/**

SPOOL
TP.LST
05/04/82
15:44:36

**/

```
SERIES-III PL/M-86 V2.0 COMPILATION OF MODULE TP
OBJECT MODULE PLACED IN :F1:TP.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:TP.P86 OPTIMIZE(3) XREF SET(F1) DEBUG


              $TITLE('iLNA Transport Control Layer  Transmit Process  04/15/82 ')
              $COMPACT DEBUG NOCOND
*** WARNING 10 IN 1 (LINE 2): RESPECIFIED PRIMARY CONTROL, IGNORED
              $SET(mipform)


              $IF f7
              $ELSE
              $INCLUDE (:F1:cpyrt.dcp)

          =           /* Intel Corporation Proprietary Information.
          =           This listing is supplied under the terms of a
          =           license agrement with Intel Corporaton and
          =           may not be copied nor disclosed except in
          =           accordance with the terms of that agreement. */

              $ENDIF


                        /*  George D Marshall   SC6-213   x7-5117  */

                       /* This is TCL's Transmit Process and supporting routines.
                       TP is the only thing in TCL that can send segments (The
                       Receive Process, RP, causes segments (such as SYN-ACK or ACK)
                       to be sent by sending a short message (an Internal Request
                       Block, or IRB) to TP to request that the appropriate segment
                       be sent.  The Interface Process, IP, causes segments to be sent
                       in the same way. */


                       /*
                         modified 03/10/82 to fix problem in TP$PROC. When the AYTimer
                         went off, the routine that sets it lets RP run which can also
                         set it, resulting in a twice set ACB
                       */

1             tp: DO;



              $IF f7
              $ELSE
              $INCLUDE (:F1:TCLGBL.INC)
          =                                                    /**********************/
          =                                                    /** Global Literals **/
          =                                                    /**********************/
          =   /* TCL Global Literals                                          04/15/82 */
2    1    =   DECLARE
          =       max$send$seg        LITERALLY    '07H',  /* max no of back-to-back segs that  one connection */
```

```
=                                                   /* can send at a time */
=       tcl$header$len      LITERALLY   '20',       /* bytes in tcl header */

=                                       /* ETHERNET-SPECIFIC VALUES */
=       dll$header$len      LITERALLY   '14',       /* bytes in dll header */
=       min$pkt$len         LITERALLY   '46',       /* minimum total pkt len - bytes */
=       max$seg$data$len$lit LITERALLY  '1480',     /* (1480) max no. of client bytes in seg */
=       tcl$protocol$code   LITERALLY   '5001H',    /* DLLCONNECT user type field */
=       tcl$protocol$code$rev LITERALLY '0150H',    /* packet header user type field */

=                                       /* Misc values */
=       tcl$mip$port        LITERALLY   '4',        /* mip port for IP$IN$MBX */
=       log$rb$mip$port     LITERALLY   '5',        /* debugging: mip port for logging */
=       mip$echo$port       LITERALLY   '7',        /* mip port of on-bd tcl echo server */

=       tcl$version$lit     LITERALLY   '101H',     /* Version of this TCL for seg header */
=       def$net$id$lit      LITERALLY   '1',        /* default Network ID: "this network" */
=       on$bd$tcl$echo$port LITERALLY   '7',        /* TCL port of on-board tcl echo server */
=       true                LITERALLY   'OFFH',
=       false               LITERALLY   '0',
=       forever             LITERALLY   'WHILE true',

=       Timeout$increase$state  LITERALLY '1',      /* In this state the retransmission timeout
=                                                      is rapidly increased */
=       Timeout$steady$state    LITERALLY '0';      /* In this state the timeout is
=                                                   slowly decreased.  This should not be
=                                                   changed, it is the initial state since
=                                                   a cdb is intialised to zero */
    $ENDIF
        /* Some variables */
3   1   DECLARE
            lcid$vector(*)       WORD   EXTERNAL,
            loc$net              WORD   EXTERNAL,
            loc$host(3)          WORD   EXTERNAL,
            max$seg$data$len     WORD   EXTERNAL,
            max$window$size      BYTE   EXTERNAL,   /* in IP */
            cur$cdb$index        BYTE,
            cur$cid              WORD,
            tot$pkts$retran      WORD   EXTERNAL,
            tot$pkts$rej         WORD   EXTERNAL,
            ayt$timer$dw         DWORD  EXTERNAL,
            ayt$count$max        WORD   EXTERNAL,
            j                    WORD,              /* DO index */
            tp$timestamp         DWORD,             /* timestamp temp storage */
        $IF dbg
        $ENDIF
            irb_type             BYTE;              /* non-based copy of current irb.type */

                                                    /* Definitions of the "flag" field */
                                                    /* in KAOS alarm control blocks */
4   1   DECLARE
            timer$running        LITERALLY   '0',    /* alarm was SET, has not expired */
            timer$expired        LITERALLY   '1',    /* alarm was SET, and has expired */
            timer$cleared        LITERALLY   '2';    /* alarm has been CLEARED or CREATED */
```

```
            $IF f7
            $ELSE
            $SAVE NOLIST INCLUDE(:F1:TCLMBX.INC)

  6   1     DECLARE
              sp$p            POINTER,          /* Send Packet */
              sp$o            WORD AT(@sp$p),
              sp BASED sp$o                     /* Based on offset for code size and sped */
            $IF f7
            $ELSE
            $INCLUDE(:F1:TCLSEG.INC)
        =                                                  /**********************/
        =                                                  /*** Segment Format **/
        =                                                  /**********************/
        =     STRUCTURE (                                     /*  11/15/81  */
        =                              /* Offset (hex/ dec) */
        =         kaos$msg$hdr POINTER,      /*  0 =  0T  ptr to link seg bufs */
        =         buf$len      WORD,         /*  4 =  4T  # of bytes (dest addr thru last data byte) */
        =                                                 /* DATA LINK FIELDS */
        =         dl$dest(3)   WORD,         /*  6 =  6T  data link destination */
        =         dl$source(3) WORD,         /*  C = 12T  data link source */
        =         dl$type      WORD,         /* 12 = 18T  */
        =                                                 /* TRANSPORT CONTROL FIELDS */
        =         tcl$version WORD,          /* 14 = 20t Version number of sending TCL */
        =         dest$port   WORD,          /* 16 = 22T */
        =         source$port WORD,          /* 18 = 24T */
        =         dest$cid    WORD,          /* 1A = 26T */
        =         source$cid  WORD,          /* 1C = 28T */
        =         seg$seq$no  WORD,          /* 1E = 30T */
        =         seg$ack$no  WORD,          /* 20 = 32T */
        =         seg$data$len WORD,         /* 22 = 34T  # segment data bytes:for pad */
        =         ctl          WORD,         /* 24 = 36T  subfields mapped out as:
        =             (reserved)      bit,                8000H
        =             cks             bit,                4000H
        =             ayt             bit,                2000H
        =             ack             bit,                1000H
        =             syn             bit,                0800H
        =             fin             bit,                0400H
        =             eom             bit,                0200H
        =             rst             bit,                0100H
        =             (reserved)      bit,                0040H
        =             credit        6 bits                003FH       */
        =         checksum    WORD,          /* 26 = 38T */
        =         seg$data(1) BYTE) /* 28 = 3AT segment data bytes, length unknown */
            $ENDIF
                ,
                sp$array BASED sp$p (tcl$header$len) BYTE;

            $IF f7
            $ELSE
            $INCLUDE (:F1:TCLSCF.INC)
        =                                                  /*************************/
        =                                                  /* Segment Control Field */
        =                                                  /*************************/
        =   /*              TCL Segment Control Field Definitions                11/15/81 */
  7   1 =   DECLARE
        =       cks$mask        LITERALLY   '4000H',    /* declared for future use */
```

```
          =       ayt$mask          LITERALLY   '2000H',
          =       ack$mask          LITERALLY   '1000H',
          =       syn$mask          LITERALLY   '0800H',
          =       fin$mask          LITERALLY   '0400H',
          =       eom$mask          LITERALLY   '0200H',
          =       synack$mask       LITERALLY   '1800H',
          =       finack$mask       LITERALLY   '1400H',
          =       eom$ack$mask      LITERALLY   '1200H',
          =       eom$fin$mask      LITERALLY   '0600H',
          =       rst$mask          LITERALLY   '0100H',
          =       ctl$bits$mask     LITERALLY   '1F00H',
          =       rst$ack$mask      LITERALLY   '1100H',
          =       credit$mask       LITERALLY   '003FH';
              $ENDIF
  8   1       DECLARE
              cur$cdb$p       POINTER,
              (cur$cdb$o, cur$cdb$b)  WORD AT (@cur$cdb$p),
                                                    /**********************/
                                                    /*** Conn Data Base **/
                                                    /**********************/

              c BASED cur$cdb$p
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TCLCDB.INC)
              ,irb$p      POINTER,
              (irb$o, irb$b) WORD AT(@irb$p),
                                                    /**********************/
                                                    /******  IRB   *******/
                                                    /**********************/
              irb BASED irb$p
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TCLIRB.INC)
              ,
              lirb BASED irb$p
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TLIRB.INC)
              ,
              acb BASED irb$p             /* The alarm control block */
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TCLACB.INC)
              ;
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TCLIRC.INC)
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TCLRBC.INC)
              $IF f7
              $ELSE
              $SAVE NOLIST INCLUDE (:F1:TCLCSD.INC)

              /* External Procedure declarations */

  12  1       setup$cdb: PROCEDURE(index, cdb$p$o) EXTERNAL;          /* in IP */
```

```
13   2          DECLARE   index BYTE,   cdb$p$o WORD;
14   2          END setup$cdb;

15   1      delete$cdb: PROCEDURE(cdb$index, cdb$p, rtn$code) EXTERNAL;     /* In IP */
16   2          DECLARE (cdb$index, rtn$code)   BYTE,
                    cdb$p POINTER;
17   2          END delete$cdb;

18   1      gt$mod64k: PROCEDURE(n,m) BYTE EXTERNAL;                        /* In RP */
19   2          DECLARE (n,m) WORD;
20   2          END gt$mod64k;

21   1      ge$mod64k: PROCEDURE(n,m) BYTE EXTERNAL;                        /* In RP */
22   2          DECLARE (n,m) WORD;
23   2          END ge$mod64k;

24   1      max$mod64k: PROCEDURE(n,m) WORD EXTERNAL;                       /* in RP */
25   2          DECLARE (n,m)   WORD;
26   2          END max$mod$64k;

27   1      min: PROCEDURE(n,m) WORD EXTERNAL;                              /* in RP? */
28   2          DECLARE (n,m)   WORD;
29   2          END min;

30   1      try$to$delete$cdb: PROCEDURE(cdb$index, cdb$p, resp$code) EXTERNAL;     /* in RP */
31   2          DECLARE cdb$p   POINTER,
                    (cdb$index,resp$code) BYTE;
32   2          END try$to$delete$cdb;

33   1      search_lcid$vector: PROCEDURE(target) WORD EXTERNAL;                    /* in IP */
34   2          DECLARE target  WORD;
35   2          END search_lcid$vector;

36   1      max: PROCEDURE(n,m) WORD EXTERNAL;                          /* in TCOM */
37   2          DECLARE (n,m) WORD;
38   2          END max;

39   1      chk$sum$calc: PROCEDURE(seg$o) WORD EXTERNAL;               /* in TCOM */
40   2          DECLARE seg$o WORD;
41   2          END chk$sum$calc;

42   1      stky_incr: PROCEDURE(wd$p) EXTERNAL;                    /* in tcom */
43   2          DECLARE wd$p    POINTER;
44   2          END stky_incr;

            $IF log
            $ENDIF

            $IF f7
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:KAOS.DCP)

            $IF f7
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:MIP.DCP)

            $IF f7
```

```
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:DLL.DCP)

            $IF f7
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:THACF.INC)

            $IF log
            $ENDIF

            $IF log
            $ENDIF
                                                        /*************************/
                                                        /**  clr$set$data$alarm **/
                                                        /*************************/
146   1     clr$set$data$alarm: PROCEDURE(csda$type);

                                  /* code-saver routine to clear and set the data
                                  alarm */
147   2     DECLARE csda$type BYTE;

148   2         CALL cq$clear$alarm(@c.data$alarm$cb);

149   2         c.data$acb$irb$type = csda$type;
150   2         CALL cq$set$alarm(@c.data$alarm$cb, .tp$mbx,
                                                  high(c.retran$to$dw),
                                                  low (c.retran$to$dw));
151   2     END clr$set$data$alarm;

                                                        /**********************/
                                                        /** check$ayt$timer **/
                                                        /**********************/
152   1     check$ayt$timer: PROCEDURE(cat$cdb$p) PUBLIC;
                                  /* called to check if there are any RBs on the
                                  transmit queue, and to start the Are-You-There
                                  timer if not.  */
153   2     DECLARE
              cat$cdb$p   POINTER,          /* ptr to the cdb of interest */
              cat$c BASED cat$cdb$p
            $IF f7
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:TCLCDB.INC)
              ;
                                  /* called when we want to see if the ayt timer is
                                  needed because the xmit list is empty.  It is
                                  always ok to zero the count and clear the timer. */
154   2     cat$c.ayt$count = 0;
155   2     CALL cq$clear$alarm(@cat$c.ctl$alarm$cb);
156   2     IF cat$c.cbtq$buf$cnt = 0 THEN
157   2         DO;                      /* note: all conns have same ayt timeout for now */
158   3             cat$c.ctl$acb$irbtype = irb$ctl$timeout$mask OR irb$ayt$timer;
159   3             CALL cq$set$alarm(@cat$c.ctl$alarm$cb, .tp$mbx,
                                                          high(ayt$timer$dw),
                                                          low (ayt$timer$dw));
160   3         END;
161   2     END check$ayt$timer;
```

```
                                                        /******************/
                                                        /** get$tx$buf   **/
                                                        /******************/
162   1     get$tx$buf: PROCEDURE;
                                    /* code-saver routine to get a free transmit */
                                    /* buffer from data link (or whoever) */
163   2        CALL cq$signal(.sched$lock);    /* release the cdb lock while waiting for buf */
164   2        sp$p = cq$receive(cq$dll$tx$free$mbx);
165   2        CALL cq$waitsem(.sched$lock);   /* re-capture the lock */
                                    /* (Changed on 9/12 to try to correct excessive */
                                    /* number of empty acks being sent ) */
166   2     END get$tx$buf;

                                                        /******************/
                                                        /****  Get     ****/
                                                        /******************/
167   1     get: PROCEDURE BYTE;
                                    /* The Get routine accepts a segment number as a
                                    parameter and either returns false if that packet
                                    # is not available or returns true if it is (and
                                    also gets a transmit pkt buf and copies the data
                                    from the host into the pkt buffer. */

                                  /* This version of GET supports the full-up fragmentation
                                  scheme: the buffers in Send RBs will be fragmented as,
                                  necessary to fit them in packets, regardless of the
                                  size or number of buffers.  Zero-length blk are ignored,
                                  and cause no errors or side-effects.  Zero-length buffers
                                  (i.e., a Send with no blks, or all blks of zero length,
                                  and no other sequence-consuming controls such as EOM or FIN)
                                  are also ignored and do not cause any packet to be sent.
                                  A zero-length RB consumes no sequence number, and will
                                  remain on the xmit queue until a non-zero successor RB
                                  has been sent and acknowledged.  Exception: a zero-length
                                  Send RB submitted when the xmit queue is already empty
                                  will simply be returned immediately, marked "OK".
                                  See also the try$to$delete$cdb in IP. */

168   2     DECLARE
              target$seg      WORD,           /* sequence number of the target segment requested */
              target$offset   WORD,           /* Offset from RB of 1st byte of target segment */
              blk$index       WORD,           /* number of the current block */
              cum_rb_bytes    WORD,           /* total client bytes so far in this RB, as of last block */
              blk$offset      WORD,           /* offset of current byte w/in this block */
              blk$rem$bytes   WORD,           /* remaining bytes in this block */
              seg$offset      WORD,           /* offset of current byte in xmit seg buffer */
              seg$rem$bytes   WORD,           /* remaining space avail in xmit seg buffer */
              copylen         WORD,           /* bytes being copied to xmit seg buf this iteration */
              blk$data$p      POINTER,
              blk$data BASED blk$data$p (1) BYTE, /* client data vector in current blk */
              get$rbs$p       POINTER,
                                                        /**********************/
                                                        /***   RBS Format   ***/
                                                        /**********************/
              get$rbs BASED get$rbs$p
            $IF f7
```

```
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:TCLRBS.INC)
                ,
                get$rbv$p               POINTER,
                get$rbv BASED get$rbv$p (1)
            $IF f7
            $ELSE
            $SAVE NOLIST INCLUDE (:F1:TCLRBV.INC)
                ;

                        /* Routine used by Get */

169   2     no$more$blks: PROCEDURE BYTE;
                                        /* routine to set up the next block in this
                                        RB for use in the copy loop.  Returns true
                                        if there are no more non-zero blocks in this
                                        RB. */
170   3     nmb$top:
            blk$index = blk$index + 1;  /* bump the current block index */
171   3     IF blk$index >= get$rbs.num$blks THEN RETURN(true);     /* quit since there are no more blocks */
173   3     IF get$rbv(blk$index).blk$len = 0 THEN GOTO nmb$top;    /* ignore zero-length blks */
                                        /* found next non-zero blk--set up ptrs */
175   3     blk$rem$bytes = get$rbv(blk$index).blk$len;
176   3     blk$offset = 0;
            $IF mipform
177   3     blk$data$p = cq$mip$get$address( get$rbv(blk$index).blk$ptr);
            $ELSE
            $ENDIF
178   3     RETURN(false);
179   3     END no$more$blks;

180   2     set$up$seg$ctl$fields: PROCEDURE;
                                        /* This routine sets up the ctl, segseqno, */
                                        /* segdatalen, and buflen fields of a */
                                        /* transmit segment buffer; it is called */
                                        /* within Get if the xmit seg buf is full */
                                        /* or there is no more data in the current */
                                        /* Request Block. */
                                        /* set basic control field */
181   3                 sp.ctl = ack$mask OR min(max$window$size, c.my$credit);
                                        /* if we have copied last byte (if any) of
                                        the RB, then check to see if other controls
                                        should be inserted. */
182   3                 IF target$offset = get$rbs.buf$len THEN     /* we have passed last */
183   3                     DO;           /* byte (test doesn't chk for last blk because */
                                        /* there might be trailing zero blks) */
184   4                       IF get$rbs.req = send$eom$req THEN sp.ctl = sp.ctl OR eom$mask;
186   4                       IF get$rbs.req = close$req THEN sp.ctl = sp.ctl OR fin$mask;
188   4                     END;
                                        /* now fill in other seg header fields */
189   3                 sp.seg$seq$no = target$seg;
190   3                 sp.seg$data$len = seg$offset;   /* number of client data bytes */
                                        /* tell DLL the size of the buffer -- note */
                                        /* padding to accomodate Ethernet minimum */
                                        /* packet length. */
```

```
191   3                 sp.buf$len = dll$header$len +
                                max (min$pkt$len, tcl$header$len + seg$offset);
192   3                 RETURN;

193   3     END set$up$seg$ctl$fields;


                        /* main line code for Get */

194   2     IF c.cbtq$buf$cnt = 0 THEN RETURN(false);   /* There's no data to send */
196   2     get$rbs$p = c.cbtq$hdr;                     /* set up base on first RB */


                                          /* chk if target seq is too old, not in queue */
197   2     target$seg = c.next$transmit;  /* get always looks for this (implied param) */
198   2     IF gt$mod64k(get$rbs.first$seq,target$seg) THEN call Cqhaltandcatchfire(hacftqseq);  /* 4/15/82 */

                                          /* first, find the RB containing the first */
                                          /* byte of the target seg */
200   2     DO WHILE ( (NOT get$rbs.contents) OR gt$mod64k(target$seg, get$rbs.last$seq) );
201   3         IF (get$rbs$p := get$rbs.link) = 0 THEN return(false);
203   3     END;
                                          /* we have the RB containing start of target */
                                          /* segment, now find the block */

204   2     get$rbv$p = @get$rbs.vb;      /* set up variable part of RB addressing */
                                          /* first compute the byte offset of target */
                                          /* within RB for comparison to each blk */
205   2     target$offset = (target$seg - get$rbs.first$seq) * max$seg$data$len;
206   2     blk$index,blk$rem$bytes,blk$offset = 0;
207   2     cum_rb_bytes = 0;             /* total bytes encountered in this RB, */
                                          /* as of the last blk */

                                          /* Now, find the right block: if there */
                                          /* is any client data in this RB, search */
                                          /* thru its blks until we find the blk */
                                          /* containing the start of the target */
                                          /* seg (since contents is true, there is */
                                          /* something to send, but it may be just */
                                          /* an EOM or FIN with no client data). */
208   2     IF get$rbs.buf$len <> 0 THEN
209   2         DO;
210   3             DO WHILE (cum_rb_bytes + get$rbv(blk$index).blk$len) <= target$offset;
211   4                 cum_rb_bytes = cum_rb_bytes + get$rbv(blk$index).blk$len;
212   4                 blk$index = blk$index + 1;
213   4                 IF blk$index >= get$rbs.num$blks THEN   /* Error : target must be in this RB! */
214   4                     CALL cq$halt$and$catch$fire(hacf$tp$get$chk);
215   4             END;
                                          /* we now know which RB and which block */
                                          /* within that RB holds the first byte of */
                                          /* the target segment -- now compute the */
                                          /* offset within the block of that byte */

                        /* NOTE: some of this ought to be common with the
                        no-more-blks code ! */

216   3             blk$offset = target$offset - cum_rb_bytes;
```

```
217   3             blk$rem$bytes = get$rbv(blk$index).blk$len - blk$offset;    /* bytes remaining in blk */
          $IF mipform
218   3             blk$data$p = cq$mip$get$address( get$rbv(blk$index).blk$ptr);   /* address of start of blk */
          $ELSE
          $ENDIF
219   3         END;
                                          /* now get a transmit buffer and fill it */
                                          /* in with some data */
220   2       CALL get$tx$buf;
              /*
                check if we received an ack for this packet when we released the lock.
                if we did, then return the buffer and exit
              */
221   2       if target$seg <> c.next$transmit then
222   2       do;
223   3         call cqsend(cqdll$tx$free$mbx,sp$p);
224   3         return (FALSE);
225   3       end;
226   2       c.sendflag = false;     /* clr flag after re-aquiring semaphore since we */
                                        /* will copy seq/ack state into xmit seg now, and */
                                        /* a new sendflag from another process when it next */
                                        /* gains control should cause a new pkt to go out. */
227   2       seg$rem$bytes = max$seg$data$len;   /* NOTE ETHERNET SPECIFIC PKT LENGTH */
228   2       seg$offset = 0;
                                          /* Now go into the copy loop, trying to fill */
                                          /* the seg buffer with client data, using */
                                          /* multiple blks if necessary */
229   2       DO forever;
                                          /* on each time thru the loop, we copy the
                                          lesser of the number of bytes still available
                                          in the current block, and the amount of space
                                          left in the segment buffer.  If we don't fill
                                          the seg from one blk, try the next one, too. */
230   3           copylen = min ( blk$rem$bytes, seg$rem$bytes);
                                          /* copy in word mode for speed */
                  /* NOTE: set bus priority switch here! */
                                          /* if odd # of bytes, we will copy on next
                                          higher word boundary to avaoid odd-byte fixup.
                                          This is guaranteed to be ok as long as either:
                                          (1) there is at least one junk byte at xmit
                                          buf end (there are 4 bytes now), or: (2) the
                                          max seg data len is even. */
                                          /* note divide by two using shift-right for speed */
231   3           CALL MOVW(@blk$data(blk$offset), @sp.seg$data(seg$offset), SHR(copylen+1,1) );

                  /* NOTE: clear bus priority switch here! */
                                          /* now pick up the odd byte, if there was one */
              /*IF copylen THEN */
                  /*sp.seg$data(seg$offset+copylen-1) = blk$data(blk$offset+copylen-1);*/
                                          /* update all the offsets */
232   3           seg$offset = copylen + seg$offset;
233   3           blk$offset = copylen + blk$offset;
234   3           seg$rem$bytes = seg$rem$bytes - copylen;
235   3           blk$rem$bytes = blk$rem$bytes - copylen;
236   3           target$offset = target$offset + copylen;    /* make this track for later use */

                                          /* Now see if we are done with the seg */
```

```
237   3           IF seg$rem$bytes = 0 THEN
238   3               DO;                              /* no more space in xmit buf so fill */
239   4                   CALL set$up$seg$ctl$fields; /* fill in the control fields and quit */
240   4                   RETURN(true);
241   4               END;
242   3           IF no$more$blks THEN
243   3               DO;                              /* there's seg buffer space, but we have */
244   4                   CALL set$up$seg$ctl$fields; /* emptied this RB, and current frag policy */
245   4                   RETURN(true);               /* doesn't cross RB boundaries on send, so */
246   4               END;                             /* fill in ctl fields and exit */

247   3       END;    /* of do forever loop */

248   2       END get;

                                                              /**********************/
                                                              /*** send$packet   ***/
                                                              /**********************/
249   1       send$packet: PROCEDURE;
                                                  /* a code-saver routine to compute
                                                  the segment checksum, send it via
                                                  Data Link, and log the event */
                $IF dbg
                $ENDIF

250   2           sp.dl$type = tcl$protocol$code$rev; /* Insert TCL's Ethernet type code */
                                                      /* (I shouldn't be doing this, either) */
251   2           sp.checksum = chk$sum$calc(sp$o);   /* put in the checksum */
                $IF dbg
                $ENDIF
                $IF log
                $ENDIF
                                                    /* release the cdb lock while sending so that */
252   2                     CALL cq$signal(.sched$lock);   /* RP will not be blocked if this is a loop-back send */
253   2                     CALL cq$dll$tx$send(sp$o);     /* Send the segment */
254   2                     CALL cq$waitsem(.sched$lock);  /* re-capture the lock */
                $IF dbg
                $ENDIF
                                                /* when we waited on the tx buf */
255   2       END send$packet;

                                                              /**********************/
                                                              /** setup$seg$header */
                                                              /**********************/
256   1       setup$seg$header: PROCEDURE;
257   2           sp.dl$dest(0) = c.rem$host(0);
258   2           sp.dl$dest(1) = c.rem$host(1);
259   2           sp.dl$dest(2) = c.rem$host(2);
                                              /* Now do transport fields */
260   2           sp.tcl$version = tcl$version$lit;
261   2           sp.dest$port = c.rem$port;
262   2           sp.source$port = c.loc$port;
263   2           sp.dest$cid = c.rem$cid;        /* remcid init'd to zero: ok for SYN */
264   2           sp.source$cid = c.loc$cid;
265   2           sp.seg$ack$no = c.my$ack$no;
266   2       END setup$seg$header;
```

```
                                                        /*************************/
                                                        /**  bld$send$ctl$seg   **/
                                                        /*************************/
267   1     bld$send$ctl$seg: PROCEDURE(ctl$field, ctl$seq$no);
                                                        /* Build and send a control segment */
268   2     DECLARE
                ctl$field       WORD,
                ctl$seq$no      WORD;
                                                        /* first build it */
269   2     CALL get$tx$buf;
270   2     c.sendflag = false;     /* clr flag after re-aquiring semaphore since we */
                                      /* will copy seq/ack state into xmit seg now, and */
                                      /* a new sendflag from another process when it next */
                                      /* gains control should cause a new pkt to go out. */
                                            /* fill in the Data Link fields (TCL really
                                               shouldn't be doing this ... */
271   2     CALL setup$seg$header;
272   2     sp.seg$seq$no = ctl$seq$no;
273   2     sp.buf$len = dll$header$len + min$pkt$len; /* Pad buffer length to meet minimum */
                                                        /* Ethernet Data Link spec.  Change */
                                                        /* This if tcl$header$len gets  */
                                                        /* greater than 46 bytes */
274   2     sp.seg$data$len = 0;                        /* no client data bytes in control msgs */
275   2     sp.ctl = ctl$field;
276   2     IF ctl$field = syn$mask THEN
277   2         DO;                                     /* syn packets are special: */
278   3             sp.seg$ack$no = 0;                  /* syn always has zero ack - helps keep */
                                                        /* duplicate SYN's from causing RST. */
279   3             sp.dest$cid = 0;                        /* dest cid unknown-part of RST stuff. */
280   3         END;
281   2     ELSE sp.ctl = sp.ctl OR min(max$window$size, c.my$credit);

                                                      /* param order reversed 4-17-81. May need it elsewhere */
282   2     c.highest$sent = max$mod64k(sp.seg$seq$no, c.highest$sent);
283   2     CALL send$packet;                         /* compute chksum, send, log */

284   2     END bld$send$ctl$seg;


                                                        /*************************/
                                                        /**  incr$retran$counts **/
                                                        /*************************/
285   1     incr$retran$counts: PROCEDURE;
                                    /* routine to increment (sticky counters) re-xmit-
                                    related counters and check for abort timeout.  If
                                    abort timeout exceeded, it aborts the connection,
                                    and enters the Closed state to await an ACK/Reset. */

                                    /* note retransmit count in counters: THESE ARE
                                    APPROXIMATE ONLY, AND ARE ACTUALLY MEASURING
                                    THE NUMBER OF TIMES THE RE-TRANSMIT TIMER WENT
                                    OFF, NOT THE NUMBER OF PACKETS RETRANSMITTED !*/

286   2             CALL stky_incr (@c.no$confid);  /* sticks at max value */
287   2             CALL stky_incr(@c.pkts$retran);
288   2             CALL stky_incr(@tot$pkts$retran);
```

```
                                    /* now see if we have exceeded the abort timeout */
                                    /* ... approximated as the no-confid count times the *
                                    /* retransmit timeout value. */

                                    /* if abort timeout isn't infinite, check for */
                                    /* timer expired */
289   2             IF c.abort$to$hi <> 0FFFFH THEN
290   2                 DO;

            /* C.NOCONFID NEEDS TO BECOME A WORD, BUT NOT RIGHT NOW. */
            /* 12/13/81 - has been made a word - RJ */

                                    /* add in last timeout to cumulative retran time */
291   3                 c.cum$retran$dw = c.cum$retran$dw + c.re$tran$to$dw;
                                    /* now see if cum retran time is >= abort timeout */
                                    /* (since lsw of abort timeout is always zero, we */
                                    /* can just check the msw of each) */
292   3                 IF high(c.cum$retran$dw) >= c.abort$to$hi THEN
293   3                     DO;     /* yes it has ... close down the connection */
294   4                         CALL try$to$delete$cdb(cur$cdb$index, cur$cdb$p, loc$timeout);
        $IF log
        $ENDIF
295   4                     END;
296   3                 END;
297   2     END incr$retran$counts;
                                                    /*************************/
                                                    /**  seg$tran$timestamp **/
                                                    /*************************/
298   1     seg$tran$timestamp: PROCEDURE(seg$seq$no_to_time);
                                    /* tries to timestamp the segment being sent
                                    so that the roundtrip time of the ack can be
                                    used (in RP) to adapt the retransmit timer */
299   2     DECLARE seg$seq$no_to_time  WORD;

300   2     IF c.timed$seq$no <> 0 THEN RETURN; /* return if a seg is already being timed */
302   2     CALL cq$read$clock(.tp$timestamp);
303   2     c.seg$trans$time$dw = tp$timestamp;
304   2     c.timed$seq$no = seg$seq$no_to_time;    /* save so RP knows which seg is timed */

305   2     END seg$tran$timestamp;

                                                  /**********************/
                                                  /**   tp$err$flow    **/
                                                  /**********************/
306   1     tp$err$flow: PROCEDURE BYTE;
307   2     DECLARE
            (in$window, show$new$data,  send$data$now)   BYTE,
              temp$next$transmit word,    /* 4/1/82 fix */
            ctl$field       WORD;

          /* If this runs under a pre-emptive scheduler, this routine and its
             counter-part in the receive process should lock the connection
             data base while being executed */

          /* old-credit, old-ack-no detection occurs as a sendflag event caused by
             receive process; don't test them here.  ??? Re-check this ... */
```

```
308   2       IF cq$check$alarm(@c.data$alarm$cb) THEN
309   2           DO;                         /* we have a re-transmit event: the ACK timer went */
                                              /* off - need to reset xmit seq no and retransmit */
310   3               c.next$transmit = c.his$ack$no + 1;
311   3               CALL incr$retran$counts;    /* increment all the retransmit counters */
312   3               IF c.state = closed THEN RETURN(false);     /* abort timeout went off - quit */
314   3           END;
                                              /* see if the seg no I want to send (next$transmit) */
                                              /* is inside the receive window of the remote guy */
315   2       in$window = ge$mod64k( (c.his$ack$no + c.his$credit), c.next$transmit );

                                              /* also see if the seg I want to send next is the */
                                              /* one just after the last one remote guy has acked */
316   2       show$new$data = (c.next$transmit = (c.his$ack$no + 1) );
                                              /* We will send data if: (1) we have some to send, */
                                              /* and (2) either our next seg is within his window */
                                              /* or (even if it isn't) it is necessary to send */
                                              /* it to ensure that we aren't blocked by a zero */
                                              /* window (the new credit pkt may have been lost).*/
317   2       send$data$now = (in$window OR show$new$data) AND (c.cbtq$buf$cnt <> 0);

318   2       IF send$data$now THEN
319   2           DO;
320   3               temp$next$transmit = c.next$transmit;       /* 4/1/82 */
321   3               IF get /* (c.next$transmit)*/ THEN  /* param is implied, not explicit */
322   3                   DO;                 /* send the data segment that GET just filled */

323   4                       CALL setup$seg$header;  /* "Get" already set seg$seq$no */
324   4                       CALL send$packet;       /* compute chksum, send, log */
                                                      /* time the roundtrip time of ack to seg */
325   4                       if temp$next$transmit  <> c.next$transmit then
326   4                       DO;
                                  /* rp must have run and changed next transmit,
                                     so change it back */
327   5                           C.next$transmit = temp$next$transmit;
328   5                       END;

329   4                       CALL seg$tran$timestamp(c.next$transmit);

330   4                       c.highest$sent = max$mod64k(c.highest$sent, c.next$transmit);

                                                      /* sendflag was cleared after get$tx$buf */
331   4                       IF show$new$data THEN   /* set re-tran timer in the special */
                                                      /* case that this is top seg */
332   4                           CALL clr$set$data$alarm(irb$sendcheck OR irb$timeout$mask);
333   4                       c.next$transmit = c.next$transmit+1;
334   4                       RETURN(true);           /* true => we sent some data */
335   4                   END;
336   3           END;

337   2       IF c.sendflag THEN
338   2           DO;                         /* SEND A CONTROL SEGMENT */
339   3               CALL bld$send$ctl$seg(ack$mask,c.highest$sent);
                                              /* sendflag was cleared after get$tx$buf */
340   3           END;

341   2       IF (c.cbtq$buf$cnt <> 0) AND (c.data$acb$flag <> timer$running) THEN
```

```
342   2         DO;
                        /* NOTE: we are providing protection against the case where */
                        /* the data timer expires, but we cannot send any data (due  */
                        /* to remote window, presumably). */
343   3             CALL clr$set$data$alarm(irb$send$check OR irb$timeout$mask);
344   3         END;

345   2       RETURN(false);

346   2       END tp$err$flow;
                                                        /*********************/
                                                        /***  send$connect$seg  ***/
                                                        /*********************/
347   1       send$connect$seg: PROCEDURE(ctl$field);
                        /* common code for the send-control-seg logic, cases 0,1,7
                        of irb type.  If this not an alarm timeout then use next
                        transmit as seg seq no, and bump it for next time; however,if
                        this is not an alarm, then next transmit was previously bumped,
                        so we use highest sent, which is always ok for a re-tran of a
                        ctl seg.  */

348   2       DECLARE
                    ctl$field   WORD;
349   2       IF (irb_type AND irb$timeout$mask) <> 0 THEN
350   2           DO;
351   3               CALL incr$retran$counts;    /* keep track of how many times we tried */
352   3               IF c.state = closed THEN RETURN;    /* abort timeout went off - quit */
354   3               CALL bld$send$ctl$seg(ctl$field, c.highest$sent);       /* re-xmit */
355   3           END;
356   2       ELSE                                /* Virtual or real "first time" */
                    DO;                         /* for this req, not re-xmit timeout */
357   3               CALL bld$send$ctl$seg(ctl$field, c.next$transmit);
358   3               c.next$transmit = c.next$transmit + 1;
359   3           END;
360   2           c.timed$seq$no = 0;    /* start timer each time it is sent */
361   2           CALL seg$tran$timestamp(c.next$transmit);   /* time the roundtrip time of ack to seg */
                                        /* Now set re-transmit timer in case pkt is lost */
                                        /* Set type to be time-out retry: */
362   2           CALL clr$set$data$alarm( (irb_type AND irb$type$mask) OR irb$timeout$mask );

363   2       END send$connect$seg;

                                                        /************************/
                                                        /** send$rst$from$lirb **/
                                                        /************************/
364   1       send$rst$from$lirb: PROCEDURE;
                                        /* Sends an RST segment.  The fields for */
                                        /* the seg are in the irb, which is */
                                        /* in lirb format for this type request */

365   2                 CALL get$tx$buf;          /* get a seg buf */
366   2                 sp.buf$len = dll$header$len + min$pkt$len;  /* no client data */
367   2                 sp.dl$dest(0) = lirb.dl$dest(0);
368   2                 sp.dl$dest(1) = lirb.dl$dest(1);
369   2                 sp.dl$dest(2) = lirb.dl$dest(2);
370   2                 sp.tcl$version = tcl$version$lit;
371   2                 CALL MOVW(@lirb.dest$port, @sp.dest$port, 6);   /* these fields all aligned */
```

```
372   2                 sp.seg$data$len = 1;    /* No client data, but one byte of reason */
373   2                 sp.ctl = rst$ack$mask;
374   2                 sp.seg$data(0) = lirb.reason;
375   2                 CALL send$packet;       /* compute chksum, send, log it */

376   2       END send$rst$from$lirb;

                                                          /******************/
                                                          /***  sendloop  **/
                                                          /******************/
377   1       sendloop: PROCEDURE;
                                                  /* Code-saver loop to do send check and
                                                  send flag loop processing */
378   2           j=0;                           /*  send, or if the data timer expired */
379   2           DO WHILE tp$err$flow AND ((j:=j+1) < max$send$seg );/*  */
                                                  /* now let in RP while we're trying to */
380   3               CALL cq$signal(.sched$lock);/* run off several segments */
381   3               CALL cq$schedule;
382   3               CALL cq$waitsem(.sched$lock);
383   3           END;
384   2       RETURN;
385   2       END sendloop;



                                                        /**********************/
                                                        /****  trans$proc  ***/
                                                        /**********************/
386   1       trans$proc: PROCEDURE PUBLIC;
                                                  /* This is the Transmit Process */

387   2       DO forever;
388   3           irb$p = cq$receive(.tp$mbx);    /* wait for sendsyn irb or lirb from IP */

389   3           CALL cq$waitsem(.sched$lock);   /* lock up cdb's for safety */

            $IF dbg
            $ENDIF

390   3           irb_type = irb.type;            /* make non-based copy for speed and code size */
            $IF log
            $ENDIF

                                                  /* some consistency checking: type code */
                                                  /* must be good or we halt for debugging */
391   3           IF ((irb_type AND irb$type$mask) > irb$max$code) OR
                    ((irb_type AND irb$invalid$mask) <> 0) OR
                    ((irb_type AND irb$ctl$timeout$mask) = irb$ctl$acb$mask) THEN
392   3               DO;                        /* Its an illegal irb type - lock up */
393   4                   CALL cq$halt$and$catch$fire(hacf$tp$irb$type);
394   4               END;
                                                  /* Try to set up the cdb for processing by
                                                  validating the cdb index and lcid vector to
                                                  the extent possible.  Timer events have no
                                                  cid in the "irb", and all connection alarm cbs
                                                  are cleared at delete cdb time anyway, so no
```

```
                                        validation really needed - check for cid <> 0
                                        for the time being.  Real irb's have to match
                                        the irb cid with the cid in lcid vector. */

395   3           IF irb_type <> irb$sendrst THEN /* sendrst excluded because its indep of cdb */
396   3               DO;
397   4                   IF (irb_type AND irb$timeout$mask) = 0 THEN     /* its not an alarm cb, */
398   4                       DO;                         /* so cdbindex/cid are in irb */
399   5                           cur$cdb$index = irb.cdb$index;
400   5                           IF lcid$vector(cur$cdb$index) = 0 THEN  /* CDB has gone away: complain */
401   5                               DO;
          $IF log
          $ENDIF
402   6                                   GOTO give$back$irb;
403   6                               END;
404   5                           cur$cid = lcid$vector(cur$cdb$index);
405   5                           CALL setup$cdb(cur$cdb$index, .cur$cdb$p);
406   5                           IF lcid$vector(cur$cdb$index) <> irb.cid THEN
407   5                               DO;
          $IF log
          $ENDIF
408   6                                   GOTO give$back$irb;
409   6                               END;
410   5                       END;
411   4                   ELSE                        /* The irb is really an alarm cb, so derive */
                              DO;                     /* the cdb ptr from the known offset of ctl */
                                                      /* or data ACB in the conn db. First, filter */
                                                      /* out spurious timeouts (i.e, a clearalarm */
                                                      /* was issued, but too late to stop it from */
                                                      /* going off...note: this isn't supposed to */
                                                      /* happen anymore!), and cases where a clear */
                                                      /* or setalarm was issued after the cqreceive */
                                                      /* above, but before TP acquired the semaphore. */
412   5                       IF acb.flag <> 1 THEN   /* Its not an expired alarm */
413   5                           DO;
          $IF log
          $ENDIF
414   6                               GOTO give$back$irb; /* its obsolete: ignore it */
415   6                           END;
416   5                       cur$cdb$b = irb$b;  /* set up base portion of cdb ptr */
417   5                       IF (irb_type AND irb$ctlacb$mask) <> 0 THEN     /* its a ctl acb */
418   5                           cur$cdb$o = irb$o - (.c.ctl$alarm$cb(0) - .c);
419   5                       ELSE cur$cdb$o = irb$o - (.c.data$alarm$cb(0) - .c);
                                                      /* cur$cdb$p is now set up, so */
420   5           dummy$label:    cur$cid = c.loc$cid; /* assign value (label kills register history)  */

                                                      /* Now get the cdb index for the cid */
421   5                       IF (cur$cdb$index := search_lcid$vector(cur$cid)) = 0FFFFH THEN
422   5                           DO;                 /* the cid isnt there anymore */
          $IF log
          $ENDIF
423   6                               GOTO give$back$irb;
424   6                           END;

425   5                   END;
                                                      /* Check to see that the conn isn't in */
426   4               IF c.state = closed THEN /* Closed state - no TP request except */
```

```
427   4                     DO;                 /* send rst is valid in Closed */
          $IF log
          $ENDIF
428   5                         GOTO give$back$irb;
429   5                     END;
430   4               END;

431   3           DO CASE(irb_type AND irb$type$mask);        /* Upper bit indicates whether this*/
                                                  /* first time or a timeout of prev event */
                  /* case sendsyn - 0 */
432   4               DO;
433   5                   CALL send$connect$seg(syn$mask);
434   5               END;
                  /* case send synack - 1 */
435   4               DO;
436   5                   CALL send$connect$seg(syn$ack$mask);
437   5               END;
                  /* case sendfin - 2 */
                        /* THIS NEVER GETS EXECUTED NOW (4/29/81): IP DOES A
                        SEND CHECK ON CLOSE CALL.  TAKE IT OUT SOMEDAY. */
438   4               ;
                  /* case sendrst - 3 */
                                                  /* RP received a seg requiring an RST */
                                                  /* reply, or local client issued Abort */
439   4               CALL send$rst$from$lirb;    /* request: Send a Reset segment */

                        /* Note for send check and sendflag: letting the RP in while
                        we are still holding an irb is dead-lock prone, and has been
                        taken care of by making ip and rp defer the acquistion of an
                        irb until after they have released schedlock. */

                  /* case sendcheck - 4 */
440   4               DO;                         /* try to send some data; we get here if */
                                                  /* RP says remote tcl's window changed, */
                                                  /* or if IP says there is new data to */
441   5                   CALL sendloop;
442   5               END;

                  /* case sendflag - 5 */
443   4               DO;                         /* test to see if the shared variable */
                                                  /* Sendflag is set.  If not, ignore */
444   5                   IF c.sendflag THEN      /* this event since TP has already */
445   5                       DO;                 /* reacted to it */
446   6                           CALL sendloop;
447   6                       END;
448   5               END;

                  /* case timewait$to - 6 */
449   4               DO;
                                                  /* set state to closed, clr lists, clr alarms */
450   5                   CALL try$to$delete$cdb(cur$cdb$index, cur$cdb$p, ok$closed$resp);
          $IF log
          $ENDIF
451   5               END;

                  /* case ayt$timer - 7 */
452   4               DO;
```

```
                                        /* are-you-there timer expired - if the */
                                        /* aytcount has exceeded the max, abort */
453   5                 IF c.ayt$count <= ayt$count$max THEN        /* count hasn't hit max yet */
454   5                     DO;
455   6                         CALL bld$send$ctl$seg(ayt$mask OR ack$mask, c.highest$sent);
                                  /*
          03/10/82                   The RP could have run and set ctl timer, so see if
                                     it is set. Ignore this fct if it has
                                  */
456   6                         if c.ctl$acb$flag <> 0 then
457   6                         do;
458   7                           c.ayt$count = c.ayt$count + 1;
459   7                           CALL cq$set$alarm(@c.ctl$alarm$cb, .tp$mbx,
                                                    high(ayt$timer$dw),
                                                    low (ayt$timer$dw));
460   7                         end;
461   6                     END;
462   5                 ELSE
                            DO;                 /* count exceeded - close the connection */
                                                /* just like an abort timeout. */
463   6                         CALL try$to$delete$cdb(cur$cdb$index, cur$cdb$p, loc$timeout);
          $IF log
          $ENDIF
464   6                     END;
465   5             END;

466   4         END;    /* of do case */

467   3       give$back$irb:                    /* Now see if we have an IRB to send back */
              IF (irb_type AND irb$timeout$mask) = 0 THEN     /* its not an alarm acb, so send it back */
468   3           DO;                           /* to the right free mailbox */
469   4               IF irb_type <> irb$sendrst THEN
470   4                   DO;
471   5                       CALL cq$send(.free$irb$mbx, irb$p);
472   5                   END;
473   4               ELSE
                            DO;
474   5                       CALL cq$send(.free$lirb$mbx, irb$p);
475   5                   END;
476   4           END;
              /* else ignore it, because its an alarm cb in the connection data base */
                                        /* Release the processor temporarily to let */
                                        /* in RP in case we just sent him something */
                                        /* - maybe important in loopback. */

          $IF dbg
          $ENDIF

477   3           CALL cq$signal(.sched$lock);    /* free up cdb's */

478   3           CALL cq$schedule;

479   3       END; /* of forever loop */

480   2       END trans$proc;

481   1       END tp;
```

```
 DEFN  ADDR    SIZE  NAME, ATTRIBUTES, AND REFERENCES
-----  ------  ----  --------------------------------

   10                 ABORTREQ . . . . . .    LITERALLY '8'
    8  0000H     16   ACB. . . . . . . . .    STRUCTURE BASED(IRBP)
       0000H      4       NEXT . . . . . .    POINTER
       0004H      1       TYPE . . . . . .    BYTE
       0005H      1       FLAG . . . . . .    BYTE        412
       0006H      4       LAST . . . . . .    POINTER
       000AH      2       DLOW . . . . . .    WORD
       000CH      2       DHIGH. . . . . .    WORD
       000EH      2       MBX. . . . . . .    WORD
    7                  ACKMASK. . . . . . .    LITERALLY '1000H'       181  339  455
  110  0000H      4   ALARMP . . . . . . .    POINTER IN PROC (CQCLEARALARM) PARAMETER        110
  107  0000H      4   ALARMP . . . . . . .    POINTER IN PROC (CQCHECKALARM) PARAMETER        107
  101  0000H      4   ALARMP . . . . . . .    POINTER IN PROC (CQCREATEALARM) PARAMETER       101
  104  0000H      4   ALARMP . . . . . . .    POINTER IN PROC (CQSETALARM) PARAMETER          104
    3  0000H      2   AYTCOUNTMAX. . . . .    WORD EXTERNAL(8)        453
    7                  AYTMASK. . . . . . .    LITERALLY '2000H'       455
    3  0000H      4   AYTTIMERDW . . . . .    DWORD EXTERNAL(7)       159  459
  267  039CH    121   BLDSENDCTLSEG. . . .    PROCEDURE STACK=000EH         339  354  357  455
  168  0000H      1   BLKDATA. . . . . . .    BYTE BASED(BLKDATAP) ARRAY(1) IN PROC (GET)           231
  168  0026H      4   BLKDATAP . . . . . .    POINTER IN PROC (GET)         177*  218*  231
  168  0018H      2   BLKINDEX . . . . . .    WORD IN PROC (GET)      170*  170  171  173  175  177  206*  210
                                                211  212*  212  213  217  218
  168  001CH      2   BLKOFFSET. . . . . .    WORD IN PROC (GET)      176*  206*  216*  217  231  233*  233
  168  001EH      2   BLKREMBYTES. . . . .    WORD IN PROC (GET)      175*  206*  217*  230  235*  235
    5  0000H      2   BUFMIPMBX. . . . . .    WORD EXTERNAL(14)
  132  0000H      2   BUFO . . . . . . . .    WORD IN PROC (CQDLLRXRETBUF) PARAMETER          132
   10                 BUFTOOSHORT. . . . .    LITERALLY '8'
    8  0000H    124   C. . . . . . . . . .    STRUCTURE BASED(CURCDBP)      418  419
       0000H      1       STATE. . . . . .    BYTE        312  352  426
       0001H      1       OWNERDEVICE. . .    BYTE
       0002H      2       OWNERPROCESSID . .  WORD
       0004H      2       LOCCID . . . . .    WORD        264  420
       0006H      2       LOCPORT. . . . .    WORD        262
       0008H      2       REMNET . . . . .    WORD
       000AH      6       REMHOST. . . . .    WORD ARRAY(3)           257  258  259
       0010H      2       REMPORT. . . . .    WORD        261
       0012H      2       PERSIST. . . . .    WORD
       0014H      2       ABORTTOHI. . . .    WORD        289  292
       0016H      2       REMCID . . . . .    WORD        263
       0018H      4       RETRANTODW . . .    DWORD       150  291
       001CH      2       RESERVED . . . .    WORD
       001EH      2       TIMEDSEQNO . . .    WORD        300  304*  360*
       0020H      4       SEGTRANSTIMEDW . .  DWORD       303*
       0024H      4       CUMRETRANDW. . .    DWORD       291*  291  292
       0028H      2       PERSISTCNT . . .    WORD
       002AH      4       CBTQHDR. . . . .    POINTER          196
       002EH      4       PCBQHDR. . . . .    POINTER
       0032H      4       DEFSTATUSP . . .    POINTER
       0036H      2       MYACKNO. . . . .    WORD        265
       0038H      2       SEEN . . . . . .    WORD
       003AH      1       MYCREDIT . . . .    BYTE        181  281
       003BH      1       CURBLKINDEX. . .    BYTE
```

```
          003CH     2      CBDATAINDEX. . . .    WORD
          003EH     2      RCVBYTESCONSUMED .    WORD
          0040H     2      CURBLKLENLEFT. . .    WORD
          0042H     2      HISACKNO . . . . .    WORD        310  315  316
          0044H     2      NEXTTRANSMIT . . .    WORD        197  221  310*  315  316  320  325  327*  329  330
                                                    333*  333  357  358*  358  361
          0046H     1      CLOSEDREASON . . .    BYTE
          0047H     1      HISCREDIT. . . . .    BYTE        315
          0048H     2      HIGHESTSENT. . . .    WORD        282*  282  330*  330  339  354  455
          004AH     1      CBTQBUFCNT . . . .    BYTE        194  317  341
          004BH     1      PCBQBUFCNT . . . .    BYTE
          004CH     2      PKTSREJ. . . . . .    WORD
          004EH     2      PKTSRETRAN . . . .    WORD        287
          0050H     2      NOCONFID . . . . .    WORD        286
          0052H     2      LASTNOCONFID . . .    WORD
          0054H     1      RETRANSMITSTATE. .    BYTE
          0055H     1      SENDFLAG . . . . .    BYTE        226*  270*  337  444
          0056H     2      PENDINGRCVDATA . .    WORD
          0058H     2      RCVBUFREJCNT . . .    WORD
          005AH     2      AYTCOUNT . . . . .    WORD        453  458*  458
          005CH     4      DATAALARMCB. . . .    WORD ARRAY(2)          148  150  308  419
          0060H     1      DATAACBIRBTYPE . .    BYTE        149*
          0061H     1      DATAACBFLAG. . . .    BYTE        341
          0062H    10      DATAACBREM . . . .    BYTE ARRAY(10)
          006CH     4      CTLALARMCB . . . .    WORD ARRAY(2)          418  459
          0070H     1      CTLACBIRBTYPE. . .    BYTE
          0071H     1      CTLACBFLAG . . . .    BYTE        456
          0072H    10      CTLACBREM. . . . .    BYTE ARRAY(10)

  153     0000H   124  CATC . . . . . .    STRUCTURE BASED(CATCDBP) IN PROC (CHECKAYTTIMER)
          0000H     1      STATE. . . . . .    BYTE
          0001H     1      OWNERDEVICE. . .    BYTE
          0002H     2      OWNERPROCESSID . .  WORD
          0004H     2      LOCCID . . . . .    WORD
          0006H     2      LOCPORT. . . . .    WORD
          0008H     2      REMNET . . . . .    WORD
          000AH     6      REMHOST. . . . .    WORD ARRAY(3)
          0010H     2      REMPORT. . . . .    WORD
          0012H     2      PERSIST. . . . .    WORD
          0014H     2      ABORTTOHI. . . .    WORD
          0016H     2      REMCID . . . . .    WORD
          0018H     4      RETRANTODW . . .    DWORD
          001CH     2      RESERVED . . . .    WORD
          001EH     2      TIMEDSEQNO . . .    WORD
          0020H     4      SEGTRANSTIMEDW . .  DWORD
          0024H     4      CUMRETRANDW. . .    DWORD
          0028H     2      PERSISTCNT . . .    WORD
          002AH     4      CBTQHDR. . . . .    POINTER
          002EH     4      PCBQHDR. . . . .    POINTER
          0032H     4      DEFSTATUSP . . .    POINTER
          0036H     2      MYACKNO. . . . .    WORD
          0038H     2      SEEN . . . . . .    WORD
          003AH     1      MYCREDIT . . . .    BYTE
          0038H     1      CURBLKINDEX. . .    BYTE
          003CH     2      CBDATAINDEX. . .    WORD
          003EH     2      RCVBYTESCONSUMED .  WORD
          0040H     2      CURBLKLENLEFT. . .  WORD
```

```
        0042H     2       HISACKNO . . . . .   WORD
        0044H     2       NEXTTRANSMIT . . .   WORD
        0046H     1       CLOSEDREASON . . .   BYTE
        0047H     1       HISCREDIT. . . . .   BYTE
        0048H     2       HIGHESTSENT. . . .   WORD
        004AH     1       CBTQBUFCNT . . . .   BYTE        156
        004BH     1       PCBQBUFCNT . . . .   BYTE
        004CH     2       PKTSREJ. . . . . .   WORD
        004EH     2       PKTSRETRAN . . . .   WORD
        0050H     2       NOCONFID . . . . .   WORD
        0052H     2       LASTNOCONFID . . .   WORD
        0054H     1       RETRANSMITSTATE. .   BYTE
        0055H     1       SENDFLAG . . . . .   BYTE
        0056H     2       PENDINGRCVDATA . .   WORD
        0058H     2       RCVBUFREJCNT . . .   WORD
        005AH     2       AYTCOUNT . . . . .   WORD        154*
        005CH     4       DATAALARMCB. . . .   WORD ARRAY(2)
        0060H     1       DATAACBIRBTYPE . .   BYTE
        0061H     1       DATAACBFLAG. . . .   BYTE
        0062H    10       DATAACBREM . . . .   BYTE ARRAY(10)
        006CH     4       CTLALARMCB . . . .   WORD ARRAY(2)          155  159
        0070H     1       CTLACBIRBTYPE. . .   BYTE        158*
        0071H     1       CTLACBFLAG . . . .   BYTE
        0072H    10       CTLACBREM. . . . .   BYTE ARRAY(10)
  153   0004H     4  CATCDBP. . . . . . . .   POINTER IN PROC (CHECKAYTTIMER) PARAMETER AUTOMATIC        153  155
                                                156  159
   31   0000H     1  CDBINDEX . . . . . . .   BYTE IN PROC (TRYTODELETECDB) PARAMETER          31
   16   0000H     1  CDBINDEX . . . . . . .   BYTE IN PROC (DELETECDB) PARAMETER        16
   31   0000H     4  CDBP . . . . . . . . .   POINTER IN PROC (TRYTODELETECDB) PARAMETER      31
   16   0000H     4  CDBP . . . . . . . . .   POINTER IN PROC (DELETECDB) PARAMETER           16
   13   0000H     2  CDBPO. . . . . . . . .   WORD IN PROC (SETUPCDB) PARAMETER         13
  152   0036H    59  CHECKAYTTIMER. . . . .   PROCEDURE PUBLIC STACK=0012H
   39   0000H        CHKSUMCALC . . . . . .   PROCEDURE WORD EXTERNAL(25) STACK=0000H         251
    7                CKSMASK. . . . . . . .   LITERALLY '4000H'
   11                CLOSED . . . . . . . .   LITERALLY '9'           312  352  426
   10                CLOSEREQ . . . . . . .   LITERALLY '2'           186
   11                CLOSING. . . . . . . .   LITERALLY '8'
  146   0000H    54  CLRSETDATAALARM. . . .   PROCEDURE STACK=0010H         332  343  362
   11                CLSWAIT. . . . . . . .   LITERALLY '7'
  168   0024H     2  COPYLEN. . . . . . . .   WORD IN PROC (GET)      230*  231  232  233  234  235  236
  106   0000H        CQCHECKALARM . . . . .   PROCEDURE BYTE EXTERNAL(48) STACK=0000H          308
  109   0000H        CQCLEARALARM . . . . .   PROCEDURE EXTERNAL(49) STACK=0000H       148  155
  100   0000H        CQCREATEALARM. . . . .   PROCEDURE EXTERNAL(46) STACK=0000H
   47   0000H        CQCREATELIST . . . . .   PROCEDURE EXTERNAL(28) STACK=0000H
   70   0000H        CQCREATEMAILBOX. . . .   PROCEDURE EXTERNAL(36) STACK=0000H
   55   0000H        CQCREATEPROCESS. . . .   PROCEDURE EXTERNAL(31) STACK=0000H
   58   0000H        CQCREATESEMAPHORE. . .   PROCEDURE EXTERNAL(32) STACK=0000H
   79   0000H        CQCRECEIVE . . . . . .   PROCEDURE POINTER EXTERNAL(39) STACK=0000H
   67   0000H        CQCWAIT. . . . . . . .   PROCEDURE BYTE EXTERNAL(35) STACK=0000H
  142   0000H        CQDLLCONNECT . . . . .   PROCEDURE BYTE EXTERNAL(61) STACK=0000H
  136   0000H        CQDLLREAD. . . . . . .   PROCEDURE WORD EXTERNAL(59) STACK=0000H
  139   0000H        CQDLLREADC . . . . . .   PROCEDURE WORD EXTERNAL(60) STACK=0000H
  131   0000H        CQDLLRXRETBUF. . . . .   PROCEDURE EXTERNAL(57) STACK=0000H
  134   0000H        CQDLLSTART . . . . . .   PROCEDURE EXTERNAL(58) STACK=0000H
  127   0000H     2  CQDLLTXFREEMBX . . . .   WORD EXTERNAL(55)       164  223
  128   0000H        CQDLLTXSEND. . . . . .   PROCEDURE EXTERNAL(56) STACK=0000H        253
```

```
  52  0000H        CQHALTANDCATCHFIRE . .   PROCEDURE EXTERNAL(30) STACK=0000H       199  214  393
  94  0000H        CQICRECEIVE. . . . . .   PROCEDURE POINTER EXTERNAL(44) STACK=0000H
  88  0000H        CQICWAIT . . . . . .     PROCEDURE BYTE EXTERNAL(42) STACK=0000H
  91  0000H        CQISEND. . . . . .       PROCEDURE EXTERNAL(43) STACK=0000H
  85  0000H        CQISIGNAL. . . . . .     PROCEDURE EXTERNAL(41) STACK=0000H
 115  0000H        CQMIPCONNECT . . . . .   PROCEDURE BYTE EXTERNAL(51) STACK=0000H
 121  0000H        CQMIPGETADDRESS. . . .   PROCEDURE POINTER EXTERNAL(53) STACK=0000H     177  218
 124  0000H        CQMIPGETMIPFORM. . . .   PROCEDURE POINTER EXTERNAL(54) STACK=0000H
 118  0000H        CQMIPREGISTER. . . . .   PROCEDURE BYTE EXTERNAL(52) STACK=0000H
 112  0000H        CQMIPSEND. . . . . .     PROCEDURE BYTE EXTERNAL(50) STACK=0000H
  82  0000H        CQMRECEIVE . . . . . .   PROCEDURE POINTER EXTERNAL(40) STACK=0000H
  97  0000H        CQREADCLOCK. . . . . .   PROCEDURE EXTERNAL(45) STACK=0000H       302
  76  0000H        CQRECEIVE. . . . . .     PROCEDURE POINTER EXTERNAL(38) STACK=0000H     164  388
  50  0000H        CQSCHEDULE . . . . . .   PROCEDURE EXTERNAL(29) STACK=0000H       381  478
  73  0000H        CQSEND . . . . . .       PROCEDURE EXTERNAL(37) STACK=0000H       223  471  474
 103  0000H        CQSETALARM . . . . . .   PROCEDURE EXTERNAL(47) STACK=0000H       150  159  459
  61  0000H        CQSIGNAL . . . . . .     PROCEDURE EXTERNAL(33) STACK=0000H       163  252  380  477
  45  0000H        CQSTART. . . . . .       PROCEDURE EXTERNAL(27) STACK=0000H
  64  0000H        CQWAITSEM. . . . . .     PROCEDURE EXTERNAL(34) STACK=0000H       165  254  382  389
   7               CREDITMASK . . . . . .   LITERALLY '003FH'
 147  0004H    1   CSDATYPE . . . . . .     BYTE IN PROC (CLRSETDATAALARM) PARAMETER AUTOMATIC          147  149
   7               CTLBITSMASK. . . . . .   LITERALLY '1F00H'
 268  0006H    2   CTLFIELD . . . . . .     WORD IN PROC (BLDSENDCTLSEG) PARAMETER AUTOMATIC     268  275  276
 307  0034H    2   CTLFIELD . . . . . .     WORD IN PROC (TPERRFLOW)
 348  0004H    2   CTLFIELD . . . . . .     WORD IN PROC (SENDCONNECTSEG) PARAMETER AUTOMATIC          348  354
                   357
 268  0004H    2   CTLSEQNO . . . . . .     WORD IN PROC (BLDSENDCTLSEG) PARAMETER AUTOMATIC     268  272
 168  001AH    2   CUM_RB_BYTES . . . . .   WORD IN PROC (GET)     207*  210  211*  211  216
   8  000EH    2   CURCDBB. . . . . .       WORD AT          416*
   3  0036H    1   CURCDBINDEX. . . . . .   BYTE       294  399*  400  404  405  406  421*  450  463
   8  000CH    2   CURCDBO. . . . . .       WORD AT          418*  419*
   8  000CH    4   CURCDBP. . . . . .       POINTER            8  148  150  181  194  196  197  221  257
                    258  259  261  262  263  264  265  281  282  286  287
                    289  291  292  294  300  308  310  312  315  316  317
                    320  325  329  330  333  337  339  341  352  354  357
                    358  361  405  418  419  420  426  444  450  453  455
                    456  458  459  463
   3  0000H    2   CURCID . . . . . .       WORD       404*  420*  421
  98  0000H    2   DATARTNO . . . . . .     WORD IN PROC (CQREADCLOCK) PARAMETER      98
   2               DEFNETIDLIT. . . . . .   LITERALLY '1'
  10               DEFSTATUSREQ . . . . .   LITERALLY '4'
 104  0000H    2   DELAYH . . . . . .       WORD IN PROC (CQSETALARM) PARAMETER      104
 104  0000H    2   DELAYL . . . . . .       WORD IN PROC (CQSETALARM) PARAMETER      104
  15  0000H        DELETECDB. . . . . .     PROCEDURE EXTERNAL(17) STACK=0000H
   2               DLLHEADERLEN . . . . .   LITERALLY '14'          191  273  366
 420  0769H        DUMMYLABEL . . . . . .   LABEL IN PROC (TRANSPROC)
  56  0000H    2   ENTRYO . . . . . .       WORD IN PROC (CQCREATEPROCESS) PARAMETER        56
   7               EOMACKMASK . . . . . .   LITERALLY '1200H'
   7               EOMFINMASK . . . . . .   LITERALLY '0600H'
   7               EOMMASK. . . . . .       LITERALLY '0200H'       185
  53  0000H    2   ERRORCODE. . . . . .     WORD IN PROC (CQHALTANDCATCHFIRE) PARAMETER          53
  11               ESTAB. . . . . .         LITERALLY '3'
   2               FALSE. . . . . .         LITERALLY '0'           178  195  202  224  226  270  313  345
   7               FINACKMASK . . . . . .   LITERALLY '1400H'
   7               FINMASK. . . . . .       LITERALLY '0400H'       187
  11               FINWAIT1 . . . . . .     LITERALLY '4'
```

```
 11                FINWAIT2 . . . . . .   LITERALLY '5'
  2                FOREVER. . . . . .     LITERALLY 'WHILE true'        229  387
  5  0000H      2  FREEIRBMBX . . . . . .  WORD EXTERNAL(12)       471
  5  0000H      2  FREELIRBMBX. . . . . .  WORD EXTERNAL(13)       474
 11                FROMLISTEN . . . . . .  LITERALLY '80H'
 21  0000H         GEMOD64K . . . . . .    PROCEDURE BYTE EXTERNAL(19) STACK=0000H        315
167  0093H    439  GET. . . . . .          PROCEDURE BYTE STACK=000CH          321
168  0000H     32  GETRBS . . . . . .      STRUCTURE BASED(GETRBSP) IN PROC (GET)
     0000H      1      CONTENTS . . . . .  BYTE        200
     0001H      1      CREDIT . . . . . .  BYTE
     0002H      2      LASTSEQ. . . . . .  WORD        200
     0004H      4      MIPBUFBASE . . . .  POINTER
     0008H      2      MIPLENGTH. . . . .  WORD
     000AH      1      MIPIDSID . . . . .  BYTE
     000BH      1      MIPOWNERDEVID. . .  BYTE
     000CH      2      INTERNALPROCESSID.  WORD
     000EH      1      REQ. . . . . .      BYTE        184  186
     000FH      1      RESP . . . . . .    BYTE
     0010H      2      RTNMIPSKT. . . . .  WORD
     0012H      4      LINK . . . . . .    POINTER          201
     0016H      2      CID. . . . . .      WORD
     0018H      2      FIRSTSEQ . . . . .  WORD        198  205
     001AH      2      CLIENTUSE. . . . .  WORD
     001CH      2      BUFLEN . . . . . .  WORD        182  208
     001EH      1      NUMBLKS. . . . . .  BYTE        171  213
     001FH      1      VB . . . . . .      BYTE        204
168  002AH      4  GETRBSP. . . . . .      POINTER IN PROC (GET)        171  182  184  186  196*  198  200
                                             201*  201  204  205  208  213
168  0000H      6  GETRBV . . . . . .      STRUCTURE BASED(GETRBVP) ARRAY(1) IN PROC (GET)
     0000H      4      BLKPTR . . . . . .  POINTER          177  218
     0004H      2      BLKLEN . . . . . .  WORD       173  175  210  211  217
168  002EH      4  GETRBVP. . . . . .      POINTER IN PROC (GET)        173  175  177  204*  210  211  217
                                             218
162  0071H     34  GETTXBUF . . . . . .    PROCEDURE STACK=0006H        220  269  365
467  082AH         GIVEBACKIRB. . . . . .  LABEL IN PROC (TRANSPROC)          402  408  414  423  428
 18  0000H         GTMOD64K . . . . . .    PROCEDURE BYTE EXTERNAL(18) STACK=0000H        198  200
145                HACFCHKACB . . . . . .  LITERALLY '438'
145                HACFDLLCONN. . . . . .  LITERALLY '401'
145                HACFDLLREADHOST. . . .  LITERALLY '402'
145                HACFIPDEFIRB . . . . .  LITERALLY '432'
145                HACFMIPCONNECT . . . .  LITERALLY '404'
145                HACFMIPREGISTER. . . .  LITERALLY '434'
145                HACFMIPSEND. . . . . .  LITERALLY '416'
145                HACFPUTCNT . . . . . .  LITERALLY '405'
145                HACFRPCOMPLETE . . . .  LITERALLY '413'
145                HACFRPDEFIRB . . . . .  LITERALLY '433'
145                HACFRPPUTDATAP . . . .  LITERALLY '412'
145                HACFRPPUTNEWBLK. . . .  LITERALLY '409'
145                HACFRPPUTPTR . . . . .  LITERALLY '411'
145                HACFRPPUTSENDRBSBACK .  LITERALLY '410'
145                HACFSENDBLKLEN . . . .  LITERALLY '403'
145                HACFTPGETBLKLEN. . . .  LITERALLY '408'
145                HACFTPGETCHK . . . . .  LITERALLY '407'        214
145                HACFTPIRBACB . . . . .  LITERALLY '415'
145                HACFTPIRBP . . . . . .  LITERALLY '414'
145                HACFTPIRBTYPE. . . . .  LITERALLY '406'        393
```

```
  145                  HACFTPTIMEOUT. . . . .   LITERALLY '435'
  145                  HACFTQSEQ. . . . . .     LITERALLY '417'          199
  145                  HACFXQ . . . . . .       LITERALLY '436'
  145                  HACFZQ . . . . . .       LITERALLY '437'
                       HIGH . . . . . .         BUILTIN            150  159  292  459
   10                  ILLEGALREQ . . . . . .   LITERALLY '10'
  285  0415H     92    INCRRETRANCOUNTS . . .   PROCEDURE STACK=000CH          311  351
   13  0000H      1    INDEX. . . . . .         BYTE IN PROC (SETUPCDB) PARAMETER          13
   59  0000H      2    INIT . . . . . .         WORD IN PROC (CQCREATESEMAPHORE) PARAMETER        59
   10                  INVALIDPOINTER . . . .   LITERALLY '20'
   10                  INVALIDREQ . . . . . .   LITERALLY '2'
  307  0038H      1    INWINDOW . . . . . .     BYTE IN PROC (TPERRFLOW)       315*  317
    5  0000H      2    IPINMBX. . . . . .       WORD EXTERNAL(11)
    8  0000H      8    IRB. . . . . .           STRUCTURE BASED(IRBP)
       0000H      4        CMXPTR . . . . . .   POINTER
       0004H      1        TYPE . . . . . .     BYTE          390
       0005H      1        CDBINDEX . . . . .   BYTE          399
       0006H      2        CID. . . . . .       WORD          406
    9                  IRBAYTTIMER. . . . . .   LITERALLY '7'            158
    8  0012H      2    IRBB . . . . . .         WORD AT             416
    9                  IRBCTLACBMASK. . . . .   LITERALLY '4CH'          391  417
    9                  IRBCTLTIMEOUTMASK. . .   LITERALLY '0C0H'         158  391
    9                  IRBINVALIDMASK . . . .   LITERALLY '38H'          391
    9                  IRBMAXCODE . . . . . .   LITERALLY '7'            391
    9                  IRBNULL. . . . . .       LITERALLY '0FFH'
    8  0010H      2    IRBO . . . . . .         WORD AT             418  419
    8  0010H      4    IRBP . . . . . .         POINTER               8  367  368  369  371  374  388*  390  399
                                                 406  412  471  474
    9                  IRBSENDCHECK . . . . .   LITERALLY '4'            332  343
    9                  IRBSENDFIN . . . . . .   LITERALLY '2'
    9                  IRBSENDFLAG. . . . . .   LITERALLY '5'
    9                  IRBSENDRST . . . . . .   LITERALLY '3'            395  469
    9                  IRBSENDSYN . . . . . .   LITERALLY '0'
    9                  IRBSENDSYNACK. . . . .   LITERALLY '1'
    9                  IRBTIMEOUTMASK . . . .   LITERALLY '80H'          332  343  349  362  397  467
    9                  IRBTIMEWAITTO. . . . .   LITERALLY '6'
    9                  IRBTYPEMASK. . . . . .   LITERALLY '0FH'          362  391  431
    3  0037H      1    IRB_TYPE . . . . . .     BYTE          349  362  390*  391  395  397  417  431  467  469
    3  0002H      2    J. . . . . .             WORD          378*  379*  379
    3  0000H           LCIDVECTOR . . . . . .   WORD ARRAY(0) EXTERNAL(0)          400  404  406
    8  0000H     24    LIRB . . . . . .         STRUCTURE BASED(IRBP)
       0000H      4        CMXPTR . . . . . .   POINTER
       0004H      1        TYPE . . . . . .     BYTE
       0005H      1        REASON . . . . . .   BYTE          374
       0006H      6        DLDEST . . . . . .   WORD ARRAY(3)            367  368  369
       000CH      2        DESTPORT . . . . .   WORD          371
       000EH      2        SOURCEPORT . . . .   WORD
       0010H      2        DESTCID. . . . . .   WORD
       0012H      2        SOURCECID. . . . .   WORD
       0014H      2        SEGSEQNO . . . . .   WORD
       0016H      2        SEGACKNO . . . . .   WORD
   11                  LISTEN . . . . . .       LITERALLY '0'
   48  0000H      4    LISTP. . . . . .         POINTER IN PROC (CQCREATELIST) PARAMETER        48
   10                  LOCABORT . . . . . .     LITERALLY '12'
    3  0000H      6    LOCHOST. . . . . .       WORD ARRAY(3) EXTERNAL(2)
    3  0000H      2    LOCNET . . . . . .       WORD EXTERNAL(1)
```

```
 10                    LOCTIMEOUT . . . . . LITERALLY '16'          294  463
  2                    LOGRBMIPPORT . . . . LITERALLY '5'
                       LOW. . . . . . . . . BUILTIN            150  159  459
 37  0000H      2      M. . . . . . . . . . WORD IN PROC (MAX) PARAMETER          37
 28  0000H      2      M. . . . . . . . . . WORD IN PROC (MIN) PARAMETER          28
 25  0000H      2      M. . . . . . . . . . WORD IN PROC (MAXMOD64K) PARAMETER          25
 22  0000H      2      M. . . . . . . . . . WORD IN PROC (GEMOD64K) PARAMETER           22
 19  0000H      2      M. . . . . . . . . . WORD IN PROC (GTMOD64K) PARAMETER           19
 80  0000H      2      MAILBOX0 . . . . . . WORD IN PROC (CQCRECEIVE) PARAMETER         80
 77  0000H      2      MAILBOX0 . . . . . . WORD IN PROC (CQRECEIVE) PARAMETER          77
 74  0000H      2      MAILBOX0 . . . . . . WORD IN PROC (CQSEND) PARAMETER             74
 71  0000H      2      MAILBOX0 . . . . . . WORD IN PROC (CQCREATEMAILBOX) PARAMETER          71
104  0000H      2      MAILBOX0 . . . . . . WORD IN PROC (CQSETALARM) PARAMETER        104
 95  0000H      2      MAILBOX0 . . . . . . WORD IN PROC (CQICRECEIVE) PARAMETER        95
 92  0000H      2      MAILBOX0 . . . . . . WORD IN PROC (CQISEND) PARAMETER            92
 36  0000H              MAX. . . . . . . . . PROCEDURE WORD EXTERNAL(24) STACK=0000H          191
 24  0000H              MAXMOD64K. . . . . . PROCEDURE WORD EXTERNAL(20) STACK=0000H          282  330
  3  0000H      2      MAXSEGDATALEN. . . . WORD EXTERNAL(3)        205  227
  2                    MAXSEGDATALENLIT . . LITERALLY '1480'
  2                    MAXSENDSEG . . . . . LITERALLY '07H'         379
  3  0000H      1      MAXWINDOWSIZE. . . . BYTE EXTERNAL(4)        181  281
143  0000H      2      MBX0 . . . . . . . . WORD IN PROC (CQDLLCONNECT) PARAMETER       143
116  0000H      2      MBX0 . . . . . . . . WORD IN PROC (CQMIPCONNECT) PARAMETER       116
 74  0000H      4      MESSAGEP . . . . . . POINTER IN PROC (CQSEND) PARAMETER          74
 92  0000H      4      MESSAGEP . . . . . . POINTER IN PROC (CQISEND) PARAMETER         92
 27  0000H              MIN. . . . . . . . . PROCEDURE WORD EXTERNAL(21) STACK=0000H          181  230  281
  2                    MINPKTLEN. . . . . . LITERALLY '46'          191  273  366
  2                    MIPECHOPORT. . . . . LITERALLY '7'
122  0000H      4      MIP_FORM . . . . . . POINTER IN PROC (CQMIPGETADDRESS) PARAMETER       122
140  0000H      2      MODIFIER . . . . . . WORD IN PROC (CQDLLREADC) PARAMETER     140
137  0000H      2      MODIFIER . . . . . . WORD IN PROC (CQDLLREAD) PARAMETER      137
                       MOVW . . . . . . . . BUILTIN              231  371
113  0000H      4      MSGP . . . . . . . . POINTER IN PROC (CQMIPSEND) PARAMETER          113
 37  0000H      2      N. . . . . . . . . . WORD IN PROC (MAX) PARAMETER          37
 28  0000H      2      N. . . . . . . . . . WORD IN PROC (MIN) PARAMETER          28
 25  0000H      2      N. . . . . . . . . . WORD IN PROC (MAXMOD64K) PARAMETER          25
 22  0000H      2      N. . . . . . . . . . WORD IN PROC (GEMOD64K) PARAMETER           22
 19  0000H      2      N. . . . . . . . . . WORD IN PROC (GTMOD64K) PARAMETER           19
170  024DH              NMBTOP . . . . . . . LABEL IN PROC (NOMOREBLKS)          174
169  024AH     97      NOMOREBLKS . . . . . PROCEDURE BYTE IN PROC (GET) STACK=0008H          242
 10                    NORESOURCESRESP. . . LITERALLY '4'
140  0000H      2      OBJECT . . . . . . . WORD IN PROC (CQDLLREADC) PARAMETER     140
137  0000H      2      OBJECT . . . . . . . WORD IN PROC (CQDLLREAD) PARAMETER      137
 10                    OKCLOSEDRESP . . . . LITERALLY '9'                450
 10                    OKEOMRESP. . . . . . LITERALLY '3'
 10                    OKFINRESP. . . . . . LITERALLY '5'
 10                    OKRESP . . . . . . . LITERALLY '1'
  2                    ONSDTCLECHOPORT. . . LITERALLY '7'
 10                    OPENAREQ . . . . . . LITERALLY '0'
 10                    OPENCONFLICT . . . . LITERALLY '18'
 10                    OPENPREQ . . . . . . LITERALLY '1'
 56  0000H      2      PCB0 . . . . . . . . WORD IN PROC (CQCREATEPROCESS) PARAMETER          56
129  0000H      2      PKT0 . . . . . . . . WORD IN PROC (CQDLLTXSEND) PARAMETER    129
116  0000H      1      PORTID . . . . . . . BYTE IN PROC (CQMIPCONNECT) PARAMETER          116
 10                    POSTRBUFREQ. . . . . LITERALLY '7'
 56  0000H      2      PRI. . . . . . . . . WORD IN PROC (CQCREATEPROCESS) PARAMETER          56
```

```
 119  0000H     2  PROCEDUREO . . . . . .  WORD IN PROC (CQMIPREGISTER) PARAMETER  119
 125  0000H     4  PTR. . . . . .  POINTER IN PROC (CQMIPGETMIPFORM) PARAMETER  125
  10             REMABORT . . . . . .  LITERALLY '14'
  10             REQMAX . . . . . .  LITERALLY '8'
  31  0000H     1  RESPCODE . . . . . .  BYTE IN PROC (TRYTODELETECDB) PARAMETER  31
 140  0000H     4  RETURNBUFP . . . . . .  POINTER IN PROC (CQDLLREADC) PARAMETER  140
 137  0000H     4  RETURNBUFP . . . . . .  POINTER IN PROC (CQDLLREAD) PARAMETER  137
   5  0000H     2  RPMBX. . . . . .  WORD EXTERNAL(10)
   7             RSTACKMASK . . . . . .  LITERALLY '1100H'  373
   7             RSTMASK. . . . . .  LITERALLY '0100H'
  16  0000H     1  RTNCODE. . . . . .  BYTE IN PROC (DELETECDB) PARAMETER  16
   5  0000H     2  SCHEDLOCK. . . . . .  WORD EXTERNAL(15)  163  165  252  254  380  382  389  477
  33  0000H       SEARCH_LCIDVECTOR. . .  PROCEDURE WORD EXTERNAL(23) STACK=0000H  421
  40  0000H     2  SEGO . . . . . .  WORD IN PROC (CHKSUMCALC) PARAMETER  40
 168  0020H     2  SEGOFFSET. . . . . .  WORD IN PROC (GET)  190  191  228*  231  232*  232
 168  0022H     2  SEGREMBYTES. . . . . .  WORD IN PROC (GET)  227*  230  234*  234  237
 299  0004H     2  SEGSEQNO_TO_TIME . . .  WORD IN PROC (SEGTRANTIMESTAMP) PARAMETER AUTOMATIC  299  304
 298  0471H    51  SEGTRANTIMESTAMP . . .  PROCEDURE STACK=0008H  329  361
  68  0000H     2  SEMAPHOREO . . . . . .  WORD IN PROC (CQCWAIT) PARAMETER  68
  65  0000H     2  SEMAPHOREO . . . . . .  WORD IN PROC (CQWAITSEM) PARAMETER  65
  62  0000H     2  SEMAPHOREO . . . . . .  WORD IN PROC (CQSIGNAL) PARAMETER  62
  59  0000H     2  SEMAPHOREO . . . . . .  WORD IN PROC (CQCREATESEMAPHORE) PARAMETER  59
  89  0000H     2  SEMAPHOREO . . . . . .  WORD IN PROC (CQICWAIT) PARAMETER  89
  86  0000H     2  SEMAPHOREO . . . . . .  WORD IN PROC (CQISIGNAL) PARAMETER  86
 347  05ADH    93  SENDCONNECTSEG . . . .  PROCEDURE STACK=0016H  433  436
 307  003AH     1  SENDDATANOW. . . . . .  BYTE IN PROC (TPERRFLOW)  317*  318
  10             SENDEOMREQ . . . . . .  LITERALLY '6'  184
 377  0668H    55  SENDLOOP . . . . . .  PROCEDURE STACK=0018H  441  446
 249  0321H    49  SENDPACKET . . . . . .  PROCEDURE STACK=0006H  283  324  375
  10             SENDREQ. . . . . .  LITERALLY '5'
 364  060AH    94  SENDRSTFROMLIRB. . . .  PROCEDURE STACK=000AH  439
  12  0000H       SETUPCDB . . . . . .  PROCEDURE EXTERNAL(16) STACK=0000H  405
 180  02ABH   118  SETUPSEGCTLFIELDS. . .  PROCEDURE IN PROC (GET) STACK=0008H  239  244
 256  0352H    74  SETUPSEGHEADER . . . .  PROCEDURE STACK=0002H  271  323
 307  0039H     1  SHOWNEWDATA. . . . . .  BYTE IN PROC (TPERRFLOW)  316*  317  331
                  SHR. . . . . .  BUILTIN  231
 113  0000H     2  SOCKET . . . . . .  WORD IN PROC (CQMIPSEND) PARAMETER  113
   6  0000H    41  SP . . . . . .  STRUCTURE BASED(SPO)
      0000H     4      KAOSMSGHDR . . . .  POINTER
      0004H     2      BUFLEN . . . . . .  WORD  191*  273*  366*
      0006H     6      DLDEST . . . . . .  WORD ARRAY(3)  257*  258*  259*  367*  368*  369*
      000CH     6      DLSOURCE . . . . . .  WORD ARRAY(3)
      0012H     2      DLTYPE . . . . . .  WORD  250*
      0014H     2      TCLVERSION . . . .  WORD  260*  370*
      0016H     2      DESTPORT . . . . .  WORD  261*  371
      0018H     2      SOURCEPORT . . . .  WORD  262*
      001AH     2      DESTCID. . . . . .  WORD  263*  279*
      001CH     2      SOURCECID. . . . .  WORD  264*
      001EH     2      SEGSEQNO . . . . .  WORD  189*  272*  282
      0020H     2      SEGACKNO . . . . .  WORD  265*  278*
      0022H     2      SEGDATALEN . . . .  WORD  190*  274*  372*
      0024H     2      CTL. . . . . .  WORD  181*  185*  185  187*  187  275*  281*  281  373*
      0026H     2      CHECKSUM . . . . .  WORD  251*
      0028H     1      SEGDATA. . . . . .  BYTE ARRAY(1)  231  374*
   6  0000H    20  SPARRAY. . . . . .  BYTE BASED(SPP) ARRAY(20)
   6  0008H     2  SPO. . . . . .  WORD AT  185  187  231  251  253  281  282  371
```

```
     6  0008H      4  SPP. . . . . .    POINTER            6  164*  223
    56  0000H      2  STACKO . . . . .  WORD IN PROC (CQCREATEPROCESS) PARAMETER      56
    11             .  STATEMASK. . . .  LITERALLY 'OFH'
    10             .  STATUSREQ. . . .  LITERALLY '3'
    42  0000H      .  STKY_INCR. . . .  PROCEDURE EXTERNAL(26) STACK=0000H      286  287  288
     7             .  SYNACKMASK . . .  LITERALLY '1800H'      436
     7             .  SYNMASK. . . . .  LITERALLY '0800H'      276  433
    11             .  SYNRCVD. . . . .  LITERALLY '2'
    11             .  SYNSENT. . . . .  LITERALLY '1'
    34  0000H      2  TARGET . . . . .  WORD IN PROC (SEARCH_LCIDVECTOR) PARAMETER      34
   168  0016H      2  TARGETOFFSET . . .  WORD IN PROC (GET)      182  205*  210  216  236*  236
   168  0014H      2  TARGETSEG. . . . .  WORD IN PROC (GET)      189  197*  198  200  205  221
     2             .  TCLHEADERLEN . . .  LITERALLY '20'          6  191
     2             .  TCLMIPPORT . . . .  LITERALLY '4'
     2             .  TCLPROTOCOLCODE. .  LITERALLY '5001H'
     2             .  TCLPROTOCOLCODEREV  LITERALLY '0150H'      250
     2             .  TCLVERSIONLIT. . .  LITERALLY '101H'       260  370
   307  0032H      2  TEMPNEXTTRANSMIT .  WORD IN PROC (TPERRFLOW)     320*  325  327
     2             .  TIMEOUTINCREASESTATE  LITERALLY '1'
     2             .  TIMEOUTSTEADYSTATE .  LITERALLY '0'
     4             .  TIMERCLEARED . . .  LITERALLY '2'
     4             .  TIMEREXPIRED . . .  LITERALLY '1'
     4             .  TIMERRUNNING . . .  LITERALLY '0'          341
    11             .  TIMEWAIT . . . . .  LITERALLY '6'
     3  0000H      2  TOTPKTSREJ . . . .  WORD EXTERNAL(6)
     3  0000H      2  TOTPKTSRETRAN. . .  WORD EXTERNAL(5)       288
        0000H      .  TP . . . . . .     PROCEDURE STACK=0000H
   306  04A4H    265  TPERRFLOW. . . . .  PROCEDURE BYTE STACK=0014H          379
     5  0000H      2  TPMBX. . . . . .  WORD EXTERNAL(9)       150  159  388  459
     3  0004H      4  TPTIMESTAMP. . . .  DWORD      302  303
   386  069FH    437  TRANSPROC. . . . .  PROCEDURE PUBLIC STACK=001CH
     2             .  TRUE . . . . . .  LITERALLY 'OFFH'       172  229  240  245  334  387
    30  0000H      .  TRYTODELETECDB . .  PROCEDURE EXTERNAL(22) STACK=0000H      294  450  463
   143  0000H      2  TYPE . . . . . .  WORD IN PROC (CQDLLCONNECT) PARAMETER      143
    10             .  UNKNOWNCIDRESP . .  LITERALLY '6'
    83  0000H      2  WCBO . . . . . .  WORD IN PROC (CQMRECEIVE) PARAMETER      83
    43  0000H      4  WDP. . . . . .    POINTER IN PROC (STKY_INCR) PARAMETER      43
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 0854H   2132D
    CONSTANT AREA SIZE = 0000H      0D
    VARIABLE AREA SIZE = 003BH     59D
    MAXIMUM STACK SIZE = 001CH     28D
    1653 LINES READ
    1 PROGRAM WARNING
    0 PROGRAM ERRORS
```

END OF PL/M-86 COMPILATION