```
; Filename: IO.TEXT  (MACSBUG I/O ROUTINES)
;
;           Modification History
;
;           20-Aug-84   Added swap flag set in flipping code
;           23-Aug-84   Unfixed swap flag, added flush of keyup (Lisa)
;           27-Aug-84   Further fix of keyup flushing, almost works
;           10-Sep-84   Added keyboard delay
;           23-Sep-84   OUTPUT tests for abort of printing
;           18-Oct-84   Terminal based debuggers also do test (Ctrl C) for abort of
printing
;            8-Dec-84   Added dynamic setup of PostEvent trap location (moved from Init)
;            9-Feb-85   twm - added SCCDelay macro and added SCC init table for a Yacc

.MACRO      SCCDelay                            ; twm -- add a macro so different
hardware is easier to hide
    .IF     onYaccTrue                          ; Delay for a Yacc of 1.728 us
            MOVE.W      (SP), (SP)              ; 12 * 108 ns = 1296ns
            MOVE.L      SP, SP                  ; 4 * 108 ns = 432ns
    .ELSE                                       ; **** assume sane delay for Macintosh &
XL
            MOVE.W      (SP), (SP)              ; ??? for actual timing delay
    .ENDC
.ENDM

            .IF     noTerm

KbdTalked   .EQU    KbdVars                     ; $FF    after keyboard shifts
                                                ;        something in

ReadLine
            BSR         StdEntry
            MOVEQ       #64,D7                  ; ticks counter force immed.

READBUF
 @0                                              ;loop here until char <> tilda
            BSR         INCHNE
            BEQ.S       READBUF

            .IF         swapScreen
            CMP         #$60,D0                 ; swap char (tilda)
            BNE.S       @2
            BSR         FlipSide
@1
            BSR         INCHNE
            BEQ.S       @1
            BSR         FlipSide
            CMP         #$60,D0                 ; swap char
            BEQ.S       @0
@2
            .ENDC

; we have a valid key value in D0 now, so deal with it

DealWkey
            CMPI.B      #8,D0                   ; BACKSPACE?
            BNE.S       @0                      ; no, keep going
            CMP.L       A6,A5                   ; ANY CHARS TO DELETE?
            BHS.S       READBUF                 ; NO, IGNORE BACKSPACE
            SUBQ.L      #1,A6                   ; Yes, remove char from buffer
@0
            MOVE        D0,-(SP)                ; save char
```

```
            BSR         OUTCH                   ; print it out
            MOVE        (SP)+,D0

            CMPI.B      #8,D0                   ; BACKSPACE?
            BEQ.S       READBUF                 ; YES, SKIP

            CMPI.B      #$D,D0                  ; Is it a Carriage Return?
            BNE.S       keepIt                  ; no, stuff into IO buffer

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

            BSR.S       UpLoop                  ; wait for the key-up event


StdExit
; restore post event stuff
            MOVE.L      (SP)+,RamBase           ; save ram base

            MOVE.L      postEvent,A0            ; restore to original postevent
            MOVEQ       #$2F,D0
            _SetTrapAddress

            MOVE.L      (SP)+,$28               ; restore Atrap vector

            MOVEM.L     (SP)+,D0-D7/A0-A3       ; restore registers
            RTS

upLoop
            .IF         onLisaTrue               ; wait for all keys to be up (kwk)
            TRAPTO      _POLL                   ; get pending key events
            CLR.W       D0                      ; set up for keycode
            MOVE.B      #$48,D0                 ; RETURN keycode = $48
            TRAPTO      _KeyIsDown              ; test the key
            TST.W       D1                      ; is the return key still down?
            BNE.S       upLoop                  ;  yes, keep testing

            TRAPTO      _POLL                   ; now flush the key up event
            MOVEQ       #0,D0                   ; no repeats
            MOVEQ       #0,D1                   ; don't wait for event
            TRAPTO      _KeybdEvent             ; get the key up event & ignore it

            .ELSE
            BSR.S       InchNE                  ; wait for post up
;            CMP.W       #KeyDwnEvt,keyEvt       ; was it a keydown?
;            BNE.S       @0                      ; no, keep checking

; it was a key down while we waited for a key-up, so abort uploop and process the key

;            ADDQ        #4,SP                   ; get rid of return address
;            BRA         DealWkey                ; jump into key processing loop

; check for the keyup event

@0          SUBQ        #KeyUpEvt,keyEvt        ; was it a keyup?
            BNE.S       upLoop                  ; no, keep looping
            .ENDC

            RTS                                 ; and return

keepIt
            MOVE.B      D0,(A6)+                ; stuff into buffer
;            BSR.S       UpLoop                  ; wait for key up event (???)
```

```
            BRA         READBUF                 ; and branch to start of read again

;---------------------------------------
; INPUT A CHAR (NO ECHO)                ;
;---------------------------------------

INCHNE
            .IF         onLisaTrue=0            ; kwk - twm
            MOVE.L      VIA,A1                  ; convenient address...

            BTST        #1,VIFR(A1)             ; did 24/60ths sec roll over
            BEQ.S       goKey
            MOVE.B      #02,VIFR(A1)            ; clear it
            ADDQ        #1,D7

            .IF         DDBG
            CMP.W       KeyRoll,D7              ; anything gotten recently?
            .ELSE
            CMP.W       #kbdRoll,D7             ; anything gotten recently?
            .ENDC

            BLT.S       goKey
            MOVEQ       #0,D7                   ; reset "watchdog" timer (OS 201 --
Dynbr Vs[ld)

            CLR.B       KbdTalked               ; fake the keyboard non-talk
            MOVE.L      JKybdTask,A0            ; lomem vector
            JSR         (A0)
goKey
            .ENDC                               ; (kwk)


            CLR.W       keyHit                  ; flag no character

            .IF         onLisaTrue              ; kwk
            TRAPTO      _POLL                   ; since interrupts off, poll
            MOVEQ       #0,D0                   ; key repeats not okay
            MOVEQ       #0,D1                   ; don't wait for event
            TRAPTO      _KeybdEvent             ; get the keyboard event
            TST.B       D0                      ; has anything happened?
            BEQ.S       GotKey                  ;   no event
            TST.B       D2                      ; disk/power event? (ascii 0)
            BEQ.S       GotKey                  ;   yes, ignore
            CMP.B       #1,D2                   ; mouse event? (ascii 1)
            BEQ.S       GotKey                  ;   yes, ignore
            MOVE.B      D2,keyHit               ; high bits off, from CLR keyHit

            .ELSE                               ; (kwk)
            MOVE.L      VIA,A1
            BTST        #2,VIFR(A1)             ; key board event?
            BEQ.S       GotKey

            MOVEQ       #0,D7                   ; cancel delay

            .IF         DDBG
            MOVE.W      KeyWait,D0              ; ho-hum, wait for the keyboard
            .ELSE
            MOVE.W      #kbdWait,D0             ;
            .ENDC

@0          DBRA        D0,@0                   ; to catch up
```

```
            MOVE.L      Lvl1DT+0,A0             ; call the kbd interrupt routine
            JSR         (A0)
            .ENDC

GotKey
            MOVE.B      KeyHit,D0               ; get ascii key value
            CMPI.B      #'a',D0                 ; and convert to UPPER CASE
            BLT.S       @2
            BCLR        #5,D0                   ; clear lower case bit
@2
            TST.B       D0                      ; set 'we have a char' flag
            RTS

post                                            ; replaces PostEvent MacTrap
            MOVE        A0,KeyEvt               ; remember the last event
            CMP         #KeyDwnEvt,A0           ; is it key down?
            BNE.S       @0
            MOVE.B      D0,KeyHit               ;
@0
            RTS


stdEntry
            MOVE.L      (SP)+,temp

            MOVEM.L     D0-D7/A0-A3,-(SP)

; redirect post event stuff

            MOVE.L      $28,-(SP)               ; remember debugger A trap vector
            MOVE.L      SAVEA,$28               ; restore original A-trap vector value

            MOVEQ       #$2F,D0                 ; get address of postEvent
            _GetTrapAddress
            MOVE.L      A0,postEvent            ; save true PostEvent trap

            MOVE.L      RamBase,-(SP)           ; save ram base
            LEA         INCHNE,A0
            MOVE.L      A0,RamBase              ; redirect ram base

            LEA         post,A0                 ; rearrange the post event
            MOVEQ       #$2F,D0
            _SetTrapAddress                     ; traps ok, line 1010 enabled

            SF          AbortPrint              ; clear the aborting print flag

            MOVE.L      temp,A0
            JMP         (A0)                    ; return to caller, RamBase & A-trap
stuff restored in
                                                ; StdExit


WriteLine
            BSR.S       OUTPUT                  ; print all the characters in the IO
buffer
            TST.B       AbortPrint              ; did user abort?
            BNE.S       FixBuf                  ; yes, flush buffer and return

CRLF
            MOVE.B      #$0D,(A6)+
            MOVE.B      #$0A,(A6)+
```

```
OUTPUT
            BSR         StdEntry
@0
            CMP.L       A6,A5
            BGE.S       chThere

            MOVE.B      (A5)+,D0
            BSR.S       OUTCH
            BRA.S       @0

chThere
            BSR         INCHNE                  ; see if user is halting display by
typing
            BNE.S       @1                      ; yes, deal with it
            BSR         INCHNE                  ; better double check (slow keyboard)
            BEQ.S       @3                      ; no, finish the output

; user typed a character while printing.  If backspace, set abort flag and exit, else
; loop, waiting for another character to be typed.

@1          CMP.B       #8,D0                   ; was it a backspace?
            BNE.S       @2                      ; no, go wait for another

            ST          AbortPrint              ; yes, set the flag
            BRA.S       @3                      ; and bail out

@2          BSR         INCHNE                  ; wait for another character
            BEQ.S       @2

@3          BSR.S       FIXBUF
            BRA         StdExit

FIXBUF
            LEA         BUFFER,A5
            MOVE.L      A5,A6
            RTS

; D0 contains the character to print

OUTCH
            LEA         ScreenVars,A2           ; point to variables
            MOVE        (A2)+,D1                ; get Y position

            CMPI.B      #'a',D0                 ; and convert to UPPER CASE
            BLT.S       shifted
            BCLR        #5,D0                   ; clear lower case bit
shifted
            MOVEQ       #0,D7
            MOVE.B      D0,D7                   ; get the character
            MOVEQ       #$3B,D6
@0
            SUB         #$20,D7                 ; normalize to a space
            BMI         nonPrint                ; if under then skip
            CMP         D6,D7                   ; too big
            BLT.S       @1
            MOVE        D6,D7
@1
            MOVE        (A2),D0                 ; get X position
            ADDQ        #1,(A2)+                ; and point to next
skipChar
            MULU        #6,D0                   ; each char = 6 bits wide
```

```
            MOVE        D0,D6                   ; save off bit count
            LSR         #3,D0                   ; turn into byte count

            LEA         font,A0                 ; point to fonts
            MOVE.L      SCRNBASE,A3             ; point to the screen

            ADD         D0,A3                   ; point to the dest byte

            LSL         #3,D1                   ; scanlines/char row = 8
            MULU        screenRow,D1            ; * rowbytes = byte offset
            ADD         D1,A3                   ; point to the first dest byte

            ADD         (A2)+,A3                ; add in offset

            NOT         D6                      ; flip to bit #
            AND         #7,D6                   ; bitize the dest counter

            MOVEQ       #7,D5                   ; row counter
rowLoop
            MOVE.L      A3,A1                   ; restore row

            MOVE        D6,D3                   ; dest bit offset

            MOVE        D7,D2                   ; D7 contains char to plot

            MOVEQ       #7,D1                   ; 1st bit source bit
            LSR         #1,D2                   ; nibble index
            BCC.S       @0
            MOVEQ       #3,D1                   ; second nibble source bit
@0
            MOVEQ       #3,D4                   ; bit counter
bitLoop
            BTST        D1,0(A0,D2)             ; is bit set
            BEQ.S       @1
            BSET        D3,(A1)                 ; set the dest bit
            BRA.S       @2
@1
            BCLR        D3,(A1)                 ; set the dest bit
@2
            BSR.S       NextBit                 ; next dest bit/byte

            DBRA        D4,bitLoop              ; next bits

; clear next two bits along dest
            BCLR        D3,(A1)                 ; set the dest bit
            BSR.S       NextBit                 ; next dest bit/byte
            BCLR        D3,(A1)                 ; set the dest bit

            ADD         #30,A0                  ; next row of font data
            ADD         ScreenRow,A3            ; next row on screen

            DBRA        D5,rowLoop

; clear next two rows along dest

            MOVE.L      A1,D0
            SUBQ        #1,D0
            AND         #$FFFE,D0
            MOVE.L      D0,A1
            CLR.L       (A1)                    ; clear this one
            ADD         ScreenRow,A1            ; next row on screen
            CLR.L       (A1)                    ;
```

```
@3          BRA.S       noScroll                ; go home

; A2 points 1 word in screen vars to x position

nonPrint
            SUB         #$08-$20,D7             ; back space?
            BNE.S       notBack

            SUB         #1,(A2)                 ; back up one
            BPL.S       doSpace
            CLR         (A2)
doSpace     MOVEQ       #0,D7                   ; normalized space
            MOVE        (A2)+,D0                ; get X position
            BRA         skipChar

;--------
NextBit
            SUBQ        #1,D1                   ; next source bit
            SUBQ        #1,D3                   ; next dest bit
            BPL.S       @1                      ; spilled into next byte?
            MOVEQ       #7,D3                   ; high bit
            ADDQ        #1,A1                   ; next byte
@1
            RTS

;--------
notBack
            SUBQ        #$0D-$08,D7             ; Carriage return ?
            BNE         doSpace

; Do a carriage return
            CLR         (A2)                    ; zero x position
            ADDQ        #1,-(A2)                ; increment Y position
            MOVE        (A2),D0                 ; get Y position
            CMP         6(A2),D0                ; compare to max
            BLT.S       noScroll
            SUBQ        #1,(A2)+                ; readjust Y position
            ADDQ        #2,A2                   ; skip x

; Scroll up nLines-1 lines

            MOVE.L      SCRNBASE,A1             ; point to the screen
            ADD         (A2)+,A1                ; add in offset
            MOVE.L      A1,A0                   ; point to source

            MOVE        screenRow,D0            ; * rowbytes
            LSL         #3,D0                   ; lines/char row
            ADD         D0,A0

            MOVE        (A2)+,D0                ; get # lines
            SUBQ        #1,D0                   ; scroll n-1
            MULU        screenRow,D0            ; * rowbytes
            LSL         #1,D0                   ; *8 / 4 for # lines/longs
@0
            MOVE.L      (A0)+,(A1)+
            SUBQ        #1,D0
            BNE.S       @0

            MOVE        screenRow,D0            ; 8 * rowbytes / 4
            LSL         #1,D0                   ; clear the last line
@1
            CLR.L       (A1)+
```

```
            SUBQ        #1,D0
            BNE         @1

noScroll

            RTS


;                         !"#  $%&'  ()*+  ,-./  0123  4567  89:;  <=>?  @ABC
;                        DEFG  HIJK  LMNO  PQRS  TUVW  XYZ^
font
            .WORD       $04A5,$7D44,$4200,$0001,$6666,$9F6F,$6600,$1086,$66E6
            .WORD       $EFF6,$9E39,$8996,$E6E6,$E999,$9AFF
            .WORD       $04A5,$ADA4,$8144,$0002,$9299,$9891,$9966,$2049,$9999
            .WORD       $9889,$941A,$8FD9,$9999,$4999,$9A1F
            .WORD       $040F,$A2A0,$81E4,$0002,$9211,$9E81,$9966,$4F21,$B998
            .WORD       $9888,$941C,$89B9,$9998,$4999,$9A2F
            .WORD       $0405,$6440,$814E,$0F04,$9222,$F1E2,$6700,$8012,$BFE8
            .WORD       $9EEB,$F41A,$8999,$E9E6,$4999,$642F
            .WORD       $040F,$58B0,$81A4,$6064,$9241,$1192,$9166,$4F24,$B998
            .WORD       $9889,$9419,$8999,$89A1,$4999,$944F
            .WORD       $0005,$5B90,$8104,$2068,$9299,$1994,$9962,$2040,$8999
            .WORD       $9889,$9499,$8999,$8B99,$495F,$948F
            .WORD       $0405,$EB60,$4200,$4008,$67F6,$1664,$6604,$1084,$79E6
            .WORD       $EF86,$9E69,$F996,$8796,$4629,$94FF


            .ELSE                               ; twm -- combine 2 conditionals into .IF
- .ELSE - .ENDC
                                                ; first 1/2 is for using screen and 2nd
half is for
                                                ; talking out serial port to a terminal

WriteLine
            BSR.S       OUTPUT                  ; print all the characters in the IO
buffer
            TST.B       AbortPrint              ; did user abort?
            BNE.S       FixBuf                  ; yes, flush buffer and return

CRLF
            MOVE.B      #$0D,(A6)+              ; stuff carriage return/line feed
            MOVE.B      #$0A,(A6)+

OUTPUT
            MOVEM.L     D0-D3/A0-A1,-(SP)       ; save regs
            SF          AbortPrint              ; reset print abort flag
OUTP1
@0
            CMP.L       A6,A5
            BGE.S       OutDone

            MOVE.B      (A5)+,D0
            BSR.S       OUTCH
            BRA.S       @0
OutDone
            CMP.B       #$0A,-1(A6)             ; see if last was CR/LF
            BNE.S       @10

            MOVE        #$FFF,D0                ; delay after CR/LF
@11
            SUB         #1,D0
```

```
            BNE.S       @11
@10
            MOVEM.L     (SP)+,D0-D3/A0-A1

FIXBUF
            LEA         BUFFER,A5
            MOVE.L      A5,A6
            RTS


OUTCH
            BSR.S       GetSCC
            MOVE.B      (A0),D1                 ; synchronize pointing
@1
            SCCDelay                            ; delay for the slow SCC chip...
            MOVE.B      (A0),D1                 ; get stuff
            BTST        #TxBE,D1                ; wait until the buffer is empty
            BEQ.S       @1

@2
            ADDA.L      #SCCWrite,A0            ; generate write address
            MOVE.B      D0,SCCData(A0)          ; put the byte

;----------------------------------------
; CHECK FOR ^S                          ;
;----------------------------------------

CTLW
            BSR.S       GetSCC

            BTST        #RxBF,(A0)              ; Any character to be read?
            BEQ.S       CTLW9                   ; No, return

            SCCDelay                            ; delay for the slow SCC chip...
            MOVEQ       #$7F,D0                 ; mask parity
            AND.B       SCCData(A0),D0          ; get char

            CMPI.B      #$13,D0                 ; X off? (Ctrl - S)
            BEQ.S       CTLWH                   ; Yes, freeze the screen

            CMPI.B      #$3,D0                  ; Control C?
            BNE.S       CTLW9                   ; No, return

            ST          AbortPrint              ; yes, set the flag
            BRA.S       CTLW9                   ; and return

CTLWH
            BTST        #RxBF,(A0)              ; Now wait for a character to be typed
            BEQ.S       CTLWH
            SCCDelay                            ; delay for the slow SCC chip...

            MOVEQ       #$7F,D0                 ; mask parity
            AND.B       SCCData(A0),D0          ; READ IT
            CMP.B       #$11,D0                 ; X on? (Ctrl - Q)
            BNE.S       CTLWH                   ; No, keep looping for char

CTLW9
            RTS                                 ; RETURN

;----------------------------------------
; INITIALIZE ACIA                       ;
;----------------------------------------
```

```
;----------------------------------------------------------------------;
; initialization data for SCC: 9600 baud RS-232 async communication    ;
;   using the baud rate generator; interrupt conditions unchanged      ;
;----------------------------------------------------------------------;

.IF         onMacTrue
boddata:
            .byte       4,$4C                   ; x16 clk, 2 stop bits, no parity
            .byte       14,$00                  ; disable baud rate generator
            .byte       11,$50                  ; baud rate gen clk to receiver,
transmitter
            .byte       12,$A                   ; set baud rate to 9600x16 baud
            .byte       13,$00                  ; "
            .byte       14,$01                  ; enable baud rate generator
            .byte        3,$C1                  ; 8 bits/char recv, enable receiver
            .byte        5,$6A                  ; 8 bits/char xmit, enable xmitter
boddlth     .equ        *-boddata
.ENDC
.IF         onVaccTrue                          ; twm
boddata:
            .byte        4,$4C                  ; x16 clk, 2 stop bits, no parity
            .byte       14,$00                  ; disable baud rate generator
            .byte       11,$D0                  ; XTAL, baud rate gen clk to receiver,
transmitter
            .byte       12,$A                   ; set baud rate to 9600x16 baud
            .byte       13,$00                  ; "
            .byte       14,$01                  ; enable baud rate generator
            .byte        3,$C1                  ; 8 bits/char recv, enable receiver
            .byte        5,$6A                  ; 8 bits/char xmit, enable xmitter
boddlth     .equ        *-boddata
.ENDC


GetSCC
            .IF         APort
            MOVE.L      #SCCRBase+ACtl,A0       ; pointer to SCC base read address
            .ELSE
            MOVE.L      #SCCRBase+BCtl,A0       ; pointer to SCC base read address
            .ENDC

            ADD.L       outAddress,A0           ; add in offset to other port.

            RTS


INITACIA
            BSR.S       InitSCC
            .IF         APort
            SUBQ.L      #2,outAddress
            .ELSE
            ADDQ.L      #2,outAddress
            .ENDC                               ; if aport
            BSR.S       InitSCC
            CLR.L       outAddress
            RTS

InitSCC
            MOVEM.L     D1-D2/A0-A2,-(SP)
            BSR.S       GetSCC
            MOVE.L      A0,A1
            ADDA.L      #SCCWrite,A1            ; generate SCC write address
```

```
            LEA         BODDATA,A2              ; set channel a
            MOVE.W      #BODDLTH,D1

            MOVE.B      (A0),D2                 ; read to make sure SCC is sync'ed up
            BRA.S       @2                      ; delay for timing, too
@1          MOVE.B      (A2)+,(A1)
@2          SCCDelay                            ; twm - delay for the slow SCC chip...
            DBRA        D1,@1
            TST.B       SCCData(A0)             ; CLEAR RX BUFFER

            MOVEM.L     (SP)+,D1-D2/A0-A2

            RTS


;----------------------------------------
; ACIA INPUT ROUTINES                   ;
;----------------------------------------
ReadLine
            MOVEM.L     D0-D4/A0-A2,-(SP)
            BSR         GetSCC
READBUF
            BSR         INCHNE
            TST.B       D0
            BEQ.S       READBUF
RB1
            MOVE.W      D0,D4
            CMPI.B      #8,D0           ;BACKSPACE?
            BEQ.S       RB11            ;YES, DON'T ECHO IT
            BSR         OUTCH
RB11
            MOVE.W      D4,D0

;----------------------------------------
; PROCESS BACKSPACES (FOR CRT TERMINAL) ;
;----------------------------------------
RB2
            CMPI.B      #8,D0                   ;BACKSPACE?
            BNE.S       RB5                     ;NO, SKIP

            CMP.L       A6,A5                   ;ANY CHARS TO DELETE?
            BHS.S       READBUF                 ;NO, IGNORE BACKSPACE
            BSR         OUTCH                   ;ELSE ECHO BACKSPACE
            MOVE        #$20,D0                 ;  OUTPUT A SPACE
            BSR         OUTCH
            MOVE        #$8,D0                  ;  AND BACKSPACE
            BSR         OUTCH
            SUB.L       #1,A6                   ;AND REMOVE CHAR FROM BUFFER
            BRA         READBUF
RB5
            CMPI.B      #$D,D0
            BNE.S       RB6
            MOVE.B      #$A,D0
            BSR         OUTCH

            MOVE.B      #' ',(A6)               ; pad end with a blank

            MOVEM.L     (SP)+,D0-D4/A0-A2
            RTS
RB6
            MOVE.B      D0,(A6)+
```

```
            BRA         READBUF

;---------------------------------------
; INPUT A CHAR (NO ECHO)                ;
;---------------------------------------

INCHNE
            MOVE.B      (A0),D0                 ; read to sync up

@1          SCCDelay                            ; twm - delay for the slow SCC chip...
            BTST        #RxBF,(A0)              ; wait until a byte is available
            BEQ.S       @1
            SCCDelay                            ; twm - delay for the slow SCC chip...
            MOVEQ       #$7F,D0                 ; mask parity
            AND.B       SCCData(A0),D0          ; get the byte

            CMPI.B      #'a',D0                 ; and convert to UPPER CASE
            BLT.S       INCHNEX
            BCLR        #5,D0
INCHNEX
            RTS                                 ; RETURN


          .ENDC                                 ; twm - end of .IF - .ELSE - .ENDC
conditional for
                                                ; use of the screen or an external
terminal.
```