boards. This data depends on the board to be programmed. To program any ProtoSYSTEM device, bits 55,54,3,2 of the X register are always 1 and bits 5,4,1,0 contain the device select code (bits 5,4 specify the slot for the ProtoROM). The remaining bits contain the data required by the board to be programmed.

The X register and all calculator data registers consist of 56 "bits" of information. Each bit can be either 1 or 0 (on or off, set or clear). To save space when writing this data, these 56 bits are gro080d into 14 blocks of 4 bits each, called nybbles or digits. Each nybble is represented by a hexadecimal ("hex") digit (0-9,A,B,C,D,E,F). The following table lists hex digits and their binary equivalents.

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

By convention the bits are numbered from 55 (leftmost or high order) to 0 (rightmost or low order). The digits or nybbles are numbered from 13 to 0.
EXAMPLE: X contains 02 9F EA 7B 14 6D 35 hex (spaces are for clarity) which in binary is:
0000 0010 1001 1111 1110 1010 0111 1011 0001 0100 0110 1101 0011 0101
Nybble 9 is E in hex which means that bits 39, 38 and 37 are 1 (on) and bit 36 is 0 (off).

WRITING DATA TO THE ProtoSYSTEM

To program any ProtoSYSTEM board, consult the appropriate section in this manual to determine the 14 hex nybbles for that board and enter in ALPHA as a string of hex digits (0-F). Then convert this data to binary in the X register by executing CODE in the PCODER-1A EPROM set. Various other versions of CODE are available and will also work correctly.

After CODEing this data into the X register, execute ABS to write it to the correct ProtoSYSTEM board. The ProtoCODER2 monitors the ISA line of the calculator and when it sees ABS executed it picks apart the X register and uses what is needed to program the appropriate board.

This sequence can be executed from the keyboard or from a program and requires no external EPROM to program the system; however, the PCODER-1A is very useful in aiding programming of the ProtoCODER2.

The ProtoCODER2 can also be programmed from microcode by setting up the data as above in the internal C register (bits 55 and 54 are ignored), then performing a GOSUB 1078. 1078 is the hex address of the RTN in the ABS function, therefore each ProtoCODER2 write operation requires only 3 instruction cycle times (about .5 milliseconds).

INITIAL SETUP

As shipped, the ProtoCODER2 contains a dummy catalog containing the functions *PROTOCODER* and *END*. You can plug in the ProtoCODER2 and run a CAT 2 to verify that it is there.

When you change the catalog linkage table in the ProtoCODER2, you may need to set the page select switches to 3 hex, since the contents of page 3 is ignored by the HP 41 operating system (use page 2 for the HP 41CX). If you lose power, and the contents of the ProtoCODER2 becomes garbage, set the page select switches to 3 (or 2) when loading a ROM image, and remember to clear words FF4-FFA (see "XROM Word Format"). In setting up a ROM image, it is good practice to leave 50-100 words unused at the beginning of the ROM for catalog space to be used later as you add in additional functions. Unused words other than in the catalog linkage table and the interrupt locations need not be cleared.

SETTING THE SELECT SWITCHES

Each ProtoCODER has two sets of four switches. The set nearest the 25-pin connector specifies the device select address (0-F hex) of the board. Determine an address that is not used by any other ProtoSYSTEM board, convert to binary, and set the switches left to right accordingly. Usually these switches will all be set to 0 (off) since support and programming functions in the PCODER-1A EPROM set program the device with address 0.

The other set of switches specifies in which page (0-F hex) the ProtoCODER will be addressed in the HP 41C memory (See "ROM Addressing").

PROGRAMMING THE ProtoCODER

The first step in programming the ProtoCODER is to have a list of code to be entered, for example the following function, CLY. Unless you are using the interrupt locations (addresses FF4-FFA), they must all be set to zero. Instructions can be loaded manually or by using LODE in PCODER-1A. Instructions are normally loaded one at a time; however, you can load and save blocks of instructions by using BOOT and DUMP in PCODER-1A. This allows you to save microcode on a cassette tape or any other medium you choose.

The data word format contained in X for the ProtoCODER is:

dd dx xx xx aa ax ss

"d"-contains the 10 bits of data plus hex C00

"x"-ignored - hex F is easiest to enter

"a"-address where data is to be written (hex 000-FFF)

"s"-device select address of the ProtoCODER. Bits are coded as: xxss 11ss.

EXAMPLE: You have written a routine called CLY which clears the Y register to zero, and you wish to copy it into the ProtoCODER. The listing is:

ADDRESS	DATA WORD	MNEMONIC
x100	04E	C=0 ALL
x101	0A8	REGN=C 2 (Y)
x102	3E0	RTN

Set the device select switches to 0 then set up the X register as C4 EF FF FF 10 0F 0C by entering "C4EFFFFF100F0C" in ALPHA and executing CODE. "C4E" is the C=0 instruction (04E) plus hex C00; x100 is the address which should contain the C=0 instruction; the final 0C is the device select address and the enable bits. Now execute ABS. This will write one word to the ProtoCODER. Repeat the above with "CA8FFFFF101F0C" then "FE0FFFFF102F0C" to finish loading CLY. Set up the FAT (Function Address Table) - see "XROM Word Format" and then try executing CLY.

USING THE ProtoCODER AFTER PROGRAMMING

After programming a valid ROM image into the ProtoCODER, it will function without user intervention. It will appear to be an HP module to the calculator. If the ROM image is not correct then crashes or unpredictable results may occur (will occur! Murphy hides in all ProtoCODERs). To have a correct ROM image, the FAT table at the beginning of the ROM (starting at address 000) must be set up correctly and the interrupts (addresses FF4-FFA) must be zero (or used very carefully).

CHAPTER 3: ProtoROM INSTRUCTIONS

ProtoROM PURPOSE

The ProtoROM expands the number of available ports for the user to plug in HP Application Modules. Each ProtoROM attached allows the user to plug in four additional HP modules. Each module can be switched on to or off from any port of the calculator under program or keyboard control. The HP 41C can have at most four Application Modules on-line simultaneously, but the ProtoROM will allow the user to have more modules plugged in and switched on only when they are needed.

PLUGGING IN HP MODULES

To insert a module, turn off the calculator then place the module into one of the slots in the ProtoROM with the printing on the module upright and the ProtoROM printing upwards. Gently push the module in until the extended module handle is flush with the ProtoROM box. Do not force it. To remove, grasp the module handle with your fingernail or a small knife and pull straight out.

SETTING THE SELECT SWITCHES

Each ProtoROM has a set of four switches so that the ProtoSYSTEM can tell the boards apart when more than one is connected. The switches set the device select address (see Chapter 2). To set these switches, determine an unused device select address (0-3 in hex) then convert to binary and set the switches left to right according to the binary address (off for 0, on for 1). Older ProtoROMs use all four switches. ProtoROMs for use with the ProtoCODER2 ignore the leftmost two switches.

PROGRAMMING THE ProtoROM

Before programming the ProtoROM, reread Chapter 2. The format of the data word to be created in the X register is (in hex):

Cp xx xx xx xx xx ms

"C" is hex C

"p" is the port and on/off code:

p=0 to turn module off
p=1 to turn module on in port 1
p=3 to turn module on in port 2
p=5 to turn module on in port 3
p=7 to turn module on in port 4

"x" can be anything - hex F is easiest to enter

"m" tells which slot of the ProtoROM to alter. Slots are numbered from 3 (nearest the 25-pin connector) to 0

"s" is the device select address which can be C (device = 0), D (for 1), E (for 2), or F (for 3). The ProtoROM only accepts device selects of 0-3. Older ProtoROMs need a slight modification to work with the ProtoCODER2 - write to Prototech, Inc. for details.

The ProtoROM will retain its programming until it is reprogrammed or until the ProtoCODER2 battery is removed.

CHAPTER 4: ProtoEPROM INSTRUCTIONS

ProtoEPROM PURPOSE

The ProtoEPROM allows the user to plug in HP-format EPROM sets to the HP 41C and use them just like HP modules. EPROM sets may contain user language programs, assembly language (microcode) routines, and/or data tables. EPROMS provide an inexpensive means for the user to have his programs available without using HP 41C memory, and allow the user complete control over the calculator with microcode. Each EPROM set costs approximately $13 (4K) compared to about twice as much for an HP module or custom ROM. In addition, there is no setup charge for EPROMs (several thousand dollars for HP custom ROMs) - you just set up a file on a EPROM burner.

PLUGGING IN EPROMS

The ProtoEPROM will accept one standard HP-format EPROM set which consists of 2 or 3 EPROM chips. To insert or remove EPROMs, push the lever on the side of each of the 3 EPROM sockets towards the 25-pin connector. This releases the chips, and you can remove or change the chips in the sockets. Place the chips in the sockets from right to left: U, L1, L2. The notch in the end of the chip must point towards the 25-pin connector. When placing a 24-pin EPROM in L1 or L2, leave the top two rows in the socket empty. When the chips are flat in the sockets, pull the socket levers away from the 25-pin connector until the chips are locked into the sockets.

SETTING THE SELECT SWITCHES

Determine which page to have the EPROM addressed (See "ROM Addressing"). Convert this hex page number to binary then set switches 4-5-6-7 on the ProtoEPROM to correspond to this address. For example to address the EPROM in page E, set switches 4-5-6 on and switch 7 off. 8K EPROMs occupy two consecutive pages (4-5, 6-7, 8-9, A-B, C-D, E-F) and 16K sets occupy four consecutive pages (4-5-6-7, 8-9-A-B, C-D-E-F). For 8K or 16K EPROM sets, set switches 4-7 to the address of the lowest page of the block.

After setting the address select switches, you need to tell the ProtoEPROM what type and size EPROMs you are using. Switches 1-3 do this. Set them as:

SW1	SW2	SW3	size	#chips
OFF	ON	ON	4K	2
OFF	OFF	ON	8K	2
ON	OFF	ON	8K	3
ON	OFF	OFF	16K	3

Example: To use the 4K 2 chip PCODER-1A EPROM set in page E, set switches 1-7 to 0111110 (off-on-on-on-on-on-off). To use the 8K 3 chip NFCROM-1B/IDEAL EPROM set, set switches 1-7 to 1011110 (on-off-on-on-on-on-off).

CHAPTER 5: ProtoPARIO INSTRUCTIONS

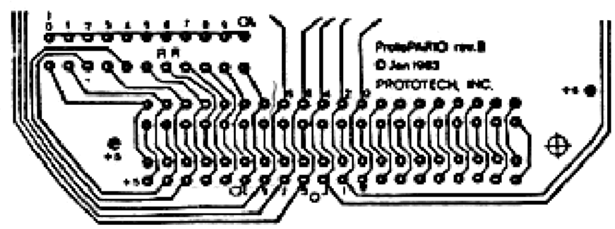
ProtoPARIO PURPOSE

The ProtoPARIO allows the HP 41C user to interface to and from almost any 5 volt device, providing 10 input lines, 10 output lines, and 2 output handshake lines. It attaches to the ProtoEPROM/ProtoCODER2 combination, and looks like an 8K ROM to the calculator. The lower 4K is occupied by the PARIO-1A or some other EPROM set, and the upper 4K is interpreted as Input/Output signals by the ProtoPARIO. Although the PARIO-1A EPROM set is not mandatory, it greatly simplifies programming and automates use of the ProtoPARIO.

PHYSICAL CONNECTIONS

The three IC sockets on the ProtoPARIO attach directly to the ProtoEPROM board. Flip the EPROM socket levers forward and line up the pins on the ProtoPARIO sockets into the ProtoEPROM sockets, then reach between the boards with a pencil or small knife and pull the levers back to lock the ProtoPARIO in place. In addition to this, the free wire on the ProtoPARIO must be connected to V+ in the ProtoSYSTEM. V+ is available as the leftmost hole in the 25 pin connector. The 4K controlling EPROM is plugged into the rightmost 2 sockets in the ProtoPARIO, oriented the same as for the ProtoEPROM. Set the ProtoEPROM select switches 1-2-3 to ON-OFF-ON, and set switches 4-5-6-7 to an even page.

As shown below, two complete sets of data lines are available at the bottom of the ProtoPARIO. Each set is organised as 2 rows of holes centered .1" apart for standard connectors. The leftmost 12 holes of the upper row are (left to right) I0, I1, I3, I5, I7, I9, Output Accepted, O8, O6, O4, O2, and O0. The lower row (left to right) is V+, I2, I4, I6, I8, Output Ready, O9, O7, O5, O3, O1, and GND. All inputs (I9-I0 and OA) are pulled to GND with 100K resistors if unused. With two complete sets of I/O lines, it is possible for the user to hook up two devices or to customize the connector by using the unused pads to the right.



ELECTRICAL CHARACTERISTICS

Both inputs and outputs are buffered to minimize possible damage to the ProtoSYSTEM or calculator from external signals (overvoltage, etc.). All inputs should be in the range 3.5-5.5 volts for ON and 0-1.5 volts for OFF, and will require at most 1 uamp drive current per data bit. Propagation delay time (and set/reset time for OA and OR) is at most 180 nsec. OR can source .5 mamp and the outputs O9-O0 can source 1.75 mamp each, both at 4.5-5.5 volts. Outputs are latched in two CMOS flipflop arrays: 74C174, 74C175. The OR signal is latched in a 4013, and can only be reset by asserting OA. The OR signal is provided for handshaking with devices that are faster than the HP 41C. It does not prevent subsequent outputs from the calculator from being accepted. Inputs are buffered through two 4503 (70C97) CMOS chips. All buffer chips are socketed for easy replacement. All specifications given above are for V+=5 volts and ambient temperature 25C.

The GND line should always be connected to the external device ground. The V+ output should only be used through passive components such as switches back to I9-I0 since it is the regulated power from the calculator which does not provide much current capability. No external signal should be connected to V+.

PROGRAMMING WITHOUT THE PARIO-1A EPROM SET

The PARIO-1A EPROM set provides various I/O functions for simplified use of the ProtoPARIO, but is not required. The ProtoPARIO is programmed by using the CXISA (FETCH - hex 330) microcode instruction. For the following discussion it is assumed that the ProtoEPROM board is addressed at page E which places the ProtoPARIO in page F.

In this arrangement, a fetch to addresses F000-F3FF or FC00-FFFF will return 000. A fetch to any address F800-FBFC which has the final digit 0, 4, 8, or C will return I9-I0 in the exponent field of the C register. A fetch to any address F801-FBFF which has the final digit 1-3, 5-7, 9-B, or D-F will return I7-I0 in digits 1-0 of C and 0 in digit 2.

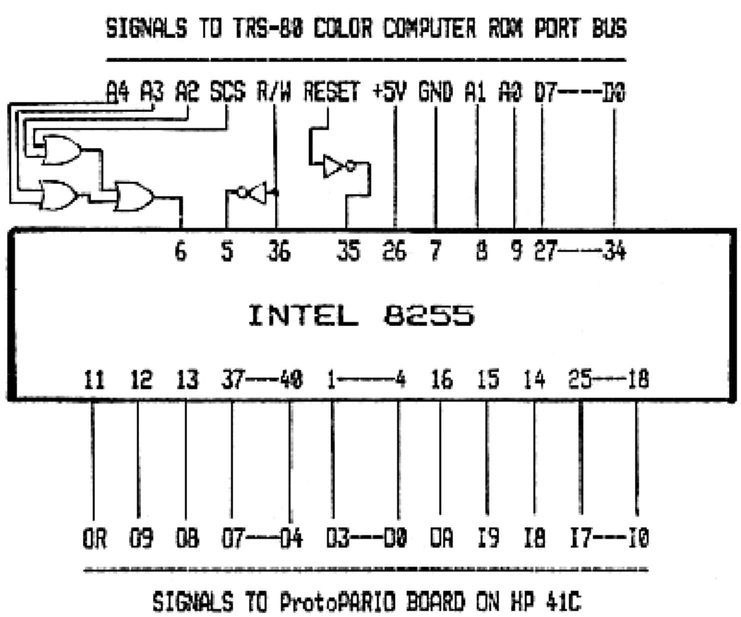
Outputs are generated by a fetch to addresses F400-F7FF. Add F400 to the 10-bit binary number to be output. For example, to output 000, do a fetch at F400. A fetch at F654 will output 254 hex. Whenever an output is received by the ProtoPARIO, a flipflop in the 4013 chip (available as Output Ready) is set. If OR was already set, the new output data overwrites what was previously output. OR can only be reset to 0 by asserting Output Accepted. This provides handshaking capabilities with external devices that are faster than the HP 41C - see the interface for the TRS-80 Color Computer below.

ProtoPARIO INTERFACE TO TRS-80 COLOR COMPUTER

The following circuit diagram illustrates a possible interface between the HP 41C and the Radio Shack Color Computer. The bus connections to the 74LS04, 74LS32 and Intel 8255 are all available at the ROM port on the side of the computer. Use the PARIO-1A EPROM for programming on the calculator side. To initialize the interface, POKE &HFF43,152. This will set up the A port and C7-C4 as outputs from the calculator, and the B port and C3-C0 as inputs to the calculator on the 8255. Input O7-O0 from the calculator to the computer with PEEK (&HFF40). Output I7-I0 to the calculator with POKE &HFF41,I where I is 0-255 decimal. O9, O8, and OR are available as C5, C4, and C6 with PEEK (&HFF42). I9, I8, and OA (C1, C0, C2) can be programmed by POKEing to &HFF42.

If you build this interface, test it carefully before attaching to the HP 41C. This circuit is presented as an example only: Prototech, Inc. assumes no responsibility for the accuracy or use of this information.

Parts required: 74LS32, 74LS04, Intel 8255, 3-.1 uf bypass capacitors: 1 per chip.



CHAPTER 6: PARIO-1A EPROM SET

This EPROM set provides the user with a variety of input and output functions to control the ProtoPARIO:

A-XB	converts the last 12 or less binary characters (0-1) in ALPHA into the exponent of X
A-XD	converts the last 4 or less decimal characters (0-9) in ALPHA into the exponent of X
A-XH	converts the last 3 or less hex characters (0-F) in ALPHA into the exponent of X
A-XO	converts the last 4 or less octal characters (0-7) in ALPHA into the exponent of X
ESCAPE	converts the next to last character in ALPHA to be >= hexcode 20
F-X	converts flags 11-0 into the exponent of X
FETCH	executes a CXISA instruction at address in digits 3-0 of X
GOKEY	uses ProtoPARIO inputs to specify keycode from external keyboard
LISTU2	lists upper 2 bits of any ROM page in EPROM format
PACK5	packs 5 10-bit blocks of data into X
PACK7	packs 7 8-bit blocks of data into X
READ	immediate read of I9-I0 into exponent of X
READ1	loops with wait inputting data into consecutive registers (1 input/reg)
READ5	loops with wait inputting data into consecutive registers (5 inputs/reg)
READ7	loops with wait inputting data into consecutive registers (7 inputs/reg)
READA	asynchronous looped read waits for non zero changed input (1 input/reg)
ROM	displays PARIO message (try it)
RVIEW	views first 3 digits of X, Y, Z, T
STK<>E	provides alternate stacks by exchanging with sum regs (no normalize)
U2	microcode subroutine used by LISTU2
UNPACK5	unpacks X into 5 registers each containing a 10-bit block of data
UNPACK7	unpacks X into 7 registers each containing an 8-bit block of data
WAITNZ	loop until I9-I0 is not zero then return input in exponent of X
WAITX	loop until I9-I0 matches exponent of X
WRIT	immediate write of O9-O0 from exponent of X
WRIT1	loop with wait writing O9-O0 from consecutive regs (1 output/reg)
WRIT5	loop with wait writing O9-O0 from consecutive regs (5 outputs/reg)
WRIT7	loop with wait writing O9-O0 from consecutive regs (7 outputs/reg)
WRITA	asynchronous write upon input equal to 0
X-AB	converts exponent of X to up to 12 binary digits in ALPHA
X-AD	converts exponent of X to up to 4 decimal digits in ALPHA
X-AH	converts exponent of X to up to 3 hexadecimal digits in ALPHA
X-AO	converts exponent of X to up to 4 octal digits in ALPHA
X-F	converts exponent of X to user flags 11-0
X<>REG	exchanges X with absolute user register specified by Y
XE-M	converts hex exponent of X to decimal mantissa of X
XM-E	converts decimal mantissa of X to hex exponent of X

CREATING DATA IN X FOR OUTPUT

The functions A-XB (binary), A-XD (decimal), A-XH (hexadecimal), A-XO (octal), F-X, and XM-E can be used to create data in the exponent of X. To use A-XB, A-XD, A-XH, or A-XO, set ALPHA to contain the number in the appropriate base to be put in the exponent of X:

"1010010" A-XB will set exponent of X to 052 hex
"1023" A-XD will set exponent of X to 3FF hex
"7C" A-XH will set exponent of X to 07C hex
"47" A-XO will set exponent of X to 027 hex

F-X converts flags 11-0 as a binary number into the exponent of X: if flags 11, 10, 9, 8, 6, 4, 3, 2 are clear and flags 7, 5, 1, 0 are set then F-X will set exponent of X to 0A3 hex. XM-E converts decimal number in mantissa of X to hex number in exponent of X: if X contains 64.0000 then XM-E will set exponent of X to 040 hex.

DECODING DATA IN X AFTER INPUT

The functions X-AB (binary), X-AD (decimal), X-AH (hexadecimal), X-AO (octal), X-F, and XE-M can be used to decode the hex data in the exponent of X. To use X-AB, X-AD, X-AH or X-AO, execute the function for the appropriate base and the exponent of X will be returned in ALPHA in that base. If the exponent of X contains hex 0FD:

X-AB will return "11111101" in ALPHA
X-AD will return "253" in ALPHA
X-AH will return "FD" in ALPHA
X-AO will return "375" in ALPHA
X-F will set flags 7-2 and 0 and clear flags 11-8 and 1
XE-M will return 253.0000 in X

INPUTTING DATA

Five functions are provided for inputting data from the ProtoPARIO: READ, READ1, READ5, READ7, and READA.

READ performs a single read without any waiting and returns the input in the exponent of X. Digits 12-3 are returned as 0 and digit 13 is 1. This causes X to look like ALPHA DATA so that it will not be normalized.

READ1 performs a set of reads, storing inputs in consecutive registers in the same format as READ: 10 00 00 00 00 0I II. X contains the destination registers: eee.bbb where eee is the last register to be written and bbb is the first. Y contains the wait loop constant (0-999) which is counted down before each read occurs. Experimentation will provide the actual time delay between reads. The instruction can be terminated before completion by pressing R/S. If in a program, execution will continue with the next instruction.

READ5 is identical to READ1 except that 5 consecutive reads of 10 bits are stored per register instead of 1. The data word is initialized to 0 then at each read the register is shifted 10 bits to the left and the data is transferred into the bottom 10 bits. After 5 reads or if the X-loop terminates, digit 13 is set to 1 so that the data will not be normalized. See instructions for READ1 above for X and Y register usage and loop termination.

READ7 is identical to READ1 except that 7 consecutive reads of 8 bits are stored per register instead of 1. The data word is initialized to 0 then at each read the register is shifted 8 bits to the left and the data is transferred into the bottom 8 bits. After 7 reads or if the X loop terminates, the data is written to a user register. Note that all 56 bits are used so that a RCL, VIEW, or X() instruction will normalize the register, changing the data. To get around this, use UNPACK7 or X()REG so that normalization is avoided. See instructions for READ1 above for X and Y register usage and loop termination.

READA is identical to READ1 except that the Y register is not used as a timing loop. Data is read continuously, but is only stored at 10 bits per register when the input changes from the previously stored input AND is non zero.

OUTPUTTING DATA

Five functions are provided for outputting data to the ProtoPARIO: WRIT, WRIT1, WRIT5, WRIT7, and WRITA.

WRIT provides a single write from the exponent of X without any waiting loop then returns. The exponent of X should contain 0-0-09-08 07-06-05-04 03-02-01-00.

WRIT1 performs a set of writes from the exponent of consecutive registers at 1 data output per register in the same format as for WRIT. See instructions for READ1 above for X and Y register usage and loop termination.

WRIT5 performs a set of writes from consecutive registers at 5 data outputs per register in the same format as for READ5. See instructions for READ1 above for X and Y register usage and loop termination.

WRIT7 performs a set of writes from consecutive registers at 7 data outputs per register in the same format as for READ7. See instructions for READ1 above for X and Y register usage and loop termination.

WRITA is identical to WRIT1 except that the Y register is not used as a timing loop. WRITA outputs one data word then continuously reads data until 000 appears at the input. This can be used to synchronize the calculator with an external by jumpering Output Ready to an input and asserting Output Accepted after each output from the calculator has been received by the external device.

PAUSING

Two functions are provided to introduce wait loops into I/O control for the ProtoPARIO: WAITNZ and WAITX.

WAITNZ continuously reads data from the ProtoPARIO until the input is non zero. The input is returned in X in the format 10 00 00 00 00 0I II. WAITNZ can be aborted by pressing R/S which will return X as 0 and continue with the next program line (if any).

WAITX continuously reads data from the ProtoPARIO until the input matches the contents of the exponent of X in the same format as for WAITNZ. WAITX can be aborted by pressing R/S.

REFORMATTING DATA

Four functions are provided to convert data between the 3 storage formats of 1, 5 or 7 data words per register: PACK5, PACK7, UNPACK5, and UNPACK7. The register formats are:
10 00 00 00 00 0D DD (1-10 bit datum per register, in digits)
dd DD dd DD dd DD dd (7-8 bit data per register, in digits, dd and DD are consecutive data)
0001 00DD DDDD DDDD dddd dddd ddDD DDDD DDDD dddd dddd ddDD DDDD DDDD (5-10 bit data per register, in bits, dd and DD are consecutive data).

Conversion is done between the X register and the first 5 (for PACK5 or UNPACK5) or first 7 (for PACK7 or UNPACK7) statistics registers. Any block of consecutive registers can be selected by using the summation register function - see the HP 41C Owners Manual.

PACK5 compresses the data in the first 5 statistics registers which are in 1-10 bit data word per register format into the X register in 5-10 bit data words per register format.

UNPACK5 reverses PACK5 by separating the X register into the 5 statistics registers.

PACK7 compresses the data in the first 7 statistics registers which are in 1-10 bit data word per register format into the X register in 7-8 bit data words per register format. The upper two bits in each data word are ignored.

UNPACK7 reverses PACK7 be separating the X register into the 7 statistics registers. The upper two bits are set to 0.

USING AN EXTERNAL KEYBOARD

The GOKEY function is designed to accept any 6-bit non zero input and map it onto the calculator keyboard as a key press. Note that the input is accepted as a keycode, therefore to enter ALPHA characters, you must first input the code that maps onto the ALPHA key. The table below shows the key press mapping for all 6-bit input combinations.

	LEAST SIGNIFICANT DIGIT															
MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		sum+	1/X	SQR	LOG	LN	X-Y	RDN	SIN	COS	TAN	XEQ	STO	RCL	ENT	CHS
1	EEX	-	7	8	9	+	4	5	6	*	1	USR	PGM	ALP	ENT	BSP
2	0	ON	R/S	SHF	SST	/	/	/	COS	TAN	*	+	.	-	.	/
3	0	1	2	3	4	5	6	7	8	9	/	.	COS	2	TAN	3

X-Y is X<>Y, SQR is SQRT, ALP is ALPHA, ENT is ENTER, BSP is back-arrow, SHF is SHIFT. Note that in ALPHA mode, hex inputs of 01-1A will generate alpha characters A-Z. Row 2 and 3 characters are mapped as closely as possible to ALPHA mode inputs versus ASCII; some may need to be SHIFTed. Upon execution, GOKEY will loop until any non zero input in the bottom 6 bits is received, then jump to the system routine to handle that keypress. GOKEY can be aborted by pressing R/S which will continue with the next program line (if any).

MISCELLANEOUS UTILITIES AND ROUTINES

ESCAPE will examine the character that is second from the right in ALPHA. If this character has a hex code < 20, it will be replaced with a space (hex code 20). This may be used in conjunction with DISASM on the NFCROM EPROM set to remove hex codes that would be interpreted as control codes or escape sequences by an external printer.

FETCH executes a CXISA (FETCH) microcode instruction at the address given in digits 3-0 of X. The result from the fetch is returned in X, and the fetched address is incremented and stored in L.

LISTU2 prints the encoded contents for the "U" EPROM in an EPROM set. This is useful when programming the contents of the Prototech, Inc. ProtoCODER (a user-programmable ROM emulator) onto EPROMs. To use, get the hex starting address (a multiple of 4) into digits 3-0 of X then execute LISTU2. A printer is required. U2 is a microcode subroutine used by LISTU2.

ROM displays a PARIO message.

RVIEW displays the first 3 digits of X, Y, Z, and T, separated by dots. The registers are not changed.

STK<>sum provides the user with multiple stack capability. By using sumREG (see the HP 41C Owners Manual) any block of consecutive registers can be selected. When executed, registers T, Z, Y, X, and L are exchanged with the first 5 summation registers. No normalization occurs.

U2 is a subroutine used by LISTU2 (see LISTU2).

X<>REG exchanges the X register with the absolute RAM register specified by the exponent of Y. No normalization occurs.

CHAPTER 7: PCODER-1A EPROM SET

This EPROM set contains many functions of general use, and several functions specifically for use with the ProtoCODER2, including microcode and user language manipulation functions:

PCODER-1A	demonstrates a possible display use
+1	demonstrates microcode speed
()	exchanges IND X with IND Y
ADELX	deletes X characters from left of ALPHA
AK	programmable hex code key assignments
AND	logical and of Y into X
BOOT	loads a block of user registers into the ProtoCODER2
CHKSUM	computes and stores checksum in the ProtoCODER2
CLRRAM	clears a block of words in the ProtoCODER2
CODE	converts hex code in ALPHA to non-normalized number in X
COPYPC	copies a user-language program into the ProtoCODER2
COPYXYZ	copies a block of ROM into the ProtoCODER2
D>H	converts X from floating point to hexadecimal
DECODE	converts non-normalized number in X to hex code in ALPHA
DECX	decrements X
DISASM	disassembles a ROM word
DUMP	copies a block of ROM words into user registers
GET	loads a 4K ROM image from cassette to the ProtoCODER2
H>D	converts X from hexadecimal to floating point
INCX	increments X
INIT	initializes ProtoCODER2
LOADB	user register byte examiner/loader
LODE	loads data into the ProtoCODER2
LOOK	ROM examiner
MANT	returns mantissa of X
MNEM	provides microcode mnemonic for disassembled ROM word
NOT	returns complement of X
OR	logical or of Y into X
PROMT	prompts for a hexadecimal input
RCLA	recalls user register at absolute address
ROMLIST	lists mnemonics of ROM to printer
ROMSUM	computes checksum of any ROM page
RXL1	rotates X left 1 bit
RXL4	rotates X left 1 digit
RXR1	rotates X right 1 bit
RXR4	rotates X right 1 digit
SAVE	copies any ROM page to cassette
STOA	stores data in user register at absolute address
SXL4	logically shifts X left 1 digit
SXR4	logically shifts X right 1 digit
TOGF	toggles user flag
X+Y	binary addition of Y into X
XOR	logical exclusive or of Y into X
*K	subroutine for AK
*ODE	subroutine for LODE

PCODER-1A FUNCTION INSTRUCTIONS

PCODER-1A (XROM 16,00): This function demonstrates the flexability of the display. To execute it, use AK to assign "A400" to a key then press that key.

+1 (XROM 16,01): This function demonstrates the speed of microcode on the HP 41C. When executed, it sets a counter to zero then continuously adds 1 to the counter until any key is pressed. +1 runs about 125 times faster than the equivalent RPN program:

```
1
ENTER
ENTER
ENTER
LBL 01
+
GTO 01
```

() (XROM 16,02): exchanges the contents of the two registers pointed to by X and Y. To exchange user register R02 with user register R07:

```
2
ENTER
7
()
```

ADELX (XROM 16,03): deletes X characters from the left of ALPHA. If X is greater than the number of characters remaining in ALPHA, then ALPHA will be cleared. Since ADELX uses the 24 character ALPHA register, if X>24, an error will result. To shorten "ABCDEFGHIJKL" to "EFGHIJKL":

```
"ABCDEFGHIJKL"
4
ADELX
```

AK (XROM 16,04): allows the user to assign any instruction hex code to any key. If executed from the keyboard, AK will prompt for 4 hex inputs which specify the instruction to be assigned, and then prompt for a key to which the assignment will be made. Any key assignable with ASN is also assignable from the keyboard with AK. AK will display the row and column of the key to be assigned (negative for shifted keys, just as with ASN). When AK is executed in a program, the instruction hex code is specified in the rightmost 4 digits of X - use CODE or PROMT. The Y register contains the hex code for the assigned key in the rightmost 2 digits. To determine this hex code, use AK to display the row R and column C, but hold the key down until NULL is displayed. The digits of the hex code to be stored will be C-1 and R, or C-1 and R+8 for a shifted key.

To assign "PCODER-1A" to the LN key (key #15), execute AK from the keyboard and enter A400 to the prompts, then press the LN key. Now press LN in USER mode to execute PCODER-1A.

To assign "RCL M" to the shifted LN key (key #-15) from a program, enter the following in PRGM mode:

```
LBL "ASSIGN"
"49"          4=5-1, 9=1+8 (shifted) as above
CODE
ENTER
"9075"        hex code for RCL M
CODE
AK
```

then leave PRGM mode, press RTN then R/S. In USER mode, press shift-LN to execute RCL M.

AND (XROM 16,05): logically ands Y into X, bit by bit. Y is unchanged, and the result is placed in X. The resulting bit in X will be 0 unless the corresponding bits in both X and Y were 1: then the resulting bit will be 1.

```
"76"    (= binary 01110110)
CODE
ENTER
"65"    (= binary 01100101)
CODE
AND
DECODE
```

will display "00000000000064" = binary 0...0 01100100.

BOOT (XROM 16,06): copies encoded data from user registers into the ProtoCODER2. Each ProtoCODER word consists of 10 bits of data, therefore 5 ProtoCODER words can be stored in 1-56 bit user register. The 5 words (a,b,c,d,e) are stored as:

0001 00aa aaaa aaaa bbbb bbbb bbcc cccc cccc dddd dddd ddee eeee eeee

The leftmost digit is set to 1 so that the data is treated as an ALPHA string and not normalized. If less than 5 words are contained in one register, only the leftmost portions of the register are used - a first, then b,c,d,e as needed.

Execute BOOT with:

- Z = READRX format of number of registers to use (e.g., for size = 275, set Z = 0.274)
- Y = CODEd (rightmost 6 digits) sssnnn where sss is the first address in the ProtoCODER to be loaded and nnn is the number of words to load
- X = floating point number specifying the first user register from which to load (e.g., 0.0000 or 1.0000)

When BOOT returns to the user, any ProtoCODER which had its device select switches set to 0000 will have been loaded, and:

- Z = number of registers for use with next READRX
- X = 0 if all words specified by Y were loaded, otherwise it contains the new sssnnn to be used as Y

For an example of the use of BOOT, list GET in the PCODER-1A EPROM set.

CHKSUM (XROM 16,08): computes and stores the checksum of a ROM page into any ProtoCODER with device select switches set to 0000. In keyboard mode, the user is prompted for the page for which the checksum is to be calculated. To put a valid checksum in your ProtoCODER, first LODE address FFF with 000, then XEQ"CHKSUM" and enter the page number of the ProtoCODER to the prompt. When CHKSUM is executed from a program, the page is specified by digit 12 (second from the left) of the X register. The checksum stored in word FFF of the ProtoCODER will be returned in digits 2-0 of X.

CLRRAM (XROM 16,09): clears a block of words in any ProtoCODER with device select switches set to 0000. The address (000-FFF) of the first word to be cleared is specified by digits 2-0 of X. The hexadecimal number of words to be cleared is specified by digits 2-0 of Y. If Y(2:0) = 000 or X + Y > FFF, then the ProtoCODER will be cleared starting at word X, ending at word FFF. To clear the whole ProtoCODER, execute:

```
0
ENTER
CLRRAM
```

which specifies starting address of 000, and clear to end-of-page (FFF).

CODE (XROM 16,10): converts the hexadecimal number in ALPHA into a non-normalized number in X. To create 4 full-man characters in X:

```
"10000001010101"    (alpha data, character code = 01)
CODE
```

COPYPC (XROM 16,11): copies an RPN (user language) program from user program memory into the ProtoCODER with device select switches set to 0000. No more than one ProtoCODER can be loaded at one time. COPYPC will prompt the user to enter an alpha string giving the name of the program to be copied. Press ALPHA twice with no input to copy the current program. Copying will begin with line 01 of the specified (or current) program and continue up to the END. If the specified program is PRIVATE, then the version stored in the ProtoCODER will also be PRIVATE. All short and long GTOs and XEQs will be compiled if possible. If a GTO is found to a nonexistent label or if a short (2 byte) GTO is found with the label out of range (more than 127 words away), then the jump distance will be specified as 0. This means that this GTO, when executed, will search for the specified label, giving a NONEXISTENT error message for a 2 byte GTO with a missing label, or continuing with the next program line for a 3 byte GTO. Therefore, the user should make sure that all LBLs specified in GTOs and XEQs actually exist.

In addition, the copy data will be set up so that the user can COPY his RPN program from the ProtoCODER back into user registers (unless PRIVATE).

COPYPC will attempt to enter every global LBL into the function address table at the beginning of the ProtoCODER. There must be space for at least one additional function when COPYPC is executed, otherwise the error message "FUN TBL FULL" will be displayed and no loading will occur. If there is space for at least one function in the FAT (Function Address Table) then each global label will be loaded into the FAT until it is full, or until there are no more global labels. No error will occur, but any remaining global labels will not appear in the CATalog.

Before attempting to load a program, COPYPC will determine if there is enough room in the ProtoCODER. If you used INIT to initialize your ProtoCODER, you may have noticed that two functions were loaded into the ProtoCODER: *PROTOCODER* and *END*. If the *END* exists as the last function in your ProtoCODER, COPYPC will attempt to load starting where *END* begins, and move *END* to follow the loaded program. If *END* is not there, COPYPC will prompt the user for STARTADR - the address of the first word to be loaded in the ProtoCODER. Be sure to enter the first digit as specifying the page (0-F) where the ProtoCODER is located. If COPYPC determines that there is not enough room from the *END* or from the STARTADR up to address FF4 to load your program, no loading will occur, and the error message "PCODER FULL" will be displayed.

If any additional space in the FAT exists after loading, and there is room in the ProtoCODER, a new *END* will be stored. This is also useful for the microcode programmer since it points to the first unused location in the ProtoCODER.

Displayed messages:

NONEXISTENT - program specified does not exist or contains no global labels

FUN TBL FULL - no more space is available in the CATalog of the ProtoCODER

PCODER FULL - insufficient space exists in the ProtoCODER to load the entire program

STARTADR---- - no *END* was found: enter a starting address where your program should be loaded

NO PCODER - the page specified in STARTADR does not contain a ProtoCODER or the device select switches on the ProtoCODER are not set to 0000

PACKING - the specified program was not packed: COPYPC will pack it then continue

LOADING - no fatal error occured, and the program will be loaded

If the global label specified in COPYPC is not found, the program counter will remain where it was. Otherwise, if the load is not successful, the program counter will point to the END of the current (or specified) program. If the load is successful, the program counter will point to the END of the copied program in ROM.

COPYXYZ (XROM 16,12): copies a block of Y ROM words starting from address X and copying starting at address Z. To use it, determine where in ROM you wish to start the copy (0000-FFFF). Use CODE to put this address in digits 3-0 of Z. Next, determine the number of ROM words to be copied. Use CODE to put this hex number in digits 2-0 of Y. If you specify Y as 0, or Z + Y would pass a page boundary (xFFF), then the copy will stop at location FFF of the page specified in Z. CODE the first address to be loaded in the ProtoCODER into digits 2-0 of X. When COPYXYZ is executed, any ProtoCODERs with device select switches set to 0000 will be loaded with the ROM words specified. To copy the system ABS function (which resides at addresses 1073-1078 hex) into the ProtoCODER beginning at address B00:

```
"1073"    start of ABS
CODE
ENTER
"6"       length of ABS (1078-1073+1)
CODE
ENTER
"B00"     ProtoCODER start address
CODE
COPYXYZ
```

To copy a complete 4K ROM plugged into port 4 (addresses E000-EFFF) into the ProtoCODER:

```
"E000"    start of ROM
CODE
0         specify copy to end of page
ENTER     also specify copy to start at address 000 into the ProtoCODER
COPYXYZ
```

A block of ROM can be moved around within the ProtoCODER with COPYXYZ, but remember that the copy is performed one word at a time sequentially from the start to the end, therefore you can move a block of memory down (to a lower address) by one word, but to move a block up (to a higher address) by one word, you must copy it to an unused portion of ProtoCODER and then copy it to your desired location.

D>H (XROM 16,13): converts a floating point (e.g., 1.0000) number in X into a hexadecimal number in X. The hex number is right justified in X, and zero filled to the left. Before the conversion, D>H converts the number in X so that it is non-negative and an integer. If the exponent of the number in X is 12 or greater, X is set to 0. Example:

```
1048575
D>H
DECODE
```

will display 0...0FFFFF = 2 ** 20 - 1 = 1048575. D>H is the inverse function of H>D.

DECODE (XROM 16,14): converts a non-normalized number in X into a hexadecimal number in ALPHA. To examine the hexadecimal representation, i.e., the way the calculator stores, the floating point number -.0123:

```
.0123
CHS
DECODE
```

to see 9123000000000998. Digit 13 (=9) specifies that the number is negative; digits 12-3 (=1230000000) are the mantissa; digits 2-0 are the exponent - complemented since it is negative. DECODE is the inverse function of CODE.

DECX (XROM 16,15): decrements the hexadecimal contents of the X register:

```
"1101"
CODE
DECX
DECX
DECODE
```

will display 0...010FF = 1101 - 2. DECX is the inverse function of INCX.

DISASM (XROM 16,16): disassembles ROM words. The address to be examined is specified as a hexadecimal number (0000-FFFF) in digits 3-0 of X (use CODE or PROMT). DISASM returns with X incremented by 1, digits 6-3 of Y contain the ROM address, digits 2-0 of Y contain the contents of that ROM address (000-3FF), and ALPHA contains "aaaa ddd c " where aaaa is the ROM address, ddd is the ROM data at that address, and c is the character interpretation of the data. Use DISASM with MNEM to provide fully disassembled ROM listings with mnemonics. ROMLIST will do this for you - list it to see how to use DISASM with MNEM. Example:

```
0        starting ROM address 0000
DISASM   displays "0000 201 A " - contents at address 0000 of system ROM
DISASM   displays "0001 006 F " - contents at address 0001
DISASM   displays "0002 2B5 5 " - contents at address 0002
```

DUMP (XROM 16,17): encodes ROM data into user data registers. ROM words each consist of 10 bits of data, therefore 5 ROM words can be stored in 1-56 bit user data register. The 5 words are stored as:

0001 00aa aaaa aaaa bbbb bbbb bbcc cccc cccc dddd dddd ddee eeee eeee

The leftmost digit is set to 1 so that the data is treated as an ALPHA string and not normalized. If less than 5 words are copied into one register, only the leftmost portions of the register are used - a first, then b,c,d,e.

Execute DUMP with:

Y = CODEd (rightmost 8 digits) sssseeee where ssss is the first ROM address to be dumped and eeee is the last ROM address to be dumped,

X = floating point number specifying the first user register to be loaded (e.g., 0.0000 or 1.0000) - only the integer portion of X is used.

When DUMP returns to the user, user registers starting at the register specified by X and continuing up to the last user register (if that many registers were needed) will have been loaded, and:

Y = 0 if all data words have been dumped, otherwise it contains the new sssseeee to be used for another execution of DUMP, after the dumped registers are saved to cassette or magcards (or whatever),

X = floating point of bbb.eee where bbb is the starting register and eee is the ending register containing data - to be used with WRITRX.

For an example of the use of DUMP, list SAVE in the PCODER-1A EPROM set.

GET (XROM 16,18): prompts the user for a file name to be loaded from cassette into any ProtoCODER with device select switches set to 0000. GET will load data files created by SAVE. The first record is a header with the format:

10 00 ll ls ss sn nn

where lll is the number of records in the file, ssss is the ROM address (0000-FFFF) from which the original ROM data came, and nnn is the number of ROM words minus 1 contained in the file. This header record and all data records are compatable with cassette files created by functions in ASSEMBLER 3 (an Australian microcoding EPROM set for the MLI). The file loaded by GET need not be 821 records (as created by SAVE): GET uses the data in the first record to load the ROM data into the ProtoCODER. nnn + 1 words will be loaded beginning at address xsss in the ProtoCODER. The use of GET and SAVE greatly simplifies the transfer of EPROM, ROM, and ProtoCODER images between two users - no EPROM burning is necessary, and GET and SAVE are automated enough that very little user intervention is required. If you wish to create cassette files other than of a full 4K ROM page, use BOOT and DUMP. GET should be executed with as large a SIZE as possible to minimize the number of copy loops required, and speed up the copy time.

H>D (XROM 16,19): converts a hexadecimal number in X into a floating point number (e.g., 1.0000) in X. If the hex number is greater than 2540BE3FF, roundoff error may occur, and the exponent may be negative, and in hex. If the hex number is greater than FFFFFFFFF then X will be returned as 0.
Example:

```
"FFFFF"    = 2 ** 20 - 1 = 1048575
CODE
H>D
```

to see 1048575. H>D is the inverse function of D>H.

INCX (XROM 16,20): increments the hexadecimal contents of the X register:

```
"10FF"
CODE
INCX
INCX
DECODE
```

will display 0...01101 = 10FF + 2. INCX is the inverse function of DECX.

INIT (XROM 16,21): initializes any ProtoCODER with device select switches set to 0000. INIT first clears all 4K of the ProtoCODER then prompts the user for a hexadecimal XROM#. Input the XROM number you wish to use: 01-1F. INIT then prompts for the maximum number of entries you plan to have in the catalog of your ProtoCODER (02-3F). It is better to waste a few ROM words by overestimating this number, than to later try to move the ProtoCODER contents around to make more catalog space available. INIT will then load a function called *PROTOCODER* as the first function in your ProtoCODER, and a function called *END* as the second. *END* is used by COPYPC (and possible future software) to point to the last words in use in the ProtoCODER. You can change the ROM name (*PROTOCODER*) if you wish be examining locations x002-x003 in the ProtoCODER with LOOK. These locations point to the first executable word of the first function. Use LODE to enter a new function name preceding this address - see "XROM Word Format" and "Function names, Prompting, and Non-programmability".

LOADB (XROM 16,22): examines and modifies user register contents. It is non-programmable, and can be executed in PRGM mode. When LOADB is executed, it determines where the program counter is pointing. If it is in a ROM program, the error message ROM will be displayed. Otherwise, LOADB will begin at the current program counter location. The display will show:

```
R=rrr B=b cc
```

where rrr is the current register number (000-FFF) and b is the current byte within the register (6-0). cc is the hexadecimal contents of that byte. For example, do a master clear then enter the following program:

```
LBL"TEST"
STO 30
DECODE
```

then BST,BST to see 01 LBL"TEST". XEQ"LOADB" to see R=0EE B=6 C0. "C0" is the first byte of the LBL"TEST" instruction, which begins at the leftmost byte (B=6) of register 0EE. Press SST,SST to see R=0EE B=4 F5. "F5" is the byte specifying the length of the global label (plus 1). Again SST,SST to see R=0EE B=2 54. "54" is the ASCII character code for "T", the first character of the global label. To change the "T" to a "B", press backarrow to tell LOADB that you want to change the current contents of the byte displayed. LOADB will replace the "54" with two prompts to which you can enter any hexadecimal value. To change LBL"TEST" to LBL"BEST", enter "42", the hex code for "B". Now press shift,SST,SST,SST,SST to get back to the first byte of the global label. Press shift again to leave the backstep mode, then press R/S to leave LOADB. The display shows:

01 LBL"BEST". Now XEQ"LOADB" again. Examining the above program, you guess that the STO 30 instruction is about 1 register beyond the global label (i.e., 1 register lower in user memory), so press GTO and enter 0ED (= 0EE - 1) to the prompts. The display will show R=0ED B=6 54. This "54" is the last byte of the global label instruction. Press SST to see R=0ED B=5 91 - the hex code for STO. Press SST again to see R=0ED B=4 1E - the hex code for register 30. Press backarrow and enter 75 to the prompts. Then press shift,SST,shift,R/S to see 02 STO M. Now press SST to see 03 DECODE. XEQ"LOADB" to see R=0ED B=3 A4 - the first byte of the XROM code for DECODE. Press SST,backarrow,0,A,shift,SST,shift,R/S to see 03 CODE.
If you wish to insert a byte into user memory and no null already exists at that point, just position the program counter in LOADB using SST and BST then press ENTER. LOADB will insert a register of 7 null bytes that can then be modified to whatever you wish. Use PACK to remove these nulls when you have finished.

LOADB moves the program counter as you GTO, SST, or BST. Therefore, before you leave LOADB, you should position the program counter to point to the first byte of an instruction. Also note that the shift key is a direction mode - if shift is on (check the display), then pressing SST will execute a BST and leave shift set. If shift is off then pressing SST will execute an SST and leave shift cleared.

Allowable keypresses for LOADB:

- backarrow - prompts the user for a hexadecimal byte to replace the current byte at the displayed location,
- ENTER - to insert a block of 7 nulls if no null exists at the current location,
- GTO - prompts the user for a hex register which will become the current location - enter 000-FFF. The program counter will be changed to point to the left-most byte of the specified register,
- OFF - turns calculator off. Also can be used to leave LOADB if you pressed back-arrow and do not wish to alter the current contents of the specified byte,
- R/S - exits LOADB. Use SST or BST to move the program counter to point to the first byte of an instruction before pressing R/S,
- shift - changes the direction mode for SST. If shift is on, pressing SST will execute a BST. If shift is off, pressing SST will execute a SST,
- SST - moves the program counter one byte forward (shift clear) or one byte backward (shift set) in the current program.

LODE (XROM 16,23): loads bytes into any ProtoCODER with device select switches set to 0000. When LODE is executed, it will prompt for "ADDRESS ---". Enter the address of the first word of the ProtoCODER (000-FFF) to be loaded. Then LODE will prompt for successive data words to load into the displayed address. Input 000-3FF. To leave LODE, press backarrow. Example: After you used INIT to initialize your ProtoCODER, and you have entered a few programs, you wish to change the XROM number of your ProtoCODER. The XROM number is stored at word 000 in the ProtoCODER. XEQ"LODE" then press 000 (or just R/S) to the ADDRESS --- prompt. This calculator will now display 000 ---. Enter the new XROM number (e.g., 005). The display will now show 001 ---. Press backarrow to leave LODE.

LOOK (XROM 16,24): displays hexcodes and mnemonics of ROM locations. When executed, LOOK jumps to the GTO routine and prompts you for an address to examine. Enter 0000-FFFF. LOOK will then display either the result of DISASM (flag 0 clear) or the result of MNEM (flag 0 set). For example, start with flag 0 clear. XEQ"LOOK" and enter 0000 (or just press R/S) to the GOTO ---- prompt. The display will show 0000 201 A - the contents of ROM word 0000. Press SST to see 0001 006 F. To see the mnemonic, press shift,SST,shift to return to location 0000, the press PRGM to toggle flag 0, and display the mnemonics. The display will be blank - the first line of an absolute GOTO. Press SST to see GOLONG 0180.

Allowable keypresses:

- GTO - prompts the user for a new hex address to examine - 0000-FFFF,
- OFF - turns calculator off,

PRGM - toggles flag 0, which specifies if the display will show the result from DISASM (flag 0 clear) or MNEM (flag 0 set),
R/S - exits LOOK,
shift - changes the direction mode of SST. If shift is clear, then SST will execute an SST. If shift is set, then SST will execute a BST,
SST - moves the current location displayed forward one word (shift clear) or backward one word (shift set).

Note that some GOTO and GOSUB instructions are followed by one or more words of data which can appear as random instructions to MNEM. Sometimes a data word will appear as a two-word instruction and make MNEM think that the following word, which is actually a valid one-word instruction, is the second word of a two-word instruction. To display the true mnemonic for the second word, just press PRGM twice.

MANT (XROM 16,25): clears the exponent of X to 000 - returns the mantissa of X to X. Example:

```
1234.56
MANT
```

returns 1.23456 - the exponent was set to zero. As pointed out by Heinz Schaefer and others, MANT can be used with XOR to isolate the exponent of X:

```
1234.56
ENTER
MANT
XOR
DECODE
```

to see 0...03 - the exponent of 1234.56.

MNEM (XROM 16,26): provides mnemonics for disassembler listings of ROMs. It uses data from the X and Y registers as provided by DISASM. The first half of the mnemonic is returned in Z; the second half in T. L is also used for two-word instructions (GOTO, GOSUB, LDI). List ROMLIST for an example of how MNEM can be used.

NOT (XROM 16,28): complements the X register, replacing each 0 bit with a 1 and each 1 bit with a 0. In hexadecimal, each digit (0-F) is replaced by F minus the digit. Example:

```
"0123456789ABCD"
CODE
NOT
DECODE
```

will display "FEDCBA98765432".

OR (XROM 16,29): logically ors Y into X, bit by bit. Y is unchanged, and the result is placed in X. The resulting bit in X will be 1 if either of the corresponding bits in X or Y were 1, otherwise the resulting bit will be 0.

```
"33"     (= binary 00110011)
CODE
ENTER
"85"     (= binary 10000101)
CODE
OR
DECODE
```

will display 0...0B7 = binary 0...0 10110111.

PROMT (XROM 16,30): provides the user with simplified hexadecimal inputting capability. PROMT will examine digit 12 of the X register to determine how many prompt digits to accept for input (0-9). It will then add that many overline prompt characters to the display (shifting out part of the display if there is not enough room for the prompt marks) and wait for the user to enter hexadecimal inputs. The resulting inputs will be returned in the X register, right justified and zero filled on the left. For you software hackers: If digit 12 is 0,D,E, or F, PROMT will return with no inputting and X=0. If digit 12 is A,B, or C, PROMT will prompt for 10, 11, or 12 digits respectively. PROMT with digit 12 = C shifts 4 digits in at a time - the display looks a little strange, but the input is correct.

Allowable keypresses:

- any digit 0-9,
- any letter A-F = hex digits corresponding to decimal 10-15,
- backarrow - to delete the last digit entry. If you backarrow when no digits have been input, PROMT will return X=0, and skip the next line if executing a program,
- R/S - returns with the input as it is, and fills the input with zeroes on the left - hold R/S down to see the input as it will be returned,
- OFF - turns calculator off - can be used to exit PROMT if no exiting is provided by the calling program.

Example: The following program accepts 3 digits of input and complements them. Press backarrow to quit.

```
LBL"SAMPLE
"INPUT"
LBL 01
AVIEW
3
PROMT
FS? 30    skip next instruction
RTN
NOT
ENTER
MANT
XOR
DECODE
11
ADELX
GTO 01
```

RCLA (XROM 16,31): recalls the contents of the user register with absolute address as given as a floating point number in X. The recalled register overwrites the current contents of X. No normalization occurs. Example:

```
SIZE 001
100
STO 00
CLST
511       = absolute address of user register R00 = hex 1FF
RCLA
```

to see 100 in the X register.

ROMLIST (XROM 16,32): lists mnemonics of ROM words to the printer. When executed, ROMLIST will prompt for START ----. Enter the starting ROM address to be listed - 0000-FFFF. ROMLIST will then list ROM words sequentially from the input address until R/S is pressed. The output format is:

```
aaaa ddd c mmmmmmm mmmmmmm
```

where aaaa is the address, ddd is the data at that address, c is the character representation of the data, and mmmmmmmmmmmmmmm is the mnemonic interpretation.

ROMSUM (XROM 16,33): computes the checksum of any ROM page. In keyboard mode, the user is prompted for PAGE -. Input the hex page - 0-F. When executed from a program, ROMSUM computes the checksum of the page as specified by digit 12 of X. The 3-digit checksum is returned in digits 2-0 of X.

RXL1 (XROM 16,34): rotates the contents of the X register left by 1 bit. For example:

```
"A1234567890FED"   = binary 1010 0001 0010 0011 0100 0101 0110 0111 1000 1001 0000 1111
                     1110 1101
CODE
RXL1
DECODE
```

shows "42468ACF121FDB" = binary 0100 0010 0100 0110 1000 1010 1100 1111 0001 0010 0001 1111 1101 1011. Notice that the leftmost bit shifted around into the rightmost position. RXL1 is the inverse function of RXR1.

RXL4 (XROM 16,35): rotates the contents of the X register 1 digit (4 bits) to the left. Example:

```
"1234"
CODE
RXL4
DECODE
```

shows 0...012340. The leftmost digit is rotated around into the rightmost position. RXL4 is the inverse function of RXR4.

RXR1 (XROM 16,36): rotates the contents of the X register right 1 bit. The rightmost bit is rotated around into the leftmost position. RXR1 is the inverse function of RXL1.

RXR4 (XROM 16,37): rotates the contents of the X register right 1 digit (4 bits). The rightmost digit is rotated around into the leftmost position. RXR4 is the inverse function of RXL4.

SAVE (XROM 16,38): prompts the user for a ROM page (0-F) and then a file name. The specified ROM page is copied to cassette with the specified file name. SAVE creates an 821 record file. The first (header) record is of the format:

```
10 00 33 5p 00 0F FF
```

where p is the page number copied to cassette. The next 820 records contain the ROM data at 5 ROM words per record in the binary format:

```
0001 00aa aaaa aaaa bbbb bbbb bbcc cccc cccc dddd dddd ddee eeee eeee
```

where a,b,c,d,e are 5 consecutive ROM words. The last record uses only the a portion - b,c,d,e are all zero. This file format is compatable with the cassette file format as created by functions in ASSEMBLER 3 (an Australian microcoding EPROM set for the MLI). SAVE creates files to be read into the ProtoCODER with GET. SAVE should be executed with as large a SIZE as possible to minimize the number of copy loops required, and speed up the copy time.

STOA (XROM 16,39): stores the contents of the Y register into the absolute register as specified by a floating point number in X. No normalization occurs. Example:

```
SIZE 001
300
ENTER
511      = absolute address of register R00 = hex 1FF
STOA
RCL 00  to see 300.
```

SXL4 (XROM 16,40): logically shifts the contents of the X register left 1 digit (4 bits). A zero is shifted into the rightmost digit. Example:

```
"1234"
CODE
SXL4
DECODE
```

to see 0...012340.

SXR4 (XROM 16,41): logically shifts the contents of the X register right 1 digit (4 bits). A zero is shifted into the leftmost digit. Example:

```
"1234"
CODE
SXR4
DECODE
```

to see 0...0123.

TOGF (XROM 16,42): toggles the user flag as specified by the floating point number in the X register. If the specified flag was on (=1) then TOGF turns it off (=0). If the flag was off then TOGF turns it on. Example:

```
49
TOGF
```

to see the BAT annunciator. Execute TOGF again to turn BAT off.

X+Y (XROM 16,43): adds the binary number in Y into the binary number in X. Y remains unchanged. Example:

```
"110F"
CODE
ENTER
"322"
CODE
X+Y
DECODE
```

to see 0...01431 = hex 110F + 322.

XOR (XROM 16,44): logically exclusive-ors Y into X, bit by bit. Y is unchanged, and the result is placed in X. The resulting bit in X will be 1 if the corresponding bits in X and Y were not equal (0 and 1, or 1 and 0). The resulting bit will be 0 if the corresponding bits were equal (0 and 0, or 1 and 1). Example:

```
"33"    (= binary 00110011)
CODE
ENTER
"85"    (= binary 10000101)
CODE
XOR
DECODE
```

will display 0...0B6 = binary 0...0 10110110.

*K (XROM 16,45): is a subroutine used by AK to return from the keypress prompt. The system ASN routine prompts for a keypress which specifies the key to which the assignment will be made. After accepting this keypress, the system does not return to the calling program; therefore, *K was necessary. *K can be used from the microcode level by putting the hexcode of the instruction to be assigned into digits 3-0 of X and the coded keycode in digits 1-0 of the internal A register, then executing *K.

*ODE (XROM 16,46): is a subroutine used by LODE. To use it, CODE the hex value of the instruction to be loaded into digits 2-0 of X. CODE the address where the instruction is to be loaded into digits 2-0 of Z. Then execute *ODE in a program. The address which was in Z is incremented and returned in X. Also, the next line of the program is skipped. *ODE executed from the keyboard will function properly but display the message "NO" when it returns.

APPENDIX 1: PROTOTECH, INC. PRODUCTS

The following products are available from Prototech, Inc.:

ProtoCODER2 is the main control box that plugs into the HP 41C. It provides the peripheral boards with data, control, and address signals. It includes one 4K ProtoCODER.

ProtoCODER provides the user with 4096 (4K) words of memory which is programmable in microcode (the machine language of the microprocessor in the calculator). No external EPROM is necessary to program the ProtoCODER; however, the PCODER-1A EPROM set provides many useful functions to assist in microcode programming.

ProtoEPROM allows the user to plug in one HP-format EPROM set containing user language programs and/or microcode.

ProtoROM allows the user to plug in up to 4 HP Application modules. Each module can be individually switched on or off into any port. The switching can be done from the keyboard or under program control.

ProtoPARIO is a general purpose 10-bit input/output interface for the HP 41C. Applications might include interfacing two calculators, or interfacing to a computer, light controller, full size ASCII keyboard, or a voltmeter. With appropriate software, data from external devices can be sampled and stored at up to 700 times per second with an unmodified HP 41C. The ProtoPARIO is not enclosed in a box, since the user will be required to make hardware connections to the circuit board. It requires a ProtoEPROM.

NFCROM-1B is an EPROM set containing many routines useful for programming the original ProtoCODER (which uses the SIGN function to perform a write, whereas the ProtoCODER2 uses ABS). It contains the following functions which are similar to those in the PCODER-1A EPROM set: DUMP, CODE, +1, PROMT, MANT, (), BOOT, DISASM, RCLA, DECODE, AK, LODE, MNEM, ROMSUM, DEC-HEX (D)H), HEX-DEC (H)D), STA (STOA), TOGF, LOAD (*ODE), INIT, X+Y, OR, AND, XOR, NOT. It also contains:

NFCROM-1B displays message
, appends left goose to display
CL clears system flag 12
. appends right goose to display
ROM? displays ROM 0,1,2 revisions
CAT lists online ROMS or CAT2 starting at any page
LODB unfinished byte loader
M10INT returns INT(X)MOD10 to X
BJUMP byte jumper
LEFT rotates display to left
DIS appends any character to display
DISTST display test
X=1? comparison
POW2 unfinished extended precision powers of 2
COPEE copies any ROM into ProtoCODER
SST fast continuous single step
BST fast continuous back step

NFCROM-1C is an EPROM set which is identical to NFCROM-1B except that the bugs in AK and HEX-DEC have been patched.

IDEAL is an 8K EPROM set including the NFCROM-1B and 4K of additional routines for use with the original ProtoCODER (SIGN type - not ABS). Of major importance to the ProtoCODER user are the START and RESTART programs. START initializes the ProtoCODER. RESTART copies a user language (RPN) program from user memory into the ProtoCODER, computing all the GTO distances, etc. For users with a modified 82143A printer, barcode printing programs are provided. To aid in debugging, two programs list all LBL, GTO, XEQ instructions and all registers, labels and flags used. Various other utility routines are also provided. This EPROM set is shipped as written, and with a xerox of the instructions as provided by the authors. Prototech, Inc. has this EPROM set available for sale only, and will not provide any support for the IDEAL portion of this EPROM set.

This ProtoCODER2 manual is included with any ProtoCODER2 ordered, but is also available separately.

For a limited time, Prototech, Inc. will provide upgrading services to change your SIGN function ProtoSYSTEM into a ProtoCODER2 by adding appropriate jumpers on your board. If you have a ProtoROM, it will need a slight modification also. For details and pricing of this modification, contact Nelson Crowle at Prototech, Inc.

APPENDIX 2: WARRANTY, SERVICE, ASSISTANCE

LIMITED WARRANTY

The ProtoCODER2 and all ProtoSYSTEM peripherals manufactured by Prototech, Inc. are warranted against defects in materials and workmanship for a period of ninety (90) days from the date shipped from Prototech, Inc. Within this warranty period, Prototech, Inc. will repair or at its option replace a defective part at no charge to the owner, provided that Prototech, Inc. is contacted within the warranty period for shipping instructions. There will be a charge for repairs after the warranty period has expired. Prototech, Inc. assumes no responsibility for damages, either direct or consequential, from the use of its products. Prototech, Inc. will have no obligation to modify or update products after sale. This warranty does not apply to products damaged by accident or misuse, or to products that have been modified by anyone other than Prototech, Inc., and does not apply to the 4013, 4503, 74C174, and 74C175 interfacing chips in the ProtoPARIO. This warranty is made in lieu of all other warranties, either express or implied.

SERVICE

If your ProtoCODER2 or any Prototech, Inc. product requires service, contact Prototech, Inc. for instructions.

ASSISTANCE

If you need technical or applications assistance relating to the use of the ProtoCODER2, please contact Prototech, Inc. at (303)-499-5541 (no collect calls), or write to:

PROTOTECH, INC.
P. O. BOX 12104
BOULDER, CO 80303 USA

APPENDIX 3: PPC INFORMATION

PPC is the Personal Programming Center which is an organization of users dedicated to personal computing. It is the oldest personal computing group in the world. PPC publishes the PPC Calculator Journal which disemenates information and programs for HP calculators. For information on membership, obtaining back issues of the PPCCJ, and information about the PPC ROM or PPC EPROMs, send a 9x12" envelope with 2oz of postage or equivalent international postal coupons to:

PPC
2545 W. CAMDEN PLACE
SANTA ANA, CA 92704

PPC Technical Notes is a publication of the Melbourne Chapter of the PPC. For subscription information, send a self addressed envelope and international postal coupons to:

PPCTN
J. E. McGECHIE
P O BOX 512
RINGWOOD, VICTORIA 3134
AUSTRALIA

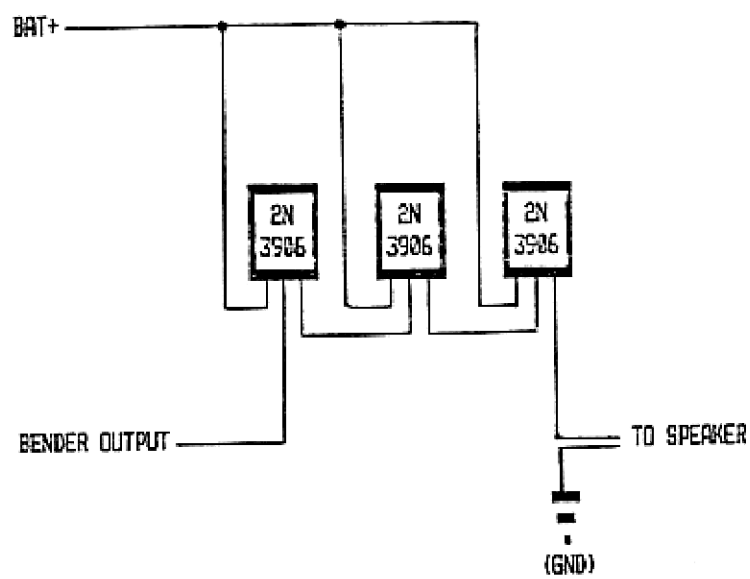
PPC EPROM sets are currently available from:

JOE BELL
SURVEY CALCULATIONS JOURNAL
P O BOX 6674
SAN BERNARDINO, CA 92412

APPENDIX 4: INTERNAL BENDER AMPLIFIER

This simple circuit can be built within the calculator to provide a large volume increase from the bender output into an external speaker. The only parts you need are a miniature speaker (about 1 1/2 inches), 3 2N3906 transistors, a small plug and jack, and some wire and solder. Total cost is about $3. Note that this modification is not supported by HP and will void your warranty. Prototech, Inc. assumes no responsibility for the use of this information. It is provided for the users reference only.

Remove the battery pack and all modules then remove the four screws from under the rubber pads on the back of the calculator and lift the back of the calculator off. Locate the bender (1-inch flat metal disk stuck onto the CPU) and unstick it. There are two wires connected to the bender. The inner one on the smaller section of the bender is the bender output signal. Solder a wire on top of the wire that is already there. Locate the plastic-copper battery contacts (where the battery pack plugs in) and scrape a small hole in the plastic at some location on both the BAT+ and GND contacts that will not get in the way of the battery pack. Solder a wire to each contact. Locate a place to put the output jack. Radio Shack sells a plug and jack combination that fits tightly into the battery charger hole. You should now have 3 wires added on to your calculator: bender output, BAT+, and GND. Solder the 3 transistors together as shown below and attach the 3 wires and the jack. Also wire the two speaker contacts to the plug. The transistors will fit easily in the calculator along the side of the I/O ports. After verifying that you have wired everything correctly, reassemble and try it out. Note that this does draw significantly more power than the bender alone. Transistors in the diagram are shown with their flat face forwards.



APPENDIX 5:HP 41C MICROCODE

The HP 41's brain (microprocessor) defines what can and cannot be done with the calculator by having a specific set of instructions. These instructions are referred to as microcode. They are stored in ROMs (Read Only Memories) or anything that looks like a ROM to the calculator, such as a ProtoCODER or EPROM. The sequence of these instructions determines what the calculator will actually do. It gives the calculator its personality that makes it act like an HP 41C. The calculator will function just as well with some other operating system or language and could be changed to a completely different personality just by changing these ROMs. By using a disassembler program (such as ROMLIST in the PCODER-1A) you can list the contents of the ROMs in the calculator to get a general idea of how things are done in microcode.

The processor of the HP 41 has a set of internal registers in which all of its operations are performed. Registers A, B, C (different from the user stack a,b,c registers), M, and N are 56-bit registers - the same size as user and stack registers. Arithmetic, logical, and input/output operations are performed with A, B, and C. M and N are used for temporary storage. The PC register is 16 bits long and contains the address of the next ROM word to be executed. It is normally incremented after each instruction is executed, but can be modified by a GOLONG, GOSUB, GOC, GONC, or RTN instruction. Since PC is 16 bits in length, the calculator can address 65536 (2 ** 16) locations.

There are also 4-8 bit registers (G, KEY, ST, and T(or F)), 2-4 bit registers (P and Q), 14 system flags (13-0), a KB flag which is set when a key has been pressed, and a C (Carry or Condition) flag. The G register is used for temporary storage. The KEY register contains the keycode of the last key pressed, the ST register contains system flags 0-7, and the T or F register controls the bender to make beeps. The P and Q registers point to a digit (13-0 from left to right) in the 56-bit registers. The active "pointer" (either P or Q) is called R. Normally, only one pointer is active at one time. System flags 7-0 can be accessed by using ST. System flags 13-8 can only be accessed individually. Flags 13-10 are dedicated for certain system uses: Flag 13 is set if a program is running; flag 12 is set to indicate a PRIVATE program; flag 11 is set to enable a user stack lift at the end of an instruction; flag 10 is set to indicate that the program pointer in the user stack register b is a ROM address. The C flag is set when a test is true, when a carry occurs, and when the calculator is first turned on by pressing the ON key. It remains set for one instruction, then is cleared.

All 56-bit registers are separated into several fields. The user can select which field within the register is affected by an operation. The part of the register outside that field is not affected. The 56 bits are separated into blocks of 4 bits, each called a digit or nybble. They are numbered from left to right (high order to low order) as 13-0. Each digit or a contiguous block of digits can be operated upon using the P and/or Q pointers. Other named fields are:

S Mantissa sign - digit 13
M Mantissa - digits 12-3
XS Exponent sign - digit 2
X Exponent - digits 2-0
ADR Address field - digits 6-3
KB Key buffer - digits 4-3

There are four classes of microcode instructions numbered as 0, 1, 2, and 3. The class is determined by the right-most two bits of the 10-bit instruction. The following tables list both the HP mnemonics for microcode instructions and the mnemonics first published by Steve Jacobs (PPC #5358) in PPC ROM LISTINGS 2. The HP mnemonic is listed followed in parenthesis by Steve Jacobs' mnemonic. If you examine HP microcode listings you will find instructions that are not listed below. These instructions are actually macros in the HP assembler to simplify the programming. A macro is an instruction which has no meaning to the microprocessor, but which is replaced by a sequence of 1 or more instructions that the microprocessor does understand. For example, the instruction "C=A" is not listed below because it is a macro. When the HP assembler encounters "C=A" it replaces it with the sequence "AC EX", "A=C".

CLASS 0 INSTRUCTIONS

There are two types of Class 0 instructions: parametric and special. The parametric instruction hex codes specify a field or register upon which the operation will occur:

Sp=0	(CLRF p)	Clear system flag p
Sp=1	(SETF p)	Set system flag p
?Sp=1	(?FSET p)	Set C flag if system flag p is set
LC p	(LD@R- p)	Load p into C at PT, decrement PT
?PT=p	(?R= p)	Set C flag if pointer equals p
PT=p	(R=p)	Set selected pointer to p
SELPp	(SELP p)	Transfer control to peripheral p
REGN=Cp	(WRIT p)	Write C to peripheral or memory
?Fp=1	(?FI p)	Set C flag if peripheral flag set
C=REGNp	(READ p)	Read C from peripheral or memory
RCRp	(RCR p)	Rotate C right by p digits

Hex codes for Class 0 parametric instructions are:

INSTR MNEMONIC	p=0 (T)	1 (Z)	2 (Y)	3 (X)	4 (L)	5 (M)	6 (N)	7 (O)	8 (P)	9 (Q)	10 (+)	11 (a)	12 (b)	13 (c)	14 (d)	15 (e)
Sp=0	384	304	204	004	044	084	144	284	104	244	0C4	184	344	2C4	—	—
Sp=1	388	308	208	008	048	088	148	288	108	248	0C8	188	348	2C8	—	—
?Sp=1	38C	30C	20C	00C	04C	08C	14C	28C	10C	24C	0CC	18C	34C	2CC	—	—
LCp	010	050	090	0D0	110	150	190	1D0	210	250	290	2D0	310	350	390	3D0
?PT=p	394	314	214	014	054	094	154	294	114	254	0D4	194	354	2D4	—	—
PT=p	39C	31C	21C	01C	05C	09C	15C	29C	11C	25C	0DC	19C	35C	2DC	—	—
SELPp	3A4	324	224	024	064	0A4	164	2A4	124	264	0E4	1A4	364	2E4	1E4	3E4
REGN=Cp	028	068	0A8	0E8	128	168	1A8	1E8	228	268	2A8	2E8	328	368	3A8	3E8
?Fp=1	3AC	32C	22C	02C	06C	0AC	16C	2AC	12C	26C	0EC	1AC	36C	2EC	1EC	3EC
C=REGNp	038	078	0B8	0F8	138	178	1B8	1F8	238	278	2B8	2F8	338	378	3B8	3F8
RCRp	3BC	33C	23C	03C	07C	0BC	17C	2BC	13C	27C	0FC	1BC	37C	2FC	1FC	3FC

Hex codes for Class 0 special instructions are:

HEX	MNEMONIC	OPERATION
3C4	CLR ST (ST=0)	Clears ST and flags 7-0
3C8	RST KB (CLRKEY)	Clears KB flag
3CC	CHK KB (?KEY)	Set C flag if key pressed
3D4	DEC PT (R=R-1)	Decrement current pointer
3DC	INC PT (R=R+1)	Increment current pointer
058	G=C (G=C @R,+)	Copy digits R,R+1 from C into G
098	C=G (C=G @R,+)	Copy G into digits R,R+1 of C
0D8	CG EX (C<>G @R,+)	Exchange G with digits R,R+1 of C
158	M=C (M=C ALL)	Copy C into M
198	C=M (C=M ALL)	Copy M into C
1D8	MC EX (C<>M ALL)	Exchange M with C
258	F=ST (T=ST)	Copy ST into F
298	ST=F (ST=T)	Copy F into ST
2D8	FST EX (ST<>T)	Exchange F with ST
358	ST=C (ST=C X)	Copy digits 1,0 from C to ST
398	C=ST (C=ST X)	Copy ST into digits 1,0 or C
3D8	CST EX (C<>ST)	Exchange ST with digits 1,0 of C
020	SPOPND (XQ->GO)	Drop return stack to convert GOSUB to GOTO
060	POWOFF (POWOFF)	Go to standby mode

Class 0 special instructions hex codes, continued:

HEX	MNEMONIC	OPERATION
0A0	SEL P (SLCT P)	Select P as the active pointer
0E0	SEL Q (SLCT Q)	Select Q as the active pointer
120	?P=Q (?P=Q)	Set C flag if pointers are equal
160	?LLD (?LOWBAT)	Set C flag if low battery
1A0	CLRABC (A=B=C=0)	Clear registers A, B, C to zero
1E0	GOTOC (GOTO ADR)	Copy digits 6-3 of C into PC
220	C=KEYS (C=KEY KY)	Copy KEY register into digits 4-3 of C
260	SETHEX (SETHEX)	Use hexadecimal arithmetic
2A0	SETDEC (SETDEC)	Use decimal arithmetic
2E0	DISOFF (DSPOFF)	Turn off display
320	DISTOG (DSPTOG)	Toggle display off to on or on to off
360	RTN C (?C RTN)	If C set then return from subroutine
3A0	RTN NC (?NC RTN)	If C clear then return from subroutine
3E0	RTN (RTN)	Pop stack into PC for subroutine return
070	N=C (N=C ALL)	Copy C into N
0B0	C=N (C=N ALL)	Copy N into C
0F0	NC EX (N<>C ALL)	Exchange C with N
130	LDI (LDI S&X)	Load next ROM word into digits 2-0 of C
170	STK=C (PUSH ADR)	Push digits 6-3 from C onto return stack
1B0	C=STK (POP ADR)	Pop return stack into digits 6-3 of C
230	GOTOKEYS (GOTO KEY)	Load digits 4-3 of C into lower 8 bits of PC
270	DADD=C (RAM SLCT)	Use digits 2-0 of C as RAM address
2F0	DATA=C (WRITE DATA)	Write C to peripheral or memory
330	CXISA (FETCH S&X)	Load digits 2-0 of C from ROM address ADR of C
370	C=C OR A (C=C OR A)	Logical OR of C with A
3B0	C=C AND A(C=C AND A)	Logical AND of C with A
3F0	PFAD=C (PRPH SLCT)	Use digits 2-0 of C as peripheral address
000	NOP (NOP)	No operation

The following hex codes are not used by the basic HP 41C operating system: x34,x74,xB4,xF4,x18,030,1F0,2B0,100,200,300,x40,x80,xC0. Some of these hexcodes are used as instructions for HP-IL and for page switching within the HP 41CX.

CLASS 1 INSTRUCTIONS

Class 1 instructions are two-word instructions which perform an absolute address GOTO or GOSUB. The first word contains the least significant 8 bits of the address, followed by 01. The second word contains the most significant 8 bits of the address, followed by pp, which is:

pp=00 GOSUB (or ?NC XQ or GOSNC) Execute subroutine if C is clear
pp=01 GOSC (or ?C XQ) Execute subroutine if C is set
pp=10 GOLONG (or ?NC GO or GOLNC) Goto ROM address if C is clear
pp=11 GOLC (or ?C GO) Goto ROM address if C is set

For example, the hex code for GOLONG 0232 (MEMORY LOST) is:

0011 0010 01 = 0C9 for first word
0000 0010 10 = 00A for second word

CLASS 2 INSTRUCTIONS

Class 2 instructions are used for arithmetic and logical operations. Arithmetic operations are performed in hexadecimal or decimal depending on the last mode operation (SETDEC or SETHEX) executed. In DEC mode, all operations are performed on digits 0-9 (A-F work also, but not in the expected manner). In HEX mode, all operations are performed on digits 0-F. The C flag is set if the operation performed causes the most significant digit in the selected field to exceed 9 (in DEC) or F (in HEX), or if the result causes a borrow (result is less than 0).

MNEMONIC	OPERATION	HEX CODE IN FIELD AND DIGITS OF C AFFECTED							
		PT @PT	X 2-0	WPT PT-0	ALL 13-0	PQ P-Q	XS 2	M 12-3	S 13
A=0	Clear A	002	006	00A	00E	012	016	01A	01E
B=0	Clear B	022	026	02A	02E	032	036	03A	03E
C=0	Clear C	042	046	04A	04E	052	056	05A	05E
AB EX(A<>B)	Exchange A with B	062	066	06A	06E	072	076	07A	07E
B=A	Copy A into B	082	086	08A	08E	092	096	09A	09E
AC EX(A<>C)	Exchange A with C	0A2	0A6	0AA	0AE	0B2	0B6	0BA	0BE
C=B	Copy B into C	0C2	0C6	0CA	0CE	0D2	0D6	0DA	0DE
BC EX(B<>C)	Exchange B with C	0E2	0E6	0EA	0EE	0F2	0F6	0FA	0FE
A=C	Copy C into A	102	106	10A	10E	112	116	11A	11E
A=A+B	Add B into A	122	126	12A	12E	132	136	13A	13E
A=A+C	Add C into A	142	146	14A	14E	152	156	15A	15E
A=A+1	Increment A	162	166	16A	16E	172	176	17A	17E
A=A-B	Subtract B from A	182	186	18A	18E	192	196	19A	19E
A=A-1	Decrement A	1A2	1A6	1AA	1AE	1B2	1B6	1BA	1BE
A=A-C	Subtract C from A	1C2	1C6	1CA	1CE	1D2	1D6	1DA	1DE
C=C+C	Double C	1E2	1E6	1EA	1EE	1F2	1F6	1FA	1FE
C=A+C	Add A into C	202	206	20A	20E	212	216	21A	21E
C=C+1	Increment C	222	226	22A	22E	232	236	23A	23E
C=A-C	A-C into C	242	246	24A	24E	252	256	25A	25E
C=C-1	Decrement C	262	266	26A	26E	272	276	27A	27E
C=-C	Complement C	282	286	28A	28E	292	296	29A	29E
C=-C-1	9's or F's complement C	2A2	2A6	2AA	2AE	2B2	2B6	2BA	2BE
?B#0	Set C flag if B not 0	2C2	2C6	2CA	2CE	2D2	2D6	2DA	2DE
?C#0	Set C flag if C not 0	2E2	2E6	2EA	2EE	2F2	2F6	2FA	2FE
?A<C	Set C flag if A<C	302	306	30A	30E	312	316	31A	31E
?A<B	Set C flag if A<B	322	326	32A	32E	332	336	33A	33E
?A#0	Set C flag if A not 0	342	346	34A	34E	352	356	35A	35E
?A#C	Set C flag if A not = C	362	366	36A	36E	372	376	37A	37E
A SR(RSHFA)	Shift A right 1 digit	382	386	38A	38E	392	396	39A	39E
B SR(RSHFB)	Shift B right 1 digit	3A2	3A6	3AA	3AE	3B2	3B6	3BA	3BE
C SR(RSHFC)	Shift C right 1 digit	3C2	3C6	3CA	3CE	3D2	3D6	3DA	3DE
A SL(LSHFA)	Shift A left 1 digit	3E2	3E6	3EA	3EE	3F2	3F6	3FA	3FE

CLASS 3 INSTRUCTIONS

Class 3 instructions allow the program to jump up to 63 words forward or backward from its present location. The Mnemonics are GONC (or GOTO or JNC) and GOC (or JC). In assembler listings the GONC is followed by a label. In disassembled listings the GONC is followed by "*+pp" or "*-pp" which indicates a jump relative to the current instruction address ("*"). GONC branches if the C flag is clear. GOC branches if C is set.

DISTANCE	JNC-	JC-	JNC+	JC+	DISTANCE	JNC-	JC-	JNC+	JC+
*- or+01	3FB	3FF	00B	00F	*- or+02	3F3	3F7	013	017
03	3EB	3EF	01B	01F	04	3E3	3E7	023	027
05	3DB	3DF	02B	02F	06	3D3	3D7	033	037
07	3CB	3CF	03B	03F	08	3C3	3C7	043	047
09	3BB	3BF	04B	04F	0A	3B3	3B7	053	057
0B	3AB	3AF	05B	05F	0C	3A3	3A7	063	067
0D	39B	39F	06B	06F	0E	393	397	073	077
0F	38B	38F	07B	07F	10	383	387	083	087
11	37B	37F	08B	08F	12	373	377	093	097
13	36B	36F	09B	09F	14	363	367	0A3	0A7
15	35B	35F	0AB	0AF	16	353	357	0B3	0B7
17	34B	34F	0BB	0BF	18	343	347	0C3	0C7
19	33B	33F	0CB	0CF	1A	333	337	0D3	0D7
1B	32B	32F	0DB	0DF	1C	323	327	0E3	0E7
1D	31B	31F	0EB	0EF	1E	313	317	0F3	0F7
1F	30B	30F	0FB	0FF	20	303	307	103	107
21	2FB	2FF	10B	10F	22	2F3	2F7	113	117
23	2EB	2EF	11B	11F	24	2E3	2E7	123	127
25	2DB	2DF	12B	12F	26	2D3	2D7	133	137
27	2CB	2CF	13B	13F	28	2C3	2C7	143	147
29	2BB	2BF	14B	14F	2A	2B3	2B7	153	157
2B	2AB	2AF	15B	15F	2C	2A3	2A7	163	167
2D	29B	29F	16B	16F	2E	293	297	173	177
2F	28B	28F	17B	17F	30	283	287	183	187
31	27B	27F	18B	18F	32	273	277	193	197
33	26B	26F	19B	19F	34	263	267	1A3	1A7
35	25B	25F	1AB	1AF	36	253	257	1B3	1B7
37	24B	24F	1BB	1BF	38	243	247	1C3	1C7
39	23B	23F	1CB	1CF	3A	233	237	1D3	1D7
3B	22B	22F	1DB	1DF	3C	223	227	1E3	1E7
3D	21B	21F	1EB	1EF	3E	213	217	1F3	1F7
3F	20B	20F	1FB	1FF	*-40	203	207	---	---

ROM ADDRESSING

The HP 41 calculator can address up to 65536 (64K) 10-bit words of information in ROMs (Read Only Memories). This includes the operating system ROMs, HP Extension and Application modules, EPROMS (Erasable Programmable ROMs), and the ProtoCODER. These 64K words are separated into 16 "pages" of 4096 (4K) words each, numbered in hexadecimal from 0 to F. The 4096 words on each page are numbered in hex from 000 to FFF. ROM addresses are specified as PWWW where P is the page and WWW is the word number on the page.

Some of these 16 pages are preassigned for system use and cannot normally be used. Any page from 5 to F can be used to contain a ROM, EPROM or ProtoCODER. Be careful to have at most one memory assigned to a page. Assigned pages are:

Pages 0,1,2 contain the HP 41 operating system. Subroutines contained in these ROMs can be called by the user.

Page 3 is not used by the HP 41C or CV, but contains the XFUNCTIONS ROM in the HP 41CX.

Page 4 is used by the HP Service Module which helps HP diagnose calculator problems. If a ROM resides here, it is automatically executed when the calculator is turned on.

Page 5 is used by the HP Time Module. The HP 41CX also uses page 5 for another system ROM.

Page 6 is used by the HP 82143 and 82162 printers.

Page 7 is used by the HP-IL.

Pages 8,A,C,E are normally used by HP modules. The page number identifies which port the module is in. Page 8 is port 1, A is 2, C is 3, E is 4.

Pages 9,B,D,F are normally used only when the HP module is an 8K (such as the REAL ESTATE module). The second half of the ROM occupies page 9 if the module is in port 1, page B for port 2, D for 3, and F for 4. Newer HP modules may be addressed to an odd page. The Auto Execute ROM is addressed in this way which means that both it and a low-page 4K ROM can be in the same port.

XROM WORD FORMAT

Each 4K ROM contains several words used by the system in addition to the routines. The lowest block of addresses in the ROM contain the XROM number and the catalog (FAT or Function Address Table) information. Words FF4-FFA may contain GOTO instructions for routines for certain interrupt conditions. Normally these words should be zeroes.

x000 contains hex XROM number, e.g. word 6000 in the Printer (XROM 29) contains 01D.

x001 contains the number of routines contained in the catalog table, in hex.

x002-x003 and each pair of words following contains the address of the next function. These words are interpreted as tab and 0cd. t is 0 for microcode routines and 2 for user language programs. abcd is the offset from word 000 of the ROM containing the catalog table which points to the address of the first executable instruction if the routine is microcode, and the address of the first byte of the LBL if the program is user language. if a is not zero, then the catalog entry points to a location within another ROM. This catalog (Function Address Table) information is followed by two words containing 000.

x...-xFF3 contain programs and routines.

xFF4 contains interrupt instruction executed during a PSE loop.

xFF5 contains interrupt instruction executed after each line.

xFF6 contains interrupt instruction executed on wakeup with no key pressed.

xFF7 contains interrupt instruction executed when the calculator is turned off.

xFF8 contains interrupt instruction executed when a peripheral flag is set.

xFF9 contains interrupt instruction executed on wakeup by pressing the ON key.

xFFA contains interrupt instruction executed on MEMORY LOST.

xFFB-xFFE contains the ROM identification and revision number.

xFFF contains the ROM checksum. This is used to verify that the ROM contents are correct. To calculate this checksum, see "ROM Checksum".

TONES IN MICROCODE

The HP 41 uses a short microcode routine located at address 16DD to control the bender for all TONE operations. Both the frequency and the duration of the tone are software controlled and are predictable given the cycle time of the calculator. The system routine accepts 2 digits of data to specify the tone. The left-most bit is chopped off and interpreted as INDIRECT if it is 1. TONE instructions appear in memory as 9F ab where a is normally 0 and b is 0-9 unless created syntheticly. The duration of the tone is determined by the contents of ROM word 16F2 + ab. This value is decremented in a loop as the tone is being heard until it becomes less than zero, which terminates the tone. The frequency is determined by b. HP intended only ten tones to be used but the TONE routine will look up ROM data for all 128 tones. This explains why some of the synthetic tones changed in duration when HP updated ROM 1.

To use the bender, store 00 in the F (or T) register and store hex FF in ST. Tones are created by turning F on and off, i.e., by swapping F and ST. The number of swaps defines the duration of the tone. The number of instructions between swaps defines the frequency. The duration and frequency will also vary depending on the cycle time of your calculator. Non speeded-up calculators have a cycle time of about 158 microseconds per microcode instruction.

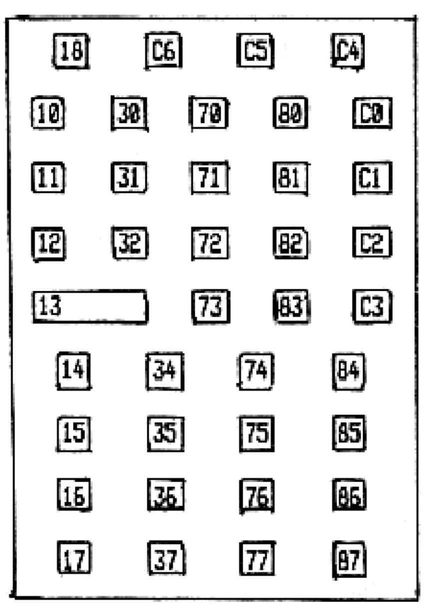
Example: TONE 9 (9F 09) has a period of 3 processor cycles per loop * 2 loops per tone cycle * 158 microseconds per cycle = .000948 seconds. Then the frequency is 1/.000948 = 1055 hertz. To determine the duration, convert the ROM data word at 16F2 + 09 = 16FB which is 215 hex, to decimal, then add one since the looper decrements the number until it is LESS than zero. This number (534 decimal) is the number of times that the bender is flopped using the F and ST registers. The duration of TONE 9 is 534 loops * 3 cycles per loop * .000158 seconds per cycle = .253 seconds.

Tones with a frequency between existing tones can be created by varying the ratio of on-to-off time of the bender. In the above example for TONE 9, the bender is on for 3 cycles then off for 3 cycles. Try the same loop but leaving the bender on for 2 cycles and off for 4.

After using the bender, you should store a 00 in the F register. If you do not, then you will get a high pitched tone whenever the processor is running or when a key is pressed.

KEYCODES RETURNED BY "C=KEYS"

The following hex keycodes are returned in digits 4-3 of C when C=KEYS is executed. If no key was pressed then 00 is returned. All other digits of C are unaffected.



ROM CHARACTER TABLE

The HP 41 recognizes two distinct character sets: modified ASCII (listed in the HP 41 hex tables) and the ROM character set. The ROM character set is used for most internal operations including coding ROM function names. The colon (3A) is displayed as a boxed star. The comma (2C) and period (2E) display as left- and right-facing geese respectively when used in a function name or in the display.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	(	=	)	?
4	⊦	a	b	c	d	e	‾	ᵀ	Γ	X̄	x̄	✱	μ	≠	Σ	∠

FUNCTION NAMES, PROMPTING, AND NON-PROGRAMMABILITY

When a function is executed, the operating system checks the ROM words containing the first two characters of the function name and the two words immediately following. The catalog table entry for a microcode function (both mainframe and XROM functions) points to the first word of executable code. The function name is listed in reverse order immediately preceding the first word of executable code. For example, CLA (hex 87) has a catalog entry at 1487 of 0D1 which means that the first executable word of CLA is at 10D1. The listing for CLA is:

```
10CE 081 A
10CF 00C L
10D0 003 C
10D1 04E C=0
10D2 168 REGN=C 5(M)
10D3 1A8 REGN=C 6(N)
10D4 1E8 REGN=C 7(O)
10D5 228 REGN=C 8(P)
10D6 3E0 RTN
```

This shows how the function name is listed in reverse order. The last character of the function name is identified by adding hex 80 to the ROM character code. For CLA, add 80 to the code for A (001) to get 081 at address 10CE. The top two bits in the first two characters of the function name can be used to provide a prompt; these bits are zeroes for CLA since CLA requires no prompt.

EXAMPLE	ADD HEX TO 1CHR	2CHR	PROMPT TYPE ACCEPTED
CLA	000	(any)	No prompt
CLP	100	000	Accept alpha (null input valid)
SIZE	100	100	Accept 3 digits (4 with EEX pressed)
	100	200	Accept non-null alpha
CAT	100	300	Accept 1 digit or IND or IND ST
STO	200	000	Accept 2 digits, IND, IND ST, or ST
RCL	200	100	Accept 2 digits, IND, IND ST, or ST
FS?	200	200	Accept 2 digits, IND, or IND ST
	200	300	Accept 2 digits, IND, or IND ST
LBL	300	000	Accept non-null alpha or 2 digits
XEQ	300	100	Accept non-null alpha, 2 digits, IND, or IND ST
	300	200	Accept non-null alpha or 2 digits
GTO	300	300	Accept non-null alpha, 2 digits, IND, IND ST, .xxx

Although STO (200,000) and RCL (200,100) appear to be the same, they are not. If you use the STO combination, the calculator will also accept + - * or / to change the instruction to ST+, etc. Your intended instruction will change to ST+ if you use this combination, and will not execute as you expect. This will also happen for the LBL, XEQ, and GTO combinations.

Following are three examples. VIEW prompts will accept 2 digits, IND, IND ST, or ST. COPY will accept an alpha string, including a null string. TONE will accept 1 digit, IND, or IND ST.

12D2 097 W	1105 099 Y	12CC 085 E
12D3 005 E	1106 010 P	12CD 00E N
12D4 109 I	1107 00F O	12CE 30F O
12D5 216 V	1108 103 C	12CF 114 T

The operating system examines these ROM bits and executes a prompt (if the appropriate bits are set) before the function is executed. If the prompt accepts an alpha string, the input data is loaded into the Q register, right justified, in reverse order, in ASCII. For example, ASN "COPY" loads 00 00 00 59 50 4F 4C into Q before the ASN routine is executed. If the prompt is numeric, the input data is loaded into the A register in binary. A numeric input of 55 returns 00 00 00 00 00 00 37 in A. Add 80 hex for IND: IND 55 returns 00 00 00 00 00 00 B7 in A.

In PRGM mode two other ROM words of a microcode function are examined by the operating system (they are ignored in RUN mode). If the first executable word is 000 then the function is non-programmable. This means that it executes rather than being entered as a program line. SIZE, ASN, and CLP are non-programmable functions. If the first two executable words of a microcode function are both 000 then the function is non-programmable-immediately-executable (NPIE). This means that no function name is displayed and that the function will not NULL. The function is executed when the key is pressed rather than when the key is released. PRGM, SHIFT, and back-arrow are NPIE functions. If you hold the key to which an NPIE function is assigned, it will be executed repeatedly unless the function checks for key release.

ROM CHECKSUM

If you wish to copy your ProtoCODER into an EPROM for permanence, you should calculate the checksum to store in word FFF of your EPROM. To do this, execute CHKSUM in the PCODER-1A EPROM set.

The ROM checksum is calculated by adding all 10-bit words together. Each time a carry or overflow into the 11th bit occurs, add 1 into your running sum. This is called a wraparound carry. Subtract 1 from your final sum to get the checksum value.

DISPLAY PROGRAMMING

To operate the display on the HP 41, you must select the display and deselect the RAM. To do this, execute the system routine (GOSUB 07F6) or peripheral select (PFAD=C) with C digits 2-0 containing 0FD then RAM select (DADD=C) with C digits 2-0 containing 010(hex). After selecting the display, you can write data from the C register into the display or annunciators and read data from the display into the C register.

Each of the 12 character positions of the display is coded with 9 bits. The leftmost bit (bit 8), if set, specifies that bits 3-0 contain a special character in for 4 of the ROM CHARACTER TABLE. If bit 8 is set and bits 5-4 contain anything but zero, a space will be displayed. Bits 7-6 define the punctuation field of the character: 00 is no punctuation, 01 is a period, 10 is a colon, and 11 is a comma. Bits 5-4 specify which row (0-3) the displayed character is from in the ROM CHARACTER TABLE, and bits 3-0 specify the character within the row.

Data can be read or written to the left or right end of the display. Data is pushed onto the display when written. The rest of the characters are shifted to make room for the incoming data. When data is read, it is pulled off the end of the display and rotated back into the other end.

Data can be read or written to several fields of the display (bits 8-0, 7-0, 7-4, 3-0, or bit 8 alone) in blocks of 1, 4, 6 or 12 characters. Only the specified field is modified: the remaining bits are unchanged. When 4 or 6 characters are read, the character on the end of the display becomes the least significant in C. During a write, the rightmost character in C is written first.

The annunciators are read and written from digits 2-0 of C. The bits are numbered as (high order to low order):

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

The following table shows all display instructions and their actions.

MNEMONIC	HEX	ACTION	#CHARS	BITS OF C	DIGITS OF C PER CHAR	ROTATION
DATA=C	2F0	WRITE	1 BIT IN C PER ANNUNCIATOR			NONE
C=REGN T	038	READ	12	3-0	1	LEFT
REGN=C T	028	WRITE	12	3-0	1	RIGHT
C=REGN Z	078	READ	12	7-4	1	LEFT
REGN=C Z	068	WRITE	12	7-4	1	RIGHT
C=REGN Y	0B8	READ	12	8	1	LEFT
REGN=C Y	0A8	WRITE	12	8	1	RIGHT
C=REGN X	0F8	READ	6	7-0	2	LEFT
REGN=C X	0E8	WRITE	6	7-0	2	RIGHT
C=REGN L	138	READ	4	8-0	3	LEFT
REGN=C L	128	WRITE	4	8-0	3	RIGHT
C=REGN M	178	READ	1 BIT IN C PER ANNUNCIATOR			NONE
REGN=C M	168	WRITE	6	7-0	2	LEFT
C=REGN N	1B8	?????				
REGN=C N	1A8	WRITE	4	8-0	3	LEFT
C=REGN O	1F8	READ	1	3-0	1	RIGHT
REGN=C O	1E8	WRITE	1	3-0	1	RIGHT
C=REGN P	238	READ	1	7-4	1	RIGHT
REGN=C P	228	WRITE	1	7-4	1	RIGHT
C=REGN Q	278	READ	1	8	1	RIGHT
REGN=C Q	268	WRITE	1	8	1	RIGHT
C=REGN +	2B8	READ	1	3-0	1	LEFT
REGN=C +	2A8	WRITE	1	3-0	1	LEFT
C=REGN a	2F8	READ	1	7-4	1	LEFT
REGN=C a	2E8	WRITE	1	7-4	1	LEFT
C=REGN b	338	READ	1	7-0	2	RIGHT
REGN=C b	328	WRITE	1	7-0	2	RIGHT
C=REGN c	378	READ	1	7-0	2	LEFT
REGN=C c	368	WRITE	1	7-0	2	LEFT
C=REGN d	3B8	READ	1	8-0	3	RIGHT
REGN=C d	3A8	WRITE	1	8-0	3	RIGHT
C=REGN e	3F8	READ	1	8-0	3	LEFT
REGN=C e	3E8	WRITE	1	8-0	3	LEFT

EXAMPLES OF PROGRAMS

"X=1?" sets up a 1 in the C register then branches to the system ROM routine that makes the comparison.

```
0BF ?
031 1
03D =
018 X
04E C=0 ALL initialize C
35C PT=12   point to most significant digit of C
050 LC 1    load a 1 so C now contains floating point 1
055 GOLONG  jump to "X=Y?" native comparison routine
05A   1615
```

"+1" is a good example of the speed of microcode compared to user code. When you execute "+1" the calculator starts counting from 0, incrementing by 1 each loop until any key if pressed. The resulting total is displayed and stored in X. By comparison, enter the user program: LBL 01, +, GTO 01, then fill the stack with all 1s (1, ENTER, ENTER, ENTER) and run it. "+1" will run about 125 times faster than the user language program. (try it)

```
0B1 1
02B +
04E C=0 ALL    initialize counter
2A0 SETDEC     select decimal mode
23A C=C+1 M    increment counter
3CC CHK KB     key pressed?
3F3 GONC *-2   if not, loop back to increment
130 LDI        load exponent of 9
009 CON 9
10E A=C ALL    put the (non-normalized) total in A for left shifting
35C PT=12      set pointer to most significant digit of mantissa
1A6 A=A-1 X    decrement exponent
3FA A SL M     shift mantissa left until MSD not zero
342 ?A#0 PT    if still zero, then
3EB GONC *-3   go back and shift and decrement EXP again
0AE AC EX ALL  get the normalized version in C
0E8 REGN=C 3(X)   store in user register X
3C8 RST KB     wait until
3CC CHK KB          key is
3F7 GOC *-2             released then
3E0 RTN        return
```

"GOOSE" appends a left-facing goose to the display. Use as a program line in the program: CLA, AVIEW, GOOSE, LBL 01, 0, ENTER, 0, GTO 01. And they said it was impossible!

```
085 E
013 S
00F O
00F O
007 G
3C1 GOSUB      execute system routine to select display
0B0    2CF0
130 LDI        load left-goose hex code
02C CON 2C
3A8 REGN=C d   write to left end of display
149 GOSUB      execute system routine to deselect display
024    0952
3E0 RTN        and go back
```

ADDITIONAL NOTES

SYSTEM STACK: The microprocessor in the HP 41 uses an address stack to keep track of subroutine calls. This stack will hold 4 address entries. Each time a GOSUB occurs, the address of the second word of the GOSUB instruction is "pushed" onto the stack - it becomes the lowest entry and the other entries are moved up by one position. If there were already four addresses in the stack, the top one is lost. Whenever a RTN occurs, the bottom entry of the stack is copied into the PC register and all other entries are moved down by one position and a zero is moved into the top stack position. When a SPOPND occurs the stack is dropped by one position and the bottom address is lost. When a C=STK occurs the bottom address is copied into digits 6-3 of C and the stack is dropped by one position. When a STK=C occurs the stack is lifted by one and digits 6-3 of C are copied into the bottom position as an address.

SYSTEM STATUS: There are three major modes of the HP 41C: sleep, standby, and active or running. In sleep mode the calculator is turned off. In standby mode the calculator is turned on but is not executing any microcode. In active mode the caculator is running microcode. The system ROM (page 0) contain code to differentiate between sleep and standby modes by the condition of the C flag when address 0000 is executed. Whenever the calculator is running a RPN (user language) program, each RPN statement is **interpreted** by microcode then executed as a pre-set sequence of instructions. The HP 41 processor does not understand RPN without translation by the operating system.

RELOCATION: When you write a microcode routine to be contained in an external ROM (or EPROM or ProtoCODER) you should make it relocatable. This means that a user can plug your ROM into any port and it will still work. If you use absolute GOLONGs or GOSUBs to access routines within the ROM then it will not function properly if you change its page addressing. There are several routines in the operating system (pages 0,1,2) to allow you to do absolute jumps or executes within your ROM. The most general purpose of these system routines is located at 00D7-00D9. To use it, put a GOSUB 00D7 in your routine. after returning from this routine digits 6-3 of C will contain the absolute address of the second byte of your GOSUB instruction. You can then modify digits 5-3 to contain any address within your ROM then execute a GOTOC. Digit 6 contains the page number where the ROM is plugged in.

MOVING USER CODE PROGRAMS TO THE ProtoCODER2

The PCODER-1A EPROM set provides a program (COPYPC) to copy a user-code program into your ProtoCODER. If you wish to do it manually, or you want to modify what has already been entered, the following lists all steps necessary.

Each ProtoCODER word is 10 bits, therefore it will hold 1 8-bit byte containing a user-code instruction. The leftmost two bits are used to signify the first byte of any instruction, and to mark the beginning and end of the program.

The first two words, which precede the actual program, are of the form rrr and 2b0. rrr specifies the number of registers needed to copy the program into user memory, including the last register which may be only partially full. b specifies the number of bytes (1-7) to be loaded into the first register. When COPY is executed, it copies b bytes into the first register then copies an even number of registers of bytes, thus insuring that the END within the copied program will be on a register boundary: in the last three bytes of the last register. The program in ROM can be made PRIVATE by modifying the 2b0 word.

The user-code program starts immediately following these two COPY parameter words. Normally it would start with a global LBL (but this is not required). Global labels can be copied exactly as stored in RAM. The catalog (Function Address Table) information at the beginning of the ProtoCODER image must be changed to reflect one or more new entries. The two-word FAT entry (see "XROM Word Format") is 2ab and 0cd where abcd is the offset from word 000 of the page containing the ProtoCODER to the first byte of the corresponding global label. Note that abcd can specify an address in any other ROM also. Each global label within a program should be entered into the FAT if the user wishes to access that label directly.

Instruction hexcodes (other than the exceptions listed below) are copied exactly as they are listed in RAM. The first two bits of the ROM word are set to 01 to signify the first byte of an instruction, and set to 00 for continuing bytes of a multibyte instruction.

The last three bytes of the program are the END. The third byte of the END is coded as 2pp where pp is the set of parameters normally associated with an END in user RAM. As suggested by Larry Lavins, the easiest code to use here is 220 (=unpacked, uncompiled, nonprivate, .END.).

All direct local GTO and XEQ functions should be stored with accompanying jump distances. If you do not compute the jump distance for two-byte GTOs (or if the distance is greater than 127 bytes), store the jump distance as 0 so that the system will search for the specified label. Jump distances must be specified for three-byte GTOs and XEQs.

The jump distance for two-byte GTOs is stored in the second byte of the instruction. Count the number of bytes from the second byte of the GTO to the byte immediately preceding the LBL. Convert this number to binary, and add 128 if the jump is forwards (to a higher program line number), to get the data byte to be stored as the second byte of the GTO. Note that in ROM, all numeric labels are non-functional (unless a label search is necessary), so that the jump distance in the GTO need not necessarily point to the corresponding LBL. Therefore, the LBL can be completely removed. This is not recommended since the program will not function properly when COPYed.

The three-byte GTOs and XEQs are coded as:

Dd dd ll for GTO, and

Ed dd ll for XEQ.

ddd is the number of bytes (jump distance) from the first byte of the instruction to the byte immediately preceding the corresponding LBL. ll is the hex label number, plus 128 if the jump is forwards (to a higher program line number). The jump distance must be computed: If you store 0 as the jump distance, the program will continue with the next program line.

USEFUL ROM ENTRY POINTS

The following are a partial list of some useful entry points into the system ROMs. These entry points are in the HP 41C and CV, but probably remain the same in the CX. Each entry point is followed the page number within the VASM listing ("s" is for page number in the supplement), the absolute address of the function, the hexcodes to call the subroutine (for GOSUB NC), and a brief note about what the routine does. Consult the VASM listings for entry and exit parameters.

NAME	PAG	ADDR	HEXCODE	PURPOSE
ABTSEQ	100	0D12	049,034	Abort partial key sequence
AD2-10	162	1807	01D,060	Floating point addition
ADRFCH	002	0004	011,000	Get user register
ANNOUT	056	075C	171,01C	Output flags to annunciators
ARGOUT	310	2C10	041,0B0	Output ALPHA register to display
ASCLCD	311	2C5D	175,0B0	Output ASCII character to display
ASCTBL	309	2C00	---,---	Table of special display characters
ASRCH	263	26C5	315,098	Search for alpha label
BCDBIN	020	02E3	38D,008	Floating point to hexadecimal conversion
CLLCDE	314	2CF0	3C1,0B0	Enable and clear display
CPGMHD	050	067B	1ED,018	Get current program head
DECAD	288	29C7	31D,0A4	Decrement program address
DV2-10	165	1898	261,060	Floating point division
ENCP00	074	0952	149,024	Enable chip 0 - user status registers
ENLCD	059	07F6	3D9,01C	Enable display
FLINK	284	2928	0A1,0A4	Find global program links
GENLNK	236	239A	269,08C	Generate program link
GENNUM	047	05E8	3A1,014	Hexadecimal to floating point conversion
GETLIN	s02	1419	065,050	Get current line number
GETPC	285	2950	141,0A4	Get current program counter
GOL0	238	23D0	341,08C	Goto within first 1K block (followed by offset)
GOL1	238	23D9	365,08C	Goto within second 1K block (followed by offset)
GOL2	238	23E2	389,08C	Goto within third 1K block (followed by offset)
GOL3	239	23EB	3AD,08C	Goto within fourth 1K block (followed by offset)
GOLONG	118	0FDA	369,03C	Goto within current 1K block (followed by offset)
GOSUB	118	0FDE	379,03C	Gosub within current 1K block (followed by offset)
GOSUB0	238	23D2	349,08C	Gosub within first 1K block (followed by offset)
GOSUB1	238	23DB	36D,08C	Gosub within second 1K block (followed by offset)
GOSUB2	238	23E4	391,08C	Gosub within third 1K block (followed by offset)
GOSUB3	239	23ED	3B5,08C	Gosub within fourth 1K block (followed by offset)
GTBYT	287	29B0	2C1,0A4	Get byte from RAM or ROM
GTLINK	228	224E	139,088	Get global program link
GTRMAD	067	0800	001,020	Get XROM function entry address
INBYT	289	29E6	399,0A4	Insert byte into RAM
INCAD	288	29CF	33D,0A4	Increment pointer address
INCAD2	288	29D3	34D,0A4	Increment pointer address twice
INSLIN	290	29F4	3D1,0A4	Insert program line
INSSUB	237	23B2	2C9,08C	Prepare for insertion into program
LEFTJ	301	2BF7	3DD,0AC	Left justify display

NAME	PAG	ADDR	HEXCODE	PURPOSE
MESSL	059	07EF	3BD,01C	Append message to display (message data follows GOSUB)
MP2-10	163	184D	135,060	Floating point multiply
NEXT	109	0E50	141,038	Enter standby mode
NOSKP	s13	1619	065,058	Execute next line or say "YES"
NULT#3	110	0E7C	1F1,038	Test for null then execute instruction
NXTBYT	315	2D07	01D,0B4	Get next byte in RAM or ROM
PACKE	217	2002	009,080	Pack then say "TRY AGAIN" and return to system
PACKN	217	2000	001,080	Pack then return to caller
PCTOC	007	00D7	35D,000	Get address of GOSUB
PTBYTA	234	2323	08D,08C	Put byte into RAM
PTLINK	234	231A	069,08C	Save global program link
PUTPC	234	2337	0DD,08C	Save program counter
ROMHED	050	066A	1A9,018	Get program head in ROM
RSTKB	005	009B	261,000	Reset and debounce keyboard
RSTSEQ	024	0384	211,00C	Reset some status bits
SEARCH	249	2433	0CD,090	Search for numeric label
SKP	s13	162E	0B9,058	Skip next line or say "NO"
SKPLIN	295	2AF9	3E5,0A8	Get address of next program line
STMSGF	024	037E	1F9,00C	Set message flag
TOGSHF	208	1FE5	395,07C	Toggle SHIFT flag
UPLINK	228	2235	0D5,088	Move up one global program link

The above hexcodes can be changed from a GOSNC to:

GOSC by adding 1 to the second word;
GOLNC by adding 2 to the second word;
GOLC by adding 3 to the second word.

Subroutine calls to GOL0,GOL1,GOL2,GOL3,GOLONG,GOSUB,GOSUB0,GOSUB1,GOSUB2,GOSUB3 are followed by one word containing an offset from the beginning of the specified quad (1K within the current ROM). For example, GOSUB2 followed by 342 hex, called from address pxxx will perform a GOSUB to p000 + 800 ("gosub to quad 2") + 342 = pB42.

A subroutine call to MESSL is followed by one or more data words giving the characters to be output. Add 200 hex to the last character.

A subroutine call to NEXT causes the calculator to go into standby mode, with the display drivers waiting for a keypress. If OFF is pressed, the calculator is turned off. Otherwise the system returns to the word following your GOSUB (if backarrow was pressed) or to the second word following your GOSUB (if any other key was pressed). The keycode is returned in digits 2:1 of N and is the key assignment keycode minus 1. Shift, if set, is included in the keycode. In addition, ST will reflect the following:

FSET 3 if a numeric key was pressed,
FSET 4 if a row 1 or row 2 key was pressed,
FSET 5 if ALPHA was pressed, and
FSET 6 if SHIFT was pressed.

Call NEXT with G=0 and the display non-blank.

The hexcode for GOSUB 1078, which is the RTN within the ABS function - used to write data to the ProtoCODER2, is 1E1,040.