```
        ;CM1.RESTART, subroutine, Level 1

        ;Subrcutine to "restart" a transfer. Merely copies the relavent IOCB
        ;into the register block cache and points L7.NEXT to 0 (expecting
        ;new frame). Enter with L4.IOCB pointing up at the IOCB to get the
        ;information from.

        cm1.restart:
 026075  l4.iocb ls                        ;make sure we're called with legit ar
 026076  l4.iocb + iocb.addr -> mar(r)      ;read the transfer start address
 026077  if zero crash -> upc              ;[1]
 026100  if zero ioerr.ddiia -> temp       ;[2] ++ illegal IOCB arg

 026101  l4.iocb + iocb.size -> mar(r)     ;read the size
 026102  mbr -> l5.addr                    ;[1] remember start addr
 026103  0 -> l7.next                      ;[2] reset the interrupt routine

 026104  mbr -> l6.left ls                 ;better not be zero
 026105  if zero crash -> upc
 026106  if zero ioerr.ilts -> temp

 026107  l6.left & 17 ls                   ;better be a multiple of 16 bits
 026110  if zero l1.ret1 -> upc            ;return to caller if ok

 026111  ifnot zero crash -> upc
 026112  ioerr.ilts -> temp                ; ++ Illegal transfer size
```

```
        ;Subrcutine CM1.EDONE, Subroutine, Level 2

        ;Come here with L4.IOCB pointing to the current IOCB which should be
        ;given and error completion code (cm1.cc.error) and removed. Enter at
        ;EDONESTOP to not let DONE (below) call START, thus preventing the
        ;DCB.IOCB window to advance to then next IOCB.

        cm1.edonestop:
 026113  l2.ret2 -> temp                    ;tell CM1.DONE0 to return to caller
 026114  cm1.edone0 -> upc
 026115  temp -> g6.temp

        cm1.edone:
 026116  cm1.start -> g6.temp               ;tell CM1.DONE0 to call START afterwa

        cm1.edone0:
 026117  14.iccb + iocb.flags -> mar(r) ;set the completion code to error
 026120  cm1.cc.error -> g2.temp          ;[1] get the code
 026121  0 -> 17.next                    ;[2] nothing else does this for us.
 026122  g2.temp ! mbr -> mbr            ;write it back
 026123  14.iccb + iocb.flags -> mar(w)
 026124  nop                             ;[1]
 026125  nop                             ;[2]

        ;fall into CM1.DONE0
```

```
        ;CM1.DONE0, Subroutine, Level <weird>

        ;Enter with L4.IOCB pointing to IOCB finished. Compute the size trans
        ;by coverting our L6.LEFT back into bits (by multiplying by 16), subt
        ;it from IOCB.SIZE (the original max size) giving a bit count. We the
        ;IO.PUT, clear the current pointer and return to the placed pointed t
        ;G6.TEMP. Enter at CM1.DONE to set G6.TEMP to CM1.START, (the normal
        ;thus causing the next IOCB to be fetched.

        cm1.done0:
026126   l4.iocb + iocb.size -> mar(r)  ;read the IOCB max length
026127   l6.left -> temp                ;[1] get a copy of LEFT
026130   0 -> l4.iocb                   ;[2] we're done with this IOCB
026131   mbr -> g2.temp                 ;move to useful place
026132   g2.temp - temp -> mbr          ;compute number of bits xferred
026133   mar -> mar(w)                  ;write it back
026134   iodd.put -> upc                ;[1] put it away
026135   l0.dcb -> temp                 ;[2] for IODD.PUT call (DCB in TEMP)
                                        ;note that IODD.PUT will return throu
                                        ;to wherever caller told it to.
        cm1.done:
026136   cm1.done0 -> upc               ;enter here to set G6 to get the next
026137   cm1.start -> g6.temp           ; ... by calling CM1.START

026140          unext == .
        # 23 "m.mic"
```

```
        # 1 "c18.mic"
```

```
        ; The NMFS C/30 IMP I/O system microcode for the 1822-Host
        ; Interfaces.
        ;
        ;                   Glenn M. Simpson, E. A. Hahn
        ;                         January, 1982
        ;
        ; Edit History:
        ;
        ;       ADO, 4 June 82, changed for new config scheme, removed NMFS (
        ;       entry point, device vectors and register block allocations.
        ;       dispatch and all tables to use one word entries.
        ;
        ; In this file the following prefixes appear often.
        ;
        ;       C18-    Common 1822-Host Interface
        ;       C18I-   Input (receiver) side of interface.
        ;       C18O-   Output (transmitter) side of interface.
        ;
        ;
        ; XDV  Code#     Command          Function
        ;
        ;       (0)      RESET            Reinitialize device, aborts current I
        ;       (1)      RAISE            Raise IMP Master Ready line.
        ;       (2)      LOWER            Drop IMP Master Ready line.
        ;       (3)      STATUS           Read CSR.
        ;       (4)      CROSS-PATCH      Cross-patch interface.
        ;       (5)      UNCROSS-PATCH    Uncross-patch interface.

        ;Local register definitions: (i)-input, (o)-output, (b)-both.

000000  10.csr  == 10                    ;(i) Saved contents of CSR (sign alwa
000000  10.1822 == 10                    ;(o) Interface physical address.
000001  11.ret1 == 11                    ;(b) Level 1 subroutine link.
000002  12.dcb  == 12                    ;(b) DCB back-pointer.
000003  13.iocb == 13                    ;(b) IOCB pointer, 0 = none.
000004  14.addr == 14                    ;(b) Macro address of IOCB buffer.
000005  15.cntr == 15                    ;(b) Count of remaining xfers.
000006  16.char == 16                    ;(b) Character transfered.
000007  17.next == 17                    ;(b) Left off address of coroutine
```

```
                ;Configuration:

000003          c18i == 3                       ;Device number for input.
000004          c18o == 4                       ;Device number for output.

                ;1822 definitions:

000000          u1822.csr == 0                  ;Control and status register.
000001          u1822.rcv == 1                  ;Receive data register (RO).
000001          u1822.xmt == 1                  ;Transmit data register (WO).

                ;CSR definitions (also for "10.csr"):

000001          h.xmtien == 1_0                 ;Xmtr interrupt enable: R/W.
000002          h.imprdy == 1_1                 ;IMP Master Ready Line: R/W.
000004          h.lstbit == 1_2                 ;Xmtr last-bit enable: R/W.
000010          h.xmtrst == 1_3                 ;Xmtr reset: WO.
000020          h.xpatch == 1_4                 ;Interface cross-patch: R/W.
000040          h.rcvien == 1_5                 ;Rcvr interrupt enable: R/W.
000100          h.rcvrst == 1_6                 ;Rcvr reset: W/O.
000200          h.spare  == 1_7                 ;Unused bit: R/W.
000400          h.rcvpnd == 1_10                ;Rcvr interrupt pending: RO.
001000          h.xmtpnd == 1_11                ;Xmtr interrupt pending: RO.
002000          h.hstdwn == 1_12                ;Host Ready Line down: RO.
004000          h.hstflp == 1_13                ;Host Ready Line was down: R/W.
```

```
        ;FUNCTION DISPATCH:

026140  uram unext

        i18.0:                          ;Input side dispatch Table.
026140   c18i.apr -> upc
026141   c18.dpr -> upc
026142   c18.xdv -> upc
026143   c18.pdv -> upc

        o18.0:                          ;Output side dispatch Table.
026144   c18o.apr -> upc
026145   c18.dpr -> upc
026146   c18.xdv -> upc
026147   c18.pdv -> upc
```

```
        ;INTERRUPT HANDLER - input side.

        c18i.int:                                   ;Here on micro-interrupts.
 026150  110 + u1822.rcv -> mar(rio)                ;Read the received character (L10=out
 026151  m2.ios -> misc2                            ;Read will clear interrupt.
 026152  13.iocb ls                                 ;[1] Is there a current IOCB?
 026153  ifnot zero 17.next -> upc                   ;Yes.  Go to next byte handler
 026154   s16mbr -> g0.temp ls                      ;this is useful for HIB routine

 026155  crash -> upc                               ;if we get an interrupt that we dont
 026156   ioerr.uint -> temp                        ; ++ unexpected input interrupt

        ;Routine to service the hi byte saved in the MBR. Come here with MBR
        ;input byte, G0.TEMP having SMBR LS'ed.

        c18i.hib:
 026157  g0.temp & 177400 -> temp                   ;mask down to high byte only
 026160  temp -> l6.char                            ;save it until we have the low byte,

 026161  ifnot odd io.ret -> upc                    ;if not last bit (see setting of G0 a
 026162   ifnot odd c18i.lob -> 17.next             ;expecting low byte next, please

 026163  l6.char ! 200 -> mbr                       ;get padding
 026164  g0.temp & (7_1) -> g0.temp ls             ;shift count

 026165  ifnot zero g0.temp - 20 -> g0.temp        ;shifts done by h/w plus ours
 026166  ifnot zero mar(s1)                         ;may not shift at all
 026167  ifnot zero (.-1) -> upc                    ;any more shifting?
 026170  ifnot zero g0.temp + 2 -> g0.temp ls      ;account for any shifting don

        c18i.store.last:
 026171  c18i.store.mbr -> upc
 026172  (.+1) -> l1.ret1

 026173  c18.doneeom -> upc
 026174  io.ret -> l1.ret1
```

```
        ;Routine to service lo byte and transfer word into IOCB.
        ;The incoming byte is still in the MBR, SMBR in G0.TEMP, LS'ed.

        c18i.lob:
026175   mbr -> g1.temp                    ;extract only the data portion of the
026176   g1.temp & 377 -> temp, g1.temp
026177   l6.char ! temp -> l6.char          ;make up a full word

026200   if odd c18i.lo.last -> upc         ;Now check to see if this is the last
026201   c18i.hib -> 17.next
026202   c18i.store -> upc                  ;but store this one
026203    (.+1) -> l1.ret1

026204   ifnot zero io.ret -> upc           ;if not exhausted, continue along
026205   if zero c18.done -> upc            ;else try for a new transfer
026206    io.ret -> l1.ret1

        ;Here to shift and add padding to the low byte, if it the last one.

        c18i.lo.last:
026207   g0.temp & (7_1) -> g0.temp ls              ;get shift count
026210   if zero c18i.last.hard -> upc              ;no shifting needed is the ha

026211   ifnot zero g0.temp - 20 -> g0.temp         ;shifts done by h/w plus ours
026212   g1.temp + temp -> g1.temp                  ;first shift
026213   g1.temp + 1 -> mbr                         ;padding

026214   g0.temp + 2 -> g0.temp ls                  ;count first, or any other, s
026215   ifnot zero (.-1) -> upc
026216   ifnot zero mar(s1)                         ;done here

026217   l6.char & (377_10) -> l6.char              ;remove unshifted low byte
026220   c18i.store.last -> upc
026221   l6.char ! mbr -> mbr                       ;put in high byte
```

```
                ;This is the case where we must write an extra whole word of padding.
                ;we call C18I.STORE to write the current word (still in L6.CHAR) and
                ;store a word of padding after it.  If we can't, we're in big trouble
                ;to do in the next IOCB.

                c18i.last.hard:
 026222          c18i.store -> upc              ;write the last word
 026223           (.+1) -> l1.ret1
 026224          if zero c18i.last.impossible -> upc ;if IOCB is exhaused, we're REAL
 026225          100000 -> l6.char              ;set up the padding word
 026226          c18i.store -> upc              ;write it
 026227           (.+1) -> l1.ret1

 026230          c18.doneeom -> upc             ;and start anew later.
 026231          io.ret -> l1.ret1

                ;Here when we must output the 100000 into the next IOCB because there
                ;in this one.  We set L7.NEXT to -1 as a flag for START to see that w
                ;this special situation.

                c18i.last.impossible:
 026232           -1 -> l7.next                 ;but make START notice our predicamen
 026233          c18.done -> upc                ;finish this one
 026234           io.ret -> l1.ret1
```

```
         ;C18I.STORE, subroutine, level 1
         ;
         ;writes L6.CHAR into the IOCB. Returns with addr and cnt updated, cnt

         c18i.store:
026235    16.char -> mbr                ;set up for write

         ;Here with data in mbr instead of L6.CHAR.

         c18i.store.mbr:
026236    14.addr -> mar(w)             ;Store it in the IOCB buffer.
026237    14.addr + 1 -> 14.addr        ;[1] Update buffer address.
026240    11.ret1 -> upc                ;[2] return
026241    15.cntr - 20 -> 15.cntr ls    ;and count
```

```
                ;INTERRUPT HANDLER - output side.

                c18o.int:
 026242          13.iocb ls                      ;is there a transfer?
 026243          ifnot zero 17.next -> upc       ;run the sate machine
 026244          if zero crash -> upc
 026245          if zero ioerr.uint -> temp      ; ++ unexpected output interrupt

                ;Routine to service the hi byte.  MBR holds the character.

                c18o.hib:
 026246          14.addr -> mar(r)               ;get a word from buffer.
 026247          14.addr + 1 -> 14.addr          ;[1] Update buffer address.
 026250          15.cntr - 20 -> 15.cntr         ;[2] update count, too
 026251          mbr -> 16.char                  ;save word
 026252          s16mbr -> g0.temp               ;get high byte
 026253          g0.temp & 377 -> mbr            ;only
 026254          10.1822 + u1822.xmt -> mar(wio)
 026255          m2.ios -> misc2
 026256          c18o.lob -> 17.next             ;<1>
 026257          io.ret -> upc                   ;<2>
 026260          nop                             ;<3> (<4>, [4,5] in IO.RET)

                ;Routine to service the lo byte. L6.CHAR has the character.
                ;Test the EOM flag in the IOCB to determine if the "last bit"
                ;flag should be transmitted along with the data.

                c18o.lob:
 026261          15.cntr ls                      ;is this the last word?
 026262          ifnot zero c18o.lob.middle -> upc
 026263          if zero 13.iocb + iocb.flags -> mar(r)
 026264          base -> g0.temp                 ;[1] get copy of BASE for later
 026265          200 -> g1.temp                  ;[2] get a mask for testing flags wor
 026266          mbr & g1.temp ls                ;is this really EOM?
 026267          if zero c18o.lob.last -> upc    ;if EOM is off, don't set last bit
 026270          if zero c18o.last -> 17.next    ;but go to same finish up routine

 026271          g0.temp & ~17 -> base           ;over to input side
 026272          10 ! h.lstbit -> 10, mbr        ;set LAST-BIT if last word
 026273          110 + u1822.csr -> mar(wio)
 026274          m2.ios -> misc2                 ;<1>
 026275          g0.temp -> base                 ;<2> then fall into code to output th
 026276          c18o.lob.last -> upc            ;<3> skip setting up L7.NEXT to HIB
 026277          c18o.last -> 17.next            ;<4> and go to LAST instead

                c18o.lob.middle:
 026300          c18o.hib -> 17.next             ;assume we're going to HIB next
                c18o.lob.last:                  ;enter here from above if not going t
 026301          16.char & 377 -> mbr            ;now send data
 026302          10.1822 + u1822.xmt -> mar(wio)
 026303          m2.ios -> misc2
 026304          nop                             ;<1>
 026305          nop                             ;<2>
 026306          io.ret -> upc                   ;<3> return
 026307          nop                             ;<4>

                ;Here after sending out the last byte.  Clear the LASTBIT (even tho i
```

```
        ;not have been set if this wasn't EOM) flag of the hardware CSR and g
        ;DONE.

        c18o.last:
026310   base -> g0.temp
026311   g0.temp & ~17 -> base              ;over to input side
026312   10 & ~h.lstbit -> 10, mbr          ;clear LAST-BIT
026313   110 + u1822.csr -> mar(wio)
026314   m2.ios -> misc2
026315   g0.temp -> base                    ;<1>
026316   c18o.hib -> 17.next                ;<2> high byte next, if any
026317   c18.done -> upc                    ;<3> finish this up (many cycles spen
026320    io.ret -> 11.ret1                 ;<4> and return when done
```

```
        ;APR, Microcall:  activate process.
        ;Entry point for APRs to C18.  Enter with G6.TEMP pointing
        ;to DCB and BASE=DCB.RB.  Set up DCB back-pointer to effect
        ;binding of a process to an I/O device.  Thus creating an I/O
        ;process.  The DCB back-pointer will be used to verify
        ;subsequent u-calls to the particular device driver from
        ;its associated process.

        c18i.apr:

026321   g6.temp -> temp                    ;copy DCB pointer
026322   temp -> 12.dcb                     ;set up in local register.

026323   100000 -> 10.csr                   ;Clear interface flags.
026324   0 -> 13.iocb                       ;No input IOCB.
026325   c18i.hib -> 17.next

026326   c18o.hib -> 117

026327   10.csr ! (h.rcvrst!h.xmtrst) -> mbr ;Set reset bits then
026330   110 + u1822.csr -> mar(wio)        ;write new control into CSR
026331   m2.ics -> misc2
026332   0 -> 112                           ;<1> interface and clear the
026333   dd.ret -> upc                      ;<2> done
026334   0 -> 113                           ;<3> output DCB and IOCB ptrs.

        c18o.apr:

026335   g6.temp -> temp                    ;copy DCB pointer
026336   dd.ret -> upc                      ;done
026337   temp -> 12.dcb                     ;set up in local register.


        ;DPR, Microcall:  delete process.
        ;Entry point for DPRs to C18.  Enter with the usual:
        ;G6.TEMP=DCB and BASE=DCB.RB.  Any current I/O is halted by
        ;discarding the active IOCB then resetting and disabling the
        ;appropriate device.

        c18.dpr:
026340   12.dcb -> temp                     ;Set up test of DCB back ptr.
026341   g6.temp - temp ls                  ;Is it the correct pointer?
026342   ifnot zero crash -> upc            ;If it's not, issue error msg.
026343    ifnot zero ioerr.rbmu -> temp ; ++ Register block messed up

026344   base -> g0.temp                    ;get to input side
026345   g0.temp & ~17 -> base
026346   0 -> 10, mbr                       ;kill both sides
026347   110 + u1822.csr -> mar(wio)
026350   m2.ios -> misc2
026351   dd.ret -> upc                      ;<1>
026352   nop                                ;<2>, lots more in NMFS

        ;Entry point for PDVs.  Enter with micro-device driver's base
        ;in BASE and the DCB in G6.TEMP.  Attempt to get an IOCB
        ;signalling start of I/O.  Then enable the device, specified in
        ;G2.TEMP, for transfers.
```

```
        c18.pdv:
026353   12.dcb -> temp                    ;Must confirm DCB.
026354   g6.temp - temp ls                 ;Is it our DCB?
026355   ifnot zero crash -> upc           ;No, go trap.
026356    ifnot zero ioerr.rbmu -> temp ; ++ Register block messed up

026357   c18.start -> upc                  ;try to start IO
026360    dd.ret -> l1.ret1
```

```
                ;XDV, Microcall:  execute device function.
                ;Entry point for XDVs.  Enter with G6.TEMP=DCB, BASE=DCB.RB.
                ;G0.TEMP contains the command to be executed.  Where a response
                ;is required, it will be passed in the accumulator, AC (=Z,NZ).

                c18.xdv:
 026361          12.dcb -> temp                  ;Run the usual check on the
 026362          g6.temp - temp ls               ;back pointer to the DCB.
 026363          ifnot zero crash -> upc         ;Test failed.  Give error msg.
 026364           ifnot zero ioerr.rbmu -> temp

 026365          c18xdv.tbl -> temp
 026366          io.xdv -> upc
 026367          c18tbl.len -> mar


                c18xdv.tbl:

 026370          c18xdv.reset -> upc             ; (0) RESET
 026371          c18xdv.raise -> upc             ; (1) RAISE
 026372          c18xdv.lower -> upc
 026373          c18xdv.status -> upc
 026374          c18xdv.iloop -> upc
 026375          c18xdv.unloop -> upc

 000006         c18tbl.len == (.-c18xdv.tbl) ;Calculate table length.
```

```
        ;Device functions.

        c18xdv.reset:                       ;Here to reinitialize device.

026376   10 ls                              ;is sign bit set?

026377   g6.temp & ~17 -> base              ;switch to input regs, always

026400   ifnot neg 10 ! h.xmtrst -> mbr              ;reset whichever side
026401   if neg 10 ! h.rcvrst -> mbr                 ;input sign has sign bit set

026402   110 + u1822.csr -> mar(wio)
026403   m2.ios -> misc2

026404   g6.temp -> base                    ;[1] restore base (no actual change h
026405   13.iocb ls                         ;an IOCB?
026406   if zero dd.ret -> upc              ;if not, we're done
026407   ifnot zero iodd.nput -> g6.temp ;tell EDONE not to cause an interrup
026410   c18.edone -> upc                   ;flush the IOCB in flight
026411    dd.ret -> 11.ret1
```

```
                c18xdv.raise:                   ;Here to raise IMP Ready line.
026412           g6.temp & ~17 -> base
026413           10.csr ! h.imprdy -> 10.csr,mbr ;Set IMP Ready flag then

                c18xdv.ios:
026414           110 + u1822.csr -> mar(wio)    ;write new control word into
026415           m2.ios -> misc2                ;CSR.
026416           dd.ret -> upc                  ;[1] Return.
026417            nop

                c18xdv.lower:                   ;Here to drop IMP Ready line.
026420           g6.temp & ~17 -> base
026421           c18xdv.ios -> upc
026422           10.csr & ~h.imprdy -> 10.csr,mbr ;Clear the IMP Ready flag then

                c18xdv.status:                  ;Here to read CSR.
026423           g6.temp & ~17 -> base
026424           110 + u1822.csr -> mar(rio)
026425           m2.ios -> misc2
026426           nop                            ;[1]
026427           mbr -> temp                    ;save CSR we just read
026430           10.csr -> mbr                  ;then turn off any error bitrs
026431           110 + u1822.csr -> mar(wio)
026432           m2.ios -> misc2
026433           rb.main -> base                ;[1] MAIN's register set.
026434           dd.ret -> upc                  ;Return to caller with
026435            temp -> 13.a                  ;status stored into AC.

                c18xdv.iloop:                   ;Here to cross-patch interface.
026436           g6.temp & ~17 -> base
026437           10.csr ! h.xpatch -> 10.csr,mbr ;Set cross-patch bit in CSR and

                c18xdv.either:
026440           110 + u1822.csr -> mar(wio)    ;write new CSR word.
026441           m2.ios -> misc2
026442           c18xdv.reset -> upc            ;[1] Go reset interface then
026443            g6.temp -> base

                c18xdv.unloop:                  ;Here to uncross-patch.
026444           g6.temp & ~17 -> base
026445           c18xdv.either -> upc
026446           10.csr & ~(h.imprdy!h.xpatch) -> 10.csr ;Clear Cross-patch,
```

```
                ;C18.DONE and C18.DONEEOM, subroutines, level 1
                ;
                ;Gets rid of the current IOCB setting the EOM bit, as necessary. Then
                ;to C18.START to try to get a new IOCB going.

                ;Enter at EDONE with G6=which IODD you want (IODD.PUT, IODD.IPUT, IOD
                ;Enter at DONEEOM and DONE to always go to IODD.PUT.

                c18.edone:
026447           c18.done0 -> upc
026450           1 -> g0.temp

                c18.doneeom:
026451           iodd.put -> g6.temp
026452           c18.done0 -> upc
026453           200 -> g0.temp                 ;set the 200 bit

                c18.done:
026454           iodd.put -> g6.temp
026455           0 -> g0.temp

                c18.done0:
026456           13.iocb + iocb.flags -> mar(r) ;set or clear the EOM bit
026457           ~(200 ! flags.ccode) -> g1.temp   ;[1]
026460           nop                            ;[2]
026461           g1.temp & mbr -> mbr           ;turn the bit off
026462           mbr ! g0.temp -> mbr           ;turn it on or off
026463           13.iccb + iocb.flags -> mar(w) ;write it back
026464           nop                            ;[1]
026465           nop                            ;[2]

026466           13.iccb + iocb.size -> mar(r)  ;fetch the size
026467           nop                            ;[1]
026470           0 -> g0.temp                   ;[2] get a nice zero for later
026471           15.cntr - mbr -> mbr           ;compute -<bits transferred>
026472           g0.temp - mbr -> mbr           ;write positive version back to IOCB
026473           13.iocb + iocb.size -> mar(w)
026474           12.dcb -> temp                 ;[1] set up for IODD.PUT call
026475           0 -> 13.iocb                   ;[2] and clear out current IOCB
026476           g6.temp -> upc                 ;get rid of it (go to right IODD.PUT)
026477           c18.start -> g6.temp           ;IODD.PUT returns through G6, START t
```

```
        ;C18.START, Subroutine, Level 1
        ;
        ;Here to try to start a new transfer.  In the normal case, if we suce
        ;turn on the enable bit for the corresponding side and return through
        ;If we can't get an IOCB, we disable the side's interrupts.  One weir
        ;if we get an IOCB and the left-off address in L7.NEXT is -1, we assu
        ;an input transfer which was in need of an extra padding word, we do
        ;L7 to the high byte routine, and start again.

        c18.start:
026500   13.iocb ls                          ;something already?
026501   ifnot zero l1.ret1 -> upc           ;yes, return
026502   if zero l2.dcb -> temp              ;Else, try to start one by
026503   iodd.get -> upc                     ;getting an IOCB (if any)
026504    (.+1) -> g6.temp                   ;from the GET queue.

026505   if zero c18.start.off -> upc        ;if we can't, shut off the side
026506   ifnot zero g0.temp -> temp          ;Copy the IOCB pointer into
026507   temp -> 13.iocb                     ;appropriate local register.

026510   13.iocb + iocb.addr -> mar(r)       ;Read the buffer address.
026511   nop                                 ;[1]
026512   nop                                 ;[2]

026513   13.iocb + iocb.size -> mar(r)       ;Read the buffer's size.
026514   mbr -> 14.addr                      ;[1] Save addr before MBR changes
026515   nop                                 ;[2]
026516   mbr -> 15.cntr ls                   ;Should be nonzero.
026517   if zero crash -> upc                ;Signal an error if zero.
026520    if zero ioerr.ilts -> temp         ;++ Illegal tranfer size ++

026521   15.cntr & 17 ls                     ;Is it a multiple of 16?
026522   ifnot zero crash -> upc             ;No, so send error msg.
026523    ifnot zero ioerr.ilts -> temp

026524   17.next + 1 ls                      ;is this the funny situation?
026525   ifnot zero c18.start.on -> upc      ;if not, start the xfer normally

        ;Here when L7.NEXT was -1. Because we have only L1, and we need to ca
        ;C18I.STORE, we must save it (ARG!)

026526   if zero 100000 -> 16.char           ;get the padding
026527   l1.ret1 -> temp
026530   temp -> g6.temp
026531   c18i.store -> upc                   ;write the padding
026532   (.+1) -> l1.ret1                    ;return to here, please

026533   c18i.hib -> 17.next                 ;assume high byte next
026534   g6.temp -> temp
026535   c18.doneeom -> upc                  ;finish up the last one, start again
026536    temp -> l1.ret1                    ;restore l1.ret1 through which DONE w
```

```
                ;Here to turn the transfer on.

                c18.start.on:
026537           10 ls                          ;is sign bit set?
026540           if neg c18.ston.in -> upc

                c18.ston.out:
026541           ifnot neg base -> g0.temp
026542           g0.temp & ~17 -> base          ;switch to other side
026543           10 ! h.xmtien -> 10, mbr
026544           110 + u1822.csr -> mar(wio)
026545           m2.ios -> misc2
026546           g0.temp -> base                ;<1>
026547           11.ret1 -> upc                 ;<2>
026550           nop                            ;<3>

                c18.ston.in:
026551           10 ! h.rcvien -> 10, mbr       ;if yes, L0 must be CSR, set up value
026552           110 + u1822.csr -> mar(wio)    ;get address from L10 (really output
026553           m2.ios -> misc2
026554           nop                            ;<1>
026555           11.ret1 -> upc                 ;<2>
026556           nop                            ;<3>

                ;Here to turn the transfer off.

                c18.start.off:
026557           10 ls                          ;is sign bit set?
026560           if neg c18.stoff.in -> upc

                c18.stoff.out:
026561           ifnot neg base -> g0.temp
026562           g0.temp & ~17 -> base          ;switch to other side
026563           10 & ~h.xmtien -> 10, mbr
026564           110 + u1822.csr -> mar(wio)
026565           m2.ios -> misc2
026566           g0.temp -> base                ;<1>
026567           nop                            ;<2>
026570           11.ret1 -> upc                 ;<3>
026571           nop                            ;<4>

                c18.stoff.in:
026572           10 & ~h.rcvien -> 10, mbr      ;if yes, L0 must be CSR, set up value
026573           110 + u1822.csr -> mar(wio)    ;get address from L10 (really output
026574           m2.ios -> misc2
026575           nop                            ;<1>
026576           nop                            ;<2>
026577           11.ret1 -> upc                 ;<3>
026600           nop                            ;<4>
```

```
026601  unext = .
        # 24 "m.mic"
```

```
# 1 "m52.mic"
```

```
        ;Known bugs and deficiencies:

        ;       not finished, not debugged
        ;       getq overrun indication to iocb in put q
        ;       usart output overrun indication

        ;Completion codes:

000001  m52.cc.error  == 1                  ;software abort
000002  m52.cc.ovrun == 2                   ;overrun
000002  m52.cc.undrun == 2
000003  m52.cc.crc    == 3                  ;crc error
000004  m52.cc.abort == 4                   ;abort sequence
000005  m52.cc.abc    == 5                  ;left over bits in frame

        ;XDV functions supported by both the input and output sides.

        ;(0) No-op
        ;(1) Abort: clear any current iocb, send abort seq, then continue nor
        ;(2) Disable 2652 and clear software, like abort but doesn't continue
        ;(3) Disable and then enable 2652 and clear software
        ;(4) retreive modem status bits into B register
        ;(5) set modem status bits from B register

        ;Register block allocations: set by configuration file from cassette.

000020  m52.in2out == 20 ; rb.input + k -> rb.output, this is K
```

```
        ; IO Timing:
        ;
        ;       CSR  read       2 iostrobes
        ;       CSR  write      1 iostrobe
        ;       MPCC read       3 iostrobes
        ;       MPCC write      2 iostrobes
        ;
        ;
        ; Interrupt release after last iostrobe:
        ;
        ;       CSR             6 microinstructions
        ;       MPCC            8 microinstructions

        ;Register definitions. "i" = input, "o" = output, "b" = both

000000  10.dcb     == 10                   ;(b) pointer back to DCB
000003   13.lobyte == 13                   ;(i) lo byte buffer, -1=none
000003   13.hibyte == 13                   ;(o) hi byte buffer, -1=none
000004  14.iocb    == 14                   ;(b) pointer to IOCB, 0=none
000005  15.addr    == 15                   ;(b) next word to go out
000006  16.left    == 16                   ;(b) number of words left to output
000007  17.next    == 17                   ;(b) UPC to use to service next chara

000013  113.flags == 113                   ;(b) holds iocb flags

020000   m52.iocb.f == 1_13.                    ; iocb wants end of frame int
010000   m52.iocb.o == 1_12.                    ; iocb starts on odd byte
000200   m52.iocb.eof == 1_7.                   ; iocb holds end of frame

000014  114.cr     == 114                  ;(i) pointed to by L16 of both (i) an
                                           ;(o) unused
000015  115.usart == 115                   ;(b) USART IO address associated w/ t
000016  116.crptr == 116                   ;(b) pointer to register containing C
                                                ; extra bits used for our pur
040000   m52.crptr.msync = 1_16                 ; 1 => device is on an msync

000017  117.char  == 117                   ;(b) holds current character (sometim
```

```
        ;Configuration

000007  m52i.mti  == 7.                    ;device number for input, mti
000010  m52o.mti  == 8.                    ;device number for output, mti

000013  m52i.mii  == 11.                   ;device number for input, mii
000014  m52o.mii  == 12.                   ;device number for output, mii

        ; MSYNC board definitions:

        ;       (for future reference.  not used at the moment.)

        ;        msync.iov    == 20000!(MSYNC_7) ;the base I/O interrupt vect

        ;        msync.mpcc00 == msync.ioa       ;address of the MPCC for SLC
        ;        msync.delta  == 10              ;the distance between device
        ;        msync.csr00  == msync.ioa!400   ;address of the CSR for SLC
        ;        msync.xios   == msync.ioa!1000  ;address of the "external pu
        ;        msync.leds   == msync.ioa!1400  ;address of the 20-bit LED r

        ; offset and bit definitions for the SLC CSR

        ;   a) the address offset from the SLC's MPCC base address
000400           mpcc.slcsr == 400

        ;   b) the write-only side of the CSR, as defined by the MSYNC board
000001           slcsr.enbl  == 1          ;MPCC enable
000002           slcsr.txien == 2          ;MPCC transmitter interrupt enable
000004           slcsr.mmode == 4          ;MPCC maintenance mode (x-patch)
000010           slcsr.rts   == 10         ;RTS to the main db-conn
000020           slcsr.dtr   == 20         ;DTR to the main db-conn
000040           slcsr.outa  == 40         ;general output "a" to secondary db-c
000100           slcsr.outb  == 100        ;general output "b" to secondary db-c
000200           slcsr.outc  == 200        ;general output "c" to secondary db-c


000003           slcsr.enall == (slcsr.enbl ! slcsr.txien)       ;set by micro
000374           slcsr.macro == 374                              ;set by macro

        ;   c) the read-only side of the CSR of an MSYNC board
000001           slcsr.rxsa  == 1          ;MPCC new receiver status available
000002           slcsr.cts   == 2          ;CTS from the main db-conn
000004           slcsr.dcd   == 4          ;DCD from the main db-conn
000010           slcsr.ina   == 10         ;general input "a" from the secondary
000020           slcsr.inb   == 20         ;general input "b" from the secondary
000040           slcsr.inc   == 40         ;general input "c" from the secondary
000100           slcsr.ind   == 100        ;general input "d" from the secondary
000200           slcsr.ine   == 200        ;general input "e" from the secondary
000377           slcsr.stat  == 377        ;all of the above

        ;   d) multiple symbols definitions for our convenience

000004           slcsr.xpatch == slcsr.mmode
```

```
                ;   e) write only half of CSR supplied by an MTI board.  unfortunatel

000001                  mticsr.rxenbl == 1
000002                  mticsr.txenbl == 2
000004                  mticsr.enable == 4
000010                  mticsr.xpatch == 10
000020                  mticsr.rts    == 20
000040                  mticsr.dtr    == 40

000003                  mticsr.enall == (mticsr.rxenbl ! mticsr.rxenbl ! mticsr.txenb
000070                  mticsr.macro  == 70

                ;   f) read only half of CSR supplied by MTI.

000001                  mticsr.dsr    == 1
000002                  mticsr.dcd    == 2
000004                  mticsr.cts    == 4
000007                  mticsr.stat   == 7
```

```
        ; the S2652 register assignments are:

000000          mpcc.rdsr    == 0        ;the receiver data and status registe
000002          mpcc.tdsr    == 2        ;the transmitter data and status regi
000004          mpcc.pcsar   == 4        ;the control/sync/address register
000006          mpcc.pcr     == 6        ;the parameter control register

000400          mpcc.csr     == 400      ;the CSR on an MSYNC board
000100          mpcc.csr.mti == 100      ;the CSR on an MTI board

        ;   RDSR - receiver data/status register

000377          rdsr.rxdb    == 377      ;receiver data byte
000400          rdsr.rsom    == 400      ;start-of-message was received
001000          rdsr.eom     == 1000     ;end-of-message was received
002000          rdsr.rabga   == 2000     ;received abort or go-ahead
004000          rdsr.ror     == 4000     ;receiver overrun detected
070000          rdsr.abc     == 70000    ;receiver assembled-bit-count
100000          rdsr.err     == 100000   ;OR of receiver errors

176000          rdsr.badstart = (rdsr.rabga!rdsr.ror!rdsr.abc!rdsr.err)
177000          rdsr.allstop ==  (rdsr.err!rdsr.rabga!rdsr.ror!rdsr.abc!rdsr.

        ;   TDSR - transmit data/status register

000377          tdsr.txdb    == 377      ;transmit data byte
000400          tdsr.tsom    == 400      ;transmit start-of-message
001000          tdsr.teom    == 1000     ;transmit end-of-message
002000          tdsr.abort   == 2000     ;transmitter abort request
004000          tdsr.tga     == 4000     ;transmitter go-ahead enable
100000          tdsr.err     == 100000   ;transmitter error detected
        ;
        ;   PCSAR - paramter control sync/address register

000377          pcsar.sar    == 377      ;sync/address byte
003400          pcsar.ecm    == 3400     ;error control mode (CRC select)
004000          pcsar.idle   == 4000     ;idle/fill character select
010000          pcsar.sam    == 10000    ;secondary address mode
020000          pcsar.ssga   == 20000    ;strip sync/go ahead enable
040000          pcsar.proto  == 40000    ;protocol select (0==BOP;1==BCP)
100000          pcsar.apa    == 100000   ;all parties address enable
        ;
        ;   PCR - parameter control register:

003400          pcr.rxcl     == 3400     ;receiver character length
004000          pcr.rxcle    == 4000     ;receive char length enable
010000          pcr.txcle    == 10000    ;transmit char length enable
016000          pcr.txcl     == 16000    ;transmitter character length
```

```
026601    uram unext

          m52i.0:                          ;function dispatch - input side

026601     m52i.apr -> upc
026602     m52.dpr -> upc
026603     m52.xdv -> upc
026604     m52.pdv -> upc

          m52o.0:

026605     m52o.apr -> upc
026606     m52.dpr -> upc
026607     m52.xdv -> upc
026610     m52.pdv -> upc
```

```
        ;Input interrupt level - input side

        m52i.int:                                       ;here on microinterupts, BASE
                                                        ;RB.M52I.n where n=unit numbe
026611   115.usart + mpcc.rdsr -> mar(rio) ls           ;always read the character
026612   if zero crash -> upc                           ;if we get an interrupt witho
026613   if zero ioerr.uint -> temp                     ; ++ unexpected interrupt
026614   do.ios -> upc                                  ; above RELIES on mpcc.rdsr=0
026615   (.+1) -> 11.ret1                               ;<1,2>
026616   mbr -> 117.char                                ;<3> remember character in L1
026617   14.iocb ls                                     ;<4> is there a current IOCB?
026620   if zero (io.ret-1) -> upc                      ;<5> if not, skip dispatch
026621   ifnot zero 17.next ls                          ;<6> if xfer in progress, dis
                                                        ;dispatch on L7.NEXT, 0 goes
026622   ifnot zero 17.next -> upc
026623   117.char & (rdsr.eom!rdsr.rabga) ls

        ;Here before first character when an IOCB is present.

        m52i.new:                                       ;here before first character

026624   117.char & rdsr.rsom LS                        ;Do we have a Start of Messag
026625   if zero io.ret -> upc                          ;No, then leave.

026626   ifnot zero 113.flags & m52.iocb.o ls           ;start at odd byte location i
026627   if zero m52i.low -> upc                        ;no, buffer starts normally.
026630   ifnot zero 15.addr -> mar(r)                   ;else read first word of buff
026631   377 -> 13.lobyte                               ;[1] a mask
026632   117.char & (rdsr.eom ! rdsr.rabga) ls          ;[2] test as if from m52i.int
026633   m52i.high -> upc                               ;this byte is the low one.
026634   13.lobyte & mbr -> 13.lobyte                   ;all fixed up now.
```

```
        ;Input machine states.

        m52i.low:                               ;L17.CHAR is the low byte

026635   ifnot zero m52i.eom -> upc
026636   if zero 117.char & 377 -> temp ;copy this char to become low byte
026637   temp -> 13.lobyte

026640   16.left - 8. -> 16.left ls     ;count down
026641   ifnot zero io.ret -> upc       ;the fast way out
026642   m52i.high -> 17.next           ;nominally, next state

026643   15.addr -> mar(r)              ;if full, get last word.
026644   (377_10) -> g0.temp            ;[1] a mask
026645   nop

026646   g0.temp & mbr -> mbr           ;remove old low byte
026647   13.lobyte ! mbr -> mbr         ;put in new one
026650   15.addr -> mar(w)              ;put word back
026651   14.iocb -> temp
026652   temp -> 117.char
026653   m52.done -> upc                ;the slow way
026654   (.+1) -> 12.ret2
026655   14.iocb ls                     ; did we get a new iocb?
026656   ifnot zero io.ret -> upc       ; if so, all is cool

026657   if zero 113.flags ! m52.cc.ovrun -> mbr        ; else set one last f
026660   io.ret -> upc
026661   117.char + iocb.flags -> mar(w)                ;[0] ([1,2] at io.ret

        ;Here for the high byte (in L17.CHAR). Low byte buffered in L3.LOBYTE

        m52i.high:

026662   ifnot zero m52i.eom -> upc
026663   if zero 117.char & 377 -> mbr  ;L17.CHAR is the high byte
026664   13.lobyte ! s16mbr -> mbr      ;set up to write the word to macromem
026665   15.addr -> mar(w)              ;write the word to macromemory
026666   15.addr + 1 -> 15.addr         ;[1] increment pointer
026667   16.left - 8. -> 16.left ls     ;[2] decrement count and check it (be

026670   ifnot zero io.ret -> upc
026671   ifnot zero m52i.low -> 17.next ;next thing in will be low byte

026672   m52.done -> upc                ;upc dangles onto next page
026673   io.ret -> 12.ret2
```

```
        ;Here if this byte marked end of frame.
        ;to check and decode error conditions, given that we know we are goin

        m52i.eom:

026674   117.char & rdsr.eom LS
026675   113.flags & m52.iocb.eof -> 113.flags  ;perhaps there is good news
026676   0 -> temp
026677   if zero m52i.cc.eom -> upc             ;if this bit is on

026700   ifnot zero 117.char & rdsr.abc LS      ;ok to look at these
026701   ifnot zero m52.cc.abc -> temp

026702   117.char & rdsr.ror LS
026703   ifnot zero m52.cc.ovrun -> temp

026704   117.char & rdsr.err LS
026705   ifnot zero m52.cc.crc -> temp

        m52i.cc.eom:

026706   117.char & rdsr.rabga LS
026707   ifnot zero m52.cc.abort -> temp

026710   m52.done.ret -> upc                    ; post this to macroland,
026711   113.flags ! temp -> 113.flags
```

```
                ;Interrupt level - output side

                m52o.int:

026712           115.usart ls                   ;make sure this isn't a bogus interru
026713           ifnot zero 14.iocb ls          ;and there is a current IOCB
026714           if zero m52o.idle -> upc       ;if not, unwanted interrupt
026715           ifnot zero 17.next ls          ;if there is, dispatch
026716           ifnot zero 17.next -> upc      ;if L7.NEXT=0, goto M520.NEW
026717           if zero m52o.new -> upc
026720           nop


                m52o.idle:

026721           crash -> upc
026722           ioerr.uint -> temp
```

```
        ;Output machine states:

        ;Here at the start of a new IOCB for output.

        m52o.new:

 026723   113.flags & m52.iocb.o ls      ;bit not zero ==> start at odd byte
 026724   ifnot zero 15.addr -> mar(r)   ;if so, read the word
 026725   if zero m52o.low -> upc        ;[1] else send low byte first
 026726   nop                            ;[2] pfui, nothing to do
 026727   s16mbr -> 13.hibyte            ;send the high byte now.


        ;Here to transmit the high byte.

        m52o.high:

 026730   m52o.low -> 17.next            ;will send the low byte next
 026731   m52o.send.mbr -> upc           ;send the high one now
 026732   13.hibyte & 377 -> mbr         ;ready to send this byte


        ;Here to fetch new word and transmit the low byte.

        m52o.low:

 026733   m52o.high -> 17.next           ;say we need to send the high byte
 026734   15.addr -> mar(r)              ;fetch it
 026735   15.addr + 1 -> 15.addr         ;[1] step to next
 026736   s16mbr -> 13.hibyte            ;[2] set up the hi byte
 026737   mbr -> 117.char                ;output this byte

        m52o.send:

 026740   117.char & 377 ->  mbr         ;low 8 bits only!

        m52o.send.mbr:

 026741   16.left - 8. -> 16.left LS     ;count down this byte
 026742   if neg m52o.eob -> upc         ;low prob case done out of line
 026743   ifnot neg 115.usart + mpcc.tdsr -> mar(wio)

        m52o.send.mar:

 026744   m2.ios -> misc2                ;[1]
 026745   m2.ios -> misc2                ;[2]<1>
 026746   (io.ret-5) -> upc              ;<3> return and dismiss
 026747   nop                            ;<4>
```

```
        ;Here when iocb is exhausted.

        m52o.eob:                                       ;the else branch from above

026750   16.left + 8. -> 16.left                        ;compensate for extra decreme
026751   113.flags & m52.iocb.eof LS
026752   if zero m52o.more -> 17.next                   ;next state if not end of fra

026753   ifnot zero m52o.eom -> 17.next                 ;next state if end of frame n
026754   ifnot zero tdsr.teom -> g0.temp
026755   ifnot zero g0.temp ! mbr -> mbr                ;tell 2652 to transmit eom

026756   m52o.send.mar -> upc                           ;rejoin main sequence
026757   115.usart + mpcc.tdsr -> mar(wio)              ;start byte on its way


        ;Here to send another iocb in same frame, or else...

        m52o.more:

026760   14.iocb -> temp                                ;save this so
026761   temp -> 117.char                               ;we can set a bit later
                                                        ;(but not set a spell later o

026762   m52.done -> upc                                ;even though it is on
026763   (.+1) -> 12.ret2                               ;the put q

026764   14.iocb LS                                     ;did we find another iocb?
026765   ifnot zero m52o.new -> upc                     ;yes, keep on sending

026766   if zero 113.flags ! m52.cc.undrun -> 113.flags, mbr
026767   117.char + iocb.flags -> mar(w)                ;could we just send idle char

026770   m52.tx0 -> upc                                 ;[1] anyway, for now, turn of
026771    io.ret -> 11.ret1                             ;[2]
```

```
        ;Here to turn off transmitter and send iocb to macroland, from interr

        m52o.eom:

026772   m52.done.ret -> l1.ret1                                ;[1]

        m52.tx0:

026773   116.crptr & m52.crptr.msync ls
026774   116.crptr -> base                                      ;[2]
026775   base -> g0.temp
026776   if zero 10 & ~mticsr.txenbl -> 10, mbr
026777   ifnot zero 10 & ~slcsr.txien -> 10, mbr
027000   g0.temp -> base

        m52.csr.wio:                                           ; as below, but msync bit alr

027001   if zero 115.usart + mpcc.csr.mti -> mar(wio)
027002   do.ios -> upc
027003   ifnot zero 115.usart + mpcc.csr -> mar(wio)

        m52.write.csr:                                         ; subroutine, level 1

027004   m52.csr.wio -> upc
027005   116.crptr & m52.crptr.msync ls
```

```
        ;Here on APRs to a M52. Clear the interface and enable the 2652, then
        ;try to start io going. Enter with BASE=DCB.RB from macrocode and G6.
        ;pointing to the DCB we're APR'ing.

        m52o.apr:

027006   0 -> 10.dcb                        ;clear back pointer to DCB to encoura
                                            ;a crash when referencing it
027007   0 -> 14.iocb                       ;no current IOCB
027010   0 -> 17.next                       ;encourage a crash if use co-routine

027011   base -> g0.temp
027012   g0.temp - m52.in2out -> base       ;switch to input side

027013   0 -> 10.dcb                        ;do same initialization
027014   0 -> 14.iocb
027015   0 -> 17.next

027016   g0.temp -> base                    ;restore base
027017   0 -> mbr                           ;disable the device

027020   m52.write.csr -> upc
027021   (.+1) -> 11.ret1

                                            ;<1> delay to make strobe at least 1
027022   g6.temp -> temp                    ;<1> set up back pointer to DCB
027023   m52.xdv.ena -> upc                 ;merge with enable and clear code
027024   temp -> 10.dcb                     ;set back pointer

        m52i.apr:

027025   g6.temp -> temp                    ;set up back pointer to DCB
027026   temp -> 10.dcb                     ;set back pointer
027027   base -> g0.temp                    ;save this base
027030   g0.temp + m52.in2out -> base       ;go look at other half
027031   10.dcb ls                          ;is it initialized
027032   ifnot zero dd.ret -> upc           ;if so, assume all ok, all done
027033   g0.temp -> base                    ;restore base anyway

027034   crash -> upc                       ;crash if wrong half
027035   ioerr.aprwh -> temp                ;was apr'd first
```

```
        ;Here with standard args (G6=DCB, BASE=base) on a DPR of the process

        m52.dpr:
027036   10.dcb -> temp                     ;make sure we point back to the DCB
027037   g6.temp - temp ls
027040   ifnot zero crash -> upc            ;if they don't match, bomb
027041   ifnot zero ioerr.rbmu -> temp      ; ++ Register blocks messed up

027042   14.iocb ls                                 ;any current IOCB?
027043   ifnot zero m52.edonestop -> upc            ;yes, error it out w/o gettin
027044   ifnot zero (.+1) -> 12.ret2
027045   14.iocb ls                         ;make sure we got rid of it (&&)
027046   ifnot zero crash -> upc            ;if it isn't...
027047   ifnot zero ioerr.csii -> temp      ; ++ cant shake IOCB when IDLE

027050   0 -> mbr                           ;turn off hardware
027051   m52.write.csr -> upc               ;this does the return for us
027052   dd.ret -> 11.ret1

        ;Here on XDV. BASE and G6 as usual. G0 contains user's AC (see
        ;IO316.ACT).

        m52.xdv:

027053   m52.xdv.table -> temp
027054   iodd.xdv -> upc
027055   (m52.xdv.size) -> mar

        m52.xdv.table:
027056   dd.ret -> upc                      ;(0) no-op
027057   m52.xdv.abort -> upc               ;(1) abort and continue
027060   m52.xdv.dis -> upc                 ;(2) disable and reset
027061   m52.xdv.ena -> upc                 ;(3) disable, enable and reset
027062   m52.xdv.stat -> upc                ;(4) retrieve status
027063   m52.xdv.set -> upc                 ;(5) set status
000006  m52.xdv.size == (.-m52.xdv.table)
```

```
        m52.xdv.ena:                        ;here on enable and reset, also from

027064   m52.zusart -> upc                  ;clear the 2652
027065   (.+1) -> 12.ret2

        m52.xdv.ret:

027066   14.iocb ls                         ;an IOCB?
027067   if zero dd.ret -> upc              ;no, return now

        m52.xdv.err:                        ; branch here with nonzero condition

027070   ifnot zero m52.edone -> upc        ;yes, error flush it
027071   dd.ret -> 12.ret2                  ;then return to caller

        m52.xdv.abort:                      ;here to abort and continue

027072   14.iocb ls                         ;an IOCB?
027073   ifnot zero 17.next ls              ;any state?
027074   if zero dd.ret -> upc              ;no, return now

027075   116.crptr ls                       ;is it output device
027076   if neg m52.xdv.err -> upc          ;if not, go error out iocb <<<neg =>

                                            ;else tell 2652 to abort after this c
027077   ifnot neg (tdsr.abort ! tdsr.teom) -> mbr
027100   115.usart + mpcc.tdsr -> mar(wio)
027101   dd.ret -> 12.ret2                  ;funny subroutine calls
027102   do.ios -> upc
027103   m52.edone -> 11.ret1               ;then go error out iocb


        m52.xdv.dis:                        ;here on disable and reset

027104   m52.zusart -> upc                  ;clear the 2652
027105   (.+1) -> 12.ret2

027106   14.iocb ls                                 ;an IOCB?
027107   if zero dd.ret -> upc                      ;no, return now
027110   ifnot zero m52.edonestop -> upc            ;yes, flush it w/o getting ne
027111   ifnot zero dd.ret -> 12.ret2               ;return directly to caller
```

```
        m52.xdv.stat:

 027112   115.usart + mpcc.csr -> mar(rio)
 027113   do.ios -> upc
 027114   (.+1) -> 11.ret1
 027115   116.crptr & m52.crptr.msync ls
 027116   rb.main -> base
 027117   ifnot zero slcsr.stat-> 14.b          ;what macrocode may see
 027120   if zero mticsr.stat -> 14.b
 027121   dd.ret -> upc
 027122   14.b & mbr -> 14.b                    ;we have our little secrets


        m52.xdv.set:

 027123   116.crptr & m52.crptr.msync ls
 027124   rb.main -> base
 027125   base -> g0.temp
 027126   ifnot zero 14.b & slcsr.macro -> mbr  ;macro mask
 027127   if zero 14.b & mticsr.macro -> mbr
 027130   g0.temp -> base                       ;restore base
 027131   116.crptr -> base                     ;go to where true copy lives
 027132   ifnot zero 10 & slcsr.enall -> 10     ;keep micro controlled bits
 027133   if zero 10 & mticsr.enall -> 10
 027134   10 ! mbr -> mbr, 10                   ;make a copy in its proper pl
 027135   g0.temp -> base                       ;restore base again
 027136   m52.csr.wio -> upc
 027137   dd.ret -> 11.ret1


        ;Here on PDVs

        m52.pdv:
 027140   10.dcb -> temp                        ;make sure back pointer is ri
 027141   g6.temp - temp ls
 027142   ifnot zero crash -> upc
 027143   ifnot zero ioerr.rbmu -> temp

 027144   m52.start -> upc                      ;try to start new IO
 027145   dd.ret -> 12.ret2
```

```
        ;M52.ZUSART, Subroutine, Level 2

        ;Clears the 2652's internal registers and CSR, thus removing any
        ;crosspatch, etc.

        m52.zusart:


027146   0 -> mbr                          ;first clear everything
027147   m52.csr.wio -> upc
027150   (.+1) -> l1.ret1
                                           ;<1> delay to make strobe at least 1

027151   l16.crptr & m52.crptr.msync ls          ;<1>
027152   l16.crptr -> base
027153   base -> g1.temp
027154   ifnot zero slcsr.enbl -> 10, mbr
027155   if zero (mticsr.enable!mticsr.rxenbl) -> 10, mbr

                                           ;now re-enable reciever interrupts
027156   m2.ios -> misc2
027157   g1.temp -> base                   ;<1> delay to make strobe at least 1

027160   0 -> mbr

        ; Set the PCR to BOP, 8 bit bytes
027161   L15.usart + mpcc.pcr -> mar(wio)
027162   m2.ios -> misc2
027163   m2.ios -> misc2                    ;[1]
027164   nop                                ;[2]

        ; Set PCSAR to CRC-CCITT preset to 1's, ABORT when underrun, BOP
027165   L15.usart + mpcc.pcsar -> mar(wio)
027166   m2.ios -> misc2
027167   m2.ios -> misc2                    ;[1]
027170   nop                                ;[2]


        ; setup MPCC to send FLAG characters
027171   tdsr.tsom -> mbr
027172   L15.usart + mpcc.tdsr -> mar(wio)
027173   m2.ios -> misc2
027174   m2.ios -> misc2                   ;[1]
027175   (.+1) -> upc                      ;[2]<1>
027176   nop                               ;<2,3>    **** this must go to 8 ****

027177   l2.ret2 -> upc                    ;<4> return
027200   nop                               ;<5> [1]
```

```
                ;M52.START, Subroutine, level 2

                ;Call .START with G6 pointing at the DCB and BASE at the right regist
                ;block and it will try to start new IO on the device. This subr is no
                ;called on APR's and PDV's for the device, as well as on IO completio

                ;First, it checks for DCB.IOCB (cached in L4.IOCB)
                ;to be non-zero. If it's non-zero, there's
                ;already an IO going on and we return immediately. If it's zero, we c
                ;IODD.GET in an attempt to get a new IOCB.

                m52.start:
027201           l4.iocb ls                     ;is there already an IOCB?
027202           ifnot zero l2.ret2 -> upc      ;yes, return now

027203           if zero l0.dcb -> temp         ;set up for IODD.GET call, put DCB in
027204           iodd.get -> upc                ;no, try to get a new one
027205           (.+1) -> g6.temp

027206           if zero m52.stout -> upc       ;if none to be had, check output half

                ;Here (with dangling UPC change) when there is an IOCB (returned in G
                ;to be output. IO.GET has already setup DCB.IOCB and removed it from
                ;get queue, etc. We now set up the register block to reflect the IOCB
                ;and then return.

027207           ifnot zero g0.temp -> temp     ;copy the IOCB address into L4.IOCB
027210           temp -> l4.iocb

027211           l2.ret2 -> temp
027212           m52.restart -> upc             ;"restart" the io which merely sets u
027213           temp -> l1.ret1                ;the register block from the IOCB

                m52.stout:
027214           l16.crptr ls                   ;is this transmit half?
027215           if neg l2.ret2 -> upc          ;if not exit now.

027216           ifnot neg l2.ret2 -> temp
027217           m52.tx0 -> upc                 ;if so, turn off transmit interrupts
027220           temp -> l1.ret1
```

```
        ;M52.RESTART, subroutine, Level 1

        ;Subroutine to "restart" a transfer. Merely copies the relavent IOCB
        ;into the register block cache and points L7.NEXT to 0 (expecting
        ;new frame). Enter with L4.IOCB pointing up at the IOCB to get the
        ;information from.


        m52.restart:

027221   14.iocb ls                          ;make sure we're called with legit ar
027222   14.iocb + iocb.addr -> mar(r)       ;read the transfer start address
027223   if zero crash -> upc                ;[1]
027224   if zero ioerr.ddiia -> temp         ;[2] ++ illegal IOCB arg

027225   14.iocb + iocb.size -> mar(r)       ;read the size
027226   mbr -> 15.addr                      ;[1] remember start addr
027227   0 -> 17.next                        ;[2] reset the interrupt routine

027230   14.iocb + iocb.flags -> mar(r)      ;read flags
027231   mbr -> 16.left ls                   ;[1] better not be zero
027232   if zero crash -> upc                ;[2]
027233   if zero ioerr.ilts -> temp

027234   mbr -> 113.flags                            ;store flags
027235   113.flags & ~flags.ccode -> 113.flags       ;clear completion code bits.

027236   16.left & 7 ls                      ;better be a multiple of 8 bits
027237   ifnot zero crash -> upc
027240   ifnot zero ioerr.ilts -> temp       ; ++ Illegal transfer size

027241   116.crptr ls
027242   if neg 11.ret1 -> upc               ;if input half return to caller

027243   ifnot neg 116.crptr & m52.crptr.msync ls
027244   116.crptr -> base                   ;if so, turn on transmit interrupts
027245   base -> g0.temp
027246   ifnot zero 10 ! slcsr.txien -> 10, mbr     ;modify mbr
027247   if zero 10 & mticsr.txenbl -> 10, mbr

027250   m52.csr.wio -> upc                  ; this will return
027251   g0.temp -> base                     ; fix base
```

```
        ;Subroutine M52.EDONE, Subroutine, Level 2

        ;Come here with L4.IOCB pointing to the current IOCB which should be
        ;given and error completion code (m52.cc.error) and removed. Enter at
        ;EDONESTOP to not let DONE (below) call START, thus preventing the
        ;DCB.IOCB window to advance to then next IOCB.

        m52.edonestop:

 027252   m52.edone0 -> upc
 027253   (.+1) -> g6.temp

 027254   116.crptr ls                        ;which side of device?
 027255   if neg 12.ret2 -> upc               ;exit if input side
 027256   12.ret2 -> temp
 027257   m52.tx0 -> upc                      ;else turn off transmit interrupt
 027260   temp -> 11.ret1                     ;then return to caller

        m52.edone:

 027261   m52.start -> g6.temp                ;tell M52.DONE0 to call START afterwa

        m52.edone0:

 027262   113.flags ! m52.cc.error -> 113.flags   ;set the completion code to

        ;fall into M52.DONE0
```

```
        ;M52.DONE0, Subroutine, Level <weird>

        ;Enter with L4.IOCB pointing to IOCB finished. Compute the size trans
        ;by converting our L6.LEFT back into bits (by multiplying by 16), sub
        ;it from IOCB.SIZE (the original max size) giving a bit count. We the
        ;IO.PUT, clear the current pointer and return to the placed pointed t
        ;G6.TEMP. Enter at M52.DONE to set G6.TEMP to M52.START, (the normal
        ;thus causing the next IOCB to be fetched.

        m52.done0:

027263   14.iocb + iocb.size -> mar(r)  ;read the IOCB max length
027264   16.left -> temp                ;[1] get a copy of LEFT
027265   113.flags & m52.iocb.f LS      ;[2]  ... while killing time, do this

027266   mbr -> g2.temp                 ;move to useful place
027267   g2.temp - temp -> g2.temp, mbr ;compute number of bits xferred
027270   14.iocb + iocb.size -> mar(w)  ;write it back
027271   10.dcb -> temp                 ;[1] set up for IODD.PUT call (DCB in
027272   ifnot zero 113.flags & m52.iocb.eof LS         ;[2] ... some more ti

027273   113.flags -> mbr               ;write flags back
027274   14.iocb + iocb.flags -> mar(w)

027275   if zero iodd.put -> upc        ;[1] put it away, maybe interrupting
027276   0 -> 14.iocb                   ;[2] we're done with this IOCB
                                        ;note that IODD.PUT will return throu
                                        ;to wherever caller told it to.

027277   ifnot zero iodd.iput -> upc    ;... force interrupt ...
027300   nop                            ;if eof and iocb.f bit is on.
                                        ;likewise returning

        m52.done.ret:
027301   io.ret -> 12.ret2

        m52.done:
027302   m52.done0 -> upc               ;enter here to set G6 to get the next
027303   m52.start -> g6.temp           ; ... by calling M52.START

027304          unext == .
        # 25 "m.mic"

000500          rnext = 500
```

A2SP	22657	ASCII.B	102
ACA	21633	ASCII.B.	142
ADD	21470	ASCII.BACKSLASH	134
ALR	22013	ASCII.BAR	174
ALS	22043	ASCII.BEL	7
ALU.CBIT	20	ASCII.BS	10
ANA	21462	ASCII.C	103
AOA	21627	ASCII.C.	143
AREG	203	ASCII.CAN	30
ARR	22013	ASCII.COLON	72
ARS	22004	ASCII.COMMA	54
ASCII..A	1	ASCII.CR	15
ASCII..B	2	ASCII.D	104
ASCII..C	3	ASCII.D.	144
ASCII..D	4	ASCII.DC1	21
ASCII..E	5	ASCII.DC2	22
ASCII..F	6	ASCII.DC3	23
ASCII..G	7	ASCII.DC4	24
ASCII..H	10	ASCII.DEL	177
ASCII..I	11	ASCII.DL	177
ASCII..J	12	ASCII.DLE	20
ASCII..K	13	ASCII.DOLLAR	44
ASCII..L	14	ASCII.E	105
ASCII..M	15	ASCII.E.	145
ASCII..N	16	ASCII.EM	31
ASCII..O	17	ASCII.ENQ	5
ASCII..P	20	ASCII.EOT	4
ASCII..Q	21	ASCII.EQ	75
ASCII..R	22	ASCII.EQUALS	75
ASCII..S	23	ASCII.ESC	33
ASCII..T	24	ASCII.ETB	27
ASCII..U	25	ASCII.ETX	3
ASCII..V	26	ASCII.EX	41
ASCII..W	27	ASCII.EXCL	41
ASCII..X	30	ASCII.EXCLAMATIO	41
ASCII..Y	31	ASCII.F	106
ASCII..Z	32	ASCII.F.	146
ASCII.0	60	ASCII.FF	14
ASCII.1	61	ASCII.FS	34
ASCII.2	62	ASCII.G	107
ASCII.3	63	ASCII.G.	147
ASCII.4	64	ASCII.GRAVE	140
ASCII.5	65	ASCII.GREATER	76
ASCII.6	66	ASCII.GS	35
ASCII.7	67	ASCII.GT	76
ASCII.8	70	ASCII.H	110
ASCII.9	71	ASCII.H.	150
ASCII.A	101	ASCII.HT	11
ASCII.A.	141	ASCII.I	111
ASCII.ACK	6	ASCII.I.	151
ASCII.AMPER	46	ASCII.J	112
ASCII.AMPERSAND	46	ASCII.J.	152
ASCII.APOSTROPHE	47	ASCII.K	113
ASCII.AS	52	ASCII.K.	153
ASCII.ASTER	52	ASCII.L	114
ASCII.ASTERISK	52	ASCII.L.	154
ASCII.AT	100	ASCII.LB	133
ASCII.ATSIGN	100	ASCII.LC	173

ASCII.LCURLY	173	ASCII.U.	165
ASCII.LEFTPAREN	50	ASCII.UA	136
ASCII.LESS	74	ASCII.UNDERSCORE	137
ASCII.LF	12	ASCII.UP	136
ASCII.LPAREN	50	ASCII.UPARROW	136
ASCII.LT	74	ASCII.US	37
ASCII.M	115	ASCII.V	126
ASCII.M.	155	ASCII.V.	166
ASCII.MINUS	55	ASCII.VT	13
ASCII.N	116	ASCII.W	127
ASCII.N.	156	ASCII.W.	167
ASCII.NAK	25	ASCII.X	130
ASCII.NOT	176	ASCII.X.	170
ASCII.NS	43	ASCII.XOF	22
ASCII.NUL	0	ASCII.XON	21
ASCII.O	117	ASCII.Y	131
ASCII.O.	157	ASCII.Y.	171
ASCII.P	120	ASCII.Z	132
ASCII.P.	160	ASCII.Z.	172
ASCII.PC	45	BADDEV	23077
ASCII.PERCENT	45	BAUD.110	62
ASCII.PERIOD	56	BAUD.1200	67
ASCII.PLUS	53	BAUD.1800	70
ASCII.POUND	43	BAUD.19200	77
ASCII.Q	121	BAUD.2400	72
ASCII.Q.	161	BAUD.300	65
ASCII.QM	77	BAUD.4800	74
ASCII.QUEST	77	BAUD.9600	76
ASCII.QUESTION	77	BLINK.ALL	17
ASCII.QUESTIONMA	77	BLINK0	1
ASCII.QUOTE	42	BLINK1	2
ASCII.R	122	BLINK2	4
ASCII.R.	162	BLINK3	10
ASCII.RB	135	BLT	22557
ASCII.RC	175	BLT2	22561
ASCII.RCURLY	175	BLT3	22571
ASCII.RIGHTPAREN	51	BREG	214
ASCII.RO	177	C18.DONE	26454
ASCII.RPAREN	51	C18.DONE0	26456
ASCII.RS	36	C18.DONEEOM	26451
ASCII.RUBOUT	177	C18.DPR	26340
ASCII.S	123	C18.EDONE	26447
ASCII.S.	163	C18.PDV	26353
ASCII.SEMI	73	C18.START	26500
ASCII.SEMICOLON	73	C18.START.OFF	26557
ASCII.SI	17	C18.START.ON	26537
ASCII.SLASH	57	C18.STOFF.IN	26572
ASCII.SO	16	C18.STOFF.OUT	26561
ASCII.SOH	1	C18.STON.IN	26551
ASCII.SP	40	C18.STON.OUT	26541
ASCII.SPACE	40	C18.XDV	26361
ASCII.STX	2	C18I	3
ASCII.SUB	32	C18I.APR	26321
ASCII.SYN	26	C18I.HIB	26157
ASCII.T	124	C18I.INT	26150
ASCII.T.	164	C18I.LAST.HARD	26222
ASCII.TILDE	176	C18I.LAST.IMPOSS	26232
ASCII.U	125	C18I.LO.LAST	26207

C18I.LOB	26175	CACTBL.SIZ	12
C18I.STORE	26235	CACXDV.ABT	25246
C18I.STORE.LAST	26171	CACXDV.ANS	25213
C18I.STORE.MBR	26236	CACXDV.BPS	25111
C180	4	CACXDV.BRK	25070
C180.APR	26335	CACXDV.CLE	25266
C180.HIB	26246	CACXDV.DIS	25003
C180.INT	26242	CACXDV.ENA	25022
C180.LAST	26310	CACXDV.MRK	25104
C180.LOB	26261	CACXDV.OFF	25006
C180.LOB.LAST	26301	CACXDV.OK	25170
C180.LOB.MIDDLE	26300	CACXDV.ON	25012
C18TBL.LEN	6	CACXDV.PAG	25066
C18XDV.EITHER	26440	CACXDV.PUT	25216
C18XDV.ILOOP	26436	CACXDV.RST	24761
C18XDV.IOS	26414	CACXDV.STS	25255
C18XDV.LOWER	26420	CACXDV.TBL	24747
C18XDV.RAISE	26412	CACXDV.XFR	25164
C18XDV.RESET	26376	CACXDV.XMT	25232
C18XDV.STATUS	26423	CACXDV.XON	25063
C18XDV.TBL	26370	CAL	21617
C18XDV.UNLOOP	26444	CALICR	22224
CAC.APRI	24675	CALL	22603
CAC.APRO	24712	CAR	21624
CAC.CC.ABT	1	CARICL	22220
CAC.CC.BRK	3	CARRY	21646
CAC.CC.OVR	2	CAS	21524
CAC.CSR.AFC	4000	CAS1	21533
CAC.CSR.ALL	37400	CBITCOLL	20
CAC.CSR.CTL	34400	CBITRESET	0
CAC.CSR.ENA	10000	CBITSET	20
CAC.CSR.ERR	2000	CHK	22546
CAC.CSR.OUT	20000	CHK2	22550
CAC.CSR.OVR	1000	CHK3	22555
CAC.CSR.XFR	400	CHS	21656
CAC.DONE	25347	CLK.EXIT	21267
CAC.DPR	24726	CLK.INTRPT	21250
CAC.OPT.AFC	2	CLK.NOERR	21322
CAC.OPT.DDT	1	CLK.SERVE	21272
CAC.OPT.HIB	4	CLK.WR1	21256
CAC.PDV	25273	CLK.WRITE	21276
CAC.RET	24672	CLK.WRRET	21325
CAC.SCC	25340	CLOCK.AGAIN	24241
CAC.START	25313	CLOCK.EXIT	24254
CAC.XDV	24744	CLR.DISP	41
CAC.XDVI	24741	CLR.MAINM	44
CAC.XDVO	24743	CLR.REG	30
CACBPS.LEN	10	CLR.URAM	34
CACBPS.SET	25134	CM1.CC.ERROR	1
CACBPS.TBL	25124	CM1.CRC	25637
CACDPR.UBD	24737	CM1.CRC.PATA	25645
CACERR.0	100	CM1.CRC.PATB	25642
CACERR.IBRR	101	CM1.CRC.PATC	25637
CACERR.NENA	100	CM1.CRC.PATD	25652
CACI	5	CM1.DLE	20
CACI20	20	CM1.DONE	26136
CACO	6	CM1.DONE0	26126
CACPDV.IO	25306	CM1.DPR	25670

CM1.EDONE	26116	CRASH.CLOCK	3460
CM1.EDONEO	26117	CRASH.IOOS	3500
CM1.EDONESTOP	26113	CRASH.MAIN	3440
CM1.ERROR	25464	CRASH.SOFT	3540
CM1.ETX	203	CRASH.STATE	3420
CM1.IN2OUT	20	CRASH.SYSTEM	3400
CM1.PDV	25777	CRASH.XXX	3520
CM1.RESET	26005	CSA	21672
CM1.RESTART	26075	DCB.A	7
CM1.RESYNC	25425	DCB.B	12
CM1.START	26060	DCB.F	11
CM1.STX	2	DCB.GET	23
CM1.SYN	26	DCB.GOAD	4
CM1.XDV	25706	DCB.IOCB	22
CM1.XDV.DIS	25725	DCB.PC	6
CM1.XDV.ENA	25717	DCB.PRIORITY	3
CM1.XDV.LOOPINT	25733	DCB.PUT	24
CM1.XDV.LOOPMOD	25764	DCB.Q.BACK	1
CM1.XDV.SIZE	6	DCB.Q.FORW	0
CM1.XDV.TABLE	25711	DCB.Q.HEDR	2
CM1.XDV.UNLOOP	25717	DCB.R1	14
CM1.ZUSART	26023	DCB.R2	15
CM1CSR.BIT3	10	DCB.RB	21
CM1CSR.ENABLE	4	DCB.RUNH	17
CM1CSR.ENALL	7	DCB.RUNL	20
CM1CSR.RXIEN	1	DCB.SP	13
CM1CSR.TXIEN	2	DCB.STATE	3
CM1I	1	DCB.TIME	16
CM1I.0	25360	DCB.TYPE	5
CM1I.APR	25661	DCB.X	10
CM1I.CHECKCRC	25502	DD.ACT.OFFSET	2
CM1I.DATA	25444	DD.ADV.OFFSET	0
CM1I.DLE	25466	DD.DDV.OFFSET	1
CM1I.GOING	25405	DD.HPK.OFFSET	3
CM1I.HIGH	25455	DD.RET	23776
CM1I.INT	25370	DDT.TCONST	1
CM1I.NEW	25412	DDV.DD.CALL	23265
CM1I.STORE	25447	DDV.IDLE	23255
CM1I.STX	25420	DDV.PENDING	23274
CM10	2	DDV.READY	23270
CM10.0	25364	DDV.TIMING	23261
CM10.APR	25665	DEV.CMD.BRK	10
CM10.CRC	25616	DEV.MAX	16
CM10.CRCEND	25633	DEV.STS.BRK	40
CM10.DATA	25556	DEV.STS.ERR	60
CM10.DLE	25603	DEV.STS.OVR	20
CM10.EOP	25606	DO.IOS	641
CM10.IDLE	25531	DSP.DEV	1600
CM10.INT	25520	DUMMY	23161
CM10.LOW	25563	EDAC.PREP	77
CM10.NEW	25542	EIGHTH1	400
CM10.SEND	25573	EIGHTH2	3
CM10.SEND1	25575	ELEVENTH1	40
CM10.SOP	25544	ENQ.PENDING.RET1	23735
CMA	21700	ENTER.DDT	24
COUNT2	140000	ERA	21465
CRA	21661	EXC	21413
CRASH	23021	EXCI	21416

EXOIA	21421	H.XMTIEN	1
EXOIB	21432	H.XMTPND	1000
EXOIX	21427	H.XMTRST	10
EXOX	21424	H.XPATCH	20
EX1	21436	HLT	21605
EX1I	21442	I18.0	26140
EX1IX	21453	IAB	21607
EX1X	21446	ICA	21642
F.MODE	177	ICAC.0	24425
F.NNULL	2000	ICAC.1ST	24506
F.OPEN	1000	ICAC.2ND	24520
FETCH	21570	ICAC.BAD	24527
FETCH.BASE	21577	ICAC.BGN	24470
FETCH.THIS	21573	ICAC.ERR	24572
FFO	22230	ICAC.HIB	24536
FIFTEENTH1	2	ICAC.INT	24435
FIFTH1	4000	ICAC.LOB	24532
FIFTH2	300	ICAC.NFC	24460
FIRST1	100000	ICAC.NOK	24502
FIRST2	140000	ICAC.STORE	24554
FIRST4	170000	ICAC.TST	24541
FIRST8	177400	ICACTST.EXT	24552
FIRSTO	2	ICL	21615
FLAGS.CCODE	17	ICR	21622
FLAGS.IALWAYS	100000	IMA	21540
FLAGS.IERROR	40000	INT.INSTR	22722
FLAGS.IO	140017	INT.SCAN	22731
FORTH4	17	IO.ACT.HPK	23303
FOURTEENTH1	4	IO.BXA1	24423
FOURTH1	10000	IO.CLEAR	24373
FOURTH2	1400	IO.CLOCK	24233
GO.TEMP	20	IO.CRASH	24371
G1.TEMP	21	IO.DD.CALL	23756
G10.STATE	30	IO.ENQ.IDLE	23472
G11.TEMP	31	IO.ENQ.PEND0	23727
G12.PROGINT	32	IO.ENQ.PEND2	23722
G12.TEMP	32	IO.ENQ.PENDING	23725
G13.MBR	33	IO.ENQ.READY	23531
G14.MISC	34	IO.ENQ.TIMING	23567
G15.REFRSH	35	IO.FIND.PHEDR	24001
G15.TEMP	35	IO.GOAD	23432
G2.TEMP	22	IO.GOADSKIP	23423
G3.RONE	23	IO.RET	24406
G4.RSIGN	24	IO.RMQ.IDLE	23451
G5.TEMP	25	IO.RMQ.PENDING	23663
G6.TEMP	26	IO.RMQ.READY	23507
G7.BASE	27	IO.RMQ.TIMING	23546
G7.TEMP	27	IO.XDV	24415
GO.VECTOR	20014	IO316.ACT	23300
GOT.BREAK	313	IO316.ADV	23206
H.HSTDWN	2000	IO316.DDV	23234
H.HSTFLP	4000	IO316.ENB	23171
H.IMPRDY	2	IO316.HPK	23321
H.LSTBIT	4	IO316.INH	23163
H.RCVIEN	40	IO316.NMFS	23104
H.RCVPND	400	IO316.PCB	23417
H.RCVRST	100	IO316.RFI	23353
H.SPARE	200	IO316.SPK	23323

```
L15.DDT                 15
L15.LITES               15
L15.TIME                15
L15.USART               15
L15.XCFLAG              15
L16.BUF                 16
L16.CRPTR               16
L16.DDBASE              16
L16.LCOUNT              16
L16.STP                 16
L16.XCCODE              16
L17.CHAR                17
L17.CTN                 17
L17.IDLH                17
L17.PASSWD              17
L2.DCB                   2
L2.RET2                  2
L2.SHFTMP                2
L3.A                     3
L3.CLK1                  3
L3.COUNT                 3
L3.HIBYTE                3
L3.IOCB                  3
L3.LOBYTE                3
L3.MAR                   3
L3.RET3                  3
L3.SHFMAR                3
L3.TTY                   3
L3.WMASK                 3
L4.ADDR                  4
L4.B                     4
L4.CLK2                  4
L4.IOCB                  4
L4.MBR                   4
L4.OUT.STAT              4
L4.RDYH                  4
L4.SC                    4
L4.STATE                 4
L4.TITIME                4
L4.WMAR                  4
L5.ADDR                  5
L5.CNTR                  5
L5.FLAGS                 5
L5.IN.STAT               5
L5.INSPC                 5
L5.LEFT                  5
L5.NEXT1                 5
L5.TEMP                  5
L5.TIMH                  5
L5.TITHIS                5
L5.X                     5
L6.CHAR                  6
L6.CURR                  6
L6.DCB                   6
L6.IN.CHAR               6
L6.INTS                  6
L6.LEFT                  6
L6.POINT                 6
L6.WMBR                  6
L7.ALUST                 7
L7.CBIT                  7
L7.DDUPC                 7
L7.NEXT                  7
L7.OUT.CHAR              7
L7.PMSCAN                7
L7.TEMP1                 7
L7.WHERE                 7
LDA                  21457
LDX                  21555
LGL                  22043
LGR                  21776
LITES                22527
LITES.PATCH          22541
LLL                  21767
LLR                  21767
LLS                  22021
LRL                  21767
LRR                  21767
LRS                  22021
M.DWFIELD              140
M.LITES               7400
M.TO.U                 753
M16K                 40000
M2.ERRORS             1403
M52.CC.ABC               5
M52.CC.ABORT             4
M52.CC.CRC               3
M52.CC.ERROR             1
M52.CC.OVRUN             2
M52.CC.UNDRUN            2
M52.CRPTR.MSYNC      40000
M52.CSR.WIO          27001
M52.DONE             27302
M52.DONE.RET         27301
M52.DONEO            27263
M52.DPR              27036
M52.EDONE            27261
M52.EDONEO           27262
M52.EDONESTOP        27252
M52.IN2OUT              20
M52.IOCB.EOF           200
M52.IOCB.F           20000
M52.IOCB.O           10000
M52.PDV              27140
M52.RESTART          27221
M52.START            27201
M52.STOUT            27214
M52.TXO              26773
M52.WRITE.CSR        27004
M52.XDV              27053
M52.XDV.ABORT        27072
M52.XDV.DIS          27104
M52.XDV.ENA          27064
M52.XDV.ERR          27070
M52.XDV.RET          27066
M52.XDV.SET          27123
```

M52.XDV.SIZE	6	MTICSR.RXENBL	1
M52.XDV.STAT	27112	MTICSR.STAT	7
M52.XDV.TABLE	27056	MTICSR.TXENBL	2
M52.ZUSART	27146	MTICSR.XPATCH	10
M52I.0	26601	MU.1	755
M52I.APR	27025	NINETH1	200
M52I.CC.EOM	26706	NODEV	23073
M52I.EOM	26674	NOPP	21570
M52I.HIGH	26662	NOTYET	21603
M52I.INT	26611	NREG	206
M52I.LOW	26635	018.0	26144
M52I.MII	13	OCAC.0	24431
M52I.MTI	7	OCAC.1ST	24626
M52I.NEW	26624	OCAC.2ND	24640
M520.0	26605	OCAC.BGN	24616
M520.APR	27006	OCAC.END	24665
M520.EOB	26750	OCAC.HIB	24635
M520.EOM	26772	OCAC.INT	24574
M520.HIGH	26730	OCAC.LOB	24632
M520.IDLE	26721	OCAC.SEND	24642
M520.INT	26712	OCAC.SGL	24654
M520.LOW	26733	OP	3
M520.MII	14	P.BUTTON	57
M520.MORE	26760	P.NEW	26
M520.MTI	10	PASSWORD	333
M520.NEW	26723	PCR.RXCL	3400
M520.SEND	26740	PCR.RXCLE	4000
M520.SEND.MAR	26744	PCR.TXCL	16000
M520.SEND.MBR	26741	PCR.TXCLE	10000
MAIN	20050	PCSAR.APA	100000
MBBIO.CSR	160	PCSAR.ECM	3400
MBBIO.DSW	140	PCSAR.IDLE	4000
MBBIO.LOADR	20	PCSAR.PROTO	40000
MBBIO.TERM	0	PCSAR.SAM	10000
MEM.EXIT	21244	PCSAR.SAR	377
MEM.LOOP	21222	PCSAR.SSGA	20000
MEM.SIZER	21217	PDT.AT	21200
MEMHI	22333	PDT.BIN	561
MPCC.CSR	400	PDT.BIN.D	556
MPCC.CSR.MTI	100	PDT.BOOT	306
MPCC.PCR	6	PDT.BOUT	612
MPCC.PCSAR	4	PDT.BOUT.D	605
MPCC.RDSR	0	PDT.BRKPNT	103
MPCC.SLCSR	400	PDT.CLOSE	454
MPCC.TDSR	2	PDT.CLRNUM	241
MSK.REFRSH	377	PDT.CLS.D	505
MSK177777	177777	PDT.CLS.I	503
MSK377	377	PDT.CLS.M	501
MSK37777	37777	PDT.CLS.R	470
MSK77777	77777	PDT.CLS.U	476
MTICSR.CTS	4	PDT.CR	344
MTICSR.DCD	2	PDT.CRPRMT	226
MTICSR.DSR	1	PDT.D	422
MTICSR.DTR	40	PDT.DIGIT	324
MTICSR.ENABLE	4	PDT.DIS	201
MTICSR.ENALL	3	PDT.DL	341
MTICSR.MACRO	70	PDT.E	20120
MTICSR.RTS	20	PDT.EXIT	575

```
PDT.FAKEIN          244
PDT.G             20147
PDT.H             20141
PDT.HREG          22765
PDT.I               431
PDT.INIT            122
PDT.INT             166
PDT.LF              350
PDT.LFCHR           366
PDT.LFEX            360
PDT.LFPNT           363
PDT.LOAD            116
PDT.M               403
PDT.MAINLP          242
PDT.MINUS         21205
PDT.MORE          22740
PDT.OPEN            440
PDT.PLP             170
PDT.PLUS          21210
PDT.PROMO             0
PDT.PROMPT          230
PDT.PRT20           517
PDT.PRT32           531
PDT.PRTN.1          533
PDT.QUEST           304
PDT.R               411
PDT.RTS             315
PDT.SPACES          234
PDT.START           214
PDT.TRAP            110
PDT.TYPE1C          224
PDT.U               371
PDT.UA            20132
PDT.URAM          20010
PDT.URAM.         20100
PDT.URAM.E        20116
PDT.WATCH         22774
PM.CLEAR          23101
PM.SCAN           24007
POP               22610
POPA              22620
PREG                200
PRIORITY.0            0
PRIORITY.ALL         37
PUSH              22573
PUSH1             22574
PUSHA             22601
Q.BACK                1
Q.DEQ             22402
Q.DEQ.RMQ         22412
Q.ENQ             22341
Q.FORW                0
Q.HEDR                2
Q.LENG                3
Q.RMQ             22443
Q316.DEQ          22505
Q316.ENQ          22471
Q316.RMQ          22516

QERR.0               20
QERR.ILLB            21
QERR.ILLF            20
QERR.ILLH            22
QERR.ILLL            23
RANDOM.0            100
RANDOM.JFFZ         100
RB.ALL             1777
RB.BITS             260
RB.CAC              400
RB.CLOCK             40
RB.CPDT             100
RB.CTTY              60
RB.GLOBAL             0
RB.ICAS             440
RB.ICON             400
RB.IOOS             300
RB.LOADER           160
RB.LPDT             140
RB.LTTY             120
RB.MAIN             200
RB.MSHIFT           220
RB.OCAS             460
RB.OCON             420
RB.SOFTINT          240
RB.STATE             20
RB.TID              250
RCB               21667
RCCLOK            22323
RCSR.ABC          70000
RCSR.ALLSTOP     177000
RCSR.BADSTART    176000
RCSR.EOM           1000
RCSR.ERR         100000
RCSR.RABGA         2000
RCSR.ROR           4000
RCSR.RSOM           400
RCSR.RXDB           377
READ.URAM           735
RET               22627
RETSKP            22635
RFI.TO.PENDING    23401
RFI.TO.READY      23373
RFI.TO.TIMING     23407
RNEXT               500
RUN.TCONST          200
SAVE.STATE           66
SCAN.ATTN1        24070
SCAN.ATTN2        24072
SCAN.BOTTOM       24055
SCAN.FROM.PENDIN  24050
SCAN.FROM.READY   24044
SCAN.FROM.TIMING  24046
SCAN.HAVE.PRIORI  24101
SCAN.NEW          24113
SCAN.NEW.INIT     24161
SCAN.NOSKIP       24042
SCAN.SAME         24230
```

SCAN.SCAN	24061	SMI	21762
SCAN.TO.PENDING	24053	SNZ	21750
SCB	21712	SOFT	22667
SECOND1	40000	SOFT.INSTR	22704
SECOND2	30000	SOFT.SCAN	22707
SECOND4	7400	SOFT.WATCH	22712
SECOND8	377	SOFTINT	20040
SETSKP	22644	SP2A	22653
SETUP.2651	116	SP2X	22663
SEVENTH1	1000	SPL	21735
SEVENTH2	14	SRC	21715
SHFARITH	22151	SREG	210
SHFINT	22204	SSC	21742
SHFL	22113	SSM	21707
SHFL1	22114	SSP	21664
SHFLA	22170	SST.FLAGS	3
SHFLA1	22173	SST.REFRSH	2
SHFLR	22137	SST.RUN	1
SHFLR1	22140	STA	21506
SHFLUP	22051	STACK.ERR	22577
SHFLUP1	22070	START	21360
SHFR	22101	STARTDUM	20157
SHFR1	22102	STATE.ALL	170000
SHFRA	22153	STATE.IDLE	100000
SHFRA1	22156	STATE.PENDING	10000
SHFRET	22210	STATE.READY	40000
SHFROT	22123	STATE.REGOAD	400
SHFRR	22125	STATE.REPOKED	2000
SHFRR1	22126	STATE.RETIMED	4000
SHFSIMPLE	22077	STATE.SKIP	1000
SHOW.LITES	766	STATE.TIMING	20000
SIXTEENTH1	1	STX	21561
SIXTH1	2000	SUB	21474
SIXTH2	60	SZE	21723
SKIP	21565	SZESRC	22213
SKP	21565	TCA	21703
SLCSR.CTS	2	TDSR.ABORT	2000
SLCSR.DCD	4	TDSR.ERR	100000
SLCSR.DTR	20	TDSR.TEOM	1000
SLCSR.ENALL	3	TDSR.TGA	4000
SLCSR.ENBL	1	TDSR.TSOM	400
SLCSR.INA	10	TDSR.TXDB	377
SLCSR.INB	20	TENTH1	100
SLCSR.INC	40	THIRD1	20000
SLCSR.IND	100	THIRD2	6000
SLCSR.INE	200	THIRD4	360
SLCSR.MACRO	374	THIRTEENTH1	10
SLCSR.MMODE	4	TI.HERE	23634
SLCSR.OUTA	40	TI.INT	23631
SLCSR.OUTB	100	TI.LOOP	23615
SLCSR.OUTC	200	TI.TOP	23607
SLCSR.RTS	10	TIME.CONST	20000
SLCSR.RXSA	1	TT.EXIT	733
SLCSR.STAT	377	TTIN.ECHO	701
SLCSR.TXIEN	2	TTIN.POLL	646
SLCSR.XPATCH	4	TTIN.URAM	21336
SLN	21755	TTOUT.POLL	706
SLZ	21730	TTOUT.URAM	21327

TTY.BRK	40	URAM0	20000
TTY.CMND	3	WATCH.GO	23006
TTY.CR.DTR	2	WATCH.OFF	23014
TTY.CR.RE	20	WRITE.DISP	507
TTY.CR.REN	4	WRITE.RB	23061
TTY.CR.RTS	40	WRITE.RBL	23063
TTY.CR.TEN	1	WRITE.URAM	743
TTY.CRLF	461	XC.BRKPNT	40
TTY.DSR	200	XC.BUTTON	2
TTY.GCTBRK	400	XC.J2ZERO	20
TTY.HALF	20	XC.MEMERR	400
TTY.HCLD	0	XC.NEWPWR	1
TTY.INIT	623	XC.TRAP	100
TTY.MR	2	XC.UCMERR	10
TTY.OUT	2	XC.UPERR	4
TTY.OWN.M	4	XREG	205
TTY.OWN.PK	10		
TTY.RRDY	2		
TTY.SECHO	40		
TTY.SENDIC	1		
TTY.STS	1		
TTY.TEMT	4		
TTY.TRANS	100		
TTY.TRDY	1		
TTYRECDUM	21356		
TTYXOCNDUM	21354		
TTYXMIT	21344		
TWELFTH1	20		
TYPE.ALL	77		
TYPE.SOFTWARE	0		
U1822.CSR	0		
U1822.RCV	1		
U1822.XMT	1		
U2651.CR	3		
U2651.CR.DTR	2		
U2651.CR.RESET	20		
U2651.CR.RTS	40		
U2651.CR.RXEN	4		
U2651.CR.TXEN	1		
U2651.MCR	200		
U2651.MR	2		
U2651.RX	0		
U2651.SR	1		
U2651.SR.DCD	100		
U2651.SR.DTR	200		
U2651.SR.OVR	20		
U2651.SR.RXRDY	2		
U2651.SR.SYN	40		
U2651.SR.TXRDY	1		
U2651.SSD	1		
U2651.SYNC.CR	5		
U2651.SYNC.MR1	214		
U2651.SYNC.MR2	0		
U2651.TX	0		
U2651.XP.CR	243		
U2651.XP.MR1	316		
U2651.XP.MR2	77		
UNEXT	27304		

```
Assembly began:          Thu Oct 14 01:15:18 1982
Assembly finished:       Thu Oct 14 01:22:20 1982
Assembly real time:      422 seconds.
Processor time used:     user 263.93, system 8.67  (total 272.60, 65% of cpu.)
Defined 1565 symbols; used 205 of 511 hash buckets.
Processed 19370 real source lines for 1377 lines/minute.
```

```
  AA     L        EEEEEE  X     X     AA     N     N  DDDDD
 A  A    L        E        X   X     A  A    NN    N  D    D
A    A   L        EEEEE     XX     A    A   N N   N  D    D
AAAAAA   L        E         XX     AAAAAA   N  N  N  D    D
A    A   L        E        X  X    A    A   N   NN  D    D
A    A   LLLLLL   EEEEEE  X    X   A    A   N    N  DDDDD
```

Wed Oct 27 21:50:29 1982

Listing file Pipe output for Margaret Alexander at bbnw.

Listing file Pipe output for Margaret Alexander at bbnw.

Listing file Pipe output for Margaret Alexander at bbnw.

Wed Oct 27 21:50:29 1982

```
  AA     L        EEEEEE  X     X     AA     N     N  DDDDD
 A  A    L        E        X   X     A  A    NN    N  D    D
A    A   L        EEEEE     XX     A    A   N N   N  D    D
AAAAAA   L        E         XX     AAAAAA   N  N  N  D    D
A    A   L        E        X  X    A    A   N   NN  D    D
A    A   LLLLLL   EEEEEE  X    X   A    A   N    N  DDDDD
```

```
   AA     L        EEEEEE  X     X     AA     N     N  DDDDD
  A  A    L        E        X   X     A  A    NN    N  D    D
 A    A   L        EEEEE     XX      A    A   N N   N  D    D
 AAAAAA   L        E         XX      AAAAAA   N  N  N  D    D
 A    A   L        E        X  X     A    A   N   NN  D    D
 A    A   LLLLLL   EEEEEE  X    X    A    A   N    N  DDDDD
```

Wed Oct 27 21:50:29 1982

Listing file Pipe output for Margaret Alexander at bbnw.

Listing file Pipe output for Margaret Alexander at bbnw.

Listing file Pipe output for Margaret Alexander at bbnw.

Wed Oct 27 21:50:29 1982


```
   AA     L        EEEEEE  X     X     AA     N     N  DDDDD
  A  A    L        E        X   X     A  A    NN    N  D    D
 A    A   L        EEEEE     XX      A    A   N N   N  D    D
 AAAAAA   L        E         XX      AAAAAA   N  N  N  D    D
 A    A   L        E        X  X     A    A   N   NN  D    D
 A    A   LLLLLL   EEEEEE  X    X    A    A   N    N  DDDDD
```

Number: 34 Length: 1568 bytes
Date: 27 Oct 1982 14:40:57 EDT (Wednesday)
From: Robin Clifford <clifford at BBN-UNIX>
Subject: Number of racks for the move
To: alexander at BBN-UNIX
Cc: clifford at BBN-UNIX, powers at BBN-UNIX

Margaret,

Following is the list of equipment racks that will be moved.

The ones with an asterisk and a serial number are the ones that BBNCC are responsible for.

ARPAnet:

BBN-TAC (H-316, 3 racks) * #1100936

NCC-TAC (H-316, 3 racks) * #1102994

IMP 63 (C/30, 1 rack) * #828165

IMP 40 (C/30, 1 rack) * #828164

Curly (test machine) (C/30, 1 rack) * #81208

UCDLA-IMP (may be for 2 lower, but is comm. dev. machine)
(C/30, 1 rack) * #83004

UCDLA-TAC (also a comm. dev. system - 2 lower or upper?)
(C/30, 1 rack) * #83001

SATnet:

MATnet (C/30, 1 rack) * #81308

SATnet (H-316, 1 rack) * #163296

INTERnet:

Internet PDP-11 gateway (3 racks)

VAN gateway (1 rack)

HP3000:

4 racks includes CPU, disk drive, tape drive, printer

PSAT:

PSAT1 (pluribus, 2 racks) * #820301

PSAT2 (pluribus, 2 racks) * (I don't have the Serial #. I have to get into the EAST BAY and get it.)

Citibank:

C-TIP (pluribus, 3 racks) * (room locked, will provide # when I get access.)

Tymnet system (1 rack) (is Tymnet involved? I'll check with Joel)

Tymnet modem (1 rack)

Honeywell 606 (1 rack) plus table top H-606 (consider it a rack?)

I'll get the missing serial #'s for Jim Powers. The total number of racks (counting the table top H-606 for Citibank) is, uh:

Let me know what other info you may need.

Regards,

Robin

<*>

```
DDDDD     SSSS   H     H  U     U  RRRRR     TTTTT  L        EEEEEEE
D    D  S       H     H  U     U  R    R      T    L        E
D    D   SSSS   HHHHHHH  U     U  R    R      T    L        EEEEE
D    D       S  H     H  U     U  RRRRR       T    L        E
D    D  S    S  H     H  U     U  R   R       T    L        E
DDDDD     SSSS   H     H   UUUU   R    R      T    LLLLLLL  EEEEEEE
```

Wed Oct 27 21:50:43 1982

Listing file list.27.2150 for David Shurtleff at bbnt.

Listing file list.27.2150 for David Shurtleff at bbnt.

Listing file list.27.2150 for David Shurtleff at bbnt.

Wed Oct 27 21:50:43 1982

DDDDD   SSSS   H   H U   U RRRRR   TTTTT L   EEEEEEE
D   D S   H   H U   U R   R   T   L   E
D   D   SSSS   HHHHHHH U   U R   R   T   L   EEEEE
D   D   S H   H U   U RRRRR   T   L   E
D   D S   S H   H U   U R   R   T   L   E
DDDDD   SSSS   H   H   UUUU   R   R   T   LLLLLLL EEEEEEE

```
 DDDDD     SSSS    H     H  U     U  RRRRR     TTTTT  L        EEEEEE
 D    D   S        H     H  U     U  R    R      T    L        E
 D    D    SSSS    HHHHHHH  U     U  R    R      T    L        EEEEE
 D    D        S   H     H  U     U  RRRRR       T    L        E
 D    D   S    S   H     H  U     U  R   R       T    L        E
 DDDDD     SSSS    H     H   UUUU   R    R      T    LLLLLL   EEEEEE

 Wed Oct 27 21:50:43 1982

Listing file list.27.2150 for David Shurtleff at bbnt.

Listing file list.27.2150 for David Shurtleff at bbnt.

Listing file list.27.2150 for David Shurtleff at bbnt.

 Wed Oct 27 21:50:43 1982

DDDDD     SSSS    H     H  U     U  RRRRR     TTTTT  L        EEEEEE
 D    D   S        H     H  U     U  R    R      T    L        E
 D    D    SSSS    HHHHHHH  U     U  R    R      T    L        EEEEE
 D    D        S   H     H  U     U  RRRRR       T    L        E
 D    D   S    S   H     H  U     U  R   R       T    L        E
 DDDDD     SSSS    H     H   UUUU   R    R      T    LLLLLL   EEEEEE
```

```
1  27 Oct 1982  Ben Littauer <littau  tac patches
2  27 Oct 1982  Ira Smotroff <smotro  hdh preliminary design
```

Number: 1 Length: 1036 bytes
Date: 27 Oct 1982 16:31:15 EDT (Wednesday)
From: Ben Littauer <littauer at BBN-UNIX>
Subject: tac patches
To: network-software-note at BBN-UNIX

A whole array of bug fixes this time, mostly TCP. Included are a fix for XON-XOFF handling on TCP input, a (large) patch for handling of anomalous conditions in TCP's SYNSENT state, and a minor change to the handling of transmit-on-linefeed. We have also fixed a bug in our handling of the TCP listen state.

The reason for two releases at the same time is that we had begun to release patch 5 when the patch 6 bug was discovered and fixed. In order to make sure that all the TACs are up to date, we decided to waste an extra patch level...

-ben-

Release 102:

patch level 05 27 oct 82	[fix TCP XIN, traps->tty bug, transmit on linefeed fix, disable FNDTRN trap, correct handling of 1/2 open TCP conn.]
patch level 06 27 oct 82	[fix TCP listen state bug]

<*>

Number: 2 Length: 19881 bytes
Date: 27 Oct 1982 16:56:29 EDT (Wednesday)
From: Ira Smotroff <smotroff at BBN-UNIX>
Subject: hdh preliminary design
To: santos at BBN-UNIX, herman at BBN-UNIX, shurtleff at BBN-UNIX, laube at BBN
>>>UNIX
Cc: smotroff at BBN-UNIX

Folks-
The following document should give you some idea of what I intend nmfs-HDH to look like.The general form is a bit of descriptive text followed by a rough first pass flow chart.Thanx to Paul who wrote much of the text in one guise or another.
Comments apprieciated.

ira

HDH preliminary design outline

The following sketches out the functionality of the routines I plan to develop.This implementation will not use stacks because the current level 2 code does not use stacks and therefore we would not gain much by using them now.The package should be converted to stacks when level 2 is converted.

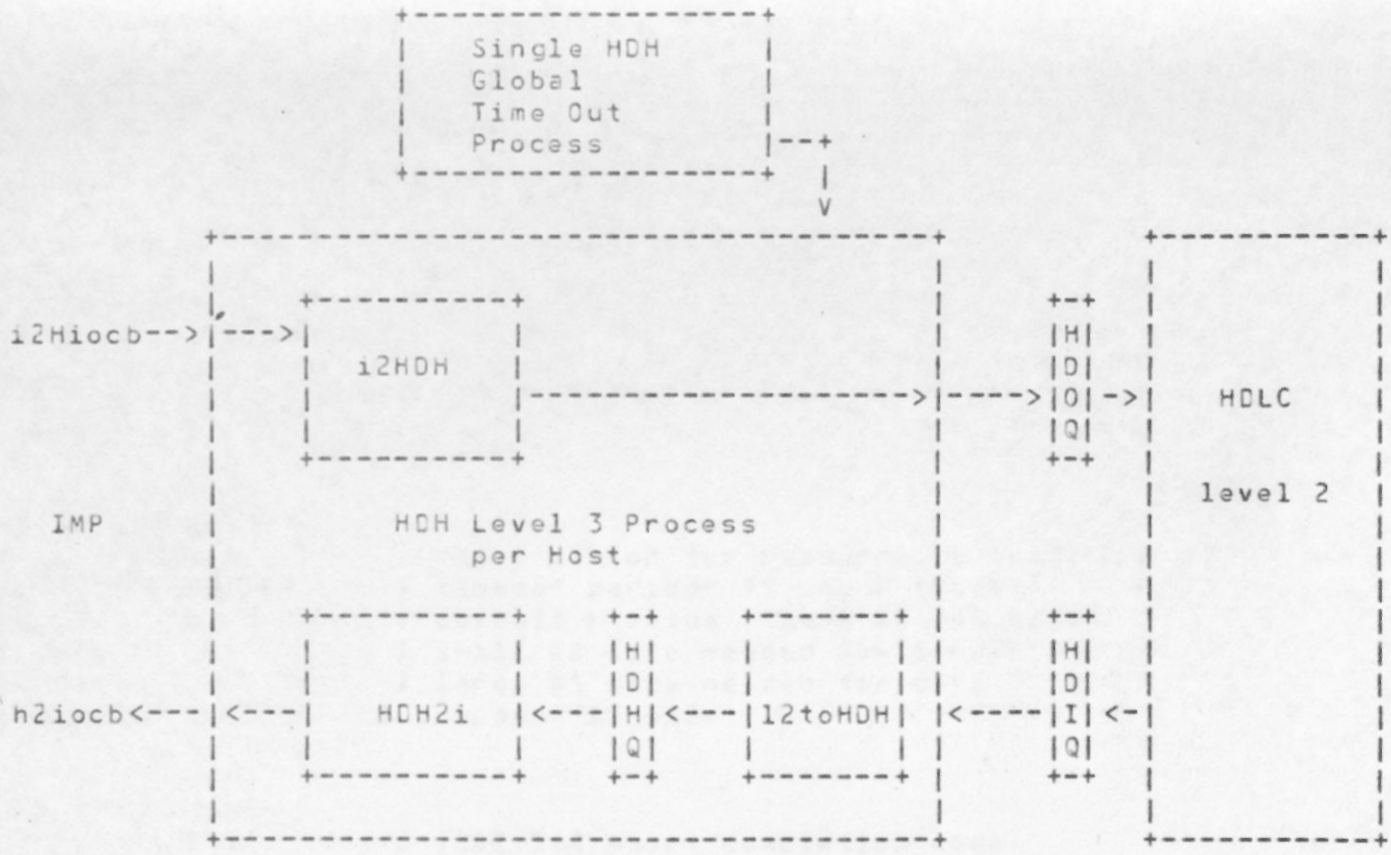

figure 1 - HDH Process Structure

special mnemonic label conventions

HDH.xxx	general level 3
iH.xxx	imp to packet level (i2HDH)
H12.xxx	packet level to hdlc (HDHto2)
12H.xxx	hdlc tp packet level (L2toHDH)
Hi.xxx	packet level to imp (HDH2i)
Hpo.xxx	packet level poll (HDHpoll)
Hto.xxx	global level 3 timeout
int.xxx	initialization
Hpxxxx	utility routines
hcb.xxx	dte block (PCB) displacements
hcbxxx	DTE block (PCB) fields
ldr.xxx	leader block displacements
ldrxxx	leader fields
Hpk.xxx	packet displacements
Hpkxxx	packet fields
iocb.xxxx	IOCB displacements
rdb.xxxxx	raw data block displacements
sas.xxxxx	simple aggregate structure block displacements
cas.xxxxx	compound aggregate structure block displacements
pcb.xxxxx	generic PCB displacements

Data Structures

HCB- HDH CONTROL Block
definitions and constants

```
; HDH process level definitions

; PCB priority levels
HDH.pri=m2ipri+(tsk-m2i)
t3.pri=m2ipri+(t.o-m2i)

; .lev priority
HDH.lev=tsk
t3.lev=t.o


; major parameters
HDH.per=%25.6ms          ; retry period for resource allocation
t3.per=(9375./nh)+1      ; timeout period- 15 sec / #hosts
HDHkval=2.               ; default k value - send window size
HDHsmrb=6.               ; small RB size needed for leader
HDHlgas=24.              ; large AS size needed for data
HDHcas=0.               ; compound AS size


; IOCB definitions
%c18compl.abt=1.         ; 1822 I/O abort completion code

; AS definitions
sas.len=sdb.len          ; data length
sas.addr=sdb.addr        ; address of first data word
sas.ldb=sdb.flags+1      ; leader block ptr in SAS
cas.forw=cdb.comqhedr    ; compound q header forward pointer
cas.back=cas.forw+1      ; compound q header backward pointer
cas.qlen=cas.forw+3      ; compound q header queue length
cas.comlen=cdb.comlen    ; compound AS compound data length

; HDH NCC command definitions
ncc.pilhst=ncc.uahst+1  ; loop host (HDLC) interface
ncc.pelhst=ncc.pilhst+1 ; loop external (modem)
ncc.pslhst=ncc.pelhst+1 ; loop host software
ncc.puahst=ncc.pslhst+1 ; unloop/reset all for host
```

```
; HDH process PCB
; dte block (HCB) definitions

.=pcb.iocb              ; first application word of PCB
dt.tic: .block 1        ; 15-second tick flag
dt.sta: .block 1        ; state/events/conditions/timer
.=pcb.num               ; should be the next word
dt.num: .block 1        ; DTE (host) number
dt.rdy: .block 1        ; ready line flag (dtimpd or 0)
dt.hcb: .block 1        ; L2 PCB pointer
dt.phq: .block 1        ; HDHto2 queue header pointer
dt.php: .block 1        ; count of data packets sent to L2
dt.nob: .block 1        ; count of block allocation failures

; i2HDH section
dt.ipa: .block 1        ; saved AS or leader block
dt.ipp: .block 1        ; saved IOCB blt pointer
dt.ipc: .block 1        ; saved IOCB blt count
dt.ipx: .block 1        ; saved SAS blt pointer
dt.ipy: .block 1        ; saved SAS blt count
dt.ipb: .block 1        ; I2H PCB pointer
dt.ipc: .block 1        ; I2H PCB IOCB word pointer
dt.ipk: .block 1        ; I2H abort flag

; L2toHDH section
dt.hpa: .block 1        ; saved AS block
dt.hpq: .block 1        ; L2to3 queue header pointer
dt.hpp: .block 1        ; count of data packets rcvd from L2

; HDH2i section
dt.pia: .block 1        ; saved AS block
dt.pii: .block 1        ; saved intermediate component block
dt.pip: .block 1        ; saved SAS blt pointer
dt.pic: .block 1        ; saved SAS blt count
dt.pih: .block qhedr.size ; HDHtoI queue header
dt.pib: .block 1        ; H2I PCB pointer
dt.pic: .block 1        ; H2I PCB IOCB word pointer
dt.pik: .block 1        ; H2I abort flag

HDHpcb.size=.           ; size of HDH process PCB
; fields in dt.rdy
dtimpd=040000   ; state of IMP interface (0 => up)

; timeout process PCB
.=pcb.iocb              ; first application word of PCB

t3pcb.size=.            ; size of timeout process PCB
```

```
  Global Level 3 variables

HDHgbeg:
;defplc(/HDH - initialization flag)
HDH.inf:        1               ; start with init flag set
;defplc(/HDH - shutdown flag)
HDH.sht:        0               ; shutdown notification is needed
;defplc(/HDH - HDH host number)
HDH.num:        .block 1        ; the current host
;defplc(/HDH - aggregate structure pointer)
HDH.as:  .block 1

;defplc(/HDH - IOCB pointer)
HDH.iob:        .block 1        ; the current host IOCB
HDH.pak:        .block 1        ; the current packet pointer
HDH.ldr: .block 1       ; the current leader block

;defplc(/HDH - HDHtoIQ pointer)
HDH.piq: .block 1       ; the current HDHtoIq header pointer

;defplc(/HDH - 15-second clock)
HDH.clk:        .block 1        ; clock, ticked once every 15 sec
HDH.tpr:        .block 1        ; TPR flag

;defplc(/HDH - table of HDH PCBs)
HDH.tab:        .block nh       ; table of HDH PCB addresses
HDHglen=.-HDHgbeg
```

```
;HD.HDR HEADER DEFINITIONS
HH.CTL=100000   ;CONTROL PACKET
HH.SEQ=040000   ;SEQUENCE BREAK
HH.IMP=020000   ;IMP FLAG
HH.LIN=010000   ;LINE UP/DOWN FLAG
HH.SOM=010000   ;START OF MESSAGE
HH.EOM=004000   ;END OF MESSAGE
hh.rfl=002000   ;reflect mode
HH.LDC=001777   ;LINE DOWN COUNT
HH.BYC=001777   ;PACKET BYTE COUNT MASK

;HD.STS STATUS FLAGS
HS.CNF=100000   ;CONFIGURED
HS.SEQ=040000   ;PACKET SEQUENCE BREAK
HS.LOP=020000   ;LINE LOOPED
HS.XOM=010000   ;SOM/EOM FLOP
HS.XER=004000   ;HDH PROTOCOL ERROR
HS.LLR=002000   ;LINK LEVEL READY
HS.PLR=001000   ;PACKET LEVEL READY
HS.LPI=000400   ;LINE LOOPED INTENTIONALLY
hs.rfl=000200   ;in reflect mode
hs.rft=000100   ;reflect mode toggle
HS.LDS=000030   ;LINE SPEED DESCRIPTOR
                ;0_3= 50K
                ;1_3= 230K
                ;2_3= 19.2K
                ;3_3= 9.6K
hs.mod=000040   ;mode (1)=packet ,(0)=msg
HS.IHY=000002   ;RECEIVED IHY FLAG
HS.HLO=000001   ;SEND HELLO FLAG

;HDH INSTRUMENTATION
HD.HPE: .block 1        ;HDH PROTOCOL ERRORS
HD.IPE: .block 1        ;1822 PROTOCOL ERRORS
HD.LLP: .block 1        ;LINE LOOPED (TRANSITIONS)
HD.INR: .block 1        ;INTERFACE NOT READY
HD.GBF: .block 1        ;GET BUFFER FAILURES
HD.GBL: .block 1        ;GET HDB FAILURES
HD.TDY: .block 1        ;HDH OUTPUT "TARDY"

;HDH TABLE
HD.HDR: .block 1     ;HDH HEADER
HD.CMD: .block 1     ;HDH COMMAND
HD.STS: .block 1     ;HDH STATUS
```

```
.STITL HDH BACKGROUND

        .lev VAR
HD.BN:  .block 1        ;HDH NUMBER
HD.BC:  .block 1        ;LOOP COUNTER
```

```
HDHlocp:main 13 process loop
  set up global variables
  call i2HDH if i2h iocn is available
call l2toHDH
  call HDHpoll
  call HDH2i if h2i iocb is available
time out if tpr flag was set
```