DIGITAL RADIO TERMINAL USING PACKET SWITCHING TECHNIQUES

March 1978

A report for: Department of Communications, Ottawa, Ontario, Canada

TABLE OF CONTENTS

SUBJECT	PAGE #
Introduction	4
Abstract	5
Hardware Functional Description	7
Software Functional Description	7 & 9
Decision Tables	I3-24
Data Structure	25
Discussion	27
APPENDIX A * Hardware	29
APPENDIX B - Software Listing	32
Acknowledgements	52
References	53

LIST OF FIGURES

fig.I	Packet Format	6
fig.2	Block Diagram of a Terminal	8
fig.3	Flow Chart of Main Program	I0
fig.4	Flow Chart of Receiver Interrupt Routine	II
fig.5	Flow Chart of Keyboard Interrupt Routine	I2
fig.6	Data Structure	26

INTRODUCTION

The ever expanding need for radio communications, sobered by the finite frequency spectrum associated with current technology and the cost of expanding the frequency frontier, together generate a powerful incentive towards maximizing the use of the available spectrum.

The basic problem is to try to relegate as many "users" as possible to a given frequency "band". Numerous techniques have been developed in recent decades to utilize the "spectrum" more efficiently. This project is concerned with a system to allow many users to transparently communicate digital data on a single channel. This kind of system is well suited to applications where user reaction is slow enough such that many closely interleaved high speed digital messages can be sent in the time it takes the user to process his own message. This method is also functional for long messages if waiting time is unimportant.

The particular application for which this project was geared was a system of mobile radio terminals such as would be used in police work. Other applications include taxi companies, aviation, marine, etc. The general technique used is Digital Packet Switching, and in this case the switching control is preformed by autonomous microprocessors in each terminal. These processors also carry out normal computer terminal functions.

This paper deals with the hardware and software implementation of a prototype system consisting of two complete interactive digital radio terminals.

ABSTRACT

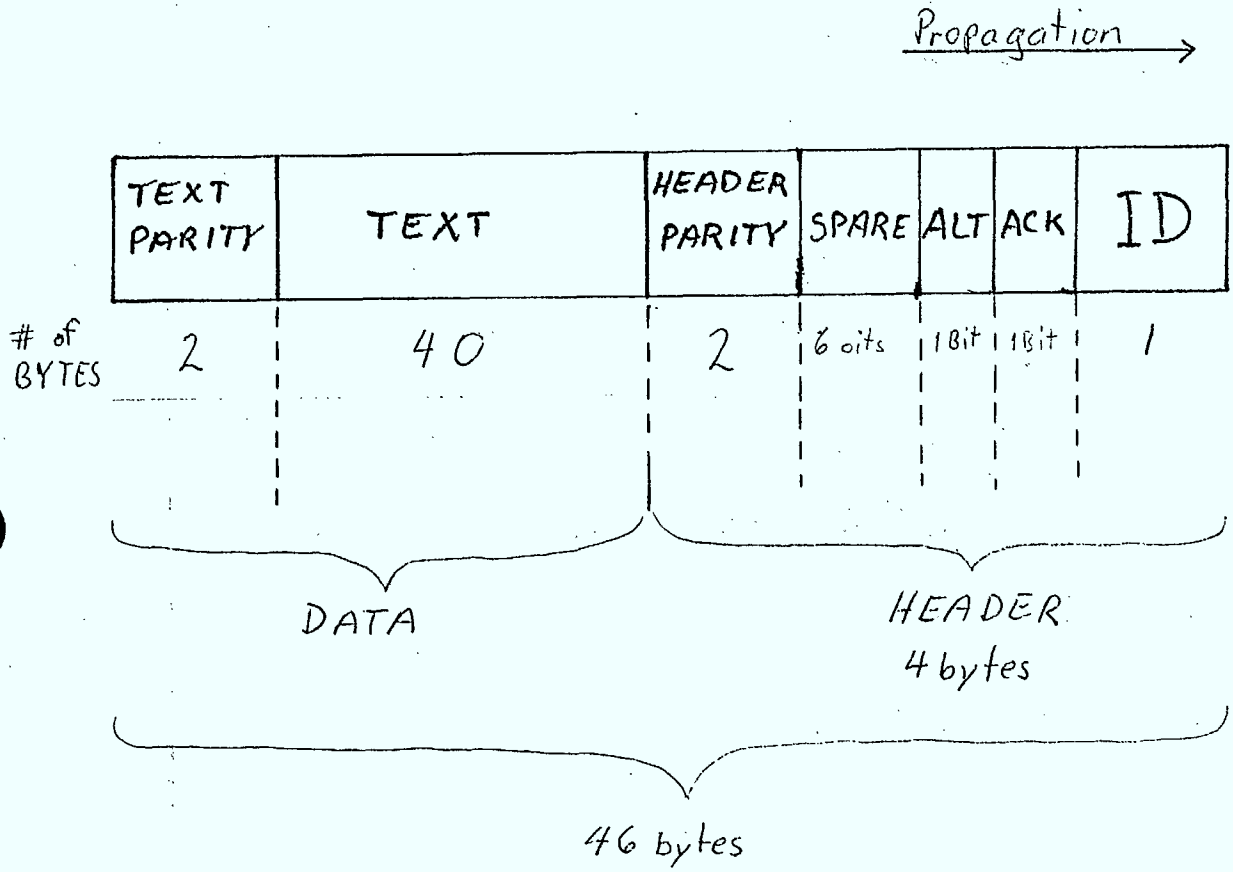
Given the autonomous nature of each terminal in a radio network, and since invalid data is received when any two terminals transmit simultanously, one of the first problems encountered with the proposed system was how to determine which terminal had priority to use the channel. It was proposed that if a message did not get through it could simply be retransmitted a short random time later provided no other units started transmitting first. The efficiency implications of this are being seperately worked out by S. Mahmoud et al.

Another problem with multiple terminals is determining which terminal sent the packet and which terminal it was meant for. This was solved by breaking the packet into two parts; a header section, which would contain various identification, and a data section for the actual message.

The Header format is detailed below (fig. 1).

Packet switching encompasses several techniques which allow multiple users to efficiently utilize a single channel by letting each user send a packet in turn. The specific switching technique which is used here to determine whose turn it is can be readily implemented on an autonomous terminal. The assumption is made that each user's need to transmit is randomly asynchronous and basically, whenthe need arises, the terminal simply waits for an opening then tries to transmit. If a conflict results it is resolved by each terminal waiting a random time before attempting to retransmit.

PACKET FORMAT



Two types of Packet:

1) Ack Pack - just header (to confirm reception of a text pack)

2) Text Packet - header and data

fig 1

The general hardware configuration is shown in fig. 2 (for details see Appendix A).

Most of the terminal functions can now be appreciated. These functions were defined as follows to enable a basic working prototype.

HARDWRE FUNCTIONAL DESCRIPTION

1. Each terminal system must receive and transmit digital data at 2400 band using FM at a carrier frequency of 172.71 MHz or 172.43 MHz.
2. The audio tones carrying the one-zero information were to be generated and decoded by a modem. set at Hz for a zero and Hz for a one.
3. The interface between the modem and computer was accomplished via a digital universal synchronous receiver transmitter (USART).
4. The processor was also interfaced to an alphanumeric keyboard for user input or control, and to a cathode-ray tube (CRT) via a video display board for output.

The basic software functions associated with controlling the various periferals and interpreting their input are summarized below.

SOFTWARE FUNCTIONAL DESCRIPTION

Receiving a Packet

1. A high priority interrupt is generated whenever the USART has received a valid byte. This succesion of bytes must be counted, stored, and checks must be preformed to determine:

   a. Identification of sender and receiver.

   b. Type of packet (Ack or Text).

   c. Validity and Priority check.

Appropriate actions must be taken for each contingency. These are outlined in detail in the decision tables on page .

SYSTEM BLOCK DIAGRAM of One TERMINAL

HARDWARE CONFIGURATION

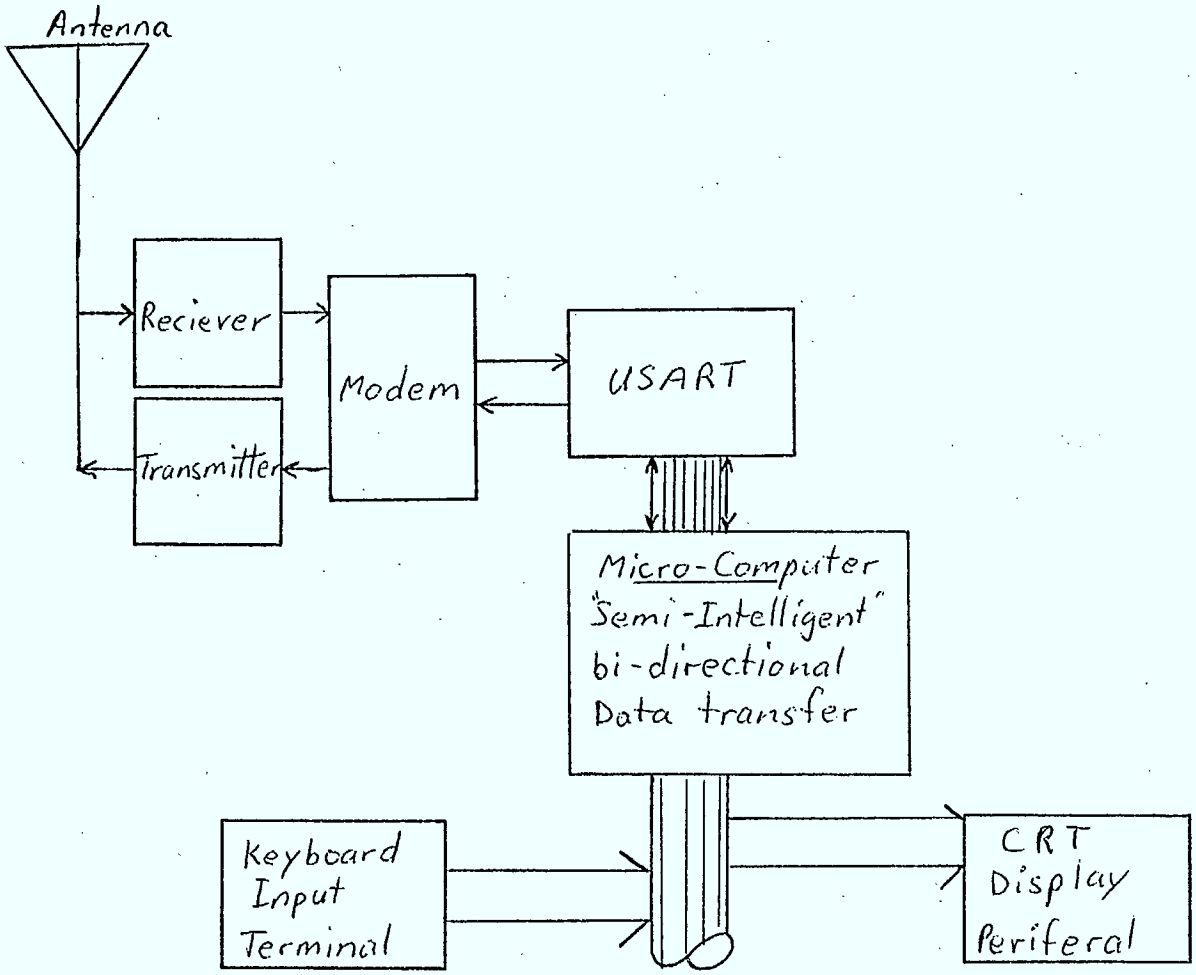


fig 2

Transmitting a Packet

1. Provision must be made to automatically format and send an Ack Pack upon receipt of a Packet.

2. Provision must also be made to manually enter text and a destination, then format and send a Text Packet repeatedly until an Ack comes back.

These routines are outlined in the decision tables on pages .

DATA DISPLAY

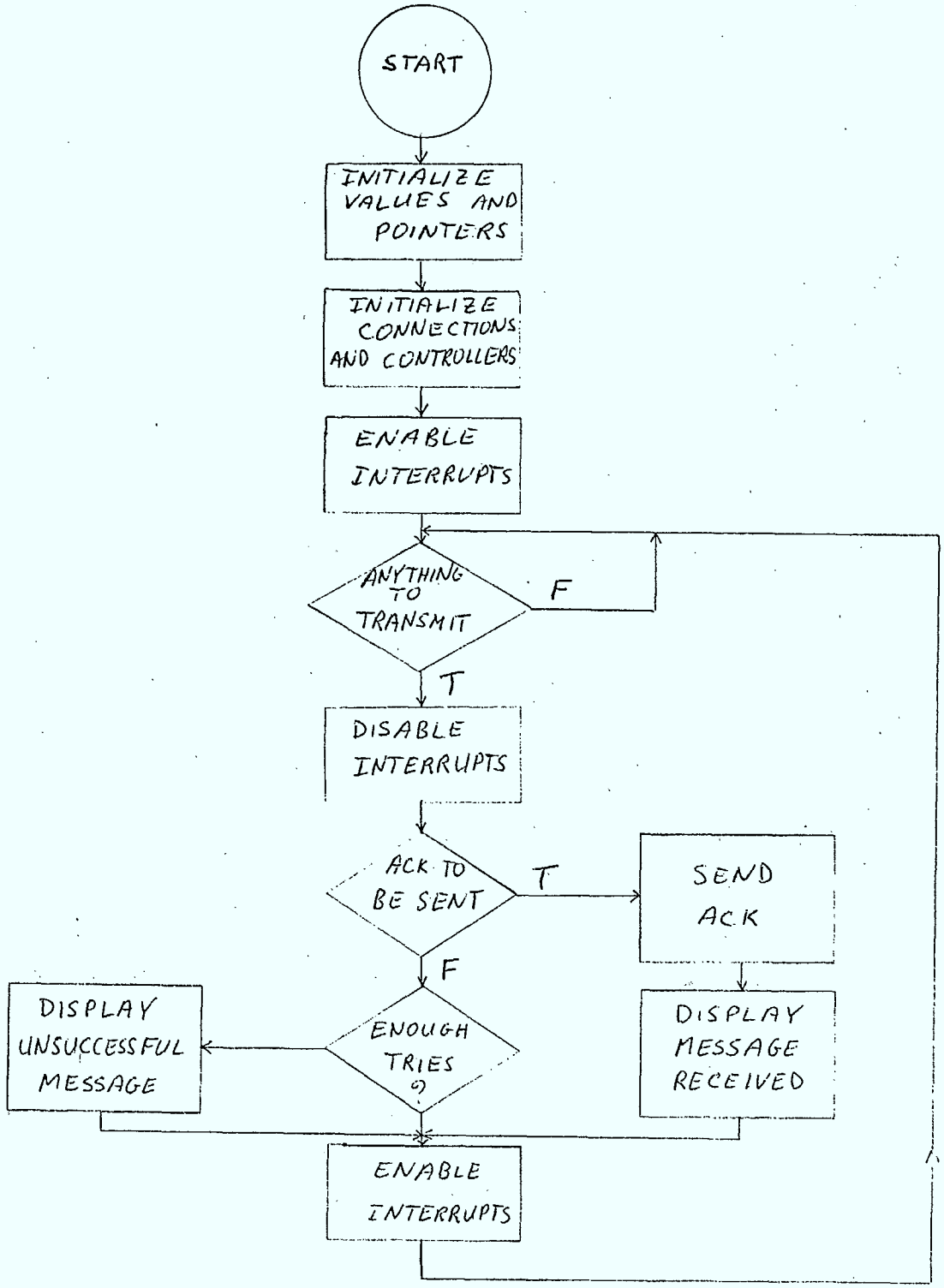
1. Text received must be displayed.

2. Text to transmit must be displayed.

3. A "failure to transmit" and other error messages must be displayed.

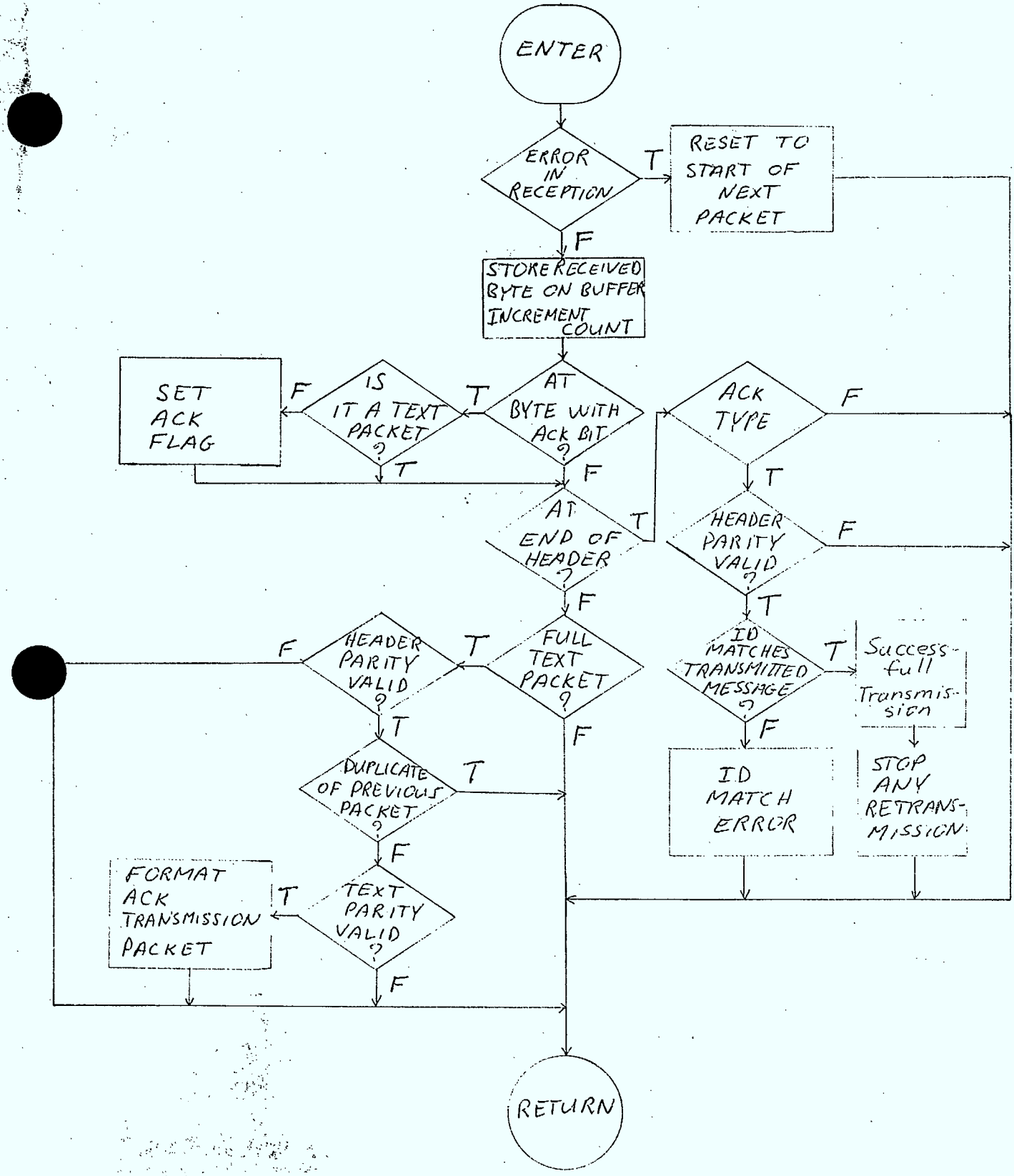
SOFTWARE BLOCK DIAGRAM

A flow chart is provided (figs. 3,4,5) to indicate how the various functions are carried out, to show the sequence and flow of control, and establish the priority of each decision and routine.



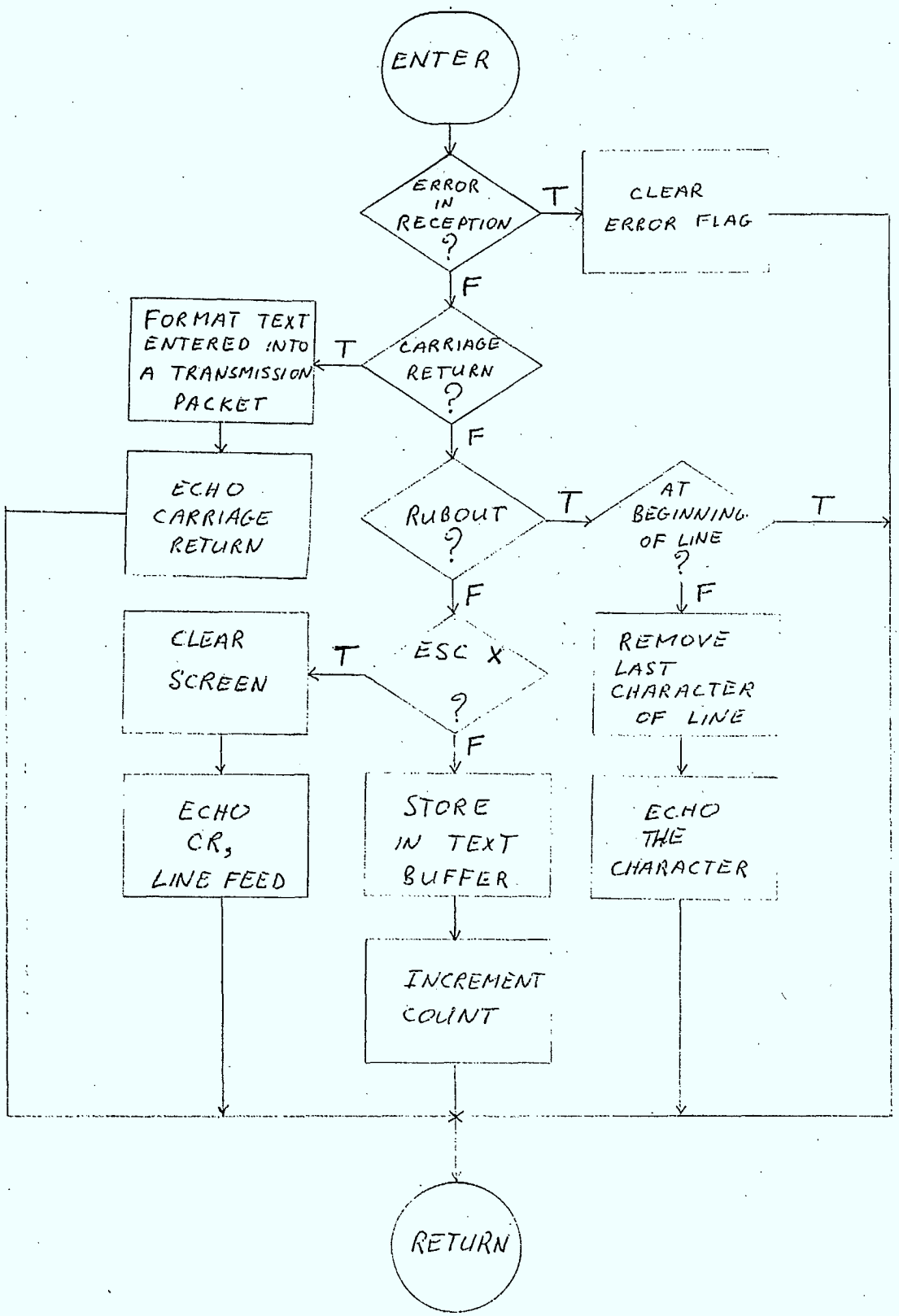
MAIN PROGRAM

fig. 3



RECEIVER INTERLIPT ROUTINE

fig. 4



KEYBOARD INTERRUPT ROUTINE

fig. 5

DECISION TABLES

The tables below provide a detailed flow of branch decisions and routine sequence which are almost directly translatable into program code.

In the upper block of each table are the branching decisions in sequence. Below each decision is the corresponding routine, defined as a series of operations to be performed in numerical order.

System Table

Do Initialization

Is there anything in Transmission List?	T	F
Do Transmission Table	1	
Repeat Table	2	1

Table 1

Transmission Table

Is it an Ack?	T	F	F
Enough Transmission tries?(tries = 0 for buffer at head of Transmission Lists?)		T	F
Determine size of packet to send	1		1
Do send packet routine	2		2
Do delay a random time			3
Display text of received packet	3		
Indicate unsuccessful transmission		1	
Free transmission buffer back to free list	4	2	
Enable receiver in hunt mode	5		4
Exit table	6	3	5

Table 2

Send Packet Table

Set pointer to first byte of head buffer of transmission list.

Enable transmitter, disabling receiver (transmitter not interrupt driven, only check for status).

Wait for transmitter ready flag of USRT.

Move synch character onto transmitter buffer.

Is pointer past number of bytes to transmit?	T	F
Wait for transmitter ready flag of USRT		1
Move byte onto transmitter buffer		2
Increment pointer		3
Repeat table		4
Decrement number of tries for buffer	1	
Enable reciever, disabling transmitter to prevent transmission (receiver interrupt driven).	2	
Exit table	3	

Table 3

Receiver Interrupt Routine

Read character from USART

Sync character received?	T	F
Received byte count < > 0?	T	
Return receiver buffer to free list	1	
Set byte count to 0	2	
Do data received routine		1
Reset interrupt environment	3	2
Return	4	3

Table 4

Data Received Routine — Decision Table

Receiver status indicate Error?	T	T	F	F	F	F	F	F	F
Received byte count value?	0	> 0	0	Ack byte	Ack byte	Header Size	Header size	Buffer Size	E
Ack bit set?				T	F				L
Ack type flag set?						T	F		S
									E
Get a buffer from free list and point receiver buffer pointer at it.			1						
Put byte on buffer of receiver			2	1	1	1	1	1	1
Increment received byte count			3	2	2		2		2
Set received byte count to 0		4				2		2	
Do Ack received table						3			
Do packet received table								3	
Set type flag to Ack				3					
Set type flag to text					3				
Error - print status value & byte count	1	1							
Clear error on receiver	2	2							
Free receiver buffer back to free list		3							
Enter hunt mode	3	5				4		4	
Return	4	6	4	4	4	5	3	5	3

Table 5

Ack Received Table

Received Ack pointer ← Received buffer pointer.

Is the parity of Ack valid?	T	T	F
Does the ID match any on transmission list?	T	F	
Remove matched buffer from transmission list	1		
Put matched buffer on free list	2		
Put received Ack buffer on free list	3	1	1
ERROR - no ID matched		2	
ERROR - parity of A ck invalid			2
Set received byte count to 0	4	3	3
Exit table	5	4	4

Table 6

Packet Received Table

Received packet pointer ← received buffer pointer.

Is header parity valid ?	**T**	**T**	**T**	**F**
Is it a duplicate of previously received message?	**T**	**F**		
Is text parity valid ?		**T**	**F**	
Do format Ack and ID (set A ck bit and tries count)		1		
Put buffer at top of transmission list		2		
ERROR - header party invalid				1
Display error and text			1	
Free buffer back to free list	1		2	2
Exit table	2	3	3	3

Table 7

Put at Front of Transmisssion List

Is header pointer null?	T	F
Tail pointer ← passed buffer pointer	1	
Link of buffer indicated by pointer passed on ← head pointer	2	1
Head pointer ← pointer passed on	3	2
Exit table	4	3

Table 8

Remove Buffer from Transmission List

Is previous buffer pointer null? (at front of list?)	T	F
Head pointer ← link of head buffer	1	
Tail pointer ← link of head buffer	2	
Link of buffer indicated by previous pointer ← link of buffer to be removed		1
Exit table	3	2

Table 9

Get From Free List

Is head pointer null?	T	F
ERROR - too few buffers	1	
Buffer pointer head pointer		1
Head pointer link of buffer that was head buffer		2
Return buffer pointer value		3
Exit table		4
Half	2	

Table 10

Put at Front of Free List

Link of buffer ← head pointer.
Head pointer ← buffer pointer passed on
Exit table

Put at End of Transmission List

Is head pointer null?	T	F
Head of list pointer ← buffer pointer passed on	1	
Link of tail ← buffer pointer passed on		1
Link of buffer indicated by passed pointer ← null	3	3
Tail pointer ← passed buffer pointer	2	2
Exit table	4	4

Table 11 + 12

DATA STRUCTURE

There are a number of functions which operate on blocks of data, as a matter of fact the primary object of the whole system is to manipulate data blocks.

Since these functions; which include sending packets, receiving packets, storing packets, displaying packets, and displaying error codes; play such an important roll in the system and require the single processor to cope with the multiple manipulations, it is desirable to work with a flexible yet efficient data structure.

The structure chosen is mapped out in fig. 6 and consists of a buffer array in memory for storage of data. Asociated with these buffers, as can be see, are several flags and lists which can be manipulated in place of the packets so that once a block of data is stored in a buffer it never has to be transfered again. "Transfers" are accomplished by altering the linked lists which contain "head" and "tail" pointers to each data block.

DATA STRUCTURE

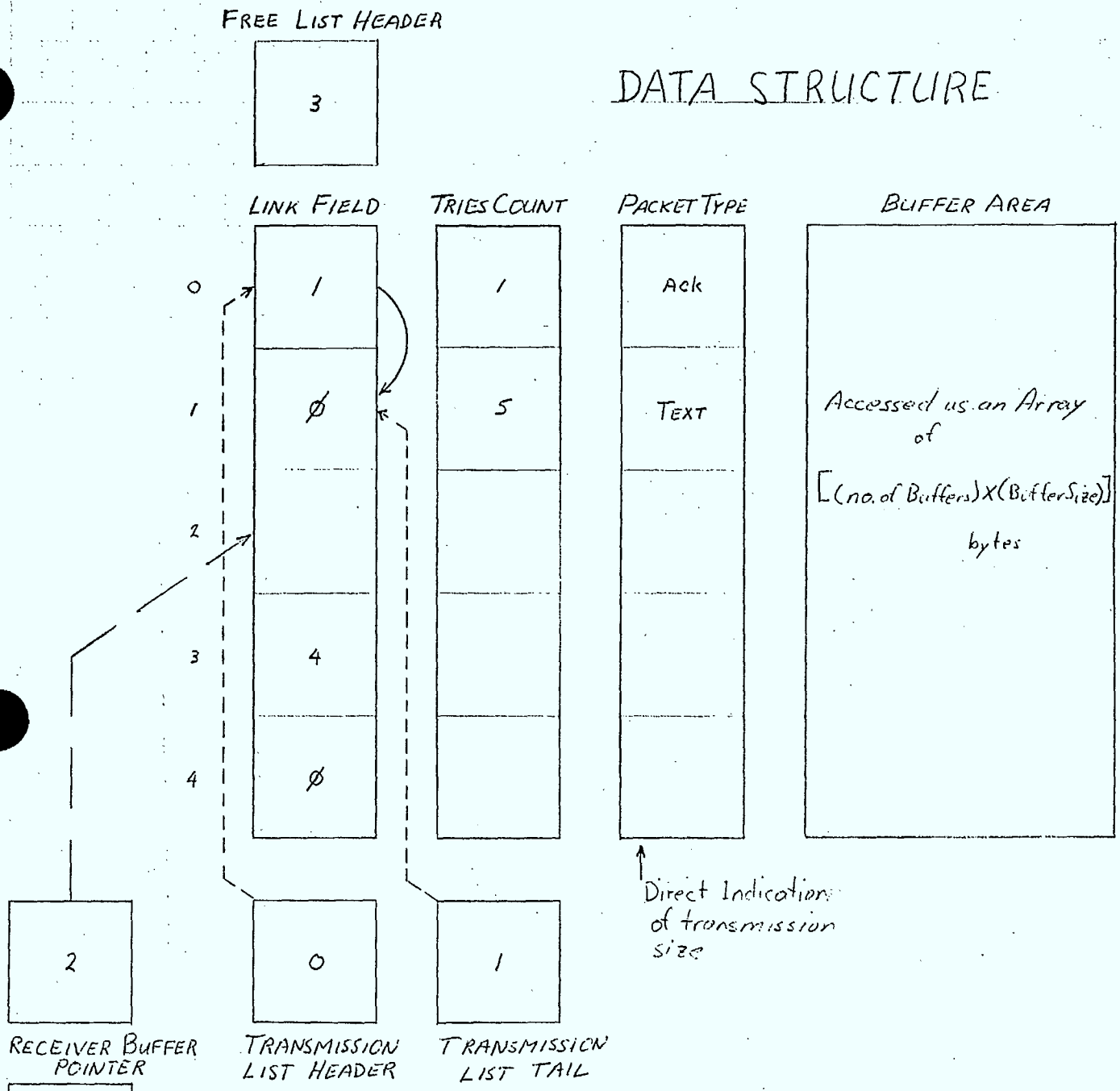


fig. 6

The software was written in PLM (a high level language for Micro Computers) to facilitate development. Intels Micro Computer Development System (MOS) was used as the primary debugging tool

The software is listed in Appendix B.

DISCUSSION

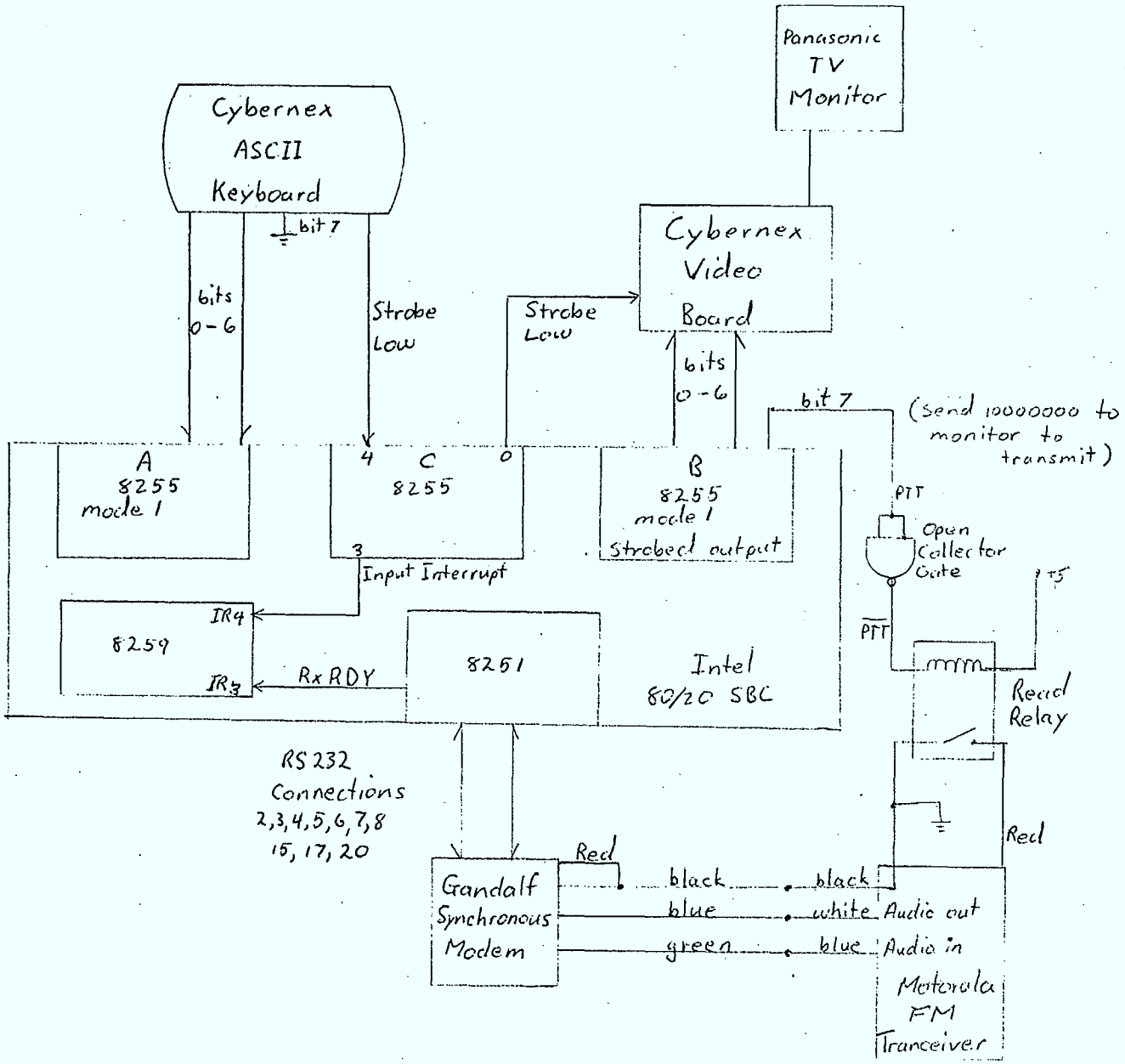
One terminal was made to be semi portable, consisting of:

1. Mains Power Supplies
2. 8020 Computer Board (Intel)
3. Video Display Board (Cybernex)
4. Keyboard (Cybernex)
5. Modem (Gandolf)
6. FM Tranceiver (Motorola)
7. TV Moniter (Panasonic)

The program was stored in EPROM on the 8020 Board. The other terminal was made by configuring the MDS to suit the application. The software has some minor differences in the I/O routine because the MDS interrupt structure is too rigidly fixed. Easily offsetting this is the indispensable debugging power afforded by the MDS.

A P P E N D I X A

Hardware Details of Mobile Terminal



MOBILE TERMINAL

DETAILED HARDWARE CONNECTIONS

fig. Aa

A P P E N D I X B

Software Listing From MDS

ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE PACKETSYSTEM
OBJECT MODULE PLACED IN :F1:PACKET.OBJ
COMPILER INVOKED BY: PLM80 :F1:PACKET.PLM

```
  1          PACKET$SYSTEM: DO;
  2    1         DECLARE DEC     LITERALLY 'DECLARE';
  3    1         DEC LIT     LITERALLY 'LITERALLY';
  4    1         DEC TRANSMITTER$CODE LIT    '0H';  /*EACH TERMINAL TO HAVE ITS OWN
                              4 BIT CODE NUMBER */
  5    1         DEC BUFFER$SIZE LIT '46';
  6    1         DEC NUMBER$BUFFERS  LIT '5';
  7    1         DEC BUFFER$SPACE    LIT '430'; /*NUMBER$BUFFERS * BUFFER$SIZE */
  8    1         DEC NULL        LIT '0FFH';
  9    1         DEC SET     LIT '1';
 10    1         DEC CLEAR       LIT '0';
 11    1         DEC FOREVER     LIT 'WHILE 1';
 12    1         DEC SYNCH$BYTE  LIT '24H';  /* PACKET SYNCHRONIZATION CHARACTER */
 13    1         DEC ACK$TYPE    LIT '0';    /* FLAG SETTINGS FOR TYPE$FLAG */
 14    1         DEC TEXT$TYPE   LIT '1';
 15    1         DEC PACKET$REPEAT$COUNT LIT '3';    /*DEBUG  MAX. RETRANS. ATTEMPTS */
 16    1         DEC TERMINAL$CONTROL    LIT '245'; /* MDSPATCH  CONTROL  PORT ADDRESS MDSPATCHED TO F7H */
 17    1         DEC TERMINAL$DATA   LIT '244';  /* MDSPATCH  DATA  PORT ADDRESS FOR INPUT */
 18    1         DEC MDS$CONTROL LIT '243';  /* PORT FOR MONITOR INTERRUPT CONTROL */
 19    1         DEC MDS$STATUS  LIT '250';  /* PORT FOR MONITOR INT STATUS */
 20    1         DEC TRANSCEIVER$CONTROL LIT '247';  /*  PORT FOR MODEM CONTROL */
 21    1         DEC TRANSCEIVER$DATA LIT '246'; /*  PORT FOR MODEM DATA */
       1         DEC LF      LIT '0AH';
       1         DEC CR      LIT '0DH';
 24    1         DEC RUBOUT      LIT '07FH';
 25    1         DEC XCHAR       LIT '058H';
 26    1         DEC ESC     LIT '01BH';
 27    1         DEC INTEL$8259$RESET    LIT '020H'; /*NON SPECIFIC END OF INTERRUPT */
 28    1         DEC INTEL$8259$CONTROL  LIT '253';  /*MDSPATCH*//*INTEL$8259 CONTROL PORT */
 29    1         DEC INTEL$8259$MASK LIT '252';  /*MDSPATCH*//*INTEL$8259 DATA PORT */
 30    1         DEC MAX$WAIT$FOR$FLAG LIT '1000';   /* NUMBER OF TIMES TO LOOP WAITING FOR FLAG */

           /*  U S A R T   D E F I N I T I O N S   */

 31    1         DEC R24AT1  LIT '00000011B';    /* 2400 BAUD AT JUMPER 1 */
 32    1         DEC R110    LIT '00000010B';    /* 110 BAUD AT JUMPER 3 */
 33    1         DEC R3AT2   LIT '00000011B';    /* 300 BAUD AT JUMPER 2 */
 34    1         DEC ST1 LIT '01000000B';    /* 1 STOP BIT */
 35    1         DEC ST2 LIT '11000000B';    /* 2 STOP BITS */
 36    1         DEC CL8 LIT '00001100B';    /* CHARACTER LENGTH 8 */
 37    1         DEC SINGLE$SYNC LIT '10000000B';
 38    1         DEC DOUBLE$SYNC LIT '00000000B';
 39    1         DEC SYNDET$OUTPUT   LIT '00000000B';
 40    1         DEC     DSR$BIT     LIT '10000000B';    /* USART DSR BIT */
 41    1         DEC TRAN$READY  LIT '00000001B';    /* USART TXRDY BIT */
 42    1         DEC CHAR$READY  LIT '00000010B';    /* USART RXRDY BIT */
       1         DEC TXEN    LIT '00000001B';    /* TRANSMIT ENABLE */
       1         DEC DTR LIT '00000010B';    /* DATA TERMINAL READY */
 45    1         DEC RXEN    LIT '00000100B';    /* RECEIVE ENABLE */
 46    1         DEC CLR$ERRS LIT'00010000B';    /* CLEAR ERRORS */
 47    1         DEC RESET   LIT '01000000B';    /* RESET USART */
```

```
 48  1          DEC RTS LIT '00100000B';     /* REQUEST TO SEND */
 49  1          DEC TXE LIT '00000100B';     /* TRANSMITTER BUFFER EMPTY */
 50  1          DEC HUNT$MODE LIT'10000000B';   /* ENTER HUNT MODE */
 51  1          DEC SYNDET  LIT '01000000B';    /* SYNC CHAR(S) FOUND */
     1          DEC ERROR$IN$RECEPTION LIT '00111000B'; /* FRAMING,PARITY OR OVERRUN ERRORS */


          /*  P A C K E T   F O R M A T   */


          /*  [            HEADER            ]              TEXT
                   EACH COLUMN ONE BIT        [   40 BYTES HERE    ]  ONE BIT PER COLUMN
               ------------------------------------------------------------------
               \TRAN    AA                                                      \
               \CODEBUFFCL*SPARE<====PARITY====><=40 BYTES OF TEXT=><=PARITY=>SPARE*\
               \    PTR KT                                                      \
               ------------------------------------------------------------------
          */
 53  1          DEC ACK$BYTE$INDEX  LIT '2';     /* INDEX INTO HEADER FOR ACK BYTE */
 54  1          DEC HEADER$LENGTH   LIT '4';     /* BYTE LENGTH OF FULL HEADER */
 55  1          DEC ACK$MASK    LIT '080H'; /* BIT INDICATING WHETHER ACK */
 56  1          DEC ID$INDEX    LIT '0';     /*BYTE INDEX TO ID OF PACKET */
 57  1          DEC ID$BYTE$LENGTH  LIT '1';     /* LENGTH OF ID */
 58  1          DEC TEXT$BYTE$INDEX LIT '4';     /* BYTE INDEX TO START OF TEXT */
 59  1          DEC TEXT$PARITY$BYTE$INDEX LIT '44'; /* BYTE INDEX TO PARITY FIELD */
 60  1          DEC HEADER$DATA$BYTE$LENGTH LIT '2'; /* LENGTH OF HEADER WITHOUT PARITY */


     1          DEC FREE$LIST$HEADER BYTE;
     1          DEC LINK$FIELD(NUMBER$BUFFERS) BYTE;
 63  1          DEC TRIES$COUNT(NUMBER$BUFFERS) BYTE;
 64  1          DEC BUFFER$AREA(BUFFER$SPACE) BYTE;
 65  1          DEC PACKET$TYPE(NUMBER$BUFFERS) BYTE;
 66  1          DEC (TRANS$LIST$HEADER,TRANS$LIST$TAIL,
                     RECEIVER$BUFFER$POINTER,
                     RECEIVED$BYTE$COUNT,
                     TEXT$INPUT$COUNT,
                     TEXT$BUFFER$POINTER,
                     CONTROL$COMMAND$FLAG,
                     RANDOM$SEED,
                     HEADER$PARITY$VALID,
                     TEXT$PARITY$VALID,
                     I,
                     GARBAGE$READ,
                     RECEIVED$BYTE,
                     TYPE$FLAG,
                     EDIT$FLAG,
                     USART$STATUS,
                     PACKET$SIZE     ) BYTE;
 67  1          DEC TEXT$PARITY$MATCH   ADDRESS; /* BOOLEAN MATCH VECTOR */
 68  1          DEC TEST$TRIES      ADDRESS; /* NUMBER OF TESTS FOR FLAG */
 69  1      /*DEBUG*/DEC WSTAT ADDRESS;/*DEBUG*/

          /*  P R I N T O U T   R O U T I N E   */

 70  1      PRINT:  PROCEDURE (CHAR$COUNT,MESSAGE$POINTER);
                 /* PRINTS OUT CHAR$COUNT CHARACTERS STARTING FROM ADDRESS
```

```
                INDICATED BY MESSAGE$POINTER */

 71  2          DEC (CHAR$COUNT, I, NOT$READY, USART$STATUS) BYTE;
 72  2          DEC TRIES ADDRESS;
 73  2          DEC MESSAGE$POINTER ADDRESS,
                    (MESSAGE BASED MESSAGE$POINTER) (1) BYTE;

 74  2          DO I = 0 TO CHAR$COUNT-1; /* LOOP THROUGH MESSAGE */
 75  3              USART$STATUS=INPUT(TERMINAL$CONTROL);
 76  3              NOT$READY=(USART$STATUS AND TRAN$READY)=CLEAR;
 77  3              TRIES=0; /* CHECK HOW MANY TRIES */

 78  3              DO WHILE NOT$READY;
 79  4                  USART$STATUS=INPUT(TERMINAL$CONTROL);
 80  4                  NOT$READY=(USART$STATUS AND TRAN$READY)=CLEAR;
 81  4                  TRIES=TRIES+1;
 82  4                  IF TRIES >= MAX$WAIT$FOR$FLAG THEN
 83  4                      HALT; /* CAN NOT DO MUCH ELSE AS TERMINAL GONE */
                        /* END IF */

 84  4              END;
 85  3              OUTPUT(TERMINAL$DATA)=MESSAGE(I);

 86  3          END;

             /** ENSURE THAT FINISHED TRANSMITTING **/
 87  2          USART$STATUS=INPUT(TERMINAL$CONTROL);

 88  2          DO WHILE (USART$STATUS AND (TRAN$READY OR TXE)) <> (TRAN$READY OR TXE);
 89  3              USART$STATUS=INPUT(TERMINAL$CONTROL);
 90  3          END;

 91  2          RETURN;
 92  2      END PRINT;


 93  1      HEXPRINT:   PROCEDURE (VALUE);
             /* PRINT OUT A BYTE VALUE USING TWO HEX ASCII CHARS */

 94  2          DECLARE (TEMP, VALUE) BYTE;

 95  2          TEMP=(ROR(VALUE,4)) AND 00001111B; /* MASK OFF UPPER NIBBLE */
 96  2          IF TEMP < 0AH THEN /* NUMERIC HEX */
 97  2              TEMP=TEMP+30H; /* RANGE 30H TO 39H */
             ELSE
 98  2              TEMP=TEMP+37H; /* RANGE 41H TO 46H */
             /*END IF */

 99  2          CALL PRINT(1,.TEMP);

100  2          TEMP=VALUE AND 00001111B; /* MASK OFF LOWER NIBBLE */
101  2          IF TEMP < 0AH THEN
102  2              TEMP=TEMP+30H;
             ELSE
103  2              TEMP=TEMP+37H;
             /*END IF*/
```

```
104  2          CALL PRINT(1,.TEMP);

105  2          RETURN;
106  2      END HEXPRINT;


           /*  E N T E R   H U N T   M O D E   I N   M O D E M   U S A R T  */

107  1      ENTER$HUNT: PROCEDURE;
108  2          DEC STATUS BYTE;

109  2          STATUS=INPUT(TRANSCEIVER$CONTROL);

               /** ENSURE FINISHED TRANSMITTING **/
110  2          DO WHILE (STATUS AND (TRAN$READY OR TXE)) <> (TRAN$READY OR TXE);
111  3              STATUS=INPUT(TRANSCEIVER$CONTROL);
112  3          END;

113  2          OUTPUT(TRANSCEIVER$CONTROL)=/*ASYNCH DEBUG HUNT$MODE OR*/  CLR$ERRS OR RXEN OR DTR;

114  2          GARBAGE$READ=INPUT(TRANSCEIVER$DATA);
115  2          GARBAGE$READ=INPUT(TRANSCEIVER$DATA);

116  2          RETURN;
117  2      END ENTER$HUNT;


           /*  L I S T   P R O C E S S I N G   R O U T I N E S  */


118  1      GETFROMFREE:    PROCEDURE BYTE;
                   /* GET A FREE BUFFER AREA FROM THE FREE LIST RETURNING
                     A POINTER INDEX TO THE BUFFER */
119  2          DEC (BUFFER$POINTER,I) BYTE;
120  2          DISABLE;


121  2          IF FREE$LIST$HEADER = NULL THEN
122  2              DO;
                   /****** NOT ENOUGH INITIAL BUFFER SPACE ERROR ******/
123  3              CALL PRINT (39,.('***** NOT ENOUGH INITIAL BUFFER SPACE',LF,CR));
124  3              HALT;
125  3              END;
               ELSE /* SOMETHING LEFT IN THE FREE LIST */
126  2              DO;
127  3              BUFFER$POINTER=FREE$LIST$HEADER;
128  3              FREE$LIST$HEADER=LINK$FIELD(BUFFER$POINTER);
129  3              END;
               /* END IF */
130  2          ENABLE;

131  2          RETURN BUFFER$POINTER;
132  2      END GETFROMFREE;

133  1      PUTATFRONTOFFREE:PROCEDURE (BUFFER$POINTER);
                   /* PUTS THE BUFFER INDICATED BY THE INDEX POINTER
                     AT THE FRONT OF THE FREE LIST */
```

```
134  2          DEC BUFFER$POINTER BYTE;

135  2          DISABLE;

     2          LINK$FIELD(BUFFER$POINTER)=FREE$LIST$HEADER;
     2          FREE$LIST$HEADER=BUFFER$POINTER;

138  2          ENABLE;
139  2          RETURN;
140  2      END PUTATFRONTOFFREE;

141  1      PUTATENDOFTRANS:PROCEDURE (BUFFER$POINTER);
                  /* PUTS THE BUFFER INDICATED BY THE INDEX POINTER
                    AT THE END OF THE TRANSMISSION LIST (LOWEST PRIORITY) */
142  2          DEC BUFFER$POINTER BYTE;

143  2          DISABLE;

144  2          IF TRANS$LIST$HEADER = NULL THEN
145  2              TRANS$LIST$HEADER=BUFFER$POINTER;
                 ELSE
146  2              LINK$FIELD(TRANS$LIST$TAIL)=BUFFER$POINTER;
                 /* END IF */

147  2          TRANS$LIST$TAIL=BUFFER$POINTER;
148  2          LINK$FIELD(BUFFER$POINTER)=NULL;

     2          ENABLE;
     2          RETURN;
     2      END PUTATENDOFTRANS;

152  1      PUTATFRONTOFTRANS:PROCEDURE (BUFFER$POINTER);
                  /* PUT THE BUFFER INDICATED BY THE INDEX POINTER
                    AT THE FRONT OF THE TRANSMISSION LIST (HIGHEST PRIORITY) */
153  2          DEC BUFFER$POINTER BYTE;

154  2          DISABLE;

155  2          IF TRANS$LIST$HEADER = NULL THEN
156  2              TRANS$LIST$TAIL=BUFFER$POINTER;
                 /*END IF*/

157  2          LINK$FIELD(BUFFER$POINTER)=TRANS$LIST$HEADER;
158  2          TRANS$LIST$HEADER=BUFFER$POINTER;

159  2          ENABLE;
160  2          RETURN;
161  2      END PUTATFRONTOFTRANS;

162  1      REMOVEFROMTRANS:PROCEDURE (PREV$BUF$POINTER,BUFFER$POINTER);
                  /* REMOVE THE BUFFER INDICATED BY THE INDEX POINTER
                    FROM THE TRANSMISSION LIST */
     2          DEC (PREV$BUF$POINTER,BUFFER$POINTER) BYTE;

     2          DISABLE;

165  2          IF PREV$BUF$POINTER = NULL THEN /* AT FRONT OF LIST */
```

```
166  2              DO;
167  3              TRANS$LIST$HEADER=LINK$FIELD(BUFFER$POINTER);
168  3              TRANS$LIST$TAIL=LINK$FIELD(BUFFER$POINTER);
169  3              END;
                  ELSE /* NOT AT FRONT OF LIST SO DELINK */
170  2              LINK$FIELD(PREV$BUF$POINTER)=LINK$FIELD(BUFFER$POINTER);
                  /*END IF*/

171  2              ENABLE;
172  2              RETURN;
173  2          END REMOVEFROMTRANS;


                  /*  P A R I T Y   R O U T I N E S  */

174  1          CALC$OF$HEADER$PARITY:PROCEDURE (BUFF$PTR);
                  /* THE HEADER HAS 16 BITS FOR PARITY WHILE IT USES FEWER
                    THAN 16 BITS SO A SIMPLE DUPLICATION OF THE HEADER BY
                    THE PARITY BITS IS USED FOR ERROR CHECKING */

175  2              DEC (I,BUFF$PTR) BYTE;

176  2              DO I = 0 TO HEADER$DATA$BYTE$LENGTH-1; /* FOR EACH HEADER BYTE */
177  3                  BUFFER$AREA(BUFF$PTR*BUFFER$SIZE+HEADER$DATA$BYTE$LENGTH+1)
                        = BUFFER$AREA(BUFF$PTR*BUFFER$SIZE+I);
178  3              END;
179  2              RETURN;
180  2          END CALC$OF$HEADER$PARITY;

181  1          DETERMINE$TEXT$PARITY: PROCEDURE (BUFF$PTR) ADDRESS;
                  /* THIS ROUTINE IS HARD CODED TO 40 CHARACERS DIVIDED INTO
                    10 FIELDS OF 4 BYTES, EACH FIELD'S PARITY INDICATED BY A BIT
                    IN THE PARITY FIELD.  */

182  2              DEC (BUFF$PTR,BIT$MASK,WHERE,SEGMENT$PARITY) BYTE;
183  2              DEC (PARITY$SHIFT$POSITION,
                        TEXT$BYTE$POSITION,
                        BIT$POSITION   ) BYTE;
184  2              DEC TEXT$PARITY ADDRESS;

185  2              TEXT$PARITY=0; /* CLEAR PARITY SETTINGS */

186  2              DO PARITY$SHIFT$POSITION = 0 TO 9; /* FOR EACH PARITY BIT */
187  3                  TEXT$PARITY=SHL(TEXT$PARITY,1); /* POSITION FOR NEXT SEGMENT */
188  3                  BIT$MASK=01H; /* START WITH LEAST SIGNIFICANT BIT */
189  3                  SEGMENT$PARITY=0;

190  3                  DO TEXT$BYTE$POSITION = 0 TO 3; /* FOR EACH OF 4 BYTES OF FIELD */
191  4                      WHERE=BUFF$PTR*BUFFER$SIZE+TEXT$BYTE$INDEX
                              +TEXT$BYTE$POSITION+(PARITY$SHIFT$POSITION*4);

192  4                      DO BIT$POSITION = 0 TO 7; /* FOR EACH BIT OF BYTE */
193  5                          SEGMENT$PARITY=SEGMENT$PARITY XOR
                                  (BIT$MASK AND BUFFER$AREA(WHERE));
194  5                          BIT$MASK=ROL(BIT$MASK,1); /* MOVE TO NEXT BIT */
195  5                      END; /* DO FOR EACH BIT */
196  4                  END; /* DO FOR EACH BYTE OF SEGMENT */
```

```
197   3             TEXT$PARITY=TEXT$PARITY OR SEGMENT$PARITY; /* PUT IN PARITY FOR
                                    SEGMENT */
198   3         END; /* DO FOR EACH SEGMENT */
      2         RETURN TEXT$PARITY;
      2       END DETERMINE$TEXT$PARITY;

201   1       CALC$OF$TEXT$PARITY: PROCEDURE (BUFF$PTR);
                /* CALCULATES PARITY AND STORES IN BUFFER */

202   2         DEC (BUFF$PTR,WHERE) BYTE ;
203   2         DEC CALC$PARITY ADDRESS;

204   2         CALC$PARITY=DETERMINE$TEXT$PARITY(BUFF$PTR);
205   2         WHERE=BUFF$PTR*BUFFER$SIZE+TEXT$PARITY$BYTE$INDEX;
206   2         BUFFER$AREA(WHERE)=HIGH(CALC$PARITY); /* PUT IN FIRST BYTE OF PARITY
                                    FIELD */
207   2         BUFFER$AREA(WHERE+1)=LOW(CALC$PARITY); /* PUT IN LOW BYTE */
208   2         RETURN;
209   2       END CALC$OF$TEXT$PARITY;

210   1       CHECK$OF$HEADER$PARITY: PROCEDURE (BUFF$PTR) BYTE;
                /* CHECKS THAT PARITY OF HEADER IS VALID AND RETURNS TRUE OR FALSE */

211   2         DEC (WHERE,BUFF$PTR) BYTE;

212   2         WHERE=BUFF$PTR*BUFFER$SIZE+ID$INDEX;
213   2         RETURN (BUFFER$AREA(WHERE)=BUFFER$AREA(WHERE+HEADER$DATA$BYTE$LENGTH))
                    AND (BUFFER$AREA(WHERE+1)=BUFFER$AREA(WHERE+HEADER$DATA$BYTE$LENGTH+1));
                /* SIMPLY ENSURE TWO FIELDS MATCH */
214   2       END CHECK$OF$HEADER$PARITY;


215   1       CHECK$OF$TEXT$PARITY: PROCEDURE (BUFF$PTR) BYTE;
                /* CHECKS TEXT PARITY AND RETURNS WHETHER VALID. ALSO GENERATES
                   TEXT$PARITY$MATCH WHICH IS A LOGICAL VECTOR INDICATING THE
                   CORRECT FIELDS */

216   2         DEC (BUFF$PTR,WHERE) BYTE;
217   2         DEC CALC$PARITY ADDRESS;

218   2         CALC$PARITY=DETERMINE$TEXT$PARITY(BUFF$PTR);

219   2         WHERE=BUFF$PTR*BUFFER$SIZE+TEXT$PARITY$BYTE$INDEX;

                /* GET PARITY FIELD INTO LOGICAL MATCH VECTOR */
220   2         TEXT$PARITY$MATCH=SHL(DOUBLE(BUFFER$AREA(WHERE)),8) OR
                        DOUBLE(BUFFER$AREA(WHERE+1));

                /* DETERMINE THE MATCHING PARITY */
221   2         TEXT$PARITY$MATCH=NOT(TEXT$PARITY$MATCH XOR CALC$PARITY);

                /* RETURN WHETHER ALL BITS MATCHED */
      2         RETURN TEXT$PARITY$MATCH=0FFFFH;
223   2       END CHECK$OF$TEXT$PARITY;
```

```
                /* NOTE: THE LOWER HALF BYTE OF THE ID COULD BE USED
                  AS THE DIRECT POINTER FOR THE MESSAGE BUT TO KEEP
                  THE SEARCH GENERAL AS THE ID MAY CHANGE IN FORMAT
                  THE LIST IS SEARCHED */

            /*  R E C E I V E R   I N T E R R U P T   R O U T I N E S  */

224   1     SEARCHFORID:    PROCEDURE (TARGET$PTR) ADDRESS;
                    /* SEARCH FOR A TEXT PACKET WITH THE SAME ID IN THE
                      TRANSMISSION LIST.  IF FOUND SEND BACK THE TWO
                      POINTERS THAT IDENTIFY IT IN THE LIST, PACKED IN 16 BITS.
                      IF NOT FOUND THEN THE PREVIOUS POINTER AND THE BUFFER
                      POINTER WILL BOTH BE NULL.  */

225   2           DEC (TARGET$PTR,PREV$PTR,BUF$PTR,J,FOUND) BYTE;
226   2           DEC PTRS ADDRESS;

227   2           PREV$PTR=NULL;
228   2           BUF$PTR=TRANS$LIST$HEADER;
229   2           FOUND=CLEAR;

230   2           DO WHILE (BUF$PTR <> NULL) AND (FOUND = CLEAR); /* FOLLOW ALONG
                    TRANSMISSION LIST LOOKING UNTIL ID MATCHED OR END OF LIST */
231   3              FOUND=SET;
232   3              DO J=ID$INDEX TO ID$BYTE$LENGTH - 1;  /*SEARCH EACH BYTE OF HEADER */
233   4                 IF BUFFER$AREA(BUFFER$SIZE * BUF$PTR + J) <>
                         BUFFER$AREA(BUFFER$SIZE * TARGET$PTR +J)
                          THEN DO;
      5                      FOUND=CLEAR; /* THE TWO DO NOT MATCH */
      5                      J=ID$BYTE$LENGTH-1; /* GET OUT OF LOOP */
237   5                   END;
                       /*END IF*/
238   4              END; /*DO LOOP*/

239   3              IF FOUND = CLEAR THEN /* ID NOT MATCHED SO FAR */
240   3                 DO;
241   4                 PREV$PTR=BUF$PTR; /* MOVE ALONG LIST */
242   4                 BUF$PTR=LINK$FIELD(PREV$PTR);
243   4                 END;
                       /*END IF*/
244   3           END; /* WHILE */

                    /**** MUST PASS BACK TWO BYTE POINTER VALUES AS ONE ADDRESS VALUE ***/
245   2           PTRS=DOUBLE(BUF$PTR)+SHL(DOUBLE(PREV$PTR),8);

246   2           RETURN PTRS;
247   2        END SEARCHFORID;

248   1     ACK$RECEIVED:   PROCEDURE;
                    /* THE RECEIVED DATA HAS BEEN DETERMINED TO BE AN ACK
                      SO MUST PERFORM THE FOLLOWING */
249   2           DEC (PREV$PTR,MATCHED$PTR,RECEIVED$ACK$POINTER,PARITY$FLAG) BYTE;
      2           DEC TWO$PTRS ADDRESS;
      2           DEC TEMP$PTR ADDRESS;

            /*DEBUGCALL PRINT(8,.('ACKRECVD'));*/
252   2           RECEIVED$ACK$POINTER=RECEIVER$BUFFER$POINTER;
```

```
                    /***** CHECK HEADER PARITY *****/
253   2             HEADER$PARITY$VALID=CHECK$OF$HEADER$PARITY(RECEIVED$ACK$POINTER);


254   2             IF HEADER$PARITY$VALID THEN
255   2                DO;
256   3                TWO$PTRS=SEARCHFORID(RECEIVED$ACK$POINTER); /* TWO POINTER VALUES PASSED AS AN ADDRESS*/
257   3                PREV$PTR=HIGH(TWO$PTRS);
258   3                MATCHED$PTR=LOW(TWO$PTRS);

259   3                IF MATCHED$PTR <> NULL THEN /* A MATCHED ID */
260   3                   DO;
261   4                   CALL PRINT(21,.('MESSAGE ACKNOWLEDGED:'));
262   4                   TEMP$PTR=.BUFFER$AREA(BUFFER$SIZE*MATCHED$PTR);
263   4                   CALL PRINT(BUFFER$SIZE,TEMP$PTR);
264   4                   CALL PRINT(2,.(LF,CR));
265   4                   CALL REMOVEFROMTRANS(PREV$PTR,MATCHED$PTR);
266   4                   CALL PUTATFRONTOFFREE(MATCHED$PTR);
267   4                   CALL PUTATFRONTOFFREE(RECEIVED$ACK$POINTER);
268   4                   END;
                       ELSE /* NO ID MATCHED */
269   3                   DO;
270   4                   CALL PUTATFRONTOFFREE(RECEIVED$ACK$POINTER);
271   4                   CALL PRINT(21,.('***** NO ID MATCHED',LF,CR));
                          /***** ERROR  NO ID MATCHED *****/
272   4                   END;
                       /*END IF*/
                    END; /* END OF TRUE PART OF IF PARITY VALID */
                 ELSE /* PARITY NOT VALID */
274   2                DO;
275   3                CALL PUTATFRONTOFFREE(RECEIVED$ACK$POINTER);

276   3                CALL PRINT(26,.('***** INVALID ACK PARITY',LF,CR));
                       /***** ERROR IN PARITY OF ACK *****/

277   3                END;
                    /*END IF*/
278   2             RECEIVED$BYTE$COUNT=0;

279   2             RETURN; /* TO DATA RECEIVED */
280   2          END ACK$RECEIVED;


281   1          FORMAT$ACK: PROCEDURE (BUF$PTR);
282   2             DECLARE BUF$PTR BYTE;
283   2             BUFFER$AREA(BUF$PTR*BUFFER$SIZE+ACK$BYTE$INDEX-1)
                    =BUFFER$AREA(BUF$PTR*BUFFER$SIZE+ACK$BYTE$INDEX-1)
                    OR ACK$MASK;

284   2             CALL CALC$OF$HEADER$PARITY(BUF$PTR);
285   2             TRIES$COUNT(BUF$PTR)=1;
286   2             RETURN;
287   2          END FORMAT$ACK;


288   1          PACKET$RECEIVED:PROCEDURE;
```

```
                    /* RECEIVED DATA PACKET DETERMINED TO BE TEXT */
289   2             DECLARE (TEMP$PTR,I,TWO$SEARCH$PTRS,RECEIVED$PACKET$POINTER) ADDRESS;

                  /*DEBUGCALL PRINT(9,.('PACKRECVD'));
                  TEMP$PTR=BUFFER$AREA(RECEIVER$BUFFER$POINTER*BUFFER$SIZE);
                  CALL HEXPRINT(TEMP$PTR);
                  CALL PRINT(2,.(LF,CR));
                  DEBUG*/

290   2             RECEIVED$PACKET$POINTER=RECEIVER$BUFFER$POINTER;
                    /***** CHECK HEADER PARITY *****/

291   2             HEADER$PARITY$VALID=CHECK$OF$HEADER$PARITY(RECEIVED$PACKET$POINTER);

292   2             IF HEADER$PARITY$VALID THEN
293   2                DO;
                       /***** CHECK WHETHER DUPLICATE PACKET *****/
294   3                TWO$SEARCH$PTRS=SEARCHFORID(RECEIVED$PACKET$POINTER);
295   3                IF LOW(TWO$SEARCH$PTRS) = NULL THEN /* NO DUPLICATES RECEIVED */
296   3                   DO;
                          /***** CHECK TEXT PARITY *****/
297   4                   TEXT$PARITY$VALID=CHECK$OF$TEXT$PARITY(RECEIVED$PACKET$POINTER);

298   4                   IF TEXT$PARITY$VALID THEN
299   4                      DO;
300   5                      CALL FORMAT$ACK(RECEIVED$PACKET$POINTER);
301   5                      CALL PUTATFRONTOFTRANS(RECEIVED$PACKET$POINTER);
302   5                      PACKET$TYPE(RECEIVED$PACKET$POINTER)=ACK$TYPE;
303   5                      END;
                          ELSE /* PACKET WAS INVALID SO DO NOT ACKNOWLEDGE */
304   4                      DO;
                             /***** ERROR INVALID TEXT PARITY *****/
305   5                      CALL PRINT(27,.('***** INVALID TEXT PARITY',LF,CR));
306   5                      DO I=0 TO BUFFER$SIZE-1;
307   6                         TEMP$PTR=BUFFER$AREA(RECEIVED$PACKET$POINTER
                                     *BUFFER$SIZE+1);
308   6                         CALL HEXPRINT(TEMP$PTR);
309   6                      END;


310   5                      CALL PUTATFRONTOFFREE(RECEIVED$PACKET$POINTER);
311   5                      END;
                          /*END IF*/
312   4                   END;
                       ELSE /* SAME AS LAST PACKET */
313   3                   DO;
314   4                   CALL PUTATFRONTOFFREE(RECEIVED$PACKET$POINTER);
                  /*DEBUG    CALL PRINT(4,.('SAME'));
                  DEBUG*/     /***** TRY ANOTHER ACK *****/
                  /*DEBUG    CALL FORMAT$ACK(RECEIVED$PACKET$POINTER);
                           CALL PUTATFRONTOFTRANS(RECEIVED$PACKET$POINTER);
                           PACKET$TYPE(RECEIVED$PACKET$POINTER)=ACK$TYPE;
                  DEBUG*/

315   4                   END;
                       /*END IF SAME AS LAST */
316   3                END;
```

```
                    ELSE /* INVALID HEADER PARITY */
317   2                 DO;
                        /***** ERROR HEADER PARITY INVALID *****/
      3                 CALL PRINT(28,.('***** INVALID$HEADER$PARITY',LF,CR));
                    /*DEBUG DO I=0 TO BUFFER$SIZE-1;
                            TEMP$PTR=BUFFER$AREA(RECEIVED$PACKET$POINTER*BUFFER$SIZE+I);
                            CALL HEXPRINT(TEMP$PTR);
                        END;
                    DEBUG*/

319   3                 CALL PRINT(2,.(CR,LF));


320   3                 CALL PUTATFRONTOFFREE(RECEIVED$PACKET$POINTER);
321   3                 END;
                    /*END IF*/

322   2             RETURN; /* TO RECEIVER INTERRUPT ROUTINE */
323   2         END PACKET$RECEIVED;

324   1         DATA$RECEIVED:PROCEDURE;
325   2             DEC STATUS BYTE;

                    /***** DO STATUS READ TO DETERMINE IF ERROR OCCURRED *****/
                    /*MDSPATCH*/
326   2             STATUS=INPUT(TRANSCEIVER$CONTROL);
327   2             IF /*RECEIVER ERROR*/ (STATUS AND ERROR$IN$RECEPTION)<>CLEAR THEN
      2                 DO;
      3                 CALL PRINT(35,.('***** RECEIVER ERROR',LF,CR,'STATUS VALUE '));
330   3                 CALL HEXPRINT(STATUS);
331   3                 CALL PRINT(17,.(' OCCURED ON BYTE '));
332   3                 CALL HEXPRINT(RECEIVED$BYTE$COUNT);
333   3                 CALL PRINT(2,.(LF,CR));
                        /***** ERROR IN RECEIVER... DISPLAY STATUS *****/
                        /***** PRINT BYTE COUNT *****/
334   3                 CALL PRINT(20,.('LAST CHAR IN BUFFER '));
335   3                 CALL HEXPRINT(RECEIVED$BYTE);
336   3                 CALL PRINT(2,.(LF,CR));

337   3                 IF RECEIVED$BYTE$COUNT > 0 THEN /* MUST RESET */
338   3                     DO;
339   4                     CALL PUTATFRONTOFFREE(RECEIVER$BUFFER$POINTER);
340   4                     RECEIVED$BYTE$COUNT=0;
341   4                     END;
                        /*END IF*/

                        /***** ENTER HUNT MODE (CLEARING ERROR FLAGS) *****/
342   3                 CALL ENTER$HUNT;
                        /*MDSPATCH*/
343   3                 OUTPUT(MDS$CONTROL)=0A4H;
344   3                 END;
                    ELSE /* NO ERROR IN RECEPTION */
      2                 IF RECEIVED$BYTE$COUNT=0 THEN
      2                     DO;
347   3                     RECEIVER$BUFFER$POINTER=GETFROMFREE;
```

```
                    /***** ASSIGN RECEIVED DATA *****/
381   2             RECEIVED$BYTE=INPUT(TRANSCEIVER$DATA);

                  /*DEBUGCALL HEXPRINT(RECEIVED$BYTE);*/
                    /*DEBUG PATCHED SHOULD USE SYNDET */

                    /***** CHECK WHETHER SYNCHRONIZATION SEQUENCE *****/
382   2             IF RECEIVED$BYTE = SYNCH$BYTE THEN /* SEE IF RESET NECESSARY */
383   2                 DO;
384   3                 IF RECEIVED$BYTE$COUNT <> 0 THEN /* RESET TO NEW PACKET */
385   3                     DO;
386   4                     CALL PUTATFRONTOFFREE(RECEIVER$BUFFER$POINTER);
387   4                     RECEIVED$BYTE$COUNT=0;
388   4                     END;
                        ELSE /* ENSURE ERRORS CLEARED */
389   3                     OUTPUT(TRANSCEIVER$CONTROL)=CLR$ERRS OR RXEN OR DTR;
                        /* END IF */
390   3                 END;
                    ELSE /* DATA COMING IN */
391   2                 CALL DATA$RECEIVED;
                    /* END IF */

                    /** RETURN IMPLICITLY ENABLES FOR INTERRUPT ROUTINES **/
392   2             RETURN;
393   2           END RECEIVER$INTERRUPT;


                  /*  T R A N S M I T T E R   R O U T I N E S  */

394   1           FORMAT$ID:PROCEDURE (BUFF$PTR);
                        /* FORMAT THE HEADER OF THE PACKET TO BE SENT */
                        /* EACH COLUMN REPRESENTS ONE BIT FOR TOTAL OF 32 BITS */
                  /*   ---------------------------------
                      \       AA                      \
                      \<==ID==>CL*SPARE<====PARITY====>\
                      \       KT                      \
                       ---------------------------------
                  */
395   2             DEC BUFF$PTR BYTE;

396   2             BUFFER$AREA(BUFF$PTR*BUFFER$SIZE+ID$INDEX)=
                        (SHL(TRANSMITTER$CODE,4))
                        OR(00001111B AND BUFF$PTR);
                    /* ID MADE UP OF TOP 4 BITS TRANSMITTER CODE AND
                      BOTTON 4 THE LEAST SIGNIFICAT 4 BITS OF MESSAGE BUFFER
                      POINTER */

397   2             BUFFER$AREA(BUFF$PTR*BUFFER$SIZE+ID$INDEX+1)=0;
                    /* CLEAR OUT ACK,ALT AND SPARE BITS */

398   2             CALL CALC$OF$HEADER$PARITY(BUFF$PTR);

      2             RETURN;
      2           END FORMAT$ID;

401   1           TRANSMIT$COMMAND:PROCEDURE;
                        /* TAKES LAST LINE OF ENTERED TEXT AND PUTS INTO TRANSMISSION LIST */
```

```
402  2          DEC I BYTE;

                /*DEBUGCALL PRINT(5,.('TRCOM'));*/
     2          CALL PUTATENDOFTRANS(TEXT$BUFFER$POINTER);
                /** BLANK OUT REST OF TEXT FIELD **/
404  2          DO I=TEXT$INPUT$COUNT+TEXT$BYTE$INDEX TO TEXT$PARITY$BYTE$INDEX-1;
405  3              BUFFER$AREA(TEXT$BUFFER$POINTER*BUFFER$SIZE+I)=20H;
406  3          END;

407  2          CALL FORMAT$ID(TEXT$BUFFER$POINTER);
408  2          CALL CALC$OF$TEXT$PARITY(TEXT$BUFFER$POINTER);
409  2          PACKET$TYPE(TEXT$BUFFER$POINTER)=TEXT$TYPE;
410  2          TRIES$COUNT(TEXT$BUFFER$POINTER)=PACKET$REPEAT$COUNT;
411  2          TEXT$BUFFER$POINTER=GETFROMFREE;
412  2          TEXT$INPUT$COUNT=0;
413  2          RETURN;
414  2      END TRANSMIT$COMMAND;

415  1      SEND$PACKET:    PROCEDURE;
                    /* INITIATES THE TRANSMISSION OF THE PACKET */
416  2          DEC (I,START,LAST) BYTE;
                /*DEBUGCALL PRINT(4,.('SEND'));*/
                /***** REQUEST TO SEND PACKET *****/
417  2          OUTPUT(TRANSCEIVER$CONTROL)=TXEN OR RTS OR DTR;
                /*MDSPATCH TO ENSURE MODEM CTS */
418  2          DO I=0 TO 5;
419  3              CALL TIME(250);
     3          END;
                /* MDSPATCH */

                /***** SEND OUT SYNC CHAR *****/
421  2          USART$STATUS=INPUT(TRANSCEIVER$CONTROL);

                /** WAIT FOR TXRDY **/
422  2          DO WHILE (USART$STATUS AND TRAN$READY)=CLEAR;
423  3              USART$STATUS=INPUT(TRANSCEIVER$CONTROL);
424  3          END;

425  2          OUTPUT(TRANSCEIVER$DATA)=SYNCH$BYTE;


426  2          START=TRANS$LIST$HEADER*BUFFER$SIZE; /* START OF PACKET */
427  2          LAST=START+PACKET$SIZE-1; /* LAST OF PACKET */


428  2          DO I=START TO LAST;
429  3              USART$STATUS=INPUT(TRANSCEIVER$CONTROL);
430  3              DO WHILE  (USART$STATUS AND TRAN$READY)=CLEAR;
                        /***WAIT FOR TRANSMITTER READY***/
431  4                  USART$STATUS=INPUT(TRANSCEIVER$CONTROL);
432  4              END;

                    /***** MOVE BYTE INTO BUFFER FOR TRANSMITTER *****/
     3              OUTPUT(TRANSCEIVER$DATA)=BUFFER$AREA(I);
434  3          END;
435  2          TRIES$COUNT(TRANS$LIST$HEADER)=TRIES$COUNT(TRANS$LIST$HEADER)-1;
```

```
                    /***** DISABLE TRANSMITTER *****/
436   2             CALL ENTER$HUNT;

      2             RETURN;
      2         END SEND$PACKET;

439   1         RANDOM$FUNCTION: PROCEDURE BYTE;
                    /* RETURNS A POSITIVE BYTE VALUE */

440   2             DEC VALUE BYTE;

441   2             VALUE=RANDOM$SEED*16807;
442   2             IF SIGN THEN VALUE = -VALUE;
444   2             RANDOM$SEED=VALUE;
445   2             RETURN VALUE;
446   2         END RANDOM$FUNCTION;

447   1         RANDOM$DELAY:   PROCEDURE;
                    /*****DELAY A RANDOM TIME *****/
448   2             DEC (I,LOOP$COUNT) BYTE;

449   2             LOOP$COUNT=20+10*RANDOM$FUNCTION; /* MUST ENSURE AT LEAST SOME TIME FOR ACK BEFORE RETRY */

450   2             DO I=0 TO LOOP$COUNT;
451   3                 CALL TIME(250);
452   3             END;
453   2             RETURN;
      2         END RANDOM$DELAY;

455   1         TRANSMIT$TABLE: PROCEDURE;
                        /* CONTROLS TRANSMISSION OF ACKS AND PACKETS */
456   2             DEC (I,BUF$PTR) BYTE;
457   2             DEC TEMP$BUF ADDRESS;

458   2             IF PACKET$TYPE(TRANS$LIST$HEADER) = ACK$TYPE THEN
459   2                 DO;
460   3                 PACKET$SIZE=HEADER$LENGTH; /* JUST SEND ACK*/
461   3                 CALL SEND$PACKET;
                        /***** LIST THE MESSAGE RECEIVED *****/
462   3                 CALL PRINT(8,.('MESSAGE:'));
463   3                 TEMP$BUF=.BUFFER$AREA(TRANS$LIST$HEADER*BUFFER$SIZE);
464   3                 CALL PRINT(BUFFER$SIZE,.BUFFER$AREA(TRANS$LIST$HEADER*BUFFER$SIZE));
465   3                 CALL PRINT(2,.(LF,CR));
                        /* NOTE THAT THE ACK IS SIMPLY THE HEADER OF THE RECEIVED
                          MESSAGE WITH A BIT SET */

466   3                 I=NULL; /* INDICATE FIRST OF TRANSMISSION LIST */
467   3                 BUF$PTR=TRANS$LIST$HEADER;
468   3                 CALL REMOVEFROMTRANS(I,BUF$PTR);
469   3                 CALL PUTATFRONTOFFREE(BUF$PTR);
470   3                 END;
                    ELSE /* PACKET TEXT TYPE */
      2                 IF TRIES$COUNT(TRANS$LIST$HEADER) = 0 THEN /* ENOUGH TRIES */
      2                     DO;
                            /***** ERROR UNSUCCESSFUL TRANSMISSION *****/
473   3                     CALL PRINT(21,.(CR,LF,'UNSUCCESSFUL TRANS:'));
474   3                     TEMP$BUF=.BUFFER$AREA(TRANS$LIST$HEADER*BUFFER$SIZE);
```

```
475  3                  CALL PRINT(BUFFER$SIZE,TEMP$BUF);
476  3                  CALL PRINT(2,.(CR,LF));
477  3                  I=NULL;
     3                  BUF$PTR=TRANS$LIST$HEADER; /* INDICATE REMOVAL OF TOP OF LIST */
     3                  CALL REMOVEFROMTRANS(I,BUF$PTR);
480  3                  CALL PUTATFRONTOFFREE(BUF$PTR);
481  3                  END;
                     ELSE /* MORE RETRANSMISSIONS ALLOWED */
482  2                   DO;
483  3                   PACKET$SIZE=BUFFER$SIZE; /* SEND FULL PACKET */
484  3                   CALL SEND$PACKET;
485  3                   END;
                     /* END IF */
486  2                RETURN;
487  2           END TRANSMIT$TABLE;
                 /*  T E R M I N A L   I N T E R R U P T   R O U T I N E  */

                 /*MDSPATCH*/
                 /* SHOULD BE INTERRUPT 4*/
488  1           TERMINAL$INPUT: PROCEDURE ;
                    /* HANDLES EDITING OF TEXT INPUT FOR MESSAGES
                       - CARRIAGE RETURN ACTS AS TRANSMIT COMMAND
                       - RUBOUT REMOVES PREVIOUS (IF ANY) CHARACTER FROM LINE
                       - ESC THEN X CLEARS LINE OF TEXT
                 */

489  2              DEC (CHAR,TERMINAL$STATUS) BYTE;

     2              CHAR=INPUT(TERMINAL$DATA);

491  2              TERMINAL$STATUS=INPUT(TERMINAL$CONTROL); /*CHECK FOR ERROR*/
492  2              IF (TERMINAL$STATUS AND ERROR$IN$RECEPTION)=CLEAR THEN
493  2              DO;
494  3              CHAR=01111111B AND CHAR;    /* STRIP OFF PARITY */

495  3              IF CHAR=CR THEN
496  3                 DO;
497  4                 CALL TRANSMIT$COMMAND;
498  4                 CALL PRINT(2,.(LF,CR));
499  4                 CONTROL$COMMAND$FLAG=CLEAR;
500  4                 END;
                   ELSE /* NOT A CR */
501  3                 IF CHAR=RUBOUT THEN
502  3                    DO;
503  4                    IF TEXT$INPUT$COUNT<>0 THEN
504  4                       DO; /* ECHO LAST CHARACTER OF LINE */
505  5                       TEXT$INPUT$COUNT=TEXT$INPUT$COUNT-1;
506  5                       CHAR=BUFFER$AREA(TEXT$BUFFER$POINTER*BUFFER$SIZE
                                    +TEXT$BYTE$INDEX+TEXT$INPUT$COUNT);
507  5                       CALL PRINT(1,.CHAR);
508  5                       END;
                         /* END IF TEXT <> 0 */
     4                   CONTROL$COMMAND$FLAG=CLEAR;
     4                   END;
                      ELSE /* NOT RUBOUT */
511  3                    IF CHAR=ESC THEN
512  3                       CONTROL$COMMAND$FLAG=SET;
```

```
                    ELSE /* NOT ESC */
513  3                  IF (CHAR=XCHAR) AND (CONTROL$COMMAND$FLAG=SET) THEN
514  3                      DO;
     4                      TEXT$INPUT$COUNT=0;
     4                      CALL PRINT(2,.(LF,CR));
     4                      CONTROL$COMMAND$FLAG=CLEAR;
518  4                      END;
                        ELSE /* NORMAL TEXT ENTRY */
519  3                      DO;
520  4                      IF TEXT$INPUT$COUNT >=
                                (TEXT$PARITY$BYTE$INDEX-TEXT$BYTE$INDEX) THEN
                                /*TOO MANY CHARACTERS FOR ONE PACKET */
521  4                          DO; /*IMPLICIT CR */
522  5                          CALL TRANSMIT$COMMAND;
523  5                          CALL PRINT(2,.(LF,CR));
524  5                          END;
                            /* END IF TOO MANY CHARS */

525  4                      BUFFER$AREA(TEXT$BUFFER$POINTER*BUFFER$SIZE
                              +TEXT$BYTE$INDEX+TEXT$INPUT$COUNT)=CHAR;
526  4                      TEXT$INPUT$COUNT=TEXT$INPUT$COUNT+1;
527  4                      CALL PRINT(1,.CHAR);
528  4                      CONTROL$COMMAND$FLAG=CLEAR;
529  4                      END;
                        /* END IF CHAR=XCHAR */
                    /* END IF CHAR=ESC */
                /* END IF CHAR=RUBOUT */
            /* END IF CHAR=CR */
     3      END;
            ELSE /* CLEAR ERROR IN RECEPTION FOR NEXT INPUT */
531  2          OUTPUT(TERMINAL$CONTROL)=CLR$ERRS OR TXEN OR RTS OR RXEN OR DTR;
            /* END IF ERROR IN RECEPTION */
            /*** RETURN IMPLICITLY ENABLES IN INTERRUPT ROUTINES ***/
532  2          RETURN;
533  2      END TERMINAL$INPUT;


        /*MDSPATCH*/
534  1      CONTROL:PROCEDURE INTERRUPT 3;
535  2          DEC STATUS BYTE;
536  2          STATUS=INPUT(MDS$STATUS);
537  2              IF (STATUS AND 20H) <> CLEAR THEN
538  2                  DO;
539  3                  CALL RECEIVER$INTERRUPT;
540  3                  OUTPUT(MDS$CONTROL)=20H; /*CLEAR MDS FLIP-FLOP*/
541  3                  END;
                    ELSE
542  2                  IF (STATUS AND 2H)<> CLEAR THEN
543  2                      DO;
544  3                      CALL TERMINAL$INPUT;
545  3                      OUTPUT(MDS$CONTROL)=2H; /*CLEAR MDS FLIP-FLOP*/
546  3                      END;
                        /*END IF*/
                    /*END IF*/

                /***** RESET INTERRUPT ENVIRONMENT *****/
                OUTPUT(INTEL$8259$CONTROL)=INTEL$8259$RESET;
```

```
548  2          GARBAGE$READ=INPUT(TERMINAL$DATA); /* CLEAR RXRDY */
549  2          OUTPUT(243)=0FFH;
550  2          RETURN;
[illegible]  2      END CONTROL;

           /******   M A I N   S Y S T E M   ******/

           /* INITIALIZATION TABLE */
552  1     DISABLE;
553  1     EDIT$FLAG=0;
554  1     RECEIVED$BYTE$COUNT=0; /* SET NUMBER OF DATA CHARACTERS RECEIVED */
555  1     TRANS$LIST$HEADER=NULL;
556  1     TRANS$LIST$TAIL=NULL;
557  1     FREE$LIST$HEADER=0;
558  1     DO I=0 TO NUMBER$BUFFERS-2;
559  2         LINK$FIELD(I)=I+1;
560  2     END;
561  1     LINK$FIELD(NUMBER$BUFFERS-1)=NULL;
           /***** INITIALIZATION OF TERMINAL *****/
           /*NDSPATCH*/
562  1     OUTPUT(TERMINAL$CONTROL)=RESET;
563  1     OUTPUT(TERMINAL$CONTROL)=ST1 OR R110 OR CL8;
564  1     OUTPUT(TERMINAL$CONTROL)=CLR$ERRS OR TXEN OR RTS OR RXEN OR DTR;


           /***** INITIALIZATION OF MODEM CONNECTION *****/
565  1     OUTPUT(TRANSCEIVER$CONTROL)=RESET;
           /*ASYNC DEBUG
           OUTPUT(TRANSCEIVER$CONTROL)=SINGLE$SYNC OR SYNDET$OUTPUT OR CL8;
           OUTPUT(TRANSCEIVER$CONTROL)=SYNCH$BYTE;
           DEBUG*/
566  1     /*DEBUG*/OUTPUT(TRANSCEIVER$CONTROL)=ST2 OR R24AT1 OR CL8; /* ASYNC SETTING */
567  1     CALL ENTER$HUNT;
568  1     /*DEBUG*/RANDOM$SEED=GARBAGE$READ; /* ATTEMPT TO GET RANDOM SEED FOR EACH TERMINAL FROM GARBAGE RECEIVED *
           - /
569  1     CALL PRINT(5,.('SEED='));
570  1     CALL HEXPRINT(RANDOM$SEED);
571  1     CALL PRINT(2,.(LF,CR));
           /*DEBUG END*/

           /***** CHECK FOR MODEM RESPONSE TO DTR *****/
572  1     USART$STATUS=INPUT(TRANSCEIVER$CONTROL);
573  1     TEST$TRIES=0;

574  1     DO WHILE (USART$STATUS AND DSR$BIT)=CLEAR;
575  2         USART$STATUS=INPUT(TRANSCEIVER$CONTROL);
576  2         TEST$TRIES=TEST$TRIES+1;
577  2         IF TEST$TRIES >= MAX$WAIT$FOR$FLAG THEN
578  2             DO;
579  3             CALL PRINT(11,.('DSR NOT SET'));
580  3             HALT;
581  3             END;
[illegible]  2     END;



583  1     TEXT$BUFFER$POINTER=GETFROMFREE; /*GET A BUFFER FOR THE TEXT */
```

```
584  1      TEXT$INPUT$COUNT=0; /* CLEAR COUNT */

            /*MDSPATCH*/
585  1      OUTPUT(MDS$CONTROL)=0FFH;

            /***** INITIALIZE INTEL$8259 TO NESTED INTERRUPT MODE *****/
586  1      OUTPUT(INTEL$8259$CONTROL)=012H;
587  1      OUTPUT(INTEL$8259$MASK)=0E0H; /* ALLOW INTERRUPTS 0 THRU 4 */
588  1      ENABLE;
            /* LOOP WAITING FOR RECEIVER INTERRUPT OR A MESSAGE TO SEND */
589  1      DO FOREVER; /* END BY SYSTEM SHUT DOWN */
590  2          IF TRANS$LIST$HEADER <> NULL THEN
591  2              DO;
            /*DEBUGCALL PRINT(7,.('TRANTAB'));*/
592  3              DISABLE;
593  3              CALL TRANSMIT$TABLE;
594  3              ENABLE;
595  3              CALL RANDOM$DELAY; /* GIVE ACKNOWLEDGEMENT A CHANCE BEFORE RETRY */
596  3              END;
                /*END IF*/
                /*MDSPATCH TO ATTEMPT TO ENSURE THAT RXRDY FOR EITHER USART
                GENERATES INTERRUPT RESPONSE. MDS STRUCTURE FOR INTERRUPTS
                INTERFERES WITH THIS AS FLAGS USED ARE NOT THE RXRDY FLAGS
                DIRECTLY BUT THEY GO THROUGH THE SAME INT3 USING FLIP-FLOPS.
                 */

597  2          OUTPUT(MDS$CONTROL)=0;
598  2          USART$STATUS=INPUT(TRANSCEIVER$CONTROL);
     2          IF (USART$STATUS AND CHAR$READY)<>CLEAR THEN
     2              CALL RECEIVER$INTERRUPT;
                /*END IF */

601  2          USART$STATUS=INPUT(TERMINAL$CONTROL);
602  2          IF (USART$STATUS AND CHAR$READY)<>CLEAR THEN
603  2              CALL TERMINAL$INPUT;
                /*END IF*/
604  2          OUTPUT(MDS$CONTROL)=0FFH;
                /*MDSPATCHEND*/
605  2      END;
606  1      END PACKET$SYSTEM;
```

MODULE INFORMATION:

```
    CODE AREA SIZE     = 0D5BH   3419D
    VARIABLE AREA SIZE = 0224H    548D
    MAXIMUM STACK SIZE = 0014H     20D
    1060 LINES READ
    0 PROGRAM ERROR(S)

    OF PL/M-80 COMPILATION
```