

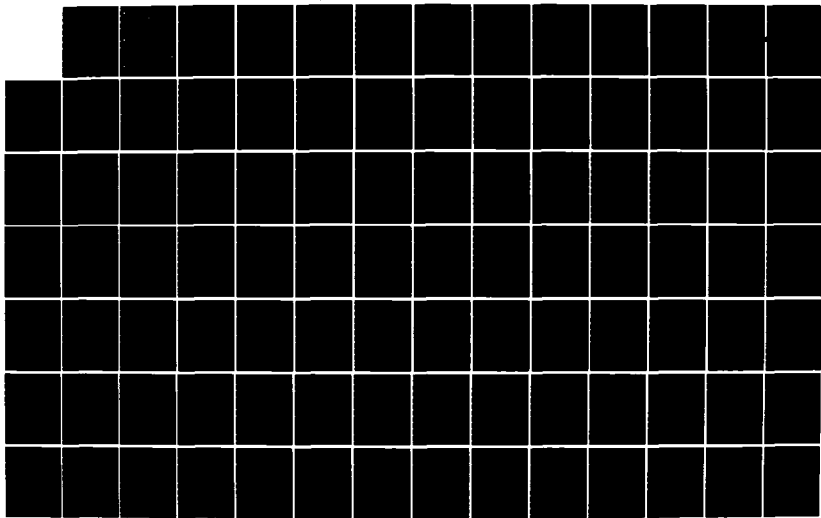
AD-A151 947

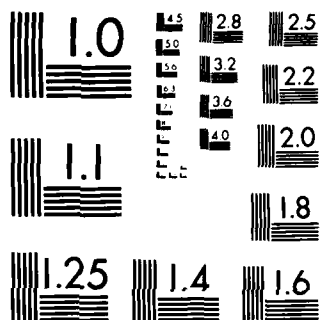
CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE
UNIVERSAL NETWORK INTERFA. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C T CHILDRESS
DEC 84 AFIT/GE/ENG/84D-17-VOL-2 F/G 17/2

1/2

UNCLASSIFIED

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

DTIC
①

AD-A151 947



CONTINUED DEVELOPMENT AND IMPLEMENTATION
OF THE
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II
IN THE
DIGITAL ENGINEERING LABORATORY NETWORK (DELNET)
VOL II OF II

THESIS

AFIT/SE/ENG/84D-17

Creed T. Childress, Jr.
Captain USAF

Approved for distribution and sale in
unlimited quantities.

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DTIC
ELECTE
APR 02 1985
S D E

DTIC FILE COPY

85 03 13 167

1

CONTINUED DEVELOPMENT AND IMPLEMENTATION
OF THE
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II
IN THE
DIGITAL ENGINEERING LABORATORY NETWORK (DELNET)
VOL II OF II

THESIS

AFIT/GE/ENG/84D-17

Creed T. Childress, Jr.
Captain USAF

CONTINUED DEVELOPMENT AND IMPLEMENTATION
OF THE
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II
IN THE
DIGITAL ENGINEERING LABORATORY NETWORK (DELNET)
VOL II OF II

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Creed T. Childress, Jr., BS EE, BS IOE
Captain, USAF

December 1984

[illegible]

Appendix H

Software Listings

This appendix contains the compiled listings of the programs used in this development effort. The list begins with the data link and network simulations followed by the validation and test software used with the UNID II and the NETOS. The simulated host software for the CP/M system follows next with the listing of the SBC 544 monitor concluding the appendix.

This appendix is subdivided into six sections which are listed as follows:

<u>Listing</u>	<u>Page</u>
1. Data Link Layer Simulation	H-2
2. Network Layer Simulation	H-38
3. SBC 544 Validation	H-73
4. Host CP/M	H-114
a. PL/M80 Listing	H-114
b. Assembly Language Listing	H-134
5. SBC 544 Monitor	H-147

1. Data Link Layer Simulation

The purpose of this program is to simulate the data link layer software on the Intel Series III software development system.

ISIS-II PL/M-86 V2.1 COMPILATION OF MODULE MAIN
 OBJECT MODULE PLACED IN NETNEW.OBJ
 COMPILER INVOKED BY: PLM86 NETNEW.U2

\$TITLE('UNID II NETWORK TEST PROGRAM, 30 SEP 84')
 \$XREF OPTIMIZE(2)

```

/*****
/*
/* DATE:      30 Sep 84
/* VERSION:   1.0
/*
/* TITLE:     UNID II Network Simulation Program
/* FILE NAME: NETNEW.U2
/* OWNER:     C. T. Childress
/*
/* SOFTWARE SYSTEM: Intel Series III, ISIS II (V 4.2, 4.3)
/* USE:       Simulate the function of the data link layer
/*            on the Intel Series III software development
/*            system.
/*
/* CONTENTS:  Listing for the data link layer simulation.
/*
*****/

```

```

/*****
PROLOGUE - MODULE N.MAIN$U2      DATE TRANSLATED:  5 APR 84
                                DATE LAST MODIFIED: 30 SEP 84, 1400 HRS

```

THE PURPOSE OF THIS MODULE IS TO PROVIDE THE UNID II NETWORK OPERATING SYSTEM (N.OS) WITH THE MAIN LINE OF PROCESSING. THE N.OS IS REQUIRED TO INPUT AND OUTPUT DATA PASSED TO IT FROM FOUR LOCAL CHANNELS OR RECEIVED FROM EITHER NETWORK CHANNEL A OR B.

THIS MODULE CONSISTS OF THE MAIN LINE PROCEDURE 'MAIN', AND PROCEDURES INIT\$N\$TAB, DET\$DEST\$ONE, DET\$DEST\$TWO, DET\$DEST\$LN, LD\$TAB\$H\$SKP, SRVC\$TAB\$H\$SKP, BUILD\$1\$FRAME, BUILD\$2\$FRAME, ROUTE\$IN, ROUTE\$OUT, TIME\$DELAY\$CHA, AND TIME\$DELAY\$CHB.

*****/

1 MAIN: DO;

/***** EXTERNAL PROCEDURES FOR ISIS SYSTEM CALLS *****/

```

2    1    DQ$DECODE$EXCEPTION:    PROCEDURE (ERRNUM, EXCEPTION$P, STATUS) EXTERNAL;
3    2                            DECLARE ERRNUM WORD;
4    2                            DECLARE (EXCEPTION$P, STATUS) POINTER;
5    2                            END DQ$DECODE$EXCEPTION;

6    1    DQ$CLOSE:                PROCEDURE (AFTN, STATUS) EXTERNAL;
7    2                            DECLARE AFTN WORD, STATUS POINTER;

```

```

8   2           END DQ$CLOSE;
9   1   DQ$DETACH: PROCEDURE (CONNECTION, EXCEPT$P) EXTERNAL;
10  2           DECLARE CONNECTION WORD, EXCEPT$P POINTER;
11  2           END DQ$DETACH;

12  1   DQ$EXIT:  PROCEDURE (COMPLETION$CODE) EXTERNAL;
13  2           DECLARE COMPLETION$CODE WORD;
14  2           END DQ$EXIT;

15  1   DQ$ATTACH: PROCEDURE (PATH$P, STATUS) WORD EXTERNAL;
16  2           DECLARE (PATH$P, STATUS) POINTER;
17  2           END DQ$ATTACH;

18  1   DQ$CREATE: PROCEDURE (PATH$P, STATUS) WORD EXTERNAL;
19  2           DECLARE (PATH$P, STATUS) POINTER;
20  2           END DQ$CREATE;

21  1   DQ$OPEN:  PROCEDURE (AFTN, MODE, NUM$BUF, STATUS) EXTERNAL;
22  2           DECLARE AFTN WORD, STATUS POINTER;
23  2           DECLARE (MODE, NUM$BUF) BYTE;
24  2           END DQ$OPEN;

25  1   DQ$READ:  PROCEDURE (AFTN, BUFFER, COUNT, STATUS) ADDRESS EXTERNAL;
26  2           DECLARE (AFTN, COUNT) WORD;
27  2           DECLARE (BUFFER, STATUS) POINTER;
28  2           END DQ$READ;

29  1   DQ$WRITE: PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL;
30  2           DECLARE (AFTN, COUNT) WORD;
31  2           DECLARE (BUFFER, STATUS) POINTER;
32  2           END DQ$WRITE;

33  1   DECLARE (ACTUAL, STATUS) ADDRESS;
34  1   DECLARE STATUS$PTR (81) BYTE;
35  1   DECLARE BUFFER (128) BYTE;
36  1   DECLARE (K$CONN, W$CONN) ADDRESS PUBLIC;
37  1   DECLARE CR          LITERALLY 'ODH',
           LF              LITERALLY 'OAH';
38  1   DECLARE CR$LF (2) BYTE DATA (ODH, OAH);
39  1   DECLARE MESSAGE(*) BYTE DATA(ODH, OAH,
           'THIS IS THE TEST MESSAGE'          THIS IS THE TEST
           MESSAGE!!!!!!');

/***** END EXTERNALS *****/

40  1   DECLARE
           FALSE          LITERALLY 'OH', /* USE AS FLAGS TO TEST */
           TRUE           LITERALLY 'OIFH', /* BITS FOR BRANCHING */
           CON$TC         LITERALLY 'OD5H', /* NETWORK MONITOR CTC PORT ADDRESS */
           CON$CMD        LITERALLY 'OJFH',
                           /* NETWORK MONITOR USART COMMAND PORT ADDRESS */

```

```

CONDAT      LITERALLY 'ODEH',
            /* NETWORK MONITOR USART DATA PORT ADDRESS */
NET$RI$DEST$ERR LITERALLY '10', /* NET ROUTE$IN DEST ERROR ENTRY */
NET$RO$DEST$ERR LITERALLY '11', /* NET ROUTE$OUT DEST ERROR ENTRY */
PACKET$SIZE    LITERALLY '133', /* PACKET IS 133 BYTE BLOCK */
PACKET$IN$TABLE LITERALLY '10', /* # PACKETS IN TABLE */
PACKET$TABLE$SIZE LITERALLY '1330',
FRAME$SIZE     LITERALLY '135',
FRAME$IN$TABLE LITERALLY '10',
FRAME$TABLE$SIZE LITERALLY '1350',
DATA$SIZE      LITERALLY '128',
IP$DATA$SIZE   LITERALLY '96',
TCP$DATA$SIZE  LITERALLY '72';

```

/* NETWORK VARIABLES FOR THIS UNID */

/* NOTES: 1. THIS\$UNID\$NBR MUST REFLECT WHICH UNID THIS IS.
 2. THIS\$COUNTRY\$CODE MUST REFLECT THE AREA TO WHICH THIS UNID IS LOCATED.
 3. MAX\$COUNTRY\$CODE WILL INDICATE WHICH COUNTRY CODES ARE CURRENTLY OPERATIONAL. CC = 0000 IS RESERVED FOR THE DELNET MONITOR.
 4. MAX\$NETWORK\$CODE WILL INDICATE HOW MANY UNIDS ARE CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
 5. FOR DETAILED INFORMATION ON THE ABOVE, REFER TO PHISTER'S THESIS, APPENDIX D. */

41 1 DECLARE

```

THIS$UNID$NBR LITERALLY '02H', /* UNIQUE ADDRESS OF THIS UNID */
THIS$COUNTRY$CODE LITERALLY '01H', /* COUNTRY WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '01H', /* COUNTRIES CURRENTLY OPERATIONAL */
MAX$NETWORK$CODE LITERALLY '03H',
            /* NUMBER OF UNIDS OPERATIONAL IN CC */
STAT$NBR LITERALLY '20', /* NUMBER OF ENTRIES IN STATUS TABLE */
/* VARIABLES USED IN THIS SIMULATION */
CTCNOA BYTE, /* PROGRESSIVE NUMBER OF TIME COUNTS FOR
            NETWORK CHANNEL A */
CTCNOB BYTE, /* PROGRESSIVE NUMBER OF TIME COUNTS FOR
            NETWORK CHANNEL B */
MAXNOA LITERALLY '3', /* PREVIOUSLY 64H = 100D */
            /* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL A */
MAXNOB LITERALLY '3', /* PREVIOUSLY 64H = 100D */
            /* MAXIMUM NUMBER OF TIMING COUNTS FOR NETWORK CHANNEL B */
RETRANS$A BYTE,
            /* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
RETRANS$B BYTE,
            /* PROGRESSIVE NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRANS$A LITERALLY '6',
            /* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */
MAXRETRANS$B LITERALLY '6';

```

/* MAXIMUM NUMBER OF RETRANSMISSIONS OF A FRAME */

```

42 1  DECLARE BUSYSTATUS  LITERALLY 'OFFH',
        NMBR$MSK  LITERALLY '07H',
        ESC  LITERALLY '1BH',
        EOF  LITERALLY '04H';
43 1  DECLARE ASCII(*)  BYTE DATA('0123456789ABCDEF');
44 1  DECLARE MSGNUM  BYTE;
45 1  DECLARE OUT$TAB$FULL  BYTE;

```

```

/*****
/*  DEFINITIONS FOR THE NETWORK SERIAL INPUT/OUTPUT USARTS  */
*****/

```

```

46 1  DECLARE
        BAUD$LSB  BYTE,
        BAUD$MSB  BYTE,
        NUM$BYTES$SENT  INTEGER;

```

/* MORE TO BE ADDED LATER */

```

/*****
/*  ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM  */
*****/

```

```

47 1  DECLARE  TIMCHA  BYTE,
        TIMCHB  BYTE,
        HSKP$ERR  INTEGER;
        /* MORE TO BE ADDED LATER */

```

```

/*****
/*  DATA TABLES USED IN THIS PROGRAM  */
*****/

```

```

48 1  DECLARE
        NT01RX(FRAME$TABLE$SIZE)  BYTE,
        NT01NS  INTEGER,
        NT01NE  INTEGER,
        NT01SZ  INTEGER,

        NT02RX(FRAME$TABLE$SIZE)  BYTE,
        NT02NS  INTEGER,
        NT02NE  INTEGER,
        NT02SZ  INTEGER,

        NT01TX(FRAME$TABLE$SIZE)  BYTE,
        TX01NS  INTEGER,
        TX01NE  INTEGER,
        TX01SZ  INTEGER,

```

```

NT02TX(FRAME$TABLE$SIZE) BYTE,
TX02NS INTEGER,
TX02NE INTEGER,
TX02SZ INTEGER,

LCNTTB(PACKET$TABLE$SIZE) BYTE,
LCNTNS INTEGER,
LCNTNE INTEGER,
LCNTSZ INTEGER,

NTLCTB(PACKET$TABLE$SIZE) BYTE,
NTLCNS INTEGER,
NTLCNE INTEGER,
NTLCNZ INTEGER,

STATTB(STAT$NBR) BYTE;

```

```

49 1 DECLARE FRAME STRUCTURE(ADDRES BYTE,
                             CONTROL BYTE,
                             PACKET$DEST BYTE,
                             PACKET$SOURCE BYTE,
                             PACKET$SEQ BYTE,
                             PACKET$SPAR1 BYTE,
                             PACKET$SPAR2 BYTE,
                             PACKET$DATA(DATA$SIZE) BYTE);

```

```

50 1 DECLARE PACKET STRUCTURE(DESTIN BYTE,
                              SOURCE BYTE,
                              SEQUEN BYTE,
                              SPARE1 BYTE,
                              SPARE2 BYTE,
                              DATAM(DATA$SIZE) BYTE);

```

```

51 1 DECLARE IPDATA STRUCTURE(VERS$IHL BYTE,
                              TYPE$SERV BYTE,
                              TOT$LEN1 BYTE,
                              TCT$LEN2 BYTE,
                              ID$1 BYTE,
                              ID$2 BYTE,
                              FLAG$FRAG BYTE,
                              FRAGOFF BYTE,
                              TIME$LIVE BYTE,
                              PROTO BYTE,
                              CHKSUM$1 BYTE,
                              CHKSUM$2 BYTE,
                              SORC$ADDR(4) BYTE,
                              DEST$ADDR(4) BYTE,
                              OPTIONS(3) BYTE,
                              PADDING BYTE,
                              DATAM(IP$DATA$SIZE) BYTE);

```

```

52  1  DECLARE      TOPDATA STRUCTURE(SOURCE$PORT(2)  BYTE,
                                DESTIN$PORT(2)        BYTE,
                                SEQUEN$NUM(4)         BYTE,
                                ACK$NUM(4)            BYTE,
                                DATA$OFF$PLUS        BYTE,
                                RESERV$PLUS           BYTE,
                                WINDOW(2)             BYTE,
                                CHECKSUM(2)           BYTE,
                                URGENT$PTR(2)          BYTE,
                                OPTIONS(3)            BYTE,
                                PADDING                BYTE,
                                DATAM(TOP$DATA$SIZE)  BYTE);

/* MISCELLANEOUS DECLARATIONS */

53  1  DECLARE
      FOREVER BYTE,
      DESTINATION WORD,      /* DESTINATION OF A FRAME */
      SEQ$BIT$A  BYTE,      /* ACKNOWLEDGE VARIABLE DECLARATIONS */
      SEQ$BIT$B  BYTE,
      INPUT$SEQ$BIT$A  BYTE,
      INPUT$SEQ$BIT$B  BYTE,
      THIS$SEQ$BIT$A  BYTE,
      THIS$SEQ$BIT$B  BYTE;

54  1  DECLARE      STARTUP$HDR(*) BYTE DATA(CR,LF,
      '              UNID 11 #3 NETWORK OS',CR,LF,
      '              VERS 1.0, 30 SEP 1984',CR,LF,
      '              EXECUTING',CR,LF);

55  1  DECLARE      TP$1(*) BYTE DATA(CR,LF,
      'TP$1          ENTERING INIT$N$TAB PROCEDURE');
56  1  DECLARE      TP$2(*) BYTE DATA(CR,LF,
      'TP$2          ENTERING INSIO PROCEDURE');
57  1  DECLARE      TP$3(*) BYTE DATA(CR,LF,
      'TP$3          STARTING ROUTE$IN - ROUTE$OUT LOOP');
58  1  DECLARE      TP$4(*) BYTE DATA(CR,LF,
      'TP$4          ENTERING ROUTE$IN PROCEDURE');
59  1  DECLARE      TP$5(*) BYTE DATA(CR,LF,
      'TP$5          INCOMING FRAME LOCATED IN NETWORK CHANNEL A');
60  1  DECLARE      TP$6(*) BYTE DATA(CR,LF,
      'TP$6          FRAME IS NT01RX TO NT02TX TRANSFER');
61  1  DECLARE      TP$7(*) BYTE DATA(CR,LF,
      'TP$7          DATA IS DESTINED FOR THIS UNID');
62  1  DECLARE      TP$8(*) BYTE DATA(CR,LF,
      'TP$8          FRAME IS INCOMING S FRAME');
63  1  DECLARE      TP$9(*) BYTE DATA(CR,LF,
      'TP$9          THIS SEQ$BIT$A = 1');
64  1  DECLARE      TP$10(*) BYTE DATA(CR,LF,
      'TP$10         THIS SEQ$BIT$A = 0');

```

```

65 1  DECLARE      TP$11(*) BYTE DATA(OR,LF,
      'TP$11
      FRAME IS INCOMING 1 FRAME');
66 1  DECLARE      TP$12(*) BYTE DATA(OR,LF,
      'TP$12
      INPUT SEQ$BIT$A = 1');
67 1  DECLARE      TP$13(*) BYTE DATA(OR,LF,
      'TP$13
      INPUT SEQ$BIT$A = 0');
68 1  DECLARE      TP$14(*) BYTE DATA(OR,LF,
      'TP$14
      INCOMING FRAME LOCATED IN NETWORK CHANNEL B');
69 1  DECLARE      TP$15(*) BYTE DATA(OR,LF,
      'TP$15
      FRAME IS NT01RX TO NT01TX TRANSFER');
70 1  DECLARE      TP$16(*) BYTE DATA(OR,LF,
      'TP$16
      DATA IS DESTINED FOR THIS UNID');
71 1  DECLARE      TP$17(*) BYTE DATA(OR,LF,
      'TP$17
      FRAME IS INCOMING S FRAME');
72 1  DECLARE      TP$18(*) BYTE DATA(OR,LF,
      'TP$18
      THIS SEQ$BIT$B = 1');
73 1  DECLARE      TP$19(*) BYTE DATA(OR,LF,
      'TP$19
      THIS SEQ$BIT$B = 0');
74 1  DECLARE      TP$20(*) BYTE DATA(OR,LF,
      'TP$20
      FRAME IS INCOMING 1 FRAME');
75 1  DECLARE      TP$21(*) BYTE DATA(OR,LF,
      'TP$21
      INPUT SEQ$BIT$B = 1');
76 1  DECLARE      TP$22(*) BYTE DATA(OR,LF,
      'TP$22
      INPUT SEQ$BIT$B = 0');
77 1  DECLARE      TP$23(*) BYTE DATA(OR,LF,
      'TP$23
      PACKET LOCATED IN LONTTB');
78 1  DECLARE      TP$24(*) BYTE DATA(OR,LF,
      'TP$24
      INCOMING PACKET DESTINED FOR CHANNEL A');
79 1  DECLARE      TP$25(*) BYTE DATA(OR,LF,
      'TP$25
      INCOMING PACKET DESTINED FOR CHANNEL B');
80 1  DECLARE      TP$26(*) BYTE DATA(OR,LF,
      'TP$26
      OUTGOING FRAME LOCATED IN NETWORK CHANNEL A');
81 1  DECLARE      TP$27(*) BYTE DATA(OR,LF,
      'TP$27
      FRAME IS NT01TX TO CHANNEL A TRANSFER');
82 1  DECLARE      TP$28(*) BYTE DATA(OR,LF,
      'TP$28
      OUTGOING FRAME LOCATED IN NETWORK CHANNEL B');
83 1  DECLARE      TP$29(*) BYTE DATA(OR,LF,
      'TP$29
      FRAME IS NT02TX TO CHANNEL B TRANSFER');
84 1  DECLARE      TP$30(*) BYTE DATA(OR,LF,
      'TP$30
      DESTINATION OF NT01RX IS NL');
85 1  DECLARE      TP$31(*) BYTE DATA(OR,LF,
      'TP$31
      DESTINATION OF NT01RX IS NN');
86 1  DECLARE      TP$32(*) BYTE DATA(OR,LF,
      'TP$32
      DESTINATION OF NT02RX IS NL');
87 1  DECLARE      TP$33(*) BYTE DATA(OR,LF,
      'TP$33
      DESTINATION OF NT02RX IS NN');
88 1  DECLARE      TP$34(*) BYTE DATA(OR,LF,
      'TP$34
      DESTINATION$UNID >= THIS$UNID$NBR');
89 1  DECLARE      TP$35(*) BYTE DATA(OR,LF,
      'TP$35
      DESTINATION$UNID < THIS$UNID$NBR');
90 1  DECLARE      TP$36(*) BYTE DATA(OR,LF,

```

```

408 3      ;      /* ZERO IS NULL AND AN ERROR */
409 3      DO,    /* IF CALLED TO BUILD S FRAME FOR CH A */
410 4          ADDRESS$WORD = ROL(NT01TX (TX01NS), 4);
411 4          IF INPUT$SEQ$BIT = TRUE THEN
412 4              CONTROL$WORD = 0A0H;      /* CONT$WD = 10100000 */
413 4          ELSE CONTROL$WORD = 0B0H;      /* CONT$WD = 10000000 */
414 4          NT01RX (NT01NE+0) = ADDRESS$WORD;
415 4          NT01RX (NT01NE+1) = CONTROL$WORD;
416 4          DO INDEX = 2 TO (FRAME$SIZE - 1);
417 5              NT01RX (NT01NE + INDEX) = '/';
418 5          END;
419 4          CALL LD$TAB$H$KP (1);
420 4      END;

421 3      DO,    /* IF CALLED TO BUILD S FRAME FOR CH B */
422 4          ADDRESS$WORD = ROL(NT02TX (TX02NS), 4);
423 4          IF INPUT$SEQ$BIT = TRUE THEN
424 4              CONTROL$WORD = 0A0H;      /* CONT$WD = 10100000 */
425 4          ELSE CONTROL$WORD = 0B0H;      /* CONT$WD = 10000000 */
426 4          NT02RX (NT02NE+0) = ADDRESS$WORD;
427 4          NT02RX (NT02NE+1) = CONTROL$WORD;
428 4          DO INDEX = 2 TO (FRAME$SIZE - 1);
429 5              NT02RX (NT02NE + INDEX) = '/';
430 5          END;
431 4          CALL LD$TAB$H$KP (2);
432 4      END;

433 3      END;
434 2      END BUILD$S$FRAME;

```

/*****

PROCEDURE SERVICE\$LOOP

PURPOSE

FORMS A FRAME AROUND AT THE DATA LINK LAYER. THE SOURCE AND DESTINATION ADDRESSES ARE SWITCHED AND THE FRAME PUT INTO THE DATA LINK LAYER TABLE. AN APPROPRIATE SUPERVISORY FRAME IS SENT FOR RECEIVED FRAMES AND THE RECEIVED SUPERVISORY FRAMES ARE DUMPED.

INTERFACE

CALLS BY XMIT\$A AND XMIT\$B.

```

435 1      SERVICE$LOOP: PROCEDURE(TABLE);
436 2          DECLARE CHANNEL BYTE,
          ADDRESS$WORD BYTE,
          INP$SEQ$BIT BYTE,
          TABLE BYTE,
          INDEX INTEGER,
          REFIN INTEGER;

437 2          ON ENIT = 0;

```

```

376 3      DO;      /* IF CALLED TO BUILD S FRAME FOR CH A */
377 4          CALL SENDSEQ(@TP$40, LENGTH(TP$40));
378 4          ADDRESS$WORD = ROL(NT01RX (NT01NS), 4);
379 4          IF INPUT$SEQ$BIT = TRUE THEN
380 4              CONTROL$WORD = 0A0H;      /* CONT$WD = 10100000 */
381 4          ELSE CONTROL$WORD = 080H;      /* CONT$WD = 10000000 */
382 4          NT01TX (TX01NE+0) = ADDRESS$WORD;
383 4          NT01TX (TX01NE+1) = CONTROL$WORD;
384 4          DO INDEX = 2 TO (FRAME$SIZE - 1);
385 5              NT01TX (TX01NE + INDEX) = '/';
386 5          END;
387 4          CALL LD$TAB$H$SKP (3);
388 4      END;

389 3      DO;      /* IF CALLED TO BUILD S FRAME FOR CH B */
390 4          CALL SENDSEQ(@TP$41, LENGTH(TP$41));
391 4          ADDRESS$WORD = ROL(NT02RX (NT02NS), 4);
392 4          IF INPUT$SEQ$BIT = TRUE THEN
393 4              CONTROL$WORD = 0A0H;      /* CONT$WD = 10100000 */
394 4          ELSE CONTROL$WORD = 080H;      /* CONT$WD = 10000000 */
395 4          NT02TX (TX02NE+0) = ADDRESS$WORD;
396 4          NT02TX (TX02NE+1) = CONTROL$WORD;
397 4          DO INDEX = 2 TO (FRAME$SIZE - 1);
398 5              NT02TX (TX02NE + INDEX) = '/';
399 5          END;
400 4          CALL LD$TAB$H$SKP (4);
401 4      END;

402 3      END;
403 2      END BUILD$S$FRAME;

```

```

/*****
PROCEDURE BUILD$S$FRAME

```

PURPOSE

USED WITH SERVICE\$LOOP TO TURN THE FRAME AROUND AT THE DATA LINK LAYER.

INTERFACE

CALLLED BY SERVICE\$LOOP

```

404 1      BUILD$S$FRAME: PROCEDURE (TABLE, INPUT$SEQ$BIT, RE$EN$FR$MT);
405 2          DECLARE TABLE WORD,
                ADDRESS$WORD BYTE,
                CONTROL$WORD BYTE,
                INPUT$SEQ$BIT BYTE,
                INDEX      INTEGER;

406 2          IF TABLE <= 1 AND TABLE <= 2 THEN
407 2              DO CASE TABLE;

```

```

                                PACKET$SIZE);
367  4          CALL LD$TAB$H$SP (4);
368  4          END;
369  5          END;

370  2  END BUILD$I$FRAME;

```

```

/*****
PROCEDURE  BUILT$I$FRAME      BUILD A SUPERVISORY FRAME

```

THE PURPOSE OF THIS PROCEDURE IS TO BUILD A SUPERVISORY FRAME AND PLACE IT INTO THE APPROPRIATE OUTGOING TABLE FOR TRANSMISSION TO THE NETWORK VIA EITHER NETWORK CHANNEL A OR CHANNEL B.

INPUT

THIS PROCEDURE IS PASSED A TWO CHARACTER ASCII VALUE INDICATING THE LOCATION OF THE OUTGOING S FRAME AND THE SEQUENCE BIT OF THE INCOMING I FRAME (MODULE 2).

PROCESSING

THE PROCESSING BEGINS WITH THE PASSING OF THE TABLE WHERE THE NEWLY CREATED S FRAME IS TO BE PLACED. THE PROCEDURE THEN PERFORMS A SERIES OF LOGICAL OPERATIONS AND DIVISIONS TO SWAP THE HIGH 4 ORDER BITS OF THE ADDRESS WORD WITH THE LOWER ORDER BITS. THIS SIMPLY INTERCHANGES THE DESTINATION AND SOURCE ADDRESSES. IT THEN SETS THE CONTROL WORD ACCORDING TO THE SEQUENCE BIT OF THE INCOMING I FRAME. THESE BYTES ARE THEN PLACED IN THE FIRST AND SECOND HEADER POSITIONS OF THE OUTGOING NETWORK TABLE. APPROPRIATE TABLE POINTERS ARE THEN UPDATED.

OUTPUT

SEE PROCESSING ABOVE.

INTERFACE

THIS PROCEDURE IS CALLED BY ROUTE\$IN WHENEVER A NEW I FRAME ARRIVES.

NOTES: 1. THE X.25 AND LAP PROTOCOLS ALLOW FOR ADDITIONAL TYPES OF S FRAMES WHICH MAY BE INCORPORATED DURING FURTHER DEVELOPMENT.

```

*****/

```

```

371  1  BUILT$I$FRAME: PROCEDURE (TABLE, INPUT$SEQ$BIT) REENTRANT;

372  2      DECLARE TABLE WORD,
              ADDRESS$WORD BYTE,
              CONTROL$WORD BYTE,
              INPUT$SEQ$BIT BYTE,
              INDEX    INTEGER;

373  2      IF TABLE >= 1 AND TABLE <= 2 THEN
374  2          DO CASE TABLE;

375  5          ;          /* ZERO IS NULL AND AN ERROR */

```

NEWLY CREATED I FRAME IS TO BE PLACED. THE DESTINATION ADDRESS IS READ FROM THE PACKET HEADER LOCATED IN TABLE LCNTTB AND PLACED INTO THE FRAME HEADER FIELD. THE SOURCE ADDRESS IS THEN ADDED BY USING THE THIS\$UNIT\$NBR VARIABLE. THE CONTROL FIELD BYTE IS INITIALIZED TO ZERO AND WILL BE CHANGED BASED UPON LATER ACTION BY THE ROUTING PROCEDURE.

OUTPUT THIS PROCEDURE PLACES THE FIRST TWO BYTES INTO THE INDICATED OUTPUT TABLE BEFORE THE PACKET IS TRANSFERRED OVER TO THE CORRECT OUTPUT NETWORK TABLE.

INTERFACE THIS PROCEDURE IS CALLED FROM THE ROUTE\$IN PROCEDURE FOR THOSE DATA PACKETS DESTINED FOR THE NETWORK ONLY.

NOTES: 1. THE HEADER INFORMATION IS BASED ON THE FIELDS DEFINED BY PHISTER'S TESTS ONLY. EVEN THOUGH THEY CORRESPOND TO THE HOLL PROTOCOL, CAUTION SHOULD BE TAKEN BEFORE ADDING OR DELETING ADDITIONAL HEADERS.

*****/

```

343 1  BITED$I$FRAME: PROCEDURE(TABLE) REENTRANT;
344 2  DECLARE TABLE WORD,
      ADDRESS$BYTE BYTE,
      CONTROL$BYTE BYTE;

345 2  ADDRESS$BYTE = (LCNTTB(LCNTNS + 0) AND 0FH) OR
      (ROL(THIS$UNIT$NBR, 4) AND 0F0H);

346 2  IF TABLE >= 1 AND TABLE <= 2 THEN
347 2  DO CASE TABLE;

348 3      ; /* ZEROth ENTRY IS NULL AND AN ERROR */

349 3  DO; /* I FRAME IS GOING OUT NETWORK CHANNEL A */
350 4  CALL SENDSEQ(@TP$38, LENGTH(TP$38));
351 4  NT01TX (TX01NE+0) = ADDRESS$BYTE;
352 4  IF SEQ$BIT$A = FALSE THEN
353 4  CONTROL$BYTE = 0H;
354 4  ELSE CONTROL$BYTE = 20H;
355 4  NT01TX (TX01NE+1) = CONTROL$BYTE;
356 4  CALL MOVBC (@LCNTTB (LCNTNS), @NT01TX (TX01NE+2),
      PACKET$SIZE);
357 4  CALL LD$TAB$H$SKP (3);
358 4  END;

359 3  DO; /* I FRAME IS GOING OUT NETWORK CHANNEL B */
360 4  CALL SENDSEQ(@TP$39, LENGTH(TP$39));
361 4  NT02TX (TX02NE+0) = ADDRESS$BYTE;
362 4  IF SEQ$BIT$B = FALSE THEN
363 4  CONTROL$BYTE = 0H;
364 4  ELSE CONTROL$BYTE = 20H;
365 4  NT02TX (TX02NE+1) = CONTROL$BYTE;
366 4  CALL MOVBC (@LCNTTB (LCNTNS), @NT02TX (TX02NE+2),

```

```

312 4          IF NT01NS >= NT01SZ THEN          /* IF TABLE WRAP */
313 4              NT01NS = 0;
314 4          END;

315 3          DO; /* IF CALLED TO HSKP INCOMING NET CH B TABLE */
316 4              NT02NS = NT02NS + FRAME$SIZE;
317 4              IF NT02NS >= NT02SZ THEN          /* IF TABLE WRAP */
318 4                  NT02NS = 0;
319 4          END;

320 3          DO; /* IF CALLED TO HSKP OUTGOING NET CH A TABLE */
321 4              TX01NS = TX01NS + FRAME$SIZE;
322 4              IF TX01NS >= TX01SZ THEN          /* IF TABLE WRAP */
323 4                  TX01NS = 0;
324 4          END;

325 3          DO; /* IF CALLED TO HSKP OUTGOING NET CH B TABLE */
326 4              TX02NS = TX02NS + FRAME$SIZE;
327 4              IF TX02NS >= TX02SZ THEN          /* IF TABLE WRAP */
328 4                  TX02NS = 0;
329 4          END;

330 3          DO; /* IF CALLED TO HSKP NETWORK TO LOCAL TABLE */
331 4              NTLONS = NTLONS + PACKET$SIZE;
332 4              IF NTLONS >= NTL0SZ THEN
333 4                  NTLONS = 0;
334 4          END;

335 3          DO; /* IF CALLED TO HOUSEKEEP LOCAL TO NET TABLE */
336 4              LONTNS = LONTNS + PACKET$SIZE;
337 4              IF LONTNS >= LONTSZ THEN          /* IF TABLE WRAP */
338 4                  LONTNS = 0;
339 4          END;

340 3          END;
341 2          ELSE HSKP$ERR = HSKP$ERR + 1;
342 2          END S$VC$TAB$HSKP;

```

```

/*****
PROCEDURE BUILD$I$FRAME      TRANSFORM A PACKET INTO AN I FRAME.

```

THE PURPOSE OF THIS PROCEDURE IS TO TRANSFORM A PACKET OF DATA DELIVERED FROM THE LOCAL SIDE OF THE UNID INTO AN I FRAME TO BE PLACED INTO ONE OF THE OUTGOING NET TABLES. THIS PROCEDURE ADDS ON THE TWO HEADER BYTES: ADDRESS BYTE AND CONTROL BYTE.

INPUT THIS PROCEDURE RECEIVES A TWO CHARACTER ASCII VALUE INDICATING THE LOCATION OF THE OUTGOING NETWORK PORT WHERE THE NEWLY CREATED I FRAME IS TO BE PLACED.

PROCESSING THE PROCESSING BEGINS WITH THE PASSING OF THE TABLE WHERE THE

```

289 4          IF TX02NE >= TX02SZ THEN /* IF TABLE WRAP */
290 4              TX02NE = 0;
291 4          END;

292 3          DO; /* HSKP NETWORK TO LOCAL TABLE */
293 4              NTLONE = NTLONE + PACKET$SIZE;
294 4              IF NTLONE >= NTLOSZ THEN /* IF TABLE WRAP */
295 4                  NTLONE = 0;
296 4          END;

297 3          DO; /* HSKP LOCAL TO NETWORK TABLE */
298 4              LONTNE = LONTNE + PACKET$SIZE;
299 4              IF LONTNE >= LONTSZ THEN /* IF TABLE WRAP */
300 4                  LONTNE = 0;
301 4          END;

302 3          END;
303 2          ELSE HSKP$ERR = HSKP$ERR + 1;
304 2          END LD$TAB$HSKP;

```

/******

PROCEDURE SRVC\$TAB\$HSKP SERVICE TABLE HOUSEKEEP

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED TABLE AFTER
SERVICING (REMOVING EITHER A PACKET OR FRAME).

INPUT THE INPUT IS AN INTEGER VALUE INDICATING THE TABLE REQUIRING
HOUSEKEEPING.

PROCESSING THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED, ADVANCES
THE NEXT BYTE TO BE SERVICED POINTER BY AA PACKET OR FRAME SIZE, AND
ADJUSTS FOR TABLE WRAP IF NECESSARY.

OUTPUT THE SPECIFIED TABLE HAS ITS NEXT BYTE TO BE SERVICED POINTER
ADVANCED BY THE LENGTH OF A PACKET OR FRAME.

INTERFACE THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN AND ROUTE\$OUT.

NOTES: NONE.

```

305 1          SRVC$TAB$HSKP: PROCEDURE(TABLE) REENTRANT;
306 2          DECLARE TABLE BYTE;

307 2          IF TABLE >= 1 AND TABLE <= 6 THEN
308 2              DO CASE TABLE;

309 3              ; /* ZEROth ENTRY IS NULL AND AN ERROR */

310 3              DO; /* IF CALLED TO HSKP INCOMING NET CH A TABLE */
311 4                  NTOINS = NTOINS + FRAME$SIZE;

```

```

266 2  END DET$DEST$LN;
      /*****
PROCEDURE  LD$TAB$H$SKP          LOAD TABLE HOUSEKEEPING

      THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED TABLE AFTER
      THE LOADING OF A PACKET OR A FRAME.

INPUT      THE INPUT IS AN INTEGER VALUE INDICATING THE TABLE REQUIRING
            HOUSEKEEPING.

PROCESSING THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED, ADVANCES
            THE NEXT EMPTY BYTE POINTER BY A PACKET OR FRAME SIZE, AND ADJUSTS
            FOR TABLE WRAP IF NECESSARY.

OUTPUT     THE SPECIFIED TABLE HAS ITS NEXT EMPTY BYTE POINTER ADVANCED BY
            THE LENGTH OF A PACKET OR A FRAME.

INTERFACE  THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE$IN.

NOTES      NONE.
*****/
267 1  LD$TAB$H$SKP: PROCEDURE(TABLE) REENTRANT;

268 2      DECLARE TABLE  BYTE;

269 2      IF (TABLE >= 1 AND TABLE <= 6) THEN
270 2      DO CASE TABLE;

271 3          ;          /* ZERO CASE IS NULL AND IS AN ERROR CASE */

272 3      DO;          /* HSKP INCOMING NET CH A TABLE */
273 4          NTO1NE = NTO1NE + FRAME$SIZE;
274 4          IF NTO1NE >= NTO1SZ THEN /* IF TABLE WRAP */
275 4              NTO1NE = 0;
276 4      END;

277 3      DO;          /* HSKP INCOMING NET CH B TABLE */
278 4          NTO2NE = NTO2NE + FRAME$SIZE;
279 4          IF NTO2NE >= NTO2SZ THEN /* IF TABLE WRAP */
280 4              NTO2NE = 0;
281 4      END;

282 3      DO;          /* HSKP OUTGOING NET CH A TABLE */
283 4          TXO1NE = TXO1NE + FRAME$SIZE;
284 4          IF TXO1NE >= TXO1SZ THEN /* IF TABLE WRAP */
285 4              TXO1NE = 0;
286 4      END;

287 3      DO;          /* HSKP OUTGOING NET CH B TABLE */
288 4          TXO2NE = TXO2NE + FRAME$SIZE;

```

INTO THE NETWORK.

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE OUTGOING NETWORK CHANNEL FOR THE PACKET LOCATED IN TABLE LCNTTB.

INPUT < THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE REQUIRING EVALUATION. >

PROCESSING THIS PROCEDURE CALCULATES THE NUMBER OF HOPS REQUIRED TO GO FROM THIS UNID TO THE DESTINATION UNID BOTH RIGHT AND LEFT.

OUTPUT THIS PROCEDURE RETURNS 'CA' IF THE DISTANCE IS SHORTER GOING OUT THE RIGHT RING AND RETURNS 'CB' IF THE DISTANCE IS SHORTER GOING OUT THE LEFT RING.

INTERFACE THIS PROCEDURE IS CALLED BY PROCEDURE ROUTESIN.

NOTES 1. THIS PROCEDURE IS A PRIMITIVE ROUTING ALGORITHM THAT JUST CALCULATES THE PATH DISTANCE BOTH RIGHT AND LEFT.

2. THIS PROCEDURE DOES NOT TAKE INTO ACCOUNT ANY FLOW CONTROL.

*****/

```

242 1  DET$DEST$LN: PROCEDURE WORD REENTRANT;

243 2      DECLARE      DESTINATION$UNID  INTEGER,
                        DISTANCE$LEFT    INTEGER,
                        DISTANCE$RIGHT    INTEGER;

244 2      DESTINATION$UNID = INT(LCNTTB (LCNTNS) AND 07H);
245 2      IF DESTINATION$UNID >= INT(THIS$UNID$NBR) THEN DO;
247 3          CALL SENDSEQ(@TP$34, LENGTH(TP$34));
248 3          DISTANCE$LEFT = DESTINATION$UNID - INT(THIS$UNID$NBR);
249 3          DISTANCE$RIGHT = INT(THIS$UNID$NBR) + (INT(MAX$NETWORK$CODE) -
                        DESTINATION$UNID + 1);
250 3          END;
251 2      ELSE DO;
252 3          CALL SENDSEQ(@TP$35, LENGTH(TP$35));
253 3          DISTANCE$LEFT = INT(THIS$UNID$NBR) +
                        (INT(MAX$NETWORK$CODE) - DESTINATION$UNID + 1);
254 3          DISTANCE$RIGHT = DESTINATION$UNID - INT(THIS$UNID$NBR);
255 3          END;
256 2      IF DISTANCE$LEFT >= DISTANCE$RIGHT THEN DO;
258 3          CALL SENDSEQ(@TP$36, LENGTH(TP$36));
259 3          DESTINATION = 'CA';
260 3          END;
261 2      ELSE DO;
262 3          CALL SENDSEQ(@TP$37, LENGTH(TP$37));
263 3          DESTINATION = 'CB';
264 3          END;
265 2      RETURN DESTINATION;

```

PROCEDURE DET\$DEST\$TWO DETERMINE DEST OF INCOMING FRAME FROM CHANNEL B
THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF A
SPECIFIED INCOMING FRAME COMING INTO CHANNEL A (B).

INPUT THE INPUT IS AN INTEGER VALUE INDICATING THE TABLE LOCATION
OF THE FRAME TO BE EVALUATED.

PROCESSING THIS PROCEDURE LOOKS AT THE FIRST BYTE OF THE NETWORK INPUT
TABLE (NT01RX) SERVICING CH A AND DETERMINES IF THE DATA FRAME IS
DESTINED FOR THIS PARTICULAR UNIT OR SOME OTHER UNIT. IF IT GOES TO
THIS UNIT, THEN 'NL', IF TO SOME OTHER UNIT, THEN 'NN'.

OUTPUT THE PROCEDURE OUTPUTS A TWO CHARACTER ASCII VALUE INDICATING THE
TABLE OR CHANNEL DESTINATION OF THE FRAME.

INTERFACE THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN FOR INPUT
FRAMES.

NOTES 1. FOR THIS PROCEDURE, S FRAMES DESTINED FOR THIS UNIT ARE 'NL'.
*****/

```

218 1  DET$DEST$ONE: PROCEDURE WORD REENTRANT;

219 2      IF (NT01RX (NT01NS) AND OFH) = THIS$UNID$NBR THEN DO;
221 3          CALL SENDSEQ(@TP$30, LENGTH(TP$30));
222 3          DESTINATION = 'NL';
223 3          END;
224 2      ELSE DO;
225 3          CALL SENDSEQ(@TP$31, LENGTH(TP$31));
226 3          DESTINATION = 'NN';
227 3          END;

228 2      RETURN DESTINATION;
229 2  END DET$DEST$ONE;

230 1  DET$DEST$TWO: PROCEDURE WORD REENTRANT;

231 2      IF (NT02RX (NT02NS) AND OFH) = THIS$UNID$NBR THEN DO;
233 3          CALL SENDSEQ(@TP$32, LENGTH(TP$32));
234 3          DESTINATION = 'NL';
235 3          END;
236 2      ELSE DO;
237 3          CALL SENDSEQ(@TP$33, LENGTH(TP$33));
238 3          DESTINATION = 'NN';
239 3          END;

240 2      RETURN DESTINATION;
241 2  END DET$DEST$TWO;

```

/*****
PROCEDURE DET\$DEST\$IN DETERMINES DESTINATION OF AN OUTGOING PACKET

THE PURPOSE OF THIS PROCEDURE IS TO PRODUCE A TIME DELAY TO BE USED BETWEEN SUCCESSIVE TRANSMISSIONS OF 1 FRAMES OUT OF CH A WHEN A VALID ACKNOWLEDGEMENT HAS NOT BEEN RECEIVED.

INPUT NONE.

PROCESSING - THE PROCEDURE BEGINS BY CALLING THE (ASSEMBLY) ROUTINE STCTOE WHICH INITIALIZES AND STARTS THE CTC CHANNEL #x. EACH TIME THE CTC RUNS OUT, ITS INTERRUPT ROUTINE (TOUTCA) INCREMENTS THE GLOBAL VARIABLE 'CTCNOA'. WHENEVER THE CTCNOA VALUE IS GREATER OR EQUAL TO THE GLOBAL VARIABLE 'MAXNOA', THE GLOBAL VARIABLE 'TIMCHA' IS CHANGED TO TRUE.

OUTPUT - THE OUTPUT IS THE VARIABLE 'TIMCHA'. WHEN TRUE THE TIME DELAY SEQUENCE IS COMPLETE.

INTERFACE - THIS PROCEDURE IS CALLED BY THE PROCEDURE ROUTE\$OUT LOCATED WITHIN THIS SAME MODULE. THIS PROCEDURE CALLS PROCEDURE STCTCA WHICH IS LOCATED IN THIS PROGRAM.

NOTES: 1. SEE PROCEDURE STCTCA FOR AN EXAMPLE OF CALCULATING THE APPROPRIATE TIME DELAY.
2. VARIABLES TOUTCA, CTCNOA, AND MAXNOA ALLOW CHANNEL A TO BE INDEPENDENT OF CHANNEL B WHEN CONSIDERING TIME DELAY TO AN ADJACENT UNIT.

```

*****/
198 1  TIME$DELAY$CHA: PROCEDURE REENTRANT;

199 2      IF TIMCHA = TRUE THEN
200 2          CALL STCTCA;     /* IF TIMER IS NOT RUNNING, THEN START IT */
      ELSE
201 2          CTCNOA = CTCNOA + 1;
202 2          IF CTCNOA >= MAXNOA THEN DO;
204 3              TIMCHA = TRUE;
205 3              CTCNOA = 0;
206 3          END;
207 2  END TIME$DELAY$CHA;

208 1  TIME$DELAY$CHB: PROCEDURE REENTRANT;

209 2      IF TIMCHB = TRUE THEN
210 2          CALL STCTCB;     /* IF TIMER IS NOT RUNNING, THEN START IT */
      ELSE
211 2          CTCNOB = CTCNOB + 1;
212 2          IF CTCNOB >= MAXNOB THEN DO;
214 3              TIMCHB = TRUE;
      CTCNOB = 0;
216 3          END;
217 2  END TIME$DELAY$CHB;

/*****
PROCEDURE DET$DEST$ONE     DETERMINE DEST OF INCOMING FRAME FROM CHANNEL A.

```

```

177 2      END ASC$HEX;

178 1      VALID$HEX: PROCEDURE (H) BYTE;

179 2          DECLARE (H, I) BYTE;

180 2          DO I = 0 TO LAST(ASCII);
181 3              IF H = ASCII(I) THEN
182 3                  RETURN TRUE;
183 3              END;
184 2          RETURN FALSE;

185 2      END VALID$HEX;

/*****
PROCEDURE SENDSEQ      SENDS DATA TO LOCAL MONITOR FOR TESTING

      THIS PROCEDURE TAKES A MESSAGE STRING AND OUTPUTS IT TO THE
      SYSTEM CONSOLE.
*****/

186 1      SENDSEQ: PROCEDURE (MSG, TOTAL);
187 2          DECLARE MSG POINTER;
188 2          DECLARE TOTAL WORD;

189 2          CALL DQ$WRITE (W$CONN, MSG, TOTAL, @STATUS);
190 2          CALL ERR$CHK;

191 2      END SENDSEQ;

/*****
      DUMMY PROCEDURES FOR ACTIVATING THE HARDWARE AND SOFTWARE TIMERS
      USED IN THE TIME$DELAY$CHA AND TIME$DELAY$CHB PROCEDURES.
*****/

192 1      STOTCA: PROCEDURE;

193 2          TIMCHA = FALSE;

194 2      END STOTCA;

195 1      STOTCB: PROCEDURE;

196 2          TIMCHB = FALSE;

197 2      END STOTCB;

/*****
PROCEDURE  TIME$DELAY$CHA      PRODUCES A DELAY OF TIME FOR CHANNEL A
          TIME$DELAY$CHB      PRODUCES A DELAY OF TIME FOR CHANNEL B

```

```

145 2      NT01NE = 0;
146 2      NT01SZ = FRAME$TABLE$SIZE;

147 2      NT02NS = 0;
148 2      NT02NE = 0;
149 2      NT02SZ = FRAME$TABLE$SIZE;

150 2      TX01NS = 0;
151 2      TX01NE = 0;
152 2      TX01SZ = FRAME$TABLE$SIZE;

153 2      TX02NS = 0;
154 2      TX02NE = 0;
155 2      TX02SZ = FRAME$TABLE$SIZE;

156 2      LONTNS = 0;
157 2      LONTNE = 0;
158 2      LONTSZ = PACKET$TABLE$SIZE;

159 2      NTLONS = 0;
160 2      NTLONE = 0;
161 2      NTLCNZ = PACKET$TABLE$SIZE;

162 2      END INIT$NSTAB;

```

/*****
 PROCEDURES TO CONVERT HEX TO ASCII FOR USE WITH DISPLAYING A FRAME
 *****/

```

163 1      HEX$ASC: PROCEDURE(B) WORD;
164 2      DECLARE B BYTE;
165 2      DECLARE W WORD;

166 2      W = ASCII(SHR(B,4) AND 0FH);
167 2      W = ROL(W,8) AND 0FF00H;
168 2      W = W OR ASCII(B AND 0FH);
169 2      W = ROL(W,8);

170 2      RETURN W;

171 2      END HEX$ASC;

172 1      ASC$HEX: PROCEDURE(C) WORD;
173 2      DECLARE C BYTE;

174 2      IF C <= '9' THEN
175 2          RETURN DOUBLE(C-30H);
176 2      ELSE
177 2          RETURN DOUBLE(C-37H);

```

```

115 1    DECLARE    TP$59(*) BYTE DATA(CR,LF,
        'TP$59          ACK B = TRUE');

116 1    DECLARE    MSG1(*) BYTE DATA(CR,LF,
        'Do you want to load the test message? ');
117 1    DECLARE    MSG2(*) BYTE DATA(CR,LF,
        'Do you want to stop the test? ');
118 1    DECLARE    MSG3(*) BYTE DATA(CR,LF,
        'How many packets to load (1-9)? ');
119 1    DECLARE    MSG4(*) BYTE DATA(CR,LF,
        'Send to which UNID (1 or 3)? ');

/*****

120 1    ERR$CHK: PROCEDURE PUBLIC;
121 2        IF STATUS <> 0 THEN          /* <> 0 IS AN ERROR */
122 2            DO;
123 3                CALL DQ$DECODE$EXCEPTION ( STATUS, @STATUS$PTR, @STATUS);
124 3                CALL DQ$WRITE (W$CONN, @STATUS$PTR(1), STATUS$PTR(0), @STATUS);
125 3                CALL DQ$WRITE (W$CONN, @CRLF, 2, @STATUS);
126 3                CALL DQ$EXIT (0);
127 3            END;
128 2    END ERR$CHK;

/*****

129 1    INIT$N$TAB: PROCEDURE;

        /* VARIABLES FOR NETWORK CHANNEL A */
130 2        SEQ$BIT$A = FALSE;          /* MOD 2 SEQUENCE BIT */

        /* VARIABLES FOR NETWORK CHANNEL B */
131 2        SEQ$BIT$B = FALSE;          /* MOD 2 SEQUENCE BIT */

        /* VARIABLES FOR BOTH NETWORK CHANNELS */
132 2        INPUT$SEQ$BIT$A = FALSE;    /* SEQ BIT GOING INTO FRAME */
133 2        INPUT$SEQ$BIT$B = FALSE;    /* SEQ BIT GOING INTO FRAME */
134 2        THIS$SEQ$BIT$A = FALSE;     /* CURRENT SEQUENCE BIT */
135 2        THIS$SEQ$BIT$B = FALSE;     /* CURRENT SEQUENCE BIT */

136 2        TIMCHA = TRUE;              /* FLAG FOR CH A WAIT LOOP */
137 2        TIMCHB = TRUE;              /* FLAG FOR CH B WAIT LOOP */
138 2        CTONOA = 0;
139 2        CTONOB = 0;

140 2        FOREVER = TRUE;             /* FLAG FOR MAIN LOOP */
141 2        MSGNJM = 0;

142 2        RETRANS$A = 0;
143 2        RETRANS$B = 0;

144 2        NT01NS = 0;

```

```

      'TP$36          DESTINATION IS CHANNEL A');
91  1  DECLARE      TP$37(*) BYTE DATA(CR,LF,
      'TP$37          DESTINATION IS CHANNEL B');
92  1  DECLARE      TP$38(*) BYTE DATA(CR,LF,
      'TP$38          BUILDING I FRAME FOR CHANNEL A');
93  1  DECLARE      TP$39(*) BYTE DATA(CR,LF,
      'TP$39          BUILDING I FRAME FOR CHANNEL B');
94  1  DECLARE      TP$40(*) BYTE DATA(CR,LF,
      'TP$40          BUILDING S FRAME FOR CHANNEL A');
95  1  DECLARE      TP$41(*) BYTE DATA(CR,LF,
      'TP$41          BUILDING S FRAME FOR CHANNEL B');
96  1  DECLARE      TP$42(*) BYTE DATA(CR,LF,
      'TP$42          ENTERING ROUTE$OUT PROCEDURE');
97  1  DECLARE      TP$43(*) BYTE DATA(CR,LF,
      'TP$43          END OF ROUTE$IN - ROUTE$OUT LOOP');
98  1  DECLARE      TP$44(*) BYTE DATA(CR,LF,
      'TP$44          HAVE EXITED ROUTE$IN - ROUTE$OUT LOOP',CR,LF);
99  1  DECLARE      TP$45(*) BYTE DATA(CR,LF,
      'TP$45          DATA IS NT01RX TO NTLCIB TRANSFER');
100 1  DECLARE      TP$46(*) BYTE DATA(CR,LF,
      'TP$46          DATA IS NT02RX TO NTLCIB TRANSFER');
101 1  DECLARE      TP$47(*) BYTE DATA(CR,LF,
      'TP$47          ERROR: UNID DESIGNATION > MAX$NETWORK$CODE');
102 1  DECLARE      TP$48(*) BYTE DATA(CR,LF,
      'TP$48          ERROR: UNID DESIGNATION:1 > MAX$NETWORK$CODE');
103 1  DECLARE      TP$49(*) BYTE DATA(CR,LF,
      'TP$49          INPUT SEQ$BIT$A = 1');
104 1  DECLARE      TP$50(*) BYTE DATA(CR,LF,
      'TP$50          INPUT SEQ$BIT$A = 0');
105 1  DECLARE      TP$51(*) BYTE DATA(CR,LF,
      'TP$51          INPUT SEQ$BIT$B = 1');
106 1  DECLARE      TP$52(*) BYTE DATA(CR,LF,
      'TP$52          INPUT SEQ$BIT$B = 0');

107 1  DECLARE      TP$53A(*) BYTE DATA(CR,LF,
      'TP$53A          GENERATING I FRAME BACK TO SENDER');
108 1  DECLARE      TP$53B(*) BYTE DATA(CR,LF,
      'TP$53B          S FRAME IN NETWORK AND DISCARDED');
109 1  DECLARE      TP$53C(*) BYTE DATA(CR,LF,
      'TP$53C          GENERATING S FRAME BACK TO SENDER');
110 1  DECLARE      TP$54(*) BYTE DATA(CR,LF,
      'TP$54          LOADED TEST PACKET IN LCNTTB');
111 1  DECLARE      TP$55(*) BYTE DATA(CR,LF,
      'TP$55          READING NTLCIB PACKET TO CONOUT');
112 1  DECLARE      TP$56(*) BYTE DATA(CR,LF,
      'TP$56          RETRANS$A >= 10; DUMPED FRAME');
113 1  DECLARE      TP$57(*) BYTE DATA(CR,LF,
      'TP$57          RETRANS$B >= 10; DUMPED FRAME');

114 1  DECLARE      TP$58(*) BYTE DATA(CR,LF,
      'TP$58          ACK A = TRUE');

```

```

438 2      DO CASE TABLE;
439 3          ;          /* ZERO IS NULL CASE */
440 3      DO;
441 4      IF ((TX01NE - TX01NS) >= FRAME$SIZE) OR
          (TX01NS > TX01NE) THEN DO;

443 5          DO INDEX = 0 TO (FRAME$SIZE - 1);
444 6          ADDR(INDEX) = NT01TX(TX01NS + INDEX);
445 6      END;
446 5      IF (ADDR(ORIGIN + 1) AND 80H) = 80H THEN DO;
          /* DUMP THE ACK FRAME */
448 6          CALL SENDSEQ(@TP$55B, LENGTH(TP$55B));
449 6      END;
450 5      ELSE DO;
451 6          IF (ADDR(ORIGIN + 1) AND 20H) = 20H THEN /* SET THE SEQ BIT */
452 6              INP$SEQ$BIT = TRUE;
453 6          ELSE INP$SEQ$BIT = FALSE;
454 6          CALL BUILDLS$FRAME(1, INP$SEQ$BIT);
455 6          CALL SENDSEQ(@TP$55C, LENGTH(TP$55C));
456 6          CALL SENDSEQ(@TP$55A, LENGTH(TP$55A));
457 6          NT01RX(NT01NE) = ROL(ADDR(ORIGIN), 4); /* SWAP FRAME HEADER */
458 6          NT01RX(NT01NE + 1) = ADDR(ORIGIN + 1);
459 6          NT01RX(NT01NE + 2) = ADDR(ORIGIN + 3); /* SWAP PACKET HEADER */
460 6          NT01RX(NT01NE + 3) = ADDR(ORIGIN + 2);
461 6          DO INDEX = 4 TO (FRAME$SIZE - 1);          /* SWAP DATA */
462 7              NT01RX(NT01NE + INDEX) = ADDR(ORIGIN + INDEX);
463 7          END;
464 5          DO INDEX = 0 TO 3;          /* SWAP IP DEST & SOURCE */
465 7              NT01RX(NT01NE + INDEX + 19) = ADDR(ORIGIN + INDEX + 23);
466 7              NT01RX(NT01NE + INDEX + 23) = ADDR(ORIGIN + INDEX + 19);
467 7          END;
468 6          CALL LD$TAB$H$SKIP(1);
469 6      END;
470 5      END;
471 4      END;          /* CASE = 1 */

472 3      DO;
473 4      IF ((TX02NE - TX02NS) >= FRAME$SIZE) OR
          (TX02NS > TX02NE) THEN DO;

475 5          DO INDEX = 0 TO (FRAME$SIZE - 1);
476 6          ADDR(INDEX) = NT02TX(TX02NS + INDEX);
477 6      END;
478 5      IF (ADDR(ORIGIN + 1) AND 80H) = 80H THEN DO;
          /* DUMP THE ACK FRAME */
480 6          CALL SENDSEQ(@TP$55B, LENGTH(TP$55B));
481 6      END;
482 5      ELSE DO;

```

```

493 6      IF (ADDR(ORIGIN + 1) AND 20H) = 20H THEN /* SET THE SEQ BIT */
494 6          INP$SEQ$BIT = TRUE;
495 6      ELSE INP$SEQ$BIT = FALSE;
496 6      CALL BUILD$LS$FRAME(2, INP$SEQ$BIT);
497 6      CALL SENDSEQ(@TP$53C, LENGTH(TP$53C));
498 6      CALL SENDSEQ(@TP$53A, LENGTH(TP$53A));
499 6      NTO2RX(NT02NE) = ROL(ADDR(ORIGIN), 4); /* SWAP FRAME HEADER */
500 6      NTO2RX(NT02NE + 1) = ADDR(ORIGIN + 1);
501 6      NTO2RX(NT02NE + 2) = ADDR(ORIGIN + 3); /* SWAP PACKET HEADER */
502 6      NTO2RX(NT02NE + 3) = ADDR(ORIGIN + 2);
503 6      DO INDEX = 4 TO (FRAME$SIZE - 1); /* SWAP DATA */
504 7          NTO2RX(NT02NE + INDEX) = ADDR(ORIGIN + INDEX);
505 7      END;
506 6      DO INDEX = 0 TO 3; /* SWAP IP DEST & SOURCE */
507 7          NTO2RX(NT02NE + INDEX + 19) = ADDR(ORIGIN + INDEX + 23);
508 7          NTO2RX(NT02NE + INDEX + 23) = ADDR(ORIGIN + INDEX + 19);
509 7      END;
510 6      CALL LD$TAB$H$OP(2);
511 6      END;
512 5      END;
513 4      END; /* CASE = 2 */

514 3      END; /* CASE TABLE */

515 2      END SERVICE$LOOP;

516 1      SERVICE$XMIT$A: PROCEDURE;

517 2          CALL SERVICE$LOOP(1);

518 2      END SERVICE$XMIT$A;

519 1      SERVICE$XMIT$B: PROCEDURE;

520 2          CALL SERVICE$LOOP(2);

521 2      END SERVICE$XMIT$B;

/******
PROCEDURE LOAD AND READ$TAB

THESE PROCEDURES ARE USED TO PUT A MESSAGE INTO THE LONTTB TABLE AND
WRITE TO THE ORT A MESSAGE IN THE NTLOTP TABLE.
******/

522 1      LOAD: PROCEDURE(DEST$UNIT);

523 2          DECLARE INDEX INTEGER;
524 2          DECLARE DEST$UNIT WORD;

525 2          DO INDEX = 0 TO (PACKET$SIZE - 1);
526 3              LONTTB(LONTNE + INDEX) = '*';

```

```

517 3      END;
      /* PACKET HEADER */
518 2      LONITB(LONTNE) = (LOW(DEST$UNIT) AND 0FH) OR 10H;
                        /* DEST = 1, CH = 1 */
519 2      LONITB(LONTNE + 1) = (THIS$UNIT$NBR AND 0FH) OR 10H;
                        /* SRC = 3, CH = 1 */
      /* IP SOURCE HEADER */
520 2      LONITB(LONTNE + 19) = 10H;      /* CF = 0, CC = 1 */
521 2      LONITB(LONTNE + 20) = (THIS$UNIT$NBR AND 0FH) OR 60H;
                        /* NC = 3, HC(L) = 6 */
522 2      LONITB(LONTNE + 21) = 70H;      /* HC(H) = 0, PC(0) = 7 */
523 2      LONITB(LONTNE + 22) = 00H;      /* PC(1) = 0, PC(2) = 0 */

      /* IP DESTINATION HEADER */
524 2      LONITB(LONTNE + 23) = 10H;      /* CF = 0, CC = 1 */
525 2      LONITB(LONTNE + 24) = (LOW(DEST$UNIT) AND 0FH) OR 60H;
                        /* NC = 1, HC(L) = 6 */
526 2      LONITB(LONTNE + 25) = 70H;      /* HC(H) = 0, PC(0) = 7 */
527 2      LONITB(LONTNE + 26) = 00H;      /* PC(1) = 0, PC(2) = 0 */

528 2      CALL MOVBC(MESSAGE, @LONITB(LONTNE + 61), TOP$DATA$SIZE);
529 2      IF MSGNUM > 9 THEN
530 2          MSGNUM = 0;
531 2      LONITB(LONTNE + 91) = '0' + MSGNUM;
532 2      MSGNUM = MSGNUM + 1;
533 2      CALL LD$TAB$H$SKP(5);
534 2      CALL SENDSEQ(@IP$54, LENGTH(IP$54));

535 2      END LOAD;

536 1      READ$TAB:  PROCEDURE;

537 2          IF ((NTLONE - NTLONS) >= PACKET$SIZE) OR (NTLONS > NTLONE) THEN DO;
538 3              CALL SENDSEQ(@TP$55, LENGTH(TP$55));
539 3              CALL SENDSEQ(@NTLCTB(NTLONS + 61), TOP$DATA$SIZE);
540 3              CALL SRVCTAB$H$SKP(5);
541 3          END;

542 3      END;

543 2      END READ$TAB;

      /******
PROCEDURE FIND$FRAME FINDS THE FIRST 1 FRAME IN THE NTO1TX, NTO2TX TABLES

      THE PURPOSE OF THIS PROCEDURE IS TO FIND THE FIRST 1 FRAME IN THE
      NTO1TX, NTO2TX TABLES AND RETURN A TRUE BOOLEAN VALUE IF THERE
      IS AN 1 FRAME IN THE SELECTED TABLE.
      *****/
544 1      FIND$FRAME:  PROCEDURE (TABLE) BY 1;

545 2      DECLARE (TABLE,
                FRAME$HERE,

```

```

                                HAVE$IFRAME) BYTE;
546  2      DECLARE INDEX    INTEGER;

547  2      FRAME$THERE = TRUE;
548  2      HAVE$IFRAME = FALSE;
549  2      INDEX = 0;

550  2      DO CASE TABLE;

551  3          ;            /* ZERO IS NULL */

552  3          DO;
553  4          DO WHILE FRAME$THERE;

554  5              IF ((TX01NE - TX01NS - INDEX) >= FRAME$SIZE) OR
                    ((TX01NS + INDEX) > TX01NE) THEN DO;
555  6                  IF (NT01TX(TX01NS + INDEX + 1) AND 80H) = 0H
                    THEN DO;            /* HAVE AN INFORMATION FRAME */
556  7                      HAVE$IFRAME = TRUE;
557  7                      FRAME$THERE = FALSE;
558  7                      END;
                    ELSE
559  6                      HAVE$IFRAME = FALSE;    /* HAVE AN S FRAME */
560  6                      END;
561  5      ELSE DO;
562  6          HAVE$IFRAME = FALSE;
563  6          FRAME$THERE = FALSE;
564  6          END;

565  5      INDEX = INDEX + FRAME$SIZE;
566  5      END;            /* DO WHILE FRAME$THERE */
567  4      END;            /* CASE TABLE = 1 */

568  3      DO;
569  4      DO WHILE FRAME$THERE;

570  5          IF ((TX02NE - TX02NS - INDEX) >= FRAME$SIZE) OR
                    ((TX02NS + INDEX) > TX02NE) THEN DO;
571  6                  IF (NT02TX(TX02NS + INDEX + 1) AND 80H) = 0H
                    THEN DO;            /* HAVE AN INFORMATION FRAME */
572  7                      HAVE$IFRAME = TRUE;
573  7                      FRAME$THERE = FALSE;
574  7                      END;
                    ELSE
575  6                      HAVE$IFRAME = FALSE;    /* HAVE AN S FRAME */
576  6                      END;
577  5      ELSE DO;
578  6          HAVE$IFRAME = FALSE;
579  6          FRAME$THERE = FALSE;
580  6          END;

```

```

585 5          INDEX = INDEX + FRAME$SIZE;
586 5          END;          /* DO WHILE FRAME$THIRD */
587 4          END;          /* CASE TABLE = 2 */

```

```

588 3          END;          /* CASE TABLE */
589 2          RETURN HAVE$FRAME;

```

```

590 2      END FIND$FRAME;

```

```

/*****

```

```

PROCEDURE    SERVICE        RESETS TRANSMIT 1 FRAME FLAGS AND COUNTERS

```

```

      THE PURPOSE OF THIS PROCEDURE IS TO RESET THE TRANSMIT 1 FRAME
      FLAGS AND COUNTERS AND CALL THE APPROPRIATE SRVC$TAB$H$K$P ROUTINE.

```

```

*****/

```

```

591 1      INTACK:    PROCEDURE(TABLE);

```

```

592 2          DECLARE TABLE BYTE;

```

```

593 2          IF (TABLE >= 3) AND (TABLE <= 4) THEN DO;

```

```

594 3              CALL SRVC$TAB$H$K$P(TABLE);

```

```

595 3              DO CASE TABLE;

```

```

597 4                  ;          /* ZEROth ENTRY IS NULL AND AN ERROR */

```

```

598 4                  ;          /* FIRST  ENTRY IS NULL AND AN ERROR */

```

```

599 4                  ;          /* SECOND ENTRY IS NULL AND AN ERROR */

```

```

600 4              DO;

```

```

601 5                  TIMCHA = TRUE;

```

```

602 5                  CTCONA = 0;

```

```

603 5                  RETRAN$A = 0;

```

```

604 5              END;

```

```

605 4              DO;

```

```

606 5                  TIMCHB = TRUE;

```

```

607 5                  CTCONB = 0;

```

```

608 5                  RETRAN$B = 0;

```

```

609 5              END;

```

```

610 4          END;          /* END CASE */

```

```

611 3      END;          /* END IF    */

```

```

612 2      END INTACK;

```

```

/*****

```

```

PROCEDURE    ROUTE$TH        ROUTES PACKETS TH FROM EITHER CHAN A OR CHAN B

```

```

      THE PURPOSE OF THIS PROCEDURE IS TO ROUTE PACKETS FROM THE NETWORK
      INPUT TABLES TO THEIR CORRECT OUTPUT TABLE. IT INITIATES THE BUILDING OF AN
      S FRAME FOR ACKNOWLEDGMENT OF GOOD 1 FRAMES. IT ALSO INSURES THE PROPER

```

SEQUENCING OF FRAMES BY THE USE OF MODULO 2 NUMBERING SCHEME.

INPUT

DATA PACKETS OR FRAMES ARE ROUTED VIA EVALUATION OF NTOTRX AND NTOZRX POINTERS AND FRAME HEADER ROUTING INFORMATION.

PROCESSING

THE PROCEDURE CHECKS NTOTRX AND NTOZRX POINTERS FOR FRAME ARRIVAL. IT THEN DETERMINES IF THE FRAME IS DESTINED FOR THIS UNIT OR ANOTHER UNIT. IF FOR ANOTHER UNIT, THE PROCEDURE SIMPLY ROUTES IT BACK TO THE NETWORK. IF THE INCOMING FRAME WAS ON CHAN A, THEN IT IS SUBSEQUENTLY TRANSMITTED OUT CHAN B. IF THE INCOMING FRAME WAS ON CHAN B, THEN IT IS ROUTED OUT CHAN A. THIS WILL CREATE A DUAL RING ON THE NETWORK SIDE OF THE UNITS. ONE DIRECTION THE FRAMES MOVE CLOCKWISE AND THE OTHER COUNTER CLOCKWISE.

IF THE INCOMING FRAME IS DESTINED FOR THIS PARTICULAR UNIT, THEN IT DETERMINES THE TYPE OF FRAME. IF IT IS AN I FRAME, THEN IT CALLS THE BUILD\$\$FRAME PROCEDURE TO PROVIDE A POSITIVE ACKNOWLEDGEMENT AND THEN MOVES THE FRAME TO THE NETWORK TO LOCAL TABLE MINUS THE FRAME HEADER. IF IT IS AN S FRAME, THEN IT TESTS TO SEE IF IT IS AN ACKNOWLEDGEMENT FOR ITS LAST TRANSMITTED I FRAME. THIS PROCEDURE USES PROCEDURE DET\$DEST\$ONE AND DET\$DEST\$TWO TO DETERMINE DESTINATION AND THE MOV\$ PROCEDURE TO MOVE THE FRAME BETWEEN TABLES.

OUTPUT

THIS PROCEDURE PRODUCES TWO OUTPUTS. THE FIRST IS A FRAME FROM THE NETWORK SIDE DESTINED FOR ONE OF THE OUTGOING LOCAL CHANNELS. THE SECOND OUTPUT IS A FRAME FROM THE NETWORK DESTINED FOR ANOTHER UNIT LOCATED ON THE NETWORK.

INTERFACE

THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY PROCEDURE MAIN.

NOTES: NONE.

*****/

```

613 1 ROUTE$IN: PROCEDURE RE-ENTRANT;
614 2     IF ((NTOINE - NTOINS) >= FRAME$SIZE) OR (NTOINS > NTOINE) THEN DO;
615 3         /* SERVICE CHAN A */
616 3         CALL SENDSEQ(@TP$5, LENGTH(IP$5));
617 3         IF (NTOTRX (NTOINS+1) AND 080H) = 080H THEN DO;
618 4             /* HAVE AN S FRAME */
619 4             CALL SENDSEQ(@TP$8, LENGTH(TP$8));
620 4             IF (NTOTRX (NTOINS+1) AND 020H) = 020H THEN DO;
621 5                 /* CHECKS SEQUENCE BIT */
622 5                 CALL SENDSEQ(@TP$9, LENGTH(TP$9));
623 5                 THIS$SEQ$BIT$A = TRUE;
624 5                 IF THIS$SEQ$BIT$A = SEQ$BIT$A THEN DO;
625 6                     SEQ$BIT$A = NOT SEQ$BIT$A;
626 6                     CALL INITACK(3);
627 6                     CALL SENDSEQ(@TP$8, LENGTH(TP$8));
628 6
629 6                 END;

```

```

630 5                   END;
631 4           ELSE DO;
632 5               CALL SENDSEQ(@TP$10, LENGTH(TP$10));
633 5               THIS$SEQ$BIT$A = FALSE;
634 5               IF THIS$SEQ$BIT$A = SEQ$BIT$A THEN DO;
635 6                   SEQ$BIT$A = NOT SEQ$BIT$A;
636 6                   CALL INITACK(3);
637 6                   CALL SENDSEQ(@TP$58, LENGTH(TP$58));
638 6               END;
639 6               END;       /* END IF XX = 20H */
640 5               END;       /* END IF XX = 80H */
641 4               END;

642 3           ELSE DO;               /* HAVE AN I FRAME */

643 4               CALL SENDSEQ(@TP$11, LENGTH(TP$11));
644 4               DESTINATION = DET$DEST$ONE;
645 4               IF DESTINATION = 'NN' THEN DO;
                      /* FRAME DESTINED FOR ANOTHER UNID */
647 5                   IF (NT01RX (NT01NS+1) AND 020H) = 020H THEN DO;
649 6                       CALL SENDSEQ(@TP$49, LENGTH(TP$49));
650 6                       INPUT$SEQ$BIT$A = TRUE;       /* SEQ$BIT$A = 1 */
651 6                       END;
652 5                   ELSE DO;
653 6                       CALL SENDSEQ(@TP$50, LENGTH(TP$50));
654 6                       INPUT$SEQ$BIT$A = FALSE;       /* SEQ$BIT$A = 0 */
655 6                       END;
656 5                   CALL BUILD$$FRAME(1, INPUT$SEQ$BIT$A);
657 5                   CALL SENDSEQ(@TP$6, LENGTH(TP$6));
658 5                   CALL MOVB( @NT01RX (NT01NS), @NT02TX (TX02NE),
                      FRAML$SIZE);
659 5                   CALL LD$TAB$H$SKP (4);
660 5                   END;       /* IF DEST = 'NN' */

661 4               IF DESTINATION = 'NL' THEN DO;
                      /* FRAME DESTINED FOR THIS UNID */
663 5                   CALL SENDSEQ(@TP$7, LENGTH(TP$7));
664 5                   IF (NT01RX (NT01NS+1) AND 020H) = 020H THEN DO;
666 6                       CALL SENDSEQ(@TP$12, LENGTH(TP$12));
667 6                       INPUT$SEQ$BIT$A = TRUE;
668 6                       END;
669 5                   ELSE DO;
670 6                       CALL SENDSEQ(@TP$13, LENGTH(TP$13));
671 6                       INPUT$SEQ$BIT$A = FALSE;
672 6                       END;
673 5                   CALL BUILD$$FRAME (1, INPUT$SEQ$BIT$A);
674 5                   CALL SENDSEQ(@TP$45, LENGTH(TP$45));
675 5                   CALL MOVB( @NT01RX (NT01NS+2), @NTLCFB (NTLCNE),
                      PACKET$SIZE);
676 5                   CALL LD$TAB$H$SKP (5);       /* HSKP 'NL' */
677 5                   END;       /* END OF DEST = 'NL' CONDITION */
678 4               END;       /* END OF I FRAME */

```

```

679 3      CALL SRVC$TAB$H$SKP (1);
680 3      END;          /* END OF IF (NT01NE - NT01NS) CONDITION */

681 2      IF ((NT02NE - NT02NS) >= FRAME$SIZE) OR (NT02NS > NT02NE) THEN DO;
                /* SERVICE CHAN B */
683 3      CALL SENDSEQ(@TP$14, LENGTH(TP$14));
684 3      IF (NT02RX (NT02NS+1) AND 080H) = 080H THEN DO;
                /* HAVE AN S FRAME */
686 4      CALL SENDSEQ(@TP$17, LENGTH(TP$17));
687 4      IF (NT02RX (NT02NS+1) AND 020H) = 020H THEN DO;
689 5      CALL SENDSEQ(@TP$18, LENGTH(TP$18));
690 5      THIS$SEQ$BIT$B = TRUE;
691 5      IF THIS$SEQ$BIT$B = SEQ$BIT$B THEN DO;
693 6      SEQ$BIT$B = NOT SEQ$BIT$B;
694 6      CALL INITACK(4);
695 6      CALL SENDSEQ(@TP$59, LENGTH(TP$59));
696 6      END;
697 5      END;
698 4      ELSE DO;
699 5      CALL SENDSEQ(@TP$19, LENGTH(TP$19));
700 5      THIS$SEQ$BIT$B = FALSE;
701 5      IF THIS$SEQ$BIT$B = SEQ$BIT$B THEN DO;
703 6      SEQ$BIT$B = NOT SEQ$BIT$B;
704 6      CALL INITACK(4);
705 6      CALL SENDSEQ(@TP$59, LENGTH(TP$59));
706 6      END;
707 5      END;          /* END 20H CONDITION */
708 4      END;          /* END 80H CONDITION */

709 3      ELSE DO;      /* HAVE AN I FRAME */

710 4      CALL SENDSEQ(@TP$20, LENGTH(TP$20));
711 4      DESTINATION = DET$DEST$TWO;
712 4      IF DESTINATION = 'NN' THEN DO;
                /* FRAME DESTINED FOR ANOTHER UNID */
714 5      IF (NT02RX (NT02NS+1) AND 020H) = 020H THEN DO;
716 6      CALL SENDSEQ(@TP$51, LENGTH(TP$51));
717 6      INPUT$SEQ$BIT$B = TRUE;
718 6      END;
719 5      ELSE DO;
720 6      CALL SENDSEQ(@TP$52, LENGTH(TP$52));
721 6      INPUT$SEQ$BIT$B = FALSE;
722 6      END;
723 5      CALL BUILD$$FRAME (2, INPUT$SEQ$BIT$B);
724 5      CALL SENDSEQ(@TP$15, LENGTH(TP$15));
725 5      CALL MOVB( @NT02RX (NT02NS), @NT01TX (TX01NE),
                FRAME$SIZE);  /* MOVE FRAME TO CH A OUTPUT TABLE */
726 5      CALL LD$TAB$H$SKP (5);
727 5      END;          /* END DEST = 'NN' */

728 4      IF DESTINATION = 'NL' THEN DO;  /* FRAME FOR THIS UNID */

```

```

730 5          CALL SENDSEQ(@TP$16, LENGTH(TP$16));
731 5          IF (NT02RX (NT02NS+1) AND 020H) = 020H THEN DO;
733 6              CALL SENDSEQ(@TP$21, LENGTH(TP$21));
734 6              INPUT$SEQ$BIT$B = TRUE;
735 6              END;
736 5          ELSE DO;
737 6              CALL SENDSEQ(@TP$22, LENGTH(TP$22));
738 6              INPUT$SEQ$BIT$B = FALSE;
739 6              END;     /* END 20H CONDITION */
740 5          CALL BUILD$$$FRAME (2, INPUT$SEQ$BIT$B);
741 5          CALL SENDSEQ(@TP$46, LENGTH(TP$46));
742 5          CALL MOV8( @NT02RX (NT02NS+2), @NTLC7B (NTLCNE),
                       PACKET$SIZE);
743 5          CALL LD$TAB$H$SKP (5);     /* DEST = NL */
744 5          END;     /* END OF DEST = NL CONDITION */
745 4          END;     /* END OF 1 FRAME */
746 3          CALL SRVC$TAB$H$SKP (2);
747 3          END;     /* END OF (NT02NE - NT02NS) CONDITION */

748 2          IF ((LCNTNE - LCNTNS) >= PACKET$SIZE) OR (LCNTNS > LCNTNE) THEN DO;
749 3              CALL SENDSEQ(@TP$23, LENGTH(TP$23));
750 3              DESTINATION = DET$DEST$LN;
751 3              IF DESTINATION = 'CA' THEN DO;     /* PACKET TO CHAN A */
752 3                  OUT$TAB$FULL = FIND$I$FRAME(1);
753 4                  IF (NOT OUT$TAB$FULL) THEN DO;
754 5                      CALL SENDSEQ(@TP$24, LENGTH(TP$24));
755 5                      CALL BUILD$I$FRAME(1);
756 5                      CALL SRVC$TAB$H$SKP (6);
757 5                      END;     /* IF NOT OUT$ */
758 4                  END;
759 3              IF DESTINATION = 'CB' THEN DO;     /* PACKET TO CHAN B */
760 3                  OUT$TAB$FULL = FIND$I$FRAME(2);
761 4                  IF (NOT OUT$TAB$FULL) THEN DO;
762 5                      CALL SENDSEQ(@TP$25, LENGTH(TP$25));
763 5                      CALL BUILD$I$FRAME(2);
764 5                      CALL SRVC$TAB$H$SKP (6);
765 5                      END;     /* IF NOT OUT$ */
766 4                  END;
767 3              END;     /* END OF (LCNTNE - LCNTNS) CONDITION */
768 2          END ROUTE$IN;

```

```

/*****
PROCEDURE    ROUTE$OUT    ROUTE FRAMES OUT TO NETWORK

```

THE PURPOSE OF THIS PROCEDURE IS TO ROUTE FRAMES FROM THE LOCAL TO NETWORK TABLE (LCNTTB) TO ONE OF THE OUTGOING NETWORK PORTS (NT01TX OR NT02TX) AND TO ROUTE FRAMES FROM THE NETWORK SIDE BACK ONTO THE NETWORK. IT ALSO MAINTAINS FLOW CONTROL AND ERROR RECOVERY BY WAITING FOR THE ROUTE\$IN PROCEDURE TO RECEIVE POSITIVE ACKNOWLEDGEMENT FOR 1 FRAMES TRANSMITTED BY THIS PROCEDURE.

INPUT

DATA FRAMES ARE ROUTED VIA EVALUATION OF NTO1TX, NTO2TX, AND LON1TB POINTERS WITH FRAME HEADER ADDRESS INFORMATION.

PROCESSING

THE PROCEDURE CHECKS EACH INPUT TABLE'S POINTERS FOR FRAMES TO BE TRANSMITTED ONTO THE NETWORK. BEFORE ANY TRANSMISSION OCCURS, MAX\$NETWORK\$CODE IS COMPARED AGAINST THE DESTINATION ADDRESS TO INSURE A FRAME CANNOT CONTINUE ON THE NETWORK WHEN IT HAS AN ADDRESS GREATER THAN THE AVAILABLE NUMBER OF UNIDS. NET TO NET FRAMES ARE SIMPLY TRANSFERRED TO THE NETWORK. ALL FRAMES WHICH ORIGINATE AT THIS UNID ARE CHECKED TO IDENTIFY THE TYPE. S FRAMES ARE SIMPLY TRANSMITTED. I FRAMES ARE TRANSMITTED AND ENTER A TIME DELAY WAITING FOR A GOOD ACKNOWLEDGEMENT. IF A GOOD ACKNOWLEDGEMENT IS NOT RECEIVED BY THE END OF THE WAIT PERIOD, IT IS TRANSMITTED AGAIN UPON THE NEXT CYCLE THROUGH THIS PROCEDURE.

OUTPUT

A FRAME OF DATA IS TRANSMITTED TO ONE OF THE NETWORK CHANNELS.

INTERFACE

THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY THE MAIN PROCEDURE.

NOTES:

1. THERE ARE TWO VERY IMPORTANT PARAMETERS RELATING TO THIS PROCEDURE THAT MUST BE CONSIDERED PRIOR TO THE MODIFICATION OF THIS PROCEDURE. THE FIRST IS VARIABLE MAX\$NETWORK\$CODE. ALL ACTIVE UNIDS ON THE NETWORK WILL START WITH 0 AND PROCEED TO INCREMENT IN A CLOCKWISE MANNER. IN THIS WAY, THE ADDRESS CAN BE COMPARED TO MAX\$NETWORK\$CODE TO BE SURE THAT NON-EXISTENT UNIDS WILL NOT APPEAR IN TRANSMITTED FRAMES.

2. THE SECOND PARAMETER IS MAXNUM. MAXNUM IS THE MAXIMUM NUMBER OF TIMES THE TIMEOUT CYCLE LOOPS THROUGH THE RETRANSMISSION PERIOD FOR AN I FRAME. THIS PERIOD ALLOWS TIME FOR A VALID ACKNOWLEDGEMENT TO BE RECEIVED AFTER THE I FRAME IS TRANSMITTED. THE BASIC PERIOD IS SET THROUGH N.INSTO\$02 FOR 27 MILLISECONDS. IF MAXNUM = 10, THEN THE WAIT LOOP IS 270 MILLISECONDS.

```

774 1 ROUTE$OUT: PROCEDURE REENTRANT;

                                     /* SERVICE UNID A */
775 2 IF ((TXOTNE - TXOTNS) >= FRAME$SIZE) OR
      (TXOTNS > TXOTNE) THEN DO;
777 3 CALL SENDSEQ(UTP$26, LENGTH(TP$26));
778 3 IF (NTO1TX(TXOTNS) AND 0FH)
      <= MAX$NETWORK$CODE THEN DO;
780 4 IF (NTO1TX(TXOTNS + 1) AND 80H) = 80H THEN DO;
782 5 CALL SERVICE$XMIT$A; /* IT'S AN S FRAME */
783 5 CALL INITACK(3);
784 5 CALL SENDSEQ(UTP$27, LENGTH(TP$27));
785 5 END;
786 4 ELSE DO;

```

```

787 5          IF TIMCHA = FALSE THEN
788 5              CALL TIME$DELAY$CHA;
789 5          ELSE DO;
790 6              CALL SENDSEQ(@TP$27, LENGTH(TP$27));
791 6              CALL SERVICE$XMIT$A;
792 6              RETRANS$A = RETRANS$A + 1;
793 6              CALL TIME$DELAY$CHA;
794 6          END;
795 5          END;      /* END NTOXTX      = 80H */
796 4          END;      /* END <= MAX$NETWORK$CODE THEN */
797 5      ELSE DO;
798 4          CALL SENDSEQ(@TP$48, LENGTH(TP$48));
799 4          STATTB (11) = STATTB (11) + 1;
800 4          STATTB (14) = STATTB (14) + 1;
801 4          CALL INITACK(3);
802 4          END;      /* END <= MAX$NETWORK$CODE */
803 5      IF RETRANS$A >= MAXRETRANS$A THEN DO;
805 4          CALL INITACK(3);
806 4          CALL SENDSEQ(@TP$56, LENGTH(TP$56));
807 4          END;
808 5      END;      /* END (TX01NE - TX01NS) CONDITION */

                                     /* SERVICE CHAN B */

809 2      IF ((TX02NE - TX02NS) >= FRAME$SIZE) OR
          (TX02NS > TX02NE) THEN DO;
811 3          CALL SENDSEQ(@TP$28, LENGTH(TP$28));
812 3          IF (NT02FX (TX02NS) AND OFH)
          <= MAX$NETWORK$CODE THEN DO;
814 4              IF (NT02TX(TX02NS + 1) AND 80H) = 80H THEN DO;
816 5                  CALL SERVICE$XMIT$B;      /* ITS AN S FRAME */
817 5                  CALL INITACK(4);
818 5                  CALL SENDSEQ(@TP$29, LENGTH(TP$29));
819 5                  END;
820 4              ELSE DO;
821 5                  IF TIMCHB = FALSE THEN
822 5                      CALL TIME$DELAY$CHB;
823 5                  ELSE DO;
824 6                      CALL SENDSEQ(@TP$29, LENGTH(TP$29));
825 6                      CALL SERVICE$XMIT$B;
826 6                      RETRANS$B = RETRANS$B + 1;
827 6                      CALL TIME$DELAY$CHB;
828 6                  END;
829 5                  END;      /* END NTOXTX      = 80H */
830 4                  END;      /* END <= MAX$NETWORK$CODE THEN */
831 3              ELSE DO;
832 4                  CALL SENDSEQ(@TP$47, LENGTH(TP$47));
833 4                  STATTB (11) = STATTB (11) + 1;
834 4                  STATTB (14) = STATTB (14) + 1;
835 4                  CALL INITACK(4);
836 4                  END;      /* END <= MAX$NETWORK$CODE */

```

```

837 3      IF RETRANS$B >= MAXRETRANS$B THEN DO;
839 4          CALL INITACK(4);
840 4          CALL SENDSEQ(@IP$57, LENGTH(IP$57));
841 4      END;
842 3      END; /* END (TX02NE - TX02NS) CONDITION */

843 2      END ROUTE$OUT;

/*****
PROCEDURE READ$LINE

PURPOSE
    READS A LINE OF DATA FROM THE HOST ISIS II CONSOLE, THEN INTERPRETED
    FOR EXECUTING EITHER LOAD ANOTHER TEST MESSAGE IN THE LCNTTB OR STOPPING
    THE TEST.
*****/

844 1      READ$LINE: PROCEDURE;
845 2          DECLARE INDEX WORD;
846 2          DECLARE LOOP WORD;

847 2          INDEX = 0;

848 2          CALL SENDSEQ(@MSG1, LENGTH(MSG1));
849 2          ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
850 2          CALL ERR$CHK;

851 2          IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN DO;
852 3              CALL SENDSEQ(@MSG4, LENGTH(MSG4));
853 3              ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
854 3              CALL ERR$CHK;
855 3              LOOP = ASC$HEX(BUFFER(0));
856 3              IF (LOOP >= 1) AND (LOW(LOOP) <= MAX$NETWORK$CODE) THEN DO;
857 4                  CALL SENDSEQ(@MSG3, LENGTH(MSG3));
858 4                  ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
859 4                  CALL ERR$CHK;
860 4                  IF (BUFFER(0) >= '1') AND (BUFFER(0) <= '9') THEN
861 5                      DO INDEX = 1 TO ASC$HEX(BUFFER(0));
862 5                      CALL LOAD(LOOP);
863 5                      END;
864 4                  END;
865 3              END;
866 2          END;

867 2          END;

868 2          CALL SENDSEQ(@MSG2, LENGTH(MSG2));
869 2          ACTUAL = DQ$READ(R$CONN, @BUFFER, 128, @STATUS);
870 2          CALL ERR$CHK;

871 2          IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN
872 2              FOREVER = FALSE;
873 2          ELSE
            FOREVER = TRUE;

```

```

374 2  END READ$LINE;
      /*****
          MAIN PROGRAM FOR THE NETWORK OPERATING SYSTEM

          THE PURPOSE OF THIS PROCEDURE IS TO PROVIDE THE MAIN LINE CODE OF
          PROCESSING FOR N.O.S.

          INPUT      NONE.

          PROCESSING
              THIS PROCEDURE SENDS A HEADER TO THE NETWORK MONITOR CONSOLE,
              INITIALIZES THE NETWORK TABLES VIA INITN$TAB, USES IN$IO TO INITIALIZE THE
              STD INTERRUPTS, AND LOOPS ENDLESSLY ROUTING FRAMES IN AND OUT VIA PROCEDURES
              ROUTE$IN AND ROUTE$OUT.

          OUTPUT
              A STARTUP MESSAGE IS SENT TO THE NETWORK MONITOR UPON STARTING.

          INTERFACE
              THIS PROCEDURE IS THE INITIAL ENTRY POINT FOR N.O.S. IT OPERATES THROUGH
              SINGLE CALLS TO CALL SENDSEQ, INITN$TAB, AND IN$IO, AND REPETITIVE CALLS TO
              ROUTE$IN AND ROUTE$OUT.

          NOTES:      NONE.
          *****/
375 1  BEGIN:

      /* THE FOLLOWING DQ$xxxxx CALLS MUST BE THE FIRST EXECUTABLE CODE IN
         THIS MODULE FOR THE ISIS-11 CONSOLE I/O TO OPERATE PROPERLY.
         DO NOT CHANGE THE LOCATION OF THIS CODE OR IT WILL NOT WORK!
         DO NOT MAKE 'BEGIN:' PUBLIC AND THE START(BEGIN) OPTION WITH
         LINK=6 OR IT WILL NOT WORK! */

      W$CONN = DQ$CREATE (0(4, '100:'), @STATUS);
376 1  CALL ENR$CHK;
377 1  CALL DQ$OPEN (W$CONN, 2, 0, @STATUS);
378 1  CALL ENR$CHK;

379 1  R$CONN = DQ$ATTACH (0(4, '101:'), @STATUS);
380 1  CALL ENR$CHK;
381 1  CALL DQ$OPEN (R$CONN, 1, 0, @STATUS);
382 1  CALL ENR$CHK;

383 1  CALL SENDSEQ(@STARTOP$HDR, LENGTH(STARTOP$HDR));
384 1  CALL SENDSEQ(@TP$1, LENGTH(TP$1));
385 1  CALL INITN$TAB;
386 1  CALL SENDSEQ(@TP$2, LENGTH(TP$2));
387 1  CALL READ$LINE;
388 1  CALL SENDSEQ(@TP$3, LENGTH(TP$3));
389 1  DO WHILE FOREVER;
390 2  CALL SENDSEQ(@TP$4, LENGTH(TP$4));

```

PL/M-86 COMPILER UNID-11 NETWORK TEST PROGRAM, 30 SEP 84

```
891 2          CALL ROUTE$IN;
892 2          CALL SENDSEQ(@TP$42, LENGTH(TP$42));
893 2          CALL ROUTE$OUT;
894 2          CALL READ$TAB;
895 2          CALL SENDSEQ(@TP$43, LENGTH(TP$43));
896 2          CALL READ$LINE;
897 2          END;
```

```
898 1          CALL SENDSEQ(@TP$44, LENGTH(TP$44));
899 1          CALL DQ$CLOSE(W$CONN, @STATUS);
900 1          CALL ERR$CHK;
901 1          CALL DQ$CLOSE(R$CONN, @STATUS);
902 1          CALL ERR$CHK;
903 1          CALL DQ$DETACH(W$CONN, @STATUS);
904 1          CALL ERR$CHK;
905 1          CALL DQ$DETACH(R$CONN, @STATUS);
906 1          CALL ERR$CHK;
907 1          CALL DQ$EXIT(0);
```

```
908 1          END MAIN;       /* END MAIN MODULE */
```

/****** THE END *****/

MODULE INFORMATION:

```
CODE AREA SIZE     = 1252H    4690D
CONSTANT AREA SIZE = 0093H    3165D
VARIABLE AREA SIZE = 2512H    8994D
MAXIMUM STACK SIZE = 0026H    36D
1529 LINES READ
0 PROGRAM ERROR(S)
```

END OF PL/M-86 COMPILATION

PL/M-80 COMPILER LOCAL NETWORK TEST PROGRAM, 30 SEP 1984

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT\$EMPTY\$BYTE ADDRESS BY ONE PACKET\$SIZE,
AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT\$EMPTY\$BYTE ADDRESS
ADVANCED BY THE LENGTH OF A SINGLE PACKET.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN
AND ROUTE\$OUT.

```
*****/
193 1  LD$TAB$H$K? : PROCEDURE(TABLE) ;
194 2      DECLARE TABLE ADDRESS;

195 2      IF (TABLE >= 1 AND TABLE <= 11) THEN /* 6 FOR REAL, 11 FOR SIM */
196 2          DO CASE TABLE;

197 3              ; /* CASE ZERO IS NULL */

198 3              DO; /* START CASE ONE */
199 4                  LCO1NE = LCO1NE + DATA$GRAM$SIZE;
/* ADVANCE NEXT$EMPTY ADDRESS */
200 4                  IF LCO1NE >= LCO1SZ THEN
201 4                      LCO1NE = 0;
202 4                  END;

203 3              DO; /* START CASE TWO */
204 4                  LCO2NE = LCO2NE + DATA$GRAM$SIZE;
205 4                  IF LCO2NE >= LCO2SZ THEN
206 4                      LCO2NE = 0;
207 4                  END;

208 3              DO;
209 4                  LCO3NE = LCO3NE + DATA$GRAM$SIZE;
210 4                  IF LCO3NE >= LCO3SZ THEN
211 4                      LCO3NE = 0;
212 4                  END;

213 3              DO;
214 4                  LCO4NE = LCO4NE + DATA$GRAM$SIZE;
215 4                  IF LCO4NE >= LCO4SZ THEN
216 4                      LCO4NE = 0;
217 4                  END;

218 3              ; /* START OF CASE FIVE */

219 3              DO; /* START OF CASE SIX */
220 4                  LONINE = LONINE + PACKET$SIZE;
221 4                  IF LONINE >= LONISZ THEN
222 4                      LONINE = 0;
223 4                  END;
```

```

        OUTPUT(PORT$1) = TRANS$STATE;
        OUTPUT(PORT$2) = TRANS$STATE;
        OUTPUT(PORT$3) = TRANS$STATE;
        OUTPUT(PORT$4) = TRANS$STATE;

    END;
ENABLE;

END TRNMIT$PKT;
*/
/*****
/* PROCEDURE  SNUSEQ  SENDS DATA TO LOCAL MONITOR FOR TESTING  */
/*
/*          THIS PROCEDURE TAKES A MESSAGE STRING AND OUTPUTS IT TO  */
/*          THE LOCAL MONITOR ATTACHED TO THE 86/12 CARD.  */
/*
/*          INPUT - A ADDRESS TO THE MESSAGE LOCATION IN MEMORY AND THE  */
/*          NUMBER OF BYTES TO BE SENT ARE INPUT.  */
/*          PROCESSING - THIS PROCEDURE CHECKS THE OUTPUT BUFFER STATUS  */
/*          IN A LOOP UNTIL THE BUFFER IS EMPTY. IT PLACES ONE  */
/*          BYTE AT A TIME IN THE OUTPUT USART UNTIL THE MESSAGE  */
/*          IS DONE.  */
/*          OUTPUT - MESSAGE TO THE USART CHANNEL.  */
/*          INTERFACE - THIS PROCEDURE IS CALLED BY THE FOLLOWING  */
/*          PROCEDURES: ROUTE$IN, ROUTE$OUT, AND MAIN.  */
*****/

180 1  ERR$CHK: PROCEDURE;
181 2      IF STATUS <> 0 THEN
182 2          DO;
183 3              CALL ERROR(STATUS);
184 3              CALL EXIT;
185 3          END;
186 2  END ERR$CHK;

187 1  SNUSEQ: PROCEDURE( MSG, TOTAL);
188 2      DECLARE MSG ADDRESS;
189 2      DECLARE TOTAL ADDRESS;

190 2      CALL WRITE( W$CONN, MSG, TOTAL, .STATUS);
191 2      CALL ERR$CHK;

192 2  END SNUSEQ;

/*****
PROCEDURE  LD$TAB$H$SKP  LOAD TABLE HOUSEKEEP

    THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED BUFFER
    TABLE AFTER LOADING OF THE USER DATA FROM THE HOST.

    INPUT - THE INPUT IS AN ADDRESS INDICATING THE TABLE REQUIRING
    CHANGES.

```

```

158 2      LC03SZ = DATA$TABLE$SIZE;

159 2      LC04NS = 0;
160 2      LC04NE = 0;
161 2      LC04SZ = DATA$TABLE$SIZE;

162 2      TX01NS = 0;
163 2      TX01NE = 0;
164 2      TX01SZ = DATA$TABLE$SIZE;

165 2      TX02NS = 0;
166 2      TX02NE = 0;
167 2      TX02SZ = DATA$TABLE$SIZE;

168 2      TX03NS = 0;
169 2      TX03NE = 0;
170 2      TX03SZ = DATA$TABLE$SIZE;

171 2      TX04NS = 0;
172 2      TX04NE = 0;
173 2      TX04SZ = DATA$TABLE$SIZE;

174 2      IX = 0;
175 2      DO WHILE IX < STAT$NBR;
176 3          STATB(IX) = 0;
177 3          IX = IX + 1;
178 3      END;

179 2      END INIT$STATB;

```

/******

PROCEDURE TRNMITS\$PKT TRANSMIT A PACKET

THE PURPOSE OF THIS PROCEDURE IS TO ENABLE THE DATA PORT FOR
PACKET TRANSMISSION.

INPUT - THE USART DATA PORT ADDRESS IS INPUT TO TRNMITS\$PKT.

PROCESSING - THE PROCEDURE ENABLES THE SELECTED USART AND SETS THE
INTERRUPT FOR THE TRANSMISSION OF A PACKET.

OUTPUT - THE INTERRUPT IS SET ON THE CHANNEL SPECIFIED BY PORT\$NUM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN.

/* THIS IS A NULL PROCEDURE FOR SIMULATION PURPOSES

TRNMITS\$PKT: PROCEDURE (PORT\$NUM);

DECLARE PORT\$NUM BYTE;

DISABLE;

DO CASE PORT\$NUM;

;

```

/* SET PIT #2, REGISTER #0 FOR USART #4 BAUD RATE */
/* INITIATE RESET ON ALL USARTS */
/* INITIALIZE THE FOUR USARTS TO: 2 STOP BITS, NO PARITY, */
/* 8 BIT CHARACTERS, AND 16X BAUD RATE */
/* INITIALIZE THE 8259 PROGRAMMABLE INTERRUPT CONTROLLER */

132 2      MCONUM = 0;
133 2      BYTES$SENT$1 = 0;
134 2      BYTES$SENT$2 = 0;
135 2      BYTES$SENT$3 = 0;
136 2      BYTES$SENT$4 = 0;
137 2      TRANS$1$RDY = TRUE;
138 2      TRANS$2$RDY = TRUE;
139 2      TRANS$3$RDY = TRUE;
140 2      TRANS$4$RDY = TRUE;

141 2      END INVTNT;

/******
/* PROCEDURE INIT$L$TAB INITIALIZES THE LOCAL TABLES */
/*
/* THIS PROCEDURE SETS THE LOCAL AND NETWORK TABLES TO THEIR
/* INITIAL VALUES PRIOR TO PROCESSING ANY MESSAGES.
/*
/*
/* INPUTS - NONE
/* PROCESSING - PUTS INITIAL VALUES IN THE LOCAL TABLES
/* OUTPUT - NONE
/* INTERFACE - CALLED BY THE MAIN PROCEDURE
/******
142 1      INIT$L$TAB: PROCEDURE;
143 2      DECLARE IX ADDRESS;

144 2      LCNTNS = 0;
145 2      LCNTNE = 0;
146 2      LCNTSZ = PACKET$TABLE$SIZE;

147 2      NTLONS = 0;
148 2      NTLONE = 0;
149 2      NTLCSZ = PACKET$TABLE$SIZE;

150 2      LCOINS = 0;
151 2      LCOINE = 0;
152 2      LCOISZ = DATA$TABLE$SIZE;

153 2      LCO2NS = 0;
154 2      LCO2NE = 0;
155 2      LCO2SZ = DATA$TABLE$SIZE;

156 2      LCO3NS = 0;
157 2      LCO3NE = 0;

```

```

111 2      DECLARE (VAL, 1) BYTE;

112 2      TEMP1(1) = ASCII(SHR(VAL,4) AND 0FH);
113 2      TEMP1(1+1) = ASCII(VAL AND 0FH);

114 2      END HEX2ASC;

115 1      ASC$HEX: PROCEDURE(C) BYTE;
116 2      DECLARE C BYTE;

117 2      IF (C >= '0' AND C <= '9') THEN
118 2          RETURN (C-50H);
119 2      ELSE
120 2          IF (C >= 'A' AND C <= 'F') THEN
121 2              RETURN (C-57H);
122 2          ELSE
123 2              RETURN C;
124 2      END ASC$HEX;

125 1      VALID$HEX: PROCEDURE(H) BYTE;
126 2      DECLARE (H,1) BYTE;

127 2      DO I = 0 TO LAST(ASCII);
128 3          IF H = ASCII(I) THEN
129 3              RETURN TRUE;
130 2      END;
131 2      RETURN FALSE;

132 2      END VALID$HEX;

/* ***** */
/* PROCEDURE INVT INITIALIZES THE HARDWARE PORTS */
/*
/* THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE VECTOR
/* TABLES AND ALL THE LOCAL NETWORK PORTS. THIS INCLUDES THE TIMERS,
/* INTERRUPT CONTROLLER, PARALLEL PORT ON THE ISBC 86/12, AND THE
/* USARTS.
/*
/* INPUT = NONE
/* PROCESSING = SETS THE VECTOR INTERRUPT TABLE VALUES AND WRITES THE
/* CONTROL WORD TO THE PIC, PIT, PPI, AND USARTS.
/* OUTPUT = NONE
/* INTERFACE = CALLED BY MAIN PROGRAM, CALLS DELAY PROCEDURE
/* ***** */
131 1      INVT: PROCEDURE;

/* INITIALIZE 8255 PROGRAMMABLE PERIPHERAL INTERFACE (PPI) */
/* SET BAUD RATES FOR LOCAL CHANNEL USARTS */
/* SET PIT #1, REGISTER #1 FOR USART #2 BAUD RATE */
/* SET PIT #1, REGISTER #2 FOR USART #3 BAUD RATE */

```

```

      'TP$55C      Reading test datagram from TX03TB');
93  1  DECLARE TP$55D(*) BYTE DATA(CR,LF,
      'TP$55D      Reading test datagram from TX04TB');
94  1  DECLARE TP$56A(*) BYTE DATA(CR,LF,
      'TP$56A      Reading test datagram from LC01TB');
95  1  DECLARE TP$56B(*) BYTE DATA(CR,LF,
      'TP$56B      Reading test datagram from LC02TB');
96  1  DECLARE TP$56C(*) BYTE DATA(CR,LF,
      'TP$56C      Reading test datagram from LC03TB');
97  1  DECLARE TP$56D(*) BYTE DATA(CR,LF,
      'TP$56D      Reading test datagram from LC04TB');

98  1  DECLARE MSG1(*) BYTE DATA(CR,LF,
      'Do you want to load the test message? ');
99  1  DECLARE MSG2(*) BYTE DATA(CR,LF,
      'Do you want to stop the test? ');
100 1  DECLARE MSG3(*) BYTE DATA(CR,LF,
      'Load into which host channel (1,2,3,4)? ');
101 1  DECLARE MSG4(*) BYTE DATA(CR,LF,
      'How many datagrams (0 - 9)? ');
102 1  DECLARE MSG5(*) BYTE DATA(CR,LF,
      'Destination network code (1,2,3) = ');
103 1  DECLARE MSG6(*) BYTE DATA(CR,LF,
      'Destination host code (0 - FFH) = ');

/*****
/*  PROCEDURE  DELAY      CAUSES A ONE SECOND DELAY      */
/*
/*  THE PURPOSE OF THIS PROCEDURE IS TO ADD DELAY TO PARTS OF THE
/*  INITIALIZATION PROCEDURE THAT IS TIME DEPENDENT.
/*
/*  INPUT - NONE
/*  PROCESSING - USES BUILT IN PROCEDURE IN A LOOP
/*  OUTPUT - NONE
/*  INTERFACE - CALLED BY INVINT
*****/

104 1  DELAY:PROCEDURE PUBLIC;
105 2  DECLARE I BYTE;
106 2  DO I=1 TO 40;
107 3  CALL TIME(250);
      /* TIME IS A BUILT IN FUNCTION OF PLM80 WHICH CAUSES */
      /* A DELAY BASED ON THE NUMBER IN PARENS      */
108 3  END;
109 2  END DELAY;

/*****
PROCEDURES TO CONVERT HEX TO ASCII FOR USE WITH DISPLAYING A FRAME
*****/

110 1  HEX$ASC: PROCEDURE(VAL, I);

```

```

      'TP$31          DATA IN NTLCB DESTINED FOR LOCAL CHANNEL-2');
68  1  DECLARE      TP$32(*) BYTE DATA(OR,LF,
      'TP$32          DATA IN NTLCB DESTINED FOR LOCAL CHANNEL-3');
69  1  DECLARE      TP$33(*) BYTE DATA(OR,LF,
      'TP$33          DATA IN NTLCB DESTINED FOR LOCAL CHANNEL-4');
70  1  DECLARE      TP$34(*) BYTE DATA(OR,LF,
      'TP$34          ERROR OCCURED IN NTLCB TO TXOXB OUT-PROCESSING');
71  1  DECLARE      TP$35(*) BYTE DATA(OR,LF,
      'TP$35          END OF ROUTE$IN-ROUTE$OUT LOOP');
72  1  DECLARE      TP$36(*) BYTE DATA(OR,LF,
      'TP$36          HAVE EXITED ROUTE$IN-ROUTE$OUT LOOP and returned to ISIS',OR,LF);

73  1  DECLARE      TP$40(*) BYTE DATA(OR,LF,
      'TP$40          Packet moved from LCNTB to NTLCB');
74  1  DECLARE      TP$50(*) BYTE DATA(OR,LF,
      'TP$50          Channel Number = ');
75  1  DECLARE      TP$51(*) BYTE DATA(OR,LF,
      'TP$51          Destination = ');
76  1  DECLARE      TP$51A(*) BYTE DATA(OR,LF,
      'TP$51A         Destination Network = ');
77  1  DECLARE      TP$51B(*) BYTE DATA(OR,LF,
      'TP$51B         Destination Control Code = ');
78  1  DECLARE      TP$51C(*) BYTE DATA(OR,LF,
      'TP$51C         Destination Country Code = ');
79  1  DECLARE      TP$51D(*) BYTE DATA(OR,LF,
      'TP$51D         Lohits = ');
80  1  DECLARE      TP$51E(*) BYTE DATA(OR,LF,
      'TP$51E         Hihits = ');
81  1  DECLARE      TP$52(*) BYTE DATA(OR,LF,
      'TP$52          Destination Address = ');
82  1  DECLARE      TP$52B(*) BYTE DATA(OR,LF,
      'TP$52B         Source Address = ');
83  1  DECLARE      TP$52A(*) BYTE DATA(OR,LF,
      'TP$52A         Destination Host = ');
84  1  DECLARE      TP$53(*) BYTE DATA(OR,LF,
      'TP$53          Source Address = ');
85  1  DECLARE      TP$53A(*) BYTE DATA(OR,LF,
      'TP$53A         Destination Network Code > Max Network');
86  1  DECLARE      TP$53AA(*) BYTE DATA(OR,LF,
      'TP$53AA        Source Network Code > Max Network');
87  1  DECLARE      TP$53B(*) BYTE DATA(OR,LF,
      'TP$53B         Country Code <> This Country - Sent to UNID 0');
88  1  DECLARE      TP$53C(*) BYTE DATA(OR,LF,
      'TP$53C         Control Code <> 0');
89  1  DECLARE      TP$54(*) BYTE DATA(OR,LF,
      'TP$54          Loaded test datagram in LCOTB');
90  1  DECLARE      TP$55A(*) BYTE DATA(OR,LF,
      'TP$55A         Reading test datagram from TX01B');
91  1  DECLARE      TP$55B(*) BYTE DATA(OR,LF,
      'TP$55B         Reading test datagram from TX02B');
92  1  DECLARE      TP$55C(*) BYTE DATA(OR,LF,

```

```

42 1  DECLARE      TP$6(*) BYTE DATA(CR,LF,
      'TP$6
      DATA LOCATED IN LOCAL CHANNEL-1' );
43 1  DECLARE      TP$7(*) BYTE DATA(CR,LF,
      'TP$7
      DATA IS LC01TB TO TX01TB TRANSFER' );
44 1  DECLARE      TP$8(*) BYTE DATA(CR,LF,
      'TP$8
      DATA IS LC01TB TO LCNTTB TRANSFER' );
45 1  DECLARE      TP$9(*) BYTE DATA(CR,LF,
      'TP$9
      ERROR OCCURED IN LOCAL CHANNEL-1 IN-PROCESSING' );
46 1  DECLARE      TP$10(*) BYTE DATA(CR,LF,
      'TP$10
      DATA LOCATED IN LOCAL CHANNEL-2' );
47 1  DECLARE      TP$11(*) BYTE DATA(CR,LF,
      'TP$11
      DATA IS LC02TB TO TX02TB TRANSFER' );
48 1  DECLARE      TP$12(*) BYTE DATA(CR,LF,
      'TP$12
      DATA IS LC02TB TO LCNTTB TRANSFER' );
49 1  DECLARE      TP$13(*) BYTE DATA(CR,LF,
      'TP$13
      ERROR OCCURED IN LOCAL CHANNEL-2 IN PROCESSING' );
50 1  DECLARE      TP$14(*) BYTE DATA(CR,LF,
      'TP$14
      DATA LOCATED IN LOCAL CHANNEL-3' );
51 1  DECLARE      TP$15(*) BYTE DATA(CR,LF,
      'TP$15
      DATA IS LC03TB TO TX03TB TRANSFER' );
52 1  DECLARE      TP$16(*) BYTE DATA(CR,LF,
      'TP$16
      DATA IS LC03TB TO LCNTTB TRANSFER' );
53 1  DECLARE      TP$17(*) BYTE DATA(CR,LF,
      'TP$17
      ERROR OCCURED IN LOCAL CHANNEL-3 IN PROCESSING' );
54 1  DECLARE      TP$18(*) BYTE DATA(CR,LF,
      'TP$18
      DATA LOCATED IN LOCAL CHANNEL-4' );
55 1  DECLARE      TP$19(*) BYTE DATA(CR,LF,
      'TP$19
      DATA IS LC04TB TO TX04TB TRANSFER' );
56 1  DECLARE      TP$20(*) BYTE DATA(CR,LF,
      'TP$20
      DATA IS LC04TB TO LCNTTB TRANSFER' );
57 1  DECLARE      TP$21(*) BYTE DATA(CR,LF,
      'TP$21
      ERROR OCCURED IN LOCAL CHANNEL-4 IN PROCESSING' );
58 1  DECLARE      TP$22(*) BYTE DATA(CR,LF,
      'TP$22
      ENTERING ROUTE$OUT PROCEDURE' );
59 1  DECLARE      TP$23(*) BYTE DATA(CR,LF,
      'TP$23
      OUTGOING DATA IS IS LC05TB' );
60 1  DECLARE      TP$24(*) BYTE DATA(CR,LF,
      'TP$24
      DATA IN LC05TB DESTINED FOR LOCAL CHANNEL-1' );
61 1  DECLARE      TP$25(*) BYTE DATA(CR,LF,
      'TP$25
      DATA IN LC05TB DESTINED FOR LOCAL CHANNEL-2' );
62 1  DECLARE      TP$26(*) BYTE DATA(CR,LF,
      'TP$26
      DATA IN LC05TB DESTINED FOR LOCAL CHANNEL-3' );
63 1  DECLARE      TP$27(*) BYTE DATA(CR,LF,
      'TP$27
      DATA IN LC05TB DESTINED FOR LOCAL CHANNEL-4' );
64 1  DECLARE      TP$28(*) BYTE DATA(CR,LF,
      'TP$28
      ERROR OCCURED IN LC05TB OUT$PROCESSING' );
65 1  DECLARE      TP$29(*) BYTE DATA(CR,LF,
      'TP$29
      OUTGOING DATA IS IN NL05TB' );
66 1  DECLARE      TP$30(*) BYTE DATA(CR,LF,
      'TP$30
      DATA IN NL05TB DESTINED FOR LOCAL CHANNEL-1' );
67 1  DECLARE      TP$31(*) BYTE DATA(CR,LF,

```

STATTB (STAT\$NBR) BYTE;

/* DECLARATIONS FOR FLAGS AND TABLES IN SYSTEM MEMORY */

```

32 1 DECLARE
      INTERRUPT    BYTE,          /* 88/45 INTERRUPT BYTE */
      (SPARE1,SPARE2,SPARE3) BYTE, /* SPARE LOCATIONS */
      (TO$HOST, TO$HOST$BUSY) BYTE, /* BOOLEAN FLAGS */
      (TCHOSTNS, TCHOSTNE) ADDRESS, /* TABLE ADDRESS$S */
      NTLCIB (PACKET$TABLE$SIZE) BYTE, /* TABLE OF 10 PACKETS */

      (TO$NET, TO$NET$BUSY) BYTE,    /* BOOLEAN FLAGS */
      (TONETNS, TONETNE) ADDRESS,    /* TABLE ADDRESS$S */
      LCHNTB (PACKET$TABLE$SIZE) BYTE; /* TABLE OF 10 PACKETS */

```

/* MISCELLANEOUS DECLARATIONS */

```

33 1 DECLARE FOREVER BYTE;
34 1 DECLARE
      BUSY      LITERALLY 'OFFH',
      TRUE      LITERALLY 'OFFH',
      FALSE     LITERALLY '00H',
      NMBR$MSK  LITERALLY '07H',
      CR        LITERALLY '0DH',
      LF        LITERALLY '0AH',
      SOURCE    LITERALLY '12',
      DESTIN    LITERALLY '16',
      ESC       LITERALLY '1BH';

```

```

35 1 DECLARE
      DESTINATION ADDRESS, /* DESTINATION OF THE PACKET */
      DESTINATION$ADDRESS BYTE, /* DESTIN ADDR OF DATAGRAM */
      SOURCE$ADDRESS BYTE; /* SOURCE ADDR OF DATAGRAM */

```

/* INTERNAL VARIABLES USED IN THIS MODULE */

```

36 1 DECLARE
      STARTUP$HDR(*) BYTE DATA(CR,LF,
      '
      UNID 11 #2 LOCAL OS',CR,LF,
      '
      VERS 1.1, 30 SEP 84',CR,LF,
      '
      EXECUTING ',CR,LF);

```

/* THE FOLLOWING TEST POINTS ARE USED TO FOLLOW THE DATA WITHIN THE UNID */

```

37 1 DECLARE
      TP$1(*) BYTE DATA(CR,LF,
      'TP$1
      ENTERING INIT$LS$TAB PROCEDURE');
38 1 DECLARE
      TP$2(*) BYTE DATA(CR,LF,
      'TP$2
      ENTERING INIT$US$HTAB PROCEDURE');
39 1 DECLARE
      TP$3(*) BYTE DATA(CR,LF,
      'TP$3
      ENTERING INVINT PROCEDURE');
40 1 DECLARE
      TP$4(*) BYTE DATA(CR,LF,
      'TP$4
      STARTING ROUTE$IN-ROUTE$OUT LOOP' );
41 1 DECLARE
      TP$5(*) BYTE DATA(CR,LF,
      'TP$5
      ENTERING ROUTE$IN PROCEDURE' );

```

```

/* DATA TABLES USED IN THIS PROGRAM */
/*****

```

```

31 1  DECLARE  LC01TB (DATA$TABLE$SIZE) BYTE,
                LC01NS ADDRESS,
                LC01NE ADDRESS,
                LC01SZ ADDRESS,

                LC02TB (DATA$TABLE$SIZE) BYTE,
                LC02NS ADDRESS,
                LC02NE ADDRESS,
                LC02SZ ADDRESS,

                LC03TB (DATA$TABLE$SIZE) BYTE,
                LC03NS ADDRESS,
                LC03NE ADDRESS,
                LC03SZ ADDRESS,

                LC04TB (DATA$TABLE$SIZE) BYTE,
                LC04NS ADDRESS,
                LC04NE ADDRESS,
                LC04SZ ADDRESS,

                TX01TB(DATA$TABLE$SIZE) BYTE,
                TX01NS ADDRESS,
                TX01NE ADDRESS,
                TX01SZ ADDRESS,

                TX02TB(DATA$TABLE$SIZE) BYTE,
                TX02NS ADDRESS,
                TX02NE ADDRESS,
                TX02SZ ADDRESS,

                TX03TB(DATA$TABLE$SIZE) BYTE,
                TX03NS ADDRESS,
                TX03NE ADDRESS,
                TX03SZ ADDRESS,

                TX04TB(DATA$TABLE$SIZE) BYTE,
                TX04NS ADDRESS,
                TX04NE ADDRESS,
                TX04SZ ADDRESS,

                LCNTNS ADDRESS,
                LCNTNE ADDRESS,
                LCNTSZ ADDRESS,

                NTLCNS ADDRESS,
                NTLCNE ADDRESS,
                NTLCNZ ADDRESS,

```

```

/* CC WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '01',
/* INDICATES COUNTRY$CODES IN USE */
MAX$NETWORK$CODE LITERALLY '03';
/* INDICATES UNIDS OPERATIONAL IN NET */

```

```

/*****
/* DEFINITIONS FOR THE LOCAL SERIAL INPUT/OUTPUT CARD AND 86/12 PPI */
*****/

```

```

25 1 DECLARE CNTRL$8255 LITERALLY '0CEH', /* 8255 CONTROL PORT ON 86/12 */
ASIN LITERALLY '0CBH', /* I/O INPUT PORT ADDRESS */
B$OUT LITERALLY '0CAH', /* I/O OUTPUT PORT ADDRESS */
C$CNTRL LITERALLY '0CCH', /* I/O CONTROL PORT ADDRESS */
ICW1$OR$OCW2 LITERALLY '0COH', /* PORT ADDRESS FOR PIC */
ICWS LITERALLY '0C2H', /* PORT ADDRESS FOR PIC MODE */
RCV$STATE LITERALLY '00010110B', /* MODE INSTRUCTION FOR */
/* 8251 USART RCV INT */
TRANS$STATE LITERALLY '00110111B', /* MODE INSTRUCTION FOR */
/* 8251 RCV & XMIT INT */
ROTATE$PRIORITY$SET LITERALLY '10100000B';
/* ROTATES PRIORITY */
/* FOR EQUAL PRIORITY TO */
/* ALL I/O PORTS */

```

```

/*****
/* ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM */
*****/

```

```

26 1 DECLARE BAUD$LSB BYTE,
BAUD$MSB BYTE,
NUM$BYTES$SENT ADDRESS,
TRANS$1$RDY BYTE,
TRANS$2$RDY BYTE,
TRANS$3$RDY BYTE,
TRANS$4$RDY BYTE,
MSGNUM BYTE;

```

```

27 1 DECLARE TEMP1(*) BYTE INITIAL('0000000000000000'),
TEMP2(*) BYTE INITIAL('0000000000000000');

```

```

28 1 DECLARE BYTES$SENT$1 BYTE,
BYTES$SENT$2 BYTE,
BYTES$SENT$3 BYTE,
BYTES$SENT$4 BYTE;

```

```

29 1 DECLARE J BYTE;

```

```

30 1 DECLARE HSNP$ERR ADDRESS;

```

```

/*****

```

```

9 2      END EXIT;

10 1      ERROR: PROCEDURE (ERRNUM) EXTERNAL;
11 2      DECLARE ERRNUM ADDRESS;
12 2      END ERROR;

13 1      DECLARE ACTUAL ADDRESS;
14 1      DECLARE STATUS ADDRESS;
15 1      DECLARE BUFFER (128) BYTE;
16 1      DECLARE R$CONN LITERALLY '1';
17 1      DECLARE W$CONN LITERALLY '0';
18 1      DECLARE ERRNUM ADDRESS;
19 1      DECLARE CRLF(*) BYTE DATA(ODH, OAH);
20 1      DECLARE MESSAGE(*) BYTE DATA(ODH, OAH,
        'THIS IS THE TEST MESSAGE'          THIS IS THE TEST
        MESSAGE!!!!!!);

21 1      DECLARE ASCII(*) BYTE DATA('0123456789ABCDEF');
22 1      DECLARE CHAN$NUM ADDRESS;

23 1      DECLARE
        L$R1$DEST$ERR LITERALLY '000H', /* LOCAL ROUTE$IN ERROR */
        L$R0$DEST$ERR LITERALLY '01H'; /* LOCAL ROUTE$OUT ERROR */

        /* THE FOLLOWING ARE UNID DEFINED VARIABLES */
        /* NOTE: THESE VARIABLES MAY CHANGE DEPENDING ON THE
        SOFTWARE CONFIGURATION USED WITHIN THE DELNET */

24 1      DECLARE DATA$GRAM$SIZE LITERALLY '128', /* NUMBER OF BYTES FROM HOST */
        PACKET$SIZE LITERALLY '133', /* DATA PACKET + HEADER */
        PACKET$IN$TABLE LITERALLY '10',
        STAT$NBR LITERALLY '20', /* STATUS ENTRIES IN STATTB */
        DATA$TABLE$SIZE LITERALLY '1280', /* NUMBR OF BYTES */
        PACKET$TABLE$SIZE LITERALLY '1350', /* NUMBER OF BYTES */
        TCP$DATA$SIZE LITERALLY '72', /* TCP DATA SIZE */

        /* FOLLOWING ARE NETWORK DEFINED VARIABLES */
        /* NOTES: 1. THIS$UNID$NBR MUST REFLECT WHICH UNID THIS IS.
        2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH
        THIS UNID IS LOCATED.
        3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
        ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR
        THE DELNET MONITOR.
        4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNIDS ARE
        CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
        5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO
        PHISTER'S THESIS, APPENDIX D. */

        THIS$UNID$NBR LITERALLY '02',
        /* UNIQUE ADDRESS FOR THIS UNID */
        THIS$COUNTRY$CODE LITERALLY '01',

```

ISIS-II PL/M-B0 V3.0 COMPILATION OF MODULE MAIN
 OBJECT MODULE PLACED IN NEWL00.OBJ
 COMPILER INVOKED BY: PLMB0 NEWL00.SRC

\$TITLE('LOCAL NETWORK TEST PROGRAM, 30 SEP 1984')

```

/*****/
/*                                     */
/* DATE:          30 Sep 1984         */
/* VERSION:       1.0                 */
/*                                     */
/* TITLE:         Network Layer Simulation */
/* FILE NAME:     NEWL00.SRC          */
/* OWNER:        C. T. Childress      */
/*                                     */
/* SOFTWARE SYSTEM: Intel System III, ISIS II (V 4.2, 4.3) */
/* USE:          Simulate the function of the network */
/*              layer software on the System III */
/*                                     */
/* CONTENTS:      Source code listing  */
/*                                     */
/*****/

```

```

/*****
PROLOGUE - MODULE L.MAIN$U1          DATE TRANSLATED: 1 SEP 85
                                     DATE LAST MODIFIED: 30 SEP 84

```

THE PURPOSE OF THIS MODULE IS TO PROVIDE THE UNID LOCAL
 OPERATING SYSTEM (L.OS) WITH THE MAIN LINE OF PROCESSING. THE L.OS
 IS REQUIRED TO INPUT AND OUTPUT DATA FROM EITHER THE FOUR LOCAL
 CHANNELS OR THE TWO NETWORK CHANNELS. THIS MODULE SUPPORTS THE
 X.25, NETWORK LAYER, SOFTWARE, AND THE INTERNET PROTOCOL (IP)
 INTERACTION WITH THE HOSTS.

THIS MODULE CONSISTS OF THE MAIN LINE PROCEDURE L.MAIN\$U1,
 AND SUBORDINATE PROCEDURES BUILD\$IP\$PACKET, DET\$ADDR, LD\$TAB\$H\$SKP,
 SRVC\$TAB\$H\$SKP, TRNMIT\$PKT, ROUTE\$IN, AND ROUTE OUT.

ALL OF THESE ROUTINES ARE CONVERTED FROM THE PL/Z PROGRAMS
 ORIGINALLY WRITTEN BY CAPT PAUL PHISTER FOR THE Z-80 UNIDS.

*****/

```

1  MAIN: DO;

2  1  READ:  PROCEDURE (AFTN, BUFFER, COUNT, ACTUAL, STATUS) EXTERNAL;
3  2      DECLARE (AFTN, BUFFER, COUNT, ACTUAL, STATUS) ADDRESS;
4  2  END READ;

5  1  WRITE: PROCEDURE (AFTN, BUFFER, COUNT, STATUS) EXTERNAL;
6  2      DECLARE (AFTN, BUFFER, COUNT, STATUS) ADDRESS;
7  2  END WRITE;

8  1  EXIT:  PROCEDURE EXTERNAL;

```

2. Network Layer Simulation

The purpose of this program is to simulate the network layer software on the Intel System III software development system.

```

224 3      DO;                /* CASE SEVEN IS NULL */
225 4          NTLONE = NTLONE + PACKET$SIZE;
226 4          IF NTLONE >= NTLCSZ THEN
227 4              NTLONE = 0;
228 4      END;

229 3      DO;
230 4          TX01NE = TX01NE + DATA$GRAM$SIZE;
231 4          IF TX01NE >= TX01SZ THEN
232 4              TX01NE = 0;
233 4      END;

234 3      DO;
235 4          TX02NE = TX02NE + DATA$GRAM$SIZE;
236 4          IF TX02NE >= TX02SZ THEN
237 4              TX02NE = 0;
238 4      END;

239 3      DO;
240 4          TX03NE = TX03NE + DATA$GRAM$SIZE;
241 4          IF TX03NE >= TX03SZ THEN
242 4              TX03NE = 0;
243 4      END;

244 3      DO;
245 4          TX04NE = TX04NE + DATA$GRAM$SIZE;
246 4          IF TX04NE >= TX04SZ THEN
247 4              TX04NE = 0;
248 4      END;

249 3      END;

250 2      ELSE HSKP$ERR = HSKP$ERR + 1;

251 2      END LD$TAB$HSKP;

```

/*****

PROCEDURE SRVC\$TAB\$HSKP SERVICE TABLE HOUSEKEEP

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED
BUFFER TABLE AFTER SERVICING (REMOVING A PACKET).

INPUT - THE INPUT IS AN ADDRESS VALUE INDICATING THE TABLE THAT
REQUIRES HOUSEKEEPING.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT\$BYTE\$TO\$BE\$SERVICED ADDRESS BY A PACKET\$SIZE,
AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT\$BYTE\$TO\$BE\$SERVICED ADDRESS
ADVANCED BY THE LENGTH OF A SINGLE PACKET.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN
AND ROUTE\$OUT.

```

252 1  SRV0$TAB$H$K$P: PROCEDURE(TAB) ;
253 2      DECLARE TAB ADDRESS;

254 2      IF (TAB >= 1 AND TAB <= 11) THEN /* 7 FOR REAL, 11 FOR SIM */
255 2          DO CASE TAB;

256 3              ; /* CASE ZERO IS NULL */

257 3              DO; /* CASE ONE STARTS HERE */
258 4                  LC01NS = LC01NS + DATA$GRAM$SIZE;
259 4                  IF LC01NS >= LC01SZ THEN
260 4                      LC01NS = 0;
261 4                  END;

262 3              DO; /* CASE TWO STARTS HERE */
263 4                  LC02NS = LC02NS + DATA$GRAM$SIZE;
264 4                  IF LC02NS >= LC02SZ THEN
265 4                      LC02NS = 0;
266 4                  END;

267 3              DO;
268 4                  LC03NS = LC03NS + DATA$GRAM$SIZE;
269 4                  IF LC03NS >= LC03SZ THEN
270 4                      LC03NS = 0;
271 4                  END;

272 3              DO; /* CASE FOUR STARTS HERE */
273 4                  LC04NS = LC04NS + DATA$GRAM$SIZE;
274 4                  IF LC04NS >= LC04SZ THEN
275 4                      LC04NS = 0;
276 4                  END;

277 3              ; /* CASE FIVE STARTS HERE */

278 3              DO; /* CASE SIX IS A NULL STATEMENT */
279 4                  LCNTNS = LCNTNS + PACKET$SIZE;
280 4                  IF LCNTNS >= LCNTSZ THEN
281 4                      LCNTNS = 0;
282 4                  END;

283 3              DO; /* CASE SEVEN STARTS HERE */
284 4                  NTLONS = NTLONS + PACKET$SIZE;
285 4                  IF NTLONS >= NTLCSZ THEN
286 4                      NTLONS = 0;
287 4                  END;

                /* CASES 8, 9, 10, 11 FOR TX01(2,3,4) ARE NOT REQUIRED AS */

```

```

/* THE TX0XNS POINTERS ARE INCREMENTED AND CHECKED IN THE */
/* TRANSMIT DATAGRAM ROUTINES */

```

```

288 3      DO;
289 4          TX01NS = TX01NS + DATA$GRAM$SIZE;
290 4          IF TX01NS >= TX01SZ THEN
291 4              TX01NS = 0;
292 4      END;

293 3      DO;
294 4          TX02NS = TX02NS + DATA$GRAM$SIZE;
295 4          IF TX02NS >= TX02SZ THEN
296 4              TX02NS = 0;
297 4      END;

298 3      DO;
299 4          TX03NS = TX03NS + DATA$GRAM$SIZE;
300 4          IF TX03NS >= TX03SZ THEN
301 4              TX03NS = 0;
302 4      END;

303 3      DO;
304 4          TX04NS = TX04NS + DATA$GRAM$SIZE;
305 4          IF TX04NS >= TX04SZ THEN
306 4              TX04NS = 0;
307 4      END;

308 3      END;

309 2      ELSE HSKP$ERR = HSKP$ERR + 1;
310 2      END SRVC$TAB$HSKP;

```

```

/*****
/* PROCEDURE BUILD$I$PACKET PROCEDURE FOR TRANSFORMING THE USER DATA */
/* INTO A DATA$PACKET FOR TRANSFER TO THE */
/* NETWORK SIDE OF THE UNID. */
/*
/* THE PURPOSE OF THIS PROCEDURE IS TO TRANSFORM THE HOST'S*/
/* USER DATA DELIVERED TO ONE OF THE LOCAL INPUT BUFFERS INTO A */
/* DATA$PACKET TO BE PLACED IN THE LOCAL TO NETWORK BUFFER. THIS*/
/* PROCEDURE ADDS THE FIVE HEADER BYTES, AS FOLLOWS: */
/* 1. 1 BYTE FOR THE DESTINATION ADDRESS. */
/* 2. 1 BYTE FOR THE SOURCE ADDRESS. */
/* 3. 1 BYTE FOR THE SEQUENCE NUMBER. */
/* 4. 1 BYTE FOR A SPARE (SPARE$01). */
/* 5. 1 BYTE FOR A SPARE (SPARE$02). */
/*
/* INPUT - THIS PROCEDURE RECEIVES AN ADDRESS THAT INDICATES THE */
/* LOCAL INPUT BUFFER WHERE THE INCOMING HOST DATA IS */
/* LOCATED. */
/* PROCESSING - THE PROCEDURE BEGINS WITH THE PASSING OF THE TABLE*/

```

```

/*          WHERE THE HOST'S DATA IS LOCATED. THE SOURCE AND          */
/*          DESTINATION ADDRESS (SOURCE$ADDRESS, DESTINATION$        */
/*          ADDRESS)                                                  */
/*          IS SUPPLIED BY THE DET$ADDR PROCEDURE. FOR NOW, THE      */
/*          SEQUENCE NUMBER AND BOTH SPARE BYTES ARE SET TO ZERO.    */
/*          THE FUTURE, THESE BYTES WILL REFLECT X.25 PROTOCOL.      */
/*          */
/*          OUTPUT - THIS PROCEDURE PLACES THE FIRST FIVE BYTES INTO THE */
/*          LOCAL TO NETWORK BUFFER TABLE BEFORE THE PACKET IS     */
/*          TRANSFERRED OVER TO THE NETWORK SIDE OF THE UNID.        */
/*          */
/*          INTERFACE - THE PROCEDURE IS CALLED FROM PROCEDURE ROUTE$IN FOR*/
/*          THOSE DATA PACKETS DESTINED FOR THE NETWORK ONLY.      */
/*          */
/*          NOTE: THE HEADER INFORMATION SUPPLIED IS FOR DATAGRAM SERVICE */
/*          ONLY. ALSO, THE SEQUENCE AND SPARE BYTES ARE SET TO ZERO*/
/*          TO ALLOW THE UNID'S MINIMAL OPERATIONAL CAPABILITY. FOR */
/*          FUTURE SOFTWARE ENHANCEMENTS THIS MUST BE MODIFIED.     */
/*          */
/*****
311  1  BUILD$1$PACKET: PROCEDURE(TABLE$PTR) ;
312  2          DECLARE TABLE$PTR ADDRESS,
                LCOXTB BASED TABLE$PTR(1) BYTE;

313  2          LONTTB(LONTNE + 0) = DESTINATION$ADDRESS;
314  2          LONTTB(LONTNE + 1) = SOURCE$ADDRESS;
315  2          LONTTB(LONTNE + 2) = 0;
316  2          LONTTB(LONTNE + 3) = 0;
317  2          LONTTB(LONTNE + 4) = 0;

318  2          CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .LONTTB(LONTNE+5));

319  2          CALL LD$TAB$H$K$P(6);

320  2  END BUILD$1$PACKET;

/*****
PROCEDURE DET$ADDR DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
COMING FROM A HOST CONNECTED TO LOCAL CHANNEL-1.

INPUT - THE INPUT IS A TWO CHARACTER ASCII VALUE INDICATING THE TABLE
LOCATION OF THE PACKET TO BE EVALUATED AND A LOCATION FOR THE
DESTINATION TO BE PLACED..

PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL$CODE FROM THE
INCOMING DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE
COUNTRY$CODE AND NETWORK$CODE ARE THEN EXTRACTED TO DETERMINE
THE DATA'S DESTINATION. IF THE DATA IS DESTINED FOR THE NETWORK
SIDE OF THE UNID THEN THE HOST$CODE IS ALSO EXTRACTED SO THAT
THE DESTINATION$ADDRESS CAN BE DETERMINED.

```

OUTPUT - THIS PROCEDURE PLACES THE DESTINATION ('LN' OR 'LL') IN MEMORY FOR THE PASSING ROUTINE AND IT RETURNS THE DESTINATION\$ ADDRESS. AN EXAMPLE OF A DESTINATION\$ ADDRESS IS 21, WHICH INDICATES UNID=2 AND CHANNEL=1.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN.

NOTES: 1. BYTE AND 0FH WILL MASK OUT THE UPPER 4-BITS.
2. BYTE AND 0F0H WILL MASK OUT THE LOWER 4-BITS.
3. BYTE / 01H WILL RETURN A VALUE 0-15 FROM THE LOWER 4-BITS.
4. BYTE / 10H WILL RETURN A VALUE 0-15 FROM THE UPPER 4-BITS.

*****/

```

321 1  DET$ADDR: PROCEDURE(TABLE$PTR);

322 2  DECLARE LOBITS BYTE,
        HIBITS BYTE,
        CONTROL$CODE BYTE,
        COUNTRY$CODE BYTE,
        NETWORK$CODE BYTE,
        HOST$CODE BYTE,
        SRC$HOST$CODE BYTE,
        SRC$NET$CODE BYTE,
        SRC$CONT$CODE BYTE,
        SRC$CONTRY$CODE BYTE,
        TABLE$PTR ADDRESS,
        LCOXTB BASED TABLE$PTR(1) BYTE;

323 2  LOBITS = 0;
324 2  HIBITS = 0;
325 2  CONTROL$CODE = 0;
326 2  COUNTRY$CODE = 0;
327 2  NETWORK$CODE = 0;
328 2  HOST$CODE = 0;
329 2  DESTINATION$ADDRESS = 0;
330 2  SOURCE$ADDRESS = 0;
331 2  DESTINATION = 8;

332 2  CONTROL$CODE = LCOXTB(16) AND 0FH;
333 2  IF CONTROL$CODE = 00 THEN
334 2  DO;
335 3  COUNTRY$CODE = (ROR(LCOXTB(16), 4) AND 0FH);
336 3  IF COUNTRY$CODE = THIS$COUNTRY$CODE THEN
337 3  DO;
338 4  NETWORK$CODE = LCOXTB(17) AND 0FH;
339 4  IF (NETWORK$CODE <= MAX$NETWORK$CODE) THEN
340 4  DO;
341 5  IF (COUNTRY$CODE <> THIS$COUNTRY$CODE) OR
        ((NETWORK$CODE)<> THIS$UNID$NBR) THEN
342 5  DESTINATION = 6; /* LOC TO NET */
        ELSE

```

```

343 5          DESTINATION = 5;          /* LOC TO LOC */
344 5          LOBITS = (ROR(LC0XTB(17), 4) AND 0FH);
345 5          HIBITS = (ROL(LC0XTB(18), 4) AND 0FH);
346 5          HOST$CODE = LOBITS OR HIBITS;
347 5          IF (HOST$CODE >= 0) AND (HOST$CODE <= 63) THEN
348 5              DESTINATION$ADDRESS = NETWORK$CODE OR 10H;
          ELSE
349 5              IF (HOST$CODE >= 64) AND (HOST$CODE <= 127) THEN
350 5                  DESTINATION$ADDRESS = NETWORK$CODE OR 20H;
          ELSE
351 5              IF (HOST$CODE >= 128) AND (HOST$CODE <= 191) THEN
352 5                  DESTINATION$ADDRESS = NETWORK$CODE OR 30H;
          ELSE
353 5              IF (HOST$CODE >= 192) AND (HOST$CODE <= 255) THEN
354 5                  DESTINATION$ADDRESS = NETWORK$CODE OR 40H;
          END;
          ELSE
356 4              DO;
357 5                  CALL SNDSEQ(.TP$53A, SIZE(TP$53A));
358 5                  STATB(04) = STATB(04) + 1;
359 5                  STATB(00) = STATB(00) + 1;
360 5              END;
361 4              END;
          ELSE
362 5              DO;          /* NOT FOR THIS COUNTRY, SEND TO NETWORK/UNIT 0 */
363 4                  CALL SNDSEQ(.TP$53B, SIZE(TP$53B));
364 4                  STATB(03) = STATB(03) + 1;
365 4                  STATB(00) = STATB(00) + 1;
366 4              END;
367 5              END;
          ELSE
368 2              DO;
          /* IT IS AT THIS POINT THAT OTHER IP CONTROL CODES WILL
          BE INCORPORATED INTO THE NETWORK */
369 5              CALL SNDSEQ(.TP$53C, SIZE(TP$53C));
370 5              STATB(02) = STATB(02) + 1;
371 5              STATB(00) = STATB(00) + 1;
372 5              END;

373 2          CALL SNDSEQ(.TP$51B, SIZE(TP$51B));
374 2          CALL HEX$ASC(CONTROL$CODE, 0);
375 2          CALL SNDSEQ(.TEMP1, 2);
376 2          CALL SNDSEQ(.TP$51C, SIZE(TP$51C));
377 2          CALL HEX$ASC(COUNTRY$CODE, 0);
378 2          CALL SNDSEQ(.TEMP1, 2);
379 2          CALL SNDSEQ(.TP$51A, SIZE(TP$51A));
380 2          CALL HEX$ASC(NETWORK$CODE, 0);
381 2          CALL SNDSEQ(.TEMP1, 2);
382 2          CALL SNDSEQ(.TP$51D, SIZE(TP$51D));
383 2          CALL HEX$ASC(LOBITS, 0);
384 2          CALL SNDSEQ(.TEMP1, 2);

```

```

385 2      CALL SNDSEQ(.TP$51E, SIZE(TP$51E));
386 2      CALL HEX$ASC(HIBITS, 0);
387 2      CALL SNDSEQ(.TEMP1, 2);
388 2      CALL SNDSEQ(.TP$52A, SIZE(TP$52A));
389 2      CALL HEX$ASC(HOST$CODE, 0);
390 2      CALL SNDSEQ(.TEMP1, 2);
391 2      CALL SNDSEQ(.TP$52, SIZE(TP$52));
392 2      CALL HEX$ASC(DESTINATION$ADDRESS, 0);
393 2      CALL SNDSEQ(.TEMP1, 2);

394 2      IF DESTINATION = 6 THEN /* LOC TO NET, GET SOURCE INFO */
395 2          DO;
396 3          SRC$NET$CODE = LCOXTB(13) AND 0FH;
397 3          IF (SRC$NET$CODE <= MAX$NETWORK$CODE) THEN
398 3              DO;
399 4              LOBITS = (ROR(LCOXTB(13), 4) AND 0FH);
400 4              HIBITS = (ROL(LCOXTB(14), 4) AND 0FH);
401 4              SRC$HOST$CODE = LOBITS OR HIBITS;
402 4              IF (SRC$HOST$CODE >= 0) AND (SRC$HOST$CODE <= 63) THEN
403 4                  SOURCE$ADDRESS = SRC$NET$CODE OR 10H;
404 4              ELSE
405 4                  IF (SRC$HOST$CODE >= 64) AND (SRC$HOST$CODE <= 127) THEN
406 4                      SOURCE$ADDRESS = SRC$NET$CODE OR 20H;
407 4                      ELSE
408 4                          IF (SRC$HOST$CODE >= 128) AND (SRC$HOST$CODE <= 191)
409 4                              THEN
410 4                                  SOURCE$ADDRESS = SRC$NET$CODE OR 30H;
411 4                                  ELSE
412 4                                      IF (SRC$HOST$CODE >= 192) AND (SRC$HOST$CODE <= 255)
413 4                                          THEN
414 4                                              SOURCE$ADDRESS = SRC$NET$CODE OR 40H;
415 4                      CALL SNDSEQ(.TP$52B, SIZE(TP$52B));
416 4                      CALL HEX$ASC(SOURCE$ADDRESS, 0);
417 4                      CALL SNDSEQ(.TEMP1, 2);
418 4                      END;
419 4                  ELSE
420 4                      DO;
421 4                      DESTINATION = 8;
422 4                      CALL SNDSEQ(.TP$53AA, SIZE(TP$53AA));
423 4                      STATTB(04) = STATTB(04) + 1;
424 4                      STATTB(00) = STATTB(00) + 1;
425 4                      END;
426 3          END;
427 2      END DET$ADDR;

```

```

/*****
PROCEDURE DET$ADDR$NL    DETERMINES THE DESTINATION OF DATA FROM THE NETWORK

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA

```

COMING FROM THE NETWORK SIDE OF THE UNIT TO A LOCAL HOST.

INPUT - THE INPUT IS AN ADDRESS VALUE INDICATING THE TABLE LOCATION OF THE DATA TO BE EVALUATED.

PROCESSING - THE PROCEDURE EXTRACTS THE DESTINATION\$ADDRESS FROM THE SECOND BYTE OF THE DATA PACKET AND RETURNS THE CORRESPONDING DESTINATION.

OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION (1,2,3, OR 4) OF THE DATA COMING FROM THE NETWORK TO A LOCAL HOST.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN.

NOTES: 1. BYTE AND 07H WILL MASK OUT THE UPPER 5-BITS.

*****/

```

422 1  DET$ADDR$NL; PROCEDURE BYTE ;
423 2      DECLARE PORT BYTE,
          BITS BYTE;

424 2      BITS = (ROR(NITLCTB(NITLONS), 4) AND 0FH);
425 2      IF (BITS >= 1 AND BITS <= 4) THEN
426 2          DO CASE BITS;
427 3              ;
428 3              PORT = 1;
429 3              PORT = 2;
430 3              PORT = 3;
431 3              PORT = 4;
432 3          END;
          ELSE
433 2              DO;
434 3              PORT = 8;
435 3              STATB(01) = STATB(01) + 1; /* INCREMENT LOCAL ERROR COUNT */
436 3              END;
437 2              RETURN PORT;
438 2  END DET$ADDR$NL;

```

PROCEDURE MOVE\$LL ROUTE OUT EITHER A DATA\$PACKET OR HOST DATA

THE PURPOSE OF THIS PROCEDURE IS TO ROUTE PACKETS FROM THE LOCAL TO LOCAL AND NETWORK TO LOCAL TABLES TO THE CORRECT OUTPUT CHANNEL.

INPUT - DATA PACKETS ARE ROUTED VIA EVALUATION OF LCLOTB AND NITLCTB ADDRESSES AND INTERNAL PACKET ROUTING INFORMATION

PROCESSING - THE PROCEDURE CHECKS EACH INPUT BUFFER'S ADDRESS FOR PACKET ARRIVAL. IF A PACKET IS READY, THE DESTINATION IS DETERMINED VIA PROCEDURE DET\$ADDR, AND TRANSMITTED

BY SETTING THE TRANSMIT INTERRUPTS THROUGH PROCEDURE
TRNMIT\$PKT. THE TABLE THAT WAS SERVICED (PACKET REMOVED)
HAS ITS ADDRESS\$S UPDATED BY SRVC\$TAB\$H\$SKP. WHEN THE
FRAME IS TRANSMITTED TO THE HOSTS FROM THE NET TO LOCAL
BUFFER TABLE, THE FIRST TWO BYTES OF HEADER INFORMATION
ARE STRIPPED OFF BEFORE TRANSMISSION.

OUTPUT - A PACKET OF DATA IS PREPARED FOR TRANSMISSION AND THE
INTERRUPTS ARE SET.

INTERFACE - THIS PROCEDURE IS CALLED IN AN INFINITE LOOP BY THE
MAIN PROGRAM.

/******

```

439 1  MOVE$LL: PROCEDURE(TABLE$PTR, PORT);
440 2      DECLARE PORT BYTE,
          TABLE$PTR ADDRESS,
          LOOKTB BASED TABLE$PTR(1) BYTE;

441 2      IF (PORT >= 1 AND PORT <= 4) THEN
442 2          DO CASE PORT;

443 3              ;      /* CASE ZERO IS NULL */

444 3          DO;      /* CASE ONE */
445 4              IF TRANS$1$RDY = TRUE THEN DO;
447 5                  CALL SNDSEQ(.TP$24, SIZE(TP$24));
448 5                  CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .TX01TB(TX01NE));
449 5                  CALL LD$TAB$H$SKP(8);
450 5              END;
451 4          END;

452 3          DO;      /* CASE TWO */
453 4              IF TRANS$2$RDY = TRUE THEN DO;
455 5                  CALL SNDSEQ(.TP$25, SIZE(TP$25));
456 5                  CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .TX02TB(TX02NE));
457 5                  CALL LD$TAB$H$SKP(9);
458 5              END;
459 4          END;

460 3          DO;      /* CASE THREE */
461 4              IF TRANS$3$RDY = TRUE THEN DO;
463 5                  CALL SNDSEQ(.TP$26, SIZE(TP$26));
464 5                  CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .TX03TB(TX03NE));
465 5                  CALL LD$TAB$H$SKP(10);
466 5              END;
467 4          END;

468 3          DO;      /* CASE FOUR */
469 4              IF TRANS$4$RDY = TRUE THEN DO;
471 5                  CALL SNDSEQ(.TP$27, SIZE(TP$27));
472 5                  CALL MOVE(DATA$GRAM$SIZE, TABLE$PTR, .TX04TB(TX04NE));
473 5                  CALL LD$TAB$H$SKP(11);
474 5              END;

```

```

475 4          END;

476 5          END;          /* END CASE */
477 2      ELSE DO;
478 3          CALL SNDSEQ(.TP$28, SIZE(TP$28));
479 3          STATB(L$RO$DEST$ERR) = STATB(L$RO$DEST$ERR) + 1;
480 3      END;

481 2      END MOVE$LL;

/*****
PROCEDURE MOVE$NL

THIS PROCEDURE MOVES A PACKET OF INFORMATION FROM THE NTLCTB INTO ONE
OF THE FOUR TABLES TO THE HOST. DESTINATION ADDRESS IS AN INPUT
PARAMETER TELLING WHICH HOST(PORT) THE PACKET SHOULD BE ROUTED TO.

CALLED BY ROUTE$IN
*****/
482 1      MOVE$NL: PROCEDURE(PORT);
483 2          DECLARE PORT BYTE;

484 2          IF (PORT >= 1 AND PORT <= 4) THEN
485 2              DO CASE PORT;

486 3              ;

487 3          DO;
488 4              CALL MOVE(DATA$GRAM$SIZE, .NTLCTB(NTLCNS + 5),
                          .TX01TB(TX01NE));
489 4              CALL SNDSEQ(.TP$30, SIZE(TP$30));
490 4              CALL LD$TAB$H$SKP(8);
491 4          END;

492 3          DO;
493 4              CALL MOVE(DATA$GRAM$SIZE, .NTLCTB(NTLCNS + 5),
                          .TX02TB(TX02NE));
494 4              CALL SNDSEQ(.TP$31, SIZE(TP$31));
495 4              CALL LD$TAB$H$SKP(9);
496 4          END;

497 3          DO;
498 4              CALL MOVE(DATA$GRAM$SIZE, .NTLCTB(NTLCNS + 5),
                          .TX03TB(TX03NE));
499 4              CALL SNDSEQ(.TP$32, SIZE(TP$32));
500 4              CALL LD$TAB$H$SKP(10);
501 4          END;

502 3          DO;
503 4              CALL MOVE(DATA$GRAM$SIZE, .NTLCTB(NTLCNS + 5),
                          .TX04TB(TX04NE));

```

```

504 4          CALL SNDSEQ(.TP$33, SIZE(TP$33));
505 4          CALL LD$TAB$H$SKP(11);
506 4          END;

507 3          END;
      ELSE
508 2          CALL SNDSEQ(.TP$34, SIZE(TP$34));

509 2  END MOVE$NL;

```

/*****

PROCEDURE ROUTE\$IN ROUTES IN EITHER DATA\$PACKETS OR HOST DATA

THE PURPOSE OF THIS PROCEDURE IS TO EITHER ROUTE HOST DATA
FROM THE FOUR LOCAL INPUT BUFFERS TO THEIR CORRECT OUTPUT
BUFFER OR ROUTE DATA\$PACKETS FROM THE NETWORK SIDE OF THE UNID
TO THEIR CORRECT OUTPUT LOCAL BUFFER MINUS THE DATA\$PACKET
HEADER.

INPUT - HOST DATA OR DATA PACKETS ARE ROUTED VIA EVALUATION OF
LCX\$TB ADDRESS AND INTERNAL PACKET ROUTING INFORMATION.

PROCESSING - THE PROCEDURE CHECKS EACH OF THE LOCAL INPUT BUFFER'S
ADDRESS FOR HOST DATA ARRIVAL. IF ANY HOST DATA IS READY,
THE DESTINATION IS DETERMINED VIA PROCEDURE DET\$ADDR. ONCE
THE DESTINATION IS DETERMINED THEN THE DATA IS EITHER SENT
TO A LOCAL BUFFER (LOCAL TO LOCAL TRANSFER) VIA PROCEDURE
MOVE\$Q OR THE HOST DATA IS CONVERTED INTO A DATA\$PACKET
(BUILD\$1\$PACKET) THEN SENT TO THE NETWORK SIDE OF THE UNID
VIA MOVE\$Q. BOTH THE BUFFER TABLE THAT IS LOADED AND THE
TABLE THAT IS SERVICED HAVE THEIR ADDRESSES HOUSECLEANED
BY LD\$TAB\$H\$SKP AND SRVC\$TAB\$H\$SKP.

OUTPUT - EITHER A DATA\$PACKET OR HOST DATA IS MOVED TO AN
APPROPRIATE DESTINATION BUFFER.

INTERFACE - THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY THE
MAIN PROGRAM

/*****/

```

510 1  ROUTE$IN: PROCEDURE;

511 2          IF ((LC$LINE - LC$INS) >= DATA$GRAM$SIZE) OR (LC$INS > LC$LINE)) THEN
512 2          DO;
513 3          CALL SNDSEQ(.TP$6, SIZE(TP$6));
514 3          CALL DET$ADDR(.LC$ITB(LC$INS));
515 3          IF DESTINATION = 5 THEN
516 3          DO;
517 4          CALL SNDSEQ(.TP$7, SIZE(TP$7));
518 4          CALL MOVE$LL(.LC$ITB(LC$INS), (ROR(DESTINATION$ADDRESS, 4)
                                         AND 0FH));
519 4          END;
      ELSE
520 3          IF DESTINATION = 6 THEN

```

```

521 3          DO;
522 4          CALL SNDSEQ(.TP$8, SIZE(TP$8));
523 4          CALL BUILD$1$PACKET(.LC01TB(LC01NS));
524 4          END;
          ELSE
525 3          DO;
526 4          CALL SNDSEQ(.TP$9, SIZE(TP$9));
527 4          STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;
528 4          END;
529 3          CALL SRVC$TAB$H$SKP(1);
530 3          END;

531 2          IF(((LC02NE - LC02NS) >= DATA$GRAM$SIZE) OR (LC02NS > LC02NE)) THEN
532 2          DO;
533 3          CALL SNDSEQ(.TP$10, SIZE(TP$10));
534 3          CALL DET$ADDR(.LC02TB(LC02NS));
535 3          IF DESTINATION = 5 THEN
536 3          DO;
537 4          CALL SNDSEQ(.TP$11, SIZE(TP$11));
538 4          CALL MOVE$LL(.LC02TB(LC02NS), (ROR(DESTINATION$ADDRESS, 4)
          AND OFH));
539 4          END;
          ELSE
540 3          IF DESTINATION = 6 THEN
541 3          DO;
542 4          CALL SNDSEQ(.TP$12, SIZE(TP$12));
543 4          CALL BUILD$1$PACKET(.LC02TB(LC02NS));
544 4          END;
          ELSE
545 3          DO;
546 4          CALL SNDSEQ(.TP$13, SIZE(TP$13));
547 4          CALL SNDSEQ(.TP$51, SIZE(TP$51));
548 4          CALL HEX$ASC(LOW(DESTINATION),0);
549 4          CALL SNDSEQ(.TEMP1, 2);
550 4          CALL SNDSEQ(.TP$52, SIZE(TP$52));
551 4          CALL HEX$ASC(DESTINATION$ADDRESS,0);
552 4          CALL SNDSEQ(.TEMP1, 2);
553 4          STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;
554 4          END;
555 3          CALL SRVC$TAB$H$SKP(2);
556 3          END;

557 2          IF(((LC03NE - LC03NS) >= DATA$GRAM$SIZE) OR (LC03NS > LC03NE)) THEN
558 2          DO;
559 3          CALL SNDSEQ(.TP$14, SIZE(TP$14));
560 3          CALL DET$ADDR(.LC03TB(LC03NS));
561 3          IF DESTINATION = 5 THEN
562 3          DO;
563 4          CALL SNDSEQ(.TP$15, SIZE(TP$15));
564 4          CALL MOVE$LL(.LC03TB(LC03NS), (ROR(DESTINATION$ADDRESS, 4)
          AND OFH));

```

```

565 4          END;
          ELSE
566 3          IF DESTINATION = 6 THEN
567 3          DO;
568 4          CALL SNDSEQ(.TP$16, SIZE(TP$16));
569 4          CALL BUILD$1$PACKET(.LC03TB(LC03NS));
570 4          END;
          ELSE
571 3          DO;
572 4          CALL SNDSEQ(.TP$17, SIZE(TP$17));
573 4          STATFB(L$R1$DEST$ERR) = STATFB(L$R1$DEST$ERR) + 1;
574 1          END;
575 3          CALL SRVC$TAB$HSKP(3);
576 3          END;

577 2          IF ((LC04NE - LC04NS) >= DATA$GRAM$SIZE) OR (LC04NS > LC04NE) THEN
578 2          DO;
579 3          CALL SNDSEQ(.TP$18, SIZE(TP$18));
580 3          CALL DET$ADDR(.LC04TB(LC04NS));
581 3          IF DESTINATION = 5 THEN
582 3          DO;
583 4          CALL SNDSEQ(.TP$19, SIZE(TP$19));
584 4          CALL MOVE$LL(.LC04TB(LC04NS), (ROR(DESTINATION$ADDRESS, 4)
                    AND OFH));
585 4          END;
586 4          END;
          ELSE
587 3          IF DESTINATION = 6 THEN
588 3          DO;
589 4          CALL SNDSEQ(.TP$20, SIZE(TP$20));
590 4          CALL BUILD$1$PACKET(.LC04TB(LC04NS));
591 4          END;
          ELSE
592 3          DO;
593 4          CALL SNDSEQ(.TP$21, SIZE(TP$21));
594 4          STATFB(L$R1$DEST$ERR) = STATFB(L$R1$DEST$ERR) + 1;
595 4          END;
596 3          CALL SRVC$TAB$HSKP(4);
597 3          END;

598 2          IF (NTLCNE - NTLCNS) >= PACKET$SIZE) OR (NTLCNS > NTLCNE) THEN
599 2          DO;
600 3          DESTINATION$ADDRESS = DET$ADDR$NL;
601 3          CALL MOVE$NL(DESTINATION$ADDRESS);
602 3          CALL SRVC$TAB$HSKP(7);
603 3          END;

604 2          END ROUTE$IN;

```

```

/*****
PROCEDURE SERVICE$LOOP

```

PURPOSE

URNS A FRAME AROUND AT THE NETWORK LAYER. THE SOURCE AND
DESTINATION ADDRESSES ARE SWITCHED AND THE PACKET IS PUT INTO THE
NETWORK LAYER TABLE.

INTERFACE

CALLED BY MAIN PROGRAM.

```

604 1  SERVICE$LOOP: PROCEDURE;
605 2      DECLARE INDEX ADDRESS;

606 2      IF (LCNTNE - LCNTNS) >= PACKET$SIZE) OR
           (LCNTNS > LCNTNE) THEN
607 2          DO;
608 3          CALL SNDSEQ(.TP$40, SIZE(TP$40));
609 3          NLCTB(NLCTNE + 0) = LCNTTB(LCNTNS + 1);
610 3          NLCTB(NLCTNE + 1) = LCNTTB(LCNTNS + 0); /*SWAP PACKET HEADER*/
611 3          DO INDEX = 2 TO (PACKET$SIZE - 1); /* SWAP DATA */
612 4              NLCTB(NLCTNE + INDEX) = LCNTTB(LCNTNS + INDEX);
613 4          END;
614 3          DO INDEX = 0 TO 3; /* SWAP IP DEST & SOURCE */
615 4              NLCTB(NLCTNE + INDEX + 17) = LCNTTB(LCNTNS + INDEX + 21);
616 4              NLCTB(NLCTNE + INDEX + 21) = LCNTTB(LCNTNS + INDEX + 17);
617 4          END;
618 3          CALL LD$TAB$H$SKP(7);
619 3          CALL SRVCTAB$H$SKP(6);
620 3          END;

621 2  END SERVICE$LOOP;

```

PROCEDURE LOAD AND READ\$TAB

THESE PROCEDURES ARE USED TO PUT A MESSAGE INTO THE LCNTTB TABLE AND
WRITE TO THE CRT A MESSAGE IN THE NLCTB TABLE.

```

622 1  LOAD: PROCEDURE(CHAN$PTR, DEST$UNIT, DEST$HOST);

623 2      DECLARE (INDEX, CHAN$PTR) ADDRESS;
624 2      DECLARE (DEST$UNIT, DEST$HOST) BYTE;
625 2      DECLARE LCOXTB BASED CHAN$PTR(1) BYTE;

626 2      DO INDEX = 0 TO (DATA$GRAM$SIZE - 1);
627 3          LCOXTB(INDEX) = ' ';
628 3      END;

/* IP SOURCE HEADER */
629 2      LCOXTB(12) = 10H; /* CC = 1, CT = 0 */
630 2      LCOXTB(13) = (THIS$UNIT$NBR AND OFH) OR
           (ROL(LOW(CHAN$NUM*56),4) AND OFOH);
           /* HC(L) = 0, NC = 2 */

```

```
26 1 DECLARE TP$8(*) BYTE DATA(OR,LF,
    'TP$8      Entering forever loop!');
```

```
27 1 DECLARE MSG1(*) BYTE DATA(OR,LF,
    'Do you want TR/TA handshake (Y/N) [N] ?');
```

```
28 1 DECLARE MSG2(*) BYTE DATA(OR,LF,
    'Which channel (1, 2, 3, 4) [1] ?');
```

/* THE FOLLOWING DECLARE STATEMENTS SET UP THE TRAP, RST5.5, RST6.5, RST7.5 INTERRUPT TABLE IN EPROM TO JUMP TO LOCATIONS IN THE 8155 RAM, WHERE ANOTHER JUMP TABLE IS ACCESSABLE BY PROGRAMS IN RAM. THE RAM TABLE IS INITIALIZED TO JMP 0 WHERE THE EPROM ENTRY JUMP IS LOCATED. */

```
29 1 DECLARE JUMP$CODE LITERALLY '0C3H';
```

```
30 1 DECLARE VECTOR$TABLE$BASE LITERALLY '7FF4H';
```

```
31 1 DECLARE VECTOR$TABLE$BASE$PTR ADDRESS,
    ADDR ADDRESS AT (.VECTOR$TABLE$BASE$PTR),
    VECTOR$TABLE BASED VECTOR$TABLE$BASE$PTR BYTE;
```

```
32 1 DECLARE CODE0(3) BYTE AT (0024H) DATA(0C3H,0F4H,7FH),
    CODE1(3) BYTE AT (002CH) DATA(0C3H,0F7H,7FH),
    CODE2(3) BYTE AT (0034H) DATA(0C3H,0FAH,7FH),
    CODE2A(3) BYTE AT (0038H) DATA(0C3H,0,0),
    CODE3(3) BYTE AT (003CH) DATA(0C3H,0FDH,7FH);
```

/* THIS DECLARE IS NOT CURRENTLY USED BUT MAY BE IN THE FUTURE
DECLARE

```
INT$VECTOR(8) ADDRESS, * INTERRUPT VECTORS *
TRAP$INT ADDRESS, * POWER FAIL SENSE *
INT$75 ADDRESS, * TIMER INPUTS, TINT0 & TINT1 *
INT$65 ADDRESS, * RING INDICATOR AND CARRIER DETECT *
INT$55 ADDRESS, * FLAG (FINT) AND MULTIBUS INTERRUPT *
RIM$MASK BYTE, * INTERRUPT MASK RETURNED BY RIM *
TRAP$INT$PTR LITERALLY '024H', * INTERRUPT LOCATIONS *
INT$55$PTR LITERALLY '02CH';
INT$65$PTR LITERALLY '034H';
INT$75$PTR LITERALLY '03CH';
```

*/

/*

ISBC 544 I/O DECLARATIONS SECTION
*/

```
33 1 DECLARE /* MASTER/SLAVE PORTS */
    SIM$MASK LITERALLY '01FH', /* SIM MASK-TURN OFF INTERRUPTS */
    MASTER LITERALLY '0E5H', /* MASTER MODE */
    SLAVE LITERALLY '0E4H'; /* SLAVE MODE */
```

```
34 1 DECLARE /* 8251 USARTS */
    US$PO$CMD LITERALLY '0D3H', /* SERIAL PORT 0 COMMAND */
```

```

DESTINATION$ADDRESS BYTE, /* DESTIN ADDR OF DATAGRAM */
SOURCE$ADDRESS BYTE,      /* SOURCE ADDR OF DATAGRAM */
FOREVER BYTE,

```

```

BRF0      ADDRESS,        /* DATA RATE FACTOR, USART 0 */
BRF1      ADDRESS,        /* DATA RATE FACTOR, USART 1 */
BRF2      ADDRESS,        /* DATA RATE FACTOR, USART 2 */
BRF3      ADDRESS,        /* DATA RATE FACTOR, USART 3 */
ISR$STAT  BYTE;           /* IRR STATUS BYTE */

```

/******

```

9  1  R$MASK: PROCEDURE BYTE EXTERNAL; /* USE THE PLM80.LIB WHEN LINKING */
10 2  END R$MASK;                      /* REQUIRED FOR THE 8085 PROCESSOR */

```

```

11 1  S$MASK: PROCEDURE (MASK) EXTERNAL;
12 2  DECLARE MASK BYTE;
13 2  END S$MASK;

```

/******

/* SIGNON MESSAGE AND DIAGNOSTIC MESSAGES */

```

14 1  DECLARE  HEADER(*) BYTE DATA(CR,LF,
      '          UNID 11 #2 LOCAL OS',CR,LF,
      '          TEST PROGRAM',CR,LF,
      '          VERS 1.1, 2 OCT 84',CR,LF,
      '          EXECUTING ',CR,LF);

```

```

15 1  DECLARE  TP$1(*) BYTE DATA(CR,LF,
      'TP$1      Received datagram, putting in LC01TX');
16 1  DECLARE  TP$2(*) BYTE DATA(CR,LF,
      'TP$2      In ROUTE$OUT');
17 1  DECLARE  TP$3(*) BYTE DATA(CR,LF,
      'TP$3      Sent TR');
18 1  DECLARE  TP$3A(*) BYTE DATA(CR,LF,
      'TP$3A     Sent TA');
19 1  DECLARE  TP$3R(*) BYTE DATA(CR,LF,
      'TP$3R     Received TR');
20 1  DECLARE  TP$3C(*) BYTE DATA(CR,LF,
      'TP$3C     Received TA');
21 1  DECLARE  TP$4(*) BYTE DATA(CR,LF,
      'TP$4      Not TRTA, sending datagram');
22 1  DECLARE  TP$4A(*) BYTE DATA(CR,LF,
      'TP$4A     Completed sending datagram');
23 1  DECLARE  TP$5(*) BYTE DATA(CR,LF,
      'TP$5      Completed initializing 544 board');
24 1  DECLARE  TP$6(*) BYTE DATA(CR,LF,
      'TP$6      Completed initializing variables');
25 1  DECLARE  TP$7(*) BYTE DATA(CR,LF,
      'TP$7      Completed initializing tables');

```

```

      AT (.SYS$BASE + 30 + PACKET$TABLE$SIZE +
          PACKET$TABLE$SIZE +
          PACKET$TABLE$SIZE +
          PACKET$TABLE$SIZE +
          FRAME$TABLE$SIZE);

```

/* END OF SHARED MEMORY TABLES AND VARIABLES */

8 1 DECLARE

```

      LC01TX (DATA$TABLE$SIZE) BYTE,
      LC02TX (DATA$TABLE$SIZE) BYTE,
      LC03TX (DATA$TABLE$SIZE) BYTE,
      LC04TX (DATA$TABLE$SIZE) BYTE,

```

```

      (LC01NS, LC01NE, LC01SZ,
       LC02NS, LC02NE, LC02SZ,
       LC03NS, LC03NE, LC03SZ,
       LC04NS, LC04NE, LC04SZ) ADDRESS,

```

```

      (NT01NS, NT01NE, NT01SZ,
       NT02NS, NT02NE, NT02SZ) ADDRESS,

```

```

      (TX01NS, TX01NE, TX01SZ,
       TX02NS, TX02NE, TX02SZ,
       TX03NS, TX03NE, TX03SZ,
       TX04NS, TX04NE, TX04SZ) ADDRESS,

```

```

      STATTB (STAT$NBR) BYTE,

```

```

      /*****
      /*  ADDITIONAL DECLARES NEEDED FOR THIS PROGRAM  */
      /*****

```

/* INTERNAL VARIABLES USED IN THIS MODULE */

```

      (SEND$1, SEND$2, SEND$3, SEND$4) BYTE,

```

```

      (BYTES$SENT$1, BYTES$SENT$2,
       BYTES$SENT$3, BYTES$SENT$4) BYTE,

```

```

      (BYTES$RECV$1, BYTES$RECV$2,
       BYTES$RECV$3, BYTES$RECV$4) BYTE,

```

```

      (TRTA$1, TXTR$1, TXTA$1, RXTR$1, RXTA$1,
       TRTA$2, TXTR$2, TXTA$2, RXTR$2, RXTA$2,
       TRTA$3, TXTR$3, TXTA$3, RXTR$3, RXTA$3,
       TRTA$4, TXTR$4, TXTA$4, RXTR$4, RXTA$4) BYTE,

```

```

      (CHAR$1, CHAR$2, CHAR$3, CHAR$4, CHAR$X) BYTE,

```

```

      DESTINATION BYTE,          /* DESTINATION OF THE PACKET */

```

/* CAUTION: THIS IS THE BEGINNING OF THE SHARED MEMORY AREAS */

/* CAUTION: THE FOLLOWING DECLARE STATEMENTS ARE CAREFULLY CHOSEN TO ALIGN THE UNID LOCAL TRANSMIT, RECEIVE AND NETWORK RECEIVE TABLES AND THE FLAGS AND SEMAPHORES, BEGINNING AT 8001H. EACH TABLE IS CAREFULLY ALLIGNED SO THAT THEY RESIDE IN CONTIGOUS MEMORY LOCATIONS. ANY CHANGE IN THESE DECLARE STATEMENTS SHOULD ONLY BE DONE AFTER CAREFULL READING OF THE THESIS BY CHILDRESS, ESPECIALLY CHAPTERS FOUR AND FIVE (4 & 5). THE ORDER IS ALSO CRITICAL. THE INITIAL DECLARES FOR THE TABLES MUST BE AT THE CORRECT OFFSET FROM SYS\$BASE IN ORDER TO RESIDE IN THE CORRECT LOCATION IN MEMORY. THE REMAINING LOCAL TRANSMIT TABLES AND OTHER PROGRAM VARIABLES THEN FOLLOW THE SHARED MEMORY AREAS. SEE THE BATCH FILE USED TO LINK AND LOAD THIS MODULE.

*/

5 1 DECLARE SYS\$MEM\$BASE LITERALLY '08001H';/* SYSTEM MEMORY BASE ADDRESS */

6 1 DECLARE SYS\$BASE BYTE AT (SYS\$MEM\$BASE);

/* DECLARATIONS FOR FLAGS AND TABLES IN SYSTEM MEMORY */

7 1 DECLARE
 (LSEM\$1, LSEM\$2, LSEM\$3, /* LOCAL TO NET SEMAPHORE AND */
 LSEM\$4, NSEM\$1, NSEM\$2) BYTE /* NET TO LOCAL SEMAPHORE */
 AT (.SYS\$BASE),

(LPTR\$1, LSPARE\$1, LPTR\$2, LSPARE\$2, /* LOC TO NET PACKET PTR */
 LPTR\$3, LSPARE\$3, LPTR\$4, LSPARE\$4, /* LOC TO NET PACKET PTR */
 NPTR\$1, NSPARE\$1, NPTR\$2, NSPARE\$2) ADDRESS /* NET TO LOC PACKET PTR */
 AT (.SYS\$BASE + 6),

LC01RX (PACKET\$TABLE\$SIZE) BYTE /* LOCAL BOARD RECEIVE TABLES */
 AT (.SYS\$BASE + 30),

LC02RX (PACKET\$TABLE\$SIZE) BYTE
 AT (.SYS\$BASE + 30 + PACKET\$TABLE\$SIZE),

LC03RX (PACKET\$TABLE\$SIZE) BYTE
 AT (.SYS\$BASE + 30 + PACKET\$TABLE\$SIZE +
 PACKET\$TABLE\$SIZE),

LC04RX (PACKET\$TABLE\$SIZE) BYTE
 AT (.SYS\$BASE + 30 + PACKET\$TABLE\$SIZE +
 PACKET\$TABLE\$SIZE +
 PACKET\$TABLE\$SIZE),

NT01RX (FRAME\$TABLE\$SIZE) BYTE /* NETWORK BOARD RECEIVE TABLES */
 AT (.SYS\$BASE + 30 + PACKET\$TABLE\$SIZE +
 PACKET\$TABLE\$SIZE +
 PACKET\$TABLE\$SIZE +
 PACKET\$TABLE\$SIZE),

NT02RX (FRAME\$TABLE\$SIZE) BYTE

```

DATA$TABLE$SIZE LITERALLY '1280', /* DATAGRAM * NBR OF
                                DATAGRAMS */
PACKET$TABLE$SIZE LITERALLY '1330', /* NUMBR OF BYTES IN TABLE */
FRAME$TABLE$SIZE LITERALLY '1350', /* NUMBR OF BYTES IN TABLE */
TCP$DATA$SIZE LITERALLY '72', /* TCP DATA SIZE */
TR LITERALLY '42H', /* TRANSMIT REQUEST CHAR */
TA LITERALLY '41H', /* TRANSMIT ACKNOWLEDGE CHAR */
MAX$RXTA$TRIES LITERALLY '3', /* MAXIMUM NUMBER OF TA WAIT
                                TRIES */
OFFSET1 LITERALLY '5', /* OFFSET FROM PACKET TO IP
                                HEADER */

```

/* FOLLOWING ARE NETWORK DEFINED VARIABLES */

/* NOTES: 1. THIS\$UNID\$NBR MUST REFLECT WHICH UNID THIS IS.

2. THIS\$COUNTRY\$CODE MUST REFLECT THE AREA TO WHICH
THIS UNID IS LOCATED.

3. MAX\$COUNTRY\$CODE WILL INDICATE WHICH COUNTRY CODES
ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR
THE DELNET MONITOR.

4. MAX\$NETWORK\$CODE WILL INDICATE HOW MANY UNIDS ARE
CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.

5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO
PHISTER'S THESIS, APPENDIX D. */

```

THIS$UNID$NBR LITERALLY '02', /* UNIQUE ADDRESS FOR THIS UNID */
THIS$COUNTRY$CODE LITERALLY '09', /* CC WHERE THIS UNID RESIDES */
MAX$COUNTRY$CODE LITERALLY '09', /* INDICATES COUNTRY$CODES IN USE */
MAX$NETWORK$CODE LITERALLY '03'; /* INDICATES UNIDS OPERATIONAL
                                IN NET */

```

/* NOTE: THE LITERAL THIS\$COUNTRY\$CODE OF 9, ABOVE, IS HERE FOR USE WITH THE
SUMMER QUARTER EE 6.90 PROJECT; I.E., INTERFACE THE UNID 11 WITH THE NETOS */

```

3 1 DECLARE L$R1$DEST$ERR LITERALLY '5', /* LOCAL ROUTE$IN ERROR */
           L$R0$DEST$ERR LITERALLY '6'; /* LOCAL ROUTE$OUT ERROR */

```

/* MISCELLANEOUS DECLARATIONS */

```

1 1 DECLARE BUSY LITERALLY 'OFFH',
           TRUE LITERALLY 'OFFH',
           FALSE LITERALLY '00H',
           READY LITERALLY '1',
           DONE LITERALLY '2',
           OR LITERALLY '0DH',
           LF LITERALLY '0AH',
           USARTON LITERALLY '0010$0111B',
           USARTOFF LITERALLY '0000$0110B';

```

```

/*****
/* DATA TABLES USED IN THIS PROGRAM */
*****/

```

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE MAIN
 OBJECT MODULE PLACED IN :F1:NEWLOC.OBJ
 COMPILER INVOKED BY: PLMB0 :F1:NEWLOC.SRC

\$TITLE('LOCAL NETWORK TEST PROGRAM, 2 OCT 1984')
 \$INTVECTOR(4,20H)

```

/*****
/*
/*      DATE:      2 OCT 84
/*      VERSION:   1.1
/*
/*      TITLE:     LOCAL NETWORK LAYER TEST PROGRAM
/*      FILE NAME:  NEWLOC.SRC
/*      OWNER:     C. T. CHILDRESS
/*
/*      SOFTWARE SYSTEM:  UNID II
/*      USE:        SIMULATE THE FUNCTION OF THE NETWORK LAYER
/*                  SOFTWARE FOR USE WITH THE SBC 544
/*
/*      CONTENTS:   PL/M SOURCE CODE FOR THE UNID II NETWORK
/*                  LAYER SOFTWARE.
/*
*****/

```

```

/*****
PROLOGUE - MODULE L.MAIN$U1
DATE TRANSLATED: 1 SEP 83
DATE LAST MODIFIED: 2 OCT 84

```

*** PURPOSE IS TO SIMULATE THE MAIN 544 CODE USING JUST ONE SET OF
 TABLES. THIS PIECE OF S/W WILL WORK WITH THE CP/M SIMULATION
 IN CPMTMP.SRC

```

/*****

```

1 MAIN: DO;

/* THE FOLLOWING ARE UNID DEFINED VARIABLES */
 /* NOTE: THESE VARIABLES MAY CHANGE DEPENDING ON THE
 SOFTWARE CONFIGURATION USED WITHIN THE DELNET */

2 1 DECLARE DATA\$GRAM\$SIZE LITERALLY '128', /* DATAGRAM FROM HOST */
 PACKET\$SIZE LITERALLY '133', /* DATAGRAM + 5 FOR HEADER */
 FRAME\$SIZE LITERALLY '135', /* PACKET + 2 FOR HEADER */
 PACKET\$INSTABLE LITERALLY '10',
 STAT\$NBR LITERALLY '20', /* STATUS ENTRIES IN STATTB */

3. SBC 544 Validation

The purpose of this program is to operate the network level software on the Intel SBC 544. All the constants, variables and procedures from the network layer simulation are used in this program with the following exceptions:

1. The READ, WRITE, EXIT and ERROR ISIS system calls are not used.
2. The constants R\$CONN and W\$CONN are not used.
3. The variables ACTUAL, BUFFER, ERRNUM and STATUS are not used.

PL/M-80 COMPILER LOCAL NETWORK TEST PROGRAM, 30 SEP 1984

/***** THE END *****/

MODULE INFORMATION:

CODE AREA SIZE = 274AH 10058D
VARIABLE AREA SIZE = 33B8H 13240D
MAXIMUM STACK SIZE = 000CH 12D
1686 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION

```

847 3      END;

848 2      CALL SNDSEQ(.TP$50, SIZE(TP$50));
849 2      CALL HEX$ASC(LOW(CHAN$NUM), 0);
850 2      CALL SNDSEQ(.TEMP1, 2);
851 2      CALL SNDSEQ(.TP$51A, SIZE(TP$51A));
852 2      CALL HEX$ASC(DEST$NET$CODE, 0);
853 2      CALL SNDSEQ(.TEMP1, 2);
854 2      CALL SNDSEQ(.TP$52A, SIZE(TP$52A));
855 2      CALL HEX$ASC(DEST$HOST$CODE, 0);
856 2      CALL SNDSEQ(.TEMP1, 2);

857 2      CALL SNDSEQ(.MSG2, SIZE(MSG2));
858 2      CALL READ(R$CONN, .BUFFER, 129, .ACTUAL, .STATUS);
859 2      CALL ERR$CHK;

860 2      IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN
861 2          FOREVER = FALSE;
      ELSE
862 2          FOREVER = TRUE;

863 2      END READ$LINE;

      /*****
/*      THIS IS THE MAIN BODY OF THE PROGRAM      */
      *****/

864 1      BEGIN;

      CALL SNDSEQ(.STARTUP$HDR, SIZE(STARTUP$HDR));
865 1      CALL SNDSEQ(.TP$1, SIZE(TP$1));
866 1      CALL INIT$L$TAB;
867 1      CALL SNDSEQ(.TP$2, SIZE(TP$2));
868 1      CALL SNDSEQ(.TP$3, SIZE(TP$3));
869 1      CALL INVINT;
870 1      CALL SNDSEQ(.TP$4, SIZE(TP$4));
871 1      FOREVER = TRUE;
872 1      CALL READ$LINE;
873 1      DO WHILE FOREVER;
874 2          CALL READ$LCTAB;
875 2          CALL SNDSEQ(.TP$5, SIZE(TP$5));
876 2          CALL ROUTE$IN;
877 2          CALL SERVICE$LOOP;
878 2          CALL READ$XTAB;
879 2          CALL SNDSEQ(.TP$35, SIZE(TP$35));
880 2          CALL READ$LINE;
881 2      END;
882 1      CALL SNDSEQ(.TP$36, SIZE(TP$36));
883 1      CALL EXIT;

884 1      END MAIN;

```

```

/*****
PROCEDURE READ$LINE

```

PURPOSE:

READS A LINE OF DATA FROM THE HOST ISIS II CONSOLE, THEN INTERPRETED FOR EXECUTING EITHER LOAD ANOTHER TEST MESSAGE IN THE LCNTTB OR STOPPING THE TEST.

```

*****/

```

```

810 1  READ$LINE: PROCEDURE;
811 2  DECLARE CHAN$PTR ADDRESS,
      (1, DEST$NET$CODE, DEST$HOST$CODE) BYTE;

812 2  I = 0;

813 2  CALL SNDSEQ(.MSG1, SIZE(MSG1));
814 2  CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
815 2  CALL ERR$CHK;

816 2  IF (BUFFER(0) = 'Y') OR (BUFFER(0) = 'y') THEN
817 2  DO;
818 3  CALL SNDSEQ(.MSG3, SIZE(MSG3));
819 3  CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
820 3  CALL ERR$CHK;
821 3  CHAN$NUM = DOUBLE(ASC$HEX(BUFFER(0)));
822 3  IF (CHAN$NUM >= 1) AND (CHAN$NUM <= 4) THEN
823 3  DO;
824 4  CHAN$PTR = UPDATE$LCPTR(CHAN$NUM);
825 4  CALL SNDSEQ(.MSG5, SIZE(MSG5));
826 4  CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
827 4  CALL ERR$CHK;
828 4  DEST$NET$CODE = ASC$HEX(BUFFER(0));

829 4  CALL SNDSEQ(.MSG6, SIZE(MSG6));
830 4  CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
831 4  CALL ERR$CHK;
832 4  DEST$HOST$CODE = 0;
833 4  DO I = 1 TO (ACTUAL-2);
834 5  DEST$HOST$CODE = ROL(DEST$HOST$CODE, 4);
835 5  DEST$HOST$CODE = DEST$HOST$CODE OR ASC$HEX(BUFFER(I-1));
836 5  END;

837 4  CALL SNDSEQ(.MSG4, SIZE(MSG4));
838 4  CALL READ(R$CONN, .BUFFER, 128, .ACTUAL, .STATUS);
839 4  CALL ERR$CHK;
840 4  IF (BUFFER(0) >= '1') AND (BUFFER(0) <= '9') THEN
841 4  DO I = 1 TO ASC$HEX(BUFFER(0));
842 5  CALL LOAD(CHAN$PTR, DEST$NET$CODE, DEST$HOST$CODE);
843 5  CALL LD$TAB$H$SKP(CHAN$NUM);
844 5  CHAN$PTR = UPDATE$LCPTR(CHAN$NUM);
845 5  END;
846 4  END;

```

```

766 4          I = I + 1;
767 4          END;
768 3          I = 0;
769 3          DO WHILE I <= 15;
770 4              TEMP2(I) = TEMP1(14-I);
771 4              TEMP2(I+1) = TEMP1(15-I);
772 4              I = I + 2;
773 4          END;
774 3          CALL SNDSEQ(.TP$56C, SIZE(TP$56C));
775 3          CALL SNDSEQ(.CRLF, SIZE(CRLF));
776 3          CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
777 3          CALL SNDSEQ(.LC03TB(LC03NS + 56), TCP$DATA$SIZE);
778 3          END;

779 2          IF ((LC04NE - LC04NS) >= DATA$GRAM$SIZE) OR (LC04NS > LC04NE) THEN
780 2              DO;
781 3                  I = 12;
782 3                  DO WHILE I <= 18;
783 4                      CALL HEX$ASC(LC04TB(LC04NS + I), (I-12)*2);
784 4                      I = I + 1;
785 4                  END;
786 3                  I = 0;
787 3                  DO WHILE I <= 15;
788 4                      TEMP2(I) = TEMP1(14-I);
789 4                      TEMP2(I+1) = TEMP1(15-I);
790 4                      I = I + 2;
791 4                  END;
792 3                  CALL SNDSEQ(.TP$56D, SIZE(TP$56D));
793 3                  CALL SNDSEQ(.CRLF, SIZE(CRLF));
794 3                  CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
795 3                  CALL SNDSEQ(.LC04TB(LC04NS + 56), TCP$DATA$SIZE);
796 3                  END;

797 2          END READ$LC04TB;

798 1          UPDATE$LOPTR: PROCEDURE (CHAN$NUM) ADDRESS;
799 2          DECLARE (CHAN$NUM, CHAN$PTR) ADDRESS;

800 2          IF (CHAN$NUM >= 1 AND CHAN$NUM <= 4) THEN
801 2              DO CASE CHAN$NUM;
802 3                  ;
803 3                  CHAN$PTR = .LC01TB(LC01NE);
804 3                  CHAN$PTR = .LC02TB(LC02NE);
805 3                  CHAN$PTR = .LC03TB(LC03NE);
806 3                  CHAN$PTR = .LC04TB(LC04NE);
807 3              END;

808 2          RETURN CHAN$PTR;

809 2          END UPDATE$LOPTR;

```

```

720 3          CALL SRVC$TAB$H$SKP(11);
721 3          END;

722 2      END READ$XTAB;

723 1      READ$LC$TAB: PROCEDURE;
724 2          DECLARE I BYTE;

725 2          IF ((LC01NE - LC01NS) >= DATA$GRAM$SIZE) OR (LC01NS > LC01NE) THEN
726 2              DO;
727 3                  I = 12;
728 3                  DO WHILE I <= 18;
729 4                      CALL HEX$ASC(LC01TB(LC01NS + I), (I-12)*2);
730 4                      I = I + 1;
731 4                  END;
732 3                  I = 0;
733 3                  DO WHILE I <= 15;
734 4                      TEMP2(I) = TEMP1(14-I);
735 4                      TEMP2(I+1) = TEMP1(15-I);
736 4                      I = I + 2;
737 4                  END;
738 3                  CALL SNDSEQ(.TP$56A, SIZE(TP$56A));
739 3                  CALL SNDSEQ(.CRLF, SIZE(CRLF));
740 3                  CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
741 3                  CALL SNDSEQ(.LC01TB(LC01NS + 56), TOP$DATA$SIZE);
742 3          END;

743 2      IF ((LC02NE - LC02NS) >= DATA$GRAM$SIZE) OR (LC02NS > LC02NE) THEN
744 2          DO;
745 3              I = 12;
746 3              DO WHILE I <= 18;
747 4                  CALL HEX$ASC(LC02TB(LC02NS + I), (I-12)*2);
748 4                  I = I + 1;
749 4              END;
750 3              I = 0;
751 3              DO WHILE I <= 15;
752 4                  TEMP2(I) = TEMP1(14-I);
753 4                  TEMP2(I+1) = TEMP1(15-I);
754 4                  I = I + 2;
755 4              END;
756 3              CALL SNDSEQ(.TP$56B, SIZE(TP$56B));
757 3              CALL SNDSEQ(.CRLF, SIZE(CRLF));
758 3              CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
759 3              CALL SNDSEQ(.LC02TB(LC02NS + 56), TOP$DATA$SIZE);
760 3          END;

761 2      IF ((LC03NE - LC03NS) >= DATA$GRAM$SIZE) OR (LC03NS > LC03NE) THEN
762 2          DO;
763 3              I = 12;
764 3              DO WHILE I <= 18;
765 4                  CALL HEX$ASC(LC03TB(LC03NS + I), (I-12)*2);

```

```

671 4      END;
672 3      I = 0;
673 3      DO WHILE I <= 15;
674 4          TEMP2(I) = TEMP1(14-I);
675 4          TEMP2(I+1) = TEMP1(15-I);
676 4          I = I + 2;
677 4      END;
678 3      CALL SNDSEQ(.TP$55B, SIZE(TP$55B));
679 3      CALL SNDSEQ(.CRLF, SIZE(CRLF));
680 3      CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
681 3      CALL SNDSEQ(.TX02TB(TX02NS + 56), TCP$DATA$SIZE);
682 3      CALL SRVC$TAB$H$SKP(9);
683 3      END;

684 2      IF ((TX03NE - TX03NS) >= DATA$GRAM$SIZE) OR (TX03NS > TX03NE) THEN
685 2          DO;
686 3              I = 12;
687 3              DO WHILE I <= 18;
688 4                  CALL HEX$ASC(TX03TB(TX03NS + I), (I-12)*2);
689 4                  I = I + 1;
690 4              END;
691 3              I = 0;
692 3              DO WHILE I <= 15;
693 4                  TEMP2(I) = TEMP1(14-I);
694 4                  TEMP2(I+1) = TEMP1(15-I);
695 4                  I = I + 2;
696 4              END;
697 3              CALL SNDSEQ(.TP$55C, SIZE(TP$55C));
698 3              CALL SNDSEQ(.CRLF, SIZE(CRLF));
699 3              CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
700 3              CALL SNDSEQ(.TX03TB(TX03NS + 56), TCP$DATA$SIZE);
701 3              CALL SRVC$TAB$H$SKP(10);
702 3          END;

703 2      IF ((TX04NE - TX04NS) >= DATA$GRAM$SIZE) OR (TX04NS > TX04NE) THEN
704 2          DO;
705 3              I = 12;
706 3              DO WHILE I <= 18;
707 4                  CALL HEX$ASC(TX04TB(TX04NS + I), (I-12)*2);
708 4                  I = I + 1;
709 4              END;
710 3              I = 0;
711 3              DO WHILE I <= 15;
712 4                  TEMP2(I) = TEMP1(14-I);
713 4                  TEMP2(I+1) = TEMP1(15-I);
714 4                  I = I + 2;
715 4              END;
716 3              CALL SNDSEQ(.TP$55D, SIZE(TP$55D));
717 3              CALL SNDSEQ(.CRLF, SIZE(CRLF));
718 3              CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
719 3              CALL SNDSEQ(.TX04TB(TX04NS + 56), TCP$DATA$SIZE);

```

```

631 2          LCOXTB( 14) = 70H OR (ROR(LOW(CHAN$NUM*56),4) AND 0FH);
                                /* PC(0) = 7, HC(H) = 0 */
632 2          LCOXTB( 15) = 00H;      /* PC(2) = 0, PC(1) = 0 */

/* IP DESTINATION HEADER */
633 2          LCOXTB( 16) = 10H;      /* CC    = 1, CT    = 0 */
634 2          LCOXTB( 17) = (DEST$UNID AND 0FH) OR
                                ROL((DEST$HOST AND 0FH), 4);
                                /* HC(L) = 6, NC    = 2 */
635 2          LCOXTB( 18) = ROR((DEST$HOST AND 0FH),4) OR 70H;
                                /* PC(0) = 7, HC(H) = 0 */
636 2          LCOXTB( 19) = 00H;      /* PC(2) = 0, PC(1) = 0 */

637 2          CALL MOVE(TCP$DATA$SIZE, .MESSAGE, .LCOXTB( 56));
638 2          IF MSGNUM > 9 THEN
639 2              MSGNUM = 0;
640 2          LCOXTB( 86) = '0' + MSGNUM;
641 2          MSGNUM = MSGNUM + 1;
642 2          CALL SNDSEQ(.TP$54, SIZE(TP$54));

643 2          END LOAD;

644 1          READ$XTAB: PROCEDURE;
645 2              DECLARE I BYTE;

646 2              IF ((TX01NE - TX01NS) >= DATA$GRAM$SIZE) OR (TX01NS > TX01NE) THEN
647 2                  DO;
648 3                      I = 12;
649 3                      DO WHILE I <= 18;
650 4                          CALL HEX$ASC(TX01TB(TX01NS + I),(I-12)*2);
651 4                          I = I + 1;
652 4                      END;
653 3                      I = 0;
654 3                      DO WHILE I <= 15;
655 4                          TEMP2(I) = TEMP1(14-I);
656 4                          TEMP2(I+1) = TEMP1(15-I);
657 4                          I = I + 2;
658 4                      END;
659 3                      CALL SNDSEQ(.TP$55A, SIZE(TP$55A));
660 3                      CALL SNDSEQ(.CRLF, SIZE(CRLF));
661 3                      CALL SNDSEQ(.TEMP2, SIZE(TEMP2));
662 3                      CALL SNDSEQ(.TX01TB(TX01NS + 56), TCP$DATA$SIZE);
663 3                      CALL SRVC$TAB$H$SKP(8);
664 3                      END;

665 2              IF ((TX02NE - TX02NS) >= DATA$GRAM$SIZE) OR (TX02NS > TX02NE) THEN
666 2                  DO;
667 3                      I = 12;
668 3                      DO WHILE I <= 18;
669 4                          CALL HEX$ASC(TX02TB(TX02NS + I),(I-12)*2);
670 4                          I = I + 1;

```

```

US$P0$STAT    LITERALLY '0D1H',    /* SERIAL PORT 0 STATUS */
US$P0$DATA    LITERALLY '0D0H',    /* SERIAL PORT 0 DATA */
US$P1$CMD     LITERALLY '0D3H',    /* SERIAL PORT 1 COMMAND */
US$P1$STAT    LITERALLY '0D3H',    /* SERIAL PORT 1 STATUS */
US$P1$DATA    LITERALLY '0D2H',    /* SERIAL PORT 1 DATA */
US$P2$CMD     LITERALLY '0D5H',    /* SERIAL PORT 2 COMMAND */
US$P2$STAT    LITERALLY '0D5H',    /* SERIAL PORT 2 STATUS */
US$P2$DATA    LITERALLY '0D4H',    /* SERIAL PORT 2 DATA */
US$P3$CMD     LITERALLY '0D7H',    /* SERIAL PORT 3 COMMAND */
US$P3$STAT    LITERALLY '0D7H',    /* SERIAL PORT 3 STATUS */
US$P3$DATA    LITERALLY '0D6H',    /* SERIAL PORT 3 DATA */
US$MODE       LITERALLY '04EH',    /* SERIAL PORT MODE */
US$COMMAND    LITERALLY '027H',    /* SERIAL PORT COMMAND */

```

```

US$RESET$CMD  LITERALLY '40H',     /* RESET USART */
US$DTR$ON     LITERALLY '27H',     /* RTS,RXE,DTR,TXE */
US$RTS$CMD    LITERALLY '37H',     /* RTS,ER,RXE,DTR,TXE */
US$TTY$CMD    LITERALLY '35H',     /* RTS,ER,RXE,TXE */
US$DTR$OFF    LITERALLY '25H',     /* RTS,RXE,TXE */
US$RXRDY      LITERALLY '02H',     /* RECIEVER READY */
US$TXE        LITERALLY '04H',     /* TRANSMITTER EMPTY */
US$TXRDY      LITERALLY '01H',     /* TRANSMITTER READY */
PARITY$MASK   LITERALLY '7FH';     /* MASK OFF PARITY BIT */

```

```

35  1  DECLARE                                /* 8253 INTERVAL TIMER */
      IT1$CONT LITERALLY '0DBH',    /* INTERVAL TIMER 1 CONTROL */
      IT1$CNTR0 LITERALLY '0D8H',   /* COUNTER 0, USART 0 */
      IT1$CNTR1 LITERALLY '0D9H',   /* COUNTER 1, USART 1 */
      IT1$CNTR2 LITERALLY '0DAH',   /* COUNTER 2, USART 2 */
      IT2$CONT LITERALLY '0DFH',    /* INTERVAL TIMER 2 CONTROL */
      IT2$CNTR0 LITERALLY '0DCH',   /* COUNTER 3, USART 3 */
      IT2$CNTR1 LITERALLY '0DDH',   /* COUNTER 4, CNTR5 OR SPLIT CLOCKS */
      IT2$CNTR2 LITERALLY '0DEH',   /* COUNTER 5, RST 7.5 */
      USART$CNTR$M3 LITERALLY '036H', /* DIVIDE BY N RATE GENERATOR, MODE 3, */
                                     /* FOR USART CLK x 16, CLK = 1.2288 MHZ */

```

```

B19200    LITERALLY '4',           /* TIMER VALUE FOR 19.2 Kbps */
B9600     LITERALLY '8',           /* TIMER VALUE FOR 9600 BPS */
B4800     LITERALLY '16',          /* TIMER VALUE FOR 4800 BPS */
B2400     LITERALLY '32',          /* TIMER VALUE FOR 2400 BPS */
B1200     LITERALLY '64',          /* TIMER VALUE FOR 1200 BPS */
B600      LITERALLY '128',         /* TIMER VALUE FOR 600 BPS */
B300      LITERALLY '256',         /* TIMER VALUE FOR 300 BPS */
B150      LITERALLY '512',         /* TIMER VALUE FOR 150 BPS */
B110      LITERALLY '698';         /* TIMER VALUE FOR 110 BPS */

```

```

36  1  DECLARE                                /* 8155 PERIPHERAL INTERFACE */
      PI$PORTA LITERALLY '0E9H',    /* PORT A (OUTPUT) */
      PI$PORTB LITERALLY '0EAH',    /* PORT B (INPUT) */
      PI$PORTC LITERALLY '0ERH',    /* PORT C (INPUT) */
      PI$STAT  LITERALLY '0ERH',    /* PPI STATUS */

```

```

PIS$CMD      LITERALLY '0E8H', /* PPI COMMAND */
PIS$CNTR$LO  LITERALLY '0E8H', /* PPI COUNTER LO BYTE */
PIS$CNTR$HI  LITERALLY '0EDH', /* PPI COUNTER HI BYTE */
PIS$CNTR$LOCNT LITERALLY '16384', /* PPI COUNTER TIME CONST */
PIS$CNTR$HICNT LITERALLY '75', /* PPI COUNTER TIME CONST */
PIS$INIT$CMD1 LITERALLY '041H', /* PPI INITIALIZATION COMMAND 1 */
/* A OUT, B & C IN, STOP COUNT */
PIS$INIT$CMD2 LITERALLY '0C1H', /* PPI INITIALIZATION COMMAND 2 */
/* A OUT, B & C IN, START COUNT */
PIS$INIT$USS$INT1 LITERALLY '0F0H', /* USART AND INT CONT RESET */
PIS$INIT$USS$INT2 LITERALLY '0C0H', /* USART AND INT CONT NORMAL */

```

```

PIS$PORTC$STAT LITERALLY '0CCH', /*PORT C STATUS*/
PIS$PORTC$CTL  LITERALLY '0CEH', /*PORT C CONTROL*/
PIS$M2M1      LITERALLY '0C6H', /*A-MODE 1, B-MODE 2*/
PIS$ORF       LITERALLY '080H', /*OUTPUT BUFFER READY*/
PIS$IBF       LITERALLY '02H'; /*INPUT BUFFER READY*/

```

```

37  1  DECLARE /*8259 INTERRUPT CONTROLLER*/
        IC$PORTA LITERALLY '0E6H', /* PORT A */
        IC$PORTB LITERALLY '0E7H', /* PORT B */
        IC$ICW1  LITERALLY '56H', /* INIT COMMAND WORD 1,
        (A7A6A5) = 010; EDGE TRIG;
        INTERVAL = 4; SINGLE; NO ICW4 */
        IC$ICW2  LITERALLY '00H', /* INIT COMMAND WORD 2,
        (A15-A8) = 0 */
        IC$ICW3  LITERALLY '00H', /* INIT COMMAND WORD 3,
        NO SLAVE IN IR */
        IC$ICW4  LITERALLY '0CH', /* INIT COMMAND WORD 4, NOT USED */
        IC$MASK  LITERALLY '0', /* INTERRUPT MASK, ICW1, NOT USED;
        0 = ALL INTERRUPTS ENABELLED */
        INIT$MASK LITERALLY '10101010B',
        /* INITIAL INTERRUPT MASK, ICW1;
        RECEIVE INTR ON, TRANSMIT INTR OFF */
        IC$EOI  LITERALLY '0A0H', /* END OF INTERRUPT CMD, ICW2,
        ROTATE (PRIORITY) ON NON-SPECIFIC EOI */
        IC$OCW3IR LITERALLY '0AH', /* READ IRR LEVEL, ICW3,
        NO SPECIAL MASK MODE; NO POLL;
        READ IR REG (NOT USED) */
        IC$OCW3IS LITERALLY '0BH', /* READ ISR LEVEL, ICW3,
        NO SPECIAL MASK MODE; NO POLL;
        READ IS REG (NOT USED) */
        IC$OCW3$SMMS LITERALLY '60H', /* SPECIAL MASK MODE SET */
        IC$OCW3$SMR LITERALLY '40H'; /* SPECIAL MASK MODE RESET */

```

```

/*****
/*
/* PROCEDURE INITIALIZE$BOARD INITIALIZES THE HARDWARE PORTS
/*

```

```

/* THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE ALL THE HARDWARE */
/* PORTS, TO INCLUDE THE PARALLEL PORTS, THE CLOCK/TIMERS, THE */
/* INTERRUPT CONTROLLER, AND THE FOUR USARTS. PORT 0 IS USED TO */
/* COMMUNICATE WITH AN EXTERNAL TERMINAL. PORTS 1, 2, AND 3 ARE */
/* USED TO COMMUNICATE WITH EXTERNAL HOSTS. */
/* */
/* INPUT - NONE */
/* PROCESSING - SETS THE VECTOR INTERRUPT TABLE VALUES AND WRITES THE */
/* CONTROL WORD TO THE PIC, PIT, PPI, AND USARTS. */
/* OUTPUT - NONE */
/* INTERFACE - CALLED BY MAIN PROGRAM. */
/* */
/*****

/* INITIALIZE 8155 PROGRAMMABLE PERIPHERAL INTERFACE (PPI) */
/* SET DATA RATES FOR LOCAL CHANNEL USARTS: */
/* SET PIT #1, REGISTER #0 FOR USART #0 DATA RATE */
/* SET PIT #1, REGISTER #1 FOR USART #1 DATA RATE */
/* SET PIT #1, REGISTER #2 FOR USART #2 DATA RATE */
/* SET PIT #2, REGISTER #0 FOR USART #3 DATA RATE */
/* INITIATE RESET ON ALL USARTS */
/* INITIALIZE THE FOUR USARTS TO: 2 STOP BITS, NO PARITY, */
/* 8 BIT CHARACTERS, AND 16X DATA RATE */
/* INITIALIZE THE 8259 PROGRAMMABLE INTERRUPT CONTROLLER */

/*****

* DEVICE INITIALIZATION SECTION *
*****/

38 1  INITIALIZE$BOARD:
    /* THIS PROCEDURE INITIALIZES THE ISBC 544 BOARD */
    PROCEDURE;

39 2  PI$INIT:      /* 8155 PARALLEL INTERFACE INITIALIZATION */
    OUTPUT(PI$CMD) = PI$INIT$CMD1;
40 2  OUTPUT(PI$PORTA) = PI$INIT$US$INT1; /* RESETS THE 8251 USARTS */
41 2  OUTPUT(PI$PORTA) = PI$INIT$US$INT2;
42 2  OUTPUT(PI$CNTR$LO) = PI$CNTR$LOCNT;
43 2  OUTPUT(PI$CNTR$HI) = PI$CNTR$HICNT;
44 2  OUTPUT(PI$CMD) = PI$INIT$CMD2;

45 2  IT$INIT:      /* 8253 INTERVAL TIMER INITIALIZATION */
    BRFO = B9600;
46 2  BRF1 = B9600;
47 2  BRF2 = B9600;
48 2  BRF3 = B9600;
49 2  OUTPUT(IT1$CNT) = USART$CNTR$M3; /* USART 0 */
50 2  OUTPUT(IT1$CNTR0) = LOW(BRF0);
51 2  OUTPUT(IT1$CNTR0) = HIGH(BRF0);
52 2  OUTPUT(IT1$CNT) = 40H OR USART$CNTR$M3; /* USART 1 */
53 2  OUTPUT(IT1$CNTR1) = LOW(BRF1);

```

```

54 2      OUTPUT(IT1$CNTR1) = HIGH(BRF1);
55 2      OUTPUT(IT1$CONT) = 80H OR USART$CNTR$M3;    /* USART 2 */
56 2      OUTPUT(IT1$CNTR2) = LOW(BRF2);
57 2      OUTPUT(IT1$CNTR2) = HIGH(BRF2);
58 2      OUTPUT(IT2$CONT) = USART$CNTR$M3;          /* USART 3 */
59 2      OUTPUT(IT2$CNTR0) = LOW(BRF3);
60 2      OUTPUT(IT2$CNTR0) = HIGH(BRF3);

61 2      IC$INIT:      /* 8259 INTERRUPT CONTROLLER INITIALIZATION */

/*THE FOLLOWING CODE INITIALIZES THE 8259A. THE INTERRUPTS ARE
USED FOR THIS PROGRAM. ALL INTERRUPTS ARE ON EXCEPT FOR THE RST TYPES */

      OUTPUT(IC$PORTA) = IC$ICW1;
62 2      OUTPUT(IC$PORTB) = IC$ICW2;
63 2      OUTPUT(IC$PORTB) = INIT$MASK;    /* OCW1 */
64 2      OUTPUT(IC$PORTA) = IC$EOI;      /* OCW2 */
65 2      OUTPUT(IC$PORTA) = IC$OCW3IS;   /* OCW3 */
66 2      ISR$STAT = INPUT(IC$PORTA);

67 2      US$INIT:      /* INITIALIZE THE USARTS */
                        /* ASYNC MODE, 8 DATA BITS, 1 STOP BIT, NO PARITY, x16 CLOCK
                        DTR, RTS ON, RECV ON, TRANSMIT OFF */
      OUTPUT(US$P0$CMD) = US$MODE;
68 2      OUTPUT(US$P0$CMD) = USARTON;
69 2      OUTPUT(US$P1$CMD) = US$MODE;
70 2      OUTPUT(US$P1$CMD) = USARTON;
71 2      OUTPUT(US$P2$CMD) = US$MODE;
72 2      OUTPUT(US$P2$CMD) = USARTON;
73 2      OUTPUT(US$P3$CMD) = US$MODE;
74 2      OUTPUT(US$P3$CMD) = USARTON;

75 2      END INITIALIZE$BOARD;

/******
PROCEDURE INIT      INITIALIZES THE TABLE POINTERS AND VARIABLES

PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE INTERRUPT JUMP TABLE, AND
      OTHER DATA TABLE VARIABLES AND BOOLEAN FLAGS

INPUT - NONE

PROCESSING - THE INTERRUPT JUMP TABLE IS INITIALLIZED ALONG WITH THE DATA
      TABLE AND BOOLEAN FLAG VARIABLES

OUTPUT - INITIALIZED INTERRUPT TABLE AND DATA TABLE VARIABLES

INTERFACE - CALLED BY MAIN PROCEDURE

*****
76 1      INIT: PROCEDURE;

```

```
77 2      DECLARE I BYTE;
```

```
/* THE FOLLOWING CODE INITIALIZES THE JUMP TABLE IN 8155 RAM TO POINT
TO THE USER'S INTERRUPT SERVICE ROUTINES AT INTR$VECTOR$5$5, ETC. */
```

```
/* ADDR = .INTR$VECTOR$5$5;
VECTOR$TABLE = JUMP$CODE;
ADDR = ADDR + 1;
VECTOR$TABLE = LOW(VECTOR$TABLE$BASE);
ADDR = ADDR + 1;
VECTOR$TABLE = HIGH(VECTOR$TABLE$BASE);

ADDR = .INTR$VECTOR$6$5;
VECTOR$TABLE = JUMP$CODE;
ADDR = ADDR + 1;
VECTOR$TABLE = LOW(VECTOR$TABLE$BASE + 3);
ADDR = ADDR + 1;
VECTOR$TABLE = HIGH(VECTOR$TABLE$BASE + 3);

ADDR = .INTR$VECTOR$7$5;
VECTOR$TABLE = JUMP$CODE;
ADDR = ADDR + 1;
VECTOR$TABLE = LOW(VECTOR$TABLE$BASE + 6);
ADDR = ADDR + 1;
VECTOR$TABLE = HIGH(VECTOR$TABLE$BASE + 6);
*/
```

```
/* INITIALIZES 8155 JUMP TABLE TO JMP 0 */
```

```
78 2      I = 0;
79 2      ADDR = VECTOR$TABLE$BASE;      /* = 7FF4H */
80 2      DO WHILE I <= 11;
81 3          VECTOR$TABLE = JUMP$CODE;
82 3          ADDR = ADDR + 1;
83 3          VECTOR$TABLE = 0;
84 3          ADDR = ADDR + 1;
85 3          VECTOR$TABLE = 0;
86 3          ADDR = ADDR + 1;
87 3          I = I + 3;
88 3      END;

89 2      BYTES$SENT$1 = 0;
90 2      BYTES$SENT$2 = 0;
91 2      BYTES$SENT$3 = 0;
92 2      BYTES$SENT$4 = 0;

93 2      BYTES$RECV$1 = 0;
94 2      BYTES$RECV$2 = 0;
95 2      BYTES$RECV$3 = 0;
96 2      BYTES$RECV$4 = 0;
```

```

97  2      SEND$1 = FALSE;
98  2      SEND$2 = FALSE;
99  2      SEND$3 = FALSE;
100 2      SEND$4 = FALSE;

101 2      TRTA$1,TXTR$1,XTA$1,RXTR$1,RXTA$1 = FALSE;
102 2      TRTA$2,TXTR$2,XTA$2,RXTR$2,RXTA$2 = FALSE;
103 2      TRTA$3,TXTR$3,XTA$3,RXTR$3,RXTA$3 = FALSE;
104 2      TRTA$4,TXTR$4,XTA$4,RXTR$4,RXTA$4 = FALSE;

105 2      CHAR$1 = '1';
106 2      CHAR$2 = '2';
107 2      CHAR$3 = '3';
108 2      CHAR$4 = '4';
109 2      CHAR$X = 'X';

110 2      END INIT;

      /*****
      /* PROCEDURE INIT$TAB  INITIALIZES THE LOCAL TABLES AND PROGRAM
      /*                      VARIABLES
      /*
      /* THIS PROCEDURE SETS THE LOCAL AND NETWORK TABLES TO THEIR
      /*      INITIAL VALUES PRIOR TO PROCESSING ANY MESSAGES. IN ADDITION,
      /*      THE PROGRAM VARIABLES ARE SET TO THEIR INITIAL VALUES.
      /*
      /* INPUTS - NONE
      /*
      /* PROCESSING - PUTS INITIAL VALUES IN THE LOCAL TABLES
      /*
      /* OUTPUT - NONE
      /*
      /* INTERFACE - CALLED BY THE MAIN PROCEDURE
      /*
      /*
      /* *****/

111 1      INIT$TAB: PROCEDURE;
112 2      DECLARE IX BYTE;

113 2      LSEM$1 = DONE; /* INITIALIZATION OF THE SEMAPHORE FLAGS */
114 2      LSEM$2 = DONE;
115 2      LSEM$3 = DONE;
116 2      LSEM$4 = DONE;

117 2      NSEM$1 = DONE;
118 2      NSEM$2 = DONE;

119 2      LSPARE$1 = 0; /* THE SPARE ADDRESS LOCATIONS SHOULD BE SET TO THE
120 2      LSPARE$2 = 0; /* SEGMENT NUMBER INTO WHICH THE LOCAL BOARD MEMORY
121 2      LSPARE$3 = 0; /* WILL BE MAPPED INTO SYSTEM MEMORY. THIS VALUE IS
122 2      LSPARE$4 = 0; /* CURRENTLY 0 FOR THE LSPARE$X VARIABLES.

```

```

123 2      NSPARE$1 = 0;  /* SAME AS FOR LSPARE$X DISCUSSION ABOVE      */
124 2      NSPARE$2 = 0;

125 2      LPTR$1 = 0;
126 2      LPTR$2 = 0;
127 2      LPTR$3 = 0;
128 2      LPTR$4 = 0;

129 2      NPTR$1 = 0;
130 2      NPTR$2 = 0;

131 2      NT01NS = 0;
132 2      NT01NE = 0;
133 2      NT01SZ = FRAME$TABLE$SIZE;

134 2      NT02NS = 0;
135 2      NT02NE = 0;
136 2      NT02SZ = FRAME$TABLE$SIZE;

137 2      LC01NS = 0;
138 2      LC01NE = 0;
139 2      LC01SZ = PACKET$TABLE$SIZE;

140 2      LC02NS = 0;
141 2      LC02NE = 0;
142 2      LC02SZ = PACKET$TABLE$SIZE;

143 2      LC03NS = 0;
144 2      LC03NE = 0;
145 2      LC03SZ = PACKET$TABLE$SIZE;

146 2      LC04NS = 0;
147 2      LC04NE = 0;
148 2      LC04SZ = PACKET$TABLE$SIZE;

149 2      TX01NS = 0;
150 2      TX01NE = 0;
151 2      TX01SZ = DATA$TABLE$SIZE;

152 2      TX02NS = 0;
153 2      TX02NE = 0;
154 2      TX02SZ = DATA$TABLE$SIZE;

155 2      TX03NS = 0;
156 2      TX03NE = 0;
157 2      TX03SZ = DATA$TABLE$SIZE;

158 2      TX04NS = 0;
159 2      TX04NE = 0;
160 2      TX04SZ = DATA$TABLE$SIZE;

```

```

161 2      IX = 0;
162 2      DO WHILE IX < STAT$NBR;
163 3          STATB(IX) = 0;
164 3          IX = IX + 1;
165 3      END;

166 2      END INIT$TAB;

      /*****
      /* PROCEDURE  SNDSEQ  SENDS DATA TO LOCAL MONITOR FOR TESTING  */
      /*                                                    */
      /* THIS PROCEDURE TAKES A MESSAGE STRING AND OUTPUTS IT TO  */
      /* THE LOCAL MONITOR ATTACHED TO PORT 0 OF THE 544 BOARD  */
      /*                                                    */
      /* INPUT - A POINTER TO THE MESSAGE LOCATION IN MEMORY AND THE  */
      /* NUMBER OF BYTES TO BE SENT.  */
      /* PROCESSING - THIS PROCEDURE CHECKS THE OUTPUT BUFFER STATUS  */
      /* IN A LOOP UNTIL THE BUFFER IS EMPTY. IT PLACES ONE  */
      /* BYTE AT A TIME IN THE OUTPUT USART UNTIL THE MESSAGE  */
      /* IS DONE.  */
      /* OUTPUT - MESSAGE TO THE USART 0 PORT.  */
      /* INTERFACE - THIS PROCEDURE IS CALLED BY VARIOUS PROCEDURES.  */
      /*                                                    */
      *****/

167 1      SNDSEQ: PROCEDURE(MSG, TOTAL);
168 2          DECLARE MSG ADDRESS,
169 2          CHAROUT BASED MSG(1) BYTE;
170 2          DECLARE (COUNT, TOTAL) BYTE;

170 2          COUNT = 0;
171 2      LOOP: DO WHILE COUNT < TOTAL;
172 3          DO WHILE NOT (INPUT(US$PO$STAT) AND 1); END;
174 3          OUTPUT(US$PO$DATA) = CHAROUT(COUNT);
175 3          COUNT = COUNT + 1;
176 3      END LOOP;

177 2      END SNDSEQ;

      /*****
      PROCEDURE  LD$TAB$H$K$P      LOAD TABLE HOUSEKEEP

      THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED BUFFER
      TABLE AFTER LOADING OF THE USER DATA FROM THE HOST.

      INPUT - THE INPUT IS A BYTE INDICATING THE TABLE REQUIRING
      CHANGES.

      PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
      ADVANCES THE NEXT$EMPTY$BYTE ADDRESS BY ONE PACKET, DATAGRAM,

```

OR FRAME AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT\$EMPTY\$BYTE ADDRESS
ADVANCED BY THE LENGTH OF A SINGLE PACKET, DATAGRAM OR FRAME.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE VARIOUS PROCEDURES.

```

*****/
178 1 LD$TAB$H$K$P: PROCEDURE(TABLE) ;
179 2 DECLARE TABLE BYTE;

180 2 IF (TABLE >= 1 AND TABLE <= 10) THEN /* 8 FOR REAL, 10 FOR SIM */
181 2 DO CASE TABLE;

182 3 ; /* CASE ZERO IS NULL */

/* CASES 1, 2, 3, 4 FOR LC01(2,3,4) ARE NOT REQUIRED AS */
/* THE LCOXNS POINTERS ARE INCREMENTED AND CHECKED IN THE */
/* RECEIVE DATAGRAM INTERRUPT ROUTINES. USE FOR SIMULATION */
/* PURPOSES ONLY. CASE 9 IS FOR SIMULATION ONLY */

183 3 DO;
184 4 LC01NE = LC01NE + PACKET$SIZE;
185 4 IF LC01NE >= LC01SZ THEN
186 4 LC01NE = 0;
187 4 END;

188 3 DO;
189 4 LC02NE = LC02NE + PACKET$SIZE;
190 4 IF LC02NE >= LC02SZ THEN
191 4 LC02NE = 0;
192 4 END;

193 3 DO;
194 4 LC03NE = LC03NE + PACKET$SIZE;
195 4 IF LC03NE >= LC03SZ THEN
196 4 LC03NE = 0;
197 4 END;

198 3 DO;
199 4 LC04NE = LC04NE + PACKET$SIZE;
200 4 IF LC04NE >= LC04SZ THEN
201 4 LC04NE = 0;
202 4 END;

203 3 DO;
204 4 TX01NE = TX01NE + DATA$GRAM$SIZE;
205 4 IF TX01NE >= TX01SZ THEN
206 4 TX01NE = 0;
207 4 END;

```

```

208 3      DO;
209 4          TX02NE = TX02NE + DATA$GRAM$SIZE;
210 4          IF TX02NE >= TX02SZ THEN
211 4              TX02NE = 0;
212 4      END;

213 3      DO;
214 4          TX03NE = TX03NE + DATA$GRAM$SIZE;
215 4          IF TX03NE >= TX03SZ THEN
216 4              TX03NE = 0;
217 4      END;

218 3      DO;
219 4          TX04NE = TX04NE + DATA$GRAM$SIZE;
220 4          IF TX04NE >= TX04SZ THEN
221 4              TX04NE = 0;
222 4      END;

223 3      DO;
224 4          NT01NE = NT01NE + FRAME$SIZE;
225 4          IF NT01NE >= NT01SZ THEN
226 4              NT01NE = 0;
227 4      END;

228 3      DO;
229 4          NT02NE = NT02NE + FRAME$SIZE;
230 4          IF NT02NE >= NT02SZ THEN
231 4              NT02NE = 0;
232 4      END;

233 3      END;

      ELSE
234 2      DO;
235 3          STATTB(7) = STATTB(7) + 1;
236 3          STATTB(0) = STATTB(0) + 1;
237 3      END;

238 2      END LD$TAB$H$SKP;

/*****
PROCEDURE  SRVC$TAB$H$SKP      SERVICE TABLE HOUSEKEEP

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED
BUFFER TABLE AFTER SERVICING (REMOVING A PACKET).

INPUT - THE INPUT IS A BYTE VALUE INDICATING THE TABLE THAT
REQUIRES HOUSEKEEPING.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT$BYTE$TO$BE$SERVICED ADDRESS BY A PACKET, DATAGRAM,

```

OR A FRAME AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT\$BYTE\$TO\$RE\$SERVICED ADDRESS
ADVANCED BY THE LENGTH OF A SINGLE PACKET, DATAGRAM OR FRAME.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN AND ROUTE\$OUT.

*****/

```

239 1  SRVC$TAB$H$K: PROCEDURE(TABLE) ;
240 2  DECLARE TABLE BYTE;

241 2      IF (TABLE >= 1 AND TABLE <= 10) THEN      /* 8 FOR REAL, 10 FOR SIM */
242 2          DO CASE TABLE;

243 3              ;                      /* CASE ZERO IS NULL */

244 3          DO;
245 4              LC01NS = LC01NS + PACKET$SIZE;
246 4              IF LC01NS >= LC01SZ THEN
247 4                  LC01NS = 0;
248 4          END;

249 3          DO;
250 4              LC02NS = LC02NS + PACKET$SIZE;
251 4              IF LC02NS >= LC02SZ THEN
252 4                  LC02NS = 0;
253 4          END;

254 3          DO;
255 4              LC03NS = LC03NS + PACKET$SIZE;
256 4              IF LC03NS >= LC03SZ THEN
257 4                  LC03NS = 0;
258 4          END;

259 3          DO;
260 4              LC04NS = LC04NS + PACKET$SIZE;
261 4              IF LC04NS >= LC04SZ THEN
262 4                  LC04NS = 0;
263 4          END;

          /* CASES 5, 6, 7, 8 FOR TX01(2,3,4) ARE NOT REQUIRED AS */
          /* THE TX0XNS POINTERS ARE INCREMENTED AND CHECKED IN THE */
          /* TRANSMIT DATAGRAM INTERRUPT ROUTINES. USE FOR SIMULATION */
          /* PURPOSES ONLY. CASE 9 IS FOR SIMULATION ONLY          */

264 3          DO;
265 4              TX01NS = TX01NS + DATA$GRAM$SIZE;
266 4              IF TX01NS >= TX01SZ THEN
267 4                  TX01NS = 0;
268 4          END;

```

```

269 3      DO;
270 4          TX02NS = TX02NS + DATA$GRAM$SIZE;
271 4          IF TX02NS >= TX02SZ THEN
272 4              TX02NS = 0;
273 4      END;

274 3      DO;
275 4          TX03NS = TX03NS + DATA$GRAM$SIZE;
276 4          IF TX03NS >= TX03SZ THEN
277 4              TX03NS = 0;
278 4      END;

279 3      DO;
280 4          TX04NS = TX04NS + DATA$GRAM$SIZE;
281 4          IF TX04NS >= TX04SZ THEN
282 4              TX04NS = 0;
283 4      END;

284 3      DO;
285 4          NT01NS = NT01NS + FRAME$SIZE;
286 4          IF NT01NS >= NT01SZ THEN
287 4              NT01NS = 0;
288 4      END;

289 3      DO;
290 4          NT02NS = NT02NS + FRAME$SIZE;
291 4          IF NT02NS >= NT02SZ THEN
292 4              NT02NS = 0;
293 4      END;

294 3      END;

      ELSE
295 2          DO;
296 3              STATTB(8) = STATTB(8) + 1;
297 3              STATTB(0) = STATTB(0) + 1;
298 3          END;

299 2      END SRVC$TAB$H$SKP;

/*****
/* INCLUDES THE INTERRUPT DRIVER ROUTINES FROM A SEPARATE FILE */
*****/

$ INCLUDE(INTR.LOC)

/*****
PROCEDURE RXIN$1      RECEIVES DATA FROM PORT ONE OF FOUR ON THE SRC 544

PURPOSE OF THIS PROCEDURE IS TO RECEIVE A CHARACTER FROM THE SRC TERMINAL

```

INPUT - INTERRUPT FROM THE TERMINAL

PROCESSING - THE RECEIVED CHARACTER IS INPUT FROM USART 0 AND ECHOED
BACK TO THE TERMINAL. THE RECEIVE INTERRUPT IS CLEARED.

OUTPUT - THE RECEIVED CHARACTER IS PUT INTO THE CHAR\$X VARIABLE.

INTERFACE - CALLED BY READ\$LINE PROCEDURE

*****/

```

300 1 = RXIN$1: PROCEDURE INTERRUPT 3;
      =
301 2 =     CHAR$X = INPUT(US$PO$DATA);
302 2 =     OUTPUT(US$PO$DATA) = CHAR$X;
303 2 =     OUTPUT(IC$PORTA) = IC$EOI;
      =
304 2 = END RXIN$1;
      =

```

/******/

PROCEDURE TXOUT\$1 A DUMMY INTERRUPT STUB

PURPOSE OF THIS PROCEDURE IS TO PROVIDE A DUMMY STUB FOR THE TERMINAL
INTERRUPT HANDLER.

INPUT - INTERRUPT FOR THE TERMINAL TRANSMIT

PROCESSING - CLEAR THE TRANSMIT INTERRUPT

OUTPUT - CLEARED INTERRUPT

INTERFACE - NONE

*****/

```

305 1 = TXOUT$1: PROCEDURE INTERRUPT 3;
306 2 =     OUTPUT(IC$PORTA) = IC$EOI;
307 2 = END TXOUT$1;
      =

```

/******/

```

= /*
= /* PROCEDURE SERVICE$RCV$1 RECEIVES DATA ONE CHANNEL ONE */
= /*
= /* THE PURPOSE OF THIS PROCEDURE IS TO TAKE A RECEIVED CHARACTER */
= /* FROM THE RECEIVE PORT ONE AND PUT IT IN THE RECEIVE ONE */
= /* BUFFER. THE TRTA FLAGS ARE CHECKED FOR USE AND THE TRTA */
= /* SET ACCORDINGLY. THIS ROUTINE IS TOTALLY INTERRUPT DRIVEN */
= /* AND OPERATES CONTINUALLY AFTER INITIALIZATION. */
= /*
= /* INPUT - INTERRUPT ON USART 1 */
= /*

```

```

= /*
= /* PROCESSING - MOVES A BYTE OF DATA FROM THE RECEIVE USART TO
= /* MEMORY. THE TRTA FLAGS ARE CHECKED AND APPROPRIATELY SET.
= /*
= /* OUTPUT - RECEIVED CHARACTER AND APPROPRIATELY SET TRTA FLAGS.
= /*
= /* INTERFACE - INTERRUPTS SET DURING INITIALIZATION.
= /* * ACTIVATED BY USART 1 *
= /*
= /* NOTE: THE SAME PROCEDURE IS USED FOR USARTS 2 AND 3 WITH APPROPRIATE
= /* MODIFICATIONS OF THE VARIABLE NAMES.
= /******
=
308 1 = SERVICE$RCV$2: PROCEDURE INTERRUPT 10;
=
309 2 = CHAR$2 = INPUT(US$P1$DATA);
=
310 2 = IF ((NOT TRTA$2) OR ((RXTR$2 AND TXTA$2) AND
= ((NOT TXTR$2) AND (NOT RXTA$2)))) THEN
311 2 = DO;
312 3 = LCO2RX(LCO2NE + 5) = CHAR$2;
313 3 = BYTES$RCV$2 = BYTES$RCV$2 + 1;
314 3 = LCO2NE = LCO2NE + 1;
315 3 = IF BYTES$RCV$2 >= DATA$GRAM$SIZE THEN
316 3 = DO;
317 4 = LCO2NE = LCO2NE + 5;
318 4 = IF (LCO2NE >= LCO2S2) THEN
319 4 = LCO2NE = 0;
320 4 = BYTES$RCV$2 = 0;
321 4 = RXTR$2 = FALSE;
322 4 = TXTA$2 = FALSE;
323 4 = SEND$2 = FALSE;
324 4 = END;
325 3 = END;
=
326 2 = IF TRTA$2 THEN
327 2 = DO;
328 3 = IF CHAR$2 = TA THEN
329 3 = IF ((TXTR$2 AND (NOT RXTA$2)) AND
= ((NOT RXTR$2) AND (NOT TXTA$2))) THEN
330 3 = DO;
331 4 = RXTA$2 = TRUE;
332 4 = SEND$2 = FALSE;
333 4 = CALL SNDSEQ(CTP$30, SIZE(CTP$30));
334 4 = END;
335 3 = IF CHAR$2 = TR THEN
336 3 = IF ((NOT RXTR$2) AND (NOT TXTA$2)) AND
= ((NOT TXTR$2) AND (NOT RXTA$2)) THEN
337 3 = DO;
338 4 = RXTR$2 = TRUE;
339 4 = TXTA$2 = TRUE;

```

AD-A151 947

CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE
UNIVERSAL NETWORK INTERFA. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. C T CHILDRESS
DEC 84 AFIT/GE/ENG/84D-17-VOL-2 F/G 17/2

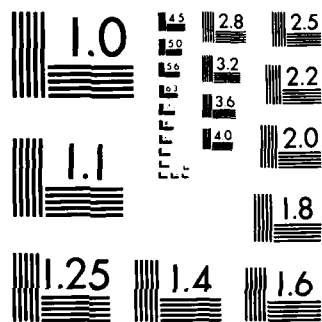
24

UNCLASSIFIED

F/G 17/2

NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

```

340 4 =          SEND$2 = TRUE;
341 4 =          CALL SNDSEQ(.TP$3B, SIZE(TP$3B));
342 4 =          CALL SNDSEQ(.TP$3A, SIZE(TP$3A));
343 4 =          OUTPUT(US$P1$DATA) = TA;
344 4 =          END;
345 3 =          END;
      =
346 2 =          OUTPUT(IC$PORTA) = IC$E01;
      =
347 2 =  END SERVICE$RCV$2;
      =
      =
/*****
=
348 1 =  SERVICE$RCV$3: PROCEDURE INTERRUPT 12;
      =
349 2 =          CHAR$3 = INPUT(US$P2$DATA);
      =
350 2 =          IF ((NOT TRTA$3) OR ((RXTR$3 AND TXTA$3) AND
      =              ((NOT TXTR$3) AND (NOT RXTA$3)))) THEN
351 2 =              DO;
352 3 =              LC03RX(LC03NE + 5) = CHAR$3;
353 3 =              BYTES$RCV$3 = BYTES$RCV$3 + 1;
354 3 =              LC03NE = LC03NE + 1;
355 3 =              IF BYTES$RCV$3 >= DATA$GRAM$SIZE THEN
356 3 =                  DO;
357 4 =                  LC03NE = LC03NE + 5;
358 4 =                  IF (LC03NE >= LC03SZ) THEN
359 4 =                      LC03NE = 0;
360 4 =                  BYTES$RCV$3 = 0;
361 4 =                  RXTR$3 = FALSE;
362 4 =                  TXTA$3 = FALSE;
363 4 =                  SEND$3 = FALSE;
364 4 =                  END;
365 3 =              END;
      =
366 2 =          IF TRTA$3 THEN
367 2 =              DO;
368 3 =              IF CHAR$3 = TA THEN
369 3 =                  IF ((TXTR$3 AND (NOT RXTA$3)) AND
      =                      ((NOT RXTR$3) AND (NOT TXTA$3))) THEN
370 3 =                      DO;
371 4 =                      RXTA$3 = TRUE;
372 4 =                      SEND$3 = FALSE;
373 4 =                      CALL SNDSEQ(.TP$3C, SIZE(TP$3C));
374 4 =                      END;
375 3 =              IF CHAR$3 = TR THEN
376 3 =                  IF (((NOT RXTR$3) AND (NOT TXTA$3)) AND
      =                      ((NOT TXTR$3) AND (NOT RXTA$3))) THEN
377 3 =                      DO;
378 4 =                      RXTR$3 = TRUE;

```

```

379 4 =      TXTA$3 = TRUE;
380 4 =      SEND$3 = TRUE;
381 4 =      CALL SNDSEQ(.TP$3B, SIZE(TP$3B));
382 4 =      CALL SNDSEQ(.TP$3A, SIZE(TP$3A));
383 4 =      OUTPUT(US$P2$DATA) = TA;
384 4 =      END;
385 3 =      END;
      =
386 2 =      OUTPUT(IC$PORTA) = IC$ECI;
      =
387 2 =      END SERVICE$RCV$3;
      =
      =
      /*****
388 1 =      SERVICE$RCV$4: PROCEDURE INTERRUPT 14;
      =
389 2 =      CHAR$4 = INPUT(US$P3$DATA);
      =
390 2 =      IF ((NOT TRTA$4) OR ((RXTR$4 AND TXTA$4) AND
      =          ((NOT TXTR$4) AND (NOT RXTA$4)))) THEN
391 2 =          DO;
392 3 =          LCO4RX(LCO4NE + 5) = CHAR$4;
393 3 =          BYTES$RECV$4 = BYTES$RECV$4 + 1;
394 3 =          LCO4NE = LCO4NE + 1;
395 3 =          IF BYTES$RECV$4 >= DATA$GRAM$SIZE THEN
396 3 =              DO;
397 4 =              LCO4NE = LCO4NE + 5;
398 4 =              IF (LCO4NE >= LCO4SZ) THEN
399 4 =                  LCO4NE = 0;
400 4 =              BYTES$RECV$4 = 0;
401 4 =              RXTR$4 = FALSE;
402 4 =              TXTA$4 = FALSE;
403 4 =              SEND$4 = FALSE;
404 4 =              END;
405 3 =          END;
      =
406 2 =      IF TRTA$4 THEN
407 2 =          DO;
408 3 =          IF CHAR$4 = TA THEN
409 3 =              IF ((TXTR$4 AND (NOT RXTA$4)) AND
      =                  ((NOT RXTR$4) AND (NOT TXTA$4))) THEN
410 3 =                  DO;
411 4 =                  RXTA$4 = TRUE;
412 4 =                  SEND$4 = FALSE;
413 4 =                  CALL SNDSEQ(.TP$3C, SIZE(TP$3C));
414 4 =                  END;
415 3 =          IF CHAR$4 = TR THEN
416 3 =              IF ((NOT RXTR$4) AND (NOT TXTA$4)) AND
      =                  ((NOT TXTR$4) AND (NOT RXTA$4))) THEN
417 3 =                  DO;

```

```

418 4 =          RXTR$4 = TRUE;
419 4 =          TXTA$4 = TRUE;
420 4 =          SEND$4 = TRUE;
421 4 =          CALL SNDSEQ(.TP$3B, SIZE(TP$3B));
422 4 =          CALL SNDSEQ(.TP$3A, SIZE(TP$3A));
423 4 =          OUTPUT(US$P3$DATA) = TA;
424 4 =          END;
425 3 =          END;
      =
426 2 =          OUTPUT(IC$PORTA) = IC$E01;
      =
427 2 =          END SERVICE$RCV$4;
      =
      = /*****
      = /*  PROCEDURE  SERVICE$TRANS$1  SENDS DATA OUT CHANNEL ONE          */
      = /*  */
      = /*  THE PURPOSE OF THIS PROCEDURE IS TO SEND A DATAGRAM OF DATA OUT  */
      = /*  LOCAL CHANNEL ONE. A SINGLE BYTE IS TRANSMITTED EACH TIME AN     */
      = /*  INTERRUPT IS GENERATED BY USART ONE ON THE TRANSMIT SIDE.        */
      = /*  */
      = /*  INPUT - INTERRUPT MASKED ON BY CALLING PROCEDURE                  */
      = /*  */
      = /*  PROCESSING - SENDS A BYTE OF DATA FROM THE TRANSMIT ARRAY TO    */
      = /*  THE DATA PORT.  WHEN MESSAGE IS DONE IT RESETS                  */
      = /*  THE TRANSMIT INTERRUPT AND SETS TRANS$1$RDY TO TRUE.              */
      = /*  */
      = /*  OUTPUT - CHARACTER OUTPUT USART 1                                */
      = /*  */
      = /*  INTERFACE - CALLED BY                                             */
      = /*  */
      = /*  NOTE: THE SAME PROCEDURE IS USED FOR USARTS 2 AND 3 WITH APPROPRIATE*/
      = /*  MODIFICATIONS OF THE VARIABLE NAMES.                             */
      = /*  */
      = /*****
428 1 =          SERVICE$TRANS$2: PROCEDURE INTERRUPT 11;
      =
429 2 =          IF ((NOT TRTA$2) OR ((TXTR$2 AND RXTA$2) AND
      =              ((NOT RXTR$2) AND (NOT TXTA$2)))) THEN
430 2 =              DO;
431 3 =              OUTPUT(US$P1$DATA) = LCO2IX(TX02NS);
432 3 =              BYTES$SENT$2 = BYTES$SENT$2 + 1;
433 3 =              TX02NS = TX02NS + 1;
434 3 =              IF BYTES$SENT$2 >= DATA$GRAM$SIZE THEN
435 3 =                  DO;
436 4 =                  OUTPUT(IC$PORTB) = INPUT(IC$PORTB) OR B;
437 4 =                  BYTES$SENT$2 = 0;
438 4 =                  IF TX02NS >= TX02NS THEN
439 4 =                      TX02NS = 0;
440 4 =                  TXTR$2 = FALSE;
441 4 =                  RXTA$2 = FALSE;
442 4 =                  SEND$2 = FALSE;

```

```

443 4 =          CALL SNDSEQ(.TP$4A, SIZE(TP$4A));
444 4 =          END;
445 3 =          END;
      =
446 2 =          OUTPUT(IC$PORTA) = IC$E01;
      =
447 2 =          END SERVICE$TRANS$2;
      =
      =
      = /*****/
      =
448 1 = SERVICE$TRANS$3: PROCEDURE INTERRUPT 13;
      =
449 2 =          IF ((NOT TRTA$3) OR ((TXTR$3 AND RXTA$3) AND
      =              ((NOT RXTR$3) AND (NOT TXTA$3)))) THEN
450 2 =              DO;
451 3 =              OUTPUT(US$P2$DATA) = LC03TX(TX03NS);
452 3 =              BYTES$SENT$3 = BYTES$SENT$3 + 1;
453 3 =              TX03NS = TX03NS + 1;
454 3 =              IF BYTES$SENT$3 >= DATA$GRAM$SIZE THEN
455 3 =                  DO;
456 4 =                  OUTPUT(IC$PORTB) = INPUT(IC$PORTB) OR 20H;
457 4 =                  BYTES$SENT$3 = 0;
458 4 =                  IF TX03NS >= TX03SZ THEN
459 4 =                      TX03NS = 0;
460 4 =                  TXTR$3 = FALSE;
461 4 =                  RXTA$3 = FALSE;
462 4 =                  SEND$3 = FALSE;
463 4 =                  CALL SNDSEQ(.TP$4A, SIZE(TP$4A));
464 4 =                  END;
465 3 =              END;
      =
466 2 =          OUTPUT(IC$PORTA) = IC$E01;
      =
467 2 =          END SERVICE$TRANS$3;
      =
      =
      = /*****/
      =
468 1 = SERVICE$TRANS$4: PROCEDURE INTERRUPT 15;
      =
469 2 =          IF ((NOT TRTA$4) OR ((TXTR$4 AND RXTA$4) AND
      =              ((NOT RXTR$4) AND (NOT TXTA$4)))) THEN
470 2 =              DO;
471 3 =              OUTPUT(US$P3$DATA) = LC04TX(TX04NS);
472 3 =              BYTES$SENT$4 = BYTES$SENT$4 + 1;
473 3 =              TX04NS = TX04NS + 1;
474 3 =              IF BYTES$SENT$4 >= DATA$GRAM$SIZE THEN
475 3 =                  DO;
476 4 =                  OUTPUT(IC$PORTB) = INPUT(IC$PORTB) OR 80H;
477 4 =                  BYTES$SENT$4 = 0;

```

```

478 4 =          IF TX04NS >= TX04SZ THEN
479 4 =              TX04NS = 0;
480 4 =              TXTR$4 = FALSE;
481 4 =              RXTA$4 = FALSE;
482 4 =              SEND$4 = FALSE;
483 4 =              CALL SNDSEQ(.TP$4A, SIZE(TP$4A));
484 4 =              END;
485 3 =          END;

```

```

486 2 =          OUTPUT(IC$PORTA) = IC$E01;

```

```

487 2 =          END SERVICE$TRANS$4;

```

```

/*****
/* PROCEDURE SEND$PACKET PROCEDURE FOR TRANSFORMING THE USER DATAGRAM */
/* INTO A PACKET FOR TRANSFER INTO THE NETWORK */
/* SIDE OF THE UNID. */
/*
/* THE PURPOSE OF THIS PROCEDURE IS TO TRANSFORM THE HOST'S
/* DATAGRAM DELIVERED TO ONE OF THE LOCAL INPUT BUFFERS INTO A
/* PACKET AND UPDATE THE POINTERS AND SEMAPHORES.
/* THE PROCEDURE ADDS THE FIVE HEADER BYTES, AS FOLLOWS:
/* 1. 1 BYTE FOR THE DESTINATION ADDRESS.
/* 2. 1 BYTE FOR THE SOURCE ADDRESS.
/* 3. 1 BYTE FOR THE SEQUENCE NUMBER.
/* 4. 1 BYTE FOR A SPARE (SPARE$01).
/* 5. 1 BYTE FOR A SPARE (SPARE$02).
/*
/* INPUT - THIS PROCEDURE RECEIVES A POINTER THAT INDICATES THE
/* LOCAL INPUT BUFFER WHERE THE INCOMING HOST DATA IS
/* LOCATED AND A VARIABLE THAT INDICATES FROM WHICH PORT THE
/* DATA CAME.
/*
/* PROCESSING - THE PROCEDURE BEGINS WITH THE PASSING OF THE TABLE
/* WHERE THE HOST'S DATA IS LOCATED. THE SOURCE AND
/* DESTINATION ADDRESS (SOURCE$ADDRESS, DESTINATION$ADDRESS)
/* ARE SUPPLIED BY THE DET$ADDR PROCEDURE. FOR NOW, THE
/* SEQUENCE NUMBER AND BOTH SPARE BYTES ARE SET TO ZERO. IN
/* THE FUTURE, THESE BYTES WILL REFLECT X.25 PROTOCOL USES.
/*
/* OUTPUT - THIS PROCEDURE PLACES THE FIRST FIVE BYTES INTO THE
/* LOCAL HOST RECEIVE BUFFER TO CREATE A PACKET. THE PACKET
/* IS POINTED TO BY LPTR AND THE SEMAPHORE LSEM IS SET.
/*
/* INTERFACE - THE PROCEDURE IS CALLED FROM PROCEDURE ROUTE$IN FOR
/* THOSE DATA PACKETS DESTINED FOR THE NETWORK ONLY.
/*
/* NOTE: THE HEADER INFORMATION SUPPLIED IS FOR DATAGRAM SERVICE
/* ONLY. ALSO, THE SEQUENCE AND SPARE BYTES ARE SET TO ZERO
*/

```

```

/*      TO ALLOW THE UNIDS MINIMAL OPERATIONAL CAPABILITY.  FOR      */
/*      FUTURE SOFTWARE ENHANCEMENTS THIS WILL BE MODIFIED.          */
/*                                                                      */
/*****
488 1  SEND$PACKET: PROCEDURE(TABLE$PTR, PORT);
489 2      DECLARE TABLE$PTR ADDRESS,
          (PORT) BYTE,
          LCOXTB BASED TABLE$PTR (1) BYTE;

490 2      LCOXTB( 0) = DESTINATION$ADDRESS;
491 2      LCOXTB( 1) = SOURCE$ADDRESS;
492 2      LCOXTB( 2) = 0;
493 2      LCOXTB( 3) = 0;
494 2      LCOXTB( 4) = 0;

495 2      DO CASE PORT;
496 3          ;

497 3      DO;
498 4          LPTR$1 = TABLE$PTR;
499 4          LSEM$1 = READY;
500 4          END;

501 3      DO;
502 4          LPTR$2 = TABLE$PTR;
503 4          LSEM$2 = READY;
504 4          END;

505 3      DO;
506 4          LPTR$3 = TABLE$PTR;
507 4          LSEM$3 = READY;
508 4          END;

509 3      DO;
510 4          LPTR$4 = TABLE$PTR;
511 4          LSEM$4 = READY;
512 4          END;

513 3      END;  /* CASE */

514 2  END SEND$PACKET;

/*****
PROCEDURE DET$ADDR      DETERMINES THE DESTINATION OF DATA FROM LOCAL HOST

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATAGRAMS
COMING FROM A HOST CONNECTED TO THE LOCAL PORTS 1 - 4.

INPUT - THE INPUT IS A POINTER TO THE DATAGRAM HEADER TO BE INTERPRETED.

```

PROCESSING - THE PROCEDURE FIRST EXTRACTS THE CONTROL\$CODE FROM THE INCOMING DATA TO DETERMINE WHICH ROUTING SCHEME IS USED. THE COUNTRY\$CODE AND NETWORK\$CODE ARE THEN EXTRACTED TO DETERMINE THE DATA'S DESTINATION. THE HOST\$CODE IS EXTRACTED SO THAT THE DESTINATION\$ADDRESS CAN BE DETERMINED. IF THE DATAGRAM IS DESTINED FOR THE NETWORK, THE SOURCE ADDRESS IS ALSO EXTRACTED FOR THE PACKET HEADER.

OUTPUT - THIS PROCEDURE PLACES THE DESTINATION (0 OR 1) IN MEMORY FOR THE PASSING ROUTINE AND RETURNS THE DESTINATION\$ADDRESS. DESTINATION = -1 DENOTES AN ADDRESSING ERROR IN THE DATAGRAM HEADER. THE CALLING PROCEDURE IS EXPECTED TO CHECK FOR DESTINATION = -1. FOR THE ERROR CASE, DESTINATION\$ADDRESS = 0. AN EXAMPLE OF A DESTINATION\$ADDRESS IS 21H, WHICH INDICATES UNITED AND CHANNEL=1.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN.

NOTES: 1. BYTE AND OFH WILL MASK OUT THE UPPER 4-BITS.
2. BYTE AND OFOH WILL MASK OUT THE LOWER 4-BITS.

```

515 1  DET$ADDR: PROCEDURE (TABLE$PTR);

516 2  DECLARE LOBITS BYTE,
        HIBITS BYTE,
        IPCNTL BYTE,
        CONTROL$CODE BYTE,
        COUNTRY$CODE BYTE,
        NETWORK$CODE BYTE,
        HOST$CODE BYTE,
        SRC$HOST$CODE BYTE,
        SRC$NET$CODE BYTE,
        SRC$CONT$CODE BYTE,
        SRC$CONTRY$CODE BYTE,
        TABLE$PTR ADDRESS,
        LCOXRX BASED TABLE$PTR(1) BYTE;

517 2      LOBITS = 0;
518 2      HIBITS = 0;
519 2      IPCNTL = 0;
520 2      CONTROL$CODE = 0;
521 2      COUNTRY$CODE = 0;
522 2      NETWORK$CODE = 0;
523 2      HOST$CODE = 0;
524 2      DESTINATION$ADDRESS = 0;
525 2      SOURCE$ADDRESS = 0;
526 2      DESTINATION = -1;

527 2      IPCNTL = LCOXRX(1) AND 0E0H;
528 2      IF IPCNTL = 0 THEN

```

```

529 2      DO;

530 3      CONTROL$CODE = ROR(LCOXR(16), 4) AND OFH;
531 3      IF CONTROL$CODE = 00 THEN
532 3          DO;
533 4          COUNTRY$CODE = LCOXR(16) AND OFH;
534 4          IF COUNTRY$CODE = THIS$COUNTRY$CODE THEN /* ELSE TO UNID 0 */
535 4              DO;
536 5              NETWORK$CODE = ROR(LCOXR(17), 4) AND OFH;
537 5              IF (NETWORK$CODE <= MAX$NETWORK$CODE) THEN
538 5                  DO;
539 6                  IF (COUNTRY$CODE <> THIS$COUNTRY$CODE) OR
                      ((NETWORK$CODE)<> THIS$UNID$NBR) THEN
540 6                      DESTINATION = 1; /* LOC TO NET */
                      ELSE
541 6                          DESTINATION = 0; /* LOC TO LOC */
542 6                      LOBITS = (ROR(LCOXR(18), 4) AND OFH);
543 6                      HIBITS = (ROL(LCOXR(17), 4) AND OFOH);
544 6                      HOST$CODE = LOBITS OR HIBITS;
545 6                      IF (HOST$CODE >=0) AND (HOST$CODE <= 63) THEN
546 6                          DESTINATION$ADDRESS = (ROL(NETWORK$CODE, 4) AND
                                                              OFOH) OR 1;
                      ELSE
547 6                          IF (HOST$CODE >=64) AND (HOST$CODE <= 127) THEN
548 6                              DESTINATION$ADDRESS = (ROL(NETWORK$CODE, 4) AND
                                                              OFOH) OR 2;
                      ELSE
549 6                          IF (HOST$CODE >= 128) AND (HOST$CODE <= 191) THEN
550 6                              DESTINATION$ADDRESS = (ROL(NETWORK$CODE, 4) AND
                                                              OFOH) OR 3;
                      ELSE
551 6                          IF (HOST$CODE >= 192) AND (HOST$CODE <=255) THEN
552 6                              DESTINATION$ADDRESS = (ROL(NETWORK$CODE, 4) AND
                                                              OFOH) OR 4;
                      END;
                      ELSE
554 5                          DO; /* NOT WITHIN NETWORK CODES FOR THIS$COUNTRY$CODE */
555 6                              STATTB(4) = STATTB(4) + 1;
556 6                          END;
557 5                          END;
                      ELSE
558 4                          DO; /* NOT FOR THIS COUNTRY, SEND TO NETWORK/UNID 0 */
559 5                              STATTR(3) = STATTB(3) + 1;
560 5                          END;
561 4                          END;
                      ELSE
562 3                          DO; /* IT IS AT THIS POINT THAT OTHER X.121 CONTROL CODES
                              SOFTWARE SUPPORT WILL BE INCORPORATED INTO THE NETWORK */
563 4                              STATTB(2) = STATTB(2) + 1;
564 4                          END;

```

```

565 3      END;
          ELSE
566 2      IF IPCNTL = 0C0H THEN
567 2          DO;
          /* IP CONTROL; LOOK AT SUBCODE, CHECK SUBCODE FOR SQUELCH
          OR UNSQUELCH AND ACT ACCORDINGLY */
568 3      END;
          ELSE
569 2          DO;
570 3          DESTINATION = -1;
571 3          STATTB(10) = STATTB(10) + 1;
572 3      END;

573 2      IF DESTINATION = 1 THEN /* LOC TO NET, GET SOURCE INFO */
574 2          DO;
575 3          SRC$NET$CODE = ROR(LCOXR(13), 4) AND OFH;
576 3          IF (SRC$NET$CODE = THIS$UNIT$NBR) THEN
577 3              DO;
578 4              LOBITS = (ROR(LCOXR(14), 4) AND OFH);
579 4              HIBITS = (ROL(LCOXR(13), 4) AND OFH);
580 4              SRC$HOST$CODE = LOBITS OR HIBITS;
581 4              IF (SRC$HOST$CODE >= 0) AND (SRC$HOST$CODE <= 63) THEN
582 4                  SOURCE$ADDRESS = (ROL(SRC$NET$CODE, 4) AND OFH) OR 1;
          ELSE
583 4              IF (SRC$HOST$CODE >= 64) AND (SRC$HOST$CODE <= 127) THEN
584 4                  SOURCE$ADDRESS = (ROL(SRC$NET$CODE, 4) AND OFH) OR 2;
          ELSE
585 4              IF (SRC$HOST$CODE >= 128) AND (SRC$HOST$CODE <= 191) THEN
586 4                  SOURCE$ADDRESS = (ROL(SRC$NET$CODE, 4) AND OFH) OR 3;
          ELSE
587 4              IF (SRC$HOST$CODE >= 192) AND (SRC$HOST$CODE <= 255)
          THEN
588 4                  SOURCE$ADDRESS = (ROL(SRC$NET$CODE, 4) AND OFH)
          OR 4;
          END;
          ELSE
590 3          DO;
591 4          DESTINATION = -1;
592 4          STATTB(9) = STATTB(9) + 1;
593 4          END;

594 5      END;

595 2      IF DESTINATION = -1 THEN
596 2          STATTB(0) = STATTB(0) + 1;

597 2      END DET$ADDR;

```

```

/*****
PROCEDURE DET$ADDR$NL    DETERMINES THE DESTINATION OF DATA FROM THE NETWORK

```

THE PURPOSE OF THIS PROCEDURE IS TO DETERMINE THE DESTINATION OF DATA
COMING FROM THE NETWORK SIDE OF THE UNIT TO A LOCAL HOST.

INPUT - THE INPUT IS A POINTER INDICATING THE TABLE LOCATION OF
THE DATA HEADER TO BE EVALUATED.

PROCESSING - THE PROCEDURE EXTRACTS THE DESTINATION\$ADDRESS FROM THE
SECOND BYTE OF THE DATA PACKET AND RETURNS THE CORRESPONDING
DESTINATION.

OUTPUT - THIS PROCEDURE RETURNS THE DESTINATION (1,2,3, OR 4) OF THE
DATA COMING FROM THE NETWORK TO A LOCAL HOST. A NEGATIVE VALUE
IS AN ERROR CONDITION.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN.

NOTES: 1. BYTE AND 07H WILL MASK OUT THE UPPER 5-BITS.

*****/

```

598 1  DET$ADDR$NL: PROCEDURE(TABLE$PTR) BYTE;

599 2      DECLARE  PORT BYTE,
                TABLE$PTR ADDRESS,
                NTOXRX BASED TABLE$PTR (1) BYTE;

600 2      PORT = NTOXRX(0) AND 0FH;
601 2      IF (PORT >= 1 AND PORT <= 4) THEN
602 2          RETURN PORT;
        ELSE
603 2          DO
604 3              PORT = -1;
605 3              STATTB(1) = STATTB(1) + 1; /* INCREMENT LOCAL ERROR COUNT */
606 3              STATTB(0) = STATTB(0) + 1;
607 3              RETURN PORT;
608 3          END;

609 2  END DET$ADDR$NL;

```

```

/*
/*
/* PROCEDURE  MOVETOSLOCAL   ROUTE A DATAGRAM TO THE LOCAL HOST
/*
/*
/* THE PURPOSE OF THIS PROCEDURE IS TO ROUTE DATAGRAMS FROM A
/*   LOCAL HOST OR FROM THE NETWORK RECEIVE TABLES TO THE CORRECT
/*   LOCAL HOST TRANSMIT CHANNEL.
/*
/*
/* INPUT - A POINTER TO THE DATAGRAM TO BE MOVED AND A PORT NUMBER
/*         TO WHERE THE DATAGRAM IS DESTINED.
/*

```

```

/* PROCESSING - THE PROCEDURE CHECKS EACH INPUT BUFFER'S ADDRESS    */
/* THE TABLE THAT WAS SERVICED (DATAGRAM REMOVED)                   */
/* HAS ITS TABLE POINTERS UPDATED BY SRVC$TAB$H$SKP. WHEN THE      */
/* FRAME FROM THE NETWORK IS TRANSMITTED TO THE HOSTS THE FIRST    */
/* SEVEN BYTES OF HEADER INFORMATION ARE STRIPPED OFF BEFORE      */
/* BEFORE PASSING THE POINTER TO THIS PROCEDURE.                   */
/*                                                                    */
/* OUTPUT - A DATAGRAM IS MOVED TO THE LOCAL HOST TRANSMIT BUFFER  */
/* TRANSMISSION TO THE LOCAL HOST.                                  */
/*                                                                    */
/* INTERFACE - THIS PROCEDURE IS CALLED BY ROUTE$IN.               */
/*                                                                    */
/*****

610 1  MOVE TO $LOCAL: PROCEDURE (TABLE$PTR, PORT);

611 2      DECLARE PORT BYTE,
          TABLE$PTR ADDRESS;

612 2      IF (PORT >= 1 AND PORT <= 4) THEN
613 2          DO;
614 3          DO CASE PORT;

615 4              ; /* CASE ZERO IS NULL */

616 4              CALL MOVE (DATA$GRAM$SIZE, TABLE$PTR, .LC01TX(TX01NE));

617 4              CALL MOVE (DATA$GRAM$SIZE, TABLE$PTR, .LC02TX(TX02NE));

618 4              CALL MOVE (DATA$GRAM$SIZE, TABLE$PTR, .LC03TX(TX03NE));

619 4              CALL MOVE (DATA$GRAM$SIZE, TABLE$PTR, .LC04TX(TX04NE));

620 4          END; /* END CASE */
621 3          CALL LD$TAB$H$SKP (PORT + 4);
622 3          END;
623 2      ELSE DO;
624 3          STATB(L$R1$DEST$ERR) = STATB(L$R1$DEST$ERR) + 1;
625 3          STATB(C) = STATB(C) + 1;
626 3      END;

627 2  END MOVE TO $LOCAL;

/*****
/* PROCEDURE ROUTE$IN ROUTES DATAGRAMS FROM THE HOST OR NETWORK    */
/*                                                                    */
/* THE PURPOSE OF THIS PROCEDURE IS TO ROUTE DATAGRAMS             */
/* FROM THE FOUR LOCAL HOST RECEIVE BUFFERS TO THE NETWORK        */
/* AND FROM THE NETWORK TO THE FOUR LOCAL HOST TRANSMIT BUFFERS.  */
/*                                                                    */
/* INPUT - HOST DATAGRAMS OR NETWORK PACKETS ARE ROUTED VIA EVALUATION */

```

```

/*      OF THE DATAGRAM OR PACKET HEADERS      */
/*      */
/*      PROCESSING - THE PROCEDURE CHECKS EACH OF THE LOCAL INPUT BUFFER'S  */
/*      ADDRESSES FOR HOST DATAGRAMS.  IF ANY HOST DATA IS READY,  */
/*      THE DESTINATION IS DETERMINED VIA PROCEDURE DET$ADDR.  ONCE  */
/*      THE DESTINATION IS DETERMINED THEN THE DATA IS EITHER SENT  */
/*      TO A LOCAL HOST TRANSMIT BUFFER (LOCAL TO LOCAL TRANSFER)  */
/*      VIA PROCEDURE MOVETO$LOCAL OR THE HOST DATA IS CONVERTED  */
/*      INTO A PACKET BY SEND$PACKET AND THEN SENT TO THE NETWORK  */
/*      SIDE OF THE UNIT VIA A POINTER AND SEMAPHORE IN SEND$PACKET.  */
/*      BOTH THE BUFFER TABLE THAT IS LOADED AND THE TABLE THAT IS  */
/*      SERVICED HAVE THEIR ADDRESS HOUSECLEANED BY LD$TAB$H$K$P AND  */
/*      SRVC$TAB$H$K$P.  */
/*      */
/*      OUTPUT - EITHER A DATAGRAM IS MOVED TO A LOCAL HOST TRANSMIT BUFFER  */
/*      OR THE DATAGRAM IN THE LOCAL HOST RECEIVE BUFFER IS CONVERTED  */
/*      TO A PACKET FOR THE NETWORK (DATA LINK LAYER).  */
/*      */
/*      INTERFACE - THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY THE  */
/*      MAIN PROGRAM.  */
/*      */
/*****/

028 1  ROUTE$IN: PROCEDURE;
029 2      DECLARE NETR1 BASED NPTR$1 (1) BYTE,
          NETR2 BASED NPTR$2 (1) BYTE;
          /* NETR1 POINTS TO A PACKET SIZED ENTRY */

030 2      IF ((LC01NE - LC01NS) >= DATA$GRAM$SIZE) OR (LC01NS > LC01NE)) THEN
031 2          DO;
032 3          CALL DET$ADDR(.LC01RX(LC01NS + 5));
033 3          IF DESTINATION = 0 THEN
034 3              DO;
035 4              CALL MOVETO$LOCAL(.LC01RX(LC01NS + 5), (DESTINATION$ADDRESS AND
                                                              OFFH));
036 4              END;
          ELSE
037 3          IF DESTINATION = 1 THEN
038 3              DO;
039 4              IF L$SEN$1 = DONE THEN
040 4                  CALL SEND$PACKET(.LC01RX(LC01NS), 1);
041 4              END;
          ELSE
042 3              DO;
043 4              STATB(L$R1$DEST$ERR) = STATB(L$R1$DEST$ERR) + 1;
044 4              END;
045 3          CALL SRVC$TAB$H$K$P(1);
046 3          END;

047 2      IF ((LC02NE - LC02NS) >= DATA$GRAM$SIZE) OR (LC02NS > LC02NE)) THEN
048 2          DO;

```

```

55 1  DECLARE MSG10(*) BYTE DATA(OR,LF,
      'Timeout on receive TR from UNID');

      /*****
      /*  PROCEDURE  DELAY      CAUSES A VARIABLE DELAY      */
      /*
      /*  THE PURPOSE OF THIS PROCEDURE IS TO ADD DELAY TO PARTS OF THE
      /*  INITIALIZATION PROCEDURE THAT IS TIME DEPENDENT.
      /*
      /*  INPUT - NONE
      /*  PROCESSING - USES BUILT IN PROCEDURE IN A LOOP
      /*  OUTPUT - NONE
      /*  INTERFACE - NONE AT THIS TIME
      *****/

56 1  DELAY: PROCEDURE(MILLISEC);
57 2      DECLARE (I, MILLISEC) BYTE;

58 2      DO I=1 TO MILLISEC;
59 3          CALL TIME(25);/* TIME IS A BUILT IN FUNCTION OF PLM80 WHICH CAUSES*/
                        /* A DELAY BASED ON THE NUMBER IN PARENS */
60 3      END;
61 2  END DELAY;

      /*****
      PROCEDURES TO CONVERT HEX TO ASCII FOR USE WITH DISPLAYING A FRAME
      *****/

62 1  HEX$ASC: PROCEDURE(VAL, I);
63 2      DECLARE (VAL, I) BYTE;

64 2      TEMP1(I) = ASCII(SHR(VAL,4) AND 0FH);
65 2      TEMP1(I+1) = ASCII(VAL AND 0FH);

66 2  END HEX$ASC;

67 1  ASC$HEX: PROCEDURE(C) BYTE;
68 2      DECLARE C BYTE;

69 2      IF (C >= '0' AND C <= '9') THEN
70 2          RETURN (C-30H);
      ELSE
71 2          IF (C >= 'A' AND C <= 'F') THEN
72 2              RETURN (C-37H);
          ELSE
73 2              IF (C >= 'a' AND C <= 'f') THEN
74 2                  RETURN (C-57H);

75 2          RETURN C

76 2  END ASC$HEX;

```

/* 8251 RCV & XMIT INT */

```

34 1  DECLARE      STARTUP$HDR(*) BYTE DATA(CR,LF,
      '          UNID 11 #2 LOCAL OS',CR,LF,
      '          SUPPORTING S/W ON CP/M',CR,LF,
      '          VERS 1.0, 25 SEP 84',CR,LF,
      '          EXECUTING ',CR,LF);

      /* THE FOLLOWING TEST POINTS ARE USED TO FOLLOW THE DATA WITHIN THE UNID */

35 1  DECLARE TP$50(*) BYTE DATA(CR,LF,
      'TP$50          Channel Number = ');
36 1  DECLARE TP$51A(*) BYTE DATA(CR,LF,
      'TP$51A        Destination Network = ');
37 1  DECLARE TP$52A(*) BYTE DATA(CR,LF,
      'TP$52A        Destination Host = ');
38 1  DECLARE TP$54(*) BYTE DATA(CR,LF,
      'TP$54          Loaded test datagram in TX01TB');
39 1  DECLARE TP$55A(*) BYTE DATA(CR,LF,
      'TP$55A        Reading test datagram from TX01TB');
40 1  DECLARE TP$56A(*) BYTE DATA(CR,LF,
      'TP$56A        Reading test datagram from RX01TB');
41 1  DECLARE TP$57(*) BYTE DATA(CR,LF,
      'TP$57          Sent TR');
42 1  DECLARE TP$58(*) BYTE DATA(CR,LF,
      'TP$58          Sent TA');
43 1  DECLARE TP$59(*) BYTE DATA(CR,LF,
      'TP$59          Received TR');
44 1  DECLARE TP$60(*) BYTE DATA(CR,LF,
      'TP$60          Received TA');

45 1  DECLARE MSG1(*) BYTE DATA(CR,LF,
      'Do you want to load the test message (Y/N) [N] ? ');
46 1  DECLARE MSG1A(*) BYTE DATA(CR,LF,
      'Do you want TR/TA handshake (Y/N) [N] ? ');
47 1  DECLARE MSG2(*) BYTE DATA(CR,LF,
      'Do you want to stop the test (Y/N) [N] ? ');
48 1  DECLARE MSG3(*) BYTE DATA(CR,LF,
      'Load into which host channel (1,2,3,4)? ');
49 1  DECLARE MSG4(*) BYTE DATA(CR,LF,
      'How many datagrams (0 - 9)? ');
50 1  DECLARE MSG5(*) BYTE DATA(CR,LF,
      'Destination network code (1,2,3) = ');
51 1  DECLARE MSG6(*) BYTE DATA(CR,LF,
      'Destination host code (0 - FFH) = ');
52 1  DECLARE MSG7(*) BYTE DATA(CR,LF,
      'Finished simulation, rebooting to CP/M',CR,LF);
53 1  DECLARE MSG8(*) BYTE DATA(CR,LF,
      'Timeout on receive datagram from UNID');
54 1  DECLARE MSG9(*) BYTE DATA(CR,LF,
      'Timeout on receive TA from UNID');

```

32 1 DECLARE

```

CHAN$PTR ADDRESS,
RXTA$TRIES BYTE,
TEMP1(*) BYTE INITIAL('XXEMPTY EMPTYXXX'),
TEMP2(*) BYTE INITIAL('XXEMPTY EMPTYXXX'),
(TRA, TXTR, TXTA, RXTR, RXTA, CHAR) BYTE,
FOREVER BYTE,
TRANS$S$ROY BYTE,
BYTES$SENT$1 BYTE,
BYTES$RECV$1 BYTE,
MSGNUM BYTE,

```

```

/*****
/* DATA TABLES USED IN THIS PROGRAM */
*****/

```

```

RX01TB (DATA$TABLE$SIZE) BYTE,
RX01NS ADDRESS,
RX01NE ADDRESS,
RX01SZ ADDRESS,

```

```

TX01TB (DATA$TABLE$SIZE) BYTE,
TX01NS ADDRESS,
TX01NE ADDRESS,
TX01SZ ADDRESS,

```

```

DESTINATION ADDRESS, /* DESTINATION OF THE PACKET */
DESTINATION$ADDRESS BYTE, /* DESTIN ADDR OF DATAGRAM */
SOURCE$ADDRESS BYTE; /* SOURCE ADDR OF DATAGRAM */

```

/* MISCELLANEOUS DECLARATIONS */

33 1 DECLARE

```

TR LITERALLY '42H', /* TR, TA FOLLOWS NETOS */
TA LITERALLY '41H', /* CONVENTION */
BUSY LITERALLY 'OFFH',
TRUE LITERALLY 'OFFH',
FALSE LITERALLY 'OOH',
NMBR$MSK LITERALLY '07H',
OR LITERALLY '0DH',
LF LITERALLY '0AH',
SOURCE LITERALLY '12',
DESTIN LITERALLY '16',
ESC LITERALLY '1BH',
BDOS2 LITERALLY '2', /* BDOS CALL 2-CONSOLE OUTPUT */
BDOS9 LITERALLY '9', /* BDOS CALL 9-PRINT STRING
UNTIL $ */
BDOS10 LITERALLY '10', /* BDOS CALL 10-READ BUFFER */
RCV$STATE LITERALLY '00010110B', /* MODE INSTRUCTION FOR */
/* 8251 USART RCV INT */
TRANS$STATE LITERALLY '00110111B'; /* MODE INSTRUCTION FOR */

```

PL/M-80 COMPILER LOCAL NETWORK TEST PROG FOR CP/M MACHINE, 25 SEP 84

```

15 1    CONIN: PROCEDURE BYTE EXTERNAL;
16 2    END CONIN;

17 1    CONOUT: PROCEDURE (CHAR) EXTERNAL;
18 2        DECLARE CHAR BYTE;
19 2    END CONOUT;

20 1    BDOS: PROCEDURE (FUN$NUM, VALUE) ADDRESS EXTERNAL;
21 2        DECLARE FUN$NUM BYTE,
                VALUE ADDRESS;
22 2    END BDOS;

23 1    EXIT: PROCEDURE EXTERNAL;
24 2    END EXIT;

25 1    DECLARE RESULT ADDRESS;
26 1    DECLARE BUFFER(128) BYTE;
27 1    DECLARE ORLF(*) BYTE DATA(ODH, OAH);
28 1    DECLARE    MESSAGE(*) BYTE DATA(ODH, OAH,
        'THIS IS THE TEST MESSAGE          THIS IS THE TEST
                                MESSAGE!!!!!!');

29 1    DECLARE    ASCII(*) BYTE DATA('0123456789ABCDEF');
30 1    DECLARE (CHAN$NUM, DEST$NET$CODE, DEST$HOST$CODE) BYTE;

31 1    DECLARE    DATA$GRAM$SIZE LITERALLY '128', /* NUMBER OF BYTES FROM HOST */
        PACKET$SIN$TABLE LITERALLY '10',
        DATA$TABLE$SIZE LITERALLY '1280', /* NUMBR OF BYTES IN TABLE */
        TOP$DATA$SIZE LITERALLY '72', /* TCP DATA SIZE */
        MAX$RXTA$TRIES LITERALLY '3', /* MAX NUMBER OF TA WAIT TRIES */

        /* FOLLOWING ARE NETWORK DEFINED VARIABLES */
        /* NOTES: 1. THIS$UNID$NBR MUST REFLECT WHICH UNID THIS IS.
        2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH
        THIS UNID IS LOCATED.
        3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
        ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR
        THE DELNET MONITOR.
        4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNIDS ARE
        CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
        5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO
        PHISTER'S THESIS, APPENDIX C. */

        THIS$UNID$NBR LITERALLY '2', /* UNIQUE ADDRESS FOR THIS UNID */
        THIS$COUNTRY$CODE LITERALLY '9', /* CC WHERE THIS UNID RESIDES */
        MAX$COUNTRY$CODE LITERALLY '9', /* INDICATES COUNTRY$CODES IN USE */
        MAX$NETWORK$CODE LITERALLY '3'; /* INDICATES UNIDS OPERATIONAL IN NET */

        /*****
        /* ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM */
        *****/

```

```

15 1    CONIN: PROCEDURE BYTE EXTERNAL;
16 2    END CONIN;

17 1    CONOUT: PROCEDURE(CHAR) EXTERNAL;
18 2        DECLARE CHAR BYTE;
19 2    END CONOUT;

20 1    BDOS: PROCEDURE(FUN$NUM, VALUE) ADDRESS EXTERNAL;
21 2        DECLARE FUN$NUM BYTE,
22 2            VALUE ADDRESS;
23 2    END BDOS;

24 1    EXIT: PROCEDURE EXTERNAL;
25 2    END EXIT;

26 1    DECLARE RESULT ADDRESS;
27 1    DECLARE BUFFER(128) BYTE;
28 1    DECLARE CRLF(*) BYTE DATA(ODH, OAH);
29 1    DECLARE MESSAGE(*) BYTE DATA(ODH, OAH,
30 1        'THIS IS THE TEST MESSAGE'          THIS IS THE TEST
31 1        MESSAGE!!!!!!');

29 1    DECLARE ASCII(*) BYTE DATA('0123456789ABCDEF');
30 1    DECLARE (CHAN$NUM, DEST$NET$CODE, DEST$HOST$CODE) BYTE;

31 1    DECLARE DATA$GRAM$SIZE LITERALLY '128', /* NUMBER OF BYTES FROM HOST */
        PACKET$IN$TABLE LITERALLY '10',
        DATA$TABLE$SIZE LITERALLY '1280', /* NUMBR OF BYTES IN TABLE */
        TCP$DATA$SIZE LITERALLY '72', /* TCP DATA SIZE */
        MAX$RXT$TRIES LITERALLY '3', /* MAX NUMBER OF TA WAIT TRIES */

        /* FOLLOWING ARE NETWORK DEFINED VARIABLES */
        /* NOTES: 1. THIS$UNID$NBR MUST REFLECT WHICH UNID THIS IS.
        2. THIS$COUNTRY$CODE MUST REFLECT THE AREA TO WHICH
        THIS UNID IS LOCATED.
        3. MAX$COUNTRY$CODE WILL INDICATE WHICH COUNTRY CODES
        ARE CURRENTLY OPERATIONAL. CC=0000 IS RESERVED FOR
        THE DELNET MONITOR.
        4. MAX$NETWORK$CODE WILL INDICATE HOW MANY UNIDS ARE
        CURRENTLY OPERATIONAL WITHIN A PARTICULAR COUNTRY.
        5. FOR DETAILED INFORMATION ON THE ABOVE REFER TO
        PHISTER'S THESIS, APPENDIX C. */

        THIS$UNID$NBR LITERALLY '2', /* UNIQUE ADDRESS FOR THIS UNID */
        THIS$COUNTRY$CODE LITERALLY '9', /* CC WHERE THIS UNID RESIDES */
        MAX$COUNTRY$CODE LITERALLY '9', /* INDICATES COUNTRY$CODES IN USE */
        MAX$NETWORK$CODE LITERALLY '3'; /* INDICATES UNIDS OPERATIONAL IN NET */

        /*****
        /* ADDITIONAL GENERAL DECLARES NEEDED FOR THIS PROGRAM */
        *****/

```

4a. CP/M Host

The purpose of this program is to operate a CP/M system as a simulated host to a UNID II. The program is similar to the SBC 544 simulation in that many of the same variables are used. In addition, an assembly language program is used to interface the PL/M I/O calls to the CP/M BDOS calls.

PL/M-80 COMPILER LOCAL NETWORK TEST PROGRAM, 2 OCT 1984

```
853 4          TRTA$1, TRTA$2, TRTA$3, TRTA$4 = FALSE;
859 4          END;
860 3          CALL SNDSEQ(.MSG1, SIZE(MSG1));
861 3          HALT;
862 3          END;

863 2          END SET$TRTA;

/*****
/*      THIS IS THE MAIN BODY OF THE PROGRAM      */
*****/

864 1          BEGIN:

          DISABLE;
865 1          CALL S$MASK(SIM$MASK);
866 1          CALL INITIALIZE$BOARD;
867 1          CALL SNDSEQ(.HEADER, SIZE(HEADER));
868 1          CALL SNDSEQ(.TP$5, SIZE(TP$5));
869 1          CALL INIT;
870 1          CALL SNDSEQ(.TP$6, SIZE(TP$6));
871 1          CALL INIT$TAB;
872 1          CALL SNDSEQ(.TP$7, SIZE(TP$7));
873 1          ENABLE;

874 1          CALL SET$TRTA;
875 1          FOREVER = TRUE;
876 1          CALL SNDSEQ(.TP$8, SIZE(TP$8));
877 1          DO WHILE FOREVER;
878 2              CALL ROUTE$IN;
879 2              CALL ROUTE$OUT;
880 2              CALL LOOP;
881 2          END;

882 1          END MAIN;

/***** THE END *****/
```

MODULE INFORMATION:

```
CODE AREA SIZE      = 18A6H  6310D
VARIABLE AREA SIZE  = 14A7H  5237D
MAXIMUM STACK SIZE  = 000EH   14D
1834 LINES READ
0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```

833 2      IF (LSEM$1 = READY AND NSEM$1 = DONE) AND (NOT SEND$1) THEN
834 2          CALL SERVICE$LOOP(LPTR$1, 1);

835 2      IF (LSEM$2 = READY AND NSEM$1 = DONE) AND (NOT SEND$2) THEN
836 2          CALL SERVICE$LOOP(LPTR$2, 2);

837 2      IF (LSEM$3 = READY AND NSEM$1 = DONE) AND (NOT SEND$3) THEN
838 2          CALL SERVICE$LOOP(LPTR$3, 3);

839 2      IF (LSEM$4 = READY AND NSEM$1 = DONE) AND (NOT SEND$4) THEN
840 2          CALL SERVICE$LOOP(LPTR$4, 4);

841 2      END LOOP;

```

```

/*****
PROCEDURE SET$TRTA      ALLOWS OPERATOR SELECTION OF WHICH HOST PORT
                        WILL USE THE TRTA HANDSHAKE

```

PURPOSE OF THIS PROCEDURE IS TO ALLOW OPERATOR INTERACTION AT THE TERMINAL
TO DETERMINE WHICH OF THE FOUR LOCAL HOST PORTS WILL USE THE TRTA
HANDSHAKE.

INPUT - DATA (PORT NUMBERS) PROVIDED INTERACTIVELY BY THE OPERATOR

PROCESSING - THE OPERATOR IS QUERIED FOR INPUT TO DETERMINE WHICH OF THE
FOUR LOCAL HOST PORTS WILL USE THE TRTA HANDSHAKE.

OUTPUT - THE TRTA BOOLEAN FLAGS ARE SET OR RESET.

INTERFACE - CALLED BY MAIN PROCEDURE

```

*****/
842 1      SET$TRTA: PROCEDURE;

843 2          CALL SNDSEQ(.MSG1, SIZE(MSG1));
844 2          HALT;
845 2          DO WHILE ((CHAR$X = 'Y') OR (CHAR$X = 'y'));
846 3              IF ((CHAR$X = 'Y') OR (CHAR$X = 'y')) THEN
847 3                  DO;
848 4                      CALL SNDSEQ(.MSG2, SIZE(MSG2));
849 4                      HALT;
850 4                      CHAR$X = CHAR$X - 31H;
851 4                      IF ((CHAR$X >= 0) AND (CHAR$X <= 3)) THEN
852 4                          DO CASE CHAR$X;
853 5                              TRTA$1 = TRUE;
854 5                              TRTA$2 = TRUE;
855 5                              TRTA$3 = TRUE;
856 5                              TRTA$4 = TRUE;
857 5                          END;
                        ELSE

```

```

811 2      NT01RX(NT01NE + 1) = 0;
812 2      NT01RX(NT01NE + 2) = LCOXR(1);
813 2      NT01RX(NT01NE + 3) = LCOXR(0);      /* SWAP PACKET HEADER */
814 2      DO INDEX = 2 TO (PACKET$SIZE - 1);      /* SWAP DATA */
815 3          NT01RX(NT01NE + INDEX + 2) = LCOXR(INDEX);
816 3      END;

817 2      DO INDEX = 0 TO 3;      /* SWAP IP DEST & SOURCE */
818 3          NT01RX(NT01NE + INDEX + 19) = LCOXR(INDEX + 21);
819 3          NT01RX(NT01NE + INDEX + 23) = LCOXR(INDEX + 17);
820 3      END;

821 2      CALL LD$TAB$H$SKP(9);
822 2      DO CASE PORT;
823 3          ;
824 3          LSEM$1 = DONE;
825 3          LSEM$2 = DONE;
826 3          LSEM$3 = DONE;
827 3          LSEM$4 = DONE;
828 3      END;
829 2      NPTR$1 = .NT01RX(NT01NS + 2);
830 2      NSEM$1 = READY;

831 2      END SERVICE$LOOP;

```

/******

PROCEDURE LOOP SIMULATES A DATAGRAM TRAVERSING A NETWORK OF UNIDS

PROCEDURE - THE LOOP PROCEDURE SIMULATES THE ACTIONS OF THE SBC 88/45 BOARD WITH THE USE OF SEMAPHORES AND POINTERS. IT CALLS THE ROUTINE SERVICE\$LOOP WHICH SWAPS DESTINATION AND SOURCE HEADERS IN THE DATAGRAM IP HEADER AND PACKET HEADER. THE PACKET IS THEN MOVED TO THE NT01RX TABLE IN SYSTEM MEMORY, SIMULATING THE ACTIONS OF THE SBC 88/45 BOARD.

INPUT - DATAGRAMS IN THE LOCAL HOST RECEIVE BUFFERS, DETERMINED BY THE SEMAPHORES LSEM AND NSEM.

PROCESSING - WHEN THE SEMAPHORES ARE CORRECT AND A DATAGRAM TRANSMISSION IS NOT IN PROGRESS, SERVICE\$LOOP IS CALLED TO MOVE A DATAGRAM INTO THE NETWORK RECEIVE BUFFER.

OUTPUT - A DATAGRAM WITH APPROPRIATE PACKET HEADER IN THE NETWORK RECEIVE BUFFER.

INTERFACE - CALLED BY MAIN PROGRAM.

/******

```

832 1      LOOP: PROCEDURE;      /* NSEM$1 IS USED FOR THE SIMULATION AND SHOULD NOT
                                BE USED FOR THE 88/45 OPERATIONAL SOFTWARE. THE 88/45
                                NEEDS ONLY TO INTERROGATE THE LSEM$X VARIABLE.      */

```

```

789 3          DO;
790 4          CALL SNDSEQ(.TP$3, SIZE(TP$3));
791 4          TXTR$4 = TRUE;
792 4          SEND$4 = TRUE;
793 4          OUTPUT(US$P3$DATA) = TR;
794 4          END;
795 3          IF ((NOT TRTA$4) OR ((TXTR$4 AND RXTA$4) AND
              ((NOT RXTR$4) AND (NOT TXTA$4)))) THEN
796 3          DO;
797 4          CALL SNDSEQ(.TP$4, SIZE(TP$4));
798 4          SEND$4 = TRUE;
799 4          DISABLE;
800 4          OUTPUT(IC$PORTB) = INPUT(IC$PORTB) AND 07FH;
801 4          ENABLE;
802 4          END;
803 3          END;

804 2          DISABLE;
805 2          OUTPUT(IC$PORTB) = INPUT(IC$PORTB) AND 0BFH;
806 2          ENABLE;

807 2          END ROUTE$OUT;

```

/*****
 PROCEDURE SERVICE\$LOOP A SOFTWARE LOOP TO SIMULATE A DATAGRAM TRAVER-
 SING A NETWORK OF UNIDS

PURPOSE - TO LOOP A FRAME AROUND AT THE NETWORK LAYER. THE SOURCE AND
 DESTINATION ADDRESSES ARE SWITCHED AND THE PACKET IS PUT INTO THE
 NETWORK LAYER RECEIVE TABLE. USED FOR SIMULATION PURPOSES ONLY.

INPUT - A POINTER TO THE DATAGRAM TO BE LOOPED AND THE RECEIVE PORT NUMBER
 FROM WHICH IT CAME.

PROCESSING - THE SOURCE AND DESTINATION ADDRESSES IN THE DATAGRAM IP
 HEADER ARE SWAPPED. THE DATAGRAM IS THEN MOVED TO THE NETWORK RECEIVE
 BUFFER AND THE APPROPRIATE POINTERS AND SEMAPHORES SET ACCORDINGLY.

OUTPUT - A DATAGRAM MOVED INTO THE NETWORK RECEIVE PORT WITH THE APPROPRIATE
 POINTER AND SEMAPHORE SET.

INTERFACE - CALLED BY LOOP

```

*****/
808 1  SERVICE$LOOP: PROCEDURE(TABLE$PTR, PORT);
809 2  DECLARE     INDEX     ADDRESS,
           PORT BYTE,
           TABLE$PTR ADDRESS,
           LCOXRX BASED TABLE$PTR (1) BYTE;

810 2          NT01RX(NT01NE + 0) = 0;

```

```

748 4          SEND$2 = TRUE;
749 4          OUTPUT(US$P1$DATA) = TR;
750 4          END;
751 3          IF ((NOT TRTA$2) OR ((TXTR$2 AND RXTA$2) AND
              ((NOT RXTR$2) AND (NOT TXTA$2)))) THEN
752 3              DO;
753 4              CALL SNDSEQ(.TP$4, SIZE(TP$4));
754 4              SEND$2 = TRUE;
755 4              DISABLE;
756 4              OUTPUT(IC$PORTB) = INPUT(IC$PORTB) AND OF7H;
757 4              ENABLE;
758 4              END;
759 3          END;

760 2          DISABLE;
761 2          OUTPUT(IC$PORTB) = (INPUT(IC$PORTB) AND OFBH) OR 10H;
762 2          ENABLE;

763 2          IF (((TX03NE - TX03NS) >= DATA$GRAM$SIZE) OR (TX03NS > TX03NE)) AND
              (NOT SEND$3) THEN
764 2              DO;
765 3              CALL SNDSEQ(.TP$2, SIZE(TP$2));
766 3              IF (TRTA$3 AND (((NOT TXTR$3) AND (NOT RXTA$3)) AND
              ((NOT RXTR$3) AND (NOT TXTA$3)))) THEN
767 3                  DO;
768 4                  CALL SNDSEQ(.TP$3, SIZE(TP$3));
769 4                  TXTR$3 = TRUE;
770 4                  SEND$3 = TRUE;
771 4                  OUTPUT(US$P2$DATA) = TR;
772 4                  END;
773 3              IF ((NOT TRTA$3) OR ((TXTR$3 AND RXTA$3) AND
              ((NOT RXTR$3) AND (NOT TXTA$3)))) THEN
774 3                  DO;
775 4                  CALL SNDSEQ(.TP$4, SIZE(TP$4));
776 4                  SEND$3 = TRUE;
777 4                  DISABLE;
778 4                  OUTPUT(IC$PORTB) = INPUT(IC$PORTB) AND 0DFH;
779 4                  ENABLE;
780 4                  END;
781 3              END;

782 2          DISABLE;
783 2          OUTPUT(IC$PORTB) = (INPUT(IC$PORTB) AND 0EFH) OR 40H;
784 2          ENABLE;

785 2          IF (((TX04NE - TX04NS) >= DATA$GRAM$SIZE) OR (TX04NS > TX04NE)) AND
              (NOT SEND$4) THEN
786 2              DO;
787 3              CALL SNDSEQ(.TP$2, SIZE(TP$2));
788 3              IF (TRTA$4 AND (((NOT TXTR$4) AND (NOT RXTA$4)) AND
              ((NOT RXTR$4) AND (NOT TXTA$4)))) THEN

```

IS PRESENT TO SEND TO THE LOCAL HOST. THE INTERRUPT MASKS ARE MASKED OFF AND MASKED ON BY THE TRANSMIT INTERRUPT ROUTINE AT THE END OF SENDING A PACKET.

INTERFACE - THIS PROCEDURE IS CALLED IN AN ENDLESS LOOP BY THE MAIN PROGRAM.

*****/

```

715 1 ROUTE$OUT: PROCEDURE;

716 2     DISABLE;
717 2     OUTPUT(IC$PORTB) = INPUT(IC$PORTB) OR 1;
718 2     ENABLE;

719 2     IF (((TX01NE - TX01NS) >= DATA$GRAM$SIZE) OR (TX01NS > TX01NE)) AND
        (NOT SEND$1) THEN
720 2         DO;
721 3         CALL SNDSEQ(.TP$2, SIZE(TP$2));
722 3         IF (TRTA$1 AND (((NOT TXTR$1) AND (NOT RXTA$1)) AND
            ((NOT RXTR$1) AND (NOT TXTA$1)))) THEN
723 3             DO;
724 4             CALL SNDSEQ(.TP$3, SIZE(TP$3));
725 4             TXTR$1 = TRUE;
726 4             SEND$1 = TRUE;
727 4             OUTPUT(US$PO$DATA) = TR;
728 4             END;
729 3         IF ((NOT TRTA$1) OR ((TXTR$1 AND RXTA$1) AND
            ((NOT RXTR$1) AND (NOT TXTA$1)))) THEN
730 3             DO;
731 4             CALL SNDSEQ(.TP$4, SIZE(TP$4));
732 4             SEND$1 = TRUE;
733 4             DISABLE;
734 4             OUTPUT(IC$PORTB) = INPUT(IC$PORTB) AND OFDH;
735 4             ENABLE;
736 4             END;
737 3         END;

738 2     DISABLE;
739 2     OUTPUT(IC$PORTB) = (INPUT(IC$PORTB) AND OFEH) OR 4;
740 2     ENABLE;

741 2     IF (((TX02NE - TX02NS) >= DATA$GRAM$SIZE) OR (TX02NS > TX02NE)) AND
        (NOT SEND$2) THEN
742 2         DO;
743 3         CALL SNDSEQ(.TP$2, SIZE(TP$2));
744 3         IF (TRTA$2 AND (((NOT TXTR$2) AND (NOT RXTA$2)) AND
            ((NOT RXTR$2) AND (NOT TXTA$2)))) THEN
745 3             DO;
746 4             CALL SNDSEQ(.TP$3, SIZE(TP$3));
747 4             TXTR$2 = TRUE;

```

```

690 4          IF LSEM$4 = DONE THEN
691 4              CALL SEND$PACKET(.LC04RX(LC04NS), 4);
692 4          END;
          ELSE
693 3              DO;
694 4                  STATTB(L$RI$DEST$ERR) = STATTB(L$RI$DEST$ERR) + 1;
695 4              END;
696 3          CALL SRVC$TAB$H$K$P(4);
697 3          END;

698 2          IF NSEM$1 = READY THEN
699 2              DO;
700 3                  DESTINATION$ADDRESS = DET$ADDR$NL(.NETR1(0));
701 3                  IF DESTINATION$ADDRESS >= 0 THEN
702 3                      CALL MOVETO$LOCAL(.NETR1(5), DESTINATION$ADDRESS);
703 3                  NSEM$1 = DONE;
704 3                  CALL SRVC$TAB$H$K$P(9); /* FOR SIMULATION ONLY */
705 3                  END;

706 2          IF NSEM$2 = READY THEN
707 2              DO;
708 3                  DESTINATION$ADDRESS = DET$ADDR$NL(.NETR2(0));
709 3                  IF DESTINATION$ADDRESS >= 0 THEN
710 3                      CALL MOVETO$LOCAL(.NETR2(5), DESTINATION$ADDRESS);
711 3                  NSEM$2 = DONE;
712 3                  CALL SRVC$TAB$H$K$P(10); /* FOR SIMULATION ONLY */
713 3                  END;

714 2          END ROUTE$IN;

```

/******

PROCEDURE ROUTE\$OUT DETECT WHEN DATAGRAMS ARE PRESENT IN THE
LOCAL HOST TRANSMIT BUFFERS AND PROCESSES FOR TRANSMISSION.

PURPOSE OF THIS PROCEDURE IS TO TRANSMIT DATAGRAMS TO THE LOCAL HOSTS
THE TRANSMIT REQUEST/TRANSMIT (TRTA) ACKNOWLEDGE ARRANGEMENT IS
INCORPORATED.

INPUT - DATAGRAMS IN THE LOCAL HOST TRANSMIT BUFFERS.

PROCESSING - THE RECEIVE INTERRUPT IS MASKED OFF BEFORE CHECKING THE
TRANSMIT BUFFER POINTERS (REQUIRED TO HAVE STABLE TABLE POINTERS)
WHEN A DATAGRAM IS PRESENT AND THE UNID IS NOT OTHERWISE SENDING
A DATAGRAM TO A PARTICULAR HOST, THE TRTA FLAGS ARE CHECK FOR THE
ALLOWABLE STATES. WHEN THE TRTA STATES ARE CORRECT, A TR IS SEND
TO THE LOCAL HOST TO INDICATE THE UNID IS READY TO TRANSMIT TO THE
HOST. WHEN A TA IS RECEIVED FROM THE LOCAL HOST, THE TRANSMIT
INTERRUPT MASK IS TURNED ON FOR THE PARTICULAR LOCAL HOST PORT
AND THE DATAGRAM SENT VIA THE TRANSMIT INTERRUPT ROUTINES.

OUTPUT - THE TRTA BOOLEAN FLAGS ARE SET APPROPRIATELY WHEN A DATAGRAM

```

649 3      CALL DET$ADDR(.LC02RX(LC02NS + 5));
650 3      IF DESTINATION = 0 THEN
651 3          DO;
652 4          CALL MOVETO$LOCAL(.LC02RX(LC02NS + 5), (DESTINATION$ADDRESS AND
                                                OFH));
653 4      END;
        ELSE
654 3      IF DESTINATION = 1 THEN
655 3          DO;
656 4          IF LSEM$2 = DONE THEN
657 4              CALL SEND$PACKET(.LC02RX(LC02NS), 2);
658 4          END;
        ELSE
659 3          DO;
660 4          STATB(L$R1$DEST$ERR) = STATB(L$R1$DEST$ERR) + 1;
661 4          END;
662 3      CALL SRVC$TAB$H$SKP(2);
663 3      END;

664 2      IF (((LC03NE - LC03NS) >= DATA$GRAM$SIZE) OR (LC03NS > LC03NE)) THEN
665 2          DO;
666 3          CALL DET$ADDR(.LC03RX(LC03NS + 5));
667 3          IF DESTINATION = 0 THEN
668 3              DO;
669 4              CALL MOVETO$LOCAL(.LC03RX(LC03NS + 5), (DESTINATION$ADDRESS AND
                                                OFH));
670 4          END;
        ELSE
671 3          IF DESTINATION = 1 THEN
672 3              DO;
673 4              IF LSEM$3 = DONE THEN
674 4                  CALL SEND$PACKET(.LC03RX(LC03NS), 3);
675 4              END;
        ELSE
676 3          DO;
677 4          STATB(L$R1$DEST$ERR) = STATB(L$R1$DEST$ERR) + 1;
678 4          END;
679 3      CALL SRVC$TAB$H$SKP(3);
680 3      END;

681 2      IF (((LC04NE - LC04NS) >= DATA$GRAM$SIZE) OR (LC04NS > LC04NE)) THEN
682 2          DO;
683 3          CALL DET$ADDR(.LC04RX(LC04NS + 5));
684 3          IF DESTINATION = 0 THEN
685 3              DO;
686 4              CALL MOVETO$LOCAL(.LC04RX(LC04NS + 5), (DESTINATION$ADDRESS AND
                                                OFH));
687 4          END;
        ELSE
688 3          IF DESTINATION = 1 THEN
689 3              DO;

```

```

77 1      VALID$HEX: PROCEDURE(H) BYTE;
78 2          DECLARE (H,I) BYTE;

79 2          DO I = 0 TO LAST(ASCII);
80 3              IF H = ASCII(I) THEN
81 3                  RETURN TRUE;
82 3          END;
83 2          RETURN FALSE;

84 2      END VALID$HEX;

      /*****
      /*
      /* PROCEDURE      INIT      INITIALIZES THE SOFTWARE VARIABLES
      /*
      /* THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE THE DATA TABLES
      /* AND OTHER VARIABLES.
      /*
      /* INPUT - NONE
      /* PROCESSING - SETS THE VARIABLES TO THEIR INITIAL VALUES.
      /* INITIALIZES THE CP/M USART PORT VARIABLES.
      /* OUTPUT - INITIALIZED DATA TABLES AND VARIABLES
      /* INTERFACE - CALLED BY MAIN PROGRAM
      /*
      /*
      /* *****/

85 1      INIT: PROCEDURE;

86 2          MSGNUM = 0;
87 2          BYTES$SENT$1 = 0;
88 2          BYTES$RECV$1 = 0;
89 2          CHAN$NUM = 0;
90 2          DEST$NET$CODE = 0;
91 2          BUFFER(0) = 120;
92 2          CHAR = ' ';
93 2          TRANS$1$RDY = FALSE;
94 2          RXTA$TRIES = 0;
95 2          TRTA, TXTR, TXTA, RXTR, RXTA = FALSE;

96 2          RX01NS = 0;
97 2          RX01NE = 0;
98 2          RX01SZ = DATA$TABLE$SIZE;

99 2          TX01NS = 0;
100 2          TX01NE = 0;
101 2          TX01SZ = DATA$TABLE$SIZE;

102 2          CALL SINIT;
103 2          CALL SOLROM;

104 2      END INIT;

```

```

/*****
/* PROCEDURE SNDSEQ SENDS DATA TO LOCAL MONITOR FOR TESTING */
/*
/* THIS PROCEDURE TAKES A MESSAGE STRING AND OUTPUTS IT TO */
/* THE CP/M SYSTEM */
/*
/* INPUT - A POINTER TO THE MESSAGE LOCATION IN MEMORY AND THE */
/* NUMBER OF BYTES TO BE SENT. */
/* PROCESSING - THIS PROCEDURE CHECKS THE OUTPUT BUFFER STATUS */
/* IN A LOOP UNTIL THE BUFFER IS EMPTY. IT CALLS THE CP/M */
/* BDOS FUNCTION 2 UNTIL THE COUNT IS ZERO. */
/* OUTPUT - MESSAGE TO THE HOST CP/M SYSTEM. */
/* INTERFACE - THIS PROCEDURE IS CALLED BY THE FOLLOWING PROCEDURES: */
/* ROUTE$IN, ROUTE$OUT, AND MAIN. */
/*
*****/

105 1 SNDSEQ: PROCEDURE(MSG, TOTAL) REENTRANT;
106 2     DECLARE MSG ADDRESS,
          CHAROUT BASED MSG (1) BYTE,
          (TOTAL, COUNT) BYTE;

107 2     COUNT = 0;
108 2     DO WHILE COUNT < TOTAL;
109 3         RESULT = BDOS(BDOS2, CHAROUT(COUNT));
110 3         COUNT = COUNT + 1;
111 3     END;

112 2 END SNDSEQ;

/*****
* READ PROCEDURE READS THE CP/M CONSOLE THROUGH BDOS CALL 10 *
*
* INPUT- BDOS FUNCTION 10, A POINTER TO A BUFFER *
*
* OUTPUT- CONTENTS OF THE BUFFER FILLED WITH CONSOLE INPUT. *
* I/O ERROR RESULT IN THE VARIABLE RESULT. *
*****/

113 1 READ: PROCEDURE;

114 2     RESULT = BDOS(BDOS10, .BUFFER(0));

115 2 END READ;

/*****
/*
/* PROCEDURE RCV$1 RECEIVES DATA */
/*
/* THE PURPOSE OF THIS PROCEDURE IS TO TAKE A CHARACTER AT A TIME */
*****/

```

```

/* FROM THE RECEIVE PORT ONE AND PUT IT IN THE RECEIVE BUFFER. */
/* THE SECOND PART RECEIVES A DATAGRAM IN TOTAL AND PUTS IT INTO */
/* THE RECEIVE BUFFER. A TIMER IS USED TO KEEP THE ROUTINE FROM */
/* HANGING UP. */
/*
/* INPUT - NONE
/* PROCESSING - MOVES A BYTE OF DATA FROM THE RECEIVE CHARACTER
/* ROUTINE IN THE ASSEMBLY LANGUAGE PROGRAM INTO THE RECEIVE
/* BUFFER.
/* OUTPUT - RECEIVED DATAGRAM IN THE RECEIVE BUFFER.
/* INTERFACE - EXTERNAL CALL TO ASSEMBLY LANGUAGE PROGRAM.
/*
/*****
/* USE THIS CODE FOR SELF TEST; CHAR BY CHAR RECEIVE */
/*
RCV$1: PROCEDURE;

    RX01TB(RX01NE) = SCMIN;
    RX01NE = RX01NE + 1;
    IF(RX01NE >= RX01SZ) THEN
        RX01NE = 0;

END RCV$1;
*/
/* USE THE FOLLOWING FOR OPERATIONAL 544 S/W TEST */
/* DATAGRAM BY DATAGRAM RECEIVE */

```

```

116 1 RCV$1: PROCEDURE;
117 2     DECLARE COUNT ADDRESS;

118 2     COUNT = 30000;
119 2     DO WHILE ((SCMCHK = 0) AND (COUNT <> 0));
120 3         COUNT = COUNT - 1;
121 3     END;
122 2     IF COUNT = 0 THEN
123 2         DO;
124 3             CALL SNDSEQ(.MSG8, SIZE(MSG8));
125 3             RXTR = FALSE;
126 3             TXTA = FALSE;
127 3             END;
128 2     ELSE
129 2         IF ((NOT TRTA) OR (RXTR AND TXTA AND
130 3             (NOT TXTR) AND (NOT RXTA))) THEN
131 4             DO;
132 4                 RX01TB(RX01NE) = SCMIN;
133 4                 RX01NE = RX01NE + 1;
134 4                 BYTES$RCV$1 = BYTES$RCV$1 + 1;
135 4             END;
136 3             BYTES$RCV$1 = 0;
137 3             IF (RX01NE >= RX01SZ) THEN

```

```

137 3          RXD1NE = 0;
138 3          RXTR = FALSE;
139 3          TXTA = FALSE;
140 3          END;

```

END RCV\$1;

```

/*****
/*
/* PROCEDURE TRANS$1 TRANSMITS DATA
/*
/* THE PURPOSE OF THIS PROCEDURE IS TO TAKE A CHARACTER AT A TIME
/* FROM THE TRANSMIT BUFFER AND SEND IT TO CP/M HOST TRANSMIT USART.
/* THE SECOND PART TRANSMITS A DATAGRAM IN TOTAL .
/*
/* INPUT - A DATAGRAM IN THE TRANSMIT BUFFER.
/* PROCESSING - MOVES A BYTE OF DATA FROM THE TRANSMIT BUFFER
/* TO THE TRANSMIT CHARACTER ROUTINE IN THE ASSEMBLY LANGUAGE
/* PROGRAM .
/* OUTPUT - UPDATED ARRAY POINTERS
/* INTERFACE - EXTERNAL CALL TO ASSEMBLY LANGUAGE PROGRAM.
/*
/*
*****/
/* USE THIS CODE FOR SELF TEST; CHAR BY CHAR TRANSMIT */
/*

```

TRANS\$1: PROCEDURE;

IF (BYTES\$SENT\$1 < DATA\$GRAM\$SIZE) THEN

```

DO;
CALL SCMOU(TX01TB(TX01NS));
BYTES$SENT$1 = BYTES$SENT$1 + 1;
TX01NS = TX01NS + 1;

```

IF BYTES\$SENT\$1 >= DATA\$GRAM\$SIZE THEN

```

DO;
BYTES$SENT$1 = 0;
IF TX01NS >= TX01SZ THEN
TX01NS = 0;
TRANS$1$RDY = FALSE;
END;

```

END;

END TRANS\$1;

*/

/* USE THE FOLLOWING FOR OPERATIONAL 544 S/W TEST */

/* DATAGRAM BY DATAGRAM TRANSMIT */

142 1 TRANS\$1: PROCEDURE;

143 2 DO;

144 3 DO WHILE (BYTES\$SENT\$1 < DATA\$GRAM\$SIZE);

```

145  4      CALL SMOJUT(TX01TB(TX01NS));
146  4      BYTES$SENT$1 = BYTES$SENT$1 + 1;
147  4      TX01NS = TX01NS + 1;
148  4      END;
149  3      BYTES$SENT$1 = 0;
150  3      IF TX01NS >= TX01SZ THEN
151  3          TX01NS = 0;
152  3      TRANS$1$RDY = FALSE;
153  3      TXTR = FALSE;
154  3      RXTA = FALSE;
155  3      END;

```

```

156  2      END TRANS$1;

```

/******

PROCEDURE LD\$TAB\$H\$SKP LOAD TABLE HOUSEKEEP

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED BUFFER
TABLE AFTER LOADING OF THE USER DATA FROM THE HOST.

INPUT - THE INPUT IS AN ADDRESS INDICATING THE TABLE REQUIRING
CHANGES.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT\$EMPTY\$BYTE ADDRESS BY ONE DATA\$GRAM\$SIZE,
AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT\$EMPTY\$BYTE ADDRESS
ADVANCED BY THE LENGTH OF A SINGLE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN AND ROUTE\$OUT.

*****/

```

157  1      LD$TAB$H$SKP: PROCEDURE(TABLE) ;
158  2          DECLARE    TABLE ADDRESS;

159  2          TX01NE = TX01NE + DATA$GRAM$SIZE;
160  2          IF TX01NE >= TX01SZ THEN
161  2              TX01NE = 0;

162  2      END LD$TAB$H$SKP;

```

/******

PROCEDURE SRVC\$TAB\$H\$SKP SERVICE TABLE HOUSEKEEP

THE PURPOSE OF THIS PROCEDURE IS TO HOUSEKEEP A SPECIFIED
BUFFER TABLE AFTER SERVICING (REMOVING A PACKET).

INPUT - THE INPUT IS AN ADDRESS VALUE INDICATING THE TABLE THAT
REQUIRES HOUSEKEEPING.

PROCESSING - THE PROCEDURE DETERMINES THE TABLE TO BE PROCESSED,
ADVANCES THE NEXT\$BYTE\$TO\$BE\$SERVICED ADDRESS BY A DATA\$GRAM\$SIZE.
AND ADJUSTS FOR BUFFER WRAP IF NECESSARY.

OUTPUT - THE SPECIFIED TABLE HAS ITS NEXT\$BYTE\$TO\$BE\$SERVICED ADDRESS
ADVANCED BY THE LENGTH OF A SINGLE DATAGRAM.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN AND ROUTE\$OUT.

*****/

```

163 1  SRVC$TAB$H$K$P: PROCEDURE(TAB) ;
164 2      DECLARE TAB ADDRESS;

165 2      DO CASE TAB;
166 3          ;

167 3          DO;
168 4              RX01NS = RX01NS + DATA$GRAM$SIZE;
169 4              IF RX01NS >= RX01SZ THEN
170 4                  RX01NS = 0;
171 4              END;

172 3          DO;
173 4              TX01NS = TX01NS + DATA$GRAM$SIZE;
174 4              IF TX01NS >= TX01SZ THEN
175 4                  TX01NS = 0;
176 4              END;

177 3      END;

178 2  END SRVC$TAB$H$K$P;

```

PROCEDURE CHK\$RX\$TA CHECK FOR RECEIVED TRANSMIT ACK

THE PURPOSE OF THIS PROCEDURE IS TO CHECK THE RECEIVED CP/M PORT FOR A
RECEIVED TRANSMIT ACK FROM THE UNID II. A TIMER IS USED TO KEEP
THE PROGRAM FROM HANGING UP IN THIS ROUTINE.

INPUT - THE INPUT IS THE FOUR TRTA BOOLEAN FLAGS .

PROCESSING - THE PROCEDURE WILL ONLY EXECUTE IF THE FOUR TRTA BOOLEAN FLAGS
ARE IN THE CORRECT STATES. WHEN THEY ARE, THE PROCEDURE INITIALIZES
THE TIMECOUNTER AND WAITS FOR A RECEIVED TA FROM THE UNID TO AN EARLIER
TRANSMITTED TR TO THE UNID. IF THE TIMER TIMES OUT, THE BOOLEAN IS
RESET AND A TIMEOUT MESSAGE IS DISPLAYED. WHEN THE TA IS RECEIVED
BEFORE THE TIMEOUT, THE PROCEDURE SETS THE BOOLEAN FLAGS ACCORDINGLY
AND SENDS THE DATAGRAM TO THE UNID II.

OUTPUT - A DATAGRAM TO THE UNID II OR RESET BOOLEAN FLAGS.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN AND ROUTE\$OUT.

```

*****/
179 1  CHK$RXTA: PROCEDURE;
180 2      DECLARE COUNT ADDRESS;

181 2      IF (TXTR AND (NOT RXTA) AND (NOT RXTR) AND (NOT TTXA)) THEN
182 2          COUNT = 30000;
183 2          DO WHILE ((SCMCHK = 0) AND (COUNT <> 0));
184 3              COUNT = COUNT - 1;
185 3          END;
186 2          IF COUNT = 0 THEN
187 2              DO;
188 3                  CALL SNDSEQ(.MSG9, SIZE(MSG9));
189 3                  TXTR = FALSE;
190 3                  END;
191 2              ELSE
192 3                  DO;
193 3                      CHAR = SOMIN;
194 3                      IF CHAR = TA THEN
195 4                          DO;
196 4                              CALL SNDSEQ(.TP$60, SIZE(TP$60));
197 4                              RXTA = TRUE;
198 4                              RXTA$TRIES = 0;
199 4                              CALL TRANS$1;
200 4                              END;
201 2                          END;
201 2      END CHK$RXTA;

```

PROCEDURE CHK\$RXTR CHECK FOR RECEIVED TRANSMIT REQUEST

THE PURPOSE OF THIS PROCEDURE IS TO CHECK THE RECEIVED CP/M PORT FOR A RECEIVED TRANSMIT REQUEST FROM THE UNID 11. A TIMER IS USED TO KEEP THE PROGRAM FROM HANGING UP IN THIS ROUTINE.

INPUT - THE INPUT IS THE FOUR TRTA BOOLEAN FLAGS .

PROCESSING - THE PROCEDURE WILL ONLY EXECUTE IF THE FOUR TRTA BOOLEAN FLAGS ARE IN THE CORRECT STATES. WHEN THEY ARE, THE PROCEDURE INITIALIZES THE TIMECOUNTER AND WAITS FOR A RECEIVED TR FROM THE UNID. IF THE TIMER TIMES OUT, THE BOOLEAN IS RESET AND A TIMEOUT MESSAGE IS DISPLAYED. WHEN THE TR IS RECEIVED BEFORE THE TIMEOUT, THE PROCEDURE SETS THE BOOLEAN FLAGS ACCORDINGLY AND SENDS FIRST A TA AND THEN WAITS FOR A DATAGRAM FROM THE UNID 11.

OUTPUT - A TA AND A DATAGRAM FROM THE UNID 11 OR RESET BOOLEAN FLAGS.

INTERFACE - THIS PROCEDURE IS CALLED BY PROCEDURE ROUTE\$IN AND ROUTE\$OUT.

```

*****/
202 1  CHK$RXTR: PROCEDURE;
203 2  DECLARE COUNT ADDRESS;

204 2  IF ((NOT RXTR) AND (NOT TXTA) AND (NOT TXTR) AND (NOT RXTA)) THEN
205 2  COUNT = 30000;
206 2  DO WHILE ((SCMCHK = 0) AND (COUNT <> 0));
207 3  COUNT = COUNT - 1;
208 3  END;
209 2  IF COUNT = 0 THEN
210 2  CALL SNDSEQ(.MSG10, SIZE(MSG10));
211 2  ELSE
212 3  DO;
213 3  CHAR = SCMIN;
214 3  IF CHAR = TR THEN
215 4  DO;
216 4  CALL SNDSEQ(.TP$59, SIZE(TP$59));
217 4  CALL SNDSEQ(.TP$58, SIZE(TP$58));
218 4  RXTR = TRUE;
219 4  TXTA = TRUE;
220 4  CALL SCMOIT(TA);
221 4  CALL RCV$1;
222 3  END;
223 2  END;

223 2  END CHK$RXTR;

/*****
PROCEDURE READ$TXTAB      PROCEDURE TO READ A DATAGRAM IN THE TRANSMIT TABLE

INPUT - THE TRANSMIT TABLE POINTERS AND RETRIES COUNT

PROCESSING - THE PROCEDURE CHECKS THE NUMBER OF ATTEMPTS TO SEND THE DATA
GRAM IN THE TRANSMIT BUFFER. IF THE NUMBER OF TIMES EXCEEDS THE
MAXIMUM, THEN THE COUNTER AND TRTA FLAGS ARE RESET. THE TRANSMIT
BUFFER IS THEN CHECKED TO DETERMINE IF A DATAGRAM IS PRESENT IN
THE TRANSMIT BUFFER. WHEN A DATAGRAM IS PRESENT, THE MESSAGE AND IT'S
IP HEADER IS DISPLAYED ON THE CP/M CONSOLE. IF THE TRTA FLAGS ARE
IN THE CORRECT STATE, THEN A TR IS SEND TO THE UNID 11 AND THE FLAGS
ARE SET TO INDICATE A DATAGRAM IS READY TO BE SENT.

OUTPUT - THE TRANSMIT DATAGRAM AND IP HEADER ARE DISPLAYED ON THE CP/M
CONSOLE.

INTERFACE - THIS PROCEDURE IS CALLED BY THE MAIN PROGRAM.
*****/
224 1  READ$TXTAB: PROCEDURE;
225 2  DECLARE I BYTE;

```

```

226 2      RXTA$TRIES = RXTA$TRIES + 1;
227 2      IF RXTA$TRIES > MAX$RXTA$TRIES THEN
228 2          DO;
229 3              TXTR = FALSE;
230 3              RXTA = FALSE;
231 3              CALL SRVC$TAB$H$SKP(2);
232 3              RXTA$TRIES = 0;
233 3              END;

234 2      IF ((TX01NE - TX01NS) >= DATA$GRAM$SIZE) OR (TX01NS > TX01NE) THEN
235 2          DO;
236 3              I = 12;
237 3              DO WHILE I <= 19;
238 4                  CALL HEX$ASC(TX01TB(TX01NS + 1), (I-12)*2);
239 4                  I = I + 1;
240 4              END;
241 3              CALL SNDSEQ(.TP$55A, SIZE(TP$55A));
242 3              CALL SNDSEQ(.CRLF, SIZE(CRLF));
243 3              CALL SNDSEQ(.TEMP1, SIZE(TEMP1));
244 3              CALL SNDSEQ(.TX01TB(TX01NS + 56), TOP$DATA$SIZE);
245 3              IF (TRTA AND (NOT TXTR) AND (NOT RXTA) AND
                  (NOT RXTR) AND (NOT TXTA)) THEN
246 3                  DO;
247 4                      TXTR = TRUE;
248 4                      CALL SNDSEQ(.TP$57, SIZE(TP$57));
249 4                      CALL SCOUT(TR);
250 4                      END;
251 3              IF (NOT TRTA) THEN
252 3                  TRANS$1$RDY = TRUE;
253 3              END;
254 2      END READ$TXTAB;

```

/*****
 PROCEDURE READ\$TXTAB PROCEDURE TO READ A DATAGRAM IN THE RECEIVE TABLE

INPUT - THE TRANSMIT TABLE POINTERS

PROCESSING - THE PROCEDURE CHECKS THE RECEIVE BUFFER TO DETERMINE IF A
 DATAGRAM IS PRESENT IN THE RECEIVE BUFFER. WHEN A DATAGRAM IS
 PRESENT, THE RECEIVED MESSAGE IS DISPLAYED ON THE CP/M CONSOLE.
 THE IP HEADER IS ALSO DISPLAYED. THE RECEIVED TABLE POINTERS ARE
 UPDATED.

OUTPUT - THE RECEIVED DATAGRAM AND ITS HEADER ARE DISPLAYED ON THE CP/M
 CONSOLE.

INTERFACE - THIS PROCEDURE IS CALLED BY THE MAIN PROGRAM.

*****/
 255 1 READ\$RXTAB: PROCEDURE;

```

256 2      DECLARE I BYTE;

257 2      IF ((RX01NE - RX01NS) >= DATA$GRAM$SIZE) OR (RX01NS > RX01NE) THEN
258 2          DO;
259 3              I = 12;
260 3              DO WHILE I <= 19;
261 4                  CALL HEX$ASC(RX01TB(RX01NS + I), (I-12)*2);
262 4                  I = I + 1;
263 4              END;
264 3              CALL SNDSEQ(.TP$56A, SIZE(TP$56A));
265 3              CALL SNDSEQ(.CRLF, SIZE(CRLF));
266 3              CALL SNDSEQ(.TEMP1, SIZE(TEMP1));
267 3              CALL SNDSEQ(.RX01TB(RX01NS + 56), TOP$DATA$SIZE);
268 3              CALL SRVC*TAB$H$SKP(1);
269 3          END;

270 2      END READ$RXTAB;

/*****
PROCEDURE LOAD      LOADS A DATAGRAM INTO THE TX01TB TABLE.

THE PURPOSE OF THIS PROCEDURE IS TO LOAD A TEST DATAGRAM INTO THE TRANSMIT
TABLE FOR TRANSMISSION TO THE UNID 11.

INPUT - A POINTER TO THE NEXT AVAILABLE SPACE IN THE TRANSMIT TABLE, THE
DESTINATION UNID NUMBER AND THE DESTINATION HOST NUMBER.

PROCESSING - A DATAGRAM IS INITIALIZED TO ZEROS, THEN THE APPROPRIATE
CONSTANTS INSERTED INTO THE HEADER, ALONG WITH THE DESTINATION UNID
AND HOST NUMBERS, AND THE TEST MESSAGE WITH A MODULO 10 NUMBER,
INSERTED AT THE CURRENT TABLE$PTR.

OUTPUT - A COMPLETED DATAGRAM AT THE CURRENT TABLE$PTR IN THE TRANSMIT
TABLE

INTERFACE - CALLED BY READ$LINE

*****/

271 1      LOAD: PROCEDURE(TABLE$PTR, DEST$UNID, DEST$HOST);

272 2          DECLARE (INDEX, TABLE$PTR) ADDRESS;
273 2          DECLARE (DEST$UNID, DEST$HOST) BYTE;
274 2          DECLARE LCOXTB BASED TABLE$PTR (1) BYTE;

275 2          DO INDEX = 0 TO (DATA$GRAM$SIZE - 1);
276 3              LCOXTB(INDEX) = 0;
277 3          END;

/* VARIOUS CONSTANTS PER THE TCP/IP SPEC */
278 2          LCOXTB(0) = 48H;

```

```

279 2      LCOXTB(3) = 80H;
280 2      LCOXTB(6) = 40H;
281 2      LCOXTB(8) = 30H;
282 2      LCOXTB(9) = 6;
283 2      LCOXTB(20) = 82H;
284 2      LCOXTB(21) = 0BH;

      /* IP SOURCE HEADER */
285 2      LCOXTB( 12) = THIS$COUNTRY$CODE AND 0FH;
           /* CT   = 0, CC   = THIS$COUNTRY$CODE */
286 2      LCOXTB( 13) = (ROL(THIS$UNID$NBR, 4) AND 0F0H) OR
           (ROR((CHAN$NUM*56) AND 0F0H), 4) AND 0FH);
           /* NC   = THIS$UNID$NBR, HC(H) = CHAN$NUM * 56 */
287 2      LCOXTB( 14) = 7H OR (ROL((CHAN$NUM*56) AND 0FH), 4) AND 0F0H);
           /* HC(L) = CHAN$NUM * 56, PC(2) = 7 */
288 2      LCOXTB( 15) = 77H; /* PC(1) = 7, PC(0) = 7 */

      /* IP DESTINATION HEADER */
289 2      LCOXTB( 16) = THIS$COUNTRY$CODE AND 0FH;
           /* CT   = 0, CC   = THIS$COUNTRY$CODE */
290 2      LCOXTB( 17) = (ROL(DEST$UNID, 4) AND 0F0H) OR
           (ROR((DEST$HOST AND 0F0H), 4) AND 0FH);
           /* NC   = DEST$UNID, HC(H) = DEST$HOST */
291 2      LCOXTB( 18) = 7H OR (ROL((DEST$HOST AND 0FH), 4) AND 0F0H);
           /* HC(L) = DEST$HOST, PC(2) = 7 */
292 2      LCOXTB( 19) = 77H; /* PC(1) = 7, PC(0) = 7 */

293 2      CALL MOVE(TOP$DATA$SIZE, .MESSAGE, .LCOXTB( 56));
294 2      IF MSGNUM > 9 THEN
295 2          MSGNUM = 0;
296 2      LCOXTB( 86) = '0' + MSGNUM;
297 2      MSGNUM = MSGNUM + 1;
298 2      CALL SNOSEQ(.TP$54, SIZE(TP$54));

299 2      END LOAD;

```

PROCEDURE READ\$LINE INTERPRETS A LINE FROM THE CP/M CONSOLE

PURPOSE OF THIS PROCEDURE IS TO READ A LINE OF DATA FROM THE HOST CP/M CONSOLE, THEN INTERPRET FOR EXECUTING EITHER LOAD ANOTHER TEST MESSAGE IN THE TRANSMIT TABLE OR STOPPING THE TEST.

INPUT - INPUT RESPONSES FROM THE CONSOLE OPERATOR. RESPONSES ARE NUMBER OF DATAGRAMS, DESTINATION UNID, AND DESTINATION HOST.

PROCESSING - THE CONSOLE OPERATOR'S RESPONSES ARE INTERPRETED AND THE APPROPRIATE NUMBER OF DATAGRAMS ARE LOADED INTO THE TRANSMIT BUFFER. THE OPERATOR ALSO HAS OPPORTUNITY TO END THE TEST.

OUTPUT - AN OPERATOR SPECIFIED NUMBER OF DATAGRAMS INSERTED INTO THE

TRANSMIT BUFFER.

INTERFACE - CALLED BY MAIN PROCEDURE

*****/

```

300 1  READ$LINE: PROCEDURE;

301 2      DECLARE TABLE$PTR ADDRESS,
          (1, COUNT) BYTE;

302 2      I = 0;

303 2      CALL SNDSEQ(.MSG1, SIZE(MSG1));
304 2      BUFFER(2) = 'N';
305 2      CALL READ;

306 2      IF (BUFFER(2) = 'Y') OR (BUFFER(2) = 'y') THEN
307 2          DO;
308 3          CALL SNDSEQ(.MSG1A, SIZE(MSG1A));
309 3          BUFFER(2) = 'N';
310 3          CALL READ;
311 3          IF (BUFFER(2) = 'Y') OR (BUFFER(2) = 'y') THEN
312 3              TRTA = TRUE;
          ELSE
313 3              TRTA = FALSE;
314 3          CALL SNDSEQ(.MSG3, SIZE(MSG3));
315 3          CALL READ;
316 3          CHAN$NUM = ASC$HEX(BUFFER(2));
317 3          IF (CHAN$NUM >= 1) AND (CHAN$NUM <= 4) THEN
318 3              DO;
319 4              TABLE$PTR = .TX01TB(TX01NE);
320 4              CALL SNDSEQ(.MSG5, SIZE(MSG5));
321 4              CALL READ;
322 4              DEST$NET$CODE = ASC$HEX(BUFFER(2));

323 4              CALL SNDSEQ(.MSG6, SIZE(MSG6));
324 4              CALL READ;
325 4              DEST$HOST$CODE = 0;
326 4              COUNT = BUFFER(1);
327 4              DO I = 1 TO COUNT;
328 5                  DEST$HOST$CODE = ROL(DEST$HOST$CODE, 4);
329 5                  DEST$HOST$CODE = DEST$HOST$CODE OR ASC$HEX(BUFFER(I+1));
330 5              END;

331 4          CALL SNDSEQ(.MSG4, SIZE(MSG4));
332 4          CALL READ;
333 4          IF (BUFFER(2) >= '1') AND (BUFFER(2) <= '9') THEN
334 4              DO I = 1 TO ASC$HEX(BUFFER(2));
335 5                  CALL LOAD(TABLE$PTR, DEST$NET$CODE, DEST$HOST$CODE);
336 5                  CALL LD$TAB$H$SKP(2);

```

```

337 5          TABLEPTR = .TX01TB(TX01NE);
338 5          END;
339 4          END;
340 3          END;

```

```

341 2          CALL SNDSEQ(.TP$50, SIZE(TP$50));
342 2          CALL HEX$ASC(CHAN$NUM, 0);
343 2          CALL SNDSEQ(.TEMP1, 2);
344 2          CALL SNDSEQ(.TP$51A, SIZE(TP$51A));
345 2          CALL HEX$ASC(DEST$NET$CODE, 0);
346 2          CALL SNDSEQ(.TEMP1, 2);
347 2          CALL SNDSEQ(.TP$52A, SIZE(TP$52A));
348 2          CALL HEX$ASC(DEST$HOST$CODE, 0);
349 2          CALL SNDSEQ(.TEMP1, 2);

```

```

350 2          CALL SNDSEQ(.MSG2, SIZE(MSG2));
351 2          BUFFER(2) = 'N';
352 2          CALL READ;

```

```

353 2          IF (BUFFER(2) = 'Y') OR (BUFFER(2) = 'y') THEN
354 2              FOREVER = FALSE;
          ELSE
355 2              FOREVER = TRUE;

```

```

356 2      END READ$LINE;

```

```

/*****
PROCEDURE READ$LINE      TRANSMITS AND RECEIVES A DATAGRAM

```

PURPOSE OF THIS PROCEDURE IS TO TRANSMIT AND RECEIVE A DATAGRAM TO AND FROM THE UNID II. THIS PROCEDURE DOES NOT MAKE USE OF THE TRTA HANDSHAKE. INSTEAD, IT TRANSMITS A FULL DATAGRAM TO THE UNID AND THEN WAITS UNTIL THE UNID SENDS A DATAGRAM BACK. USED TO TEST THE UNID II WITHOUT THE TRTA HANDSHAKE.

INPUT - DATAGRAM IN THE TRANSMIT BUFFER

PROCESSING - THE TRANSMIT DATAGRAM AND RECEIVE DATAGRAM PROCEDURES ARE CALLED TO SEND AND RECEIVE A DATAGRAM, RESPECTIVELY.

OUTPUT - A DATAGRAM IN THE RECEIVE BUFFER.

INTERFACE - CALLED BY MAIN PROCEDURE

```

*****/

```

```

357 1      LOOP2: PROCEDURE;

358 2          CALL TRANS$1;
359 2          CALL RCV$1;

```

PL/M-80 COMPILER LOCAL NETWORK TEST PROG FOR CP/M MACHINE, 25 SEP 84

```
360      2      END LOOP2;
```

```

/*****
/*      THIS IS THE MAIN BODY OF THE PROGRAM      */
*****/

```

351 1 BEGIN:

CALL INIT;

```
362      1      CALL SNOSEQ(.STARTUP$HDR, SIZE(STARTUP$HDR));
```

```
363      1      FOREVER = TRUE;
```

354 1 CALL READ\$LINE;

```
365      1      CALL SOLROM;
```

```
356      1      DO WHILE FOREVER;
```

```
367      2          CALL READ$TXTAB;
```

```

368      2          IF TRTA THEN

```

```
359      2          CALL CHK$RXTA;
```

370 2 IF TRANS\$1\$RDY THEN

```
371      2          CALL LOOP2;
```

```

372      2      IF TRTA THEN

```

575 2 CALL CHKSXTR;

```
374      2      CALL READ$RXTAB;
```

```
375      2          IF SOMCHK <> 0 THEN
```

376 2 CALL CHK\$RXTR;

377 2 CALL READ\$LINE;

573 2 END;

```
379      1      CALL ENDSEQ(.MSG7, SIZE(MSG7));
```

```

530      1      CALL EXIT;

```

```

31      !      END MAIN;

```

/***** THE END *****/

MODULE INFORMATION:

CODE AREA SIZE = 0061H 3425D

PORTABLE AREA SIZE = OAECH 27340

MAXIMUM STACK SIZE = 0000CH 120

325 - YES READ

1) PROGRAM ERPOP(S)

END OF PL M-60 COMPIATION

5. SBC 544 Monitor

The purpose of this program is to provide a monitor capability on the SBC 544 board. The functions included in this monitor are similar in function and use to CP/M tools DDT.COM and SID.COM. This monitor is adapted from the monitor source code for the SBC 86/12A board. The original monitor cannot be published in this document as it is copyrighted material from Intel. The original source code is available, however, in the Computer Networks Laboratory on an ISIS disk. For information purposes, a similar monitor can be developed for the SBC 88/45 board by starting with the original SBC 86/12A monitor and adapting the I/O port numbers to those on the SBC 88/45 board.

EXTERNAL SYMBOLS

USER SYMBOLS

BDOS	C 000F	BDOSO	A 0005	BIT7	C 040A	BIT3	C 040B	BITS1	C 0317
BITS2	C 0348	BITS7	C 0006	BITSA	C 0009	BTBL	C 03F6	CHOIC	C 0398
CLEAR	C 0412	CLRCM	C 011F	CMCHK	C 0122	CMIN	C 0127	CMOUT	C 0131
CMOUTO	C 0025	CONIN	C 0019	CONINO	C 0023	CONOUT	C 001D	CONST	C 0015
CONSTO	C 0021	CR	A 000D	CTSMK	A 0010	CXRMSK	A 0080	DGRMSK	A 0020
DTRMSK	A 0001	ERROR	A 000E	EVPAR	C 040F	EXIT	C 0012	INBUF	C 0414
INIT	C 0027	INIT1	C 00A1	INIT2	C 0076	INIT3	C 0081	INIT4	C 008C
INIT5	C 0046	LF	A 000A	MCTLA	C 0411	MENU	C 0204	MSG5	C 014F
NOPAR	C 040E	ODPAR	C 0410	OTWT	C 0131	PAR	C 037A	PARE	C 0100
PARN	C 00FA	PARX	C 0103	RATE	C 0249	RETURN	C 013B	RIMSK	A 0040
RTSMK	A 0002	RXRDY	A 0001	SAVIT	C 0094	SCLRCM	C 0003	SCMCHK	C 0006
SCMIN	C 0009	SCMOUT	C 000C	SDATA	A 0020	SIDENT	A 0022	SINIT	C 0000
SINTEN	A 0021	SLCTRL	A 0023	SLSTAT	A 0025	SMDMCT	A 0024	SMDMST	A 0026
START	C 0000	STP1	C 040C	STP2	C 040D	STPA	C 00DD	STPB	C 00E0
TXMTY	A 0020	ULCAS	C 013C	VALNUM	C 0145	WBVEC	A 0000	XOF	A 0013
CON	A 0011								

ASSEMBLY COMPLETE, NO ERRORS

```

                297 ;
LOC  OBJ      LINE      SOURCE STATEMENT

03F6 74        298 BTPL:  DB      1396 MOD 256      ;110 bps
03F7 05        299          DB      1396/256
03F8 00        300          DB      1024 MOD 256      ;150
03F9 04        301          DB      1024/256
03FA 00        302          DB      512 MOD 256       ;300
03FB 02        303          DB      512/256
03FC 00        304          DB      256 MOD 256       ;600
03FD 01        305          DB      256/256
03FE 80        306          DB      128 MOD 256      ;1200
03FF 00        307          DB      128/256
0400 40        308          DB      64 MOD 256       ;2400
0401 00        309          DB      64/256
0402 20        310          DB      32 MOD 256       ;4800
0403 00        311          DB      32/256
0404 10        312          DB      16 MOD 256       ;9600
0405 00        313          DB      16/256
0406 08        314          DB      8 MOD 256        ;19200
0407 00        315          DB      8/256
0408 04        316          DB      4 MOD 256        ;38400
0409 00        317          DB      4/256
                318 ;
040A 02        319 BIT7:  DB      00000010B        ;7 bit mask
040B 03        320 BIT8:  DB      00000011B        ;8 bit mask
040C 00        321 STP1:  DB      0                ;1 stop bit
040D 04        322 STP2:  DB      00000100B        ;2 stop bits
040E 00        323 NOPAR: DB      0                ;No parity
040F 18        324 EYPAR: DB      00011000B        ;Even parity
0410 08        325 ODPAR: DB      00001000B        ;Odd parity
0411 02        326 MCTLA: DB      00000010B        ;Modem control default (DTR off,
RTS (on)
                327 ;
0412 1A        328 CLEAR: DB      1AH,'S'            ;String to clear screen (Soroc IQ
120:
0413 24
                329 ;
0414 05        330 INBUF:  DB      5                ;Input buffer, max # chars
0415 00        331          DB      0                ;# chars in buff
0416          332          DB      5                ;Storage for chars
                333 ;
041B 656E6420  334          DB      'end GBS'
041F 834253
                335 ;
                336          END

PUBLIC SYMBOLS
BOOS  C 000F  SOMIN  C 0019  SOMOUT C 001D  CONST  C 0015  EXIT   C 0012
SOLROM C 0003  SOMCHK C 0006  SOMIN  C 0009  SOMOUT C 000C  SINIT  C 0000

```

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

0361 0A

LOC OBJ LINE SOURCE STATEMENT

```

288 ;
0362 20202064 289 DB ' d Parity '
0366 20202020
036A 20202050
036E 61726974
0372 79202020
0376 20202020
037A 3120 290 PAR: DB '1 '
037C 20202020 291 DB ' 1=None, 2=Even, 3=Odd',CR,LF,LF
0382 31204E6F
0384 6E652020
0388 32204576
038C 656E2020
0390 33204F64
0394 64
0396 00
0398 0A
039A 0A

```

```

292 ;
039B 53686F77 293 CHOIC: DB 'Show choice by letter, space or comma, number,
and CR',CR,LF

```

```

039C 2063686F
03A0 63636520
03A4 6279206C
03A8 65747465
03AC 72202073
03B0 70616365
03B4 206F7220
03B8 636F6060
03BC 6120206E
03C0 75606265
03C4 72202061
03C8 6E642043
03CC 52
03D0 00
03CE 0A
03CF 48697420 294 DB 'Hit CR when menu selections complete: $'
03D3 43522077
03D7 63656E20
03DB 60656E75
03DF 2073656C
03E3 65637469
03E7 6F6E7320
03EB 636F6070
03EF 60657465
03F3 3A2024

```

```

295 ;
296 ;Rate table for 2.45 MHz xtal

```

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

```

02D1 0D
LOC OBJ          LINE          SOURCE STATEMENT

02D2 0A
02D3 20202020      279          DB          '          9-38.4
      kbps',CR,LF,LF
02D7 20202020
02D8 20202020
02DF 20202020
02E3 20202020
02E7 20202020
02E9 20202020
02EF 2020392D
02F3 33382E34
02F7 206B6270
02FB 73
02FC 0D
02FD 0A
02FE 0A

      280 ;
02FF 20202062      281          DB          ' b      Data Bits '
0303 20202020
0307 20202044
030B 61746120
030F 42697473
0313 20202020
0317 3220          282 BITS1: DB          '2 '
0319 20202020      283          DB          ' 1-7 bit, 2-8 bit',CR,LF,LF
031D 312D3720
0321 6269742C
0325 20322D38
0329 20626974
032D 0D
032E 0A
032F 0A

      284 ;
0330 20202063      285          DB          ' c      Stop Bits '
0334 20202020
0338 20202053
033C 746F7020
0340 42697473
0344 20202020
0348 3120          286 BITS2: DB          '1 '
034A 20202020      287          DB          ' 1-1 bit, 2-2 bits',CR,LF,LF
034E 312D3120
0352 6269742C
0356 20322D32
035A 20626974
035E 73
035F 0D
0360 0A

```

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

022E 0D

LOC OBJ LINE SOURCE STATEMENT

022F 0A

0230 0A

0231 20202061 274 DB ' a Data Rate '

0235 20202020

0239 20202044

023D 61746120

0241 52617465

0245 20202020

0249 3720 275 RATE: DB '7 '

024B 20202020 276 DB ' 0- 110, 1- 150, 2- 300',CR,LF

024F 30202031

0253 31302020

0257 31202031

025B 35302020

025F 32202033

0263 3030

0265 0D

0266 0A

0267 20202020 277 DB ' 3- 600, 4-1200, 5-
2400',CR,LF

026B 20202020

026F 20202020

0273 20202020

0277 20202020

027B 20202020

027F 20202020

0283 20203320

0287 20363030

028B 20203420

028F 31323030

0293 20203520

0297 32343030

029B 0D

029C 0A

029D 20202020 278 DB ' 6-4800, 7-9600, 8-
19.2', CR,LF

02A1 20202020

02A5 20202020

02A9 20202020

02AD 20202020

02B1 20202020

02B5 20202020

02B9 20203620

02BD 34383030

02C1 20203720

02C5 39363030

02C9 20203820

02CD 31392E32

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

LOC	OBJ	LINE	SOURCE STATEMENT
018F	77697468		
0193	20746865		
0197	20554E49		
0199	44204949		
019F	20695342		
01A3	43203534		
01A7	3420496E		
01AB	7465726E		
01AF	65742050		
01B3	726F746F		
01B7	636F6C20		
01BB	23495029		
01BF	0D		
01C0	0A		
01C1	20202020	269	DB ' to simulate an actual host system.',CR,LF
01C3	746F2073		
01C9	696D756C		
01CD	61746520		
01D1	616E2061		
01D5	63747561		
01D9	6C20686F		
01DD	73742073		
01E1	79737465		
01E5	602E		
01E7	0D		
01E9	0A		
01E9	43697420	270	DB 'Hit any key to continue \$',CR,LF
01ED	616E7920		
01F1	6B657920		
01F5	746F2063		
01F9	6F6E7469		
01FD	6E756520		
0201	24		
0202	0D		
0203	0A		
		271 ;	
		272 MENU:	
0204	46656174	273	DB 'Feature Name Choice Option',CR,LF,LF
0208	75726520		
020C	20202020		
0210	204E616D		
0214	65202020		
0218	2043686F		
021C	69636520		
0220	20202020		
0224	20202020		
0228	4F707469		
022C	6F6E		

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

```

                239 ;
LOC OBJ          LINE      SOURCE STATEMENT

                240 ; Output byte to remote computer
                241 ;
                242 CMOUT:
                243 OTWT:
0131 DB25        244          IN      SLSTAT ;Get the transmit status
0133 E620        245          ANI     TXMTY
0135 CA3101      C 246          JZ     OTWT ;Not ready yet
0138 79          247          MOV     A,C ;It is now, send the byte
0139 D320        248          OUT     SDATA
013B C9          249 RETURN: RET      ;Default return
                250 ;
013C FE61        251 ULCAS: CPI      'a' ;Less than lower case a?
013E F8          252          RM      ;Yes
013F FE73        253          CPI      'z'+1 ;> lower case z?
0141 F0          254          RP      ;Yes
0142 D620        255          SUI     ' ' ;No, clear lower case bit
0144 C9          256          RET
                257 ;
0145 FE30        258 VALNUM: CPI      '0' ;Less than 0?
0147 37          259          STC
0148 F8          260          RM      ;Carry set; not valid number
0149 FE3A        261          CPI      ':' ;More than 9?
014B 37          262          STC
014C F0          263          RP      ;Carry set; not valid number
014D 3F          264          CMC
014E C9          265          RET      ;Carry cleared; valid number
                266 ;
014F 1A          267 MSGS: DB      1AH,'Configured for the CCS 2810 CPU (at 4 MHz),
                17 Aug 84',CR,LF
0150 436F6E66
0154 69677572
0158 65642066
015C 6F722074
0160 68652043
0164 43632032
0168 33313020
016C 43505520
0170 28617420
0174 34204040
0178 74292020
017C 31372041
0180 73672038
0184 34
0185 00
0186 0A
0187 576F726B    268          DB      'Working with the UNID II ISBC 544 Internet
                Protocol (IP)',CR,LF
018B 596E6720

```

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

LOC	OBJ	C	LINE	SOURCE STATEMENT
0100	3A0F04	C	190	PARE: LDA EYPAR ;Get even par mask
0103	B0		191	PARX: ORA B ;Add in 7/8 and 1/2 bits
0104	47		192	MOV B,A ;Resave
			193	;
			194	;
			195	; User provided initialization routine (Baud rate, etc.)
			196	****UART INITIALIZATION ROUTINE****
			197	;For WD or NS 8250 UART
			198	;
0105	341104	C	199	LDA MCTLA ;Get DTR, RTS, etc.
0108	D324		200	OUT SMDMCT ;Send it
010A	78		201	MOV A,B ;Get 7/8, 1/2, N/E/O bits
010B	F680		202	ORI 80H ;Set DLAB hi for data rate
010D	D323		203	OUT SLCtrl ;Send it
010F	7E		204	MOV A,M ;Get 1st byte
0110	D320		205	OUT SDATA ;Send it
0112	23		206	INX H
0113	7E		207	MOV A,M ;Get 2nd byte
0114	D321		208	OUT SINTEN ;Send it
0116	78		209	MOV A,B ;Reget 7/8, 1/2, N/E/O bits
0117	D323		210	OUT SLCtrl ;Send it
0119	AF		211	XRA A ;Init and clear last registers
011A	D321		212	OUT SINTEN
011C	D325		213	OUT SLSTAT
011E	C9		214	RET
			215	;
			216	; Clear communications channel
			217	; Do not output anything to the remote!!!!!!
			218	;
011F	D320		219	CLROM: IN SDATA
0121	C9		220	RET
			221	;
			222	; Check communications channel for anything
			223	; Return with accumulator=0 if nothing ready,
			224	; non-zero if a byte is waiting
			225	;
			226	CMCHK:
0122	D325		227	IN SLSTAT
0124	E601		228	ANI RARDY
0126	C9		229	RET
			230	;
			231	; Input byte from remote terminal
			232	;
			233	CMIN:
0127	D325		234	IN SLSTAT ;Added for speed
0129	E601		235	ANI RARDY ;added for speed
012B	CA2701	C	236	JZ CMIN ;added for speed
012E	D320		237	IN SDATA ;Get good data
0130	C9		238	RET

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

LOC	OBJ	LINE	INX	H	SOURCE STATEMENT
0095	23	141	INX	H	;Get selection
0096	7E	142	MOV	A,M	
0097	CD4501	C 143	CALL	VALNUM	;Is it a valid number?
009A	DA4600	C 144	JC	INITS	;Repeat if not
009D	12	145	STAX	D	;Save it
009E	C34600	C 146	JMP	INITS	;Another selection?
		147 ;			
00A1	3A4902	C 148	INIT1: LDA	RATE	;Get data rate
00A4	D630	149	SUI	'0'	;Make binary
00A6	FE0A	150	CPI	10	;Check for more than allowed
00A8	F24600	C 151	JP	INITS	;Repeat if so
00AB	07	152	RLC		;Double for two bytes/entry
00AC	21F603	C 153	LXI	H,BTBL	;Get data rate table addr
00AF	1500	154	MVI	D,0	;Init D
00B1	5F	155	MOV	E,A	;Put offset in E
00B2	19	156	DAD	D	;Add to table addr
		157 ;			
00B3	3A1703	C 158	LDA	BITS1	;Get data bits
00B6	FE31	159	CPI	'1'	;Check for 7 bit
00B8	CAC600	C 160	JZ	BITS7	;It's 7 bit
00BB	FE32	161	CPI	'2'	;Check for 8 bit
00BD	C24600	C 162	JNZ	INITS	;Error, do again
00C0	3A0B04	C 163	LDA	BIT8	;Get 8 bit mask
00C3	C3C900	C 164	JMP	BITSA	;Continue
00C6	3A0A04	C 165	BITS7: LDA	BIT7	;Get 7 bit mask
00C9	47	166	BITSA: MOV	B,A	;Save it
		167 ;			
00CA	3A4B03	C 168	LDA	BITS2	;Get stop bits
00CD	FE31	169	CPI	'1'	;Check for 1 bit
00CF	CAC600	C 170	JZ	STPA	;It's 1 bit
00D2	FE32	171	CPI	'2'	;Check for 2 bits
00D4	C24600	C 172	JNZ	INITS	;Error, start over
00D7	3A0C04	C 173	LDA	STP2	;Get 2 bit mask
00DA	C3E000	C 174	JMP	STPB	;Continue
00DD	3A0D04	C 175	STPA: LDA	STP1	;Get 1 bit mask
00E0	50	176	STPB: ORA	B	;Add 7/8 bit mask
00E1	47	177	MOV	B,A	;Resave it
		178 ;			
00E2	3A7A03	C 179	LDA	PAR	;get parity
00E5	FE31	180	CPI	'1'	;Check for none
00E7	DAFA00	C 181	JZ	PARN	;It's no parity
00EA	FE32	182	CPI	'2'	;Check for even
00ED	CAC600	C 183	JZ	PARE	;It's even parity
00EF	FE33	184	CPI	'3'	;Check for odd
00F1	C24600	C 185	JNZ	INITS	;Error, do again
00F4	3A1004	C 186	LDA	ODPAR	;Get odd par mask
00F7	C3C900	C 187	JMP	PARX	;Continue
00FA	3A1104	C 188	PARN: LDA	NOPAR	;Get no par mask
00FD	C3C900	C 189	JMP	PARX	;Continue

```

          92 ; Menu Initialization routines
LOC OBJ   LINE   SOURCE STATEMENT
          93 ;
          94 INIT:
0027 2A0100   95      lhd     wbrac + 1      ;Calculate the bios console
002A 110300   96      lxi     d,3          ; status, input, output vectors
002D 19      97      dad     d
002E 222100   C 98      shld    const0
0031 19      99      dad     d
0032 222300   C 100     shld    conin0
0035 19     101      dad     d
0036 222500   C 102     shld    conout0
          103 ;
0039 114F01   C 104     LXI     D,MSG5 ;CCS signon
003C 0E09     105     MVI     C,9
003E 0D0500     106     CALL    5
0041 0E01     107     MVI     C,1      ;Get keystroke
0043 0D0500     108     CALL    5
0046 111204   C 109 INITS: LXI     D,CLEAR ;Clear screen
0049 0E09     110     MVI     C,9
004B 0D0500     111     CALL    5
004E 110402   C 112     LXI     D,MENU
0051 0E09     113     MVI     C,9      ;Display menu selections
0053 0D0500     114     CALL    5
0056 111404   C 115     LXI     D,INBUF ;Get selection
0059 0E0A     116     MVI     C,10
005B 0D0500     117     CALL    5
005E 211504   C 118     LXI     H,INBUF+1 ;Check for null select
0061 7E      119     MOV     A,M
0062 B7      120     ORA     A
0063 2A0100   C 121     JZ      INIT1 ;No select, go initialize
0066 23      122     INX     H      ;Get feature
0067 7E      123     MOV     A,M
0068 0D3C01   C 124     CALL    ULCAS ;Make upper case
006B FE41     125     CPI     'A'      ;Data rate?
006D 114902   C 126     LXI     D,RATE ;Get save addr
0070 027600   C 127     JNZ     INIT2 ;No
0073 039400   C 128     JMP     SAVIT ;Save it
0076 FE42     129 INIT2: CPI     'B'      ;Data bits?
0078 111703   C 130     LXI     D,BITS1 ;Get save addr
007B 028100   C 131     JNZ     INIT3 ;No
007E 039400   C 132     JMP     SAVIT ;Save it
0081 FE43     133 INIT3: CPI     'C'      ;Stop bits?
0083 114803   C 134     LXI     D,BITS2 ;Get save addr
0086 028C00   C 135     JNZ     INIT4 ;No
0089 039400   C 136     JMP     SAVIT ;Save it
008C FE44     137 INIT4: CPI     'D'      ;Parity
008E 117A03   C 138     LXI     D,PAR ;Get save addr
0091 024600   C 139     JNZ     INITS ;No
0094 23      140 SAVIT: INX     H      ;Go past delimiter

```

LOC	OBJ	LINE	SOURCE STATEMENT
0001		43	DTRMSK EQU 00000001B ;DTR bit mask
0002		44	RTSMSK EQU 00000010B ;RTS bit mask
0010		45	CTSMSK EQU 00010000B ;CTS bit mask
0020		46	DSRMSK EQU 00100000B ;DSR bit mask
0040		47	RIMSK EQU 01000000B ;RI bit mask
0080		48	CXRMSK EQU 10000000B ;Received carrier detect
		49	;
0000		50	CR EQU 0DH
000A		51	LF EQU 0AH
0011		52	XON EQU 11H
0013		53	XOF EQU 13H
0005		54	bdos0 equ 5
0000		55	wbvec equ 0
		56	;
		57	cseg ;Relative assembly for ASM80
		58	;
		59	; Start of jump table is public
		60	;
		61	public sinit, sclrcm, scmchk, scmin, scmout
		62	;
		63	; Console, BDOS, and warm boot are public
		64	;
		65	public const, conin, conout, bdos, exit
		66	;
		67	; Jump table
		68	;
		69	START:
0000 032700	C	70	sinit: JMP INIT ;Initialization
0003 031F01	C	71	sclrcm: JMP CLRCM ;Clear comm channel
0006 032201	C	72	scmchk: JMP CMCHK ;Check channel
0009 032701	C	73	scmin: JMP CMIN ;Input byte
000C 033101	C	74	scmout: JMP CMOUT ;Output byte
		75	;
		76	; CP/M BDOS and BIOS call vectored jumps
		77	;
000F 030500		78	bdos: jmp bdos0 ;Bdos call
0012 030C00		79	exit: jmp wbvec ;Warm boot
		80	;
0015 2A2100	C	81	const: ihld const0 ;Bios console status
0018 E9		82	pcnl
0019 2A2300	C	83	conin: ihld conin0 ;Bios console input
001C E9		84	pcnl
001D 2A2500	C	85	conout: ihld conout0 ;Bios console output
0020 E9		86	pcnl
0021 0000		87	const0: dw 0 ;Bios console status vector
0023 0000		88	conin0: dw 0 ;Bios console input vector
0025 0000		89	conout0: dw 0 ;Bios console output vector
		90	;
		91	;

ISIS-II 8080/8085 MACRO ASSEMBLER, V4.1

LOC	OBJ	LINE	SOURCE STATEMENT
		1	;*****
		2	; LINK80 SUBROUTINES *
		3	;*****
		4	;
		5	; In directory as TSBS.ASM, SBS.ASM, or SBS.MOD
		6	; Set up for California Computer System 2810 CPU
		7	; (at 4 MHz) which uses the WD or NS 8250 UART
		8	; This software module was developed by Capt C. T.(Tom)
		9	; Childress on his own time and with his own resources
		10	; for use with his own CP/M system communications
		11	; program. It is a proven module of software. This
		12	; software and his system are used in his thesis as a
		13	; software validation tool for the SBC 544 board.
		14	; Any other use without prior written permission from
		15	; Capt Childress is strictly prohibited.
		16	;
		17	;Vers 1.0, 10 May 82
		18	;Vers 1.1, 25 Sep 82
		19	;Vers 1.2, 19 Oct 82
		20	;Vers 1.3, 11 Feb 83, New video board driver, 19.2 & 38.4 kbps
		21	;Vers 1.4, 24 Mar 83, Changed to 4MHz, ABCD drives
		22	;Vers 1.5, 3 Jul 83, Added XON/XOFF toggle for Terminal transmit
file mode		23	; Added Echo to remote for terminal mode
		24	;Vers 1.6, 30 Jul 84, Deleted the XON/OFF and Echo for use with
higher speeds		25	; in the terminal mode file transfer mode
		26	;Vers 1.7, 17 Aug 84, Modified to work with the UNID II ISBC 544
to support		27	; the Internet Protocol (IP) datagram exchange
running		28	; on the ISBC 544.
		29	;
		30	; EQUATES
		31	;
0020		32	SDATA EQU 20H ;Serial data port
0021		33	SINTEN EQU SDATA+1 ;Interrupt enable port
0022		34	SIDENT EQU SDATA+2 ;Interrupt identification port
0023		35	SCTRL EQU SDATA+3 ;Line control (+DLAB) port
0024		36	SMDCT EQU SDATA+4 ;Modem control port
0025		37	SLSTAT EQU SDATA+5 ;Line status port
0026		38	SMDST EQU SDATA+6 ;Modem status port
		39	;
0001		40	RXRDY EQU 0000001B ;Rx data available
0020		41	TXMTY EQU 00100000B ;Tx buffer empty
000E		42	ERRCR EQU 00001110B ;Error bit mask

4b. Assembly Language Module

The purpose of this module is to provide the main PL/M80 program with the communications to the UNID II and provide the console I/O interface with the CP/M BDOS. The module is written in Intel assembly language. It is independently written outside this thesis effort and was used here because it provides the known working high data rate (9600 bps) communications needed for the interaction with the UNID II.

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE MONITOR
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLM80 :F1:MON544.SRC NOOBJECT

\$TITLE('ISBC 544 MONITOR TEST, 1900 HRS 17 JULY 1984')
\$NOINTVECTOR

```

/*****
/*
/* DATE:          17 July 1984
/* VERSION:       1.0
/*
/* TITLE:         SBC 544 Monitor
/* FILE NAME:     SBC544.SRC
/* OWNER:         C. T. Childress
/*
/* SOFTWARE SYSTEM: Intel System III, ISIS II (V 4.2, 4.3)
/* USE:           Provide a bare minimum monitor capability
/*                for the SBC 544 board
/*
/* CONTENTS:      Source Code Listing
/*
*****/

/*****

```

ABSTRACT

THIS PROGRAM IS THE ROM BASED MONITOR FOR THE ISBC 544. IT PROVIDES THE USER WITH A MODERATE LEVEL OF CAPABILITY TO EXAMINE/MODIFY MEMORY AND REGISTERS, CONTROL PROGRAM EXECUTION, AND LOAD PROGRAMS.

ENVIRONMENT

THE SBC MONITOR COMMUNICATES WITH THE USER VIA AN INTERACTIVE TERMINAL (TTY,CRT) ATTACHED TO SERIAL PORT 0.

PROGRAM ORGANIZATION

THE PROGRAM IS DIVIDED INTO 1 DATA AND 2 CODE MODULES:

1. DATA DECLARATION MODULE. GLOBAL DATA DECLARATIONS.
2. COMMON ROUTINES. LOWER LEVEL PROCEDURES
3. COMMAND MODULE. INDIVIDUAL COMMANDS AND OUTER BLOCK

CALLING GRAPH

```

>>COMMAND DISPATCH MODULE (OUTER BLOCK)
  INDIVIDUAL COMMAND PROCEDURES
    COMMON ROUTINES

```

GLOBAL DATA STRUCTURES

THE MONITOR MAINTAINS THE USER'S MACHINE STATE (REGISTERS) IN A WORD ARRAY. THE REGISTERS ARE SAVED FROM THE USER'S STACK AS PUSHED BY PLM86 INTERRUPT PROCEDURE.

ADDRESSES TO THE 2**16 ADDRESS SPACE ARE IMPLEMENTED WITH ADDRESS STRUCTURES ALLOCATED AS TWO ADDRESS STRUCTURES.*/*

1 MONITOR:DO; /*BEGINNING OF MODULE*/

/******

GLOBAL DATA DECLARATIONS MODULE

ABSTRACT

THIS MODULE CONTAINS ALL THE GLOBAL DATA DECLARATIONS AND LITERALS (EQUATES).

MODULE ORGANIZATION

THE MODULE IS DIVIDED INTO 5 SECTIONS:

1. UTILITY SECTION GLOBAL FLAGS,VARIABLES,EQUATES
2. I/O SECTION I/O PORTS,MASKS,AND SPECIAL CHARS
3. MEMORY ARGUMENTS SECTIONS STRUCTURES FOR ADDRESSES
4. REGISTER SECTION USER REGISTER SAVE AREA

*/

/******

* UTILITY SECTION *

*****/*

2 1 R\$MASK: PROCEDURE BYTE EXTERNAL; /* USE THE PLM80.LIB WHEN LINKING */
3 2 END R\$MASK;

4 1 S\$MASK: PROCEDURE (MASK) EXTERNAL;
5 2 DECLARE MASK BYTE;
6 2 END S\$MASK;

7 1 DECLARE
INT\$VECTOR(8) ADDRESS, /* INTERRUPT VECTORS */
TRAP\$INT ADDRESS, /* POWER FAIL SENSE */
INT\$75 ADDRESS, /* TIMER INPUTS, TINT0 & TINT1 */
INT\$65 ADDRESS, /* RING INDICATOR AND CARRIER DETECT */
INT\$55 ADDRESS, /* FLAG (FINT) AND MULTIBUS INTERRUPT */
RIM\$MASK BYTE, /* INTERRUPT MASK RETURNED BY RIM */
TRAP\$INT\$PTR LITERALLY '024H', /* INTERRUPT LOCATIONS */
INT\$75\$PTR LITERALLY '03CH',
INT\$65\$PTR LITERALLY '034H',

```

                                INT$55$PTR      LITERALLY '02CH';

8   1   DECLARE
        MONITOR$STACKPTR ADDRESS;

9   1   DECLARE
        SIGNON$MSG(*) BYTE DATA (0DH,0AH,
        '                                ISBC 544 MONITOR, V1.0',0DH,0AH,0AH,
        '                                for the Universal Network Interface Device (UNID) 11',
        '                                0DH,0AH,0AH,
        '                                17 July 1984',0DH,0AH,0AH,0AH,0);

10  1   DECLARE
        START$MSG(*) BYTE DATA ('Start addr = ', 0DH,0AH,0),
        CHECK$SUM$ERR$MSG(*) BYTE DATA ('Check sum error', 0DH,0AH,0),
        MEM$RW$ERR(*) BYTE DATA ('Memory R/W error', 0DH,0AH,0);

11  1   DECLARE
        CHAR          BYTE,          /*ONE CHAR LOOK AHEAD*/
        CHECK$SUM      BYTE,          /*PAPER TAPE CHECKSUM*/
        I              BYTE,          /*INDEX*/
        J              BYTE,          /*INDEX*/
        II             ADDRESS,        /*INDEX*/
        JJ             ADDRESS,        /*INDEX*/
        END$OFF         ADDRESS,        /*END OFFSET ADDRESS*/
        WORD$MODE       BYTE,          /*ADDRESS MODE FLAG*/
        LAST$COMMAND    BYTE,          /*LAST COMMAND SAVE*/
        MODE            BYTE,          /*R MODE*/
        SAVE$MODE       BYTE,          /*SAVE MODE*/
        BRK             ADDRESS;        /*BAUD RATE FACTOR*/

12  1   DECLARE
        TRUE           LITERALLY 'OFFH',
        FALSE          LITERALLY '000H',
        MAX$DELAY       LITERALLY '60000', /*DELAY FOR READ CHAR*/
        TAPE            LITERALLY '1H',    /*TAPE MODE*/
        SERIAL          LITERALLY '2H',    /*SERIAL MODE*/
        PARALLEL        LITERALLY '4H',    /*PARALLEL MODE*/
        ASCR            LITERALLY '0DH',   /*CARRIAGE RETURN*/
        ASLF            LITERALLY '0AH',   /*LINE FEED*/
        ASBL            LITERALLY '20H';   /*BLANK OR SPACE*/

13  1   DECLARE
        ASCII(*)        BYTE DATA ('0123456789ABCDEF'),
        SIO$CMD(*)      BYTE DATA ('SGMDCFHRZ');

        /*****
        * I/O DECLARATIONS SECTION
        *****/

14  1   DECLARE
        /* 8251 USARTS */

```

```

SIO$PO$CMD      LITERALLY '001H', /* SERIAL PORT 0 COMMAND */
SIO$PO$STAT     LITERALLY '001H', /* SERIAL PORT 0 STATUS */
SIO$PO$DATA     LITERALLY '000H', /* SERIAL PORT 0 DATA */
SIO$P1$CMD      LITERALLY '003H', /* SERIAL PORT 1 COMMAND */
SIO$P1$STAT     LITERALLY '003H', /* SERIAL PORT 1 STATUS */
SIO$P1$DATA     LITERALLY '002H', /* SERIAL PORT 1 DATA */
SIO$P2$CMD      LITERALLY '005H', /* SERIAL PORT 2 COMMAND */
SIO$P2$STAT     LITERALLY '005H', /* SERIAL PORT 2 STATUS */
SIO$P2$DATA     LITERALLY '004H', /* SERIAL PORT 2 DATA */
SIO$P3$CMD      LITERALLY '007H', /* SERIAL PORT 3 COMMAND */
SIO$P3$STAT     LITERALLY '007H', /* SERIAL PORT 3 STATUS */
SIO$P3$DATA     LITERALLY '006H', /* SERIAL PORT 3 DATA */
SIO$MODE        LITERALLY '04EH', /* SERIAL PORT MODE */
SIO$COMMAND     LITERALLY '027H', /* SERIAL PORT COMMAND */

```

```

SIO$RESET$CMD   LITERALLY '40H', /* RESET USART */
SIO$CRT$MODE     LITERALLY '4EH', /* CRT MODE */
SIO$TTY$MODE     LITERALLY '0CFH', /* TTY MODE */
SIO$DTR$ON      LITERALLY '27H', /* RTS,RXE,DTR,TXE */
SIO$CRT$CMD      LITERALLY '37H', /* RTS,ER,RXE,DTR,TXE */
SIO$TTY$CMD      LITERALLY '35H', /* RTS,ER,RXE,TXE */
SIO$DTR$OFF     LITERALLY '25H', /* RTS,RXE,TXE */
SIO$RXRDY       LITERALLY '02H', /* RECEIVER READY */
SIO$TXE         LITERALLY '04H', /* TRANSMITTER EMPTY */
SIO$TXRDY       LITERALLY '01H', /* TRANSMITTER READY */
PARITY$MASK     LITERALLY '7FH', /* MASK OFF PARITY BIT */

```

```

15 1 DECLARE /* 8253 INTERVAL TIMER */
      IT1$CNT   LITERALLY '006H', /* INTERVAL TIMER 1 CONTROL */
      IT1$CNTR0 LITERALLY '008H', /* COUNTER 0, USART 0 */
      IT1$CNTR1 LITERALLY '009H', /* COUNTER 1, USART 1 */
      IT1$CNTR2 LITERALLY '00AH', /* COUNTER 2, USART 2 */
      IT2$CNT   LITERALLY '00FH', /* INTERVAL TIMER 2 CONTROL */
      IT2$CNTR0 LITERALLY '00CH', /* COUNTER 3, USART 3 */
      IT2$CNTR1 LITERALLY '00DH', /* COUNTER 4, CNTR5 OR SPLIT CLOCKS */
      IT2$CNTR2 LITERALLY '00EH', /* COUNTER 5, RST 7.5 */
      USART$CNTR$M3 LITERALLY '036H', /* DIVIDE BY N RATE GENERATOR, MODE 3, */
                                     /* FOR USART CLK x 16, CLK = 1.2288 MHZ */

```

```

B19200 LITERALLY '4', /* TIMER VALUE FOR 19.2 KBPS */
B9600  LITERALLY '8', /* TIMER VALUE FOR 9600 BPS */
B4800  LITERALLY '16', /* TIMER VALUE FOR 4800 BPS */
B2400  LITERALLY '32', /* TIMER VALUE FOR 2400 BPS */
B1200  LITERALLY '64', /* TIMER VALUE FOR 1200 BPS */
B600   LITERALLY '128', /* TIMER VALUE FOR 600 BPS */
B300   LITERALLY '256', /* TIMER VALUE FOR 300 BPS */
B150   LITERALLY '512', /* TIMER VALUE FOR 150 BPS */
B110   LITERALLY '698', /* TIMER VALUE FOR 110 BPS */

```

```

15 1 DECLARE /* 8155 PERIPHERAL INTERFACE */
      PI$PORTA LITERALLY '0E9H', /* PORT A (OUTPUT) */

```

```

PI$PORTB  LITERALLY '0EAH', /* PORT B (INPUT) */
PI$PORTC  LITERALLY '0EBH', /* PORT C (INPUT) */
PI$STAT   LITERALLY '0E8H', /* PPI STATUS */
PI$CMD    LITERALLY '0EBH', /* PPI COMMAND */
PI$CNTR$LO LITERALLY '0ECH', /* PPI COUNTER LO BYTE */
PI$CNTR$HI LITERALLY '0EDH', /* PPI COUNTER HI BYTE */
PI$CNTR$LOCNT LITERALLY '16384', /* PPI COUNTER TIME CONST */
PI$CNTR$HICNT LITERALLY '75', /* PPI COUNTER TIME CONST */
PI$INIT$CMD1 LITERALLY '041H', /* PPI INITIALIZATION COMMAND 1 */
/* A OUT, B & C IN, STOP COUNT */
PI$INIT$CMD2 LITERALLY '0C1H', /* PPI INITIALIZATION COMMAND 2 */
/* A OUT, B & C IN, START COUNT */
PI$INIT$$SIO$INT1 LITERALLY '0F0H', /* USART AND INT CONT RESET */
PI$INIT$$SIO$INT2 LITERALLY '0C0H', /* USART AND INT CONT NORMAL */

PI$PORTC$STAT LITERALLY '0CCH', /*PORT C STATUS*/
PI$PORTC$CTL  LITERALLY '0CEH', /*PORT C CONTROL*/
PI$M2M1      LITERALLY '0C6H', /*A-MODE 1, B-MODE 2*/
PI$OBF       LITERALLY '080H', /*OUTPUT BUFFER READY*/
PI$I BF      LITERALLY '02H'; /*INPUT BUFFER READY*/

```

```

17 1 DECLARE /*8259 INTERRUPT CONTROLLER*/
    IC$PORTA LITERALLY '0E6H', /* PORT A */
    IC$PORTB LITERALLY '0E7H', /* PORT B */
    IC$ICW1  LITERALLY '17H', /* INIT COMMAND ADDRESS 1 */
    IC$ICW2  LITERALLY '00H', /* INIT COMMAND ADDRESS 2 */
    IC$ICW3  LITERALLY '00H', /* INIT COMMAND ADDRESS 3 */
    IC$ICW4  LITERALLY '1DH', /* INIT COMMAND ADDRESS 4 */
    IC$MASK  LITERALLY '0FFH', /* INTERRUPT MASK, ICW1 */
    IC$EOI   LITERALLY '20H', /* END OF INTERRUPT CMD, ICW2 */
    IC$OCW3  LITERALLY '08H'; /* READ INTERRUPT LEVEL, ICW3 */

```

```

/*****
* REGISTER ASSIGNMENTS
*****/

```

```

18 1 DECLARE
    OS ADDRESS,
    IP ADDRESS;

/*****
* MEMORY ARGUMENT SECTION
*****/

```

```

19 1 DECLARE
    MEMORY$ARG1$PTR ADDRESS, /*ARGUMENT 1*/
    ARG1 STRUCTURE (OFF ADDRESS, SEG ADDRESS)
    AT (MEMORY$ARG1$PTR),
    MEMORY$ARG1 BASED MEMORY$ARG1$PTR BYTE,
    MEMORY$ADDRESS$ARG1 BASED MEMORY$ARG1$PTR ADDRESS,

```

```

MEMORY$ARG2$PTR ADDRESS,          /*ARGUMENT 2*/
ARG2 STRUCTURE (OFF ADDRESS, SEG ADDRESS)
    AT (.MEMORY$ARG2$PTR),
MEMORY$ARG2 BASED MEMORY$ARG2$PTR BYTE,

MEMORY$ARG3$PTR ADDRESS,          /*ARGUMENT3*/
ARG3 STRUCTURE (OFF ADDRESS, SEG ADDRESS)
    AT (.MEMORY$ARG3$PTR),
MEMORY$ARG3 BASED MEMORY$ARG3$PTR BYTE,

MEMORY$CSIP$PTR ADDRESS,          /*CS & IP ADDRESS*/
CSIP STRUCTURE (OFF ADDRESS, SEG ADDRESS)
    AT (.MEMORY$CSIP$PTR),
MEMORY$CSIP BASED MEMORY$CSIP$PTR BYTE,

MEMORY$USERSTACK$PTR ADDRESS,
USERSTACK STRUCTURE (OFF ADDRESS, SEG ADDRESS)
    AT (.MEMORY$USERSTACK$PTR),
MEMORY$USERSTACK BASED MEMORY$USERSTACK$PTR ADDRESS;

```

```

/*****
* REGISTER SECTION
*****/

```

```

/*****

```

COMMON PROCEDURES

ABSTRACT

THIS MODULE CONTAINS THOSE LOWER LEVEL PROCEDURES CALLED BY HIGHER LEVEL ROUTINES.

MODULE ORGANIZATION

THIS MODULE IS DIVIDED INTO 4 SECTIONS AS FOLLOWS:

1. BASIC I/O SECTION

```

SIO$CHAR$RDY      INPUT CHARACTER READY
SIO$CHECK$CONTROL$CHAR  CHECK FOR CONTROL CHARACTER
SIO$OUT$CHAR      OUTPUT CHARACTER
SIO$GET$CHAR      INPUT A CHARACTER
SIO$OUT$BYTE      OUTPUT A BYTE IN HEX
SIO$OUT$BYTE$ADDRESS  OUTPUT BYTE AT ADDRESS
SIO$OUT$WORD      OUTPUT A ADDRESS IN HEX
SIO$OUT$BLANK     OUTPUT A SINGLE BLANK
SIO$OUT$STRING    OUTPUT A STRING

```

2. UTILITY ROUTINES SECTION

```

SIO$VALID$HEX     TEST FOR VALID HEX CHAR
SIO$HEX           CONVERT TO HEX FROM ASCII
SIO$VALID$REG$FIRST  TEST FOR VALID REGISTER FIRST CHAR

```

```

SIO$VALID$REG      TEST FOR VALID REGISTER NAME
SIO$CRLF           OUTPUTS A CR AND LF
SIO$TEST$WORD$MODE TEST FOR A 'W' IN COMMAND
SIO$SCAN$BLANK     SCANS FOR OPTIONAL BLANK
SIO$SECOND$DELAY   DELAY ONE SECOND
SIO$MS$DELAY        DELAY N MS
3.ARGUMENT EXPRESSION EVALUATION SECTION
SIO$GET$WORD        GET A WORD EXPRESSION
SIO$GET$ADDR        GET AN ADDRESS EXPRESSION
SIO$UPDATE$IP       OPTIONAL UPDATE OF CS:IP
4.DEVICE INITIALIZATION SECTION
SIO$RESET$CMD$USART RESET 8251A
SIO$INIT$MODE       INITIALIZE FOR LOAD/TRANSFER
5.PAPER TAPE, SERIAL,PARALLEL READ SECTION
SIO$READ$CHAR       READ CHAR FROM TTY READER
SIO$READ$BYTE       READ A BYTE
SIO$READ$WORD       READ AN ADDRESS
SIO$READ$HEX$FILE   INPUT HEX FILE
6.INTERRUPT AND RESTORE/EXECUTE ROUTINES
SAVE$REGISTERS      SAVES USER REGISTERS
RESTORE$EXECUTE     RESTORE MACHINE STATE AND EXEC
!INTERRUPT1$ENTRY   INTERRUPT ROUTINE FOR SINGLE STEP
!INTERRUPT3$ENTRY   INTERRUPT ROUTINE FOR GO
!INTERRUPT32$ENTRY  INTERRUPT ROUTINE FOR 8259A
INIT$INT$VECTOR     INITIALIZES INTERRUPT VECTORS

```

```

*/
/*****
* BASIC I/O SECTION *
*****/

```

```

20 1  SIO$CHAR$RDY:
    /*TESTS FOR INPUT CHARACTER PENDING BY READING THE STATUS PORT
    AND MASKING WITH SIO$RXRDY(READ DATA READY.) RETURNS TRUE IF
    NOT EMPTY(CHAR PENDING) AND FALSE IF NO CHAR PENDING*/

    PROCEDURE BYTE;
21 2      IF (INPUT(SIO$PO$STAT) AND SIO$RXRDY) = 0 THEN RETURN FALSE;
23 2      RETURN TRUE;
24 2  END SIO$CHAR$RDY;

25 1  SIO$CHECK$CONTROL$CHAR:
    /*THIS ROUTINE CHECKS IF A CONTROL CHARACTER HAS BEEN INPUT TO
    THE SERIAL I/O PORT. AFTER A CONTROL-S IT WAITS FOR A CONTROL-Q
    BEFORE RETURNING TO THE CALLER. A CONTROL-C CAUSES A JUMP TO THE
    ERROR ROUTINE*/

    PROCEDURE;
26 2      CHAR = INPUT(SIO$PO$DATA) AND 07FH;
27 2      IF CHAR = 13H THEN          /*CONTROL-S*/
28 2          DO WHILE CHAR <> 11H;    /*CONTROL-Q?*/
29 3              IF SIO$CHAR$RDY THEN

```

```

30 3          DO;
31 4          CHAR = INPUT(SIO$PO$DATA) AND 07FH;
32 4          IF CHAR = 03H THEN GOTO ERROR;
34 4          END;
35 3          END;
36 2          ELSE IF CHAR = 03H THEN GOTO ERROR;
          END SIO$CHECK$CONTROL$CHAR;

39 1  SIO$OUT$CHAR:
      /*THIS ROUTINE OUTPUTS THE INPUT PARAMETER TO THE USART
      OUTPUT PORT WHEN USART IS READY FOR OUTPUT (XMIT BUFFER
      EMPTY).*/

      PROCEDURE(C);
      DECLARE C BYTE;
40 2          LOOP:  IF SIO$CHAR$RDY THEN CALL SIO$CHECK$CONTROL$CHAR;
41 2                  IF (INPUT(SIO$PO$STAT) AND SIO$TXRDY) = 0 THEN GOTO LOOP;
43 2                  OUTPUT(SIO$PO$DATA) = C;
45 2                  END SIO$OUT$CHAR;
46 2          END SIO$OUT$CHAR;

47 1  SIO$GET$CHAR:
      /*THIS ROUTINE INPUTS A CHARACTER FROM THE INPUT PORT AND RETURNS
      WITH IT IN THE GLOBAL 'CHAR'. THE CHARACTER IS ECHOED TO THE
      OUTPUT PORT IF PRINTABLE.*/
      PROCEDURE;
48 2          DO WHILE (INPUT(SIO$PO$STAT) AND SIO$RXRDY) = 0; END;
50 2          CHAR = INPUT(SIO$PO$DATA) AND 07FH;
51 2          IF CHAR>=ASBL THEN CALL SIO$OUT$CHAR(CHAR);
53 2          END SIO$GET$CHAR;

54 1  SIO$OUT$BYTE:
      /*THIS ROUTINE OUTPUTS THE SINGLE INPUT PARAMETER TO THE USART
      IN ASCII HEXADECEMAL FORMAT.*/
      PROCEDURE(B);
55 2          DECLARE B BYTE;
56 2          CALL SIO$OUT$CHAR(ASCII(SHR(B,4) AND 0FH));
57 2          CALL SIO$OUT$CHAR(ASCII(B AND 0FH));
58 2          END SIO$OUT$BYTE;

59 1  SIO$OUT$BYTE$PTR:
      /*THIS ROUTINE OUTPUTS THE BYTE BASED ON THE INPUT PARAMETER TO THE
      USART IN ASCII HEXADECEMAL FORMAT. */
      PROCEDURE (B$PTR);
60 2          DECLARE B$PTR ADDRESS, B BASED B$PTR BYTE, X BYTE;
61 2          X = B;
62 2          CALL SIO$OUT$BYTE(X);
63 2          END SIO$OUT$BYTE$PTR;

64 1  SIO$OUT$WORD:
      /* THIS ROUTINE OUTPUTS THE INPUT PARAMETER AS 4 ASCII HEXADECEMAL
      CHARACTERS TO THE USART OUTPUT PORT.*/

```

```

        PROCEDURE(W);
65  2      DECLARE W ADDRESS;
66  2      CALL SIO$OUT$BYTE(HIGH(W));
67  2      CALL SIO$OUT$BYTE(LOW(W));
68  2      END SIO$OUT$WORD;

69  1      SIO$OUT$BLANK:
        /*THIS ROUTINE OUTPUTS ONE BLANK. */
        PROCEDURE;
70  2      CALL SIO$OUT$CHAR(ASBL);
71  2      END SIO$OUT$BLANK;

72  1      SIO$OUT$STRING:
        /* OUTPUTS A STRING POINTED TO BY THE FIRST PARM. */
        PROCEDURE(PTR);
73  2      DECLARE PTR ADDRESS, STR BASED PTR (1) BYTE;
74  2      I = 0;
75  2      DO WHILE STR(I)<>0;
76  3          CALL SIO$OUT$CHAR(STR(I));
77  3          I = I + 1;
78  3      END;
79  2      END SIO$OUT$STRING;

        /*****
        * UTILITY ROUTINES SECTION*
        *****/

80  1      SIO$VALID$HEX:
        /*THIS ROUTINE TESTS IF THE INPUT PARAM IS A VALID ASCII HEX DIGIT
        AND RETURNS TRUE AS THE VALUE OF THE PROCEDURE IF SO AND FALSE
        IF NOT.*/
        PROCEDURE (H) BYTE;
81  2      DECLARE H BYTE;
82  2      DO I = 0 TO LAST(ASCII);
83  3          IF H=ASCII(I) THEN RETURN TRUE;
84  3      END;
85  3      RETURN FALSE;
86  2      END SIO$VALID$HEX;

88  1      SIO$HEX:
        /*THIS ROUTINE CONVERTS THE INPUT PARM FROM ASCII TO ITS BINARY
        EQUIVALENT AND RETURNS IT AS THE VALUE OF THE PROCEDURE. NO CHECK
        IS MADE FOR INPUT VALIDITY.*/
        PROCEDURE(C) ADDRESS;
89  2      DECLARE C BYTE;
90  2      IF C<='9' THEN RETURN DOUBLE(C-30H);
91  2      ELSE RETURN DOUBLE(C-37H);
92  2      END SIO$HEX;

94  1      SIO$CRLF:
        /* THIS ROUTINE OUTPUTS A CR AND LF TO THE OUTPUT PORT.*/

```

PL/M-80 COMPILER ISBC 544 MONITOR TEST, 1900 HRS 17 JULY 1984

```

        PROCEDURE;
95  2      CALL SIO$OUT$CHAR(ASCR);
96  2      CALL SIO$OUT$CHAR$(ASLF);
97  2      END SIO$CRLF;

98  1      SIO$TEST$WORD$MODE:
        /* THIS PROCEDURE TESTS FOR A 'W' FOLLOWING THE COMMAND AND IF SO
        SETS THE FLAG 'WORD$MODE TO TRUE OR FALSE OTHERWISE. SCANS OFF
        OPTIONAL BLANK FOLLOWING COMMAND. */
        PROCEDURE;
99  2      WORD$MODE = FALSE;
100 2      CALL SIO$GET$CHAR;
101 2      IF CHAR = 'W' THEN
```

PL/M-80 COMPILER ISBC 544 MONITOR TEST, 1900 HRS 17 JULY 1984

PL/M-80 COMPILER ISBC 544 MONITOR TEST, 1900 HRS 17 JULY 1984

```
102 2          DO;
103 3          WORD$MODE = TRUE;
104 3          CALL SIO$GET$CHAR;
105 3          END;
106 2          IF CHAR = ASBL THEN
107 2              CALL SIO$GET$CHAR;
108 2      END SIO$TEST$WORD$MODE;

109 1      SIO$SCAN$BLANK:
          /*THIS ROUTINE IS CALLED AFTER A COMMAND LETTER TO SCAN OFF THE
          OPTIONAL BLANK.*/
          PROCEDURE;
110 2          CALL SIO$GET$CHAR;
111 2          IF CHAR = ASBL THEN
112 2              CALL SIO$GET$CHAR;
113 2      END SIO$SCAN$BLANK;

114 1      SIO$SECOND$DELAY:
          /*THIS ROUTINE CAUSES A DELAY OF APPROXIMATELY 1 SECOND.*/
          PROCEDURE;
115 2          DECLARE II ADDRESS;
116 2          DO II = 1 TO 0F000H; END;
118 2      END SIO$SECOND$DELAY;

119 1      SIO$MS$DELAY:
          /*THIS ROUTINE CAUSES A DELAY OF 1 OR MORE MILLISECONDS; THE NUMBER
          IS PASSED BY THE CALLER. THE DELAY IS APPROXIMATE.*/
          PROCEDURE (N);
120 2          DECLARE (N,I,J) BYTE;
121 2          DO I = 1 TO N;
122 3              DO J = 1 TO 55; END;
124 3          END;
125 2      END SIO$MS$DELAY;

          /*****
          * ARGUMENT EXPRESSION EVALUATION SECTION *
          *****/

126 1      SIO$GET$WORD:
          /*THIS ROUTINE READS CHARS FROM THE INPUT PORT AND EVALUATES
          AN EXPRESSION CONSISTING OF '+' AND OPERANDS OF HEX NUMBERS
          AND REGISTER NAMES.*/
          PROCEDURE ADDRESS;
127 2          DECLARE (SAVE,W) ADDRESS, (OPER,T) BYTE;
128 2          OPER = '+';
129 2          W = 0;
130 2          DO WHILE TRUE;
```

```

131 3      T = CHAR;
132 3      SAVE = 0;
133 3      IF NOT(SIO$VALID$HEX(T)) THEN GOTO ERROR;
135 3      DO WHILE SIO$VALID$HEX(CHAR);
136 4          SAVE = SHL(SAVE,4) + SIO$HEX(CHAR);
137 4          CALL SIO$GET$CHAR;
138 4      END;
139 3      EVAL:      IF OPER = '+' THEN
140 3          W = W + SAVE;
141 3      ELSE
142 3          W = W - SAVE;
143 3      IF CHAR = ASCR OR CHAR = ':' OR CHAR = ',' THEN
144 3          RETURN W;
145 3      IF CHAR = '+' OR CHAR = '-' THEN
146 3          OPER = CHAR;
147 3      ELSE
148 3          GOTO ERROR;
149 2      CALL SIO$GET$CHAR;
150 2      END;
151 2      END SIO$GET$WORD;

150 1      SIO$GET$ADDR:
      /* THIS ROUTINE ACCEPTS A VALID ADDRESS EXPRESSION CONSISTING
      OF AN OPTIONAL <SEG>: AND AN DISPLACEMENT. */
      PROCEDURE(PTR,DEFAULT$BASE);
151 2      DECLARE PTR ADDRESS, DEFAULT$BASE ADDRESS,
      ARG BASED PTR STRUCTURE (OFF ADDRESS, SEG ADDRESS);
152 2      ARG.SEG = DEFAULT$BASE;
153 2      ARG.OFF = SIO$GET$WORD;
154 2      IF CHAR = ':' THEN
155 2          DO;
156 3              CALL SIO$GET$CHAR;
157 3              ARG.SEG = ARG.OFF;
158 3              ARG.OFF = SIO$GET$WORD;
159 3              IF CHAR = ':' THEN GOTO ERROR;
160 3          END;
161 2      END;
162 2      END SIO$GET$ADDR;

163 1      SIO$GET$BYTE:
      /* THIS PROCEDURE IS CALLED TO INPUT HEX CHARACTERS FROM THE
      INPUT USART AND RETURN WITH THE BINARY VALUE */
      PROCEDURE BYTE;
164 2      DECLARE SAVE BYTE;

165 2      SAVE = 0;
166 2      CALL SIO$GET$CHAR;
167 2      IF NOT (SIO$VALID$HEX(CHAR)) THEN GOTO ERROR;
168 2      DO WHILE SIO$VALID$HEX(CHAR);
169 3          SAVE = SHL(SAVE,4) + SIO$HEX(CHAR);
170 3          CALL SIO$GET$CHAR;
171 3      END;
172 2      END;

```

```

173 2      RETURN SAVE;
174 2      END SIO$GET$BYTE;

175 1      SIO$UPDATE$IP:
          /*THIS PROCEDURE IS CALLED BY 'GO' TO OUTPUT THE
          CURRENT IP AND INSTRUCTION BYTE AND OPEN THE IP FOR INPUT.*/
          PROCEDURE;
176 2          CALL SIO$OUT$BLANK;
177 2          CALL SIO$OUT$WORD(IP);
178 2          CSIP.SEG = CS;
179 2          CSIP.OFF = IP;
180 2          CALL SIO$OUT$CHAR('-');
181 2          CALL SIO$OUT$BLANK;
182 2          CALL SIO$OUT$BYTE$PTR(MEMORY$CSIP$PTR);
183 2          CALL SIO$OUT$BLANK;
184 2          CALL SIO$GET$CHAR;
185 2          IF CHAR<>' ' AND CHAR<>ASCX THEN CALL SIO$GET$ADDR(.CSIP,CS);
186 2      END SIO$UPDATE$IP;

          /*****
          * DEVICE INITIALIZATION SECTION *
          *****/

188 1      INITIALIZE$BOARD:
          /* THIS PROCEDURE INITIALIZES THE ISBC 544 BOARD */
          PROCEDURE;

189 2      PIS$INIT:      /* 8155 PARALLEL INTERFACE INITIALIZATION */
          OUTPUT(PIS$CMD) = PIS$INIT$CMD1;
190 2      OUTPUT(PIS$PORTA) = PIS$INIT$SIO$INT1;
191 2      CALL SIO$MS$DELAY(100);
192 2      OUTPUT(PIS$PORTA) = PIS$INIT$SIO$INT2;
193 2      CALL SIO$MS$DELAY(100);
194 2      OUTPUT(PIS$CNTR$LO) = PIS$CNTR$LOCNT;
195 2      OUTPUT(PIS$CNTR$HI) = PIS$CNTR$HICNT;
196 2      OUTPUT(PIS$CMD) = PIS$INIT$CMD2;

197 2      USS$INIT:      /* INITIALIZE THE USARTS */
          OUTPUT(SIO$P0$CMD) = SIO$MODE;
198 2      OUTPUT(SIO$P0$CMD) = SIO$COMMAND;
199 2      OUTPUT(SIO$P1$CMD) = SIO$MODE;
200 2      OUTPUT(SIO$P1$CMD) = SIO$COMMAND;
201 2      OUTPUT(SIO$P2$CMD) = SIO$MODE;
202 2      OUTPUT(SIO$P2$CMD) = SIO$COMMAND;
203 2      OUTPUT(SIO$P3$CMD) = SIO$MODE;
204 2      OUTPUT(SIO$P3$CMD) = SIO$COMMAND;

205 2      ITI$INIT:      /* 8253 INTERVAL TIMER INITIALIZATION */
          OUTPUT(ITI$CNT) = USART$CNTR$M3;
206 2      OUTPUT(ITI$CNTR0) = LOW(BRF);
207 2      OUTPUT(ITI$CNTR0) = HIGH(BRF);

```

```

208 2      OUTPUT(IT1$CONT) = 40H OR USART$CNTR$M3;
209 2      OUTPUT(IT1$CNTR1) = LOW(BRF);
210 2      OUTPUT(IT1$CNTR1) = HIGH(BRF);
211 2      OUTPUT(IT1$CONT) = 80H OR USART$CNTR$M3;
212 2      OUTPUT(IT1$CNTR2) = LOW(BRF);
213 2      OUTPUT(IT1$CNTR2) = HIGH(BRF);
214 2      OUTPUT(IT2$CONT) = USART$CNTR$M3;
215 2      OUTPUT(IT2$CNTR0) = LOW(BRF);
216 2      OUTPUT(IT2$CNTR0) = HIGH(BRF);

217 2      IC$INIT:      /* 8259 INTERRUPT CONTROLLER INITIALIZATION */

/*THE FOLLOWING CODE INITIALIZES THE 8259A. THE INTERRUPTS ARE NOT
USED FOR THIS MONITOR. ALL INTERRUPTS ARE SET MASKED. */

      OUTPUT(IC$PORTA) = IC$ICW1;
218 2      OUTPUT(IC$PORTB) = IC$ICW2;
219 2      OUTPUT(IC$PORTB) = IC$ICW3;
220 2      OUTPUT(IC$PORTB) = IC$ICW4;
221 2      OUTPUT(IC$PORTB) = IC$MASK;
222 2      OUTPUT(IC$PORTA) = IC$EOI;
223 2      OUTPUT(IC$PORTA) = IC$OCW3;

/* INITIALIZE INTERRUPT VECTORS */
/* CALL INIT$INT$VECTOR(.INT$VECTOR(1),.INTERRUPT1$ENTRY);
CALL INIT$INT$VECTOR(.INT$VECTOR(2),.INTERRUPT3$ENTRY);
CALL INIT$INT$VECTOR(.INT$VECTOR(3),.INTERRUPT3$ENTRY);
DO I = 32 TO 39;
      CALL INIT$INT$VECTOR(.INT$VECTOR(1),.INTERRUPT32$ENTRY);
END;
*/
/* INT$SPTR = INT$VECTOR(3); SAVE VECTOR 3 */

224 2      END INITIALIZE$BOARD;

225 1      SIO$RESET$PO$CMD:
/* THIS PROCEDURE RESETS THE 8251A USART FOR PORT 0 */
PROCEDURE;
226 2      OUTPUT(SIO$PO$CMD) = SIO$MODE;
227 2      CHAR = 0; /*DELAY*/
228 2      OUTPUT(SIO$PO$CMD) = SIO$RESET$CMD;
229 2      CHAR = 0; /* DELAY */
230 2      OUTPUT(SIO$PO$CMD) = SIO$MODE;
231 2      CHAR = 0; /*DELAY*/
232 2      OUTPUT(SIO$PO$CMD) = SIO$COMMAND;
233 2      END SIO$RESET$PO$CMD;

234 1      SIO$INIT$MODE:
/* INITIALIZES THE PORT 0 USART DATA RATE TO BRF */
PROCEDURE;
235 2      CALL SIO$RESET$PO$CMD;

```

```

236 2      OUTPUT(IT1$CONT) = USART$CNTR$M3;
237 2      OUTPUT(IT1$CNTRO) = LOW(BRF);
238 2      OUTPUT(IT1$CNTRO) = HIGH(BRF);

239 2      END SIO$INIT$MODE;

      /*****
      * PAPER TAPE, SERIAL, PARALLEL READ SECTION*
      *****/

240 1      SIO$READ$CHAR:
      /*THIS PROCEDURE READS A CHARACTER FROM PORT 0 USART. */
      PROCEDURE BYTE;
241 2          DECLARE I1 ADDRESS;

242 2      LOOP:
      DO I1 = 1 TO MAX$DELAY;
243 3          IF SIO$CHAR$RDY THEN GOTO READY;
245 3          END;
246 2          GOTO ERROR;
247 2      READY:
      CALL SIO$CHECK$CONTROL$CHAR;
248 2          IF CHAR = 11H THEN GOTO LOOP; /* GET ANOTHER IF CTL-Q */
250 2          RETURN CHAR;
251 2      END SIO$READ$CHAR;

252 1      SIO$READ$BYTE:
      /* THIS ROUTINE READS TWO HEX BYTES AND RETURNS THEIR BINARY
      BYTE VALUE. */
      PROCEDURE BYTE;
253 2          DECLARE T BYTE;
254 2          T = LOW(SIO$HEX(SIO$READ$CHAR));
255 2          T = SHL(T,4) + LOW(SIO$HEX(SIO$READ$CHAR));
256 2          CHECK$SUM = CHECK$SUM + T;
257 2          RETURN T;
258 2      END SIO$READ$BYTE;

259 1      SIO$READ$WORD:
      /* THIS ROUTINE READS FOUR HEX BYTES AND RETURNS THEIR BINARY
      ADDRESS VALUE. */
      PROCEDURE ADDRESS;
260 2          DECLARE T BYTE;
261 2          T = SIO$READ$BYTE;
262 2          RETURN SHL(DOUBLE(T),3) + DOUBLE(SIO$READ$BYTE);
263 2      END SIO$READ$WORD;

264 1      SIO$READ$HEX$FILE:
      /* THIS ROUTINE IS CALLED BY THE READ COMMAND TO
      READ A HEX FILE. */
      PROCEDURE;
265 2          DECLARE (REC$TYPE,LEN,I,T) BYTE, OFFSET ADDRESS;

```

```

266 2      IF CHAR <> ASCR THEN
267 2          CALL SIO$GET$ADDR(.ARG2,0); /* GET BIAS ADDR */
      ELSE
268 2          ARG2.SEG,ARG2.OFF = 0;
269 2          ARG1.SEG = ARG2.SEG; /* SEGMENT FOR 8080 FORMAT FILE */
270 2          IF CHAR<>ASCR THEN GOTO ERROR;
272 2          CALL SIO$CRLF;
273 2      LOOP:
      DO WHILE SIO$READ$CHAR<>';':END;
275 2          CHECK$SUM = 0;
276 2          LEN = SIO$READ$BYTE;
277 2          OFFSET = SIO$READ$WORD;
278 2          ARG1.OFF = OFFSET + ARG2.OFF;
279 2          REC$TYPE = SIO$READ$BYTE;
280 2          IF REC$TYPE=03 THEN /* START ADDR TYPE */
281 2              DO;
282 3              CS = SIO$READ$WORD;
283 3              IP = SIO$READ$WORD;
284 3              END;
285 2          IF REC$TYPE=02 THEN /* EXTENDED ADDR TYPE */
286 2              ARG1.SEG = SIO$READ$WORD + ARG2.SEG;
287 2          IF REC$TYPE=01 THEN
288 2              IF OFFSET <> 0 THEN
289 2                  IP = OFFSET; /* EOF RECORD */
290 2          IF REC$TYPE=00 THEN /* DATA TYPE */
291 2              DO I = 1 TO LEN;
292 3                  MEMORY$ARG1 = SIO$READ$BYTE;
293 3                  T = MEMORY$ARG1;
294 3                  IF MEMORY$ARG1<>T THEN
295 3                      DO;
296 4                      CALL SIO$OUT$STRING(.MEM$RW$ERR);
297 4                      CALL SIO$CRLF;
298 4                      GOTO ERROR;
299 4                      END;
300 3                  ARG1.OFF = ARG1.OFF + 1;
301 3              END;
302 2          T = SIO$READ$BYTE; /* FETCH CHECKSUM */
303 2          IF CHECK$SUM<>0 THEN
304 2              DO;
305 3              CALL SIO$OUT$STRING(.CHECK$SUM$ERR$MSG);
306 3              GOTO ERROR;
307 3              END;
308 2          IF REC$TYPE<>01 AND LEN<>0 THEN GOTO LOOP; /* EOF */
310 2          MODE = SERIAL;
311 2          CALL SIO$OUT$STRING(.START$MSG);
312 2          CALL SIO$OUT$WORD(IP);
313 2          CALL SIO$CRLF;
314 2      END SIO$READ$HEX$FILE;

```

```

/*****
* INTERRUPT AND RESTORE/EXECUTE SECTION
*****/

```

```

/*****
COMMAND MODULE
*****

```

```

ABSTRACT
=====

```

THIS MODULE CONTAIN ALL THE COMMANDS IMPLEMENTED AS INDIVIDUAL PROCEDURES AND CALLED FROM THE OUTER BLOCK OF THE COMMAND DISPATCH LOOP.

```

MODULE ORGANIZATION
=====

```

THIS MODULE CONTAIN THE FOLLOWING SECTIONS:

1. COMMANDS SECTION

```

SIO$GO          GO
SIO$EXAM$MEM    SUBSTITUTE MEMORY
SIO$MOVE        MOVE
SIO$DISPLAY     DISPLAY BYTES
SIO$COMPARE     COMPARE MEMORY
SIO$FIND        FIND BYTE/ADDRESS
SIO$HEX$ARITH   PERFORM HEX ARITHMETIC
SIO$INPUT       INPUT PORT
SIO$OUTPUT      OUTPUT PORT
SIO$READ        READ DATA RECORDS
SIO$LOAD        LOAD FILE
SIO$FILL        FILL MEMORY WITH CONSTANT

```

2. COMMAND DISPATCH (OUTER BLOCK, MAIN PROGRAM LOOP)

```

NEXT$COMMAND    DISPATCH
ERROR           ERROR ROUTINE

```

```

*/
/*****
* COMMAND SECTION
*****/

```

```

315 1  SIO$GO:
      /* IMPLEMENTS THE 'GO' COMMAND. THE USER MAY SPECIFY A NEW
      IP:PC */
      PROCEDURE;
316 2  DECLARE EXECGO(*) BYTE DATA
      ( 060H, /* MOV H,B */
        069H, /* MOV L,C */
        0E9H ), /* PCHL */
      EXECGO$PTR ADDRESS DATA (.EXECGO);

317 2  CALL SIO$UPDATE$IP;
318 2  IF CHAR <> ASCII THEN GOTO ERROR;

```

```

320 2      IP = CSIP.OFF;
321 2      CS = CSIP.SEG;
322 2      CALL SIO$CRLF;
323 2      CALL EXECGO$PTR(IP);

324 2      END SIO$GO;

325 1      SIO$EXAM$MEM:
        /* IMPLEMENTS THE EXAMINE MEMORY COMMAND. */
        PROCEDURE;
326 2      DECLARE W ADDRESS;

327 2      CALL SIO$TEST$WORD$MODE;
328 2      CALL SIO$GET$ADDR(.ARG1,CS);
329 2      IF CHAR<>',' THEN GOTO ERROR;
331 2      DO WHILE TRUE;
332 3          CALL SIO$OUT$BLANK;
333 3          IF WORD$MODE THEN
334 3              CALL SIO$OUT$WORD(MEMORY$ADDRESS$ARG1);
          ELSE
335 3              CALL SIO$OUT$BYTE$PTR(MEMORY$ARG1$PTR);
336 3              CALL SIO$OUT$CHAR('-');
337 3              CALL SIO$OUT$BLANK;
338 3              CALL SIO$GET$CHAR;
339 3              IF CHAR=ASCR THEN RETURN;
341 3              IF CHAR<>',' THEN
342 3                  DO;
343 4                  W = SIO$GET$WORD;
344 4                  IF (CHAR <> ',') AND (CHAR <> ASCR) THEN GOTO ERROR;
346 4                  IF WORD$MODE THEN
347 4                      DO;
348 5                      MEMORY$ADDRESS$ARG1 = W;
349 5                      IF MEMORY$ADDRESS$ARG1<>W THEN GOTO ERROR;
351 5                      END;
          ELSE
352 4              DO;
353 5              MEMORY$ARG1 = LOW(W);
354 5              IF MEMORY$ARG1<>LOW(W) THEN GOTO ERROR;
356 5              END;
357 4              END;
358 3              IF CHAR=ASCR THEN RETURN;
360 3              IF WORD$MODE THEN
361 3                  ARG1.OFF = ARG1.OFF + 2;
          ELSE
362 3              ARG1.OFF = ARG1.OFF + 1;
363 3              CALL SIO$CRLF;
364 3              CALL SIO$OUT$WORD(ARG1.OFF);
365 3              END;
366 2      END SIO$EXAM$MEM;

367 1      SIO$MOVE:

```

```

/* IMPLEMENTS THE MOVE COMMAND. ACCEPTS 3 ARGUMENTS AND MOVES THE
   BLOCK OF MEMORY SPECIFIED BY ARG1-ARG2 TO ARG3. ARG2<ARG1 OR THERE
   IS A DIFFERENCE WHEN THE BYTE IS READ BACK, THEN ERROR. */
PROCEDURE;

```

```

368 2      CALL SIO$SCAN$BLANK;
369 2      CALL SIO$GET$ADDR(.ARG1,CS);    /* FIRST ARGUMENT */
370 2      IF CHAR<>',' THEN GOTO ERROR;
372 2      CALL SIO$GET$CHAR;
373 2      END$OFF = SIO$GET$WORD;          /* SECOND ARGUMENT */
374 2      IF END$OFF<ARG1.OFF THEN GOTO ERROR;
376 2      IF CHAR<>',' THEN GOTO ERROR;
378 2      CALL SIO$GET$CHAR;
379 2      CALL SIO$GET$ADDR(.ARG3,CS);    /* THIRD ARGUMENT */
380 2      IF CHAR<>ASC THEN GOTO ERROR;
382 2      CALL SIO$CRLF;
383 2      DO WHILE ARG1.OFF <= END$OFF;
384 3          MEMORY$ARG3 = MEMORY$ARG1;
385 3          IF MEMORY$ARG3<>MEMORY$ARG1 THEN GOTO ERROR;
387 3          ARG1.OFF = ARG1.OFF + 1;
388 3          ARG3.OFF = ARG3.OFF + 1;
389 3      END;

```

```

390 2      END SIO$MOVE;

```

```

391 1      SIO$DISPLAY:
      /* IMPLEMENTS THE DISPLAY BYTE COMMAND. IF CALLED WITH 1 PARAM THEN
         OUTPUTS A SINGLE BYTE. IF CALLED WITH 2 PARAMS THEN OUTPUTS THE RANGE
         BETWEEN THE TWO ADDRESSES. IF OFFSET<BEGIN THEN OUTPUTS ONLY A SINGLE
         BYTE. */

```

```

PROCEDURE;

```

```

392 2      DECLARE (X,Y) BYTE,
           T ADDRESS;

393 2      ARG2.OFF, ARG2.SEG = 0;
394 2      CALL SIO$GET$CHAR;
395 2      CALL SIO$GET$ADDR(.ARG1,CS);
396 2      IF CHAR=ASC THEN
397 2          END$OFF = ARG1.OFF;
      ELSE
398 2          DO;
399 3              IF CHAR<>',' THEN GOTO ERROR;
401 3              CALL SIO$GET$CHAR;
402 3              END$OFF = SIO$GET$WORD;
403 3              IF END$OFF < ARG1.OFF THEN GOTO ERROR;
405 3              IF CHAR <> ASC THEN GOTO ERROR;
407 3          END;
408 2      NEWLINE:
           ARG2.OFF = ARG1.OFF;
409 2      CALL SIO$CRLF;
410 2      CALL SIO$OUT$WORD(ARG1.OFF);

```

```

411 2      CALL SIO$OUT$BLANK;
412 2      Y = 0;
413 2      LOOP:
          CALL SIO$OUT$BYTE$PTR(MEMORY$ARG1$PTR);
414 2      IF ARG1.OFF >= END$OFF THEN
415 2          DO;
416 3          ARG1.OFF = ARG1.OFF + 1;
417 3          T = 0;
418 3          GOTO LOOP1;
419 3          END;
420 2      ARG1.OFF = ARG1.OFF + 1;
421 2      Y = Y + 1;
422 2      IF Y >= 4 THEN
423 2          DO;
424 3          CALL SIO$OUT$BLANK;
425 3          Y = 0;
426 3          END;
427 2      T = ARG1.OFF AND 000FH;
428 2      LOOP1:
          IF T=0 THEN
429 2          DO;
430 3          CALL SIO$OUT$BLANK;
431 3          CALL SIO$OUT$CHAR('*');
432 3          DO WHILE ARG2.OFF <= (ARG1.OFF - 1);
433 4          X = MEMORY$ARG2 AND 7FH;
434 4          IF (X >= ASBL) AND (X <= 7EH) THEN
435 4          CALL SIO$OUT$CHAR(X);
          ELSE
436 4          CALL SIO$OUT$CHAR('.');
437 4          ARG2.OFF = ARG2.OFF + 1;
438 4          END;
439 3          CALL SIO$OUT$CHAR('*');
440 3          IF ARG1.OFF >= END$OFF THEN RETURN;
441 3          GOTO NEWLINE;
442 3          END;
443 2          GOTO LOOP;
444 2
445 2      END SIO$DISPLAY;

446 1      SIO$COMPARE:
          /* IMPLEMENTS THE COMPARE COMMAND */
          PROCEDURE;

447 2      CALL SIO$SCAN$BLANK;
448 2      CALL SIO$GET$ADDR(,ARG1,CS);
449 2      IF CHAR <> ',' THEN GOTO ERROR;
450 2      CALL SIO$GET$CHAR;
451 2      END$OFF = SIO$GET$WORD;
452 2      IF END$OFF < ARG1.OFF THEN GOTO ERROR;
453 2      IF CHAR <> ',' THEN GOTO ERROR;
454 2      CALL SIO$GET$CHAR;
455 2

```

```

458 2      CALL SIO$GET$ADDR(.ARG3,CS);
459 2      IF CHAR <> ASCR THEN GOTO ERROR;
461 2      CALL SIO$CRLF;
462 2      LOOP:
          DO WHILE ARG1.OFF <= END$OFF;
463 3          IF MEMORY$ARG1 <> MEMORY$ARG3 THEN
464 3              DO;
465 4                  CALL SIO$OUT$WORD(ARG1.OFF);
466 4                  CALL SIO$OUT$BLANK;
467 4                  CALL SIO$OUT$BYTE$PTR(MEMORY$ARG1$PTR);
468 4                  CALL SIO$OUT$BLANK;
469 4                  CALL SIO$OUT$WORD(ARG3.OFF);
470 4                  CALL SIO$OUT$BLANK;
471 4                  CALL SIO$OUT$BYTE$PTR(MEMORY$ARG3$PTR);
472 4                  CALL SIO$CRLF;
473 4                  END;
474 3                  ARG1.OFF = ARG1.OFF + 1;
475 3                  ARG3.OFF = ARG3.OFF + 1;
476 3          END;
477 2      END SIO$COMPARE;

478 1      SIO$FIND:
          /* IMPLEMENTS THE FIND COMMAND */
          PROCEDURE;
479 2      DECLARE SEARCH$WORD ADDRESS;

480 2      CALL SIO$TEST$WORD$MODE;
481 2      CALL SIO$GET$ADDR(.ARG1,CS);
482 2      IF CHAR <> ',' THEN GOTO ERROR;
484 2      CALL SIO$GET$CHAR;
485 2      END$OFF = SIO$GET$WORD;
486 2      IF END$OFF < ARG1.OFF THEN GOTO ERROR;
488 2      IF CHAR <> ',' THEN GOTO ERROR;
490 2      CALL SIO$GET$CHAR;
491 2      SEARCH$WORD = SIO$GET$WORD;
492 2      IF CHAR <> ASCR THEN GOTO ERROR;
494 2      CALL SIO$CRLF;
495 2      LOOP:
          DO WHILE ARG1.OFF <= END$OFF;
496 3          IF WORD$MODE THEN
497 3              DO;
498 4                  IF MEMORY$ADDRESS$ARG1 = SEARCH$WORD THEN
499 4                      DO;
500 5                          CALL SIO$OUT$WORD(ARG1.OFF);
501 5                          CALL SIO$CRLF;
502 5                          END;
503 4                      END;
          ELSE
504 3              DO;
505 4                  IF MEMORY$ARG1 = LOW(SEARCH$WORD) THEN
506 4                      DO;

```

```

507 5          CALL SIO$OUT$WORD(ARG1.OFF);
508 5          CALL SIO$CRLF;
509 5          END;
510 4          END;
511 3          ARG1.OFF = ARG1.OFF + 1;
512 3          END;
513 2          END SIO$FIND;

514 1          SIO$HEX$ARITH:
          /* IMPLEMENTS THE HEX ARITHMETIC COMMAND */
          PROCEDURE;
515 2          DECLARE (W1,W2) ADDRESS;

516 2          CALL SIO$SCAN$BLANK;
517 2          W1 = SIO$GET$WORD;
518 2          IF CHAR <> ' ', ' ' THEN GOTO ERROR;
520 2          CALL SIO$GET$CHAR;
521 2          W2 = SIO$GET$WORD;
522 2          IF CHAR <> ASCR THEN GOTO ERROR;
524 2          CALL SIO$CRLF;
525 2          CALL SIO$OUT$WORD(W1+W2);
526 2          CALL SIO$OUT$BLANK;
527 2          CALL SIO$OUT$WORD(W1-W2);
528 2          END SIO$HEX$ARITH;

529 1          SIO$FILL:
          /* THIS PROCEDURE IMPLEMENTS THE FILL MEMORY WITH CONSTANT
             FUNCTION */
          PROCEDURE;
530 2          DECLARE TEMP BYTE;

531 2          CALL SIO$GET$CHAR;
532 2          CALL SIO$GET$ADDR(.ARG1, CS);
533 2          IF CHAR <> ' ', ' ' THEN GOTO ERROR;
535 2          CALL SIO$GET$CHAR;
536 2          CALL SIO$GET$ADDR(.ARG2, CS);
537 2          IF ARG2.OFF < ARG1.OFF THEN GOTO ERROR;
539 2          IF CHAR <> ' ', ' ' THEN GOTO ERROR;
541 2          TEMP = SIO$GET$BYTE;
542 2          IF CHAR <> ASCR THEN GOTO ERROR;
544 2          CALL SIO$CRLF;

545 2          DO WHILE ARG1.OFF <= ARG2.OFF;
546 3              MEMORY$ARG1 = TEMP;
547 3              IF MEMORY$ARG1 <> TEMP THEN GOTO ERROR;
549 3              ARG1.OFF = ARG1.OFF + 1;
550 3          END;
551 2          END SIO$FILL;

552 1          SIO$READ:
          /* THIS PROCEDURE IMPLEMENTS THE PAPER TAPE READ COMMAND */

```

```

PROCEDURE;
553 2    CALL SIO$SCAN$BLANK;
554 2    SAVES$MODE = TAPE OR SERIAL;
555 2    CALL SIO$READ$HEX$FILE;
556 2    END SIO$READ;

/*****
*      COMMAND DISPATCH MAIN PROGRAM LOOP      *
*****/

/* THE RST 5.5, 6.5, AND 7.5 ARE DISABLED
BY USING A SYSTEM CALL WITH THE PARTICULAR
SIM MASK AS THE PARAMETER. */

557 1    DISABLE;
558 1    CALL S$MASK(01FH); /* DISABLE RST 5.5, 6.5, 7.5, SOD */

559 1    *MONITOR$STACKPTR = STACKPTR;
560 1    CS,IP = 0;
561 1    WORD$MODE = FALSE;
562 1    MODE = SERIAL;
563 1    BRP = B9600;

/* THE FOLLOWING CODE SETS THE DATA RATE OF PORT 0. IT
INITIALIZES BOTH THE 8251A USART AND COUNTER 0 OF 8253
INTERVAL TIMER # 0. USARTS 1, 2, AND 3 ARE NOT USED FOR
THIS MONITOR. */

564 1    CALL INITIALIZE$BOARD;
565 1    CHAR = INPUT(SIO$P0$DATA);
566 1    CALL SIO$OUT$STRING(.SIGNON$MSG);

567 1    NEXT$COMMAND:

/* THIS IS THE PERPETUAL COMMAND LOOP WHICH DISPATCHES TO EACH
COMMAND, EACH OF WHICH IS A SEPARATE PROCEDURE. */

CALL SIO$CRLF;
568 1    CALL SIO$OUT$CHAR('-');
569 1    CALL SIO$GET$CHAR;
570 1    DO I=0 TO LAST(SIO$CMND);
571 2        IF CHAR=SIO$CMND(I) THEN GOTO DISPATCH;
572 2    END;
573 1    GOTO ERROR;
574 1    DISPATCH:
575 1        LAST$COMMAND = I;
576 1        DO CASE I;
577 2            CALL SIO$EXAM$MEM;          /* S */
578 2            CALL SIO$GO;                /* G */
579 2            CALL SIO$MOVE;              /* M */
580 2            CALL SIO$DISPLAY;           /* D */

```

PL/M-80 COMPILER ISBC 544 MONITOR TEST, 1900 HRS 17 JULY 1984

```
581  2          CALL SIO$COMPARE;      /* C */
582  2          CALL SIO$FIND;         /* F */
583  2          CALL SIO$HEX$ARITH;    /* H */
584  2          CALL SIO$READ;         /* R */
585  2          CALL SIO$FILL;         /* Z */
586  2          END;
587  1          GOTO NEXT$COMMAND;
```

588 1 ERROR:

/* THIS ROUTINE HANDLES ALL ERRORS DETECTED BY THE MONITOR AND
WILL OUTPUT THE ERROR PROMPT TO THE OUTPUT PORT. */

```
          MODE = SERIAL;
589  1          STACKPTR = MONITOR$STACKPTR;
590  1          CALL SIO$OUT$CHAR('#');
591  1          CALL SIO$MS$DELAY(50);
592  1          GOTO NEXT$COMMAND;

593  1      END MONITOR;              /* END OF MODULE */
```

MODULE INFORMATION:

```
CODE AREA SIZE      = 0CD8H    3288D
VARIABLE AREA SIZE = 006BH    107D
MAXIMUM STACK SIZE = 0012H    18D
1131 LINES READ
0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

VITA

Creed F. Childress, Jr., was born on 7 October 1943 in Santa Barbara, California. He graduated from South Lake Tahoe High School in 1961. He then attended the University of California at Berkeley, California State University at Sacramento, and the University of California at Santa Barbara before enlisting in the U.S. Air Force in October 1966 when he began training as a Radio Relay Maintenance Technician. After his initial training, he was assigned to the 2063 Communications Group at Lindsey Air Station, Wiesbaden, Germany, where in 1972, he was accepted into the Airmen Education and Commissioning Program. He attended the University of Michigan, Ann Arbor, Michigan, and graduated in May 1974 with a Bachelor of Science in Industrial and Operations Engineering and a Bachelor of Science in Electrical Engineering. He then attended Officer Training School and was commissioned in August 1974 and assigned to Keesler Technical Training Center, Keesler AFB, Mississippi, for further training in the Communications Engineer specialty. His next assignment began in June 1975 as an instructor in the Wideband Systems Evaluation School (AFCC) at Richards-Gebaur AFB, Grandview, Missouri. In November 1977, he was assigned as the Detachment Commander of Detachment 27 of the 2187 Communications Group in Martina Franca, Italy. Following that challenging assignment, he was assigned to the Plans Division of the Headquarters of the Air Force Operational Test and Evaluation Center, Kirtland AFB, New Mexico in December 1979. He entered the Air Force Institute of Technology in May 1983.

Permanent Address: 9121 Cecile Way
Sacramento, CA, 95826

REPORT DOCUMENTATION PAGE

1. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS			
2. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.			
4. CLASSIFICATION/DOWNGRADING SCHEDULE						
5. PERFORMING ORGANIZATION REPORT NUMBER(S) T/GE/ENG/84D-17			6. MONITORING ORGANIZATION REPORT NUMBER(S)			
7a. NAME OF PERFORMING ORGANIZATION College of Engineering		8a. OFFICE SYMBOL (If applicable) AFIT/EN		7b. NAME OF MONITORING ORGANIZATION		
8b. ADDRESS (City, State and ZIP Code) Wright Patterson AFB, OH 45433				7c. ADDRESS (City, State and ZIP Code)		
9. NAME OF FUNDING/SPONSORING ORGANIZATION		8c. OFFICE SYMBOL (If applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
10. ADDRESS (City, State and ZIP Code)				10. SOURCE OF FUNDING NOS.		
11. TITLE (Include Security Classification) Box 19				PROGRAM ELEMENT NO.		PROJECT NO.
				TASK NO.		WORK UNIT NO.
12. PERSONAL AUTHOR(S) Box 19						
13. TYPE OF REPORT Thesis		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Yr., Mo., Da) 1984 December		15. PAGE COUNT 331
16. SUPPLEMENTARY NOTATION						
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)			
17a. LD	17b. GROUP	17c. SUB. GR	Local Area Networks, Network Interface Protocols, UNID, ISO Reference Model, X.25, X.12, TCP/IP, <i>Programs, Language, Pascal Language, Pascal</i>			
9	05					
19. ABSTRACT (Continue on reverse if necessary and identify by block number)						
Title: CONTINUED DEVELOPMENT AND IMPLEMENTATION OF THE UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II IN THE DIGITAL ENGINEERING LABORATORY NETWORK (DELNET) Thesis Advisor: Dr. Gary Lamont Author: Creed T. Childress, Jr., BS EE, BS IOE, Capt USAF Approved for public release: IAW AFR 190-17. <i>William E. Wolaver</i> (Feb 8) Director Research and Professional Development Air Force Laboratory of Technology (ALC) Wright Patterson AFB OH 45433						
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT CLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED			
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Gary Lamont			22b. TELEPHONE NUMBER (Include Area Code) 513-255-3576		22c. OFFICE SYMBOL AT/EN	

Block 19 (cont)

Abstract

↓
This research effort describes the continued development of an improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two main hardware modules; a local module for the network layer software and a network module for the datalink layer software and physical layer interface. Each module is an independent single board computer (SBC) residing on an Intel multibus chassis, complete with its own memory (EPROM and RAM), serial link interfaces, and multibus interface. The local module is an iSBC 544 and the network module is an SBC 88/45. The network layer software supports the CCITT X.25 protocol, datagram option, and the data link layer software supports the CCITT X.25 LAPB (HDLC) protocol. This report documents the detailed hardware and software design, integration, validation and test of this system. *Original supplied by word included: → front p.*

END

FILMED

4-85

DTIC