```
; FILENAME: Init.TEXT
;
;
;           Change Log
;
;           12-Aug-84   Added init of NMI key for Lisa
;           13-Aug-84   Added RM command (return to Mac)
;           16-Aug-84   Removed handling of levels 4-6 interrupts
;           20-Aug-84   Fixed Find Long (misses if out of word phase)
;           24-Aug-84   Added Exit-to-shell cmd, Find now goes 1 byte/check
;            1-Sep-84   Added Lisa check
;            8-Sep-84   Changed WhereAmI call to LookupPC, fixed Find command printout.
;            9-Sep-84   Moved most of DM code to DisplayMem routine, moved Bin2Char to
Print file
;           10-Sep-84   Re-installed 4-6 int vectors if on Mac
;           14-Sep-84   Added stack crawl command
;           22-Sep-84   Stack crawl works, even tries to get good return address/checks
for last routine
;           23-Sep-84   Changed CS cmd to use two addresses as arguments
;           25-Sep-84   Added IOPB/WIND/TERC templates
;            1-Nov-84   Fixed WH cmd w/patched traps, also consistant display
;            1-Nov-84   Fixed WH non-restore of A-trap dispatcher address
;            8-Dec-84   Removed initial save of PostEvent/BlockMove address (could be
changed by app,
;                       must be dynamic)
;

MAXBASE
            MOVEQ       #$9, D1                 ; + 9 for a magic location...
            ADD.L       RomBase, D1             ; use the global system value
            MOVE.L      D1, A1
            CMP.B       #$FF,(A1)               ; are we on a Lisa?
            .IF         onLisaTrue
                BEQ.S       @0                      ; yes & for Lisa, so cool
            .ELSE
                BNE.S       @0                      ; no & for Mac, so cool
            .ENDC
            MOVEQ       #26,D0                  ; return 'Bad Launch' error
            .WORD       $A9C9                   ; _syserror

@0          LEA         BUSERR,A0               ; install the vectors
            MOVE.L      A0,$8
            LEA         ADDRERR,A0
            MOVE.L      A0,$C
            LEA         ILLEGAL,A0
            MOVE.L      A0,$10
            LEA         DIVZRO,A0
            MOVE.L      A0,$14
            LEA         CHKINST,A0
            MOVE.L      A0,$18
            LEA         OVRFLW,A0
            MOVE.L      A0,$1C
            LEA         TRACE,A0
            MOVE.L      A0,$24

            LEA         LN1111,A0
            MOVE.L      A0,$2C
            LEA         ABORTB,A0

; kwk -- removed interrupt level 4,5,6 processing for Lisa.
; (serial driver runs at level 6, screws up MacsBug)
```

```
            .IF         onMacTrue
                MOVE.L      A0,$70                  ; level 4 interrupt vector
                MOVE.L      A0,$74                  ; level 5 interrupt vector
                MOVE.L      A0,$78                  ; level 6 interrupt vector
            .ENDC

            MOVE.L      A0,$7C                  ; level 7 interrupt vector

            LEA         CHKBP,A0                ; trap F vector
            MOVE.L      A0,$BC                  ; for breakpoints

            LEA         SAVEA,A0
            MOVE.L      $28,(A0)                ; save current A trap
            LEA         TA,A0                   ; get trap intercept
            MOVE.L      A0,$28                  ; install ours

            LEA         REGPC,A0
            MOVE.W      #dbgWrdCnt-1,D0         ; (twm) clear out globals

INIT        CLR.W       (A0)+
            DBRA        D0, INIT                ; (twm) use loop mode

SETSCREEN
            .IF         noTerm
                LEA         screenVars,A0
                CLR.L       (A0)+               ; x,y=0
                MOVE        #dOffset,(A0)+      ; offset
                MOVE        #dLines,(A0)
            .ENDC

            .IF         onLisaTrue              ; twm - kwk
                MOVEQ       #$21,D0             ; NMI key = '-' on keypad
                TRAPTO      _SetNMIKey          ;  and set it
            .ENDC                               ; (kwk)

            ST          RUN
            LEA         SysCmds,A0              ; copy the commands into the global
space
            LEA         rSysCmds,A1
            MOVEQ       #0,D0                   ; clear count
            MOVE        #sizeCmds,D0            ; sizecmds = 4*num of cmds = # of bytes
to move
            _BlockMove

            .IF         DDBG
; set up keyboard constants

                MOVE.W      #kbdRoll,KeyRoll    ; x/60 rollover value
                MOVE.W      #kbdWait,KeyWait    ; N tight loops waiting for keyboard
            .ENDC

            BRA         resident

            RTS                                 ; This all gets cut back...

; This is the command table which gets moved into the top of the global area.
; If you add or subtract a command, remember to adjust the SizeCmds
; constant in ROM4EQU.

SYSCMDS

; Memory commands
```

```
                .ASCII      'DM'
                .WORD       DMCMD-SYSCMDS
                .ASCII      'SM'
                .WORD       SMCMD-SYSCMDS

                .IF         fullSize
                .ASCII      'CS'
                .WORD       CSCMD-SYSCMDS
                .ENDC

                .IF         DDBG
                .ASCII      'DD'
                .WORD       DDCMD-SYSCMDS
                .ASCII      'KR'
                .WORD       KRCMD-SYSCMDS
                .ASCII      'KW'
                .WORD       KWCMD-SYSCMDS
                .ENDC

; Register commands

                .ASCII      'D@'
                .WORD       SETD-SYSCMDS
                .ASCII      'A@'
                .WORD       SETA-SYSCMDS
                .ASCII      'SR'
                .WORD       SETSR-SYSCMDS
                .ASCII      'PC'
                .WORD       SETPC-SYSCMDS
                .ASCII      'TD'
                .WORD       TDCMD-SYSCMDS

; Control commands

                .IF         fullSized
                .ASCII      'BR'
                .WORD       BRCMD-SYSCMDS
                .ASCII      'CL'
                .WORD       CLCMD-SYSCMDS
                .ASCII      'GT'
                .WORD       GTCMD-SYSCMDS
                .ENDC

                .ASCII      'RB'
                .WORD       RBCMD-SYSCMDS

                .ASCII      'ST'
                .WORD       STCMD-SYSCMDS
                .ASCII      'SS'
                .WORD       SSCMD-SYSCMDS

                .ASCII      'G '
                .WORD       GCMD-SYSCMDS
                .ASCII      'T '
                .WORD       TCMD-SYSCMDS
                .ASCII      'S '
                .WORD       SCMD-SYSCMDS

                .IF         fullSized
                .ASCII      'MR'                    ; Magic return command
                .WORD       MRCMD-SYSCMDS
```

```
            .ENDC

; A-trap commands

            .ASCII      'AT'                    ; Trace A traps
            .WORD       ATCMD-SYSCMDS
            .ASCII      'AB'                    ; break on a traps
            .WORD       ABCMD-SYSCMDS
            .ASCII      'AX'                    ; clear any a trap cmd
            .WORD       AXCMD-SYSCMDS

            .ASCII      'AS'                    ; spy data on a traps
            .WORD       ASCMD-SYSCMDS
            .ASCII      'AR'                    ; record Atrap info
            .WORD       ARCMD-SYSCMDS
            .ASCII      'AH'                    ; check heap on a traps
            .WORD       AHCMD-SYSCMDS
            .ASCII      'HS'                    ; scramble heap on some A traps
            .WORD       HSCMD-SYSCMDS

; heap commands
            .IF         fullSized

            .IF         noTerm=0
            .ASCII      'HP'                    ; print heap
            .WORD       HPCMD-SYSCMDS
            .ENDC

            .ASCII      'HD'                    ; dump heap
            .WORD       HDCMD-SYSCMDS
            .ASCII      'HT'                    ; dump heap totals
            .WORD       HTCMD-SYSCMDS

            .ASCII      'HC'                    ; check heap one shot
            .WORD       HCCMD-SYSCMDS
            .ASCII      'HX'                    ; toggle heap
            .WORD       HXCMD-SYSCMDS

            .ENDC                               ; fullsized

; Miscellaneous commands

            .IF         fullSized
            .ASCII      'WH'                    ; Where command
            .WORD       WHCMD-SYSCMDS
            .ASCII      'F '                    ; Find command
            .WORD       FCMD-SYSCMDS
            .ASCII      'ES'                    ; Exit command
            .WORD       ESCMD-SYSCMDS
            .ASCII      'CV'                    ; Convert
            .WORD       CVCMD-SYSCMDS
            .ASCII      'RX'                    ; Register toggle
            .WORD       RXCMD-SYSCMDS
            .ASCII      'PX'                    ; Pascal symbol toggle
            .WORD       PXCMD-SYSCMDS
            .ASCII      'SC'                    ; stack crawl
            .WORD       SCCMD-SYSCMDS
            .ENDC

; Disassembler commands

            .IF         withDis
```

```
            .ASCII      'ID'                    ; one line disassemble
            .WORD       IDCMD-SYSCMDS
            .ASCII      'IL'                    ; multi line disassemble
            .WORD       ILCMD-SYSCMDS

            .ENDC

            .LONG       $FFFFFFFF               ; terminator


;-------------------------------------------------------------------------------
--
; Resident -- Small piece of code jumped to by init code, sizes back the Mac's memory
for
; the debugger.
;-------------------------------------------------------------------------------
--

Resident
            MOVE        #dSpace-1,D0            ; space for the screen -twm for dbra
            LEA         @0,A0
@0
            .IF         noTerm
               CLR.B        -(A0)
               DBRA         D0, @0
            .ENDC

            MOVE.L      A0,offscreen            ; Install screen buffer
            MOVE.L      A0,$10C                 ; Install new memory limits (BufPtr)

            RTS

;-------------------------------------------------------------------------------
--
;
; Error messages and plain messages
;
;-------------------------------------------------------------------------------
--

Mtext
MTrap       .ASCII      'TRAP'
MBus        .ASCII      'BUS '
MAdd        .ASCII      'ADDR'
MILGL       .ASCII      'ILGL'
MDIVO       .ASCII      'DIVO'
MCHK        .ASCII      'CHK '
MOVFL       .ASCII      'OVFL'
M1111       .ASCII      '1111'
MHuh        .ASCII      '????'

MUserProg   .ASCII      'PRGM AT '
MInRom      .ASCII      'In Rom  '
MUserBrk    .ASCII      'USERBRK '
MmacTrap    .ASCII      'MACTRAP '
MBadHeap    .ASCII      'Heap? @ '
MHeapT      .ASCII      ' HLP PF '
MHeapM      .ASCII      'CNT ### '
Mchksum     .ASCII      'CHKSUM  '
MBreaks     .ASCII      '> 8 BRKS'
Meggs       .ASCII      'SCRAMBLE'
```

```
MBadEgg     .ASCII      'BADSCRAM'

;--------------------------------------------------------------------------------
--
; COLONSP -- string buffer for 'xxxxxxxx: label... '
;--------------------------------------------------------------------------------
--

COLONSP     ;           Len  Address   :     symbolic name   + offset    [four spaces]
            .BYTE       25,0,0,0,0,0,0,58,32,0,0,0,0,0,0,0,0,0,0,0,0,32,32,32,32
            .ALIGN      2

;--------------------------------------------------------------------------------
--
; SYMNAME -- data space for 'trap/routinename'
;--------------------------------------------------------------------------------
--

SYMNAME
            .BYTE       32,32,32,32,32,32,32,32,32,32


; The following commands are in this file
;
;           DM [P1] [P2]            Display memory @P1(locsave) for P2(16)
;           SC                      Stack crawl
;           SM  P1   P2             Set memory location P1 to value P2
;           CL [P1]                 Clear breakpoint P1(all breakpoints)
;           BR [P1]                 Set breakpoint @P1(show current breakpoints)
;           MR [P1]                 Use address located P1(0) bytes down stack as return
address
;

; This is the top of the event loop for the debugger

MSG
            BSR         WriteLine
MACXBUG
            MOVE        #$2700,SR

            LEA         SYSTACK,A7              ; always jam stack pointer
            CLR.B       noRegs                  ; assume rgs will print

            BSR         SWAPOUT

            CLR         TraceSpy                ; shut down if dangling till/spy

            BSR         FIXBUF                  ; prompt the user
            MOVE.B      #'>',(A6)+
            BSR         OUTPUT


            BSR         FIXBUF                  ; get the user's command
            BSR         ReadLine                ; reads into A6+++

MACSBUG2
            CMP.L       A6,A5                   ; any command there?
            BMI.S       decode                  ; skip if so

            MOVE.W      lastCmd,(A6)+           ; repeat previous command
decode
            CMP.L       A6,A5                   ; see if characters
```

```
            BHI.S       what                    ; if none err

            CMPI.B      #$20,(A5)+              ; skip null's and blanks
            BLE.S       decode

            MOVE.B      -(A5),D1                ; pack the two chars into a word
            LSL.W       #8,D1
            MOVE.B      1(A5),D1                ; and save in D1

            MOVE        D1,lastCmd              ; save the command

            LEA         rSYSCMDS,A0
cmdLookup
            MOVE.W      (A0)+,D0
            MOVE.W      (A0)+,D7
            BMI.S       what

            CMPI.B      #'@',D0                 ; see if wild card lookup (D@)
            BNE.S       @1

            MOVE.B      D1,D0                   ; stuff the wild card #
            CMPI.B      #'0',D0
            BMI.S       cmdLookup               ; fail if below range
            CMPI.B      #'8',D0
            BPL.S       cmdLookup               ; fail if above range
@1
            CMP.W       D1,D0                   ; does this command match?
            BNE.S       cmdLookup               ; keep looking

            LEA         SysCmds,A0              ; point to the command list
            JMP         0(A0,D7)                ; dispatch to the command

what
            BSR         FIXBUF
            MOVEQ       #MHuh-MText,D0
            BSR         MFOUR
            BSR         WriteLine
Go1Bug      BRA         Macxbug


SaveDot
            MOVE.L      D0,dotAddress
            RTS


; Debugger debugging-the-debugger commands

            .IF         DDBG
DDCMD
            BSR         FixBuf                  ; print out the two debugging registers
            MOVE.L      DDBG1,D0
            BSR         Pnt8Hx
            MOVE.B      #' ',(A6)+
            MOVE.L      DDBG2,D0
            BSR         Pnt8Hx
            BSR         WriteLine
            BRA         Go1Bug

; set key rollover value

KRCMD
            BSR         ReadXToken
```

```
            MOVE.W      D0,KeyRoll
            BRA         GoIBug

; set key wait value

KWCMD
            BSR         ReadXToken
            MOVE.W      D0,KeyWait
            BRA         GoIBug
            .ENDC

; display memory command

DMCMD
            BSR         ReadLToken              ; get the memory location to display
            BEQ.S       PUTADR                  ; no input value, assume A3/A4 still
good, do it up

            BSR.S       SaveDot                 ; save for dot symbol

            MOVE.L      D0,A3                   ; salt address away for pointers
            MOVE.L      D0,A4

            BSR         ReadToken               ; try for display count
            BNE.S       @0                      ; we got a value
            MOVEQ       #16,D0                  ; default amount
            BRA.S       NoTemplate              ; clear the template ptr, set up end
address

@0          CMP.L       #'IOPB',D0              ; was it a parameter block display
request
            BNE.S       NotIOPB                 ; no, use default template

            LEA         DMiopb,A0               ; set up template ptr

DMsetTemp
            MOVEQ       #0,D0                   ; force display of only one template's
worth of mem
            BRA.S       DMsetEnd                ; and skip the clear

NotIOPB
            CMP.L       #'WIND',D0              ; was it window block display
            BNE.S       NotWIND

            LEA         DMwind,A0               ; set up template
            BRA.S       DMsetTemp

NotWIND
            CMP.L       #'TERC',D0              ; was it TextEdit display?
            BNE.S       NoTemplate              ; nope

            LEA         DMterc,A0               ; set up template ptr
            BRA.S       DMsetTemp

NoTemplate
            MOVEQ       #0,D1                   ; nil template setup
            MOVE.L      D1,A0                   ; clear ptr

DMsetEnd
            ADD.L       D0,A4                   ; set ending address (points one past
last value)
            ST          LocSave                 ; prime for next time
```

```
            MOVE.L      A4,LocSave+2            ; stuff the address

PutAdr
            MOVE.L      A3,DMmemPtr             ; set up memory ptr
            MOVE.L      A4,DMmemEnd             ; set up limits on memory display
            BSR         DisplayMem              ; display the memory in default format
Go2Bug      BRA         Go1Bug                  ; return to main cmd loop

;-------------------------------------------------------------------------------
--
; stack crawl command -- assumes LINK A6 has been religiouly preformed at the beginning
of each
; 'procedure', and an UNLK A6 at the end.  Continues stack crawl until something looks
funny.
;-------------------------------------------------------------------------------
--

SCCMD
            BSR         FixBuf                  ; prime IO buffer
            MOVE.L      RegA6,A2                ; get user's A6

@0          MOVE.L      A2,D0                   ; get the stack frame
            BSR.S       ValidSP                 ; check if it's something OK
            BEQ.S       Go2Bug                  ; no, bail out
            MOVE.L      #'SF @',(A6)+
            BSR         PNT6HX                  ; print out location of stack frame
            MOVE.L      4(A2),D0                ; get the return PC
            BSR.S       ValidPC                 ; is the PC in a valid range?
            BNE.S       @1                      ; yes, keep going

            BSR         WriteLine               ; Nope, write a line
            BRA.S       Go2Bug                  ; and bail out

@1          MOVE.L      #' FR ',(A6)+
            BSR         AdjustPC                ; set D0 to calling address

            MOVE.L      D0,-(SP)                ; save calling address
            BSR         PNT6HX                  ; print out return PC

            MOVE.B      #' ',(A6)+              ; print a space
            MOVE.L      (SP)+,A0                ; set up location to lookup
            MOVE.L      A6,A1                   ; set up IO location
            MOVE.L      A2,-(SP)                ; preserve A2
            BSR         LookupPC                ; try to print out routine name+offset

            MOVE.L      (SP)+,A2                ; restore A2
            MOVE.L      A1,A6                   ; reset A6
            BSR         WriteLine               ; flush everything
            TST.B       AbortPrint              ; did user abort?
            BNE.S       Go2Bug                  ; yes, bail out

            MOVE.L      (A2),A2                 ; follow the link back
            TST.B       LastRoutine             ; was LookupPC called w/PC in last
routine?
            BEQ.S       @0                      ; no, loop

            BRA.S       Go2Bug                  ; yes, bail out

ValidSP
            CMP.L       HeapEnd,D0              ; compare with top of heap
            BLT.S       SetInvalid              ; SP < HeapEnd, invalid
```

```
            CMP.L       BufPtr,D0               ; compare with highest user memory
            BGT.S       SetInvalid              ; SP > max mem value, invalid
            BRA.S       ChkOddD0                ; check for odd SP

ValidPC
            CMP.L       RamBase,D0              ; are we above the globals?
            BLT.S       SetInvalid              ; PC < rambase, out of range

            CMP.L       HeapEnd,D0              ; compare to top of heap
            BGT.S       SetInvalid              ; PC > top of heap, invalid

ChkOddD0
            BTST        #0,D0                   ; is the low bit on?
            BNE.S       SetInvalid              ; yes, odd address, bail out
            MOVEQ       #1,D1                   ; set CC to true
            RTS                                 ; and return

SetInvalid
            MOVEQ       #0,D1                   ; set CC to EQ (out of range)
            RTS

; make D0 = true calling address, given 4(A2) = return address

AdjustPC
            MOVE.L      4(A2),A0                ; get LINK return address
            MOVE.W      MaskBC,D1               ; get the mask ($00FF)
            MOVE.W      #$6100,D2               ; check for BSR.S
            BSR.S       ChkPCInst               ; is it there?

            MOVEQ       #-1,D1                  ; mask nothing
            BSR.S       ChkPCInst               ; check for BSR.L

            ADDQ        #4,A0                   ; move mem ptr to check first previous
word again
            MOVE.W      #$FFF4,D1               ; mask off lower 3 bits
            MOVE.W      #$4E90,D2               ; check for JSR (Ax)
            BSR.S       ChkPCInst               ; is it there?

            MOVE.W      #$FFC0,D1               ; mask off lower 6 bits
            MOVE.W      #$4E80,D2               ; check for JSR Abs.W or JSR d(xx)
            BSR.S       ChkPCInst               ; is it there?

            MOVEQ       #-1,D1                  ; mask nothing
            MOVE.W      #$4EB9,D2               ; check for JSR Abs.L
            BSR.S       ChkPCInst               ; is it there?

            MOVE.L      4(A2),D0                ; I give up, bail out
            RTS

ChkPCInst
            MOVE.W      -(A0),D0                ; get the previous word
            AND.W       D1,D0                   ; mask it
            CMP.W       D2,D0                   ; is there a match?
            BNE.S       @0                      ; no, exit

            ADDQ        #4,SP                   ; pop the return address
            MOVE.L      A0,D0                   ; stuff D0 with calling address
@0          RTS                                 ; and exit from AdjustPC


; Set memory command
```

```
SMCMD
            BSR         ReadXToken
            BEQ         SYNTAX

            BSR.S       SaveDot                 ; save for future reference
            MOVE.L      D0,A1                   ; save the address in A1
SETM1
            BSR         ReadToken               ; get the number
            BEQ         Go2Bug

            MOVE.L      A1,A3
@1
            MOVE.B      D0,-1(A3,D1)
            MOVE.B      -1(A3,D1),D3
            CMP.B       D0,D3
            BNE.S       what                    ; store didn't hold

            ASR.L       #8,D0                   ; get next byte
            ADDQ        #1,A1
            SUBQ.L      #1,D1                   ; more digits?
            BNE.S       @1
            BRA.S       SETM1                   ; any more numbers?

; Set register commands

SETD
            LEA         REGS,A4
            BRA.S       SETR
SETA
            LEA         AREGS,A4
            BRA.S       SETR
SETPC
            LEA         REGPC,A4
            BRA.S       SETR0
SETSR
            LEA         REGSR,A4
            BRA.S       SETR0
SETR
            BSR         GETHEX                  ; get the reg #
            LSL.L       #2,D0                   ; point to correct location
            ADD.L       D0,A4
SETR0
            BSR         ReadXToken
            BEQ.S       SETR4                   ; just print

            MOVE.L      D0,(A4)
Go3Bug      BRA         Go2Bug

SETR4
            BSR         PrintR1                 ; just print the register
            BRA         MSG


            .IF         fullSized
CSCMD
            BSR         ReadXToken              ; get the starting address
            BEQ.S       @1                      ; must want to check checksum

            MOVE.L      D0,A3                   ; set up start/stop address
            MOVE.L      D0,A4
            ADD.W       #15,A4                  ; make chksum be for 16 bytes
```

```
            BSR         ReadToken               ; get amount to checksum
            BEQ.S       @0                      ; use default

            MOVE.L      D0,A4                   ; set stop address

@0          MOVE.L      A3,CSlow                ; salt away in low memory
            MOVE.L      A4,CShigh
            BSR         CalcChecksum            ; calc checksum from A3,A4

            MOVE.L      D0,CSsum                ; salt away calculated value
            BRA         Go3Bug

; user wants to check checksum

@1          BSR         FixBuf
            MOVEQ       #Mchksum-Mtext,D0       ; fill IO buffer with 'CHKSUM '
            BSR         MEight

            MOVE.L      CSlow,A3                ; get saved values, calc checksum
            MOVE.L      CShigh,A4
            BSR         CalcCheckSum
            CMP.L       CSsum,D0
            BEQ.S       CSmatch

; the checksum didn't match, print bad message

            MOVE.L      D0,CSsum                ; first reset the checksum
            MOVE.B      #'F',(A6)+
CSExit      BSR         Writeline               ; print out result
Go3Bug      BRA         Go3Bug

; the checksum matched, print good message

CSmatch
            MOVE.B      #'T',(A6)+
            BRA.S       CSExit
            .ENDC


; Breakpoint clear command

            .IF         fullSized
CLCMD
            BSR         ReadXToken
            BEQ.S       CLAll                   ; just print

            BSR.S       FIXBP                   ; regs setup...
@0
            CMP.L       (A0)+,D0                ; is this the right one??
            BEQ.S       @1
            SUBQ        #1,D7
            BNE.S       @0
            BRA         Go3Bug
@1
            CLR.L       -(A0)                   ; clear the Break point
Go4Bug      BRA         Go3Bug
CLAll
            BSR.S       FIXBP                   ; regs setup...
@0
            CLR.L       (A0)+
            SUBQ        #1,D7
```

```
            BNE.S       @0
Go5Bug      BRA         Go4Bug

            .ENDC                                   ; fullsized

; Breakpoint command

            .IF         fullSized
BRCMD
            BSR         ReadXToken
            BEQ.S       BCMD7

            AND.L       maskBC,D0                   ; drop upper byte

            BSR.S       FIXBP                       ; set up regs
@0
            CMP.L       (A0)+,D0                    ; compare address to table
            BEQ.S       oldBreak
            ADDQ        #4,A2                       ; next one
            SUBQ.L      #1,D7
            BNE.S       @0

; Scan for a new entry

            BSR.S       FIXBP
@1
            TST.L       (A0)+
            BEQ.S       newBreak
            ADDQ        #4,A2
            SUBQ.L      #1,D7
            BNE.S       @1

; Too many break points

            MOVEQ       #MBreaks-MText,D0
            BSR         MEight
            BSR         WriteLine
            BRA.S       Go5Bug

; Returns
;           D7 to 8
;           A0 is break table
;           A2 points to break count

FIXBP
            LEA         BPADD,A0
            MOVEQ       #8,D7
            LEA         BPCNT,A2
            RTS


newBreak
            CLR.L       (A2)
            MOVE.L      D0,-(A0)

; -(A0), A2 points to break point and count entry

oldBreak
            BSR         ReadToken                   ; get the count (zero)
            MOVE.L      D0,(A2)

Go6Bug      BRA         Go5Bug
```

```
BCMD7
            BSR         FIXBUF
            BSR.S       FIXBP
BCMD8
            MOVE.L      (A2)+,D6                ; get BP count
            MOVE.L      (A0)+,D0                ; get BP location
            BEQ.S       @1                      ; no BP there

            MOVE.L      D0,TEMP                 ; save BP loc
            BSR         PNT6HX                  ; print out location

            TST.L       D6                      ; and count to print?
            BLE.S       @9                      ; nope

            MOVE.B      #':',(A6)+
            MOVE.L      D6,D0                   ; get count
            BSR         PNT4HX                  ; print it

@9          MOVE.B      #' ',(A6)+
            MOVEM.L     A0/A2,-(SP)             ; save address regs
            MOVE.L      TEMP,A0                 ; set up PC loc
            MOVE.L      A6,A1                   ; where to print at
            BSR         LookupPC                ; try to print out location info
            MOVE.L      A1,A6                   ; restore IO ptr
            MOVEM.L     (SP)+,A0/A2             ; restore regs

@0
            BSR         WriteLine               ; dump msg
@1
            SUBQ.L      #1,D7
            BNE.S       BCMD8
Go7Bug      BRA         Go6Bug

            .ENDC                               ; fullsized


; magic return command

            .IF         fullSized
MRCMD
            BSR         ReadXToken              ; get offset

            MOVE.L      REGA7,A0                ; get user's stack
            ADD.L       D0,A0                   ; offset by amount
            MOVE.L      (A0),magicPC            ; get return address off stack
            LEA         Magic,A1                ; point return to magic spot
            MOVE.L      A1,(A0)
            MOVE.W      #'T ',lastCmd           ; force previous cmd

            BRA         GOCMD1                  ; go on

            .ENDC                               ; fullsized



            .IF         fullSized

; Exit to shell command
```

```
ESCMD
            SUB.L       A0,A0                   ; get an address
            MOVE.L      A0,REGPC                ; stuff new PC (= 0)
            MOVE        #$A9F4,(A0)             ; stuff exit-to-shell trap @ 0
            BRA         UnStack                 ; let unstack jmp to 0, do trap


; Find command

FindRead
            BSR         ReadToken               ; get start location
            BEQ.S       findIt                  ; dangling A7 is OK
            MOVE.L      D0,-(A4)
            RTS

FCMD
            LEA         findStart,A4            ; point to vars

            BSR         ReadXToken              ; get start location
            BEQ.S       findIt
            MOVE.L      D0,(A4)                 ; do this one special

            BSR.S       FindRead                ; get length

            BSR.S       FindRead                ; get data
            MOVE.L      D1,-(A4)                ; and width

            MOVE.L      minusOne,-4(A4)         ; prime the mask
            BSR.S       FindRead                ; get mask
findIt
            MOVEM.L     findMask,D4-D7/A4       ; load up the regs

            MOVE.L      A4,D2                   ; get limit
            ADD.L       D7,D2                   ; by adding length
finder
            MOVEQ       #0,D3                   ; zero dest reg
            MOVE.L      A4,-(SP)                ; save pointer

            CMP         #2,D5                   ; which width
            BLT.S       findByte
            BEQ.S       findWord

            LEA         Pnt8Hx,A3               ; print longword
            MOVE.B      (A4)+,D3                ; longword case
            LSL.L       #8,D3
            MOVE.B      (A4)+,D3
            BRA.S       fGoWord
findWord
            LEA         Pnt4Hx,A3               ; print word
fGoWord
            LSL.L       #8,D3
            MOVE.B      (A4)+,D3
            BRA.S       fGoByte
findByte
            LEA         Pnt2Hx,A3
fGoByte
            LSL.L       #8,D3
            MOVE.B      (A4)+,D3

            MOVE.L      (SP)+,A4                ; restore pointer
            ADDQ        #1,A4
```

```
findCont
            MOVE.L      D3,D0
            AND.L       D4,D0                   ; mask it
            CMP.L       D6,D0                   ; and cmp to data
            BEQ.S       foundIt

            CMP.L       D2,A4                   ; are we done?
            BLT         finder
findDone
            MOVE.L      A4,findStart            ; remember the limit
Go75Bug     BRA         Go7Bug

foundIt
            BSR         FixBuf
            MOVE.L      A4,D0                   ; found it, print address
            SUBQ        #1,D0                   ; offset back 1 byte
            BSR         SaveDot
            BSR         Pnt8Hx

            MOVE.B      #'/',(A6)+

            MOVE.L      D3,D0                   ; print data
            JSR         (A3)

            BSR         WriteLine

            BRA         findDone

            .ENDC                               ; fullsized


; WH command -- where is the value give located?  If no value, use DOT.
; If Loc < 512, print out trap address for given ROM routine.  If Loc 3 RomBase,
; print out trap number (approx) for given location.  Otherwise, try to print out
; Lisa Pascal/Duvall symbolic location.

            .IF         fullSized
WHCMD
            MOVE.L      $28,-(SP)               ; remember A trap vector
            MOVE.L      SAVEA,$28               ; restore org trap

            BSR         ReadXToken              ; get trap/location
            BEQ.S       WhExit                  ; bail out w/A-trap restore

            BSR         FIXBUF                  ; set up for printing

            TST.W       TrapNum                 ; did user spec a trap name?
            BGE.S       WhInRom2                ; yes, D0 = start of trap routine

            CMP.L       #512,D0                 ; trap address or trap number
            BLT.S       trGetAdd                ; D0 < 512, input trap number

            CMP.L       RomBase,D0              ; address in ROM?
            BGE.S       WhInRom                 ; above ROM start, do rom routine lookup

            CMP.L       TheZone,D0              ; are we really in user space
            BLT.S       WhInRom                 ; D0 < start of app heap, assume it's a
Rom patch

            MOVE.L      D0,-(SP)                ; save D0 on stack
            MOVEQ       #MUserProg-MText,D0
```

```
            BSR         MEight                  ; set up 'PRGM AT '

            MOVE.L      (SP)+,A0                ; set up A0 with address
            MOVE.L      A6,A1                   ; set up output ptr
            BSR         LookupPC                ; print out result
            MOVE.L      A1,A6                   ; update IO ptr

            TST.B       SymFound                ; did we find the routine name?
            BNE.S       @0                      ; yup, normal exit

            MOVE.L      #'$$$$',(A6)+           ; stuff unknown routine value

@0          BRA.S       WhDone                  ; writeln and exit

; D0 = address in ROM or patched routine, get trap number in D3

WhInRom
            BSR.S       MapToTrap               ; make D0 be a trap number
            BRA.S       trGetAdd                ; and continue

WhInRom2
            MOVE.W      TrapNum,D0              ; set up trap number

; D0 = trap number, set dotAddress to trap address

trGetAdd
            MOVE.L      D0,D3                   ; save trap number
            _GetTrapAdd                         ; get the address
            MOVE.L      A0,dotAddress           ; and save it

; dotAddress has address of trap, D3 has trap number.
; Print out D3 as 4 hex digits, dotAddress as 6 hex digits, try to lookup what D3 is

            MOVE.W      D3,D0
            BSR         PNT4HX                  ; print trap number

            MOVE.B      #' ',(A6)+              ; stuff a space

            MOVE.L      dotAddress,D0
            BSR         PNT6HX                  ; print trap address

            .IF         Tnames
            MOVE.B      #' ',(A6)+              ; print out space

            MOVE.W      D3,D0                   ; set up trap number
            MOVE.L      A6,A0                   ; set up output ptr
            BSR         LookupTrap              ; print out trap name
            MOVE.L      A0,A6                   ; restore IO ptr
            .ENDC

WhDone
            BSR         WriteLine

WhExit
            MOVE.L      (SP)+,$28               ; restore A trap vector

Go8Bug      BRA         Go75Bug
            .ENDC                               ; fullsized

; D0.L = address in ROM, return D0.W = trap number
; Trashes D1-D4, A0
```

```
MapToTrap
            MOVE.L      D0,D1                   ; save address
            MOVE.L      #$80000001,D2           ; minimum -distance

            MOVE        #511,D4
@0
            MOVE        D4,D0
            _GetTrapAdd
            CMP.L       A0,D1                   ; compare address to target
            BLT.S       @1                      ; is this one below target?

            SUB.L       D1,A0                   ; calc abs distance
            CMP.L       D2,A0                   ; closer than before
            BLT.S       @1
            MOVE.L      A0,D2
            MOVE        D4,D3                   ; save trap
@1
            SUBQ        #1,D4
            BPL.S       @0

            MOVE        D3,D0                   ; set up trap number
            RTS                                 ; and return

; Register toggle command

            .IF         fullSized
RXCMD
            NOT.B       smallMode               ; flip sense
            BRA         Go8Bug

PXCMD
            NOT.B       showSyms                ; flip sense
            BRA         Go8Bug

; Toggle heap command

HXCMD
            NOT.B       UseSysHeap              ; flip heap
            BRA         Go8Bug

            .ENDC                               ; full sized
```