```
; FILENAME: CMDS.TEXT
;
;
;               Change Log
;
;               16-Aug-84   Added Hide/ShowCursor (can't do A-trap inside debugger)
;               22-Aug-84   Fixed re-entrancy problem (NMI while in debugger)
;               30-Aug-84   Re-displays cursor on RM
;               31-Aug-84   Fixed re-entrancy on bus error in debugger
;                5-Sep-84   Make sure IL is at even address
;                8-Sep-84   Added LookupPC to ONELINE disassembly loop
;               16-Sep-84   Longer ColonSp string, fixed OneLine ptr setup
;               17-Sep-84   Fixed CV display for '<4 ascii chars>'
;               22-Sep-84   Added PC location in ONELINE disassembly
;               23-Sep-84   Bump PC past F-trap instruction on entry
;               24-Sep-84   Added symbolic name loopup for trap names (??? Lisa routine
names later)
;               26-Sep-84   Print termination in disassembly
;               27-Sep-84   Big reg saving in GetSym from parsing cmd line.
;                8-Oct-84   Only lookup trapnames in parsing loop if at least 3 chars long.
;               18-Oct-84   Parsing loop does GetNum if all chars in name are nums or hex.
;                1-Nov-84   TrapNum set to -1 in ReadToken, valid if pos, set in LookupName
;                1-Nov-84   S. Capps special bullet-proofing w/A-trap stuff
;

TDCMD
                BSR.S       TDISPLY                     ; display those regs
                .IF         fullSized
Go9Bug          BRA         Go8Bug
                .ELSE
Go9Bug          BRA         Go3Bug
                .ENDC


TDISPLY
                .IF         withDis
                MOVE.L      REGPC,D0
                BSR         ONELINE
                TST.B       noRegs                      ; should regs be printed?
                BNE.S       TDRTS
                .ENDC
TD00
                BSR         FIXBUF
                MOVE        #'PC',(A6)+                 ; print PC
                LEA         REGPC,A4
                BSR.S       TDREG
                MOVE.W      #'SR',(A6)+                 ; print SR
                MOVE.B      #' ',(A6)+
                ADDQ        #4,A4
                BSR.S       TDREG
                MOVE.W      #'TM',(A6)+                 ; print time in 60ths
                LEA         TICKS,A4
                BSR.S       TDREG
                BSR         WriteLine
                MOVEQ       #'D',D7                     ; print data regs
                LEA         REGS,A3
                BSR         PNTCLS
                MOVEQ       #'A',D7                     ; print address regs
                LEA         AREGS,A3
                BSR         PNTCLS
TDRTS
                RTS
```

```
TDREG
            BSR         PRINTA1                     ; print =<CONTENTS>
            BSR         OUTPUT
            RTS

SAVE
            SF          RUN                         ; twm - at start to hopefully recover
from undebounced switches
            MOVE.L      A7,REGA7
            LEA         REGA7,A7
            MOVEM.L     D0-D7/A0-A6,-(A7)           ; save all regs into the global area
            LEA         SYSTACK,A7                  ; use the stack w/in the global area
            MOVE.L      REGA7,A0
            MOVE.W      (A0)+,REGSR+2
            MOVE.L      (A0)+,REGPC
            .IF         on68000=0                   ; 68010 & 68020
                MOVE.W      (A0)+,REGFMT            ; save the format word
            .ENDC
            MOVE.L      A0,REGA7

            .IF         noTerm=0
                BSR         INITACIA
            .ENDC

            .IF         swapScreen
                .IF         onLisaTrue
                    TRAPTO      _CursorHide             ; hide the cursor
                .ENDC
                BSR         flipSide                    ; flip screens
            .ENDC

            .IF         onLisaTrue                      ; set up keyboard handler
                TRAPTO      _NMISync                    ;tell COPS NMI occured
                TRAPTO      _KeyRoutine                 ;get the key routine
                CMP.L       $196,A0                     ;is it same as low-level
                BEQ.S       @0                          ;  yes, skip setup
                MOVE.L      A0,SaveKeybd                ;  no, save in globals
                MOVE.L      $196,A0                     ;get low-level routine
                TRAPTO      _SetKeyRoutine              ;and set as handler
@0
            .ENDC

            CLR.B       REGPC                       ; clear high order of PC

            LEA         MAXBASE,A0
            ADD.W       TEMP,A0
            JMP         (A0)


UNSAVE
            MOVE.L      A0,TEMP

            .IF         swapScreen
                BSR         flipSide                        ; flip screens
                .IF         onLisaTrue
                    TRAPTO      _CursorDisplay              ; re-display the cursor
                .ENDC
                SF          swapped                         ; re-enable the screen
            .ENDC

            .IF         onLisaTrue
                MOVE.L      SaveKeybd,A0                ; get old keyboard routine
                TRAPTO      _SetKeyRoutine          ; restore it
                TRAPTO      _COPSSynch              ; flush all pending COPS packets
            .ENDC

            LEA         REGS,A7
            MOVEM.L     (A7)+,D0-D7/A0-A6
            MOVE.L      REGA7,A7
            .IF         on68000=0               ; 68010 & 68020
                MOVE.W      REGFMT,-(A7)        ; restore the format word
            .ENDC
            MOVE.L      REGPC,-(A7)
            MOVE.W      REGSR+2,-(A7)
            MOVE.L      TEMP,-(A7)
            ST          RUN
            CLR.L       ReEntrFlg               ; twm - let the debugger be entered
again.  NOTE that
                                                ; there is a window of time where
something could crash it.
            RTS
TRACEOO
            TST.B       TraceGo                 ; check go & step
            BNE.S       goTrace

            .IF         fullSized
            BSR         SWAPOUT                 ; fix up those instructions
            MOVE.L      REGPC,D0                ; see if a break
            BSR         BrkSearch
            BNE.S       @0                      ; not a break point

            SUBQ.L      #1,(A2)                 ; decrement count
            BMI.S       OKTD                    ; break done, cancel tracing

@0
            TST.B       traceTill               ; trace till mode?
            BEQ.S       @2

            MOVE.L      RegPC,D0                ; does PC match?
            CMP.L       tracePC,D0
            BEQ.S       OKTD
@1
            BRA         UnTrace                 ; nope, plow ahead
@2

            TST.B       traceSpy                ; trace spy mode?
            BEQ.S       @3

            BSR         XORMem                  ; get check sum
            CMP.L       sumPlace,D0
            BEQ.S       @1                      ; keep going, it's ok
            BRA.S       OKTD                    ; break

@3
            SUBQ.L      #1,TRACECNT             ; decrement the trace count
            BGT         @1                      ; skip to untrace if not done

OKTD
            .IF         swapScreen
            SF          swapped                 ; NOW, swap the screen
            BSR         FlipSide                ; go swap it
            .ENDC

            CLR.L       TRACECNT                ; stop trace
            MOVE.B      smallMode,noRegs        ; set switch
            BSR         TDISPLY
Go10Bug     BRA         Go9Bug



; This is the trace one time after a go cmd
goTrace
            BCLR        #7,REGSR+2              ; turn off tracing
            CLR.L       TRACECNT
            CLR.B       traceGo
            BSR         SWAPIN                  ; put brk pts back in
            BRA.S       UNSTACK

; Receiver for trace exceptions

TRACE
            MOVE        #$2700,SR
            MOVE.W      #TRACE00-MAXBASE,TEMP
            .IF         swapScreen
            ST          swapped                 ; fake out the screen swap
            .ENDC
go1Save     BRA         SAVE



TCMD
            LEA         magicPC,A2
            TST.L       (A2)                    ; already in T command
            BNE.S       GO4CMD1                 ; if so do a go

            MOVEQ       #1,D0                   ; jam the trace count
            MOVE.L      D0,TRACECNT

            MOVE.L      REGPC,A0                ; see if an A-trap
            MOVE.B      (A0),D0
            LSR.B       #4,D0
            CMP.B       #$A,D0
            BNE.S       UNTRACE                 ; normal trace

            LEA         Magic,A1                ; point to magic place

            MOVE        (A0),D0                 ; get trap
            CMP         #$A9F0,D0               ; load seg?
            BEQ.S       Go10Bug                 ; if so skip to macsbug

            SUB         #$AC00,D0               ; auto pop? and tooltrap ?
            BCS.S       @0

            MOVE.L      REGA7,A0                ; get stack
            MOVE.L      (A0),(A2)               ; get real return in magicPC
            MOVE.L      A1,(A0)                 ; stuff magic address
            BRA.S       GO4CMD1                 ; go on

@0
            MOVE.W      (A0)+,-(A1)             ; copy the trap before magic
            MOVE.L      A0,(A2)                 ; save real return in magicPC
            MOVE.L      A1,REGPC                ; stuff pc with magic address
GO4CMD1     BRA.S       GOCMD1                  ; go on

SCMD
```

```
            BSR         ReadXToken              ; returns zero if none
            BNE.S       @0
            ADDQ.L      #1,D0                   ; stuff in traceCnt
@0
            MOVE.L      D0,TRACECNT
UNTRACE
            .IF         fullSized
            BSR         SWAPSOME
            .ENDC
            ORI.W       #$8000,REGSR+2          ; turn on T tracing
UNSTACK
            LEA         UNSTACK2,A0
            BRA         UNSAVE
UNSTACK2
            TST         ASAVEPC                 ; see if A - tracing is on
            BNE         TA99

            RTE


            .IF         fullSized

GTCMD
            BSR         ReadXToken
            BEQ         Go10Bug


            MOVE.L      D0,BPTILL
            CLR.B       BPTILL                  ; clean up high byte

            BRA.S       GOCMD1
            .ENDC


GCMD
            BSR         ReadXToken
            BEQ.S       GOCMD1

            MOVE.L      D0,REGPC
GOCMD1
            ST          TraceGo                 ; signal one step
            BRA.S       UNTRACE


;-----------------------------------------------------------------------------------
--
; RBCMD -- Re-boot the Mac (or reboot Lisa into the Mac environment)
; by doing a warm restart.
;-----------------------------------------------------------------------------------
--

RBCMD

            .IF         onLisaTrue
;            TRAPTO      _DriverInit             ; reset the driver globals
            TRAPTO      _CursorInit             ; make sure cursor is showing
            .ENDC
                                                ; twm - alter for independent systems
            MOVEQ       #$0A, D0                ; offset from start of ROM
            ADD.L       RomBase, D0             ; use the global value
            MOVE.L      D0, A0
            JMP         (A0)                    ; cold restart (??? or warm 400004)
            .IF         fullSized

SWAPOUT                                         ; swap out brkpts
            TST.B       BPSTATUS
            BEQ.S       SWAPEND
            LEA         SWAPOUT1,A6
            BRA.S       SWAPP
SWAPSOME                                        ; swap in all but PC
            BSR.S       SWAPOUT
            LEA         SWAPSOM1,A6
            BRA.S       SWAPP
SWAPIN                                          ; swap in all brkpts
            BSR.S       SWAPOUT
            LEA         SWAPIN1,A6
SWAPP
            BSR         FIXBP
            LEA         BPDATA,A3
SWAP1
            MOVE.L      (A0),A4
            TST.L       (A0)+
            BEQ.S       SWAP99
            JMP         (A6)

SWAPSOM1
            MOVE.W      (A4),(A3)
            CMP.L       REGPC,A4
            BEQ.S       SWAP99
SWAPIN1
            MOVE.W      (A4),(A3)
            MOVE.W      #$4E4F,(A4)             ; breakpoint
            ST          BPSTATUS
            BRA.S       SWAP99
SWAPOUT1
            MOVE.W      (A3),(A4)
            CLR.W       BPSTATUS
SWAP99
            ADDQ        #2,A3
            SUBQ.L      #1,D7
            BPL.S       SWAP1
SWAPEND
            RTS


STCMD
            BSR         ReadXToken              ; get the trace till address

            MOVE.L      D0,tracePC
            CLR.B       tracePC
            ST          traceTill
traceUn
            ADDQ.L      #1,tracecnt             ; non zero
            BRA         UNTRACE

SSCMD
            BSR.S       GetCSum                ; read in high/low limits for checksum,
save in sumplace
            ST          traceSpy
            BRA         traceUn

GetCSum
```

```
            BSR         ReadXToken              ; get the trace spy addresses
            MOVE.L      D0,ALowPC               ; get lower limit PC
            MOVE.L      ALowPC,AHighPC

            BSR         ReadToken
            BEQ.S       @0
            MOVE.L      D0,AHighPC              ; get upper limit PC
@0
            BSR         XORMem
            MOVE.L      D0,sumPlace             ; save current checksum
            RTS                                 ; and return

; D0 contains an address
; returns A0/A2 pointing to break point table of break if found plus
; CC's set EQ if it was a break.  Checks NINE breaks (GT one also)

BrkSearch
            BSR         FIXBP
@0
            CMP.L       (A0),D0
            BEQ.S       @1
            ADDQ        #4,A0
            ADDQ        #4,A2
            SUBQ.L      #1,D7
            BPL.S       @0
@1
            RTS

CHKBP00
            MOVE.L      REGPC,D0
            SUBQ.L      #2,D0
            BSR         BrkSearch
            BNE.S       Go11Bug

            SUBQ.L      #1,(A2)                 ; and decrement it
            BLE.S       CHKBP3
            BSR.S       CHKBP4
            BRA         GOCMD1
CHKBP3
            BSR.S       CHKBP4
            CLR.L       BPTILL                  ; clear out go till
Go11Bug     BRA         Go10Bug
CHKBP4
            BSR         SWAPOUT
            SUBQ.L      #2,REGPC
            BSR         TDISPLY
            RTS

CHKBP
            MOVE        #$2700,SR
            MOVE.W      #CHKBP00-MAXBASE,TEMP
go2Save     BRA         go1Save

            .ENDC

ABEnd
            CLR.L       magicPC                 ; clear out MR
            MOVE.B      smallMode,noRegs        ; set switch
            BRA.S       AbortB0
MSGTD
            BSR         WriteLine
ABORTB0
```

```
            BSR         TDISPLY
            .IF         fullSized
Go12Bug     BRA         Go11Bug
            .ELSE
Go12Bug     BRA         Go10Bug
            .ENDC
ABORTB
            .IF         onMacTrue
                MOVE        #$2700,SR               ; twm - already true on Vacc & XL (***
machine dependent ***)
            .ENDC
            ADDQ.L      #1, ReEntrFlg               ; twm - only 1 instruction allowed
before another NMI
            CMP.L       #1, ReEntrFlg               ; first time ReEntrFlg will be = 1
            BEQ.S       @1                          ; continue on first time through this
routine
            SUBQ.L      #1, ReEntrFlg               ; get rid of the count for this re-entry
            RTE                                     ; cleanup the stack and let the debugger
continue
                                                    ; twm
@1          MOVE.W      #ABORTBO-MAXBASE,TEMP

            .IF         fullSized
go3Save     BRA         go2Save
            .ELSE
go3Save     BRA         Save
            .ENDC

ABORTEO
            MOVEQ       #MTRAP-MText,D0
            BSR         MERROR
            BRA         MSG

ABORTE
            MOVE        #$2700,SR
            MOVE.W      #ABORTEO-MAXBASE,TEMP
go4Save     BRA         go3Save


BUSERRO
            MOVEQ       #MBus-MText,D0
            BSR         MERROR

PRNTADDR
            MOVE.L      WORK1,D0
            BSR         PNT8HX
            BRA         MSGTD

BEStatReg   .EQU    0               ; saved status register
BEPCHigh    .EQU    2               ; program counter High byte
BEPCLow     .EQU    4               ; and Low byte
BEOffset    .EQU    6               ; Vector offset
BESSW       .EQU    8               ; Special Status Word
BEFaultH    .EQU    10              ; Fault Address High byte
BEFaultL    .EQU    12              ; and Low byte
BEUU1       .EQU    14              ; Not used
BEDIB       .EQU    16              ; Data Output Buffer
BEUU2       .EQU    18              ; Not used
BEDOB       .EQU    20              ; Data Input Buffer
BEUU3       .EQU    22              ; Not used
BEIIB       .EQU    24              ; Instruction Input Buffer
```

```
BUSERR
            MOVE        #$2700,SR
.IF         onVaccTrue
;
; Now test for access to 40xxxx from dumb Mac programs
;
            CMP.W   #$0040, BEFaultH(SP); see if accessing MAC ROM image
            BNE.S   ReallyErr           ; if not '0040' then go to real error handler
@1          BTST    #8, BESSW(SP)       ; test for Read/~Write
            BEQ.S   ReallyErr           ; report all writes to the ROM

            MOVE.B  #$09, BEFaultH+1(SP); point to current Rom image - What a Kludge
*****
            MOVE.L  A0, -(SP)           ; save A0, note *** equ are off by 4 ***
            MOVE.L  BEFaultH+4(SP), A0  ; get the full address

            BTST    #9, BESSW+4(SP)     ; test for Byte/~Word
            BNE.S   @3                  ; handle the byte read
            MOVE.W  (A0), BEDIB+4(SP)   ; fetch the word at that location
            BTST    #13, BESSW+4(SP)    ; is it an instruction fetch?
            BEQ.S   @5                  ; no, so cleanup
            MOVE.W  (A0), BEDIB+4(SP)   ; fetch the word at that location
            BRA.S   @5
@3          BTST    #10, BESSW+4(SP)    ; test for high/~low byte
            BNE.S   @4                  ; bit was set so go fetch the high byte
            ADDQ.L  #1, A0              ; increment to the least significant byte
location
            MOVE.B  (A0), BEDIB+5(SP)   ; and put it at the proper offset
            BRA.S   @5
@4          MOVE.B  (A0), BEDIB+4(SP)   ; fetch the most significant byte

@5          MOVE.L  (SP)+, A0           ; restore A0
            MOVE.B  #$40, BEFaultH+1(SP) ; restore original value
            BSET    #15, BESSW(SP)      ; tell 68010 that it's been handled
            RTE
.ENDC

mgcSR       .equ    50                  ; offset to move SR to overwrite bus & address
error stack
mgcPC       .equ    54                  ; offset to move PC for overwrite

            .IF         on68000                     ; Mac & XL
                TST.W       (A7)+                   ; function code
                MOVE.L      (A7)+,WORK1             ; access address
                TST.W       (A7)+                   ; instruction
            .ELSE                                   ; 68010 has a different stack
                MOVE.L      BEFaultH(SP),WORK1      ; access address
                MOVE.L      (SP), mgcSR(SP)         ; move SR & top half of PC
                MOVE.L      4(SP), mgcPC(SP)        ; move bottom half of PC & Format
word
                ADDA.L      #mgcSR, SP              ; now point SP at new SR location
            .ENDC
            TST.B       RUN                     ; are we re-entrant
            BEQ.S       BUSERR0                 ; don't save regs, re-entrancy


ReallyErr   MOVE.W      #BUSERR0-MAXBASE,TEMP
go5Save     BRA         go4Save


ADDRERR0
            MOVEQ       #MAdd-MText,D0
            BSR         MERROR

            BRA         PRNTADDR

ADDRERR
            MOVE        #$2700,SR
            .IF         on68000                      ; Mac & XL
                TST.W       (A7)+                    ; function code
                MOVE.L      (A7)+,WORK1              ; access address
                TST.W       (A7)+                    ; instruction
            .ELSE                                    ; 68010 has a different stack
                MOVE.L      BEFaultH(A7),WORK1       ; access address
                MOVE.L      (SP), mgcSR(SP)          ; move SR & top half of PC
                MOVE.L      4(SP), mgcPC(SP)         ; move bottom half of PC & Format
word
                ADDA.L      #mgcSR, SP               ; now point SP at new SR location
            .ENDC
            TST.B       RUN
            BEQ.S       ADDRERR0                 ; no reg saving (re-entrancy)
            MOVE.W      #ADDRERR0-MAXBASE,TEMP
go6Save     BRA         go5Save


ILLEGAL0
            MOVEQ       #MILGL-MText,D0
MStd
            BSR         MERROR
            BRA         MSG

ILLEGAL
            MOVE        #$2700,SR
            TST.B       RUN
            BEQ.S       ILLEGAL0
            MOVE.W      #ILLEGAL0-MAXBASE,TEMP
go7Save     BRA         go6Save


DIVZR00
            MOVEQ       #MDiv0-MText,D0
            BSR         MERROR
            BRA         MSGTD

DIVZR0
            MOVE        #$2700,SR
            MOVE.W      #DIVZR00-MAXBASE,TEMP
go8Save     BRA         go7Save


CHKINST0
            MOVEQ       #MChk-MText,D0
            BRA         MStd

CHKINST
            MOVE        #$2700,SR
            MOVE.W      #CHKINST0-MAXBASE,TEMP
go9Save     BRA         go8Save


OVRFLW0
            MOVEQ       #MOvFl-MText,D0
            BRA         MStd

OVRFLW
```

```
            MOVE        #$2700,SR
            MOVE.W      #OVRFLWO-MAXBASE,TEMP
go10Save    BRA         go9Save

LN11110
            MOVEQ       #M1111-MText,D0
            BRA         MStd

LN1111
            MOVE        #$2700,SR                ; mask interrupts, turn on supervisor
bit
            ADDQ.L      #2,2(SP)                 ; bump PC past $Fxxx word

            MOVE.W      #LN11110-MAXBASE,TEMP
go11Save    BRA         go10Save


            .word       0                        ; space for trap copy
Magic
            MOVE        SR,magicSR               ; preserve cc's
            .IF         on68000=0                  ; on a 68010
                MOVE.W      #$0,-(SP)            ; use a fake format word
            .ENDC
            MOVE.L      magicPC,-(SP)            ; fake an exception
            MOVE        magicSR,-(SP)
            MOVE        #$2700,SR
            MOVE.W      #ABEnd-MAXBASE,TEMP      ; just wake up
go12Save    BRA         go11Save


; D0 contains ASCII digit -> D0 number

GETHEX
            ANDI.L      #$FF,D0
            CMPI.B      #$30,D0
            BLT         SYNTAX
            CMPI.B      #$39,D0
            BGT.S       @1
@0
            ANDI        #$F,D0
            RTS
@1
            SUBQ.B      #7,D0                    ; drop A to :
            CMPI.B      #$3F,D0
            BLE.S       @0

SYNTAX
            BSR         FIXBUF
            MOVEQ       #MHuh-MText,D0
            BRA         MStd


; Convert command

            .IF         fullSized
CVCMD
            BSR         ReadXToken

            MOVE.L      D0,D7

            BSR         FIXBUF
            MOVE.W      #' $',(A6)+
            BSR         PNT8HX                  ; print hex

            MOVE.W      #'  ',(A6)+
            MOVE.L      D7,D0
            BSR         PNTZHX                  ; print signed

            MOVE.B      #' ',(A6)+              ; print decimal
            MOVE.B      #'&',(A6)+
            MOVE.L      D7,D0
            BSR         HEX2DEC

            MOVE.B      #' ',(A6)+              ; print as characters
            MOVE.B      #$27,(A6)+              ; print '

            MOVE.L      D7,D0                   ; set up for print
            MOVEQ       #3,D3                   ; print 4 bytes as ascii

@0          ROL.L       #8,D0                   ; shuffle around next byte (char)
            BSR         Bin2Char                ; print the char
            DBRA        D3,@0                   ; and loop

            MOVE.B      #$27,(A6)+              ; print finishing '
            BRA         MSG                     ; and return



HEX2DEC
            MOVE.L      D0,D4
            BPL.S       @0
            NEG.L       D4
            MOVE.B      #'-',(A6)+
@0
            CLR.B       TEMP
            MOVEQ       #$A,D6
HX2DC0
            MOVEQ       #1,D2
            MOVE.L      D6,D1
            SUBQ.L      #1,D1
            BEQ.S       HX2DC2
HX2DC1
            MOVE.W      D2,D3                   ; 32 bit multiply
            MULU        #$A,D3
            SWAP        D2
            MULU        #$A,D2
            SWAP        D3
            ADD.W       D3,D2
            SWAP        D2
            SWAP        D3
            MOVE.W      D3,D2
            SUBQ.L      #1,D1
            BNE.S       HX2DC1
HX2DC2
            CLR.L       D0
HX2DC22
            CMP.L       D2,D4
            BLT.S       HX2DC3
            ADDQ.L      #1,D0
            SUB.L       D2,D4
            BRA.S       HX2DC22
```

```
HX2DC3
            TST.B       D0
            BNE.S       HX2DC4
            TST.B       TEMP
            BEQ.S       HX2DC5
HX2DC4
            ADDI.B      #$30,D0
            MOVE.B      D0,(A6)+
            MOVE.B      D0,TEMP
HX2DC5
            SUBQ.L      #1,D6
            BNE.S       HX2DC0
            TST.B       TEMP
            BEQ.S       HX2DC6
            RTS
HX2DC6
            MOVE.B      #'0',(A6)+
            RTS

            .ENDC                               ; fullsized


; ReadToken
; ReadXToken like ReadToken, but scans to blank first
; READLToken calls ReadXToken, if no #, uses locsave
; Pops stuff off the input buffer and returns 2 values:
; D0 - contains the resultant number
; D1 - contains the # of digits in the number (0->no number)
;

ReadLToken
            BSR.S       ReadXToken              ; to get location
            BNE.S       @0

            TST         LOCSAVE                 ; no parameter try LOCSAVE
            BEQ.S       @0
            MOVE.L      LOCSAVE+2,D0            ; use saved location
@0
            RTS

ReadXToken
            CMP.L       A6,A5                   ; any more chars?
            BGE.S       @0                      ; if not escape

            MOVE.B      (A5)+,D0                ; get next char
            CMPI.B      #' ',D0                 ; scan to blank
            BNE.S       ReadXToken
@0

ReadToken
            MOVEM.L     D4-D6,-(SP)             ; save regs
            MOVEQ       #-1,D0                  ;
            MOVE.W      D0,TrapNum              ; no trap names yet (also cleared in
LookupName)

            MOVEQ       #0,D4                   ; clear accumulator
            MOVEQ       #0,D6                   ; max # digits

ReadMore
            MOVEQ       #0,D5                   ; count # digits

            CLR.B       SIGN                    ; assume positive
            MOVEQ       #0,D3                       ; assume no indirection
            MOVEQ       #0,D0
            MOVEQ       #0,D1                       ; sub-number built here
            MOVE        #16,Base                    ; assume base 16
blanks
            CMP.L       A6,A5                       ; any more chars?
            BGE         ReadExit                    ; if not escape
leading
            MOVE.B      (A5)+,D0                    ; get next char
            CMPI.B      #' ',D0                     ; skip blanks
            BLE.S       blanks

; See if leading sign or indirection

            CMP.B       #'@',D0                     ; leading @ indirection
            BNE.S       @0
            ADDQ        #1,D3                       ; bump indirection counter
            BRA.S       leading
@0
            CMP.B       #'+',D0                     ; leading plus
            BEQ.S       leading

            CMP.B       #'-',D0                     ; leading minus
            BNE.S       getBase
            NOT.B       Sign                        ; record sign change
            BRA.S       leading
getBase
            CMP.B       #$27,D0                     ; leading '
            BEQ         getString

            CMP.B       #'$',D0                     ; leading $
            BEQ.S       @1

            CMP.B       #'&',D0                     ; leading &
            BNE         getLabel
            MOVE        #10,Base
@1
            MOVE.B      (A5)+,D0                    ; next character
getNumber

            BSR         GetHex                      ; D0-ascii==>D0-digit
            CMP.W       BASE,D0                     ; > base?
            BHI.S       getError

            ADDQ        #1,D5                       ; increment # digits

            MOVE.L      D1,D2                       ; save reg
            SWAP        D1                          ; multiply high half
            MULU        BASE,D1
            SWAP        D1
            TST.W       D1                          ; look for overflow
            BNE.S       getError

            MULU        BASE,D2                     ; do low half
            ADD.L       D2,D1
            BVS.S       getError                    ; overflow??

            ADD.L       D0,D1                       ; add in this digit
            BVS.S       getError                    ; overflow?

; Are we done?
```

```
            CMP.L       A6,A5                   ; any more chars?
            BGE.S       ReadExit                ; if not escape

            MOVE.B      (A5)+,D0                ; get next char

            CMPI.B      #'0',D0                 ; < ASCII zero
            BGE.S       GetNumber               ; go get more
ReadExit
            TST         D3                      ; any indirection?
            BEQ.S       @1
            MOVEQ       #8,D5                   ; jam eight digits
@0
            MOVE.L      D1,A0                   ; indirect it
            MOVE.L      (A0),D1
            SUBQ        #1,D3
            BNE.S       @0
@1

            TST.B       Sign                    ; negative?
            BEQ.S       @2
            NEG.L       D1
@2
            ADD.L       D1,D4                   ; add into result

; Clean up the bytes counter

            ADDQ        #1,D5                   ; round up
            LSR         #1,D5                   ; = number of bytes
            CMP         #4,D5
            BLE.S       @3
            MOVEQ       #4,D5                   ; jam to a long
@3

            CMP         D6,D5                   ; maximize # digits
            BLE.S       @4
            MOVE        D5,D6                   ; max # digits
@4

            CMPI.B      #'+',D0                 ; if plus or minus add a new one
            BEQ.S       @5
            CMPI.B      #'-',D0
            BNE.S       @6
@5
            MOVE.B      D0,-(A5)                ; push back on sign
            BRA         ReadMore
@6
            MOVE.L      D4,D0                   ; return the number
            MOVE.L      D6,D1                   ; return max digits

            MOVEM.L     (SP)+,D4-D6             ; restore regs

            TST         D1                      ; return CC's set to # digits
            RTS

getError
            BRA         what

getString
            MOVEQ       #1,D2                   ; amount to clean up

            CMP.L       A6,A5                   ; any more chars?
            BGE.S       cleanExit               ; if not escape
            MOVE.B      (A5)+,D0                ; get char
            CMP.B       #$27,D0                 ; final quote
            BEQ.S       cleanExit               ; if not escape

            CMP         #8,D5                   ; more to go?
            BGE.S       getString

            ADDQ        #2,D5                   ; two digits at a time

            LSL.L       #8,D1                   ; swap in new char
            MOVE.B      D0,D1

            BRA         getString

; See if the text matches a label

getLabel
            CMP.B       #'.',D0                 ; dot?
            BNE.S       notDot

            MOVE.L      dotAddress,D1
            MOVEQ       #1,D2                   ; amount to skip
goLabel
            MOVEQ       #8,D5                   ; all eight bytes
cleanExit
            MOVE.B      (A5)+,D0
            SUBQ        #1,D2
            BNE         cleanExit

            BRA         ReadExit

notDot
            MOVE        D0,D2                   ; build a word
            LSL         #8,D2
            MOVE.B      (A5),D2                 ; get second byte

            CMP         #'PC',D2                ; PC?
            BNE.S       notPC

            MOVE.L      REGPC,D1                ; return the PC
go2Label
            MOVEQ       #2,D2                   ; amount to skip
            BRA         goLabel
notPC
            CMP         #'TP',D2                ; The port?
            BNE.S       notTP

            MOVE.L      REGA7-8,A0              ; get A5
            MOVE.L      (A0),A0                 ; get the grafglobals
            MOVE.L      (A0),D1                 ; and thePort

            BRA         go2Label
notTP
            CMP.B       #'R',D0                 ; reg references start with R
            BNE.S       getSym                  ; try for hex number

            CMP.B       #'A',D2                 ; aregs
            BNE.S       notAs

            LEA         AREGS,A0                ; point to address regs loc
doReg
```

```
            MOVEQ       #0,D2                     ; calculate the index
            MOVE.B      1(A5),D2
            SUB         #$30,D2
            BMI.S       getSym                    ; RAx, x < '0'

            CMP         #7,D2
            BGT.S       getSym                    ; RAx, x > '7'

            LSL         #2,D2
            MOVE.L      0(A0,D2),D1

            MOVEQ       #3,D2                     ; amount to skip
            BRA         goLabel

notAs
            CMP.B       #'D',D2                   ; Dregs
            BNE.S       getSym                    ; Rx, x <> A, x <> D

            LEA         REGS,A0                   ; point to data regs loc
            BRA         doReg

; Try to look up a value for the name.  We enter here not knowing if what follows is a
number,
; trap name, or Pascal routine name, so must be able to back out.  Remember that almost
all
; regs are used in the above parsing loop (restore ad infinitum).

GetSym
            .IF         Tnames=0
            BRA         getNumber                 ; try for number

            .ELSE
            MOVEM.L     D0-D2/A1-A2,-(SP)         ; save off regs I use (if adj., set pop
value below)

            MOVE.L      A5,A0                     ; get current input ptr
            SUBQ.L      #1,A0                     ; pt it to first char

            MOVEQ       #0,D0                     ; prime char count
            MOVEQ       #0,D2                     ; prime number count

@0          MOVE.B      (A0)+,D1                  ; get a byte
            ADDQ        #1,D0                     ; bump char count

            CMP.B       #'A',D1
            BLT.S       @3                        ; char < 'A', can't be an alpha

            CMP.B       #'F',D1                   ; are we out of the hex char range?
            BGT.S       @1                        ; yup

            ADDQ        #1,D2                     ; bump numeric count

@1          CMP.B       #'Z',D1                   ; is it really an alpha char?
            BLE.S       @0                        ; yup, keep looping

@2          SUBQ        #1,D0                     ; back the count down one
            BRA.S       EndGetSym                 ; and bail out of parsing

@3          CMP.B       #'0',D1                   ; is it a char?
            BLT.S       @2                        ; no, bail out of parsing

            CMP.B       #'9',D1                   ; char > '9'?
            BGT.S       @2                       ; yup, not a number

            ADDQ        #1,D2                    ; a number, bump count
            BRA.S       @0                       ; and keep looping

; A0 now pts one past end of string, D0 has count.  D2 = number of numeric chars, will
; be <= D0.  If same, we must have a number, or D0 & D2 = 0 and also not a symbol

EndGetSym
            CMP.B       D2,D0                    ; as many nums as chars, or both = 0?
            BNE.S       GotAname                 ; nope, process the symbol

; Here we should look for a routine name (if symbols are enabled).  Also branch here if
the
; trap name search doesn't find anything.  For now, bail out.

getSymExit
            MOVEM.L     (SP)+,D0-D2/A1-A2        ; restore D0-D2/A1-A2
            BRA.S       getNumber                ; and continue parsing

GotAname
            SUB.L       D0,A0                    ; move A0 back to 1 past 1st char
            SUBQ.L      #1,A0                    ; now A0 pts to first char

            BSR         LookupName               ; try to find a value for the name
            BEQ.S       getSymExit               ; no value, assume a number

            MOVE.L      D0,D2                    ; set amount of input line to skip

            ADD.W       #20,SP                   ; pop five regs off stack (pop value)

            BRA         GoLabel                  ; D1 already has long value, D2 has bump
amount
            .ENDC


            .IF         withDis


;       immediate disassemble (n lines)
;

ILCMD
            BSR         ReadLToken               ; to get location
            BSR.S       SaveDot                  ; D0 has address
            BCLR        #0,D0                    ; make sure it's even

            MOVE.W      #NumIL,-(SP)             ; init counter to # of lines

            MOVE.L      D0,-(SP)                 ; save address
            BSR         ReadToken                ; see if # lines
            BEQ.S       @2
            MOVE        D0,4(SP)                 ; jam # lines

@2          MOVE.L      (SP)+,D0
            ST          ShowPC                   ; display the PC

@3          BSR.S       ONELINE                  ; disassemble and print next line
            TST.B       AbortPrint               ; did user abort output during IL?
            BEQ.S       @4                       ; no, keep going

            BSR         WriteLine                ; flush any unfinished lines
            BRA.S       @5                      ; and exit

@4          SUB.W       #1,(SP)
            BNE.S       @3                      ; loop until zero

@5          TST.W       (SP)+                   ; delete counter
            SF          ShowPC                  ; and reset the flag

Go13Bug     BRA         Go12Bug

;
;       immediate disassemble (1 line)
;
IDCMD
            BSR         ReadLToken              ; to get location

            BSR.S       SaveDot                 ; D0 has address
            BSR.S       ONELINE
@3
            BRA         Go13Bug

;
;       SETUP56 -- A5 = ptr to pascal string, makes A5 pt to first char,
;                  A6 pt to one past last one.

SETUP56
            MOVE.L      A5,A6
            CLR.W       D0
            MOVE.B      (A5),D0                 ; get string length
            ADDQ        #1,D0
            ADD.W       D0,A6                   ; A6 at end
            ADDQ        #1,A5                   ; A5 at start
            RTS

;-----------------------------------------------------------------------------
--
; Routine Name          OneLine
;
; Registers             D0 (input)              ; location to disassemble at.
;                       D0 (output)             ; next location to disassemble
;
; Function              Disassemble and print one line at D0
;-----------------------------------------------------------------------------
--

ONELINE
            BCLR        #0,D0                   ; make sure PC is even
            MOVE.L      D0,-(SP)                ; push location
            LEA         COLONSP,A6
            ADDQ        #1,A6                   ; A6 now points to first char of ColonSP
string
            BSR         PNT6HX                  ; stuff location at start of string
            LEA         COLONSP,A0
            ADD.L       #9,A0                   ; bump string position to imm past ': '
            MOVE.L      A0,-(SP)                ; save on stack

            MOVEQ       #17,D0                  ; set up for spaces loop
BLANKFL
            MOVE.B      #' ',(A0)+              ; print out 17 spaces (eg.
'SetupMem+0204    ')
            SUBQ        #1,D0
            BNE.S       BLANKFL
            MOVE.L      (SP)+,A1                ; set up where to print location out
            MOVE.L      (SP),A0                 ; get location
            TST.B       ShowPC                  ; do we print out the PC?
            BEQ.S       @0                      ; no, skip next part
            CMP.L       regPC,A0                ; is this location the PC
            BNE.S       @0                      ; no, keep going
            MOVE.B      #'P',14(A1)             ; stuff 'PC' label
            MOVE.B      #'C',15(A1)

@0          BSR         LookupPC                ; try to fill the label
            LEA         COLONSP,A5
            BSR         SETUP56
            BSR         OUTPUT                  ; to print '<address>: <label>+<offset>'

            MOVE.L      (SP),A3                 ; A3 = PC
            CMP.L       #$00004E56,(A3)         ; is PC = $0000 LINK A6,xxxx?
            BNE.S       OLdisAsm                ; nope, normal disassembly

            MOVEQ       #Mhuh-MText,D0          ; unknown word symbol
            BSR         MFOUR
            BSR         WriteLine               ; flush it

            MOVE.L      (SP)+,A5                ; get PC location
            ADDQ        #2,A5                   ; bump by two (past null)

OLsetLoc
            LEA         LOCSAVE,A0              ; set flag and save location
            MOVE.W      #1,(A0)+                ; set true flag
            MOVE.L      A5,(A0)+                ; stuff new PC location
            MOVE.L      A5,D0                   ; set up D0 for next pass through
OneLine
            RTS                                 ; and return

OLdisAsm
            MOVE.L      (SP)+,A5                ; pop location into A5
            LEA         OPCOD,A4                ; setup A3, A4 and A6
            LEA         OPERAND,A3

B
            LEA         B,A6                    ; base pointer for XJWP macro in
disassembler
            BSR         DISASM
            BSR         OLsetLoc                ; set up the saved location/flag

; concat spaces to opcode until length = 8

            LEA         OPCOD,A5
@1
            MOVE.L      A5,A6
            CLR.W       D0
            MOVE.B      (A5),D0                 ; get length
            CMP.W       #8,D0                   ; done?
            BGE.S       @2                      ; yup
            ADDQ        #1,D0                   ; bump length
            ADD.W       D0,A6                   ; A6 = next space to fill
            MOVE.B      #32,(A6)                ; write the space out
            ADDQ.B      #1,(A5)                 ; bump length
            BRA.S       @1
@2
            LEA         OPCOD,A5
            BSR         SETUP56                 ; get A5/A6 pointing correctly
            BSR         OUTPUT                      ; print opcode

            LEA         OPERAND,A5
            BSR         SETUP56
            BSR         WriteLine                   ; print operand
            LEA         LOCSAVE,A0
            MOVE.L      2(A0),D0                    ; leave next location in D0
            RTS                                     ; for disassembly of next line

            .ENDC


; This is the off screen buffer if that feature is enabled

            .IF         swapScreen
flipSide
            TST.B       swapped
            BNE.S       @1                          ; skip if disabled

            MOVEM.L     A0-A1/D0-D1,-(SP)

            MOVE.L      SCRNBASE,A1                 ; point to the screen
            ADD         ScreenVars+4,A1             ; offset down
            MOVE.L      offScreen,A0

            MOVE        #dSpace/4,D0              ; do them longs at a time
@0
            MOVE.L      (A0),D1                     ; get source
            MOVE.L      (A1),(A0)+
            MOVE.L      D1,(A1)+

            SUBQ        #1,D0
            BNE.S       @0


            MOVEM.L     (SP)+,A0-A1/D0-D1
@1
            RTS


            .ENDC
```