```
;-----------------------------------------------------------------------------
;       Multiplexor Drivers for XENIX - 6000, by Frank Durda IV
;       Tandy doesn't want 'em, so Copyright 1988 Frank Durda IV
        .asciz  "Mux (c) 1988 fdiv"
;-----------------------------------------------------------------------------
;       Mux Interrupt Service Routine - Handles all requests
;-----------------------------------------------------------------------------
Mux0Intr:
        exx                             ;<127>Save normal regs
        ld      b,4                     ;<127>TTY04
        push    ix                      ;<127>Save that
        ld      ix,Mux1Map1+MuxChan4    ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux0LPB              ;<127>Point at parms
        jp      MuxCom                  ;<127>Common

Mux1Intr:
        exx                             ;<127>Save normal regs
        ld      b,5                     ;<127>TTY05
        push    ix                      ;<127>Save that
        ld      ix,Mux1Map1+MuxChan3    ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux1LPB              ;<127>Point at parms
        jp      MuxCom                  ;<130>Common

Mux2Intr:
        exx                             ;<127>Save normal regs
        ld      b,6                     ;<127>TTY06
        push    ix                      ;<127>Save that
        ld      ix,Mux1Map1+MuxChan2    ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux2LPB              ;<127>Point at parms
        jp      MuxCom                  ;<130>Common

Mux3Intr:
        exx                             ;<127>Save normal regs
        ld      b,7                     ;<127>TTY07
        push    ix                      ;<127>Save that
        ld      ix,Mux1Map1+MuxChan1    ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux3LPB              ;<127>Point at parms
        jp      MuxCom                  ;<130>Common

Mux4Intr:
        exx                             ;<127>Save normal regs
        ld      b,8                     ;<127>TTY08
        push    ix                      ;<127>Save that
        ld      ix,Mux2Map2+MuxChan4    ;<127>Second Board, 1st UART
        push    iy                      ;<127>And that
        ld      iy,Mux4LPB              ;<127>Point at parms
        jr      MuxCom                  ;<127>Common
eject
Mux5Intr:
        exx                             ;<127>Save normal regs
        ld      b,9                     ;<127>TTY09
        push    ix                      ;<127>Save that
        ld      ix,Mux2Map2+MuxChan3    ;<127>
```

```
        push    iy                      ;<127>And that
        ld      iy,Mux5LPB              ;<127>Point at parms
        jr      MuxCom                  ;<127>Common

Mux6Intr:
        exx                             ;<127>Save normal regs
        ld      b,10                    ;<127>TTY10
        push    ix                      ;<127>Save that
        ld      ix,MuxMap2+MuxChan2     ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux6LPB              ;<127>Point at parms
        jr      MuxCom                  ;<127>Common

Mux7Intr:
        exx                             ;<127>Save normal regs
        ld      b,11                    ;<127>TTY11
        push    ix                      ;<127>Save that
        ld      ix,MuxMap2+MuxChan1     ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux7LPB              ;<127>Point at parms
        jr      MuxCom                  ;<127>Common

Mux8Intr:
        exx                             ;<127>Save normal regs
        ld      b,12                    ;<127>TTY12
        push    ix                      ;<127>Save that
        ld      ix,MuxMap3+MuxChan4     ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux8LPB              ;<127>Point at parms
#ifdef MUXEXTEND
        jr      MuxCom                  ;<127>Common

Mux9Intr:
        exx                             ;<127>Save normal regs
        ld      b,13                    ;<127>TTY13
        push    ix                      ;<127>Save that
        ld      ix,MuxMap3+MuxChan3     ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux9LPB              ;<127>Point at parms
        jr      MuxCom                  ;<127>Common

Mux10Intr:
        exx                             ;<127>Save normal regs
        ld      b,14                    ;<127>TTY14
        push    ix                      ;<127>Save that
        ld      ix,MuxMap3+MuxChan2     ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux10LPB             ;<127>Point at parms
        jr      MuxCom                  ;<127>Common

Mux11Intr:
        exx                             ;<127>Save normal regs
        ld      b,15                    ;<127>TTY15
        push    ix                      ;<127>Save that
        ld      ix,MuxMap3+MuxChan1     ;<127>
        push    iy                      ;<127>And that
        ld      iy,Mux11LPB             ;<127>Point at parms
#endif  ;MUXEXTEND
```

```
eject
;--------------------------------------------------------------------------
;       One Guy does-all interrupt service routine for UART boards
;--------------------------------------------------------------------------

MuxCom:
        exaf                            ;<127>At least this can be common
        call    RETI                    ;<129>Do this early on.  Trying
                                        ;<129>to cover lost interrupts
                                        ;<129>we were having.
#ifdef MUXDEBUG
        call    sprint
        .asciz  "I"
#endif
..anymore:
        ld      c,(ix+MUXINTID)         ;<127>Was the interrupt real?
#ifdef MUXDEBUG
        ld      a,c
        call    sputb
#endif
        bit     #MuxNoIntPend,c         ;<127>If Bit 0 is 0, we have IRQ
        jr      z,..isint               ;<127>Service

        exaf                            ;<127>
        exx                             ;<127>
        pop     iy                      ;<127>
        pop     ix                      ;<127>
        ei      ;<127>------------------------------------------------------
        ret                             ;<129>Already reti'ed
;<129>  reti                            ;<127>

;       Here we have an interrupt.  The order we service the chip is
;       supposed to be important.
;       It says,        (1)     line status (overrun, parity, etc)
;                       (2)     receive characters
;                       (3)     transmit characters
;                       (4)     modem changes

..isint:

;       Now we test to see if we have any characters to read

        ld      d,(ix+MUXLSTAT)         ;<127>
        bit     #MuxRxChar,d            ;<127>Any characters to receive?
        jr      z,..norcv               ;<127>No characters to receive
eject
;       We have at least one character to read

        ld      c,b                     ;<127>Save B
        bit     #MuxRxOver,d            ;<127>Has there been an overrun?
        jr      z,..noover              ;<127>No, so far so good

        set     #OVERRUN,b              ;<127>Add to line number

..noover:
        bit     #MuxRxPar,d             ;<127>Is there a parity error?
        jr      z,..nopar
        set     #PARERR,b               ;<127>
```

```
..nopar:
        bit     #MuxRxFrame,d           ;<127>
        jr      z,..nofra               ;<127>
        set     #FRAMERR,b              ;<127>

..nofra:
#ifdef MUXDEBUG
        call    sprint
        .asciz  "R"
#endif
        ld      a,(ix+MUXIO)            ;<127>Read data byte (good or bad)
#ifdef MUXDEBUG
        call    sputb
#endif
        call    Sttyinput               ;<127>Pass to 68k
        ld      b,c                     ;<127>Get TTY number back

;       Here we clear any transmit interrupts and service them

..norcv:
        ld      d,(ix+MUXINTID)         ;<127>Reading this register clears
                                        ;<127>
        ld      d,(ix+MUXLSTAT)         ;<127>Get line status
        bit     #MuxTxRdy,d             ;<127>Do we want to transmit?
        jr      z,..chkmodem            ;<128>Not ready to transmit

        call    Sttyoutput              ;<127>Get a character
        jr      z,..chkmodem            ;<128>None available
        ld      (ix+MUXIO),a            ;<127>Transmit character
#ifdef MUXDEBUG
        call    sprint
        .asciz  "T"
        call    sputb
#endif
;       Now check on the modem for any status changes

..chkmodem:
        call    MuxModem                ;<128>Get current state and report
                                        ;<128>This code is shared by setstat
        jr      ..anymore               ;<128>Make sure all is resolved

eject
;       Now we will check to see it there was a line status change
;       The hardware lets us know of changes, but we can't use that info
;       since there is no reasonable way to prime the pump.  The IRQ
;       will serve as reporting a change.

MuxModem:
        ld      a,(ix+MUXMSTAT)         ;<128>Read modem status
        and     MuxCTS+MuxDSR+MuxDCD+MuxRI      ;<128>Lose deltas
        cp      (iy+LSrctl)             ;<128>Same as last one we saw?
        ret     z                       ;<128>Don't report it
        ld      (iy+LSrctl),a           ;<128>Update our copy
        ld      d,a                     ;<128>Store it here
        xor     a                       ;<127>Get a zero
        bit     #MuxDCD,d               ;<127>Do we still have carrier?
        jr      z,..isof                ;<127>Nope
```

```
        ld      a,DSRON+CTSON+DCDON     ;<127>Set DCD plus a few other things
..isof:
#ifdef MUXDEBUG
        call    sprint
        .asciz  "C"
        call    sputb
#endif
        ld      d,b                     ;<127>Save D
        set     #MCTLNTFY,b             ;<127>Add notify flag to line number
        call    Sttyinput               ;<127>Let 68k know of status change
        ld      b,d                     ;<127>Restore D
        ret                             ;<127>All done


;---------------------------------------------------------------------------
;       Start transmissions on Mux lines
;       This routine runs with interrupts disabled
;---------------------------------------------------------------------------
MuxTxStart:
#ifdef MUXDEBUG
        call    sprint
        .asciz  "G"
#endif
        ld      e,(iy+LSCtlPort)        ;<127>Actually contains LSB of addr
        ld      d,0xf2                  ;<127>MSB of address
        push    de                      ;<127>Put in IX
        pop     ix                      ;<127>Memory map address

        ld      d,(ix+MUXLSTAT)         ;<127>Get line status
        bit     #MuxTxRdy,d             ;<127>Do we want to transmit?
        ret     z                       ;<127>Still transmitting, so
                                        ;<127>it will prime itself

        ld      b,(iy+LSTTY)            ;<127>Get TTY number
        call    Sttyoutput              ;<127>Get a character
        ret     z                       ;<127>Should not happen
        ld      (ix+MUXIO),a            ;<127>Transmit character
#ifdef MUXDEBUG
        call    sputb
#endif
        ret                             ;<127>It will feed itself
eject
;---------------------------------------------------------------------------
;       Set Status on MUX channels
;---------------------------------------------------------------------------
MuxSetStatus:
#ifdef MUXDEBUG
        call    sprint
        .asciz  "Set"
#endif
        ld      e,(ix+zs_mode)          ;<127>e = iopb.mode
        ld      d,(ix+zs_wctl)          ;<127>d = iopb.wctl
        ld      l,(ix+zs_baud)          ;<127>l = iopb.baud

        ld      a,(iy+LSMode)           ;<127>Does this change the mode?
        cp      e                       ;<127>
        jr      nz,..ok                 ;<127>Mode needs changing
```

```
        ld      a,(iy+LSwctl)           ;<127>Any line status changes?
        cp      d                       ;<127>
        jr      nz,..ok                 ;<127>Some signal changes state

        ld      a,(iy+LSBaud)           ;<127>Does baud rate or word length
        cp      l                       ;<127>change?
;<129>  jp      z,..done                ;<127>Nothing changes, signal done
        ret     z                       ;<129>Nothing changes, then exit

..ok:
        ld      (iy+LSMode),e           ;<127>Store new mode flags
        ld      (iy+LSwctl),d           ;<127>Store new line status

;       First we will build word length, stop bit and parity info

        ld      a,l                     ;<127>Get word length & baud rate
        and     WORDMASK                ;<127>Lose baud rate for now
        rrca                            ;<127>The kernels numbering system
        rrca                            ;<127>is the same as the 8250's
        rrca                            ;<127>This will work as long as
        rrca                            ;<127>the kernels scheme is constant
        ld      b,a                     ;<127>Put it in B

        bit     #PARENAB,e              ;<127>Is parity enabled
        jr      z,..nopar               ;<127>No parity
        set     #MuxParity,b            ;<127>Set parity
        bit     #PARODD,e               ;<127>Even or Odd?
        jr      nz,..odd                ;<127>Odd
        set     #MuxEven,b              ;<127>Set even parity
..odd:
..nopar:
        bit     #STOP2,e                ;<127>One or two stop bits
        jr      z,..one                 ;<127>One will do
        set     #Mux2stop,b             ;<127>Two stop bits
..one:
eject
;       The Line Control register is now constructed.
;       Now we handle modem control

        xor     a
        bit     #RTSON,d                ;<127>Turn RTS on?
        jr      z,..norts               ;<127>No, turn it off
        or      MuxRTS                  ;<127>Turn it on
..norts:
        bit     #DTRON,d                ;<127>Turn DTR on?
        jr      z,..nodtr               ;<127>
        or      MuxDTR                  ;<127>
..nodtr:

;       A = modem control, b = line control

        push    ix                      ;<127>Save IOPB
        ld      c,a                     ;<127>Move modem control

;       Calculate the device address

        ld      e,(iy+LSCtlPort)        ;<127>Actually contains LSB of addr
        ld      d,0xf2                  ;<127>MSB of address
```

```
        push    de                      ;<127>Put in IX
        pop     ix                      ;<127>Memory map address
#ifdef NEVER
;<130>  Althougvh the idea is sound, we could hang here for the time
;<130>  it takes to transmit 30 characters at a low baud rate.
;<130>  Hopefully, leaving the Tx mask intact will handle the problem
;<130>  as well as it appears to have done on the PCI boards.
        ld      a,(ix+MUXDECODE)        ;<128>See if we have the iron
        inc     a                       ;<128>If it is there, we got a 0xff
        jr      z,..wait                ;<128>We have hardware

;       Here we do not have the hardware, so update the tables in RAM
;       and exit

        ld      (iy+LSBaud),1           ;<128>Set it so we won't keep trying
        pop     ix                      ;<128>
        ret                             ;<128>Exit now

;       We have hardware, so program it

..wait:
        ld      a,(ix+MUXLSTAT)         ;<128>Make sure last char has cleared
        bit     #MuxTxEmpty,a           ;<128>Well?
        jr      z,..wait                ;<128>Still transmitting
#endif

;       Now get baud rate

        ld      a,(iy+LSBaud)           ;<127>Does baud rate or word length
        and     BAUDMASK                ;<127>Determine if baud rate changes
        ld      d,a                     ;<127>Save old baud
        ld      a,1                     ;<127>Get new setting
        and     BAUDMASK                ;<127>Lose wordsize
        cp      d                       ;<127>Is there a difference?
eject
;       Before we decide, store updated WORD and BAUD (even if the same)

        ld      (iy+LSBaud),1           ;<127>

        jp      z,..doline              ;<127>Baud stays the same, so skip this

;       Here we know they want to change the baud rate, so do it.

        ld      hl,..MuxBaudTbl         ;<127>
        rlca                            ;<127>Multiply by two for wordsize
        ld      e,a                     ;<127>
        ld      d,0                     ;<127>
        add     hl,de                   ;<127>

        ld      a,(hl)                  ;<127>Get LSB of divisior
        inc     hl
        ld      d,(hl)                  ;<127>Get MSB of divisior
        ld      e,a                     ;<127>Put LSB here
        or      d                       ;<127>Is it legal?
        jr      z,..doline              ;<127>Don't change baud rate

;       c = modem control, b = line control
;       e = LSB divisior, d = MSB divisior
```

```
        di      ;<127>--------------------------------------------------
        ld      a,b                     ;<127>Get line control
        or      MuxDivisor              ;<127>Access divisor registers
        ld      (ix+MUXLINE),a          ;<127>Until this is reset, no IRQ's

        ld      (ix+DIVLSB),e           ;<127>Store divisor
        ld      (ix+DIVMSB),d           ;<127>Store MSB divisor
#ifdef MUXDEBUG
        call    sprint
        .asciz  "Div"
        ld      a,d
        call    sputb
        ld      a,e
        call    sputb
#endif

..doline:
        ld      (ix+MUXLINE),b          ;<127>Turn divisor off
        ld      (ix+MUXMODEM),c         ;<127>Set modem status
#ifdef MUXDEBUG
        call    sprint
        .asciz  "Line"
        ld      a,b
        call    sputb
        ld      a,c
        call    sputb
#endif
        ld      b,(iy+LSTTY)            ;<127>Get TTY to report on
        call    MuxModem                ;<127>Check for status changes

        xor     a                       ;<127>
        bit     #RXDISAB,(iy+LSMode)    ;<127>Is Rx enabled?
        jr      nz,..ena                ;<127>Yes
        ld      a,MuxAllIEnab           ;<127>Turn everything on
        ld      c,(ix+MUXIO)            ;<127>Read and toss current char
                                        ;<127>if any
..ena:
        ld      (ix+MUXINTEI),a         ;<127>Set interrupts
        ei      ;<127>--------------------------------------------------
        pop     ix                      ;<127>Restore IX (did we need to?)
..done:
        ret                             ;<127>All done
eject
;       The values are derived from the formula, Clk / baud rate / 16.
;       On this board, the system clock / 2 is used = 2,000,000.
;       If a specific crystal was used, the resulting values would be
;       more accurate.  19,200 is between 6 and 7 (6.51) and is quite
;       inaccurate.

..MuxBaudTbl:
        .word   0                       ;<127>0   NOTE: 0 means invalid rate
        .word   2500                    ;<127>50
        .word   1667                    ;<127>75
        .word   1136                    ;<127>110
        .word   932                     ;<127>134
        .word   833                     ;<127>150
        .word   625                     ;<127>200
```

```
        .word   417                     ;<127>300
        .word   208                     ;<127>600
        .word   104                     ;<127>1200
        .word   69                      ;<127>1800
        .word   52                      ;<127>2400
        .word   26                      ;<127>4800
        .word   13                      ;<127>9600
        .word   6                       ;<127>extra1
        .word   3                       ;<127>extra2

..MuxMaxRate    ==      $-..MuxBaudTbl >> 1     ;<127>Number of entries

eject
```