PPC

MC EPROM SET

CONTENTS

MC EPROM SET

for use with

Mountain Computer MC00506A EPROM Programmer

1). This software package allows HP-41 formatted EPROMs to be programmed using the Mountain Computer MC00506A HPIL EPROM Programmer. The ROM to be programmed must reside in the ROM address space of the HP-41. This can be a ROM mouule, another EPROM set (using an EPROM box), or a 4k block of RAM (in an MLDL type of device).

2). The EPROM set and its source code was placed in the public domain July 9, 1983 by Paul Lind, and may be copied, distributed, or modified in any way you see fit.

3). This software package is not an official product of Mountain Computer, Inc., and must be considered NOMAS (Not Manufacturer Supported).

4]. Copies of this EPROM set are available through PPC. Cost for manual and EPROM set is $20.00.* (1-2716, 1-2732). Order From:

PPC - POB 9599
Fountain Valley, CA 92728-9599
USA

*USA Price. Add shipping and 6% sales tax if shipped to a California address. Shipping weight is 6 oz.

INSTRUCTIONS FOR
MOUNTAIN COMPUTER EPROM SET

*** EPROM FUNCTIONS ***

BACKGROUND ON HP-41 EPROMS

The HP-41 ROM space uses a 10 bit word size. To store these ROM words in standard EPROMs (byte-wide), it was decided to use separate EPROMs for the lower eight bits and the upper two bits. Each byte of the L8 EPROM contains the lower eight bits for one of the 10 bit ROM words. However, to conserve space, each byte of the U2 EPROM contains the upper two bits for FOUR consecutive 10 bit ROM words. Therefore, burning a 4k HP-41 ROM block requires 4k of L8 EPROM space, but only 1k of U2 EPROM space.

Since a 2716 (the minimum size U2 EPROM) is 2k long, there is room for the U2 information from two separate HP-41 4k blocks. If a 2732 is used as the U2 EPROM, there is room for the U2 information from four separate HP-41 ROM blocks. So that the user does not have to remember EPROM addresses, the programs that burn or read EPROMs ask for a section number. For example, a section number of 0 specifies that the lowest 1k area in a U2 EPROM. L8 EPROMs also use section numbers, since a 2764 may store the L8 information from two separate 4k blocks.

BURNL

FUNCTION: Program an L8 EPROM from an existing 4k HP-41 ROM block, using the MC00506 EPROM Programmer. The first MC00506 found on the loop starting with the primary device is used. The program may be terminated by pressing the "ON" key.

INPUT: Y REG - The section number to burn (0 - 1).
X REG - The ROM block to burn from (0 - 15).

OUTPUT: NONE

ERRORS:	DATA ERROR	Block number) 15 or section number) 1.
	ALPHA DATA	Block number or section number is type ALPHA rather than NUMERIC.
	NO HPIL	HPIL module is not plugged in.
	NO MC00506A	EPROM Programmer not on loop or loop in MANIO and MC00506A not the primary device.
	ERASE ERR	Specified section of EPROM not erased.
	BURN ERR	Byte not programmed successfully.
	LOW BAT ERR	The HP-41 battery voltage dropped too low to continue reliable operation.
	TRANSMIT ERR	Loop not operating correctly.

BURNU

FUNCTION: Program a U2 EPROM from an existing 4k HP-41 ROM block, using the MC00506A EPROM Programmer. The first MC00506A found on the loop starting with the primary device is used. The program may be terminated by pressing the "ON" key.

INPUT: Y REG - The section number to burn (0 - 3).
X REG - The ROM block to burn from (0 - 15).

OUTPUT: NONE

ERRORS:

DATA ERROR	Block number) 15 or section number) 3.
ALPHA DATA	Block number or section number is type ALPHA rather than NUMERIC.
NO HPIL	HPIL module is not plugged in.
NO MC00506A	EPROM Programmer not on loop or loop in MANIO and MC00506A not the primary device.
ERASE ERR	Specified section of EPROM not erased.
BURN ERR	Byte not programmed successfully.
LOW BAT ERR	The HP-41 battery voltage dropped too low to continue reliable operation.
TRANSMIT ERR	Loop not operating correctly.

GETEP

FUNCTION: Reads the contents of an L8 and a U2 EPROM into a 4k block of RAM. The EPROMs are read from the first MC00506A on the loop starting with the primary device. The user is first prompted to "RDY L8 EPROM". When the EPROM (and its personality module) are ready, any key may be pressed to start the read. The "ON" key may be pressed instead to halt the program at this point. When the L8 EPROM has been read, the user will be prompted with "RDY U2 EPROM". When the U2 EPROM is ready, press any key to start the read. As before, the "ON" key will stop the program.

INPUT: Z REG - The section number to read of the L8 EPROM (0 - 1).
Y REG - The section number to read of the U2 EPROM (0 - 3).
X REG - The block number to read into (0 - 15).

OUTPUT: NONE

ERRORS:

DATA ERROR	Block number (X)) 15 or L8 section number (Z)) 1 or U2 section number (Y)) 3.
ALPHA DATA	Block number or section number is type ALPHA rather than NUMERIC.

NO HPIL	HPIL module is not plugged in.
NO MC00506A	EPROM Programmer not on loop or loop in MANIO and MC00506A not the primary device.
TRANSMIT ERR	Loop not operating correctly.

*** MASS STORAGE FUNCTIONS ***

BACKGROUND ON HP-41 ROM STORAGE

The programs "ROM)REG" and "REG)ROM" were developed and published in the PPC Calculator Journal (V9N3P40) to facilitate the transfer of ROM programs between HP-41 users. They allow ROM words to be transferred to and from the data registers of the HP-41, so that any mass storage device may be used to permanently store them.

Since HP-41 object code is not position independent, the programs use a header code that allows ROM words to be easily loaded back at the address from which they were saved. This allows HP-41 assembly language programmers to exchange single routines on magnetic cards, for instance.

The most common use of ROM)REG and REG)ROM is to save an entire 4k block on cassette tape. Because 4k of ROM code will not fit into the HP-41 data registers all at once, the 4k block has to be 'ROM)REGed' in several smaller pieces, which are each written to tape as DATA files.

Since this process is rather slow, and requires the use of almost all the HP-41 memory as a temporary buffer, this EPROM set includes two new routines: READROM and WRTROM. These routines read and write directly from an MLDL type device to the HPIL mass storage devices, and do not use any memory within the HP-41. They are fully compatible with the 'ROM)REG' standard, however. They store a ROM in four separate 1k blocks within an 824 register DATA file.

ROM)REG
REG)ROM

See the original ROM)REG article in PPC Calculator Journal V9N3P40.

READROM

FUNCTION: Load a 'ROM)REG' format ROM image file from an HP-IL mass storage device into a block of MLDL RAM. If the loop is in AUTOIO mode, all mass storage devices on the loop will be searched (starting with the primary device) to find the file. If the loop is in MANIO, only the current primary device will be used. The specified block is not checked to make sure that it is really RAM -- this is up to the user.

INPUT: X REG - The block number (0 - 15) to load into.
ALPHA - Filename of the ROM image file.

OUPUT: NONE

ERRORS:	DATA ERROR	Block number) 15.
	ALPHA DATA	Block number is type ALPHA rather than NUMERIC.
	NO HPIL	HPIL module is not plugged in.
	NO DRIVE	An HPIL mass storage device is not on the loop.
	NO MEDM	Medium is not installed properly.
	MEDM ERR	Medium improperly installed, worn out, or damaged.
	DRIVE ERR	Medium stalled or bad drive.
	NAME ERR	ALPHA register is empty.
	FL NOT FOUND	Named file is not on the medium.
	FL TYPE ERR	Specified file is not a DATA file.
	FORMAT ERR	The specified file is not in the 'ROM)REG' format or is not exactly 824 registers long.
	TRANSMIT ERR	Loop not operating correctly.

WRTROM

FUNCTION: Store the contents of an HP-41 4k ROM block to a mass storage file in the 'ROM)REG' format. If the named file exists and is of the correct type and length, it is overwritten with the new ROM image (unless the file is SECURED). If the file does not exist, it is automatically created.

INPUT: X REG - The block number (0 - 15) to write.
ALPHA - The filename of the new file.

OUTPUT: NONE

ERRORS:	DATA ERROR	Block number) 15.
	ALPHA DATA	Block number is type ALPHA rather than NUMERIC.
	NO HPIL	HPIL module is not plugged in.
	NO DRIVE	An HPIL mass storage device is not on the loop.
	NO MEDM	Medium is not installed properly.
	MEDM ERR	Medium improperly installed, worn out, or damaged.
	DRIVE ERR	Medium stalled or bad drive.
	NAME ERR	ALPHA register is empty.
	DUP FL NAME	The specified file already exists on the medium, but is not an 824 register DATA file (which would be overwritten).
	FL SECURED	A file of the specified name and of the correct size and type exists on the medium, but cannot be overwritten because it is secured.
	DIR FULL	The file cannot be created because the directory is full.
	MEDM FULL	The file cannot be created because the medium does not have enough storage space left.

TRANSMIT ERR Loop not operating correctly.

*** OTHER FUNCTIONS ***

RCOPY

FUNCTION: Copy any 4k ROM into a block of RAM. The destination block is not checked to assure that it is really RAM, this is up to the user.

INPUT: Y REG - The block number (0 - 15) of the source ROM.
X REG - The block number (0 - 15) of the destination ROM (RAM).

OUTPUT: NONE

ERRORS: DATA ERROR   Either block number (X or Y) > 15.
ALPHA DATA   Either block number (X or Y) ALPHA DATA.

PCAT

FUNCTION: Displays a CAT 2 listing starting with a particular port number. Numbers 1 through 4 start the CAT listing with that port (ROM addresses 8000, A000, C000, and E000, respectively) and continue through all modules in higher numbered ports. Numbers 5, 6, or 7 start the Catalog at ROM addresses 5000, 6000 , or 7000 which correspond to the addresses of the Timer, Printer, and HPIL modules.

INPUT: Single digit 1-7 at prompt.

OUTPUT: NONE

ERRORS: NONE

KASN

FUNCTION: Makes 2 byte key assignments in a similar manner to "1K" of the PPC ROM. Differences are that an assignment will always overwrite any existing assignment to a key, and that if both assignment bytes are zero, the key will have any present key assignment cleared.

INPUT: Z REG - First byte of assignment (0-255)
Y REG - Second byte of assignment (0-255)
X REG - Keycode (Standard row/column)

OUTPUT: NONE

ERRORS: DATA ERROR   Either assignment byte greater than 255 or invalid keycode.

PACKING, TRY AGAIN   Not enough room to make assignment.

EPROM LISTING

```
Puget Sound Programing HP-41 Assembler: mceprom.a
0000             start      equu                       $A000
A000                        org      start

A000 00F                    con      15                XROM #
A001 00B                    con      11                # of functions

A002 000                    fat      romname
A003 099
A004 000                    fat      burnl
A005 09F
A006 000                    fat      burnu
A007 0A6
A008 002                    fat      getep
A009 0F6
A00A 003                    fat      kasn
A00B 0B9
A00C 004                    fat      pcat
A00D 00C
A00E 001                    fat      rcopy
A00F 051
A010 001                    fat      readrom
A011 06D
A012 001                    fat      regrom
A013 0AF
A014 002                    fat      romreg
A015 096
A016 002                    fat      wrtrom
A017 003

A018 000                    con      0
A019 000                    con      0

A090                        org      $A090

                 ************************************************************
                 * ROM name                                                 *
                 ************************************************************
A090 08D                    con      $0D
A091 00F                    con      $0F
A092 012                    con      $12
A093 010                    con      $10
A094 005                    con      $05
A095 020                    con      $20
A096 003                    con      $03
A097 00D                    con      $0D
A098 02D                    con      $2D               name: -MC EPROM
A099 3E0         romname    rtn



                 **************************************************************
                 * BURNL - Burn an L8 EPROM image on the MC00506A             *
                 *       - INPUT: X - The HP41 ROM block to burn (0-15)       *
                 *       -        Y - The 4k section of the L8 to burn        *
                 *                       (used for upper/lower half of 2764)  *
                 **************************************************************

A09A 08C                    con      $8C
A09B 00E                    con      $0E
A09C 012                    con      $12
A09D 015                    con      $15
```

```
A09E 002               con     $02            name: BURNL

A09F 3C4    burnl      st=0
A0A0 043               jnc     burnlu10


            ******************************************************************
            * BURNU - Burn a U2 EPROM on the MC00506A                        *
            *       - INPUT: X - The HP41 ROM block to burn (0-15)           *
            *       -        Y - The 1k EPROM section to burn (0-4)          *
            ******************************************************************

A0A1 095               con     $95
A0A2 00E               con     $0E
A0A3 012               con     $12
A0A4 015               con     $15
A0A5 002               con     $02            Name: BURNU

A0A6 3C4    burnu      st=0                   clear flags
A0A7 208               setf    2              indicates U2 burn

            * Display function name, do PILTST, and find the MC00506
A0A8 10E    burnlu10   a=c     all            move entry address to A
A0A9 36D               xqrel   dispname       display name with ':'
A0AA 0BC
A0AB 0A0
                       xqrel   findmc         PIL test and MC00506 find
A0AC 36D
A0AD 0BC
A0AE 126

            * Use the block number in X to make address and counter
A0AF 36D               xqrel   bcdbinXF       X -> bin, max value is 15
A0B0 0BC
A0B1 0E8
A0B2 13C               rcr     8              left 6, make it an adr
A0B3 266               c=c-1   s&x            S&X = FFF (ctr for L8)
A0B4 20C               ?fset   2              doing U2 ?
A0B5 01B               jnc     burnlu20       ..no, skip
A0B6 130               ldi     $3FF           S&X = 3FF (ctr for U2)
A0B7 3FF
A0B8 070    burnlu20   n=c                    save ctr/adr in N

            * Use section number in Y to make start & end EPROM address
A0B9 130               ldi     2              1 is max section # for L8
A0BA 002
A0BB 20C               ?fset   2              doing U2 ?
A0BC 01B               jnc     burnlu30       ..no, skip
A0BD 130               ldi     4              3 is max section # for U2
A0BE 004
A0BF 0E6    burnlu30   c<>b    s&x            max value to B
A0C0 0B8               read    2              get the Y register
A0C1 10E               a=c     all            put it in A
A0C2 36D               xqrel   bcdbin         convert to binary
A0C3 0BC
A0C4 0ED

A0C5 20C               ?fset   2              doing U2 ?
A0C6 037               jc      burnlu40       ..yes, skip the L8 part
A0C7 1BC               rcr     11             left 3 - adr is 0000 or 1000
A0C8 158               m=c                    start adr to M
A0C9 266               c=c-1   s&x            end adr (xFFF)
A0CA 10E               a=c     all            end adr to A
A0CB 04B               jnc     burnlu50       skip past U2 stuff

```

```
00104    A0CC 1E6    burnlu40  c=c+c  s&x
00105    A0CD 1E6              c=c+c  s&x             0,1,2,3 -> 0,4,8,C
00106    A0CE 37C              rcr    12              left 2, adr is 0000 to 0C00
00107    A0CF 158              m=c                    start adr in M
00108    A0D0 10E              a=c    all             adr to A
00109    A0D1 130              ldi    $3FF
00109    A0D2 3FF
00110    A0D3 146              a=a+c  s&x             end = start + 3FF
00111
00112                * Initialize the MC00506 - start adr in M, end adr in A
00113                *       Does init, blank check, resets start adr
00114    A0D4 36D    burnlu50  xqrel  initmce
00114    A0D5 0BC
00114    A0D6 037
00115
00116                * Do first fetch, then start burning loop
00117    A0D7 349              xqrel  prefetch        get 1st byte of ROM
00117    A0D8 08C
00117    A0D9 0EA
00118    A0DA 308              setf   1               indicate 2nd to Nth bytes
00119
00120
00121                ***********************************************************
00122                * This is the main programming loop                       *
00123                ***********************************************************
00124    A0DB 375    burnlu    ?ncxq  $70DD           LAD to MC00506
00124    A0DC 1C0
00125    A0DD 0CE              c=b    all             get the byte from B
00126    A0DE 099              ?ncxq  $7126           send 1st DAB
00126    A0DF 1C4
00127
00128                * Source a GET to burn the byte
00129    A0E0 064              selp   4               IL reg 1
00130    A0E1 205              con    $205            init R1 for CMD
00131    A0E2 0A4              selp   5               IL reg 2
00132    A0E3 021              con    $021            source the GET (hex08)
00133
00134                * Display the current address and increment it
00135    A0E4 36D              xqrel  adr2dis
00135    A0E5 08C
00135    A0E6 0AF
00136    A0E7 198              c=m
00137    A0E8 22E              c=c+1  all
00138    A0E9 158              m=c
00139
00140                * Flag 2 determines L8 or U2
00141    A0EA 20C    prefetch  ?fset  2
00142    A0EB 04F              jc     u2fetch
00143
00144                * Fetch the next L8 ROM word
00145    A0EC 0B0              c=n                    get the adr/ctr
00146    A0ED 0E6              c()b   s&k             save ctr in B
00147    A0EE 330              fetch                  get the word
00148
00149    A0EF 0E6              c()b   s&k             ROM word to B, ctr to C
00150    A0F0 23A              c=c+1  m               incr address
00151    A0F1 070              n=c                    save adr & ctr in N
00152    A0F2 0A3              jnc    burn10          bypass u2fetch
00153
00154    A0F3 343    brnlujmp  jnc    burnlu          stepping stone
00155
00156
```

```
                      * Fetch the next U2 ROM word
A0F4 0B0    u2fetch     c=n                         get adr to C
A0F5 0E6                c()b     s&x                save ctr in B
A0F6 006                a=0      s&x                clr bit ACC
A0F7 05C                r=       4                  set loop ctr

A0F8 330    u2fet10     fetch                       get ROM word
A0F9 23A                c=c+1    m                  incr ROM adr
A0FA 116                a=c      xs                 u2 bits to A
A0FB 0AE                a()c     all
A0FC 1E6                c=c+c    s&x
A0FD 1E6                c=c+c    s&x                push bits to upper end of s&x
A0FE 3C6                rshfc    s&x                put u2 bits in C d1,0
A0FF 0AE                a()c     all
A100 3D4                r=r-1                       decr loop ctr
A101 394                ?r=      0                  end of loop
A102 3B3                jnc      u2fet10            ..no, loop

A103 0C6                c=b      s&x                ctr to C, with ROM adr
A104 070                n=c                         save ctr/adr
A105 086                b=a      s&x                save u2 byte in B

A106 30C    burn10      ?fset    1                  F1 clear for initialization
A107 3A0                ?nertn                      return if init call


            * error check the GET frame
A108 2F5                ?ncxq    $70BD
A109 1C0

            * TAD to MC00506
A10A 2C9                ?ncxq    $70B2              TAD r5
A10B 1C0

            * Send the SST and get the DAB
A10C 130                ldi      $061
A10D 061

A10E 0D9                ?ncxq    $7136
A10F 1C4

            * Do a TRANSMIT ERROR check
A110 39D                ?ncxq    $77E7
A111 1DC

            * The status byte was returned in digits 1 & 2 of A.
A112 386                rshfa    s&x                status to A d0,1
A113 0A6                a()c     s&X                status to C
A114 3D8                c()st                       status into flags
A115 38C                ?fset    0
A116 0E7                jc       burnerr            BAD BURN
A117 160                ?lowbat
A118 157                jc       lowbaterr          LOW BATTERY HALT
A119 3D8                c()st                       put back the flags
A11A 3C8                clrkey
A11B 3CC                ?key                        key down ?
A11C 02B                jnc      loopctrl           ..no, continue
A11D 220                c=key                       get the key
A11E 01C                r=       3                  pt to LS key digit
A11F 1E2                c=c+c    @r                 gen carry if key was 'ON'
A120 037                jc       burnexit           quit if key was 'ON'

            * Looping control
A121 0B0    loopctrl    c=n                         get adr, ctr
A122 266                c=c-1    s&x
```

```
00216   A123 01F            jc      burnexit
00217   A124 070            n=c                     restore new ctr
00218   A125 273            jnc     brnlujmp        do another loop
00219
00220   A126 3C1  burnexit  ?ncxq   $2CF0           clear the display
00220   A127 0B0
00221   A128 149            ?ncxq   $0952           deselect display & rtn
00221   A129 024
00222   A12A 375            ?ncxq   $70DD           LAD to MC00506
00222   A12B 1C0
00223   A12C 130            ldi     $004            SDC - reset the MC00506
00223   A12D 004
00224   A12E 2E9            ?ncxq   $70BA           CMD frame
00224   A12F 1C0
00225   A130 0D9            ?ncgo   $7336           UNL, error check, & RTN
00225   A131 1CE
00226
00227
00228             * Bad Burn error message
00229   A132 2CD  burnerr   ?nckq   $77B3           Put mssg in dsp
00229   A133 1DC
00230   A134 002            con     $02
00231   A135 015            con     $15
00232   A136 012            con     $12
00233   A137 20E            con     $20E            mssg: BURN
00234   A138 015  brnERRmsg ?ncxq   $7D05           Add ERR and do error
00234   A139 1F4
00235
00236             * Erase Error mssg
00237   A13A 2CD  eraserr   ?ncxq   $77B3           put mssg in dsp
00237   A13B 1DC
00238   A13C 005            con     $05
00239   A13D 012            con     $12
00240   A13E 001            con     $01
00241   A13F 013            con     $13
00242   A140 205            con     $205            mssg: ERASE
00243   A141 3BB            jnc     brnERRmsg
00244
00245             * Low battery error
00246   A142 2CD  lowbaterr ?ncxq   $77B3           put mssg in dsp
00246   A143 1DC
00247   A144 00C            con     $0C
00248   A145 00F            con     $0F
00249   A146 017            con     $17
00250   A147 020            con     $20
00251   A148 002            con     $02
00252   A149 001            con     $01
00253   A14A 214            con     $214            mssg: LOW BAT
00254   A14B 36B            jnc     brnERRmsg
00255
00256             *************************************************************
00257             *RCOPY - Function to copy 4k blocks.                        *
00258             *       - INPUT: Y - Source block.                          *
00259             *       -        X - Destination block.                     *
00260             * No error checking is done to insure that the destination  *
00261             * block is really RAM. This is up to the user.              *
00262             *************************************************************
00263
00264   A14C 099            con     $99
00265   A14D 010            con     $10
00266   A14E 00F            con     $0F
00267   A14F 003            con     $03
00268   A150 012            con     $12             name: RCOPY
```

```

A151 36D   rcopy     xqrel   bcdbinXF       X reg (dest #) range 0-15
A152 08C
A153 0EB
A154 13C             rcr     8              left 6 to make adr
A155 0FA             c<>b    m              put dest adr in B Mant
A156 0B8             read    2              get src blk #
A157 10E             a=c     all
A158 36D             xqrel   bcdbin         convert to bin, range 0-15
A159 08C
A15A 0ED
A15B 13C             rcr     8              left 6 to make adr
A15C 00E             a=0     all            init the counter
A15D 330   rcopy10   fetch                  get the src word
A15E 23A             c=c+1   m              incr src adr
A15F 0FA             c<>b    m              get dest adr
A160 040             writr                  write the dest word
A161 23A             c=c+1   m              incr dest adr
A162 0FA             c<>b    m              src adr back to C
A163 166             a=a+1   s&x            incr ctr
A164 3CB             jnc     rcopy10
A165 3E0             rtn

 -         **************************************************************
           * READROM - Read a ROM>REG format tape file into emulated ROM*
           *         - INPUT: Filename in ALPHA                         *
           *                  Block number in X                         *
           *         - OUTPUT: 4K block loaded to EROM                  *
           **************************************************************

A166 08D             con     $8D
A167 00F             con     $0F
A168 012             con     $12
A169 004             con     $04
A16A 001             con     $01
A16B 005             con     $05
A16C 012             con     $12            name: READROM

A16D 36D   readrom   xqrel   piltst         test for existence of PIL
A16E 08C
A16F 176
A170 36D             xqrel   bcdbinXF       X reg (blk #) in range 0-15
A171 08C
A172 0E8
A173 2B8             read    10             status scratch
A174 0A6             a<>c    s&x            put in block number
A175 2A8             writ    10             save block # in Status 10

A176 021             ?ncxq   $7808          get file params from DIR
A177 1E0
A178 130             ldi     824            exp. file size
A179 338
A17A 106             a=c     s&x
A17B 0B0             c=n                    get file params
A17C 366             ?a#c    s&x            is filesize <> 824 regs
A17D 05B             jnc     rdrom20        ..ok, continue

A17E 2CD   fmterr    ?ncxq   $77B3          put mssg in display
A17F 1DC
A180 006             con     $06
A181 00F             con     $0F
A182 012             con     $12
A183 00D             con     $0D
A184 001             con     $01
```

```
A185 214             con     $214        mesg: FORMAT
A186 015             ?ncxq   $7D05       Add ERR and do error
A187 1F4


A188 1DD   rdrom20   ?ncxq   $7F77       Seek the file
A189 1FC
A18A 399             ?ncxq   $70E6       Send buffer 0 and SDA
A18B 1C0
A18C 0E0             slctq
A18D 05C             r=      4           loop counter
A18E 104             clrf    8           indicate reg fetch from PIL

A18F 0A0   rdrom30   slctp
A190 0F9             ?ncxq   $763E       get the header reg
A191 1D8
A192 070             n=c                 header to N
A193 1BC             rcr     11          left 3
A194 280             ilw     2           retransmit the last DAB
A195 130             ldi     $3FF        header count must be 3FF
A196 3FF
A197 366             ?a#c    s&x         count <> 3FF ?
A198 337             jc      fmterr      ..yes, format error
A199 1BE             a=a-1   ms
A19A 1BE             a=a-1   ms          check MS = 1 ?
A19B 31B             jnc     fmterr
A19C 2B8             read    10          get status 10, block # in bin.
A19D 05A             c=0     m
A19E 158             m=c                 block # in M
A19F 349             xqrel   regrom10    1024 bytes to EROM
A1A0 08C
A1A1 1BD
A1A2 0E0             slctq
A1A3 3D4             r=r-1               decr loop ctr
A1A4 394             ?r=     0           end of loop ?
A1A5 353             jnc     rdrom30     ..no, continue

A1A6 2B1             ?ncgo   $70AC       UNT and rtn
A1A7 1C2


                     **************************************************************
                     *REG>ROM                                                     *
                     **************************************************************

A1A8 08D             con     $8D
A1A9 00F             con     $0F
A1AA 012             con     $12
A1AB 03E             con     $3E
A1AC 007             con     $07
A1AD 005             con     $05
A1AE 012             con     $12         name: REG>ROM

A1AF 0B8   regrom    read    2           get the start adr from Y
A1B0 158             m=c                 save in M
A1B1 0F8             read    3           get start reg #
A1B2 38D             ?ncxq   $02E3       convert to binary
A1B3 008
A1B4 106             a=c     s&x         move reg # to A
A1B5 378             read    13
A1B6 03C             rcr     3           move curtain to s&x
A1B7 206             c=c+a   s&x         compute reg adr
A1B8 270             ramslct
A1B9 0E6             c<>b    s&x         save reg adr in B
A1BA 038             read    0           get the header
```

```
00378   A1BB 070             n=c                  save in N
00379   A1BC 108             setf    8            indicate REG>ROM
00380
00381                 * Entry point for READROM
00382   A1BD 0B0   regrom10  c=n                  get header from N
00383   A1BE 03C             rcr     3            AAAA to LS 4 digits
00384   A1BF 10E             a=c     all          move to A
00385   A1C0 198             c=m                  get the start adr
00386   A1C1 33C             rcr     1            split start adr across MS,S&X
00387   A1C2 05A             c=0     m            clear rest of digits
00388   A1C3 2E6             ?c#0    s&x          test upper 3 digits
00389   A1C4 047             jc      regrom20     > 000F - abs address
00390   A1C5 0FC             rcr     10           <= ? , move 4 left
00391   A1C6 0A6             a<>c    s&x          move lowr 3 of AAAA to C
00392   A1C7 01C             r=      3            pt to MS address digit
00393   A1C8 2E2             ?c#0    @r           test that digit
00394   A1C9 027             jc      regrom30     if <> 0, it's 4k blk adr
00395   A1CA 0A2             a<>c    @r           else same 4k blk as before
00396   A1CB 013             jnc     regrom30     unconditional
00397   A1CC 2FC   regrom20  rcr     13
00398   A1CD 1BC   regrom30  rcr     11           left 3
00399   A1CE 158             m=c                  save current ROM adr in M
00400   A1CF 10E             a=c     all          and in A
00401   A1D0 0B0             c=n                  get the header again
00402   A1D1 05A             c=0     m            clr upper digits, NNN in s&x
00403   A1D2 1BC             rcr     11           left 3 to adr field in Mant
00404   A1D3 15A             a=a+c   m            add NNN to strt to get final
00405   A1D4 09A             b=a     m            final ROM adr in B Mantissa
00406
00407
00408
00409                 * Main loop for REG>ROM
00410   A1D5 10C   regrom40  ?fset   8            src from regs or IL ?
00411   A1D6 037             jc      regrom50     get src from Regs
00412
00413                 * Fetch register from HPIL (Cassette)
00414   A1D7 0F9             ?ncxq   $763E        fetch next tape register
00414   A1D8 1D8
00415   A1D9 1BC             rcr     11           left 3
00416   A1DA 280             ilw     2            retransmit last DAB
00417   A1DB 03B             jnc     regrom60     continue...
00418
00419                 * Fetch from data registers
00420   A1DC 0C6   regrom50  c=b     s&x          get old reg adr
00421   A1DD 226             c=c+1   s&x          incr to current reg
00422   A1DE 270             ramslct              slct the reg
00423   A1DF 0E6             c<>b    s&x          save the new reg adr
00424   A1E0 038             read    0            get the register
00425   A1E1 10E             a=c     all          move register to A
00426
00427                 * Resume common processing with register in A
00428   A1E2 198   regrom60  c=m                  get current ROM adr
00429   A1E3 0AE             a<>c    all          ROM adr to A, register to C
00430   A1E4 2FC             rcr     13
00431   A1E5 1EE             c=c+c   all
00432   A1E6 1EE             c=c+c   all          left justify bits in C
00433   A1E7 09C             r=      5            set the inner loop counter
00434
00435   A1E8 37C   regrom70  rcr     12           rotate C left 8 bits
00436   A1E9 056             c=0     xs           clear upper bits of word
00437   A1EA 1EE             c=c+c   all
00438   A1EB 013             jnc     *+02
00439   A1EC 22E             c=c+1   all
00440   A1ED 1EE             c=c+c   all
00441   A1EE 013             jnc     *+02         rotate C left 2 more bits
```

```
A1EF 22E               c=c+1   all           end with ROM word in s&x
A1F0 106               a=c     s&x           move ROM word to A
A1F1 0AE               a<>c    all           string to A, Adr-word to C
A1F2 040               writr                 write to ROM
A1F3 0AE               a<>c    all           adr-word to A, string to C
A1F4 33A               ?a<b    m             current adr < final adr
A1F5 3A0               ?ncrtn                ..no, stop
A1F6 17A               a=a+1   m             incr the address
A1F7 3D4               r=r-1                 decr ctr
A1F8 394               ?r=     0             end of inner loop?
A1F9 37B               jnc     regrom70      ..no, continue inner loop

A1FA 0AE               a<>c    all           ROM adr to C
A1FB 158               m=c                   ROM adr to M
A1FC 2CB               jnc     regrom40      MAIN Loop

          ***************************************************************
          *WRTROM - Copy a ROM image (in ROM)REG format) to a tape file*
          *       - The file is created if neccessary.                  *
          *       - INPUT: Filename in ALPHA                            *
          *                Block number to save in X                    *
          *       - OUTPUT: Block saved                                 *
          ***************************************************************

A1FD 08D               con     $8D
A1FE 00F               con     $0F
A1FF 012               con     $12
A200 014               con     $14
A201 012               con     $12
A202 017               con     $17           name: WRTROM

A203 36D  wrtrom       xqrel   piltst        test for HPIL existence
A204 08C
A205 176
A206 36D               xqrel   bcdbinXF      test X reg for range 0-15
A207 08C
A208 0E8

          * Check for named file existent
A209 05E               c=0     ms
A20A 02D               ?ncxq   $780B         ck any type, same name
A20B 1E0
A20C 198               c=m                   M = 0 if no file
A20D 2EE               ?c#0    all           did file exist ?
A20E 08B               jnc     newfile       ..no, create it

          * Named file exists: must be DATA, 824 Regs, Unsecured
A20F 0B0               c=n                   get the file pointer
A210 10E               a=c     all           FP to A
A211 130               ldi     $D
A212 00D
A213 33C               rcr     1
A214 37E               ?a#c    ms            is the file DATA, type D
A215 01B               jnc     wrtrom10      ..yes, continue
A216 249  duperr       ?ncgo   $7692         DUP FL NAME error
A217 1DA
A218 130  wrtrom10     ldi     824
A219 338
A21A 366               ?a#c    s&x           size = 824 regs ?
A21B 3DF               jc      duperr        ..no, DUP FL NAME
A21C 031               ?ncxq   $7D0C         test secured
A21D 1F4
```

```
A21E 06B             jnc     wrtrom30        file OK, use it

                 * Create a new file - type DATA, 824 regs, 26 records
A21F 04E   newfile   c=0     all             ready to make new file ptr
A220 130             ldi     $D0             type and status
A221 0D0
A222 23C             rcr     2
A223 130             ldi     824             # of regs
A224 338
A225 07C             rcr     4
A226 130             ldi     26              # of records
A227 01A
A228 07C             rcr     4
A229 2F1             ?ncxq   $76BC           make the file
A22A 1D8

                 * File pointer is in N, get ready to write the file
A22B 1C9   wrtrom30  ?ncxq   $7F72           seek to file, set BP = 0
A22C 1FC
A22D 130             ldi     $A2             DDL 2 - WRITE mode
A22E 0A2
A22F 2E9             ?ncxq   $70BA           CMD
A230 1C0
A231 369             ?ncxq   $70DA           DDL 0 - Write buffer 0
A232 1C0
A233 39D             ?ncxq   $77E7           error check
A234 1DC
A235 064             selp    4
A236 005             con     $005            set r1 for DAB



                 * Make first header register
A237 36D             xqrel   bcdbinXF        X reg (blk #) to binary (0-15)
A238 08C
A239 0E8
A23A 33C             rcr     1               block # to ms
A23B 05A             c=0     m
A23C 130             ldi     206             # of regs in file
A23D 0CE
A23E 2BC             rcr     7
A23F 130             ldi     $3FF            byte count
A240 3FF
A241 2DC             r=      13
A242 0D0             ld@r    3               loop ctr in MS of header

A243 158   wrtrom50  m=c                     save new header in M
A244 05E             c=0     ms
A245 23E             c=c+1   ms              type header as ALPHA
A246 33D             ?ncxq   $75CF           send the register
A247 1D4
A248 39D             ?ncxq   $77E7           error check
A249 1DC
A24A 198             c=m                     get the header back
A24B 00E             a=0     all
A24C 106             a=c     s&x             copy the 3FF to A
A24D 07C             rcr     4
A24E 05A             c=0     m               clear header except st adr
A24F 2FC             rcr     13              adr in C d3-0
A250 14E             a=a+c   all             end adr in A
A251 3EE             lshfa   all
A252 3EE             lshfa   all
A253 3EE             lshfa   all             end adr to d6-3 of A
A254 1BC             rcr     11              st adr to d6-3 of C
A255 0EE             c<>b    all             st adr to B
```

```
00545   A256 108             setf    8
00546   A257 349             xqrel   rmrgguts      ROM)REG the 1k block
00546   A258 08C
00546   A259 268
00547   A25A 198             c=m                   get the header
00548   A25B 09C             r=      5             pt to A2 digit (of A3-0)
00549   A25C 222             c=c+1   @r
00550   A25D 222             c=c+1   @r
00551   A25E 222             c=c+1   @r
00552   A25F 222             c=c+1   @r            add $400 to start adr
00553   A260 27E             c=c-1   ms            decrement loop ctr
00554   A261 313             jnc     wrtrom50      next 1k block
00555
00556                * Close the record and clean up
00557   A262 130             ldi     $A8           DDL 8
00557   A263 0A8
00558   A264 2E9             ?ncxq   $70BA         CMD
00558   A265 1C0
00559   A266 2BD             ?ncgo   $70AF         UNL
00559   A267 1C2
00560
00561
00562                * This is the guts of ROM)REG and WRTROM
00563
00564   A268 06E     rmrgguts a()b   all
00565   A269 09C             r=      5
00566   A26A 0AE     rmrg10  a()c    all           adr to C, str to A
00567   A26B 3EE             lshfa   all
00568   A26C 3EE             lshfa   all           A 2 digits left
00569   A26D 330             fetch
00570   A26E 106             a=c     s&x           ROM word to A
00571   A26F 0AE             a()c    all           str to C, adr to A
00572   A270 1E6             c=c+c   s&x
00573   A271 1E6             c=c+c   s&x           push word against end of str
00574   A272 1EE             c=c+c   all
00575   A273 1EE             c=c+c   all           shift str to even digit
00576   A274 17A             a=a+1   m             incr adr
00577   A275 3D4             r=r-1
00578   A276 394             ?r=     0             end of loop ?
00579   A277 39B             jnc     rmrg10        ..no, continue looping
00580   A278 3CE             rshfc   all           right justify str in C
00581   A279 23E             c=c+1   ms            type as ALPHA
00582   A27A 10C             ?fset   8             ROM)REG or WRTROM ?
00583   A27B 03F             jc      rmrg20        ..do WRTROM
00584
00585   A27C 1D8             c()m                  str to M/reg adr to C
00586   A27D 226             c=c+1   s&x           incr reg adr
00587   A27E 270             ramslct
00588   A27F 1D8             c()m                  str to C/reg adr to M
00589   A280 2F0             writd                 write the register
00590   A281 053             jnc     rmrg30        skip past WRTROM section
00591
00592   A282 0AE     rmrg20  a()c    all           str to A, adr to C
00593   A283 070             n=c                   save adr in N
00594   A284 0AE             a()c    all           str to C
00595   A285 33D             ?ncxq   $75CF         send to PIL
00595   A286 1D4
00596   A287 39D             ?ncxq   $77E7         error check
00596   A288 1DC
00597   A289 0B0             c=n                   get adr back
00598   A28A 10E             a=c     all           adr to A
00599
00600   A28B 06E     rmrg30  a()b    all           swap start/end adr
00601   A28C 33A             ?a(b    m             EEEE ( BBBB ?
00602   A28D 2DB             jnc     rmrgguts
```

```
A28E 3E0             rtn



                *****************************************************************
                * ROM)REG - Pack ROM words into HP-41 registers.                *
                *         - INPUT: X - Starting register number.                *
                *         -            Y - Binary start/end ROM adrs (BBBBEEEE). *
                *         - OUTPUT: L - The number of the last register used.   *
                * See PPCCJ V9N3P41 for source code comments.                   *
                *****************************************************************

A28F 087             con     $87
A290 005             con     $05
A291 012             con     $12
A292 03E             con     $3E
A293 00D             con     $0D
A294 00F             con     $0F
A295 012             con     $12           name: ROM)REG

A296 349   romreg    xqrel   lastreg
A297 0BC
A298 2DD
A299 046             c=0     s&x
A29A 270             ramslct
A29B 0F8             read    3
A29C 38D             ?nckq   $02E3
A29D 008
A29E 106             a=c     s&x
A29F 378             read    13
A2A0 03C             rcr     3
A2A1 206             c=c+a   s&x
A2A2 158             m=c
A2A3 106             a=c     s&x
A2A4 066             a()b    s&k
A2A5 246             c=a-c   s&k
A2A6 106             a=c     s&k
A2A7 1E6             c=c+c   s&k
A2A8 1E6             c=c+c   s&k
A2A9 146             a=a+c   s&k
A2AA 086             b=a     s&x
A2AB 098             read    2
A2AC 00E             a=0     all
A2AD 01C             r=      3
A2AE 10A             a=c     r(-
A2AF 0BC             rcr     5
A2B0 05A             c=0     m
A2B1 2FC             rcr     13
A2B2 070             n=c
A2B3 24A             c=a-c   r(-
A2B4 3EE             lshfa   all
A2B5 3EE             lshfa   all
A2B6 3EE             lshfa   all
A2B7 106             a=c     s&x
A2B8 066             a()b    s&x
A2B9 326             ?a(b    s&x
A2BA 381             ?cgo    $02E0         NONEXISTENT
A2BB 00B
A2BC 198             c=m
A2BD 106             a=c     s&x
A2BE 0B0             c=n
A2BF 1BC             rcr     11
A2C0 0FA             c()b    m
A2C1 0CE             c=b     all
```

```
A2C2 2BC             rcr     7
A2C3 0A6             a<>c    s&x
A2C4 070             n=c

A2C5 104             clrf    8
A2C6 349             xqrel   rmrgguts
A2C7 0BC
A2C8 268

A2C9 198             c=m
A2CA 106             a=c     s&x
A2CB 0B0             c=n
A2CC 270             ramslct
A2CD 246             c=a-c   s&x
A2CE 226             c=c+1   s&x
A2CF 2BC             rcr     7
A2D0 05E             c=0     ms
A2D1 23E             c=c+1   ms
A2D2 2F0             writd
A2D3 046             c=0     s&x
A2D4 270             ramslct
A2D5 378             read    13
A2D6 03C             rcr     3
A2D7 1C6             a=a-c   s&x
A2D8 36D             xqrel   binbcd
A2D9 08C
A2DA 102
A2DB 128             writ    4
A2DC 3E0             rtn


           ***************************************************************
           * LASTREG - Return the address of the last existent RAM reg.*
           *         - INPUT: NONE                                     *
           *         - OUTPUT: Last register adr in B,C S&X            *
           *         - USES: A,B,C, RAMSLCTED REG                      *
           ***************************************************************

A2DD 130   lastreg   ldi     $23F
A2DE 23F
A2DF 0E6             c<>b    s&x           move adr to B
A2E0 130   lstreg10  ldi     $40
A2E1 040
A2E2 066             a<>b    s&x
A2E3 246             c=a-c   s&x           sub $40 from adr
A2E4 270             ramslct
A2E5 0E6             c<>b    s&x           save adr in B
A2E6 038             read    0             get reg contents
A2E7 10E             a=c     all           save in A
A2E8 2A6             c=-c-1  s&x           complement part of C
A2E9 2F0             writd                 write it
A2EA 038             read    0             read it back
A2EB 2A6             c=-c-1  s&x           complement it again
A2EC 36E             ?a#c    all           same as original ?
A2ED 39F             jc      lstreg10      ..no, try lower adr
A2EE 2F0             writd                 restore original contents
A2EF 0C6             c=b     s&x           adr to C
A2F0 3E0             rtn

```

```
                 ***************************************************************
                 * GETEP - Read the L8 and U2 EPROMs into a ROME block.        *
                 *       - INPUT: X - The block number to load into.           *
                 *       -        Y - The U2 section to read from.             *
                 *       -        Z - The L8 section to read from.             *
                 * Both the L8 and the U2 EPROMs are prompted for by the       *
                 * program.                                                    *
                 ***************************************************************

A2F1 090                   con      $90
A2F2 005                   con      $05
A2F3 014                   con      $14
A2F4 005                   con      $05
A2F5 007                   con      $07            name: GETEP

A2F6 3C4         getep     st=0
A2F7 008                   setf     3              rtn early from initmce
A2F8 36D                   xqrel    findmc         PILTST and MC00506 findid
A2F9 08C
A2FA 126
A2FB 36D                   xqrel    bcdbinXF       X -> bin, 15 max value
A2FC 08C
A2FD 0E8
A2FE 13C                   rcr      8              6 left - make it an adr
A2FF 266                   c=c-1    s&x            set up ctr in s&x (FFF)
A300 070                   n=c                     adr/ctr in N

                 * Get the U2 section # - save in Q
A301 130                   ldi      4              burn sec # max is 3
A302 004
A303 0E6                   c<>b     s&x
A304 0B8                   read     2              get U2 burn section #
A305 10E                   a=c      all
A306 36D                   xqrel    bcdbin         burn sec to binary
A307 08C
A308 0ED
A309 268                   writ     9              save U2 # in Q

                 * Get the L8 section #
A30A 130                   ldi      2              burn sec # max is 1
A30B 002
A30C 0E6                   c<>b     s&x
A30D 078                   read     1              get L8 section #
A30E 10E                   a=c      all
A30F 36D                   xqrel    bcdbin         burn sec. to bin
A310 08C
A311 0ED


                 * Make L8 EPROM start and end adrs
A312 1BC                   rcr      11             left 3, adr is 0000 or 1000
A313 158                   m=c                     start adr to M
A314 266                   c=c-1    s&x            end adr (xFFF)
A315 10E                   a=c      all            end adr to A

                 * Init the MC00506 - RTN after setting the range
A316 36D                   xqrel    initmce
A317 08C
A318 037

                 * Display prompt, beep, wait for response
A319 3D9                   ?ncxq    $07F6          turn on display
A31A 01C
```

```
A31B 3BD             ?nexq   $07EF           mssg to display
A31C 01C
A31D 012             con     $12
A31E 004             con     $04
A31F 019             con     $19
A320 020             con     $20
A321 00C             con     $0C
A322 038             con     $38
A323 020             con     $20
A324 005             con     $05
A325 010             con     $10
A326 012             con     $12
A327 00F             con     $0F
A328 20D             con     $20D            mssg: RDY L8 EPROM
A329 104             clrf    8
A32A 349             xqrel   gepwait
A32B 08C
A32C 392

          * Set MC00506 as talker, and source SDA
A32D 2C9             ?ncxq   $70B2           TAD r5
A32E 1C0
A32F 3A9             ?ncxq   $70EA           SDA, wait for FRAV
A330 1C0

A331 0B0             c=n                     get adr/ctr
A332 10E             a=c     all             adr/ctr to A
A333 09A             b=a     m               adr to B
A334 013             jnc     rd120

          * Main Loop for Read1
A335 280  rd110      ilw     2               retransmit the DAB
A336 041  rd120      ?ncxq   $7110           get a DAB
A337 1C4
A338 39D             ?ncxq   $77E7           XMIT ERR on non-dab
A339 1DC
A33A 0EE             c()b    all             DAB to B, ROM adr to C
A33B 330             fetch                   get ROM word
A33C 0F6             c()b    xs              put U2 with new DAB in B
A33D 0C6             c=b     s&x             new ROM word to C
A33E 040             writr                   write new ROM word
A33F 23A             c=c+1   m               incr ROM adr
A340 0FA             c()b    m               save ROM adr in B
A341 1A6             a=a-1   s&x             decr the ctr
A342 39B             jnc     rd110           repeat for 4k block

A343 3D9             ?ncxq   $70F6           NRD and retrans last DAB
A344 1C0
A345 39D             ?ncxq   $77E7           XMIT ERR ??
A346 1DC



A347 3C4             st=0
A348 00B             setf    3               rtn early from initmce
A349 0DA             c=b     m               get the block adr
A34A 15C             r=      6               pt to M5 adr digit
A34B 262             c=c-1   @r              set to correct block (n000)
A34C 130             ldi     $3FF            set up ctr in s&x (3FF)
A34D 3FF
A34E 070             n=c                     adr/ctr in N
A34F 278             read    9               get U2 sec # from Q

```

```
00818                   * Make EPROM start and end adrs
00819   A350 1E6                c=c+c   s&x
00820   A351 1E6                c=c+c   s&x             0,1,2,3 -> 0,4,8,C
00821   A352 37C                rcr     12              left 2, adr is 0000 to 0C00
00822   A353 158                m=c                     start adr to M
00823   A354 10E                a=c     all             adr to A
00824   A355 130                ldi     $3FF
00824   A356 3FF
00825   A357 146                a=a+c   s&x             end = start + 3FF
00826
00827                   * Init the MC00506 - RTN after setting the range
00828   A358 36D                xqrel   initmce
00828   A359 08C
00828   A35A 037
00829
00830                   * Display prompt, beep, wait for key
00831   A35B 3D9                ?ncxq   $07F6           enable display
00831   A35C 01C
00832   A35D 3BD                ?ncxq   $07EF           chars to disp
00832   A35E 01C
00833   A35F 012                con     $12
00834   A360 004                con     $04
00835   A361 019                con     $19
00936   A362 020                con     $20
00837   A363 015                con     $15
00938   A364 032                con     $32
00839   A365 020                con     $20
00840   A366 005                con     $05
00841   A367 010                con     $10
00842   A368 012                con     $12
00843   A369 00F                con     $0F
00844   A36A 20D                con     $20D            mssg: RDY U2 EPROM
00845   A36B 108                setf    8
00846   A36C 349                xqrel   gepwait
00846   A36D 08C
00846   A36E 392
00847
00848                   * Set MC00506 as talker, and source SDA
00849   A36F 2C9                ?ncxq   $70B2           TAD r5
00849   A370 1C0
00850   A371 3A9                ?ncxq   $70EA           SDA, wait for FRAV
00850   A372 1C0
00851
00852   A373 0B0                c=n                     get adr/ctr
00853   A374 10E                a=c     all             adr/ctr to A
00854   A375 09A                b=a     m               adr to B
00855   A376 013                jnc     rdu20
00856
00857                   * Main loop for Readu
00858   A377 280        rdu10   ilw     2               retransmit the DAB
00859   A378 041        rdu20   ?ncxq   $7110           get a Loop DAB
00859   A379 1C4
00860   A37A 39D                ?ncxq   $77E7           XMIT ERR on non-dab
00860   A37B 1DC
00861   A37C 37C                rcr     12              byte left 2 digits
00862   A37D 05C                r=      4               loop ctr
00863   A37E 0EE        rdu30   c()b    all             adr to C
00864   A37F 330                fetch                   get current ROM word
00865   A380 0D6                c=b     xs              get u2 bits
00866   A381 040                writr                   write new ROM word
00867   A382 23A                c=c+1   m               incr adr
00868   A383 0EE                c()b    all             pkd u2 bits to C/adr to B
00869   A384 1EE                c=c+c   all
00870   A385 1EE                c=c+c   all
```

```
00871   A386 33C             rcr     1             shift right 2 bits
00872   A387 3D4             r=r-1                 decr loop ctr
00873   A388 394             ?r=     0
00874   A389 3AB             jnc     rdu30         do all 4 words
00875   A38A 1A6             a=a-1   s&x           decr outer loop ctr
00876   A38B 363             jnc     rdu10         Main loop
00877
00878   A38C 3D9             ?ncxq   $70F6         NRD and retrans last DAB
00878   A38D 1C0
00879   A38E 39D             ?ncxq   $77E7         XMIT ERR ??
00879   A38F 1DC
00880   A390 2B1             ?ncgo   $70AC         UNT and return
00880   A391 1C2
00881
00882   A392 149   gepwait   ?ncxq   $0952         deselect display
00882   A393 024
00883   A394 36D             ?ncxq   $16DB         tone 7
00883   A395 058
00884   A396 04E             c=0     all
00885   A397 05C             r=      4
00886   A398 390             ld@r    14
00887   A399 05C             r=      4
00888
00889   A39A 3C8   gepw10    clrkey
00890   A39B 22A             c=c+1   r<-           add to ctr
00891   A39C 067             jc      gepoff
00892   A39D 3CC             ?key                  key down ?
00893   A39E 3E3             jnc     gepw10        wait for key
00894
00895   A39F 220             c=key                 get the key
00896   A3A0 01C             r=      3
00897   A3A1 1E2             c=c+c   @r            gen carry if key is 'ON'
00898   A3A2 037             jc      gepoff
00899
00900   A3A3 3C1             ?ncxq   $2CF0         clear display
00900   A3A4 0B0
00901   A3A5 149             ?ncxq   $0952         deselect display
00901   A3A6 024
00902   A3A7 3E0             rtn                   rtn for any other key
00903
00904   A3A8 10C   gepoff    ?fset   8
00905   A3A9 01F             jc      gepclr        clear block if f8 set
00906   A3AA 3C1   gepoff10  ?ncgo   $00F0
00906   A3AB 002
00907
00908   A3AC 00E   gepclr    a=0     all
00909   A3AD 1A6             a=a-1   s&x           A = FFF (ctr)
00910   A3AE 0B0             c=n                   get adr to C
00911   A3AF 046             c=0     s&x
00912   A3B0 040   gepclr10  writr
00913   A3B1 23A             c=c+1   m             incr adr
00914   A3B2 1A6             a=a-1   s&x           decr ctr
00915   A3B3 3EB             jnc     gepclr10      clear ROM
00916   A3B4 3B3             jnc     gepoff10
00917
00918                ************************************************************
00919                *                                                          *
00920                * KASN - Key Assign (Similar to 1K in PPC ROM)             *
00921                *                                                          *
00922                *  Input:  Z - First assignment byte                       *
00923                *          Y - Second assignment byte                      *
00924                *          X - Keycode                                     *
00925                *  If both Y & Z are zero, the assignment to the key (if any)*
00926                *  is cleared.                                             *
00927                ************************************************************
```

```


A3B5 08E                 con    $8E
A3B6 013                 con    $13
A3B7 001                 con    $01
A3B8 00B                 con    $0B          name: KASN

A3B9 0F8     kasn        read   3            get keycode from X
A3BA 2F6                 ?c#0   ks           check for keycode < 0
A3BB 01B                 jnc    kasn10

A3BC 0B5     kasnerr     ?ncgo  $282d        "DATA ERROR"
A3BD 0A2

             * This section of code lifted from PASN in X-Functions
A3BE 10E     kasn10      a=c    all
A3BF 266                 c=c-1  s&x
A3C0 3E7                 jc     kasnerr
A3C1 266                 c=c-1  s&x
A3C2 3D3                 jnc    kasnerr
A3C3 1BC                 rcr    11
A3C4 106                 a=c    s&x
A3C5 39C                 r=     0
A3C6 2E2                 ?c#0   @r
A3C7 3AB                 jnc    kasnerr
A3C8 130                 ldi    $90
A3C9 090
A3CA 31C                 r=     1
A3CB 302                 ?a<c   @r
A3CC 383                 jnc    kasnerr
A3CD 130                 ldi    $46
A3CE 046
A3CF 302                 ?a<c   @r
A3D0 017                 jc     kasn20
A3D1 266                 c=c-1  s&x
A3D2 39C     kasn20      r=     0
A3D3 302                 ?a<c   @r
A3D4 343                 jnc    kasnerr
A3D5 050                 ld@r   1
A3D6 31C                 r=     1
A3D7 362                 ?a#c   @r
A3D8 027                 jc     kasn30
A3D9 36A                 ?a#c   r<-
A3DA 013                 jnc    kasn30
A3DB 16A                 a=a+1  r<-
A3DC 1A6     kasn30      a=a-1  s&x
A3DD 0A6                 a<>c   s&x
A3DE 106                 a=c    s&x
A3DF 3C6                 rshfc  s&x
A3E0 3E6                 lshfa  s&x
A3E1 0A2                 a<>c   @r
A3E2 106                 a=c    s&x
A3E3 35E                 ?a#0   ms
A3E4 023                 jnc    kasn40
A3E5 130                 ldi    8
A3E6 008
A3E7 146                 a=a+c  s&x          binary keycode in A

             *put keycode in status reg 10
A3E8 2B8     kasn40      read   10
A3E9 0AA                 a<>c   r<-
A3EA 2A8                 writ   10

```

```
                  *convert the 2 assignment bytes to binary
A3EB 130                      ldi     256           bytes must be < 256
A3EC 100
A3ED 0E6                      c<>b    s&x           compare value in B
A3EE 078                      read    1             get ms FP value from Z
A3EF 10E                      a=c     all           move to A
A3F0 36D                      xqrel   bcdbin        convert to bin < 256
A3F1 08C
A3F2 0ED
A3F3 37C                      rcr     12            left 2 digits
A3F4 158                      m=c                   save ms byte in M
A3F5 0B8                      read    2             get ls FP value from Y
A3F6 10E                      a=c     all           move to A
A3F7 36D                      xqrel   bcdbin        convert to bin < 256
A3F8 08C
A3F9 0ED
A3FA 198                      c=m                   get ms value
A3FB 31C                      r=      1             set field
A3FC 0AA                      a<>c    r<-           combine low & high bytes
A3FD 0AE                      a<>c    all           save 2 bytes in A

A3FE 2B8                      read    10            get the binary keycode
A3FF 0AE                      a<>c    all           keycode in A / 2 bytes in C
A400 01C                      r=      3             field is lower 2 bytes
A401 2EA                      ?c#0    r<-           both assignment bytes = 0 ??
A402 331                      ?ncgo   $27cc         ..yes, clear the key
A403 09E
A404 0FC                      rcr     10            move 2 bytes to digits 4-7
A405 208                      setf    2
A406 1DD                      ?ncgo   $2777         mainframe key assignment
A407 09E




                  ***************************************************************
                  *                                                             *
                  * PCAT - Port Addressable Catalog 2                           *
                  *                                                             *
                  *    Non-programmable function that prompts for a single      *
                  *    digit in the range 0-9. This is interpreted as a ROM     *
                  *    address at which to start the CAT 2 list.                *
                  *                                                             *
                  *              1   -  Port 1 (Address $8000)                  *
                  *              2   -  Port 2 (Address $A000)                  *
                  *              3   -  Port 3 (Address $C000)                  *
                  *              4   -  Port 4 (Address $E000)                  *
                  *              5   -  Timer ROM (Address $5000)               *
                  *              6   -  Printer ROM (Address $6000)             *
                  *              7   -  HPIL ROM (Address $7000)                *
                  *          0,8,9   -  Normal CAT 2 (Address $5000)            *
                  *                                                             *
                  * Note:  This function prints strangely on the printer when   *
                  *        in NORM or TRACE modes. This is a bug in the HP41    *
                  *        mainframe. It can only be avoided by doing the digit *
                  *        prompt from within this routine, rather the using    *
                  *        the built-in prompting.                              *
                  ***************************************************************

A408 094                      con     $094
A409 001                      con     $001
A40A 303                      con     $303
A40B 110                      con     $110          name: PCAT (With 1 dig. prompt
)
```

```

A40C 000   pcat     nop                non-programmable
A40D 0AE            a()c    all        port # to C
A40E 358            st=c               move to ST
A40F 151            ?ncxq   $0054      check & process indirect
A410 000
A411 398            c=st               get port #
A412 39C            r=      0          set field
A413 1E2            c=c+c   @r         double port #
A414 0BF            jc      pcdflt     default on 8 or 9
A415 2E6            ?c#0    s&x        ck for port 0
A416 0AB            jnc     pcdflt     default on 0
A417 106            a=c     s&x        doubled port # to A
A418 130            ldi     6
A419 006
A41A 202            c=c+a   @r         +6 to get port adr for 1-4
A41B 017            jc      pcat10     if port was 5,6,7 use original
A41C 358            st=c               save port adr in ST
A41D 04E   pcat10   c=0     all        clear C
A41E 15C            r=      6          set ptr
A41F 398            c=st               get port adr
A420 13C            rcr     8          port adr to digit 6
A421 10E            a=c     all        port adr in d6 of A/ S&X clear
A422 150            ld@r    5          start adr = 5000
A423 23A            c=c+1   m          adr = 5001
A424 15C            r=      6          reset ptr

A425 362   pcat20   ?a#c    @r         test port adr
A426 033            jnc     pcat30     jump if equal
A427 330            fetch              get # of items in cat
A428 146            a=a+c   s&x        accumulate in A
A429 222            c=c+1   @r         incr to next ROM
A42A 3DB            jnc     pcat20     loop

A42B 00E   pcdflt   a=0     all
A42C 130   pcat30   ldi     2          cat '2'
A42D 002
A42E 1BC            rcr     11         move to digit 3
A42F 11A            a=c     m          move to A digit 3
A430 238            read    8          get the P reg
A431 0AE            a()c    all
A432 07C            rcr     4          move 2abc to 4 MS digits of C
A433 09C            r=      5
A434 0AA            a()c    r(-        put ALPHA from P into C
A435 239            ?ncgo   $0b8e      jump into catalog function
A436 02E






**************************************************************
* Init MC00506A - Assume: Loop addressed, MC00506 in r5     *
*                - REN to get ready for CMDS                 *
*                - Listen the primary device                 *
*                - SDC to reset the device                   *
*                - Send R, D, Y, B (each followed by LF)     *
*                - Send the start adr (in M d3-0)            *
*                - Send the term. adr (in A d3-0)            *
*                - RTN if F3 set                             *
*                - Do a verify blank (ERASE ERR if not]      *
*                - Send start adr (in M d3-0)                *
*                - Send NRE (get ready for data)             *
**************************************************************

```

```
01104   A437       LF       equ                  10
01105
01106   A437 130   initmce  ldi     $092
01106   A438 092
01107   A439 2E9            ?ncxq   $70BA        REN command
01107   A43A 1C0
01108   A43B 375            ?ncxq   $70DD        LAD to r5
01108   A43C 1C0
01109   A43D 130            ldi     $004
01109   A43E 004
01110   A43F 2E9            ?ncxq   $70BA        SDC command
01110   A440 1C0
01111   A441 36D            xqrel   ildata       send string to loop
01111   A442 08C
01111   A443 096
01112   A444 052            con     $52          'R' reset
01113   A445 00A            con     LF
01114   A446 044            con     $44          'D' disable CRLF
01115   A447 00A            con     LF
01116   A448 059            con     $59          'Y' 1 status byte
01117   A449 00A            con     LF
01118   A44A 042            con     $42          'B' binary mode
01119   A44B 00A            con     LF
01120
01121
01122                  * Send start address (SbbLF)
01123   A44C 253            con     $253         'S' - end of string
01124   A44D 198            c=m                  get address from M
01125   A44E 23C            rcr     2            MS byte in C d1,0
01126   A44F 0A1            ?ncxq   $7128        send loop DAB
01126   A450 1C4
01127   A451 198            c=m                  LS byte in C d1,0
01128   A452 0A1            ?ncxq   $7128        send 2nd adr byte
01128   A453 1C4
01129   A454 36D            xqrel   ildata       send string to loop
01129   A455 08C
01129   A456 096
01130   A457 00A            con     LF           punctuate start adr
01131
01132                  * Send term address (TbbLF)
01133   A458 254            con     $254         'T' - end of string
01134   A459 0AE            a<>c    all
01135   A45A 10E            a=c     all          copy end adr from A d3-0
01136   A45B 23C            rcr     2            MS byte in C d1,0
01137   A45C 0A1            ?ncxq   $7128        DAB
01137   A45D 1C4
01138   A45E 0AE            a<>c    all          get end adr from A
01139   A45F 0A1            ?ncxq   $7128        DAB
01139   A460 1C4
01140   A461 130            ldi     LF
01140   A462 00A
01141   A463 0A1            ?ncxq   $7128        DAB
01141   A464 1C4
01142   A465 39D            ?ncxq   $77E7        error check
01142   A466 1DC
01143   A467 00C            ?fset   3
01144   A468 360            ?crtn                RTN if F3 set
01145
01146                  * Verify that this address range is blank
01147   A469 36D            xqrel   ildata       send string to loop
01147   A46A 08C
01147   A46B 096
01148   A46C 056            con     $56          'V' - verify blank
01149   A46D 20A            con     $20A         LF - end of string
```

```
A46E 2C9             ?ncxq  $70B2        TAD to primary device
A46F 1C0
A470 130             ldi    $061         SST byte
A471 061
A472 0D9             ?ncxq  $7136        Send SST, get DAB in A
A473 1C4
A474 39D             ?ncxq  $77E7        do err ck
A475 1DC
A476 386             rshfa  s&x          move stat byte to A d1,0
A477 0A6             a<>c   s&x
A478 3D8             c<>st               status byte to ST
A479 00C             ?fset  3            test for bad verify
A47A 023             jnc    imce10       ..no, continue
A47B 341             gorel  eraserr      ERASE ERR
A47C 08C
A47D 13A
A47E 3D8   imce10    c<>st               replace ST


           * Send start adr again
A47F 375             ?ncxq  $70DD        LAD to primary device
A480 1C0
A481 130             ldi    $53          'S' char
A482 053
A483 099             ?ncxq  $7126        DAB
A484 1C4
A485 198             c=m                 get address from M
A486 23C             rcr    2            MS byte in C d1,0
A487 0A1             ?ncxq  $7128        send loop DAB
A488 1C4
A489 198             c=m                 LS byte in C d1,0
A48A 0A1             ?ncxq  $7128        send 2nd adr byte
A48B 1C4
A48C 130             ldi    LF
A48D 00A
A48E 0A1             ?ncxq  $7128        DAB
A48F 1C4

           * Send NRE so subsequent data bytes may be sent
A490 130             ldi    $093
A491 093
A492 2E9             ?ncxq  $70BA        send NRE cmd
A493 1C0
A494 39D             ?ncgo  $77E7        do the error check and return

A495 1DE



           ****************************************************************
           * ildata - Send the string of bytes after the XQ to the loop  *
           *        - Terminates on byte whose XS is not 0                *
           *        - Returns following the byte string                   *
           *        - Loop is assumed to be addressed, with 41 as talker  *
           ****************************************************************
A496 1B0   ildata    pop                 get the rtn adr
A497 330             fetch               fetch the data byte there
A498 23A             c=c+1  m            incr adr
A499 170             push                back on rtn stk
A49A 2F6             ?c#0   xs           is this last DAB
A49B 099             ?cgo   $7126        ..yes, rtn after DAB
A49C 1C7
A49D 099             ?ncxq  $7126        ..no, continue doing DABs
A49E 1C4
A49F 3BB             jnc    ildata

```

```
                    ***************************************************************
                    * DISPNAME -  Display the name of the function whose entry  *
                    *             point is in the adr field of A.               *
                    *           -  The name MUST be 6 or less chars.            *
                    *           -  The last char of the name gets a ':' added.  *
                    *           -  The name is right-justified in chars 1-6.    *
                    ***************************************************************

A4A0 3C1            dispname  ?ncxq   $2CF0          clear the display
A4A1 0B0
A4A2 0BA                      a<>c    m              get the function adr to C Mant
A4A3 34D                      ?ncxq   $05D3          function name to display
A4A4 014
A4A5 3B8                      read    14             remove extra space from right
A4A6 3B8                      read    14             get last char of name
A4A7 3D8                      c<>st                  put into flags
A4A8 288                      setf    7
A4A9 144                      clrf    6              add ':' to last char
A4AA 3D8                      c<>st
A4AB 3E8                      writ    15             put it back in the display
A4AC 0F8                      read    3              shift left 6 chars
A4AD 149                      ?ncgo   $0952          deselect display and RTN
A4AE 026



                    *****************************************************************
                    * ADR2DIS - Address to display                                  *
                    *         - INPUT: M - 4 digit address in d3-0                  *
                    *         - OUTPUT: hex equiv of adr in right 4 chars of disp   *
                    *         - USES: A,B,C                                         *
                    *****************************************************************

A4AF 3D9            adr2dis   ?ncxq   $07F6          select the display
A4B0 01C
A4B1 078                      read    1              get upper 4 of dis

                    * Reverse the digits returned from dis
A4B2 37C                      rcr     12             left 2 digits
A4B3 35C                      r=      12             loop ctr
A4B4 11E            disp10    a=c     ms             transfer char
A4B5 3BE                      rshfa   all
A4B6 2FC                      rcr     13             C left 1
A4B7 3D4                      r=r-1
A4B8 394                      ?r=     0
A4B9 3DB                      jnc     disp10         reverse 12 chars
A4BA 3BE                      rshfa   all            dsp string to A d11-0

                    * Enter hex adr for upper 4 bits of display
A4BB 130                      ldi     $A
A4BC 00A
A4BD 33C                      rcr     1              hex 'A' in C MS
A4BE 0FE                      c<>b    ms             hex 'A' in B MS
A4BF 198                      c=m                    get adr from M
A4C0 07C                      rcr     4              adr in d13-10
A4C1 0AE                      a<>c    all            adr in A, dsp str in C
A4C2 01C                      r=      3
A4C3 33E            disp20    ?a<b    ms             adr char < 0A
A4C4 01F                      jc      disp30
A4C5 010                      ld@r    0              A-F
A4C6 013                      jnc     disp40
A4C7 0D0            disp30    ld@r    3              0-9
A4C8 3EE            disp40    lshfa   all            next adr char
```

```
A4C9 2D4             ?r=     13
A4CA 3CB             jnc     disp20
A4CB 068             writ    1                write upper 4 bits

                * Read lower 4 digits and reverse them
A4CC 038             read    0
A4CD 37C             rcr     12               left 2
A4CE 35C             r=      12               loop counter
A4CF 11E    disp50   a=c     ms               transfer the char
A4D0 38E             rshfa   all
A4D1 2FC             rcr     13               C left 1
A4D2 3D4             r=r-1
A4D3 394             ?r=     0
A4D4 3DB             jnc     disp50

                * add hex adr to right 4 chars of display (lower 4 bits)
A4D5 198             c=m                      get adr
A4D6 07C             rcr     4                adr to d13-10
A4D7 0AE             a<>c    all              adr to A, dsp str to C
A4D8 0BC             rcr     5                position dsp str
A4D9 2DC             r=      13

A4DA 322    disp60   ?a<b    @r               digit < $A?
A4DB 01F             jc      disp70           ..yes, branch
A4DC 182             a=a-b   @r               ..no, subtract 9
A4DD 162             a=a+1   @r
A4DE 0A2    disp70   a<>c    @r               adr char to disp
A4DF 3AE             rshfb   all              move the $A down
A4E0 3D4             r=r-1
A4E1 254             ?r=     9                process digits 13-10
A4E2 3C3             jnc     disp60
A4E3 0FC             rcr     10
A4E4 028             writ    0                write lower 4 bits to dis
A4E5 149             ?ncxq   $0952            deselect the display
A4E6 024
A4E7 3E0             rtn

                ***************************************************************
                *BCDBIN - Decimal to Binary conversion                        *
                *       - INPUT: A - FP number to convert                     *
                *       -        B - Maximum number to convert in S&X         *
                *       - OUTPUT: Binary in A & C S&X                         *
                *       -         A&C are zero except the binary in S&X       *
                *       - USES:  A,C,B S&X,FB                                 *
                * Tests for ALPHA DATA, and DATA ERROR on values greater than *
                * 999 decimal or the number in B.                             *
                * BCDBINX does the conversion on the X reg (rather than A).   *
                * BCDBINXF does the conversion on X with limit of 15 (0xF).   *
                ***************************************************************

A4E8 130    bcdbinXF ldi     16               max value is 15
A4E9 010
A4EA 0E6             c<>b    s&x

A4EB 0F8    bcdbinX  read    3                get the X register
A4EC 10E             a=c     all

A4ED 0AE    bcdbin   a<>c    all              FP to C
A4EE 361             ?ncxq   $14D8            ALPHA DATA check
A4EF 050
A4F0 260             sethex
A4F1 10E             a=c     all
A4F2 356             ?a#0    xs               is XP negative
```

```
A4F3 03F              jc      gobcdbin        ..yes
A4F4 130              ldi     3
A4F5 003
A4F6 306              ?a<c    s&x             XP less than 3 ?
A4F7 085   bcdDE      ?ncgo   $282D           ..no, DATA ERROR
A4F8 0A2
A4F9 0AE              a<>c    all             original FP to C
A4FA 39D   gobcdbin   ?ncxq   $02E7           mainframe BCDBIN
A4FB 008
A4FC 05E              c=0     ms
A4FD 05A              c=0     m               clr C except s&x
A4FE 106              a=c     s&x             return # in A & C
A4FF 326              ?a<b    s&x             # < B ?
A500 3BB              jnc     bcdDE           ..no, DATA ERROR
A501 3E0              rtn

           ****************************************************************
           * BINBCD - Binary to floating point decimal conversion.        *
           *        - INPUT: Binary number in A S&X (up to FFF hex).      *
           *        - OUTPUT: Floating point equiv in C.                  *
           *        - USES: A,C,R - returns in Hex mode.                  *
           ****************************************************************

A502 2A0   binbcd     setdec
A503 0A6              a<>c    s&x
A504 106              a=c     s&x
A505 01A              a=0     m
A506 25C              r=      9
A507 033              jnc     binbcd1
A508 1FA   binbcd0    c=c+c   m
A509 1FA              c=c+c   m
A50A 1FA              c=c+c   m
A50B 1FA              c=c+c   m
A50C 11A              a=c     m
A50D 05A   binbcd1    c=0     m
A50E 2FC              rcr     13
A50F 23A              c=c+1   m
A510 27A              c=c-1   m
A511 21A              c=c+a   m
A512 3DC              r=r+1
A513 354              ?r=     12
A514 3A3              jnc     binbcd0
A515 260              sethex
A516 01B              jnc     binbcd2
A517 35C              r=      12
A518 0AE              a<>c    all
A519 130   binbcd2    ldi     $009
A51A 009
A51B 11A              a=c     m
A51C 2FA              ?c#0    m
A51D 02F              jc      binbcd4
A51E 04E              c=0     all
A51F 3E0              rtn
A520 266   binbcd3    c=c-1   s&x
A521 3FA              lshfa   m
A522 342   binbcd4    ?a#0    @r
A523 3EB              jnc     binbcd3
A524 0BA              a<>c    m
A525 3E0              rtn
```

```
           ***************************************************************
           * FINDID for MC00506A                                         *
           * Initializes the loop.                                       *
           * Searches from the primary device, leaving the first         *
           *   MC00506A found as the selected device in PIL register 5.*
           *   This device is also left addressed as a talker.           *
           *                                                             *
           * USES: A,B S&X ,C,F8,F9,R, 3 Subroutine levels.              *
           ***************************************************************

A526 36D   findmc    xqrel   piltst        Test for existence of PIL
A527 08C
A528 176
A529 18D             ?ncxq   $7063         address the loop
A52A 1C0
A52B 39D             ?ncxq   $77E7         check XMIT err
A52C 1DC
A52D 086             b=a     s&x           save # of devices in B
A52E 155             ?ncxq   $7155         set primary device in r5, r6
A52F 1C4
A530 104             clrf    8
A531 2C9   fndmc10   ?ncxq   $70B2         TAD (r5)
A532 1C0
A533 064             selp    4
A534 285             con     $285          RDY type frame
A535 0A4             selp    5
A536 189             con     $189          SDI frame
A537 31C             r=      1             field defn
A538 36D             xqrel   fndmc30       push adr of string
A539 08C
A53A 144
A53B 04D             con     $4D
A53C 043             con     $43
A53D 030             con     $30
A53E 030             con     $30
A53F 035             con     $35
A540 030             con     $30
A541 236             con     $236          last char: MC00506

A542 0A6   fndmc20   a<>c    s&x           get back the DAB
A543 280             ilw     2             retransmit it
A544 244   fndmc30   clrf    9
A545 041             ?ncxq   $7110         get a DAB
A546 1C4
A547 24C             ?fset   9             set if frame not DAB
A548 087             jc      fmcnext       try next device
A549 106             a=c     s&x           save DAB in A
A54A 1B0             pop                   get the current table adr
A54B 330             fetch                 get the table character
A54C 23A             c=c+1   m             incr table adr
A54D 170             push                  save it for next time
A54E 36A             ?a#c    r<-           DAB same as table ?
A54F 027             jc      fmcnrd        ..no, send NRD
A550 2F6             ?c#0    xs            end of table ?
A551 38B             jnc     fndmc20       ..no, match next char
A552 108             setf    8             indicate device found
A553 0A6   fmcnrd    a<>c    s&x           get DAB back to C
A554 020             xq->go                pop the rtn adr
A555 3D9             ?ncxq   $70F6         NRD
A556 1C0
A557 013             jnc     *+02          skip the extra pop off
A558 020   fmcnext   xq->go                pop the rtn adr
A559 10C             ?fset   8
```

```
A55A 360                  ?crtn                    RTN if MC00506A was found
A55B 066                  a()b    s&x
A55C 086                  b=a     s&x              copy # of devices to A
A55D 185                  ?ncxq   $7161            try next device (if AUTOID)
A55E 1C4
A55F 013                  jnc     notfound         ..error - MC00506A not found
A560 2BB                  jnc     fndmc10          SDI to next device

A561 2CD    notfound      ?ncxq   $77B3            mssg to disp
A562 1DC
A563 00E                  con     $0E
A564 00F                  con     $0F
A565 020                  con     $20
A566 00D                  con     $0D
A567 003                  con     $03
A568 030                  con     $30
A569 030                  con     $30
A56A 035                  con     $35
A56B 030                  con     $30
A56C 036                  con     $36
A56D 201                  con     $201             mssg: NO MC00506A
A56E 3F9                  ?ncgo   $7CFE            error halt
A56F 1F2



            **************************************************************
            *ERRMSSG - Display following characters as an error mssg.    *
            *        - Execution returns following error string.         *
            *        - Functions using HPIL should use $77B3             *
            **************************************************************

A570 3A1    errmssg       ?ncxq   $22E8
A571 088
A572 3C1                  ?ncxq   $2CF0            Clear the display
A573 0B0
A574 3BD                  ?ncgo   $07EF            Disp the mssg and RTN
A575 01E


            **************************************************************
            *PILTST - Test for existence of HPIL Chip.                   *
            *       - RTN if it exists, else "NO HPIL" error.            *
            *       - The test is simple - look for 28 (XROM #) at $7000*
            *       - USES: C, A S&X, R                                  *
            **************************************************************

A576 04E    piltst        c=0     all
A577 15C                  r=      6
A578 1D0                  ld@r    7                C adr field = $7000
A579 330                  fetch                    get the word @ $7000
A57A 106                  a=c     s&x              word to A
A57B 130                  ldi     28               XROM # is 28
A57C 01C
A57D 366                  ?a#c    s&x              is PIL Chip existent ?
A57E 3A0                  ?ncrtn                   ..yes, return
A57F 36D                  xqrel   errmssg          ..no, disp "NO HPIL"
A580 08C
A581 170
A582 00E                  con     $0E
A583 00F                  con     $0F
A584 020                  con     $20
A585 008                  con     $08
A586 010                  con     $10
A587 009                  con     $09
A588 20C                  con     $20C             mssg: NO HPIL
```

```
                ****************************************************************
                *ERRHALT - Assumes a message in the display (ERRMSSG).         *
                *        - Left justifies the mssg and halts (F25 checked).    *
                *        - Functions using HPIL should use $7CFE (2 bytes).    *
                ****************************************************************

A589 3DD        errhalt   ?ncxq   $2BF7         Left justify display
A58A 0AC
A58B 108                  setf    8
A58C 201                  ?ncxq   $1C80         Mssg to printer
A58D 070
A58E 3ED                  ?ncgo   $22FB         Mainframe error handler
A58F 08A






                **** Beginning of Free Space
A590 000        begfree   con     0



                **** End of Free Space
AFEE                      org     $AFEE
AFEE 000        endfree   con     0


AFF0                      org     $AFF0

                * Beg and end of free space ptrs
AFF0 005                  fat     begfree
AFF1 090
AFF2 00F                  fat     endfree
AFF3 0EE



                * Polling points

AFF4 000                  con     0             FF4 pause poll
AFF5 000                  con     0             FF5 prgm line/IO activity
AFF6 000                  con     0             FF6 wakeup by peripheral
AFF7 000                  con     0             FF7 power down (OFF key)
AFF8 000                  con     0             FF8 IO service flag set
AFF9 000                  con     0             FF9 wakeup by ON key
AFFA 000                  con     0             FFA memory lost


                * ROM Revision code (FFB - FFE)

AFFB 003                  con     $03           Rev: MC-1C
AFFC 031                  con     $31
AFFD 003                  con     $03
AFFE 00D                  con     $0D


                * ROM checksum
AFFF 217                  cksm


00000 errors
00000 warnings
00107 symbols defined
```