ERAMCO SYSTEMS

ES 84091A
MLDL ROM
annotated listing

September 1984

84091A-M005001

Printed in the Netherlands

ERAMCO SYSTEMS
Kruiszwin 2102
1788 LN Den Helder

```
*
*
*
* XROM number is 11
*
* Number of functions is 28
*
        ORG         $000
*
        FCB         $00B    XROM number is 11
        FCB         $01E    number of functions is 28
        FAT         $FDE    ERAMCO-MLDL
        FAT         $4AE    RAMWR
        FAT         $D8A    MMTORAM
        FAT         $7A4    AFAT
        FAT         $756    DFAT
        FAT         $680    MOVE
        FAT         $6FC    CLBL
        FAT         $EE2    COPYR
        FAT         $E5F    ROMSUM
        FAT         $CAE    REG>ROM
        FAT         $7DA    --
        FAT         $316    COMPILE
        FAT         $6D3    LOCA
        FAT         $6B4    LROM
        FAT         $6EC    COD
        FAT         $6F6    DECOD
        FAT         $F01    ROMCHKX
        FAT         $C4B    ROM>REG
        FAT         $803    MNEM
        FAT         $E8B    DISASM
        FAT         $44C    CAT
        FAT         $715    CBT
        FAT         $F88    SYNT
        FAT         $F9D    GE
        FAT         $7DA    --
        FAT         $364    SAVEROM
        FAT         $3BE    GETROM
        FAT         $089    CMPDL
        FAT         $047    IPAGE
        FAT         $FE3    MKPR
*
        FCB         $000
        FCB         $000    end of FAT
*
*
*
*****************************************************************************
*                                                                           *
*                          BEGIN OF ASSEMBLY LISTING                        *
*                                                                           *
*****************************************************************************
*
```

```
*
*
* Initialize ram PAGE
*
        NAM       20 IPAGE    say we want a two digit numeric argument
*
        FCB           $000    say non programmable
        A<>B      X           save xrom number in "B(X)"
        GOSUB         VPAG    page number to "C[6]" and $400 to "C(X)"
        C=0       X           get rid of the loop counter
        R= 5
        A=C       A
        A=A-1     R<
        A<>C      M           end of page to "C(M)" start to "A(M)"
*
* here the page is completely cleared
*
IPAGE1  WRIT
        C=C-1     M
        ?A<C      M
        JC-           IPAGE1
*
* now we have to load the xrom number, the start address of the rom name
* and the number of functions
*
        C=B       X
        WRIT                  write the xrom number
        C=C+1     M
        LDI           $001
        WRIT                  say we have only a name
        C=C+1     M
        C=C+1     M
        LDI           #091
        WRIT                  load address of rom name
        A<>C      X
        RCR 3
        A<>C      X
        RCR 11                create address of rom name
        LDI           $3E0
        WRIT                  rom name is terminated by a RTN
        M=C                   save addreess of name
*
* we convert the name into the lcd characters by getting the name out of
* alpha to the display. the name is left justified in the lcd
*
        ?NC XQ        ARGOUT  convert name
        ?NC XQ        ENLCD
        C=M
        C=C-1     M           create address of first character
        R= 13
        LD@R 10               load only 11 characters
        A<>C      A
*
* the rom name has to be written in inverted order. this is accomplished
* by getting the characters from the left of the display and write them to
* the ram page
*
```

```
IPAGE2  READ E              get leftmost character and left justify
        A<>C     M
        WRIT                write character of name
        C=C-1    M
        A<>C     M
        A=A-1    MS         are we finished yet
        JNC-         IPAGE2 no, do the rest
*
* the end of the rom name is flagged by bit 7 of the last character. we
* have to tell the mainframe where our rom name ends by setting this bit
*
        A<>C     M
        C=C+1    M
        FETCH               get last character
        C<>ST
        SETF 7
        C<>ST               flag the character
        WRIT
        ?NC XQ       ENCP00
        GOTO         EXIT   say "READY" and give address of first free word
*
*
*
* CoMPile and Delete Labels
*
        NAM      10 CMPDL   indicate an alpha prompt
*
        FCB         $000    say this program is non programmable
*
* indicate we want a pre compiled program by clearing flag 0,1 and 2
*
        C=0      A
        RAMSLCT
        READ E
        RCR 3
        R= 1
        C=0     R<          clear flags 0,1, and 2
        RCR 11
        WRIT E
*
* before compiling a program it is necessary to make sure that the program
* has a normal END. in case we have a program that is the last program in
* memory, we make the .END a normal one and load a new .END
*
COMPIL  GOSUB       FPAL
        RCR 8
        A<>C    R<          address of end of program to "A[3-0]"
        C=0      A
        RAMSLCT
        READ C              get address of .END
        ?A#C     X          is it the .END
        JC+         NO.END  no, continue
        C=C-1    X          yes, create address of new .END
        C<>B     A          and save reg C in "B"
        ?NC XQ      AVAIL
        ?C#0     X          is there still an empty register under the .END
        ?NC GO      PACKE   no, pack and say "TRY AGAIN"
```

```
*
* we do have enough room to create the new .END this end has a link of one
* register to the old .END
*
        C=B     X
        RAMSLCT                 address the new .END
        C=0     A
        R= 5
        LD@R C                  say it is an end
        LDI         #120        link is 1 reg, make it .END, decompiled
        WRITDAT                 new .END to memory
        C=B     X
        C=C+1   X
        RAMSLCT
        READDAT                 get old .END
        R= 1
        LD@R 0                  make it a normal END
        WRITDAT                 END to memory
        C=0     A
        RAMSLCT
        C<>B    A
        WRIT C                  store the updated reg C
*
*
*
NO.END  GOSUB       FPAL        find links
        CLRF 10                 say we are in ram
*
* FPAL returns from FLINK. the address of the end of the program is then
* returned in "C[11-8]". this is used by CPGMHD to find the start of the
* program we have to handle. next thing to do is to pack the program.
*
        RCR 8
        A=C     R<
        ?NC XQ      CPGMHD
        ?NC XQ      PUTPCF
        ?NC XQ      PACKN
*
* tell the user we are handling the 2 byte goto's
*
        ?NC XQ      CLLCDE
        ?NC XQ      MESSL
        RMB         $0A         "COMPL 2B G"
        ?NC XQ      LEFTJ
        ?NC XQ      ENCP00
        ?NC XQ      GETPC
*
* the main loop for the handling of the 2 byte goto's is entered with the
* address of a function in "A[3-0]" in MM format
*
DO2B    A<>C    R<
        A=C     R<          copy PC to "C[3-0]"
        N=C                 and to "N"
        ?NC XQ      NXBYTA  get the first byte of this instruction
*
* split the function byte into "A(MS)" and "A[0]". in this way we can save
* the number we have to jump to in "A(MS)" and test if we have a two byte
* GTO
*
```

```
        C=0     XS
        RCR 1
        C=0     XS
        A<>C     A
        LDI         $00B
        ?A#C     X          do we have a two byte GTO
        JNC+        F2BINS  yes, go handle it
        C=N
        A=C     R<
        ?NC XQ      NXLSST  get address of next instruction into "A[3-0]"
        ?FSET 6             have we reached the end of the program yet
        JNC-        DO2B    no, try the next instruction
        JNC+        STPP3   yes, go handle the 3 byte GTO and XEQ
*
* here we have found the 2 byte instruction we have to compile. first thing
* we must do is to get the PC at the right position for calculating the
* distance to its label. therefore the PC has to be positioned to the last
* byte of the GTO
*
F2BINS  A<>B     A          save label number in "B(MS)"
        C=N
        A=C     R<
        ?NC XQ      INCAD2
        ?NC XQ      PUTPC   PC is positioned to the last byte of the GTO
        CLRF 8              say we are jumping forwards
        C<>B       A
        C=0      X
        RCR 13              get label number to "C(X)"
        C=C-1    X          decrement it by one for SEARCH
        A=C      X          and put it in "A(X)" for SEARCH
*
* SEARCH returns the address of the wanted label in "C[3-0]" or if it is
* not found it will clear "C" also is the label number saved in "G". this
* is important, because INBYT expects the byte to be inserted in "G"
*
        GOSUB       FLABEL  find address of label, is label found
*
* FLABEL will return at the next instruction if the label is found, else
* it will skip this instruction
*
        JNC+        5       yes, go calculate the distance and direction
        GOTO        NOLBLF  no, error with "NO LBL NN" and position to GTO
*
* stepping stone
*
STPP4   JNC-        DO2B
*
        M=C                 save address of label in "M"
        ?NC XQ      GETPC   address last byte of GTO to "A[3-0]" + "C[3-0]"
        N=C                 and save it in "N"
        C=M
*
* in the next part we test if we have to jump backwards or forward and we
* compute the distance in registers and bytes to the label
*
```

```
        ?A#C     X          are we in the same register
        JC+         4       no, test if we are in a higher register
        ?A<C     R          is it a jump forward
        JNC+        6       yes, we can just compute the distance
        JNC+        3       go say we are jumping backwards
        ?A<C     X          is it a jump forward
        JNC+        3       yes, we can just compute the distance
        SETF 8              say we are jumping backwards
*
* when we jump backwards we have to make sure we get valid results when we
* compute the distance. this is done by exchanging the label and gto
* addresses because the label addres is in fact in a higher numbered part
* of MM and distance is calculated by subtracting "C" from "A"
*
        A<>C    R<
        ?NC XQ      CALDSP  go compute the distance
*
* when the distance is more then 15 register we have to convert the 2 byte
* GTO into a 3 byte GTO
*
        ?A#0    XS          is the distance more then 256 registers
        JC+         MK3BG   yes, go convert
        LSHFA    X          no, left justify number of registers
        ?A#0    XS          is it more then 15 registers
        JC+         MK3BG   yes, go convert
        LSHFA    X          no, left justify number of registers
        RSHFA   R<          put # reg and # bytes into "A(X)"
        C=N
        A<>C    R<          address of second byte to "A[3-0]"
        ?FSET 8             is the jump forward
        JNC+        2       yes, leave direction bit zero
        C=C+1    R          no, make direction bit one
        C=C+C   R<
        C=C+C   R<
        C=C+C   R<
        C=C+C    X
        RCR 2               left justify distance byte in "C[1-0]"
        ?NC XQ      PTBYTA  and put it in the GTO
        JNC-        STPP4   do the next instruction
*
* stepping stone
*
STPP3   JNC+        DO3BGX
*
*
* a three byte GTO is made by inserting one byte. the byte is inserted
* after the last byte of the two byte GTO. the byte to be inserted is
* expected to be found in "G". this byte has to have the number of the
* label wee want to jump to. this is all acounted for, because the SEARCH
* routine transferred the byte to "G" and the address in "N" is of the last
* byte of the 2 byte GTO. also is the first byte of the 2 byte GTO replaced
* with $0D0, the identifier of a 3 byte GTO
*
```

```
MK3BG   C=N
        A=C     R<          get address of last byte to "A[3-0]"
        ?NC XQ      INBYT   insert the byte at next address
        R= 3
        ?NC XQ      DECADA
        ?NC XQ      DECADA  place the address at the first byte of the GTO
        LDI         $0D0
        ?NC XQ      PTBYTA  and make it a 3 byte GTO
        C=0     A
        RAMSLCT
        GOTO        COMPIL  find start of program, pack and start again
*
* in the next part the 3 byte GTO and XEQ are handled
*
* the address of the first instruction is saved and the message
* "COMPL 3B G/X" is placed in the display to keep the user informed of what
* is happening.
*
D03BGX  ?NC XQ      PUTPC   save PC
        ?NC XQ      CLLCDE
        ?NC XQ      MESSL
        RMB         $0C     "COMPL 3B G/X"
        ?NC XQ      LEFTJ
        ?NC XQ      ENCP00  message is in display
        ?NC XQ      GETPC   get PC back to start
NXTINS  A<>C    R<
        A=C     R<
        N=C                 copy current address to "A[3-0]" and "N"
*
* in this part we getthe first byte of an instruction out and are testing
* if it is a 3 byte GTO or a XEQ. if so, we are compiling it
*
        ?NC XQ      NXBYTA  get first byte of instruction
        C=0     XS
        RCR 13              get row byte to "C(XS)"
        C=C+1   XS          is it a text identifier
        JC+         ROWF    yes, go do the next instruction
        C=C+1   XS          is it a XEQ
        JC+         DOCOMP  yes, go compile it
        C=C+1   XS          is it a GTO
        JC+         DOCOMP  yes, go compile it
*
* we have no GTO or XEQ, so we just find the next instruction and loop
* if we haven't reached the end of the program yet
*
ROWF    C=N
        A=C     R<          get current address to "A[3-0]"
        ?NC XQ      NXLSST  find next instruction
        ?FSET 6             have we reached the end
STPP2   JNC-        NXTINS  no, do next instruction
        GOTO        ?READY  yes, decide wether we can exit or do more
*
* here we are first trying to find the desired label. the label number is
* given in the third byte of the instruction. also we find a direction bit
* here. this bit must be cleared because if it is set, we would search for
* a label number that is 128 to high and will always find an error then
*
```

```
DOCOMP  ?NC XQ      PUTPC   save address of first byte in PC
        SETF 8              say jump is forward
        ?NC XQ      GT3DBT  byte with label number in "ST" address in "M"
        CLRF 7              clear direction bit
        C<>ST
        A=C       X         byte to "A(X)" for SEARC1
        SETF 6              say we are at  first byte of a 3 byte instr.
*
* FLABEL returns the address of the label if it exists. if it exists, it
* will execute the next step, else the next step is skipped
*
        GOSUB       FLABEL  find label address, have we found it
        JNC+        LBLF
        GOTO        NOLBL   no, go error "NO LBL NN"
*
* stepping stone
*
STPP1   JNC-        STPP2
*
*
* in this part we find out in which direction we have to jump.
*
LBLF    A=C     R<          address of label to "A[3-0]"
        C<>M                address of GTO to "C[3-0]"
        ?A#C     X          are we in the same register
        JC+         4       no, test if we are in a higher register
        ?A<C     R          are we jumping backwards
        JNC+        DIST    no, go calculate the distance
        JNC+        3       yes, say it and fix addresses for calculation
        ?A<C     X          are we jumping backwards
        JNC+        DIST    no, go calculate the distance
        CLRF 8              say we jump backwards
        A<>C    R<          and fix the addresses for the calculation
*
* here we are computing the distance in registers and bytes
*
DIST    C=A-C    X          calculate number of registers
        C=A-C    R          calculate number of bytes
        JNC+        4
        C=C-1    X          if we pass register boundary decrement reg 1
        C=C-1    R
        C=C-1    R          and set byte count to right value
*
* now we have found the distance in registers and bytes. in bit 3-1 of
* digit "C[3]" we have the byte distance. bit 0 is used to have one bit
* of the register count. this count is placed in bit 0 of "C[3]" and
* "C[2-1]".
*
```

```
         C=C+C     X
         C=C+C     X
         C=C+C     X
         C=C+C     X              left justify reg count and use bit 0 if needed
         JNC+          2
         C=C+1     R              we have to use bit 0
         RSHFC    R<              place distance digits in "C(X)"
         C<>B      X              and save them in "B(X)"
         ?NC XQ        GETPC      get address of first byte
         ?NC XQ        GTBYTA     and get the byte
         RCR 12                   place row digit in "C[3]"
         C=B       X              add the distance digits
         RCR 2                    leave first byte in "C[1-0]"
         C<>B      M
         C<>B     MS              save secomd byte in "B[13-12]"
         ?NC XQ        PTBYTA     put first byte in memory
         ?NC XQ        INCADA
         C=B       A
         RCR 12                   get second byte
         ?NC XQ        PTBYTA     put second byte in memory
         ?NC XQ        NXBYTA     get last byte
         C<>ST
         CLRF 7
         ?FSET 8                  are we jumping backwards
         JNC+          2
         SETF 7                   yes, indicate a jump backwards
         C<>ST
         ?NC XQ        PTBYTA     put byte in memory
         JNC-          STPP1      go do the next instruction
*
?READY   C=0       A
         RAMSLCT
         READ E                   get flags
         RCR 3
         C<>ST
         ?FSET 0                  do we have to do a precompiled program
         JNC+          4
         GOTO          READY      no, do exit or return to calling program
         SETF 0                   say we do compile for a second run
         SETF 1                   say we use the user regs as label addresses
         C<>ST
         RCR 11
         WRIT E                   save flags
*
         ?NC XQ        GETPC
         ?NC XQ        FLINK
         RCR 8
         A=C      R<
         ?NC XQ        CPGMHD
         ?NC XQ        PUTPC      position PC to start of program
*
* here the loop for finding GTO or XEQ indirect functions start. we fetch
* the first byte fo every function and testif it is IND. if so, we exit
*
```

```
FNDIND  ?NC XQ      NXBYTA
        A<>B    R<          save current address in "B[3-0]"
        A<>C     X
        A=0     XS          instr. byte to "A(X)"
        LDI         $0AE
        ?A#C     X          is it XEQ/GTO indirect
        JC+         4       no
        GOTO        X/GIND  yes, position PC to this line and exit
        A<>B    R<
        ?NC XQ      DECADA  address to a standard position
        ?NC XQ      NXLSST  find next instruction
        ?FSET 6             are we finished yet
        JNC-        FNDIND  no, test next instruction
*
* the addresses of the labels are stored in the user registers. therefore
* it is needed to clear these registers and to find the address of the
* first register. this address is stored in reg 10 that is used to keep
* track of the next free register.
*
        C=0      A
        RAMSLCT
        READ C
        RCR 3               get address of reg 0
        A<>C     X
        READ 10
        A<>C     X
        WRIT 10             put it in reg 10
        SETF 8
        ?NC XQ      CLR     clear the user rgisters
*
* in this loop the labels are searched, and if one is found the label is
* cleared and its address is saved in the user registers and its number.
*
STDLBL  ?NC XQ      GETPC   get start address of program
        M=C                 save it in "M"
DLLBL1  ?NC XQ      NXBYTA  get first byte of an inst.
        A<>B    R<          save current address
        R= 1
        C=C-1    R          is it row 0
        JC+         LBL014  yes, go handle row 0
        A=C      X
        LDI         $0BF    first byte of 2 byte LBL - $10
        ?A#C    R<          is it a 2 byte label
        JNC+        LBL>14  yes, go handle it
DLLBL2  R= 3
        A<>B    R<          get current address back to "A[3-0]"
        ?NC XQ      DECADA  to standard position
        ?NC XQ      NXLSST  find next instr.
        ?NC XQ      PUTPC   save address in PC in case we have to pack
        ?FSET 6             have we reached the end yet
        JNC-        DLLBL1  no, do next instr.
        GOTO        NO.END  yes, go compile the program
*
* in this part we seperate the numeric labels from the local labels. in
* case we have a local label we must not delete it. it is necessary to
* store the address of the label for the label find routine.
*
```

```
LBL>14  R= 3
        A<>B    R<
        ?NC XQ      NXBYTA  get label number
        A<>B    R<          save address
        A=C      X
        A=0     XS          label number to "A(X)"
        LDI         $064
        ?A<C     X          is it a local label
        JC+         3       no
        A<>B    R<
        JNC-        DLLBL1
        R= 0
        A<>C     X
        C=C+1    X          increment LBL for it is decremented when stored
        G=C                 save label number in "G"
        R= 3
        A<>B    R<          get address of second byte
        C=0      A
        ?NC XQ      PTBYTA  delete the second byte
        ?NC XQ      DECADA  point to first byte
        JNC+        SECBYT
*
* here is checked if we have to do with a null byte, and if so we return to
* the main loop
*
LBL014  LD@R 0              clear the first digit and set pointer to 0
        ?C#0     R          is it a null byte
        JNC-        DLLBL2  yes, go do the next instr.
        G=C                 save label number in "G"
        R= 3
        A<>B    R<          current address to "A[3-0]"
*
SECBYT  C=0      A
        ?NC XQ      PTBYTA  delete the byte
*
* here we are combining the start address of the program, the address of
* the label and its number
*
        R= 3
        ?NC XQ      GETPC   get the address of the label to "A[3-0]"
        C=M                 get start address of program
        RCR 10
        A<>C    R<          add label address
        RCR 11
        R= 0
        C=G                 add label number
        C=C-1    X
        C=0     XS
        C=0     MS
        C=C+1   MS          type it as alpha data
        A<>C     A
        C=0      A
        RAMSLCT
        READ 10             get storage address
        C=C+1    X
        WRIT 10             update storage address
        C=C-1    X          create the right storage address
        RAMSLCT
        A<>C     A
        WRITDAT             put the label address in the register
```

```
*
* after we have cleared the label we have to pack the program again and
* start from the beginning.  this is necessary,  because we have created a
* null byte in the memory. we have to change the status of the program from
* packed into unpacked for PACKN skipps packed programs.
*
         ?NC XQ       GETPC
         ?NC XQ       FLINK    get address of END to "C[11-8]"
         ?NC XQ       FIXEND   make the program unpacked
         ?NC XQ       PACKN
         GOSUB        FPAL
         RCR 8
         A<>C     R<
         ?NC XQ       CPGMHD
         ?NC XQ       PUTPC    start address of program to PC
         GOTO         STDLBL
*
*
*
NOLBL    ?NC XQ       GETPC    set PC to the 2 byte GTO that has no LBL
         ?NC XQ       DECADA
         JNC+         3
NOLBLF   ?NC XQ       GETPC    set the PC to the GTO/XEQ that has no LBL
         ?NC XQ       DECADA
         ?NC XQ       PUTPC
         ?NC XQ       CLLCDE
         ?NC XQ       MESSL
         RMB          $07      "NO LBL_" put this in the display
         R= 0
         C=G
         A=C       X           get label number to "A(X)"
         A=0      MS
         A=0      XS
         A=A+1    MS
         A=A+1    MS           say we want two digits in the display
         ?NC XQ       GENNUM   append label number to the display
         ?NC XQ       LEFTJ
         ?NC XQ       ENCP00
         ?NC XQ       MSGDLY
         ?NC GO       ERR110   put total message in the display
*
* here we indicate the user that we are ready and place the PC at the first
* step of the compiled program
*
READY    ?NC XQ       CLLCDE
         ?NC XQ       MESSL
         RMB          $05      "READY"
         ?NC XQ       LEFTJ
         ?NC XQ       ENCP00
         ?NC XQ       STMSGF   message is in display
*
```

```
         GOSUB        FPAL
         RCR 8
         A=C      R<
         ?NC XQ       CPGMHD  get start of program to "A[3-0]"
         ?NC XQ       PUTPCF  put user at start of program
         READ E
         RCR 3
         C<>ST
         ?FSET 2
         ?NC GO       TONE7X  give audio warning that we finished and exit
         READ P
         RCR 7
         GOTO ADR
*
*
*
* Find Program And Links
*
* Entry: register Q holds the inverted right justified name of the program
*         you want to have the begin and end address from
* Exit : "C" has links to the desired label as returned by FLINK
* Uses : all
*
FPAL     R= 3
         READ Q
         M=C                  get a copy of the program name to "M"
         ?C#0     R<          do we have to do the current program
         JNC+         CURPG
         ?NC XQ       ASRCH   no, find program
         ?C#0     R<
NONEX    ?NC GO       ERRNE   say program is nonexistent
         ?FSET 9
         JC-          NONEX   it is a microcode function
         ?FSET 2              is the program hold in main memory
         JNC+         PFOUND  yes
ROM      ?NC XQ       ERROR
         FCB          $06A    specify message 'ROM'
CURPG    ?FSET 10             are we in ROM at the moment
         JC-          ROM     yes
         ?NC XQ       GETPC
PFOUND   ?NC GO       FLINK   find program head and it's END *
*
*
*
X/GIND   A<>B     R<          get address of first byte of GTO/XEQ ind
         ?NC XQ       DECADA
         ?NC XQ       PUTPC   set PC to the error line
         ?NC XQ       CLLCDE
         ?NC XQ       MESSL
         RMB          $0B     "GTO/XEQ IND" message to display
         ?NC XQ       LEFTJ
         ?NC XQ       ENCP00
         ?NC XQ       STMSGF
         LDI          $004
         ?NC GO       TONEB   give low warning beep if audio is enabled
*
*
*
```

```
*
* this is the label find routine. we decide here if we have to do with
* the normal compiler or if we have to use the user registers with the
* addresses of the labels and their label number. it depends on flag 1 in
* reg E which compiler we are doing
*
FLABEL  A=0     XS
        LDI         $064
        ?A<C     X
        JNC+        LOCLBL
        C=0      A
        RAMSLCT
        READ E
        RCR 3
        C<>ST               get compiler flags
        ?FSET 1             do we have to use the user regs
        JC+         DUSREG  yes, go handle the user regs
*
* it depends on the setting of flag 6 of the original flag status if we
* have to do 2 or 3 byte GTO/XEQ
*
        C<>ST               get old flagstatus back
        ?FSET 6             is it 3 byte instr.
        JC+         4       yes, do three byte instr.
        ?NC XQ      SEARCH  no, do 2 byte
        JNC+        3
LOCLBL  ?NC XQ      SEARC1  do 3 byte
*
* if we have found the label we will do a normal return to the instruction
* after the calling subroutine, else we skipp this instruction and do the
* next instruction.
*
        ?C#0     A          does the desired label exist
        ?C RTN              yes, do a normal return
        POP                 no, skipp instruction after the call
        C=C+1    M
        GOTOADR
*
* for a correct compilation of the program we must start searching for the
* desired label from our current program position. this makes it neccessary
* to search for the register with the address after our current position.
* when this register is found, we can search for the desired label. when we
* have reached the end of the labels and haven't found the desired label
* yet, we must start searching from the start of the program. this means we
* have to start with the first user register.
*
* this is also the procedure that is followed by the normal search routine
* in the mainframe
*
DUSREG  C<>ST               restore flags for compile
        A<>C     X          get the desired label number
        R= 0
        G=C                 and store it in "G"
*
* in this loop we search for a label with an address that lays behind the
* current position in the program. we exit when we have found such a label
* or when we have reached the end of the labels.
*
```

```
         R= 3
         ?NC XQ       GETPC    current address to "A[3-0]"
         READ C
         RCR 3                 address of first reg to "C(X)"
*
FLADD    RAMSLCT               select a label register
         C<>B      X           save address counter in "B(X)"
         READDAT
         RCR 3                 get address right justified
         ?C#0      A           have we had all labels
         JNC+         FLBL     yes, go find the desired label
*
* in the next instructions we test if the label is after the current
* position in the program.
*
         ?A#C      X           same register
         JC+          FLBL     no, test if higher reg
         ?A<C     R<           is address of label higher
         JNC+         6        yes, go search for desired label
         C<>B      X           we have not found such a label yet and are
         C=C+1     X           going to do the next label
         JNC-         FLADD    go try next register
         ?A<C      X           is address of label higher
         JC-          4        no, do next label
*
* after we have found where we must start searching for the labels we will
* do it here. if the desired label is not found yet when we reach the end
* of the labels we will start at the beginning of the labels.
*
FLBL     R= 0
         C=G
         A=C       X
         A=0      XS           desired label number to "A(X)"
         C<>B      X
*
DNLBL    RAMSLCT
         C<>B      X           save label counter
         READDAT               get a label
         ?C#0      A           have we had all labels
         JNC+         WRAP     yes, start at the beginning of the labels
         ?A#C      X           is this the desired label
         JNC+         LBLFND   yes, exit
         C<>B      X
         C=C+1     X           point to next label
         JNC-         DNLBL    do next label
*
WRAP     C=0       A
         RAMSLCT
         READ C
         RCR 3                 get counter for first address and go search
         JNC-         DNLBL
*
* we have to exit the same as the SEARCH and SEARC1 routine. this means
* status registers must be selected and address of label is in "C[3-0]"
* and label number is in "G"
*
```

```
LBLFND  RCR 3                   address to right position
        R= 3
        A=C     R<              save in "A" while we select status regs
        C=0      A
        RAMSLCT                 select status regs
        A<>C    R<              address of label back to "C[3-0]"
        RTN                     return to calling point
*
*
*
* COMPILE
*
        NAM      10 COMPILE     indicate an alpha prompt
*
        FCB         $000        say non programmable
        C=0      A
        RAMSLCT
        READ E
        RCR 3
        ST=0
        SETF 0                  say we want the normal compile routine
        C=ST
        RCR 11
        WRIT E                  save flag status for compile
        GOTO        COMPIL      do it
*
*
*
* this routine checks if the HPIL rom is present
*
PILTST  C=0      A
        R= 6
        LD@R 7
        FETCH                   get xrom number of page 7
        A=C      X
        LDI         $01C
        ?A#C     X              is it the HPIL chip
        ?NC RTN                 yes, then return to calling routine
        GOSUB       INERR       no, initialize for error message
        RMB         $07         "NO HPIL"
*
ERREX   ?NC XQ      LEFTJ       exit error message and return to mainframe
        SETF 8
        ?NC XQ      MSG105
        ?NC GO      ERR110
*
*
*
* this routine checks if the page address is valid and it sets the loop
* counter to $400. in the loop we store 4 rom words, so we do a complete
* rom block of 4K.
*
```

```
VPAG    READ X
        ?NC XQ      BCDBIN
        A<>C    X           get wanted page to "A(X)"
        LDI         $010
        ?A<C    X           is it a valid page address
        JC+         VPG
        GOSUB       INERR   no, put message to display and exit
        RMB         $09     "PAGE > 15"
        JNC-        ERREX
*
VPG     A<>C    X
        RCR 8               start address of page to "C(M)"
        R= 2
        LD@R 4              set loop counter for 4K
        RTN
*
*
*
* here is the initialization done for an error message
*
INERR   ?NC XQ      ERRSUB
        ?NC XQ      CLLCDE
        ?NC GO      MESSL
*
*
*
* SAVEROM
*
        NAM         SAVEROM
*
        GOSUB       PILTST
        GOSUB       VPAG
        C=0     MS
        ?NC XQ      780B    check if there is already afile with this name
        C=M
        ?C#0    A           did the file exist
        ?C GO       7692    yes, say DUP FL NAME and exit to mainframe
*
* the romfiles do have an file type of 7. this is done so we can't destroy
* it with any of the normal HPIL functions. the length is 640 registers and
* these register are stored in 20 records. the file is automatically
* secured when it is created.
*
        C=0     A
        LDI         $078    file type 7 and secured
        RCR 2
        LDI         $280    640 registers
        RCR 4
        LDI         $014    20 records
        RCR 4
        ?NC XQ      76BC    create the file
        ?NC XQ      7F72    set tape on start of file and BP to zero
*
        LDI         $0A2
        ?NC XQ      70BA    give DDL 2 - write mode
*
        ?NC XQ      70DA    give DDL 0 - write buffer 0
*
        ?NC XQ      77E7    do error check
*
```

```
        SELP 4
        FCB         $005    set REG 1 for DAB
*
        GOSUB       VPAG    get rom page and loop counter to "C"
*
* the main loop of SAVEROM is started here. we store 5 bytes after each
* other and these bytes are containing the four lower bytes and one upper
* byte. they are stored sequentially. the first byte is the upper byte
*
MLSR    C=C-1    X          have we had the whole 4K block
        JC+         CLUP    yes, clean up and exit
        R= 4                set loop counter to do 4 romwords
        A=0      X
        C<>B     X          save main loop counter in "B(X)"
*
* in this loop we make five bytes containing four rom words
*
D4RWRD  FETCH               get a rom word
        C=C+1     M
        A<>C     XS
        A<>C      X         lower part to "A(X)", high PART TO "C(XS)"
        LSHFA     X
        LSHFA     A
        LSHFA     A         right justify low bytes in "A(M)"
        C=C+C     X
        C=C+C     X
        RSHFC     X
        A<>C      X         left justify high bits in "A[1-0]"
        R=R-1
        ?R= 0               have we had 4 romwords
        JNC-        D4RWRD  no, do next word
*
        C<>B      X
        M=C                 save mainloopcounter and current address
        LSHFA     X
        LSHFA     A         right justify the 5 bytes in "A[11-2]"
        A<>C      A
        R= 5                set loop counter for 5 bytes to be saved
        WRIT Q              string bytes to reg Q
*
* it is neccessary to use register Q because we can't use "N" for this
* holds some essential file parameters
*
S5BYT   READ Q
        RCR 2
        WRIT Q              get one byte to "A[1-0]"
        ?NC XQ      7126    write it on tape
        ?NC XQ      77E7    test for transmit error and exit if error
        R=R-1
        ?R= 0               have we written 5 bytes
        JNC-        S5BYT   no, do next byte
        C=M                 get loop counter and current address
        JNC-        MLSR    do next words
*
* here the last record is closed and the cassette drive retrieves its
* normal status and we exit to mainframe
*
```

```
CLUP    LDI         $0A8
        NOP         (0C0)   this is done to solve a bug,  these 4
                            words could also be deleted
        NOP
        ?NC GO      70AF
*
*
*
* GETROM
*
        NAM         GETROM
*
        GOSUB       PILTST  error if there is no HPIL chip
        GOSUB       VPAG    test for valid page
        LDI         $007
        ?NC XQ      780A    error if this is not our file type
*
        ?NC XQ      7F77    seek the file
        ?NC XQ      70E6    send buffer 0 and SDA
        GOSUB       VPAG    find rampage to write at and set loop counter
*
* the mainloop of getrom starts here. we get 5 bytes out of the tape after
* each other and combine these into 4 romwords
*
MLGR    C=C-1   X           have we had the whole 4K
        ?C GO       70AC    yes, UNT and return to mainframe
        M=C                 save loop counter and current address
        R= 5                set counter for getting 5 bytes
*
* here we are getting 5 bytes out and save them in reg Q
*
G5BYT   ?NC XQ      7110    get a byte
        ?FSET 9             did we have any type of error
        ?C GO       7634    yes, decide which error and say it
        WRITHPIL            send byte out again
        A<>C    X
        READ Q
        A<>C    X
        RCR 2
        WRIT Q              append byte to string in reg Q
        R=R-1
        ?R= 0               have we done 5 bytes
        JNC-        G5BYT   no, do next byte
*
* after we have found 5 bytes, we have to store them in the desired page
*
        A=C     A           save the 4 lower bytes left justified in "A"
        RCR 2               get upper byte to "C[3-2]"
        C<>B    A
        C=M                 get current address and main loop counter
        R= 4                set counter for 4 romwords
        C<>B    A           initialize for loop to store 4 words
*
* the loop has to be entered with the 4 lower bytes left justified in "A"
* and the upper byte right justified in "C[3-2]"
*
```

```
D5BYT   A<>C     A
        RCR 12
        A<>C     A          shift next lower byte to "A[1-0]"
        A<>C    XS          append the upper bits to the word
        A<>C     X          word to "C(X)"
        C<>B     M          get destination address
        WRIT                word to page
        C=C+1    M          increment destination address
        C<>B     M          dest address to "B(M)"
        C=C+C    A
        C=C+C    A
        RCR 1               shift upper byte two bits to the right
        R=R-1
        ?R= 0               have we done 4 words
        JNC-        D5BYT   no, do next word
        C<>B     A          get loop counter and current address
        JNC-        MLGR    do next 4 words
*
*
*
* CAT and RAMWR routines
*
*
*
* APPend Hex Digit
*
* after a key has been pushed during partial key sequence, the main program
* may call this subroutine for processing the key. only hex digits are
* accepted ( i.e. digits 0-9 characters A-F ).
*
* Entry: data as produced by the HP-41 operating system after a key has
*        been pushed. the display should be enabled on entry
* Exit : if the key is digit 0-9 or chracter A-F the corresponding
*        character is appended to the display. the address after the call
*        is skipped. for all other keys 70 ms visual feedback. the next
*        instruction is not skipped
* Uses : "A(MS)", "A(X)", "C", "N" and the active pointer
*
APPHD   ?FSET 4             upper 10 keys
        JC+         APP1    yes
        ?FSET 3             is it a digit key
APP2    ?NC GO      BLINK   no, give 70 ms feedback
        A<>C    MS          recall numerical value of the key
        RCR 13
        R= 1
        LD@R 3              make it a display character
APP3    WRIT E              append it to the display
        POP
        C=C+1    M
        GOTOADR             skipp byte after the call
```

```
APP1    ?A#0    MS          is it the TAN key
        JNC-        APP2    yes, it is not allowed
        R= 13
        LD@R 7
        ?A<C    MS          is it G-J
        JNC-        APP2    yes, it is not allowed
        R= 2
        LD@R 5
        A=C     XS
        C=N
        C=C+A   XS          add 5 to keycode
        RCR 12
        R= 6
        LD@R 1
        LD@R 5              load address of character in defaultcode table
        FETCH               get characte
        R= 1
        C=0      R          convert it to display character
        JNC-        APP3    add character to display and exit
*
*
*
* NULL
*
* waits one second for key release. if key not released after this period
* "NULL" is displayed for at least ,3 seconds. when the key is released
* within 1 second, the byte after the subroutine call is skipped
*
* Entry: appropriate text in the display ( display is left justified by
*        NULL ). assumes display is enabled on entry.
* Exit : display is still enabled if the key is released in time, otherwise
*        chip 0 is enabled
* Uses : "A", "C", active pointer, sethex, flag 8
*
NULL    ?NC XQ      LEFTJ
        LDI         $2B3
        C=C+C   X           load timing constant for one second
NULL1   CLRKEY
        ?KEY                wait one second for key release
        JNC+        NULL3   key is released in time
        C=C-1   X
        JNC-        NULL1
        CLRF 8              say no blinking
        ?NC XQ      MSGA
        FCB         $03C    "null"
        LDI         $3E8    load timing constant for .3 seconds
NULL2   C=C-1   X
        JNC-        NULL2   show message for .3 seconds
        ?NC GO      RSTKB   wait for key release
NULL3   POP
        C=C+1   M
        GOTOADR             skip byte after the call
```

```
*
*
* Read HEX Digit 2
*
* reads specified number of hex digits from the display and returns these
* digits in "A(M)" right justified
*
* Entry: number of digits minus one is specified in "A(MS)" display must
*        be enabled on entry
* Exit : the wanted digits are returned in "A(M)" display is still
*        enabled
* Uses : "A", "C", "B(MS)", active pointer
*
* RHEXD1 reads 4 digits from the display
*
RHEXD1  R= 13
        LD@R 3
        A=C     MS              say we want four digits
RHEXD2  B=A     MS              save number of wanted digits in "B(MS)"
        A=0      M              initialize "A(M)" for holding digit string
        LDI         $009
        A=C      X              conversion constant for dig A-F to "A(X)"
RHEX 1  READ D                  move digits to be read to the left of display
        A=A-1   MS              have we had all digits
        JNC-        RHEX 1      no, do next
        A<>B    MS              recall counter
RHEX 2  R= 1
        READ E                  read one digit
        ?C#0     R              is it 9-0
        JC+         RHEX 3      yes
        C=C+A   R<              no, convert to A-F
RHEX 3  C=0      R
        RCR 11
        LSHFA    M              make place for new digit
        R= 3
        A=C      R              append digit
        A=A-1   MS              have we had all
        JNC-        RHEX 2      no, do next digit
        RTN
*
*
*
* CATalog
*
* ATTENTION !! this routine does not work in the expected way with the HP-
* 41 CX. this is caused by the fact, that the CAT routine from the 41CX is
* moved to page three with partial handling in the normal mainframe roms.
* however, this is not so important, because the 41CX already has a sort of
* port selectable catalog.
*
        NAM         CAT
*
        FCB         $000        say non programmable
        FCB         $000        say direct executing
*
        A=0      X
        ?NC XQ      CLLCDE
        ?NC XQ      PROMFC      display "CAT "
        JNC+        CATAL2
```

```
*
* when the user pressed backarrow we return here and cancel the function
* after we have done some housekeeping
*
CANCEL  ?NC XQ      CLLCDE
        ?NC XQ      ANNOUT
        ?NC XQ      RSTSEQ
        ?NC GO      NFRKB
*
* append a prompt sign and wait in light sleep for a key to be pressed
* if the key is a backarrow we return at byte after call, otherwise this
* byte is skipped
*
INVASK  ?NC XQ      NEXT1
        JNC-        CANCEL
        GOSUB       APPHD   append the hex digit to the display
        JNC-        INVASK  invalid digit, ask again
        A=0     MS          say we want to read one digit from display
        GOSUB       RHEXD2  get digit to "A(M)"
        GOSUB       NULL    wait for key release
        JNC-        CANCEL  if key hold down to long, cancel
        ?NC XQ      ENCP00
        ?NC XQ      RSTSEQ  clear user flags
        R= 3
        C=0     A
        A<>C    R           get CAT # to "C[3]"
        C=C+1   X
        RCR 3
        ST=C                save CAT # in "ST" ( needed by 0B84 )
        RCR 8
        A=C     M           address of second word at page to "A(M)"
        R= 6
        LD@R 5
        R= 6
        ?A<C    R           is it a regular CAT ( 1, 2 or 3 )
        ?C GO       0B84    yes, go do the CAT
        A=0     X
CAT 2   ?A#C    R           have we had all blocks before this page
        JNC+        CAT 1   yes
        FETCH
        A=A+C   X           add number of functions on page to total
        C=C+1   R           point to next page
        JNC-        CAT 2   do next page
CAT 1   A<>C    X
        RCR 4               number of functions to "C[12-10]" page address
        A=C     A           to "A(XS)" + "C(XS)" + functions to "A[12-10]"
        ST=0
        SETF 1              do rest of initialisation for 0B85
        ?NC GO      0B85    fall in the regular CAT 2
*
*
* DisPlay ADdRess
*
* this routine takes a hexadecimal digit from reg Q and writes it to the
* display ( right justified ). a space is appended.
*
* Entry: none
* Exit : display is enabled
* Uses : "A", "C", and active pointer
*
```

```
DPADR   ?NC XQ      ENCP00
        READ Q                  get the address to be put in the display
        R= 13
        LD@R 3                  load loop counter for 4 characters
        A=C     A
        ?NC XQ      CLLCDE
DPBYT   LDI         $009        load comparing constant
        A=C     X
        A<>C    M
        RCR 7                   digits to display to "C[13-10]"
*
* in the loop is tested if the character is A-F or a normal digit. in case
* of the A-F we only have to subtract 9 to get the LCD character. else we
* must add $30 to the digit to get the LCD character.
*
DPADR2  RCR 13                  get one digit to "C[1-0]"
        R= 1
        C=0     R
        ?A<C    R<              is it a A-F
        JNC+        5           no, make the right LCD character
        A<>C    R<
        A=A-C   R<              make the LCD character
        A<>C    R<
        JNC+        2
        LD@R 3
        WRIT C                  append character to the display
        A=A-1   MS              have we had all characters
        JNC-        DPADR2      no, do next one
        LDI         $020
        WRIT C                  append a space to the string
        RTN
*
*
*
* RAMWRite
*
        NAM         RAMWR
*
        FCB         $000        say nonprogrammable
*
ASKAD   ?NC XQ      CLLCDE
        ?NC XQ      MESSL
        RMB         $04         "ADR "
        JNC+        9
*
EXRMWR  ?NC XQ      CLLCDE
        ?NC XQ      ANNOUT      update the display to the current status
        ?NC XQ      RSTSEQ      clean up the partial key sequence
        ?NC GO      NFRKB       exit to mainframe
*
P4D     LDI         01F
        WRIT E                  append one prompt sign
        ?NC XQ      NEXT3       append 3 prompt signs and wait for key pressed
        JNC-        EXRMWR      it was backarrow, so exit
*
        GOSUB       APPHD
        JNC-        P4D         no hex digit, so try again
*
P3D     ?NC XQ      NEXT3       append 3 prompt signs and wait for key pressed
        JNC+        6           it was backarrow
*
```

```
        GOSUB       APPHD
        JNC-        P3D     no hex digit, so try again
*
        JNC+        3       it was a hex digit, so do next digit
*
* the backarrow key was hit, so we remove the rightmost character and try
* again
*
        READ D              remove digit
        JNC-        P4D     ask for four digits again
*
P2D     ?NC XQ      NEXT2   append 2 prompts and wait for key
        JNC+        6       process backarrow key
*
        GOSUB       APPHD
        JNC-        P2D     no hex digit, so try again
*
        JNC+        3       skip removing part
*
        READ D              remove one digit and prompt for 3 again
        JNC-        P3D
*
P1D     ?NC XQ      NEXT1   append 1 prompt and wait for key
        JNC+        6       process backarrow key
*
        GOSUB       APPHD
        JNC-        P1D     no hex digit, so try again
*
        JNC+        3       skip removing part
*
        READ D              remove one digit and prompt for 2 again
        JNC-        P2D
*
        GOSUB       RHEXD1  get address to "A(M)"
        GOSUB       NULL    wait for key release
ASKAD1  JNC-        ASKAD   if nulled, ask for address again
        ?NC XQ      ENCP00
ANDW    A<>C    M
        A=C     M           copy address to "C(M)"
        WRIT Q              save address in reg Q
        R= 5
        C=0     R<          create address of first word
        ?A#C     M          are we at first word
        JNC+        NFWRD   yes, continue
        FETCH
        ?C#0     X          is xrom zero
        JC+         NFWRD   no, continue
*
* when we have a first word that is zero, we must assume that there is no
* rampage. then we say "NO ROM" and go asking for the address again.
*
```

```
         ?NC XQ      CLLCDE
         ?NC XQ      MESSL
         RMB         $06       "NO ROM"
ADRAG    ?NC XQ      LEFTJ
         ?NC XQ      BLINK
         LDI         $3E8
         C=C-1    X            leave message .3 sec in display
         JNC-        1
         JNC-        ASKAD1
*
NFWRD    GOSUB       DPADR     put the 4 hex address digits in the display
         ?NC XQ      ENCP00
         ?NC XQ      OFSHFT
         READ Q
         FETCH                 get data at this word
         RCR 10
         R= 13
         LD@R 2                say we want to append 3 digits
         A=C      A
         ?NC XQ      ENLCD
         GOSUB       DPBYT     PRINTDISPLAY
*
* the display now contains the address and the data at this address. now
* we are waiting for the user to press a key.
*
WFKEY    ?NC XQ      NEXT      wait for key
         JNC-        ASKAD1    if backarrow, ask for the address again
         JNC+        2
*
* stepping stone
*
STP1     JNC-        NFWRD
*
         C=N
         RCR 1
         C=0      XS
         A=C       X           keycode to "A(X)"
*
* in this part we check for some special keys. these are SHIFT, BST, STO,
* SST and TAN. the keycodes that are returned are the synthetic keycodes
* minus one
*
         LDI         $002
         ?A#C     X            is it shift
         JNC+        TSHFT     yes, go set shift and wait for next key
         LDI         $00A
         ?A#C     X            is it shift/shift
         JC+         TSTK      no, test for other keys
TSHFT    ?NC XQ      TOGSHF
         ?NC XQ      ANNOUT
         ?NC XQ      ENLCD
         JNC-        WFKEY
*
TSTK     LDI         $022
         ?A#C     X            is it STO
         JC+         4
         GOTO        RMWSTO    yes, go display contents of rom
*
```

```
         LDI          $041
         ?A#C     X           is it the TAN key
         JC+          4
DOBSTP   GOTO         RWRBST   yes, do a backstep
*
         C=C+1    X
         ?A#C     X           is it the SST key
         JC+          4
         GOTO         RMWSST   yes, do a single step
*
         LDI          $04A
         ?A#C     X           is it shift SST
         JNC-         DOBSTP   yes, go do a backstep
*
         ?FSET 3              is it a digit key
         JC+          4       yes
BDD+A    ?NC XQ       BLINK   no
STP2     JNC-         STP1    display address and data again
*
* if we have to do with a digit key, we remove the data digits
*
         READ D
         READ D
         READ D
         R= 13
         LD@R 4
         ?A<C     MS          is it digit 0-3
         JNC-         BDD+A   no, go display address and data again
         GOSUB        APPHD
         JNC-         STP2    no hex digit, go display address and data again
*
* if key is accepted we return here
*
DP2D     ?NC XQ       NEXT2   add 2 prompts and wait for key
         JNC+         6       process backarrow key
*
         GOSUB        APPHD
         JNC-         DP2D    no hex digit, try with 2 prompts again
*
         JNC+         DP1D    skip backarrow processing
*
         READ D               remove data digit
         JNC-         STP2    display address and data again
*
DP1D     ?NC XQ       NEXT1   add 1 prompt and wait for key
         JNC+         6       process backarrow key
*
         GOSUB        APPHD
         JNC-         DP1D    no hex digit, try with 1 prompt again
*
         JNC+         3       skipp backarrow processing
*
         READ D               remove last digit
         JNC-         DP2D    try with 2 prompts again
*
```

```
         R= 13
         LD@R 2
         A=C     MS
         GOSUB       RHEXD2
         GOSUB       NULL
         JNC-        STP2     key hold to long, display address + data again
         ?NC XQ      ENCPOO
         A<>C    A
         RCR 3
         A=C     X            save input data in "A(X)"
         READ Q               get address to write to
         FETCH
         A<>C    X
         A<>B    X            save original data word in "B(X)"
         A=C     X            copy input data to "A(X)"
         WRIT
         FETCH
         ?A#C    X            is it written
         JC+         NWBRAM   no, test for no write or bad mldl
DONDW1   READ Q               yes
         C=C+1   M            point to next address
DONDW    A=C     M            save address in "A(M)"
         GOTO        ANDW
*
NWBRAM   A<>B    X
         ?A#C    X            is the original data still the same
         JC+         BMLDL    no, go say bad mldl
         ?NC XQ      CLLCDE
         ?NC XQ      MESSL
         RMB         $08      "NO WRITE"
*
* put message in the display, wait for .3 seconds and ask for the address
* again
*
         GOTO        ADRAG
*
RMWBST   ?NC XQ      ENCPOO
         ?NC XQ      OFSHFT   clear shift annunciator
         READ Q
         C=C-1   M            decrement address
         JNC-        DONDW    go display address + data and wait for new data
*
* in case we are in rom 0 at word 0 and a BST is wanted, we would wrap to
* address FFFF. to avoid this we do a single step to warn the user
*
RMWSST   ?NC XQ      ENCPOO
         JNC-        DONDW1
*
* put message in display and reset partial key sequence and exit to the
* mainframe
*
BMLDL    ?NC XQ      CLLCDE
         ?NC XQ      MESSL
         RMB         $08      "BAD MLDL"
         ?NC XQ      LEFTJ
         SETF 8
         ?NC XQ      MSG105
         ?NC XQ      RSTSEQ
         ?NC XQ      STMSGF
         ?NC GO      NFRKB
```

```
*
*
*
RMWSTO  GOSUB       DPADR
        ?NC XQ      ENCP00
        READ Q
        FETCH                   get data at current address
        C=C+1   M
        WRIT Q                  save updated address
        RCR 10
        R= 13
        LD@R 2
        A=C     A
        ?NC XQ      ENLCD
        GOSUB       DPBYT       append data at this address to the display
        ?NC XQ      LEFTJ       view data PRINTDISPLAY
        LDI         $250        load timing constant
        CLRKEY
WAIKEY  ?KEY
        JNC+        NOKEY
        C=KEY                   there is a key pressed
        RCR 3
        A=C     X
        A=0    XS               keycode to "A(X)"
        R= 0
        C=C+C   R               is it the ON key
        ?C GO       OFF         yes, go to deepsleep
        LDI         $087
        ?A#C    X               is it the R/S key
        JNC+        DOR/S       yes, go process it
        RCR 11                  no, get timing constant back
NOKEY   C=C-1   X               is time expired
        JNC-        WAIKEY      no, keep waiting
        JNC-        RMWSTO      yes, display next byte
*
DOR/S   ?NC XQ      RSTKB       wait until key is released
        LDI         3E8
        C=C+C   X
        C=C-1   X
        JNC-        1           wait a while
        GOTO        RMWBST      do a bst and display address and data again
*
*
*
* Hex TO Decimal ( floating point )
*
* Entry : "A(X)" holds hex number ( smaller then 999 )
* Exit  : "A" , "C" holding floating point number
* Uses  : "A" , "B(X)" , "C"
*
```

```
HTOD    A=0      M
        A=0     MS          initialize "A[13-3]"
        LDI         $064
        C<>B     X          100d in "B(X)"
        C=0      A
D 100   ?A<B     X
        JC+         DO 10
        A=A-B    X          subtract 100d
        C=C+1    X          count the Hundreds in "A(X)"
        JNC-        D 100
DO 10   C<>B     A
        LDI         $00A
        C<>B     A          10d in "B(X)"
        RCR 13
D 10    ?A<B     X
        JC+         NORM
        A=A-B    X          subtract 10d
        C=C+1    X          count the Tens in "A(X)"
        JNC-        D 10
*
* normalize the number by setting up an exponent of 2 and shifting the
* mantissa to the left left one place and decrementing the exponent as long
* as the mantissa isn't in place
*
NORM    RCR 13              Units are left in "A(X)"
        A=A+C    X          add H,T and U to a dec. number
        LDI         $002    normalize number
        A<>C     A
        ?C#0    XS
        JC+         DONEHD
        RCR 13
        A=A-1    X
        ?C#0    XS
        JC+         DONEHD
        RCR 13
        A=A-1       X
*
* make the floating point number by combining the exponent and the mantissa
*
DONEHD  RCR 4               put mantissa in place
        C=C+A    X          add the exponent
        A=C      A          save floating point in "A" and "C"
        RTN
*
*
*
* COD
*
* Entry: alpha has the hex string that has to be converted
* Exit : NNN is in "B" and "C"
* Uses : "A", "B", "C"
*
```

```
COD     B=0      A          Initialize "B" for the new hex number
        SETF 9              Say we do the loop for the first time
        SLCT Q              Select the loop counter
        READ M              get first 7 digits to convert
        JNC+        3       start converting loop
DO N    READ N              get the next 7 digits
        CLRF 9              say we are finished after this loop
        A=C      A          save digits in "A"
        R= 7                set the loop counter for 7 digits
*
* converting the ascii is done by testing if it lays in between $30 and
* $39 and if so, subtracting $30. if it isn't a number we test for ascii
* between $41 and $46 and if so subtract $37
*
COD 1   SLCT P
        R= 1                say we only compare the right digit
        ?A#0    R<          add a zero to the NNN in "B"
        JNC+        DO-9
        LDI         $030    load test constant for ascii < $30
        A=A-C   R<
        JC+         DATERR  we do have ascii < $30
        LDI         $00A
        ?A<C    R<          test if this is ascii for a digit 0-9
        JC+         DO-9    we do have a digit 0-9
        LDI         $011
        ?A<C    R<          test if the ascii code is less then $41
        JC+         DATERR
        LDI         $017    test for ascii codes > $46
        ?A<C    R<
        JNC+        DATERR
        LDI         $007
        A=A-C   R<          make the hex value for a digit A-F
DO-9    R= 0
        B=A      R          add this digit to the NNN in "B"
        C<>B     A
        RCR 1
        C<>B     A          shift NNN one place left
        RSHFA    A
        RSHFA    A          get next digit to "A[1-0]"
        SLCT Q
        R=R-1
        ?R= 0               have we done 7 digits
        JNC-        COD 1   no
        ?FSET 9             do we have to do 7 more digits
        JC-         DO N    yes
CODEND  C=B      A          copy the NNN to "C"
        RTN
DATERR  ?NC XQ      ERROR
        FCB         $022    specify message 'DATA ERROR'
*
*
*
* Decode subroutine
*
* Entry: "B" holds NNN to decode
*        flag 8 must be set and 9 must be cleared
* Exit:  Alpha register holds the decoded NNN
*
```

```
DECOD   SLCT Q
        JNC+        3
DECOD2  WRIT N              save the last 7 digits
        CLRF 8              say we do the loop for the last time
        R= 7                set loop counter to 7
*
* decoding a hex digit to it's ascii representation is done by adding $30
* if the hex number is smaller then $0A and adding $37 if it is greater
* then $09
*
DECOD1  LSHFA    A
        LSHFA    A          make room for the next ascii coded digit
        C<>B     A
        RCR 13
        C<>B     A          get next digit to transform
        SLCT P
        R= 0                point to digit to transform
        A<>B     R          digit to "A[0]"
        ?A#0     R          if it is a leading zero, it must be suppressed
        JNC+        2
        SETF 9              say we have had all leading zero's
        ?FSET 9             do we have a leading zero
        JNC+        DIG=0   yes
        R= 1
        LDI         $00A
        ?A<C    R<          is it a number
        JC+         4       yes
        LDI         $007
        A=A+C   R<          make it a digit A-F minus $30
        LDI         $030
        A=A+C   R<          make it a ascii coded digit
 DIG=0  A<>C     A
        A=C      A          copy ascii string to "C"
        SLCT Q
        R=R-1
        ?R= 0               are we finished yet
        JNC-        DECOD1  no, loop again
        ?FSET 8             have we done the loop twice
        JC-         DECOD2  no, save 7 digits in N and do the rest
        ?FSET 9             was the NNN zero
        JC+         3       no
        LDI         $030    yes, load only one zero
        WRIT M              save last 7 digits in M
        ?NC GO      XAVIEW  display and return to mainframe
*
*
* MOVE
*
        NAM     MOVE
*
        GOSUB       COD     get the desired addresses to "C"
        A=0      A
        B=0      A          make sure we only leave the wanted addresses
        SLCT Q
        R= 6
        SLCT P
        R= 3                devine the address field with P and Q
```

```
*
* split the begin, end, and destination addresses
*
        RCR 1
        A<>C    PQ          EEEE to "A"
        RCR 4
        C<>B    PQ          BBBB to "B"
        RCR 6
        C<>B    PQ          DDDD to "B" BBBB to "C"
        ?A<B     M          test if EEEE<DDDD
        JC+         MOVE2
        A<>B     M          DDDD to "A" EEEE to "B"
        ?A<C     M          test if DDDD<BBBB
        JC+         MOVE3
*
* if the destination address lays in between the begin and end address of
* the block we want to move, we have to start copying at the begin of the
* source block and place it at the begin of the destination block because
* we overwrite the source block
*
        M=C                 save BBBB in "M"
        A<>B     M
        C=A-C    M
        A<>B     M          number of bytes to copy to "C"
        A=A+C    M          end of destination block to "A"
        C=M                 BBBB to "C"
        A<>B     M          end of dest to "B"
        A<>C     M          EEEE to "C"  BBBB to "A"
        C=C+1    M          this is necessary to get all bytes copied
MOVE1   C=C-1    M          decrement source address
        FETCH               get source byte
        C<>B     M          get destination address
        WRIT                write byte to destination
        C=C-1    M          decrement destination address
        C<>B     M          get source address back to "C"
        ?A<C     M          are we finished yet
        JC-         MOVE1   no, loop again
        RTN
*
* when the destination block is not overwriting the source block we just
* copy from begin to end
*
MOVE3   A<>B     M          EEEE to "A"  DDDD to "B"
MOVE2   FETCH               get source byte
        C=C+1    M          increment source address
        C<>B     M          get destination address
        WRIT                write byte to destination
        C=C+1    M          increment destination address
        C<>B     M          get source address back to "C"
        ?A<C     M          are we finished yet
        JNC-        MOVE2   no, loop again
        RTN
*
*
*
* Last ROM word
*
        NAM         LROM
*
```

```
         GOSUB        COD      get start address to "C[3-0]"
         A=C      M            get page digit to "A(M)"
         LSHFA    M
         LSHFA    M
         LSHFA    M            page digit to "A[6]"
         RCR 11                start address to "C[6-3]"
LROM1    C=C-1    M            point to next word to examine
         FETCH
         ?C#0     X            is it a zero byte
         JC+          LROM2    no, we have the first non zero word
         ?A<C     M            have we searched the whole rom yet
         JC-          LROM1    no, loop again
NONE     ?NC XQ       ERROR
         FCB          $031     specify message 'NONE'
LROM2    A=0      A            initialise for DECOD and put address and byte
         B=0      A            in alpha
         C<>B     A
         SETF 8
         CLRF 9
         WRIT 0
         WRIT P
         GOTO         DECOD
*
*
*
* LOCate by Alpha
*
         NAM          LOCA
*
         GOSUB        COD      get start address and data word
         R= 6
         A=0      A
         A=A-1    R<           create FFF in "A[5-3]"
         A=C      R            create the end address of this page in "A[6-3]
         A=C      X            save the target word in "A(X)"
LOCA     C=C+1    M            point to next word to test
         FETCH
         ?A#C     X            is it the desired data word
DECODE   JNC-         LROM2    yes, send it to alpha and the display
         ?A<C     M            have we reached the end of the rom yet
         JNC-         LOCA     no, loop again
*
* set user flag 10 when we do not find the target word
*
         READ D                get user flags
         RCR 11
         C<>ST                 user flag 10 to CPU flag 1
         SETF 1                set user flag 10
         C<>ST
         RCR 3
         WRIT D                restore the updated user flags
         JNC-         NONE     and say 'NONE'
*
*
*
* CODe
*
         NAM          COD
*
```

```
        GOSUB       COD     get the NNN to "C" and "B"
        ?NC GO      RCL     put the NNN on the user stack and take care of
*                           a stacklift
*
*
*
* DECODe
*
        NAM         DECOD
*
        READ X
        JNC-        DECODE
*
*
*
* CLear BLock
*
        NAM         CLBL
*
        GOSUB       COD     get the desired page or block to "C"
        RCR 12
        ?C#0     M          do we have to do the entire page
        JC+         CLBL2   no, only a little block
        RCR 10              get page address to "C[6]"
        R= 5
        A=C      A          page to "A[6]" and clear the rest of "A"
        A=A-1    R<         create end address of this page
        A<>C     M          EEEE to "C"  BBBB to "A"
CLBL1   WRIT                clear this word
        C=C-1    M          point to next word
        ?A<C     M          are we finished yet
        JC-         CLBL1   no, loop again
        WRIT                clear the first word of the block
        RTN
CLBL2   RCR 3               BBBB to "A(M)"
        A=0      M
        A<>C     M
        RCR 10              EEEE to "C(M)"
        JNC-        CLBL1   clear the block
*
*
*
* Count ByTes
*
        NAM      10 CBT     indicate an alpha prompt
*
        FCB         $000    indicate not programmable
*
        GOSUB       FPAL
        SETHEX
        RCR 8
        A=C      A          address of end to "A[3-0]"
        CLRF 10             say we are in ram
        N=C                 save end address in "N"
        ?NC XQ      CPGMHD  get start address to "A[3-0]"
        ?NC XQ      PUTPCF  place PC to start of program
        C=N
        ?NC XQ      CALDSP  # bytes*2 in "A[3]" # regs in "A(X)"
        A<>C     R<
```

```
        A=C     R<
        C=C+C    X
        C=C+C    X
        C=C+C    X
        A<>C     X
        A=A-C    X          compute registers * 7
        RCR 1
        C=0      M
        RCR 2
        C=C+C    X
        C=C+C    X
        C=C+C    X
        RCR 1               right justify number of bytes
        C=C+A    X
        C=C+1    X
        C=C+1    X          total program length to "C(X)"
        A=0      A
        A=C      X
        ?NC XQ      CLLCDE
        ?NC XQ      GENNUM  get number of bytes to display
        LDI         $020
        WRIT E              append a space
        ?NC XQ      MESSL
        RMB         $06     "BYTES_"
        ?NC XQ      LEFTJ
        LDI         $020
        WRIT D
        ?NC XQ      ENCPOO
        ?NC XQ      STMSGF
        ?NC GO      NFRKB
*
*
*
* Delete FAT entry
*
        NAM         DFAT
*
        GOSUB       COD     get entry address to "C[4-0]"
        R= 1                set pointer for comparing digit 0 and 1
        C<>B     X          save AAA in "B(X)"
        C=0      X
        RCR 11              first address of this page to "C[6-3]"
        A=C      A          save also in "A"
        C=C+1    M
        FETCH               get number of functions on this page
        C=C+C    X          double it to make number of bytes in FAT
        A<>C     X
        RCR 3
        C=C+A    X          create address of last FAT word
        RCR 11
        A<>C     M          end address to "A(M)"
        C<>B     X          get AAA to "C(X)"
        A=0      X
        A<>C    XS
        RSHFA    X
        RSHFA    X          first digit of AAA in "A[0]"
        RCR 6               get 0 to "C[1]"
        A=C R               complete first word of FAT in A"(X)"
        RCR 8
        C<>B     X          second word of FAT in B"(X)"
```

```
*
* in this loop we find the address of the first entry word of the entry
* to delete
*
DFAT1   C=C+1    M          point to word before an entry
        A<>B     X          these two instructions are necessary to assure
DFAT2   A<>B     X          proper operation in case we compare both words
        C=C+1    M          point to first FAT word
        ?A<C     M          have we had all entry's
        JC+         NO ENT  yes, say there is not such an entry
        FETCH               get first FAT word
        ?A#C     R<         is it the same as our first FAT word
        JC-         DFAT1   no, do the next entry
        A<>B     X
        C=C+1    M
        FETCH               get second FAT word
        ?A#C     R<         is it the same as our second FAT word
        JC-         DFAT2   no, restore our FAT words and do the next entry
*
* after we have found the FAT entry to delete, we have to move the whole
* FAT down by two words and make sure that the last entry is replaced by
* the two neccessary null words to say that this is the end of the FAT
*
        A=A+1    M
        A=A+1    M          point to the last null word after the FAT
        C=C-1    M          point to the first word of the undesired entry
        C<>B     M
        C=B      M          save destination address in "B(M)"
        C=C+1    M
DFAT3   C=C+1    M          increment source address
        FETCH               get source word
        C<>B     M
        WRIT                write it to destination address
        C=C+1    M          increment destination address
        C<>B     M          destination to "B(M)"  source to "C(M)"
        ?A<C     M          are we finished yet
        JNC-        DFAT3   no, do next FAT word
        R= 5
        LD@R 0
        LD@R 0
        LD@R 1
        FETCH               get number of functions on this page
        C=C-1    X          decrement number of functions
        WRIT                save new number of functions
        RTN
NO ENT  GOSUB       INERR
        RMB         $08     "NO ENTRY"
DFERRE  GOTO        ERREX
*
*
*
* Append entry to FAT
*
*
```

```
* AFAT1
*
* Entry: "A" has the entry as follows in "C[5-0]" UOPAAA
*        "B(X)" has the digits AAA
* Exit : the entry is appended when there is enough room in the FAT, other
*        wise the message ENTRY>64 is displayed
* Uses : "A", "B(X)", "C"
*
        NAM         AFAT
*
        GOSUB       COD     get UOPAAA to "C" and "B"
        A<>C    A
AFAT1   A<>C    A
        A=C     M           save user code and offset in "A[5-4]"
        C=0     X
        RCR 11              address of first word to "C[6-3]"
        C=C+1   M
        FETCH               get number of functions
        C=C+1   X           add one function
        A<>C    X
        LDI         $040    this is max number of functions on one page
        A<>C    X
        ?A<C    X           can we append another function
        JC+         E>64    no, say 'ENTRY>64'
        WRIT                increment the number of functions with one
        C=C+C   X           this is address of first word of new function
        A<>C    X
        RCR 3
        A<>C    X
        RCR 11              "C[6-3]" has first word address
        C=B     X           get AAA to "C(X)"
        RSHFA   A           offset and user to "A[4-3]"
        A=0     X
        A<>C   XS           first digit AAA to "A[2]" second word to "C(X)"
        RSHFA   A
        RSHFA   A           first entry word in "A(X)"
        A<>C    X
        WRIT                append first entry word to FAT
        C=C+1   M
        A<>C    X
        WRIT                append second entry word to FAT
        C=0     X           make null word after FAT in "C(X)"
        C=C+1   M
        WRIT                first null word after FAT in RAM
        C=C+1   M
        WRIT                second null word
        RTN
E>64    GOSUB       INERR
        RMB         $08     "ENTRY>64"
        JNC-        DFERRE
*
*
*
* --
*
        NAM         --
*
        RTN
*
```

```
*
*
        ORG         $7FF
*
*
*
* MNEM
*
        NAM         MNEM
*
        C=0     A
        RAMSLCT             select status registers
        PRPHSLCT            and deselect all peripherals
        READ L              get an instruction byte to "C[2-0]"
*
* when this is the first byte of an instruction there will happen nothing
* assumed is that reg L is empty when we start. when it is the second byte
* of an instruction the first byte is found in "C[2-0]". it is used to find
* out to which class the instruction belonged and what to do now
*
        C<>ST               instr. class of first byte to flag 1 and 0
        READ Y              read first byte of instruction
*
* here is tested if we have to do with a class 1 instruction or if we
* handle a LDI instruction
*
        ?FSET 0             is it a class 1instruction
        JC+         STP  1  yes, handle this instruction type
        ?FSET 1             is it a LDI
        JC+         STP  2  it is a LDI, go handle it
*
* here is the class tested in case we do the first byte of an instruction
*
        ST=C                get class to flag 1 and 0
        ?FSET 0             is it class 1 or 3
        JC+         STP  3  yes, go handle class 1 or 3
        ?FSET 1             is it class 0 or 2
        JNC+        STP  4  go handle class 0
*
* this part takes care of the arithmetic instructions
*
        A=C     A
        B=A     A           save instruction code and it's address in "B"
        C=C+C   A
        C=C+C   A
        RSHFC   X           strip of the class bits
        C=C+C   A
        RSHFC   X           strip off the field bits and right justify
        A=C     A           save number of instruction in "A(X)"
        ?NC XQ      PCTOC   get current page address
        R= 5
        LD@R A
        LD@R 0
        LD@R 0              load start-1 of instruction table
*
* all the instructions in class two are numbered in sequence, determined
* by the first 5 bits of the instruction code. we pick the right point in
* the table by decrementing the number evrey time we have had an ascii
* string representing one instruction and exit when we have reached the
* right string
*
```

```
        R= 1                    set pointer for instruction counter
        CLRF 9                  say we only are doing the instruction type
CLS 23  A=A-1    R<             test if we have reached our string number
        JC+          CLS 20     we have found our string
        C=C+1     M
        FETCH                   get next ascii character of a string
        C=0      R<
        C=C-1     X             is this the end of a string
        JC-          4          no, pick next character of string
        JNC-         CLS 23     yes, go test if we are ready yet
*
* stepping stones
*
STP  4  JNC+         STP  5
STP  3  JNC+         STP  6
STP  2  JNC+         DOLDI
STP  1  JNC+         STP  7
*
*
* we enter the fetch of the string we want  to have in "A" with the address
* of the first character-1 in "C(M)"
*
CLS 20  A=0       A             initialize "A" for holding a string
        R= 1                    look only for the ascii byte
CLS 21  C=C+1     M
        FETCH                   get next character
        ST=C                    save it in "ST"
        LSHFA     A
        LSHFA     A
        A<>C     R<             add character to the string
        C=C-1     X             have we done all characters
        JC-          CLS 21     no, do next character
        A=A+1    MS             type string as alpha data
        A<>C      A             and put string in "C"
        ?FSET 9                 have we had the 'field' string yet
        JC+          CLS 22     yes, write field string to stack and exit
        SETF 9                  no, say we do the field string
        M=C                     save instruction string in "M"
        C=B       A             get instruction code back
        C=C+C     X
        C=C+C     X
        RSHFC     X             strip off the class bytes
        C<>ST
        CLRF 3
        C<>ST                   clear last bit of instruction type
        A=C       X             save field byte in "A(X)"
        ?NC XQ       PCTOC
        R= 5
        LD@R A
        LD@R 8
        LD@R A                  load start address of field table
PARAM   R= 1
        A=0       R             leave the field byte only in "A[0]"
        JNC-         CLS 23     and make the field string in "A"
CLS 22  WRIT T                  write field string to reg T
        C=M
        A=C       A             instruction string to "A"
        LDI          $020       load ascii byte for a space
```

```
CLS 24  R= 11
        ?A#0     R          do we have a character in byte 6
        JC+         CLS 25  yes, then we are finished
        LSHFA    R<
        LSHFA    R<         no, make place for a space at the right end
        R= 1
        A=C      R<         add space to string
        JNC-        CLS 24  and test if we are done
CLS 25  A<>C      A
        WRIT Z               save field string in reg Z
        C=0      A
        WRIT L               say we have first byte of instruction
        RTN
*
* stepping stones
*
STP  5  JNC+        STP  8
STP  6  JNC+        STP  9
STP  7  JNC+        DOCLS1
*
*
* this part converts the hex constant of the LDI instruction to its decimal
* value in "A[3-1]"
*
DOLDI   R= 2                 set loop counter for conversion
        A=0      A           initialize "A" for conversion
LDI2    C=C-1    R           have we done this digit
        JC+         LDI1     yes, test if we are ready
        SETDEC
        A=A+1    A           add converted hex value to "A"
HMNDIG  SETHEX
        JNC-        LDI2
LDI1    ?R= 0                are we finished yet
        JC+         CONRDY
        R=R-1                no, decrement loop counter and multiply by 16
        C<>B     A           save hex number in "B(X)"
*
* the conversion is done by counting how many times we have a factor of 16
* in the hex number. this is done in decimal. after we have found this
* number, we multiply it by 16 and add the rest of the hex number to the
* decimal number. this leaves the converted result in "A"
*
        A<>C     A
        A=C      A           copy decimal string to "C
        LSHFA    A           multiply "A" by 10
        SETDEC
        C=C+C    A
        A=A+C    A
        A=A+C    A
        A=A+C    A           "A" is multiplied by 16
        C<>B     A           get hex number back
        JNC-        HMNDIG   set mode to hex and do next digit
CONRDY  A<>C     A
*
* at this point "C[3-0]" contains the converted hex string right justified
* we only have to make it an ascii coded decimal string. this is done by
* inserting the digit "3" before every decimal digit
*
```

```
        RCR 11                  initialize string for ascii conversion
        R= 3                    set loop counter
LDI3    RSHFC   R<
        LD@R 3                  convert one digit to ascii
        R=R+1
        R=R+1                   increment loop counter
        ?R= 6                   are we done yet
        JNC-        LDI3        no, loop again
        R= 7
        LD@R 3                  do the last character
        C=C+1   MS              type string as alpha data
        WRIT T                  and put it in reg T
        C=0      M
        R= 11
        LD@R 4
        LD@R 3
        LD@R 4
        LD@R F
        LD@R 4
        LD@R E
        LD@R 2
        LD@R 0
        LD@R 2
        LDI         $020        load the 'CON___' string
        WRIT Z                  and save it in reg Z
        C=0      A
        WRIT L                  say we are doing the first byte
        RTN
*
* stepping stones
*
STP  8  JNC+        STP 10
STP  9  JNC+        STP 11
*
*
* here is the class 1 instruction type field handled and the jump
* address is done
*
DOCLS1  ST=C                    get the second byte into "ST" for testing type
        A=C      A              save first byte in "A(X)"
        R= 13
        LD@R 1
        LD@R 0                  load alpha identifier
        LD@R 4
        LD@R 7
        LD@R 4
        LD@R F                  first two letters are always 'GO'
        ?FSET 1                 is it a jump or an xeq
        JNC+        DOXEQ       it is an xeq
        LD@R 4
        LD@R C                  load a 'L' for the 'GOL'
        ?FSET 0                 is it a jump on carry
        JC+         GOLC        yes, handle this one
        LD@R 4
        LD@R F
        LD@R 4
        LD@R E
        LD@R 4
        LD@R 7                  finish the instruction to 'GOLONG'
        JNC+        JMPTOZ
```

```
DOXEQ   LD@R 5
        LD@R 3                  make the string 'GOS'
        ?FSET 0                 is it an xeq on carry
        JNC+        GOSUB       no, handle this one
GOLC    LD@R 4
        LD@R 3
        LD@R 2
        LD@R 0                  make instruction 'GOSC_' or 'GOLC_'
        JNC+        ADDSP       add a space for 6 characters total
        LD@R 5
        LD@R 5
        LD@R 4
        LD@R 2                  make string 'GOSUB'
        LD@R 2
ADDSP   LD@R 0                  add a space to the string to get 6 characters
JMPTOZ  WRIT Z                  save string in reg Z
        A<>C    X               get the two msb digits of address to "C[2-0]"
        C=C+C   A
        C=C+C   A
        RCR 12                  shift them into "C[2-1]"
        A<>C    A               save them in "A(M)"
        READ L                  get the lsb digits
        A<>C    X
        A<>C    A
        C=C+C   X
        C=C+C   X               left justify them in "C(X)"
        RSHFC   A               right justify address in "C"
        CLRF 9                  say we are doing a class 1 type instruction
        JNC+        DIST        skip the stepping stones
*
* stepping stones
*
STP 10  JNC+        STP 12
STP 11  JNC+        CLS1FB
*
*
* in this part we handle the addresses of the jump and gosub instructions
* and we handle the distance part of the short jumps. when we have a short
* jump we also add a asterisk and the direction
* first thing we have to do for this is to translate the hex representation
* of the address or the distance to the ascii coded hex form
*
DIST    R= 3                    set loop counter to the starting position
        A<>C    R<
        C=0      A
        A<>C    R<
        RCR 11
        A<>C     A
        C=0      A
        SETDEC
        C=-C-1   A
        SETHEX
        A<>C     A              get all 9's to "A" and address to "C"
NDIG    RSHFC   R<              shift digit one right
        LD@R 3                  and make it ascii '0'-'9'
        ?A<C     R              was it a number
        JNC+        NUMBER      yes, then we are ready
        C=A-C    R
        C=-C     R              no, convert A-F to 1-6
        R=R+1
        LD@R 4                  and make it ascii 'A'-'F'
```

```
NUMBER  R=R+1
        R=R+1                   increment loop counter by one
        ?R= 6                   are we finished yet
        JNC-          NDIG      no, do next digit
        R= 7
        LD@R 3
        ?A<C      R
        JNC+          NUMB
        C=A-C     R
        C=-C      R
        R= 7
        LD@R 4
        ?FSET 9                 is it the address of a xeq or jump
        JNC+          DONE      yes, exit
        R= 7                    no, we have to add an asterisk and direction
        LD@R 2
        LD@R A                  load the asterisk
        LD@R 2
        LD@R B                  load direction forward
        ?FSET 8                 is it backwards
        JC+           FWRD      no, do forward
        R= 4
        LD@R D                  load direction backwards '-'
FWRD    C=C+1    MS             type string as alpha data
        WRIT T                  save it in reg T
        C=0       A
        WRIT L                  say we are doing the first byte
        RTN
*
* stepping stones
*
STP 12  JNC+          CLS 0
DIST1   JNC-          DIST
*
*
* in this part we handle the class 1 instructions in case we have them for
* the first byte
* also are the class three instructions handled here
*
CLS1FB  ?FSET 1                 is it class 1 or 3
        JNC+          BYTE11    we have class 1
        SETF 9                  say we do class 3 distance for converting part
        A<>C       A            save instruction code in "A(X)"
        R= 13
        LD@R 1
        LD@R 0                  load alpha identifier
        LD@R 4
        LD@R 7
        LD@R 4
        LD@R F                  make string 'GO'
        ?FSET 2                 is it a gonc
        JC+           GOC       no, do goc
        LD@R 4
        LD@R E
        LD@R 4
        LD@R 3                  make string 'GONC'
        JNC+          ADDSPA
GOC     LD@R 4
        LD@R 3
        LD@R 2
        LD@R 0                  make string 'GOC_'
```

```
ADDSPA  LD@R 2
        LDI          $020     add two spaces to string
        WRIT Z                save string in reg Z
        A<>C     A            get instruction code back
        SETF 8                say we do a forward jump
        C=C+C    A
        RSHFC    X            strip of class and c/nc bits
        ST=C                  direction bit to "ST 6"
        ?FSET 6               is it a forward jump
        JNC-         DIST1    yes, convert distance and say forward
*
* when we have a jump backwards we have to convert the coding, because the
* distance is saved in the inverted format
*
        SETF 7                make sure that the first bit becomes zero
        C=ST
        C=-C     X            invert the bit pattern
        CLRF 8                say we jump backwards
        JNC-         DIST1    convert distance and say backward
*
* here is the handling of class 1 done in case we have to do the first byte
* of the instruction
*
BYTE11  WRIT L                save instruction code in reg L for second pass
        C=0      A
        C=C+1    MS
        WRIT Z
        WRIT T                write a zero string to reg Z and T
        RTN
*
* here the handling of the class 0 instructions is done
*
CLS 0   C=C+C    X
        C=C+C    X
        RSHFC    X            strip off the class bits and right justify
        ST=C                  instr. to "ST" for testing
        A=C      X            and save in "A(X)"
        CLRF 8
        CLRF 9                say we are handling row 6
        LDI          $006
        R= 0
        ?A#C     R            is it row 6
        JNC+         DOROW6   yes, go handle it
        LDI          $008
        ?A#C     R            is it row 8
        JNC+         DOROW8   yes, go handle it
        LDI          $00C
        ?A#C     R            is it row C
        JC+          STP14    no, we have not to do with row 6,8 or C
        SETF 9                say we do row C
        JNC+         DOROW6
DOROW8  SETF 8                say we do row 8
DOROW6  A<>B     X            save instr byte in "B(X)"
        C=B      X            copy instr. byte to "C(X)"
        RSHFC    X            strip of the row byte
        R= 1
        ?FSET 8               do we have to do row 8
        JNC+         2
        C=C+1    R            yes, add 16 to instr. byte
        ?FSET 9               are we doing row C
```

```
         JNC+          3
         C=C+1     R
         C=C+1     R           yes, add 32 to instr. byte
         A=C       X           save instr. byte in "A(X)"
         ?NC XQ        PCTOC
         R= 5
         LD@R A
         LD@R 9
         LD@R 9                load start address of instr. of row 6,8,C
         R= 1
         A=A-1     X           have we found the wanted instruction
         JC+           7       yes, get the string to "A"
         C=C+1     M
         FETCH
         C=0       R<
         C=C-1     X           is this the end of an instruction
         JC-           4       no, try next character of instruction
         JNC-          7       yes, test if we are finished
         A=0       A           initialize "A" for string
         C=C+1     M
         FETCH                 get a character
         LSHFA     A
         LSHFA     A
         A<>C      R<          append character to string
         C=C-1     X           are we finished yet
         JC-           6       no, do next character
         A=A+1     MS          type it as an alpha string
         LDI           $020    load ascii byte for a space
         R= 11
         ?A#0      R           is the string left justified
         JC+           6       yes, exit
         LSHFA     R<
         LSHFA     R<
         A=A+C     X           no, shift one byte left and append a space
         JNC-          5       and try again
*
* stepping stone
*
STP14    JNC+          DOCLSO
*
         A<>C      A
         WRIT Z                save instruction string in reg Z
         A<>B      A           get instruction code into "A(X)"
         LDI           $ODC
         ?A#C      X           isi it a C=C or A
         JC+           CANDA   no, test for C=C and A
         C=0       A
         R= 7
         LD@R 4
         LD@R F
         LD@R 5
         LD@R 2                load 'OR'
APP_A    LD@R 2
         LD@R 0
         LD@R 4
         LD@R 1                load '_A'
```

```
 EX1      C=C+1    MS            type as alpha
          WRIT T                 and save in reg T
          LDI          $04C
          ?A#C      X            is it a LDI
          JNC+         LDI1
          C=0       A            no, normal instruction go clear reg T and L
C=0       A
          WRIT L
          RTN
 CANDA    LDI          $0EC
          ?A#C      X
          JC+          A
          C=0       A
          R= 9
          LD@R 4
          LD@R 1
          LD@R 4
          LD@R E
          LD@R 4
          LD@R 4                 load 'AND'
          JNC-         APP_A     append '_A' and exit
          C=0       A
          JNC-         EX1
LDI1      LDI          $002
          WRIT L                 tell MNEM for second byte that we have a LDI
          RTN
 *
 * here are the normal instructions from class 0 handled. these are the
 * instructions that do have a parameter with them
 *
 DOCLS0   ?NC XQ       PCTOC
          R= 5
          LD@R B
          LD@R 8
          LD@R 5                 load start address of first string
 FSTR     B=A       X            save instr. code in "B(X)"
 CLS 03   R= 0                   use only the instruction number and not param.
          A=A-1     R            have we reached our instruction
          JC+          CLS 01    yes
 CLS 02   C=C+1     M
          FETCH                  get next character
          R= 2
          ?C#0      R            is it the end of the string
          JNC-         CLS 02    no, try next character
          JNC-         CLS 03    yes, go test if we are finished
 CLS 01   R= 1                   set pointer to fetch and build instr. string
          A=0       A
 CLS 04   C=C+1     M
          FETCH                  get a character of our string
          LSHFA     A
          LSHFA     A
          A<>C      R<           append it to the string
          C=C-1      X           have we done all characters
          JC-          CLS 04    no, do next character
          A=A+1     MS           type string as alpha data
          A<>C       A
          M=C                    and save it in "M"
          JNC+         8
          LD@R 8
          LD@R 4
          LD@R C
```

```
         PUSH
         C<>B     A
         RTN
*
* stepping stone
*
STPP20   JNC-        6
*
         WRIT Z              put string in reg Z
         ?FSET 9             are we ready
         JNC+        5       no, continue
         C=0      A
         C=C+1    MS         yes, clear reg T
         WRIT T
         RTN
*
         A<>B     A          get instruction code back
         R= 1
         LDI         $0F7
         ?A#C     R          is it a instr. from colum 15
         JC+         CLS 09  no, do parameter tables
         ?FSET 3             is it row 8-F
         JC+         CLS 09  yes, do parameter tables
         ?FSET 2             is it row 0-3
         JC+         TST57   no, test for row 5 or 7
         R= 0
         ?A#0     R          is it the NOP from row 0
         JC+         DOEXCP  no, then it is the an exception
TST57    ?FSET 0             is it row 5 or 7
         JNC+        CLS 09  no, do parameter tables
DOEXCP   R= 1
         A=0      R          leave instr. byte only
         ?NC XQ      PCTOC
         R=5
         LD@R B
         LD@R B
         LD@R 6              load address of exception row D table
         SETF 9              say we don't have to do a parameter anymore
         JNC-        FSTR    go get instruction string
*
CLS 09   ?NC XQ      PCTOC
         R= 5
         LD@R B
         LD@R D
         LD@R 7              load address of normal parameter table
         R= 0
         LDI         $00A
         ?A#C     R          do we have to do the parameters of REGN=C
         JC+         4       no
TREG     R= 4                load table address of C=REGN and REGN=C
         LD@R F
         JNC+        REGN=   finish address of table
         LDI         $00E
         ?A#C     R          do we have to do the parameters of C=REGN
         JNC-        TREG    yes, load address of table
         LDI         $004
         ?A#C     R          do we have to do the LD@R parameters
         JC+         NPARAM  no, do the normal parameters
```

```
LD@R    R= 4
        LD@R E
REGN=   LD@R D              load table address of LD@R parameters
NPARAM  SETF 9              say we don't have to do a parameter anymore
        RSHFA    A          field byte to "A[0]"
        C<>B     A
        C=B      A          copy table address to "B(M)" page digit to "C"
        R= 5
        JNC-        STPP20  go do PARAM
*
*
*
        ORG         $A00
*
        ASCTBL
*
        RMB         $143
*
*
* the following ascii tables belong to the MNEM program. they have to start
* at the above specified address. the last character of every entry in the
* tables has bit 9 of the romword set. e.g. the entry A=0 is coded as
* follows A = 041
*          = = 03D
*          0 = 230
* the @ character means that there is just an empty entry in the table. it
* is coded with 200
*
CLASS2  A=0                 these are the mnemonics for the class 2
        B=0                 instructions
        C=0
        AB EX               the field specifier is picked from the next
        B=A                 table
        AC EX
        C=B
        BC EX
        A=C
        A=A+B
        A=A+C
        A=A+1
        A=A-B
        A=A-1
        A=A-C
        C=C+C
        C=A+C
        C=C+1
        C=A-C
        C=C-1
        C=-C
        C=-C-1
        ?B#0
        ?C#0
        ?A<C
        ?A<B
        ?A#0
        ?A#C
        A SR
        B SR
        C SR
        A SL
```

```
FTBL2   PT              this is the table with the field specifiers
        X               from the class 2 instructions
        WPT
        ALL
        PQ
        XS
        M
        S
CLSS0A  UNUSED          this is a part of the class 0 special
        G=C             instructions
        C=G             they do not have an additional field specifier
        CG EX           with them
        UNUSED
        M=C
        C=M
        MC EX
        UNUSED
        F=ST
        ST=F
        FST EX
        UNUSED
        ST=C
        C=ST
        CST EX
        SPOPND
        POWOFF
        SEL P
        SEL Q
        ?P=Q
        ?LLD
        CLRABC
        GOTOC
        C=KEYS
        SETHEX
        SETDEC
        DISOFF
        DISTOG
        RTN C
        RTN NC
        RTN
        UNUSED
        N=C
        C=N
        NC EX
        LDI
        STK=C
        C=STK
        UNUSED
        GO KEY
        DADD=C
        UNUSED
        DATA=C
        CXISA
        C=C             these two instructions are for the C=C and A
        C=C             and the C=C or A instructions
        PFADD=C
```

```
CLSSOB  S=0                 this is the table of the regular class 0
        S=1                 instructions.
        ?S=1                these instructions can have 3 different
        LC                  parameters.
        ?PT=                these parameters are specified in 3 tables
        @                   empty entry
        PT=
        @                   empty entry
        SELP
        REGN=C
        ?F=1
        @                   empty entry
         C=REGN
        RCR
CLSSOC  @                   empty entry
        CLR ST              this is part of the class 0 special instruction
        RST KB              set.
        CHK KB
        @                   empty entry
        DEC PT
        @                   empty entry
        INC PT
CLSSOD  3                   class 0 parameter table for the following
        4                   instructions
        5                   NOP   S=0   S=1   ?S=1   RCR   ?F=1
        10
        8                   SELP   ?PT=   PT=
        6
        11
        14
        2
        9
        7
        13
        1
        12
        0
        15
CLSSOE  0                   this is the table with the parameters for
        1                   LC instruction
        2
        3
        4
        5
        6
        7
        8
        9
        A
        B
        C
        D
        E
        F
```

```
CLSSOF  0(T)                    this is the parameter table for the
        1(Z)                    instructions C=REGN and REGN=C
        2(Y)
        3(X)
        4(L)
        5(M)
        6(N)
        7(O)
        8(P)
        9(Q)
        10(')                   the ' stands for the lazy T, code 07F
        11(a)
        12(b)
        13(c)
        14(d)
        15(e)
*
*
*
* ROM>REG
*
        NAM         ROM>REG
*
* start initialization ROM>REG
*
        GOSUB       LREG    get address of last existenet register
        C=0      X
        RAMSLCT
        READ X              get start reg from X
        ?NC XQ      BCDBIN
        A=C      X          save start reg number in "A(X)"
        READ C
        RCR 3               address of reg 00 to "C(X)"
        C=C+A    X
        M=C                 address of start reg SSS to "M"
        A=C      X          and to "A(X)"
        A<>B     X          last reg to "A(X)" SSS to "B(X)"
        C=A-C    X          number of available registers to "C(X)"
        A=C      X
        C=C+C    X          *2
        C=C+C    X          *4
        A=A+C    X          *5
        A=A-1    X          number of rom words that can be stored
        B=A      X          save max rom words in "B(X)"
        READ Y              get BBBBEEEE
        A=0      A
        R= 3
        A<>C    R<          EEEE to "A[3-0]"
        RCR 4               BBBB to "C[3-0]"
        N=C                 save BBBB in "N"
        C=A-C   R<          number of words to store NNN
        LSHFA    A
        LSHFA    A
        LSHFA    A          EEEE to "A(M)"
        A=C      X          NNN to "A(X)"
        A<>B     X          NNN to "B(X)" available to "A(X)"
        ?A<B     X          is there enough room to store NNN
        ?C GO       ERRNE   no, say 'NONEXISTENT'
        C=M
```

```
         A=C      X           SSS to "A(X)"
         C=N
         RCR 11
         C<>B     M           BBBB to "B(M)"
         C=B      A           x000000BBBBNNN to "C"
         C=0     MS           clear the x "C" = 0000000BBBBNNN
         RCR 7                "C" = BBBBNNN0000000
         A<>C     X           "C" = BBBBNNN0000SSS
         N=C                  save header in "N"
*
* start main loop
*
* the main loop is entered with the CPU registers as follows
* "A(M)" = EEEE this is start address in rom
* "B(M)" = BBBB this is end address in rom
* "M(X)" = SSS  this is the first MM register to use
* "N" = BBBBNNN0000SSS this is the header
*
 MAINLP A<>B     A           EEEE to "B(M)" BBBB to "A(M)"
         R= 5                 set loop counter to 5
 ONEREG A<>C     A           address to "C" and string to "A"
         LSHFA   A
         LSHFA   A
*
* this makes place for the next rom word and shifts a zero into the
* alpha indicator digit, so we are sure that an increment of this digit
* returns the alpha indicator
*
         FETCH
         A=C      X          put the rom word into the string
         A<>C     A          string to "C" for bit shifting
         C=C+C    X
         C=C+C    X          push word up against rest of string
         C=C+C    A
         C=C+C    A          shift string left to even digit
         A=A+1    M          increment address
         R=R-1               decrement loop counter
         ?R= 0               did we do one register
         JNC-        ONEREG  no, do another rom word in this reg
         RSHFC    A          right justify string in "C"
         C=C+1   MS          set alpha indicator digit
         C<>M                save string in "M"
         C=C+1    X          increment register address
         RAMSLCT
         C<>M                save reg address and get back string
         WRITDAT             move string to memory
         A<>B     A
         ?A<B     M          did we do all the rom words we had to do
         JNC-        MAINLP  no, continue
*
* termination of ROM>REG
*
```

```
        C=M
        A=C     X       last used register address to "A(X)"
        C=N             get the header
        RAMSLCT         and select the header register
        C=A-C   X
        C=C+1   X       number of used registers to "C(X)"
        RCR 7           header is 0000RRRBBBBNNN   RRR is used regs
        C=C+1   MS      make it alpha data
        WRITDAT         save header in header register
        C=0     X
        RAMSLCT
        READ C
        RCR 3           reg 00 to "C(X)"
        A=A-C   X       reg number of last used register
        GOSUB       HTOD    make it floating point
        WRIT L          and put it in L for the user
        RTN
*
*
*
* REG>ROM
*
        NAM         REG>ROM
*
        READ Y
        N=C             start address to "N"
        READ X
        ?NC XQ      BCDBIN
        A=C     X       start register number to "A(X)"
        READ C
        RCR 3
        C=C+A   X       address of header register
        RAMSLCT
        M=C             save SSS
        READDAT         get the header register 100000AAAANNN
        RCR 3
        A=C     A       start address of file to "A[3-0]"
        C=N             get the wanted start address
        RCR 1
        C=0     M       leave only the start address
        ?C#0    X       is it a complete address
        JC+         8   yes, go put it in place
        RCR 10          put page number in "C[3]"
        A<>C    X       get address in this page
        R=3
        ?C#0    R       was there a page address given
        JC+         4   yes, put address in place
        A<>C    R       no, write it back at the original place
        JNC+        2
        RCR 13
        RCR 11
        A=C     A       put address in "A(M)"
        READDAT         get the header back again
        C=0     M
        RCR 11          leave only NNN in "C(M)"
        C=C+A   M
        C<>B    A       save end address in "B(M)"
*
* start of main loop
*
```

```
LOOP1   C=M
        C=C+1    X          point to next register
        RAMSLCT             select it
        M=C                 save new register address
        READDAT             get 5 rom words
        RCR 13
        C=C+C    A
        C=C+C    A          left justify string in "C"
        R= 5                set LOOP2 counter for 5 words
LOOP2   RCR 12              shift 2 digits into "C(X)"
        C=C+C    A          shift one bit left
        JNC+        2       have we shift a one bit of at the left
        C=C+1    A          yes, append this bit at the right
        C=C+C    A
        JNC+        2
        C=C+1    A          do the second bit to
        A=C      X          rom word to "A(X)"
        A<>C     A
        WRIT                put the rom word in the ram
        A<>C     A
        ?A<B     M          have we done all the words
        ?NC RTN             yes, we are finished
        A=A+1    M          no, increment address
        R=R-1
        ?R=0                have we done the complete register
        JNC-        LOOP2   no, do the rest of this register
        JNC-        LOOP1   yes, do the next register
*
*
*
* Last REGister
*
* Entry: none
* Exit : "B(X)","C(X)" are holding address of last reg
*         last reg is enabled
* Uses : "A","B(X)","C", enabled chip
*
LREG    LDI         $23F
        C<>B     X          address of last reg + $040 to "B(X)"
LREG 1  LDI         $040
        A<>B     X
        C=A-C    X          subtract $040 of address
        RAMSLCT             select last reg of a 64 reg block
        C<>B     X          save address
        READDAT
        A=C      A          save reg in "A"
        C=-C-1   X          complement exponent of reg
        WRITDAT
        READDAT
        C=-C-1   X          reg ought to be good again
        ?A#C     A
        JC-         LREG 1  it isn't, so its nonexistent
        WRITDAT             restore reg to its original status
        C=B      A          address to "C(X)"
        RTN
*
*
*
* these are a few subroutines that belong to the MMTORAM program
*
```

```
*
* this part takes care of the C handling
*
DO C    R= 0                    set pointer to zero
        ?A<C     R              exceptions in C row
        ?NC RTN                 return if exception
        C=M
        C=C+1    M              address of label to "C(M)"
        RCR 3                   right justify address for AFAT1
        B<>C     A
        RCR 11                  current PC to "B[6-3]" and AAA to "B(X)"
        C=B      X
        B<>C     A
        R= 5
        LD@R 2
        LD@R 0                  say we have user code and offset is zero
        A<>C     A
*
* "B" has the current PC in [6-3] for the main loop of MMTORAM and the last
* three digits of the address in "B(X)" for AFAT1
* "A[5-0]" has U0PAAA for AFAT1
*
        GOSUB       AFAT1       append FAT entry and error if entrys > 64
        B<>C     A
        RCR 3
        B<>C     A              put PC back into "B[3-0]"
        RTN
*
*
*
* this part takes care of the B-E bytes ( 2 and 3 bytes GTO and XEQ and the
* alpha labels the last have to be appended to the FAT
*
DO B-E  LDI         $0CE        set up DE, C, B compare values
        ?A<C     R              is it B
        JNC+        NO B        no, continue
        GOTO        DO B        yes, handle B
NO B    ?A#C     R              is it C
        JNC-        DO C        yes, handle C
*
* handling the D bytes consist of converting the register length into a
* byte count, add the normal byte count to it and to invert the direction
* bit
*
        A<>C     X
        ST=C                    save byte in "ST" for bit testing
        R= 3
        A<>B    R<              get PC back to "A[3-0]"
        ?NC XQ      NXBYTA      get second byte of this instr.
        A<>B    R<              save PC in "B[3-0]"
        R= 2
        C=0      R              set up for conversion
        ?FSET 0                 is 1xx bit set
        JNC+        MLTREG
        C=C+1    R              yes, add 256 registers to the count
        CLRF 0                  leave byte count only in "ST"
*
* multiply the register count by 7 and add the byte count to it
*
```

```
MLTREG  A=C       X
        C=C+C     X
        C=C+C     X
        C=C+C     X
        A<>C      X
        A=A-C     X          regs * 7 in "A(X)" this is byte count of regs
        C=ST
        LD@R 0
        LD@R 0               get the byte count
        C=C+C     X
        C=C+C     X
        C=C+C     X
        C=0       M
        RCR 1                shift byte count three bits right
        A=A+C     X          total byte count of label in "A(X)"
        R= 2
        C=G                  get identifier back to "C[3]"
        A<>C      X          add link bytes
        G=C                  save first byte in "G"
        ST=C                 and second in "ST"
        C=M
        C=C+1     M          get storage address
        LDI          $100    set first byte of instr. flag
        R= 0
        C=G                  get first byte
        WRIT                 and put it in ram
        CLRF 8               say we have had the first byte
        C=C+1     M          make storage address of second byte
        M=C                  and save it
        C=ST                 get second byte
        C=0      XS          clear first byte flag
        WRIT                 and save it
*
* last thing to do here is to get the last byte and invert the direction
* bit
*
        R= 3
        A<>B     R<          get PC to "A[3-0]"
        ?NC XQ       NXBYTA  get the last byte
        A<>B     R<          save PC in "B[3-0]"
        ST=C                 put byte in "ST" for inverting direction bit
*
* invert the direction bit
*
        ?FSET 7
        JC+          DIR1
        SETF 7
        JNC+         DIR2
DIR1    CLRF 7
DIR2    R= 0
RMLOOP  C=ST
        G=C                  save byte in "G"
        RTN                  return to the main loop
*
*
*
* this part takes care of the handling of the B bytes
*
```

```
DO B    C=M
        C=C+1    M          get storage address
        M=C                 save it
        R= 2
        LD@R 1              set first byte flag
        R= 0
        C=G                 get first byte
        WRIT                put it in ram
*
* test direction bit of distance byte and compute the distance in bytes
*
        CLRF 8              use this flag for copy of direction bit
        R= 3
        A<>B    R<          get PC
        ?NC XQ      NXBYTA  get the distance byte
        A<>B    R<          save PC
        ST=C                put distance byte in "ST" for testing direction
        ?FSET 7             is direction bit set
        JNC+        3
        SETF 8              yes, copy it into flag 8
        CLRF 7              clear distance byte for calculation of distance
        C=ST
        C=0     XS
        R= 1
        G=C                 save bytes in "G"
*
* multiply register length by 7
*
        LD@R 0
        A=C     X
        C=C+C   X
        C=C+C   X
        C=C+C   X
        A<>C    X
        A=A-C   X           REG * 7 = bytes in "A[1-0]"
        C=0     X
        C=G                 get bytelength to "C[0]"
        C=C+A   X           total byte length to "C[1-0]"
        ST=C
        ?C#0    X           is the distance to far
        JNC+        4       yes, leave direction bit zero
        ?FSET 8             was direction bit set
        JC+         2       yes, invert it
        SETF 7
        CLRF 8              say we have had the first byte
        JNC-        RMLOOP  save byte and return to the main loop
*
*
*
* Main Memory TO RAM
*
        NAM     10  MMTORAM give an alpha prompt
*
        FCB         $000    indicate nonprogrammmable
*
* initialize for the main loop
*
```

```
         GOSUB        INMMTR   go do compile
         GOSUB        PATCH    get start address and say "LOADING PGM"
         CLRF 10               indicate we are in MM
         RCR 8
         A=C     R<            set up end for CPGMHD
         ?NC XQ       CPGMHD
         ?NC XQ       PUTPCF   set PC to begin of MM program
         C=0      X
         RAMSLCT               select the status registers
         READ A                get begin address of storage
         RCR 11                rotate to "C[6-3]"
         WRIT A                begin address to reg A
         C=C+1    M            skip the two header words
         M=C                   save code address-1 in "M"
*
* start of the main loop of MMTORAM
*
DINSTR   A<>C     R<
         N=C
         A<>C     R<           save begin of program in "N"
         ?NC XQ       NXLSST   end of this instruction to "A[3-0]"
         CLRF 9                clear the copy of the end flag
         ?FSET 6               have we found an END
         JNC+         NOEND
         SETF 9                copy status of flag 6 into flag 9
NOEND    A<>C     R<
         C<>N
         A<>C     R<           begin of instr. to "A[3-0]", end to "N[3-0]"
         SETF 8                indicate we do first byte of instr.
*
* start of the 'instruction loop'. in this loop one instruction is saved
* in ram memory
*
DOBYTE   ?NC XQ       NXBYTA   get a byte of this instr.
         A<>B     R<           current PC to "B[3-0]"
         R= 0
         G=C                   fetched byte to "G"
         ?FSET 8               is this the first byte
         JNC+         NOTB-E   no, skip byte test
         ?FSET 9               have we hit an end
         JC+          NOTB-E   yes, skip byte test
         R= 1
         A=C      X            save byte in "A[1-0]"
         C=C+1    R            test for F byte
         JC+          NOTB-E   skip B-E handling
         LDI          $0BE     set up comparing constant
         ?A<C     R            if smaller than B, we can skip B-E handling
         JC+          NOTB-E
         GOSUB        DO B-E   take care of B-E bytes
NOTB-E   C=M                   get storage address-1
         C=C+1    M
         C=0      X            clear data space
         M=C                   save updated storage address
         R= 2
         ?FSET 8               are we handling the first byte
         JNC+         NOTONE   if not, skip the flagging of byte one
         LD@R 1                flag first byte
```

```
NOTONE  R= 0
        C=G                     get byte
        CLRF 8                  say first byte is done
        WRIT                    put byte in ram
        C=N                     get end of instr.
        R= 3
        A<>B    R<              address of byte to "C[3-0]" end to "A[3-0]"
        ?FSET 9                 are we handling an END
        JC+             DOEND   skip PC compare
        ?A#C    R<              have we handled the whole instr.
        JC-             DOBYTE  no, do the next byte of this instr.
        JNC-            DINSTR  yes, do next instr.
*
* make the END instr. in ram and test if it has to be a private END or not
* and if so make it private
*
DOEND   ?NC XQ          NXBYTA  get next byte
        ST=C                    save byte in "ST"
        C=M
        C=C+1    M              make new storage address
        C=ST                    get byte back
        C=0      XS             clear first byte indicator
        WRIT                    put byte in ram
        C=C+1    M              point to last address of program in ram
        M=C                     save last used address for getting out first
*                               free byte after program for the user
        LDI             $22F    set up last byte of END
        C<>B     A              address and byte to "B"
        C=0      X
        RAMSLCT                 select the status registers
        READ D
        RCR 13
        ST=C                    user flags 0-3 into CPU 3-0
        C=B      A              get back address and byte
        R= 1
        ?FSET 2                 must it be a private END
        JNC+            NOPRIV  no
        LD@R 6                  yes, indicate write protected file with $26F
NOPRIV  WRIT                    put byte in ram
*
* construct the header words for a ROM program file the first header word
* contains the length of the program in registers and the second word
* holds the status of the file ( private or not ) in the bit 8 and the
* number of bytes that are not hold by a completely filled up register in
* digit 1
*
        C=C-1    M              set up "C" for byte count
        A<>C     M              save end of program in "A"
        READ A                  get start address
        A=A-C    M              number of bytes to "A(M)"
        C=0      A              set up for divide by 7 for header words
        R= 3
        LD@R 7
DIVIDE  C=C+1    X              open at least 1 register
        ?A<C     M              end of division
        JC+             EOFDIV  yes
        ?A#C     M              integer division
        JNC+            EOFDIV  yes, exit
        A=A-C    M              subtract ( this is a dumb form of division )
        JNC-            DIVIDE
```

```
EOFDIV  A<>C     X          save number of regs in "A[2-0]"
        READ A              get address of first header word
        A<>C     X
        WRIT                write first header word in ram
        C=C+1    M          point to second header word
        A<>C     M          number of bytes to "C(M)" address to "A(M)"
        RCR 2
        A<>C     X          bytecount to "A[1]"
        LDI         $300    say status is private
        ?FSET 3             does the user want it to be private
        JC+         PRIVAT  yes
        LD@R 2              no, make it an open file
PRIVAT  A<>C    XS          combine status and number of bytes in "A(X)"
        A<>C     A
        WRIT                put second header word in ram
*
* last thing to do is to compute the link in the first alpha label to the
* label that will be in front of the program, when it will be copied to MM
* for this we need to know how many empty bytes are left in the first reg
* and the distance from the program start to the first alpha label
*
        A=C      M          start address-1 to "A(M)"
        C=0      M
        C=0     XS          clear file status, preserve bytes
        RCR 1
        A<>C     X          number of bytes to "A[0]"
        LDI         $007
        A<>C     X
        A=A-C    X          calculate 7-bytes ( zero bytes in first reg )
        A=A+1    X
        A=A+1    X          add displacement to previous link-1
        B=A      A          save start-1 and offset
*
* compute the distance to the first label starting from the first byte
*
        LDI         $1CD
        A<>C     X          comparing constant for G labels to "A(X)"
        C=B      M          get start-1
        R= 1
NXTBYT  C=C+1    M
        FETCH               get next byte of the new file
        ?C#0    XS          is this a start of an instr.
        JNC-        NXTBYT  no, continue with next byte
        ?A#C     R          is it a C byte
        JC-         NXTBYT  no, continue with the next byte
        ?A<C     X          one of the exceptions in C row
        JC-         NXTBYT  yes, look up another byte
        A<>C     M          start-1 to "C(M)", end to "C(M)"
        B=A      M          save end ( bytes to change )
        A=A-C    M          number of bytes to first label in "C(M)"
*
* we have the number of bytes to the previous link
*
        C=0      A          set up for link distance computation
        C=B      X
        RCR 11              offset value to "C(M)"
        A=A+C    M          total link distance in "A(M)"
        C=0      A          set up for division
        R= 3
        LD@R 7
```

```
DIVLNK  ?A<C     M          division finished
        JC+         LFOUND  yes, exit
        A=A-C    M
        C=C+1    X          increment register count
        JNC-        DIVLNK
LFOUND  A<>C     A          bytes in "C(M)", regs in "A(X)"
        C=C+C    M          double byte count
        C=0      X          clear dat area
        RCR 1               get bytes in place
        C=C+A    X          add register count
        ST=C                save second byte
        RCR 2               create first byte
        LD@R 1
        LD@R C
        C=B      M          get address for first byte
        WRIT                put it to ram
        C=C+1    M          point to second byte
        C=0      X
        C=ST                get second byte
        WRIT                put it in ram
EXIT    C=M                 get back last word of program
        C=C+1    M          make first free word after program
        RCR 4
        C=0      M          leave only the address
        RCR 13              make the output the same as the input format
*
* now we initialize for DECOD and put the next free byte address back to
* the alpha register. then we exit through READY of compile, saying "READY"
* and giving an audio warning
*
        C<>B     A
        A=0      A
        C=0      A
        SETF 8
        CLRF 9
        WRIT 0
        WRIT P              initialization done
        GOSUB       DECOD   put address in alpha
*
* we can exit through READY when we make sure that flag 0 of reg E is
* cleared.
*
EXIT2   READ E
        RCR 3
        ST=0
        C=ST                indicate we have finished compile
        RCR 11
        WRIT E
        GOTO        READY   and do exit and say "READY"
*
*
*
* ROMSUM
*
        NAM         ROMSUM
*
```

```
         GOSUB        CODE    get the page to romsum
         RCR 11               P to "C[3]"
         C=0       X
         C=-C-1    X
         RCR 11               PFFF to "C(M)"
         C=0       X
         WRIT                 clear the romsum word
         R= 5
         C=0      R<
         C<>B      A          start address to "B(M)"
         C=0       A
         RAMSLCT              select the status registers
         PRPHSLCT             and deselect all peripherals
         LDI          $300
         C=C+C     X
         C=C+C     X
         A=C       A          "A(X)" has %1100 0000 0000
         C=C+1     X
         C<>B      A          "B(X)" has %1100 0000 0001  "C(M)" has start
*
* the checksum is computed by adding all the words to each other and when a
* carry occurs to add this carry to the total checksum. the final checksum
* is made by taking the 2's complement of the added words.
*
* detection of the carry is done by making the two left bits high. when the
* carry in the right ten bits occur, it will be transferred to bit 12 and
* this is the carry bit. we add the carry and restore the status of the two
* left bits in the S&X field of "A" by adding %1100 0000 0001 to "A(X)"
*
ROMS     FETCH                get a rom word
         A=A+C     X          and add it to the checksum
         JNC+         2       we didn't had a carry
         A=A+B     X          add one to the checksum and restore bit 10 + 11
         C=0       X
         C=C+1     M          point to next word to add
         ?C#0     R<          have we had the whole rom yet
         JC-          ROMS    no, do next word
         A=A-B     X          decrement "A(X)" and get rid of bit 10 and 11
         LDI          $3FF
         A<>C      A
         C=A-C     A          get the 2's complement of bit 0-9 to "C(X)"
         C=C-1     M          point to checksum word
         WRIT                 put checksum into the ram
         RTN
*
*
*
* DISASSeMble
*
         NAM          DISASM
*
         SETHEX
         ?NC XQ       CLA
         LDI          $006
         RCR 11
         LDI          $009
         A=C       A          $006009 to "A[5-0]"
         READ X               get word to disassemble
         C=C+1     A
         WRIT X               save address of next word for the user
```

```
        C=C-1    A
        RCR 11
        FETCH                 get the word to disassemble
        WRIT Y                word and address to reg Y for MNEM
        RCR 7                 left justify string for conversion to ascii
DISAS1  LDI          $003     load first ascii digit of a character
        RCR 13
        R= 0
        ?A<C     R            is it a number 9-0
        JNC+         5        yes, then we are ready with converting
*
* when the hex digit is A-F we have to make it 1-6 and add 1 to the first
* digit of the ascii character
*
        C=A-C    R
        C=-C     R            convert digit to 1-6
        R= 1
        C=C+1    R            make first digit 4
        C<>B     A            save string with converted character in "B"
        READ M                get the already converted digits
        RCR 12                and make place for the new character
        C<>B     X
        C<>B    XS            put new character in place
        WRIT M                save converted digits
        C<>B     A            get string back for next character
        A=A-1    M            are we finished with characters
        JNC-         DISAS1   no, do one more
*
* the disassembled word and address are written in alpha as follows
* AAAA_DDD_C_ where '_' stands for a space the C stands for the ascii
* representation of the word and is added in the conversion part
*
        READ M                get string back
        A=C      A            and save it in "A"
        R= 5
        C=0     R<            leave only the address part
        RCR 6                 right justify it
        WRIT N                and write it to reg N of the alpha register
        A<>C     A            get string back
        RCR 8
        LD@R 2                load three spaces after the data word
        LD@R 0
        LD@R 2
        LD@R 0
        LD@R 2
        LD@R 0
        R= 13
        LD@R 2                load one space before data word
        LD@R 0
        WRIT M                write string to reg M of the aloha register
        READ Y
*
* get the data word back and decide by which character it has to be
* represented. the upper bits of the word are ignored for the ascii
* character as is bit three of the second digit ( bit 7 )
*
* words where the second digit is 4,5,6 or 7 are represented by the
* characters which are  picked from the ASCTBL at $2CF0. which character
* is used is determined by the first digit of the word
*
```

```
* words where the second digit is 2 or 3 are represented by their ascii
* value
*
* words where the second digit is 0 or 1 are represented by the characters
* that are represented by the value of the word added with $40
*
        ST=C
        CLRF 7
        C=ST                    bit 7 of the word is ignored
        ?FSET 6                 is first digit 4,5,6 or 7
        JNC+        8           no, test for digit 0,1 or 2,3
        R= 3                    load address of ASCTBL the character to
        LD@R 2                  be picked is determined by the first digit
        LD@R C                  of the word we use
        LD@R 0
        RCR 11                  address to "C(M)"
        FETCH                   get ascii value
        JNC+        5           end of conversion
        ?FSET 5                 is second digit a 2 or 3
        JC+         3           yes, represent by ascii #20 upto $3F
        SETF 6                  no, represent by ascii $40 upto $5F
        C=ST                    get converted word $00 upto #1F
        A=C      A              save character in "A[1-0]"
        READ M
        RCR 2
        A<>C      X
        A<>C     XS
        RCR 12                  append character to alpha string
        WRIT M
        ?FSET 13                is there a program running
        ?C RTN                  yes, return to mainframe
        ?NC GO      AVIEW       no, put string in display
*
*
*
* COPY Rom
*
        NAM         COPYR
*
        GOSUB       COD         get SD to "C[1-0]"
        R= 6
        RCR 9                   S to "C[6]"
        A=0      A
        A<>C     R              leave S000 in "A(M)"
        RCR 13
        B=0      A
        C<>B     R              D to "B[6]"
        A<>C     R
        A=C      R              S to "C[6]"
        A=A+1    R              S+1 to "A[6]" for stop criterium
COPY    FETCH                   get a source word
        C<>B     R              get destination address
        WRIT                    write source word to destination
        C<>B     R              get source address back
        C=C+1    M              increment source and destination address
        ?A#C     R              have we had the whole page yet
        JC-         COPY        no, do the next word
        LDI         $027
        ?NC GO      TONEB       give a beep and return to mainframe
*
*
```

```
*
* ROMCHecKX
*
         NAM          ROMCHKX
*
         C=0      A
         RAMSLCT
         READ X
         ?NC XQ       BCDBIN
         C<>B     X           xrom number to check to "B(X)"
         C=0      A
         R= 6
         LD@R 4
         C<>B     M
         CLRF 9               reset error flag
TRYNRM   C=B      M
         R= 6
         C=C+1    R           point to next rom
         JNC+         4       rom not found yet
         GOTO         NOROM   say specified rom not found
         C<>B     M
         C=B      M
         FETCH                get rom ID
         ?C#0     X           is this an existent rom
         JNC-         TRYNRM  no, try next rom
         A<>B     X
         B=A      X
         ?A#C     X           is it the one we want to check
         JC-          TRYNRM  no, do next rom
         ?NC XQ       CLLCDE
         A=0     MS
         ?NC XQ       GENNUM  get xrom number in 2 characters to display
         LDI          $020
         WRIT E               append a blank
         C=0      A
         R= 5
         LD@R F
         LD@R F
         LD@R E
         A=C      M
         C=B      M
         C=C+A    M           construct address of revision number
         FETCH
         WRIT E               append one rev. character to display
         C=C-1    M
         FETCH
         WRIT E               append second rev. character to display
         LDI          $02D
         WRIT E               seperate rev. level and characters with a "-"
         C=C-1    M
         FETCH
         WRIT E               append one revision level to display
         C=C-1    M
         FETCH
         WRIT E               append second revision level to display
         ?NC XQ       MESSL
         RMB          $04     " TST"
*
* display has now message NN RR-LL TST where NN is the desired xrom number
* RR is the revision code and LL is the revision level.
*
```

```
         C=B      A           get start address of rom to "C(M)"
         LDI         $300
         C=C+C    X
         C=C+C    X
         A=C      A           "A(X)" = 1100 0000 0000 start address to "A(M)"
         C=C+1    X
         C<>B     X           "B(X)" = 1100 0000 0001
         C=0      A
         C=C+1    A
         RCR 8
         A<>C     M           "C(M)" has start address
         A=A+C    M           "A(M)" has end address
*
* in this loop all the rom words are added to create the chcksum. when the
* rom is alright the checksum should end up as zero.
*
WORD     FETCH               get word
         C=C+1    M          increment address
         A=A+C    X          add to checksum
         JNC+        2
         A=A+B    X          add cary to checksum and fix high order bits
*
* we need to fix the high order bits to detect the next carry.
*
         ?A#C     M          are we finished yet
         JC-         WORD    no, do next word
         A=A-B    X
         A=A-1    X          is checksum zero
         JC+         2
         SETF 9              no, set error flag
         READ D
         READ D
         READ D              get the TST characters of the display
         ?FSET 9             is rom ok
         JC+         BAD     no, append message BAD
         ?NC XQ      MESSL   yes, append message OK
         RMB         $03     "OK "
         JNC+        EXRCHK
         ?NC XQ      MESSL
         RMB         $03     "BAD"
         JNC+        EXRCHK
NOROM    SETF 9              set error flag
         ?NC XQ      CLLCDE
         ?NC XQ      MESSL
         RMB         $07     "NO ROM "
         A<>B     X
         A=0      MS
         ?NC XQ      GENNUM
         ?NC XQ      LEFTJ
EXRCHK   ?NC XQ      ENCPOO
         ?NC XQ      STMSGF
         ?FSET 9
         ?NC RTN
         SETF 7
         ?NC GO      SKP
*
*
*
* SYNTesize
*
```

```
        NAM         SYNT
*
        FCB         $000    say that this function is non programmable
        READ Y              get the prefix byte
        ?NC XQ      CHK#S   check for alpha data
        SETHEX
        ?NC XQ      BCDBIN
        C<>B    X           save prefix in "B[1-0]"
        READ X              get the postfix byte
        ?NC XQ      CHK#S   check for alpha data
        SETHEX
        ?NC XQ      BCDBIN
        RCR 2               postfix to "C[13-12]"
        C<>B    X
        RCR 2               postfix to "C[11-10]" prefix to "C[13-12]"
        ?NC GO      INSLIN  insert the byte in main memory
*
*
*
* Go to End
*
        NAM         GE
*
        C=0     A
        WRIT A
        WRIT B              delete all pending return addresses
        CLRF 10             say PC is in RAM
        READ C              get address of register of .END
        R= 3
        LD@R 6              make address of .END
        R=3
        A<>C    R<          initialize for PUTPC
        ?NC GO      PUTPCF  place PC at .END and set line number to $FFF
*
*
*
* INitialize Main Memory To Ram
*
* here we save the return address for MMTORAM in the first 4 digits of
* scratch register P. these digits are used by COMPILE toget it's return
* address from, in case we use it as a subroutine.
*
INMMTR  POP                 get return address
        RCR 4
        C=0     M
        RCR 3               leave return address only
        WRIT P
*
* second thing we have to do here is to find out wether the user wants to
* delete the labels from the program, or just wants to do a normal compile
* and load program. this is indicated by flag 3 of the user flags.
*
```

```
        READ D
        RCR 13
        ST=C                    type of compile to flag 0
        CLRF 1                  delete status of user flag 2
        SETF 2                  say we use compile as a subroutine
        READ E
        RCR 3
        C=ST                    put flags in place
        RCR 11
        WRIT E                  store flags in reg E
        GOTO        COMPIL
*
*
*
* PATCH MMTORAM
*
PATCH   GOSUB       COD         get the start address
        WRIT A
        ?NC XQ      CLLCDE
        ?NC XQ      MESSL
        RMB         $0C         "LOADING PGM "
        ?NC XQ      LEFTJ       put message in display
        ?NC XQ      ENCP00
        GOTO        FPAL        find links and return to MMTORAM
*
*
*
* ROM NAME
*
        NAM         ESMLDL-OS
*
        RTN
*
*
*
* MaKe PRivate
*
        NAM      10 MKPR
*
        FCB         $000
*
        GOSUB       FPAL
        RCR 8
        A<>C    R<
        ?NC XQ      OFEE        get third byte to "ST" from address in "A[3-0]"
        CLRF 5
        SETF 6                  set private bit
        C<>ST
        SETF 12                 set indication for cpu of private
        ?NC XQ      PTBYTA
        ?NC GO      NFRPU       return and update private status
*
*
```

```
*
* no entry points used
*
*
        ORG         $FFB
*
        FCB         $001    revision level is  A
        FCB         $020
*
        FCB         $013    revision code is AS
        FCB         $001
*
        FCB         $168    checksum
*
*
        END
*
*****************************************************************************
*                                                                           *
*                          END OF ASSEMBLY LISTING                          *
*                                                                           *
*****************************************************************************
*
*
*
*
*                              ERAMCO SYSTEMS
*
*
*
```