THE DESIGN AND APPLICATION OF A MICROPROCESSOR DEVELOPMENT SYSTEM

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Jerry Burns Pace

December 1978

ACKNOWLEDGEMENTS

I sincerely thank Dr. Olin G. Johnson, my thesis advisor, for his help and encouragement. Thanks also to Dr. Willis K. King and Dr. James D. Bargainer who served on my thesis committee.

A special thanks to Holly Frost, who designed the cards used in the system, for his aid in accquiring the parts for the system. Thanks to Buddy Peiser, a good friend, for his support and encouragement. And thanks to MOSTEK for donating the Z80 cross assembler which was especially appreciated.

Finally, I want to thank my wife, Jackie, who typed many pages, my son, Jack, and my daughter, Elise, for continuing support and patience through many years. They never doubted I would make it.

THE DESIGN AND APPLICATION OF A MICROPROCESSOR DEVELOPMENT SYSTEM

An Abstract

of a Thesis

Presented To

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Jerry Burns Pace

December 1978

ABSTRACT

The material presented in this Thesis concerns two topics: the first is the design of a Microprocessor Development System and the second is the application of this system for developing a rather extensive programming example.

The Microprocessor Development System was designed around a Z-80 microprocessor. The system contains 8K of RAM, 12K of ROM, 4 serial I/O ports and room for 3 additional cards. A 2K monitor was implemented in ROM and a cross assembler was set up on a large mainframe HOST system. An I/O routine was written to allow the microprocessor system to converse directly with the HOST system. Programs could then be developed on the HOST system, assembled with the cross assembler and loaded directly into the microprocessor for debugging.

The programming example discussed is a program to emulate a multi-terminal network processor, a device which is used to multiplex several terminals on a timesharing system via a single modem line. Excellent results were obtained when using the HOST/Microprocessor combination for developing and testing programs for the microprocessor system.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

INTRODUCTION

The recent availibility of low-cost microprocessors has opened the door for many new and useful applications. This thesis will discuss the design of one of these microprocessor systems; specifically, a Z-80 microprocessor and the application of this unit as a network processor.

The work reported here divides naturally into two parts: Part I was the development of the hardware. This included purchasing and assembling the microprocessor, modifying the software monitor so that the microprocessor could communicate with a large HOST computer (HIS 66/60) and installing a cross assembler on the HOST to assemble programs for the microprocessor.

Part II involved choosing a development example which would illustrate the capabilities of the development system. It was decided to write a program which would use the development system to develop an emulator for a Remote Network Processor (HONEYWELL RCP 707). The network processor was chosen to demonstrate the ability of the microprocessor to do complex jobs with relatively inexpensive hardware.

PART I

MICROPROCESSOR DEVELOPMENT SYSTEM

Chapter 1

HARDWARE

The microprocessor development system hardware is composed of two 7" by 9" printed circuit cards (the CPU card and the I/O card), a printed circuit CPU BUS Mother Board with provisions for 5 cards, and a multi-output power supply. Detailed wiring diagrams of the CPU cards can be found in Appendix A.

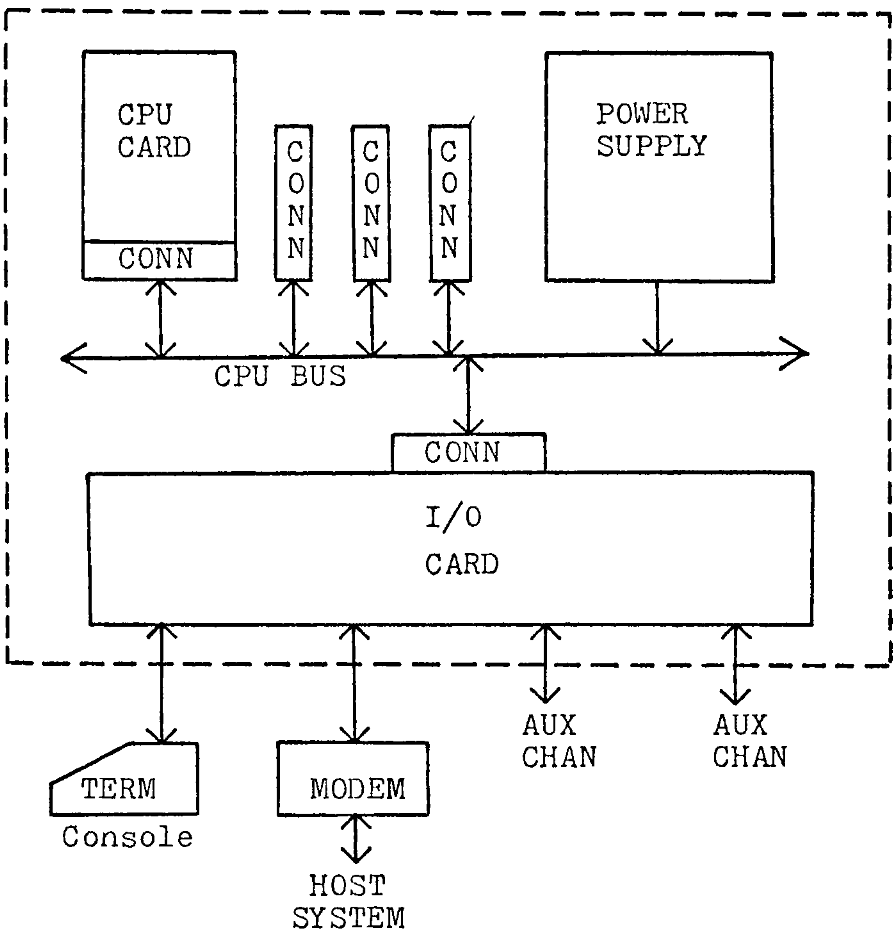


Fig. 1A SYSTEM BLOCK DIAGRAM

The <u>CPU card</u> contains the Z-80 microprocessor, a 2 MHZ crystal clock, 8K of dynamic RAM, and 4K of programmable ROM, along with all the decoders, drivers, and receivers necessary to handle the CPU bus.

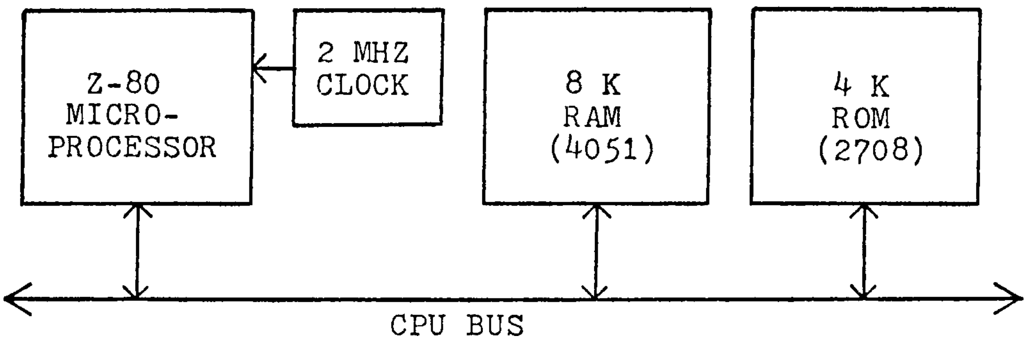


Fig. CPU CARD BLOCK DIAG.

The CPU chosen for this project was a Z-80 microprocessor. The reasons for choosing this particular unit were:

(1) It was one of the fastest and most powerful 8 bit microprocessors available.

(2) It was very easy to design a system around.

(3) Parts for this system were readily available and relatively inexpensive.

A crystal clock was used, instead of another type, due to its inherent stability and accuracy.

The 2708 programable ROMs used to store the programs, both on the CPU card and the I/O card, combined a large storage capacity in a relatively small space, and were also very cost effective. The 4051 4K dynamic RAMs were used because, at this time, 4K dynamic RAMs were the least expensive type; and since the Z-80 had a built-in refresh counter, no extra hardware was required for refresh circuity.

The <u>I/O card</u> holds all the I/O interfaces consisting of 4 programmable I/O controllers (3 asynchronous and 1 synchronous unit), a 4 channel programmable real time clock/timer unit, an 8 input interrupt request register, and an additional 8K of programmable ROM.

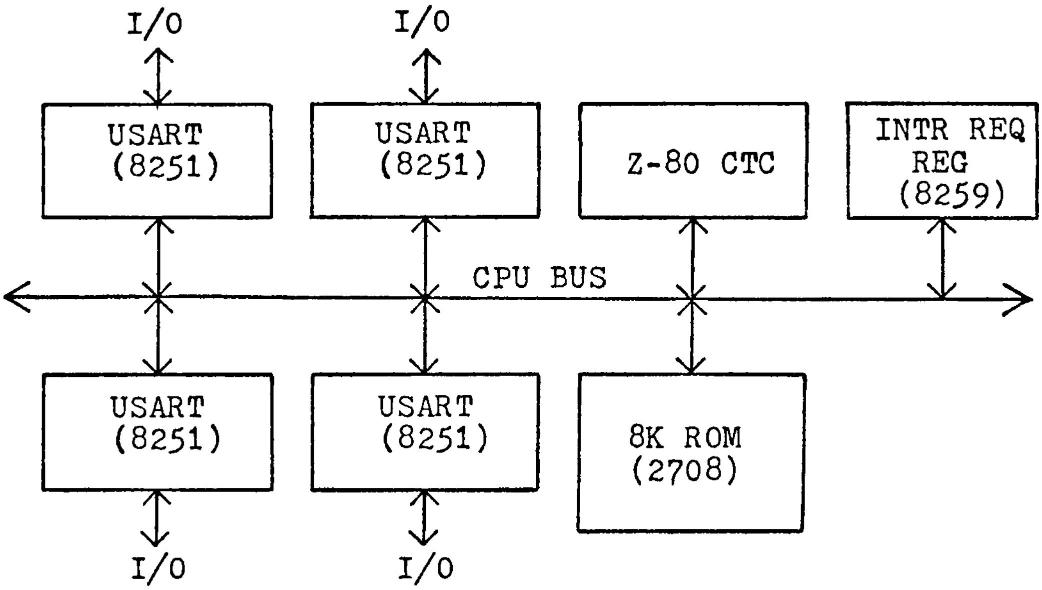


Fig. 1C I/O CARD BLOCK DIAG.

The programmable I/O controllers used on the I/O card were 8251 USARTs (UNIVERSAL SYNCHRONOUS/ASYNCHRONOUS RECEIVER/TRANSMITTER). These units were chosen for their ability to be programmed by the CPU, to operate in virtually any serial data transmission technique presently in use. They will operate in full duplex asynchronous mode to 9600 baud, and in full duplex synchronous mode to 50K baud. They also connect directly to the CPU bus and require no special interface circuitry.

The programmable clock/timer used on the I/O board was a Z80-CTC. It contains 4 independent programmable 8 bit counter/16 bit timer channels. Each channel can be programmed to operate either as a counter or a timer, which can generate interrupts and automatic interrupt vectoring with no external logic.

The 8259 interrupt controller is used here only as an interrupt request holding register. The software interrupt routine uses a polling technique to find the correct device to service.

The <u>CPU BUS Mother Board</u> serves simply as a mounting surface for five 100 pin card edge connectors which interconnect the signals and supply power to the cards. All CPU bus signals and all power lines are connected through this bus card.

The power supply is a three output regulated supply which produced 5 volts at 6 amps, +12 volts at 1.5 amps and -12 volts at 1.5 amps. Since a -5 volt supply was also required by the system, a -5 volt regulator driven by the -12 volt supply was used to supply -5 volts at 1 amp. The power supply delivers much more power than is required by the present system, which allows for the addition of other cards for future expansion of the basic system.

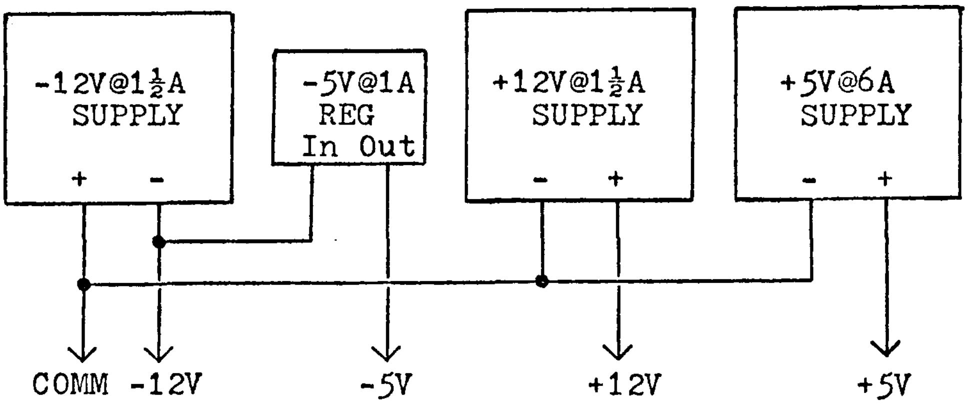


Fig. 1D POWER SUPPLY BLOCK DIAG.

Chapter 2

MONITOR PROGRAM

A system monitor program (ZAPPLE MONITOR by TDL), located on the CPU card in the upper 2K of ROM, provided all necessary functions for loading, displaying, modifing, and debugging assembly language programs. This monitor, however, had no provisions for connecting a HOST processor or for loading assembled code from a HOST processor. Therefore, an I/O routine had to be written to connect the Z-80 system to the HOST system (Appendix D).

The monitor contains, among others,routines for the following functions:

(1) assign alternate peripherial devices for I/O or console,

(2) display and/or change any single location in memory on the console,

(3) display blocks of memory on the console,

(4) fill blocks of memory with a single constant,

(5) display and/or change registers from the console, and

(6) set up one or two software break points.

Altogether, there are 23 separate functions in the standard monitor and provisions for 3 user defined functions. In addition, the monitor has many useful subroutines for I/O and data conversion which can be called by other programs. Appendix B gives a list of all the commands and a brief explaination of their use.

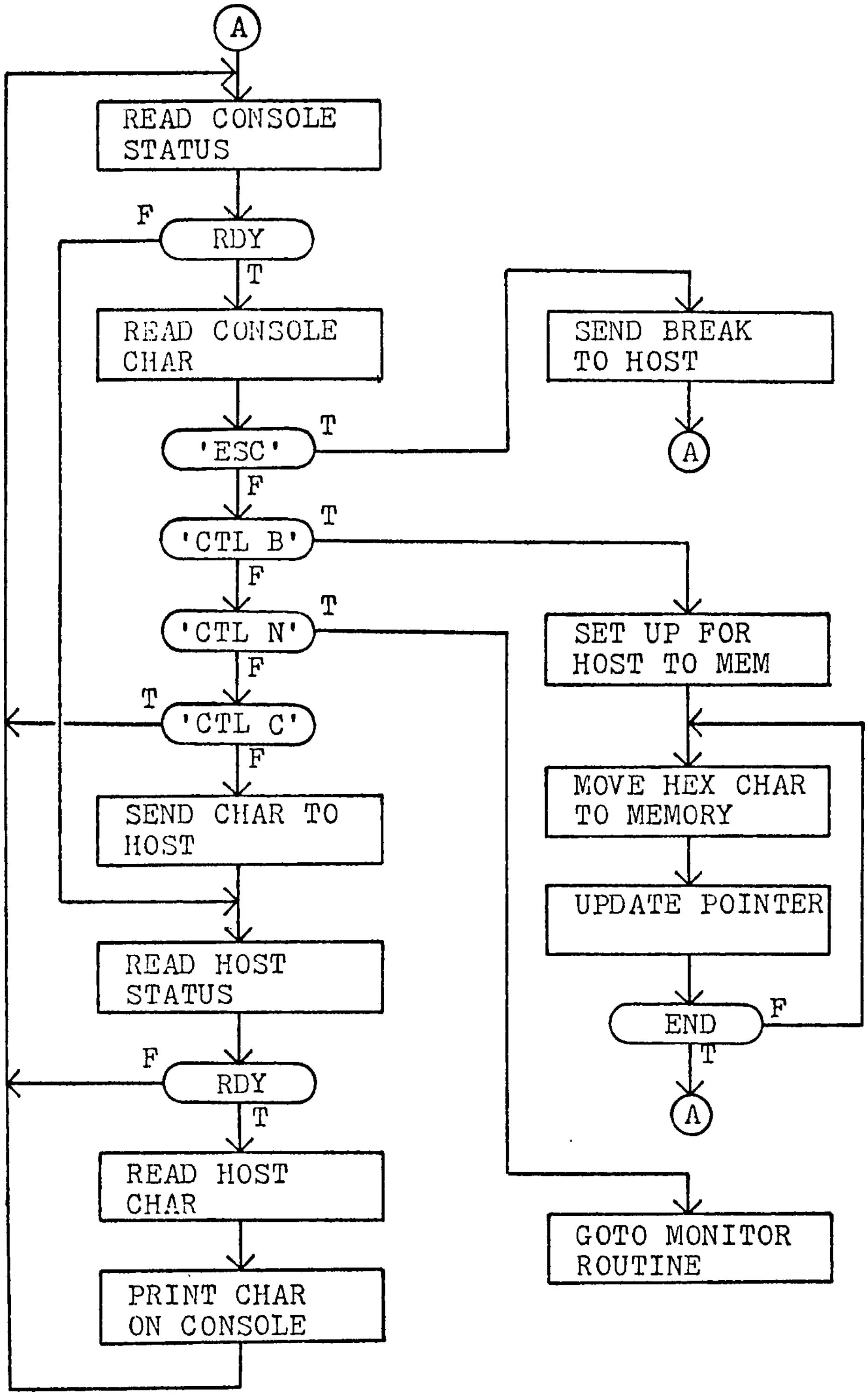


Fig. 2A Z-80 to HOST I/O Program

To make the task of conversing with the HOST processor as simple as possible, the Z-80 system was made to emulate a terminal and connected to the standard time sharing network (TSS) of the HOST processor. A special routine, which could be entered from this program, was written to load assembled code from the HOST processor to the memory of the Z-80 system. This made it possible to use the HOST system for writing, editing, assembling, and storing programs for the Z-80. The assembled code from these programs could then be down loaded to the Z-80 system for testing and debugging.

Figure 2A is a flow chart of the Z-80 to HOST I/O routine. The main loop of the program continously teststhe status of the console input and the modem input. When a character is ready to be read, the status flag will be set and the character will be read. If it is a console character, it will be tested and if it is also one of the command characters, a special routine will be entered to execute the command; otherwise, the character will be sent directly to the modem for transmission to the HOST. If, however, the character comes from the modem, it will immediately be printed on the console.

There are 4 command characters input from the console:

(1) An 'ESC' character is used instead of the conventional break key because the I/O channel cannot detect a break. This input causes the program to go to the BREAK routine, which sends a break to the HOST for 250 MS and then returns to the main loop.

(2) A 'CTL B' (CONTROL B) is used to enter the HOST to memory routine. This routine first asks for an offset value, next asks for the HOST file name, and then sends the command to the HOST to start input to memory. The input to this program, which must be a standard HEX FORMAT file — if not the programs aborts and returns to the main loop —, is then loaded and printed on the console at the same time. When the file is completely loaded, the routine returns to the main loop.

(3) A 'CTL C' is normally used to cause an immediate disconnect. However, this was considered an undesirable feature. Therefore, this character is ignored and not sent to the HOST.

(4) A 'CTL N' is used to cause a direct return to the monitor program.

Other than the above 4 characters, all keyboard characters are treated the same as in any standard TSS terminal and sent directly to the HOST.

Chapter 3

Z-80 CROSS ASSEMBLER

To allow Z-80 programs to be assembled on the HOST system, a cross assembler was acquired (XFOR-80 by MOSTEK). This cross assembler, although written in FORTRAN, was not written specifically for the HONEYWELL 66/60. Therefore, some slight modifications had to be made before it would work on this system. Some of the special characters had to be changed because they were not allowed on the TSS network and some special file instructions had to be added to the program.

The XFOR-80 is a 2 pass assembler which will assemble all standard Z80 source statement and also MACROS. As implemented on the HOST system (HIS 66/60), the input can be in the form of a TSS file, created on line, or a deck of punched cards or any other compatible file storage medium.

The output from the program is in the form of two separate disk files. One file is the line printer listing containing the assembled code along with the listing of the program instructions. It can be displayed on the console of the Z-80 system and/or printed on the line printer of the HOST system. The other file is the assembled code in standard HEX format which can be loaded into the Z-80 memory for execution or debugging.

PART II

DEVELOPMENTAL EXAMPLE

RNP EMULATION PROGRAM

CHAPTER 4

RNP OVERVIEW

The Remote Network Processor is a device used for combining several terminals and/or several remote computers and/or remote batch facilities, in such a way that they can communicate with a host processor on a single high speed modem line. There are basically two protocols: one called RMC (REMOTE MESSAGE CONCENTRATION) for remote terminals and remote computers, and the other called RBS (REMOTE BATCH SYSTEM) for remote batch stations. It was decided to only implement the first, RMC, because it was simpler and would still serve well as an example. The following is a brief explanation of the RMC protocol. A more detailed explanation is available in the HONEYWELL RNP/FNP INTERFACE manual, number DB92.

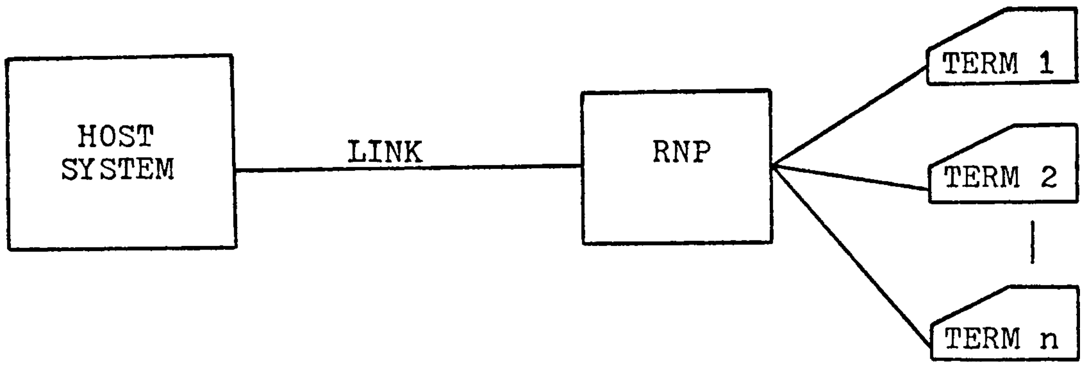


Fig. 4A RNP LOGICAL CONFIGURATION

A logical configuration of the RNP system is shown in Fig. 4A. The connection between the HOST and the RNP is called the link. This can be in the form of a modem or a direct wired connection.

The HOST/RNP link must be in one of the three following states:

(1) Physically disconnected.

(2) Logically disconnected (physically connected but idle).

(3) Logically connected (physically connected and active).

Control of the link is carried on through the exchange of Q-Frames during all periods in which the link is active and there is no link or logical messages to exchange. This exchange is always initiated by the HOST, therefore, avoiding contention of the line.

All communication between the HOST and the RNP related only to the HOST/RNP link is carried in the link message, and communication between the HOST and each individual terminal is carried in the logical message. These messages, over the link, are carried in TRANSMISSION BLOCKS (XMIT BLOCK), which consists of a link message as the first, or only message. They may also contain one or more logical messages, each of which contains a unique address identifying its destination. The entire block is terminated by the 'EOT' character.

Each XMIT BLOCK must be acknowledged (ACK) in the next received block or the same block is retransmitted (NAK). No new XMIT BLOCK (one having a new sequence code and different logical messages) may be sent until the previous one is acknowledged (ACK). The sequence code and the acknowledgement of the link message are used to insure against lost or duplicate XMIT BLOCKS. In addition, if no answer is received to a transmission within a specific amount of time, the same XMIT BLOCK is retransmitted.

There are two types of XMIT BLOCKS: the Service Message and the Data Message. The Service message differs from the Data message by the presents of only the link message and no logical messages in the XMIT BLOCK and the header of the link message contains a 102_8 instead of a 110_8 in the FC.

The Service message is used to control the link and conveys the following 4 messages:

(1) RFD - Tells the receiver that the sender is going to disconnect the link. Must be acknowledged with an RFD.

(2) DIS - Tells the receiver that the sender is disconnecting the link. No reply is necessary and both processors disconnect.

(3) A CALL - Sent by the HOST to tell the RNP to accept all incoming calls.

(4) N CALL - Sent by the HOST to tell RNP to accept no new incoming calls.

The following table illustrates the error recovery rules. In these rules, the sequence code (SC) in the link message refers to the code in the header which alternates between 101_8 and 102_8. A changed SC indicates a new XMIT BLOCK. The ACK or NAK refers to whether or not a message is received in error. All retransmissions repeat the full XMIT BLOCK.

ERROR RECOVERY RULES

RECEIVED BLOCK

Same SC	New SC	ACK	NAK	ERR	
				X	Transmit link message with NAK and same SC.
	X	X			Process Logical messages. Change SC and transmit next XMIT BLOCK with ACK.
	X		X		Process Logical messages. Change SC and transmit next XMIT BLOCK with ACK.
X		X			Disregard Logical messages. Change SC and transmit next XMIT BLOCK with ACK.
X			X		Disregard Logical messages. Retransmit last XMIT BLOCK with same SC and ACK.

The XMIT BLOCKS are made up of messages which are composed of strings of characters. All characters used on the HOST/RNP link must be ASCII 8 bit (7 data bits + parity) characters. The bit notation is shown below:

MSB LSB

P	6	5	4	3	2	1	0

DATA

A detailed description of the XMIT BLOCKS plus a description of the link and logical messages, along with a break down of their respective headers, is given in Appendix C.

Chapter 5

RNP EMULATION PROGRAM

The RNP Emulation Program has been written to simulate the actions of the HONEYWELL Remote Network Processor, configured to handle only the RMC protocol. It is written in modular form with individual subroutines for all major functions. This makes the program easily adaptable to many differing hardware configurations, without requiring major programming changes. Also, some of the functions are table driven to allow them to be changed or enlarged more easily. The program is also written such that it is 'ROMable' (i.e. written such that it can be stored in ROM and executed), therefore, no program variables are located within the program itself, but are all stored in RAM outside the program.

In addition to the normal RNP functions (HOST to terminal I/O), several additional functions were implemented. The program also allows device to device transmissions, without involving the HOST system; so this type of communication can go on even when the HOST is off line. Each device on the RNP has the ability to assign a destination device to itself with a keyboard command, and it can connect to or disconnect from the HOST with a similar command. This capability eliminates the need for a control console.

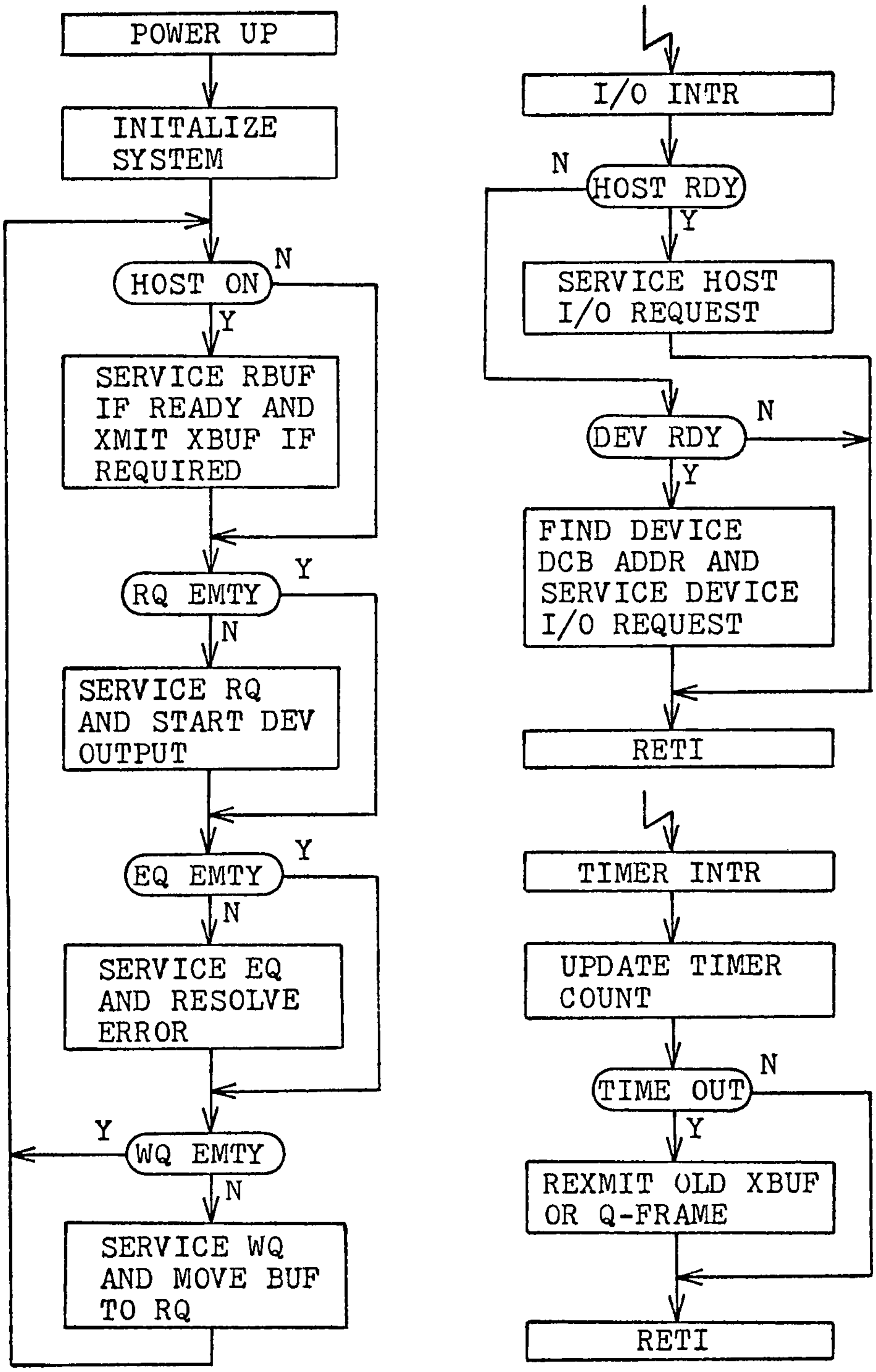


Fig. 5A RNP Basic Flow Diag.

The RNP program (Fig. 5A) accepts character input from terminals and/or remote computers (designated devices), and puts them into input buffers. Each completed buffer is then placed on a queue, to await output to the HOST or to another device. When the queue is services, each buffer is either output to another device or converted to a logical message. This logical message is then sent to the HOST in an XMIT BLOCK which normally contains logical messages from other devices.

Input from the HOST, in the form of an XMIT BLOCK, is converted from logical messages for several devices into individual output buffers, which are then placed on a queue. When this queue is services, the buffers are output to the respective devices by the I/O subroutines.

Device I/O buffers are composed of 64 byte buffer blocks. All available blocks are kept on the AQUE. When a block is needed by a process it is removed from the top of the AQUE, and when it is no longer needed it is put back on the bottom of the AQUE. The total number of blocks available is limited only by the amount of RAM available in the system.

All device I/O buffers are dynamic in size with a basic block size of 64 bytes, of which 58 are usable for character storage, 4 are used for the buffer header, and 2 are used for the linkage pointer. The basic buffer structure is shown in Figure 5B, along with the method of block linkage. Buffer size can vary from 1 to a maximum of 4 blocks for a total of 232 characters.

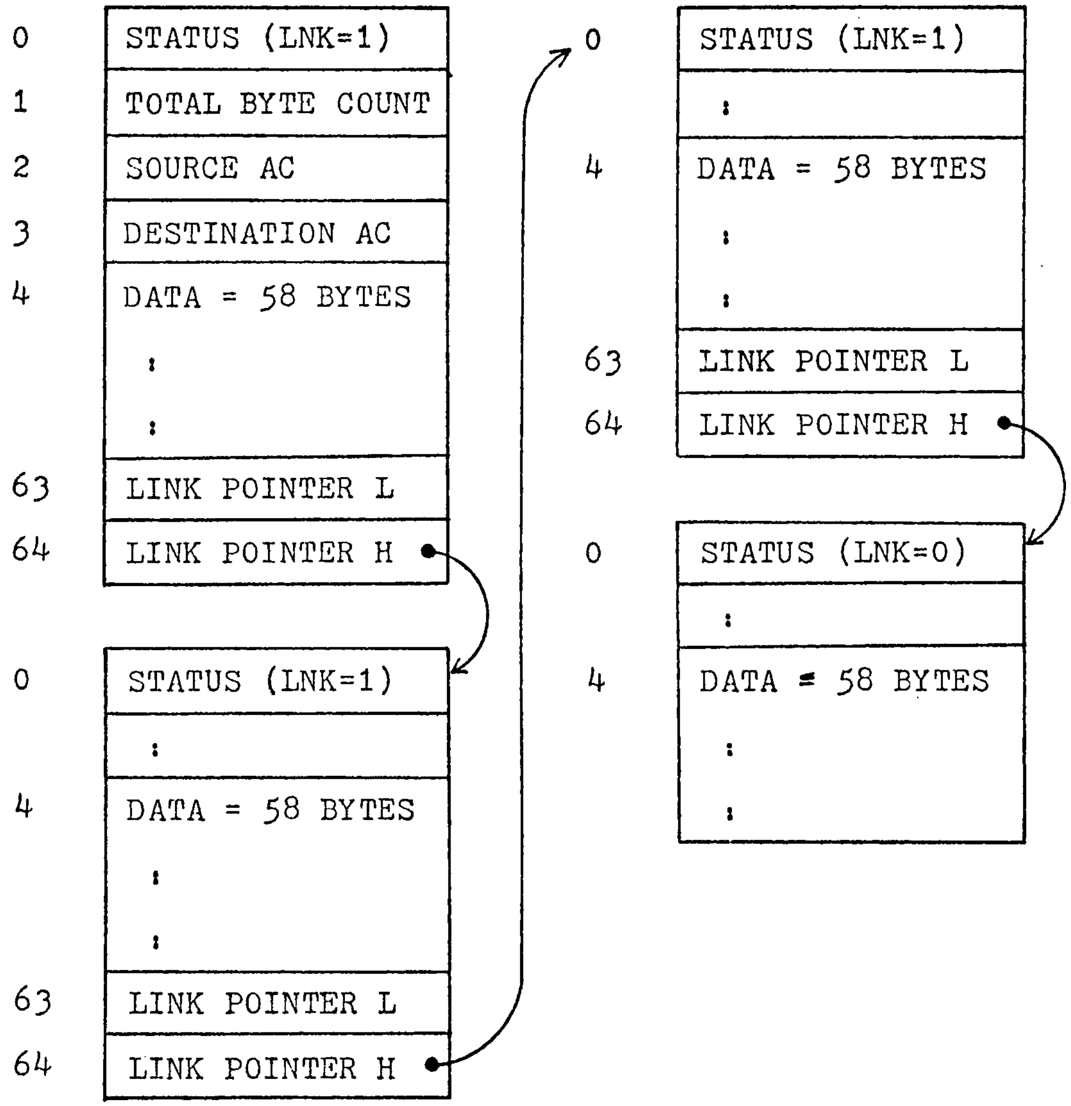


Fig. 5B MULTI-BLOCK BUFFER FORMAT AND LINKAGE

Blocks are linked up into a buffer by the use of a linkage pointer in the last 2 bytes of the block. There is also a bit in the status byte (link bit) which must be set to indicate that one block is linked to another. If the link bit is not set, it indicates this is the last block, or the only block, in the buffer

The buffer header (Fig. 5B) is only present in the first block of each buffer. It contains the buffer status byte as the 1st byte, the total byte count of the buffer ($1 \leq$ count ≤ 232) as the 2nd byte, the source address code of the buffer as the 3rd byte, and the destination address code as the 4th byte. The 5th through the 62nd byte is used for data storage and the 63rd and 64th hold the linkage pointer if necessary.

The status bytes also carries other information in addition to the link bit. The following is the bit arrangement of the status byte and the meaning of each bit.

	7	6	5	4	3	2	1	0
BUFs	---	LNK	ETB	PAR	HOST	---	---	---

LNK - The Link bit indicates this is not the last block in the buffer and that a linkage address will be found as the end of the block.

ETB - The Extended Buffer bit indicates this is not the last buffer in this I/O operation (i.e. Input was more than 232 bytes).

PAR - The Parity bit indicates a character with bad parity occured somewhere within this buffer.

HOST - The HOST bit indicates this buffer's destination is the HOST system.

The last 3 bits above only occur in the 1st status byte of each buffer (the buffer header), however, the link bit is in every status byte of every block in the buffer to indicate the presents or absence of another block.

The <u>HOST buffers</u> are a fixed size with 1024 bytes being allocated for each (Fig. 5C). There are two HOST buffers; one buffer (XBUF) for transmitting messages to the HOST, and one buffer (RBUF) for receiving HOST message transmissions. The buffer header for the HOST buffers consists of 4 bytes. The 1st byte holds the status information, the 2nd and 3rd bytes hold the total byte count ($6 \leq$ count ≤ 1020) and the 4th is reserved for future designation.

0	STATUS
1	TOTAL BYTES L
2	TOTAL BYTES H
3	— — —
4	DATA $\leq$ 1020 BYTES : :

Fig. 5C HOST BUFFER FORMAT

The status bytes for the HOST buffers are explained below:

	7	6	5	4	3	2	1	0
RBUF -	ERR	LOG	SVM	NSC	ACK	EMTY	FULL	BSY
XBUF -	ERR	---	---	---	---	EMTY	FULL	BSY

ERR - The Error bit indicates an error of some kind has occured in the buffer.

LOG - The Logical message bit indicates there are logical messages present in the RBUF for processing.

SVM - The Service Message bit indicates this RBUF is a service message.

NSC - The New SC bit indicates this RBUF has a different SC in the header than the last transmission.

ACK - The Acknowledge bit indicates the HOST has acknowledged the reception of the last transmission.

EMTY - The Empty bit indicates there is no data in the buffer.

FULL - The Full bit indicates no more data can be put into the buffer.

BSY - The Busy bit indicates the buffer is now in the act of being changed by some process and cannot be accessed by another.

Each I/O device has associated with it a Device Control Block (DCB) which contains all pertinent data for that device. All permanent data such as bus channel address and interrupt mask are stored here, and all temporary parameters such as byte count and buffer address used during I/O operations are also stored here. Figure 5D shows the format of the DCB used for device I/O.

0	STATUS
1	INTR MASK
2	DEVICE BUS ADDR
3	AC (DEVICE LU#)
4	DST AC (READ)
5	BUF ADDR L
6	BUF ADDR H
7	REMAINING BYTES
8	BLOCK STATUS
9	TOTAL BYTES
10	DATA POINTER L
11	DATA POINTER H

Fig. 5D DEVICE CONTROL BLOCK FORMAT

The individual DCB bytes are as follows:

(1) The device <u>status</u> holds the status bits for this device.

(2) The <u>interrupt mask</u> is used for enabling/disabling the interrupt register.

(3) The <u>device bus address</u> is the number of the I/O port.

(4) The <u>address code</u> is the logical unit number of the device.

(5) The <u>destination address code</u> is the logical unit number of the device.

(6) The present <u>buffer address</u> is the address of the buffer assigned to this device during an I/O cycle.

(7) The <u>remaining bytes</u> are the bytes left to be input to or output from this block.

(8) The present <u>block status</u> is a save area for the status of the block in use.

(9) The <u>total bytes</u> is the total remaining bytes left to be input to or output from the buffer.

(10) The buffer <u>data pointer</u> is the pointer to the next byte of input or output.

The status byte of the DCB is explained below:

	7	6	5	4	3	2	1	0
DCB	ERR	IDM	ETB	ONLN	ACK	XBF	RD	BSY

ERR - The Error bit indicates that an error has occured on an I/O transfer to this device.

IDM - The Identification Message bit indicates that an ID header should be attached to the start of each buffer sent by this device.

ETB - The Entended Buffer bit indicates that the present buffer is not the end of the message.

ONLN - The On Line bit indicates that this device is ready for I/O.

ACK - The Acknowledge bit indicates that the last transmission from this device to the host has been received.

XBF - The Xmit message buffer bit indicates that there is now a message waiting on the XQUE to be sent to the HOST (only 1 message is allowed to be on the XQUE from any single device at any particular time).

RD - The Read bit indicates that this is an input operation.

BSY - The Busy bit indicates that this device is performing an I/O operation and cannot start another until this one is complete.

The HOST has associated with it a <u>Host Control Block</u> (HCB) which contains all pertinent data for the HOST. Since the HOST I/O driver routines are not shared by any other devices, the HCB does not have to hold nearly as much information as the DCB. Figure 5E shows the format of the HCB.

0	STATUS
1	REMAINING BYTES L
2	REMAINING BYTES H
3	DATA POINTER L
4	DATA POINTER H

Fig. 5E HOST CONTROL BLOCK FORMAT

The individual bytes are as follows:

(1) The Device Status byte holds the HOST status.

(2) The Remaining Bytes indicates the total bytes left to be received or transmitted in this buffer.

(3) The Data Pointer is the pointer to the next byte to be received or transmitted.

The status byte for the HCB is explained below:

	7	6	5	4	3	2	1	0
HCB	ERR	SYNC	ETB	ONLN	ACPT	---	RD	BSY

ERR -The Error bit indicates that an error has occured on an I/O transfer to the HOST.

SYNC-The Synchronized bit indicates that the HOST I/O unit has received the correct sync characters.

ETB - The Extended Buffer bit indicates the present buffer is not the end of the message.

ONLN - The On Line bit indicates the HOST is ready for I/O.

ACPT - The Accept all calls bit indicates the HOST will accept all new devices which sign on.

RD - The Read bit indicates this is a HOST receive operation.

BSY - The Busy bit indicates the HOST is performing I/O.

The DCB for any particular device is acquired through a table (DCBTAB). The AC of the device is all that is needed to calculate the offset for the table, which contains all of the DCBs for every device in the system. Figure 5F is an example of how this table is set up.

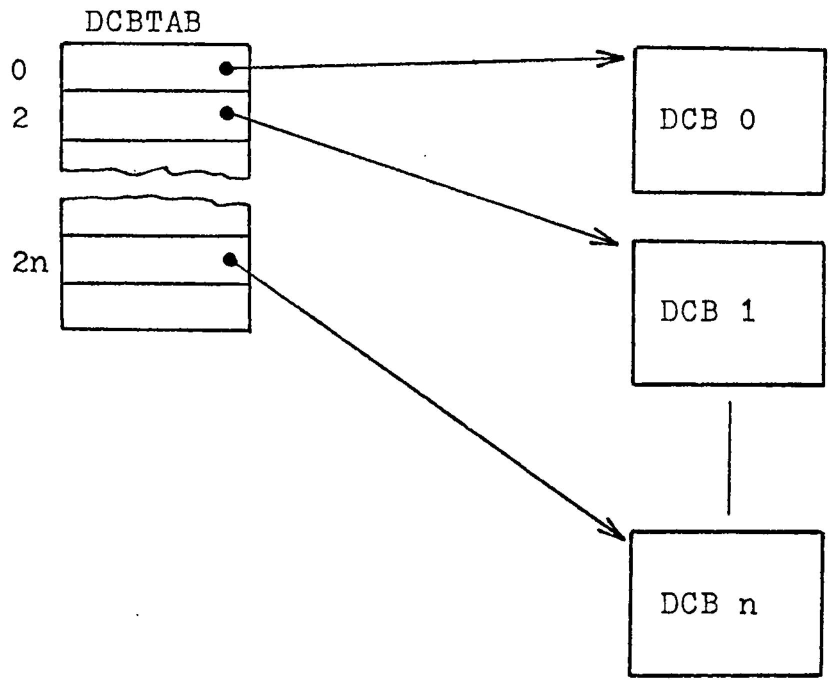


Fig. 5F DCB POINTER TABLE

All I/O within the program between two devices and between devices and the HOST is carried on through the I/O buffer. To prevent the possibility of interference between devices, all buffers are handled through queues on a first in first out basis. If a buffer taken from the top of a queue is destined for a device which is presently busy, it is put back on the bottom of the same queue to wait until the device is not busy. Figure 5G is a diagram of the queue buffer relationship.

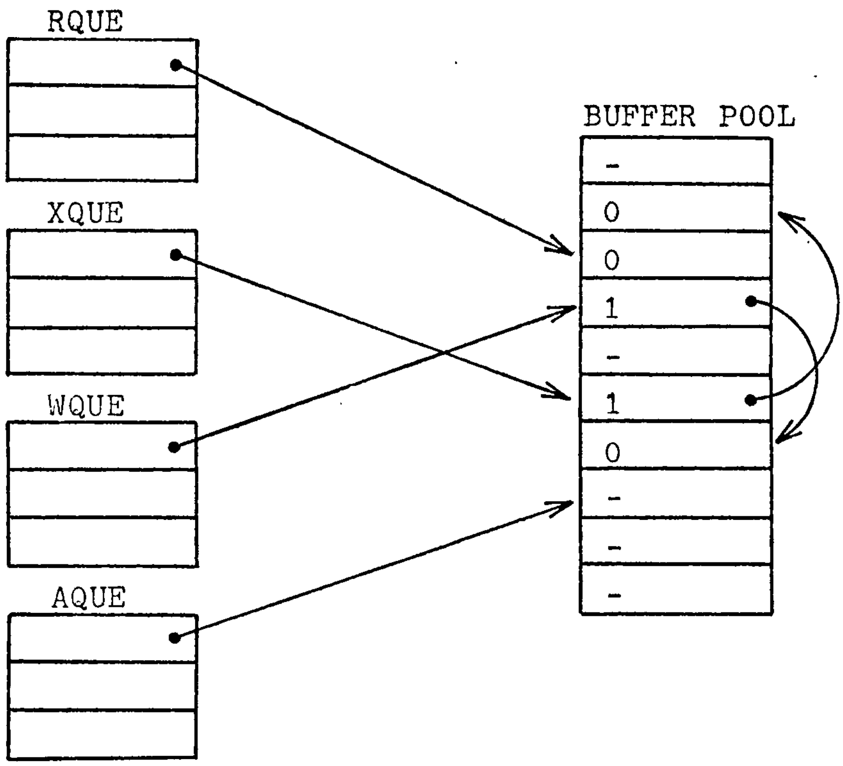


Fig. 5G QUEUE - BUFFER LINKAGE

The I/O is structured such that when a read or a write is started, for a particular device, the operation must run to completion before another can be started. This means that for a write operation, the complete buffer must be output, and for a read, either a line delimiter (CR) must be found, or the total length of input must exceed 232 characters. In the latter case, the present I/O buffer is terminated with an 'ETB' and put on a queue, and a new I/O buffer is started to receive the rest of the input.

There are 5 queues used by the program:

(1) The Available blocks queue (AQUE) holds the addresses of all blocks not presently in use.

(2) The Receive queue (RQUE) holds the address of all output buffers ready for output to a device.

(3) The Transmit queue (XQUE) holds the address of all output buffers ready for output to the HOST.

(4) The Write queue (WQUE) holds all overflow from the XQUE and the RQUE, and buffers which need to wait for output to a device or to the HOST.

(5) The Error queue (EQUE) holds both buffer addresses and queue designators which are placed there when an error occurs in any other queue

Chapter 6

SUBROUTINE DESCRIPTIONS

MAIN PROGRAM

The Main program loop (MAIN) is continously executed by the system until an interrupt occurs from one of the timers. If no device requires service at this time, then control is returned to this routine. During its execution the MAIN routine services each queue, if a buffer is on the RQUE it takes the buffer off the queue and starts output to the device indicated by the buffer, if a buffer is on the WQUE it moves this buffer to either the XQUE or the RQUE, and if a buffer is on the EQUE it moves the buffer to the required new queue.

MAIN also services the host receive buffer (RBUF) and the host transmit buffer (XBUF) when necessary. When the RBUF needs service, a flag causes the RBUF service routine to be entered which takes RBUF and converts all of its messages to output buffers and stores the addresses on the RQUE. And when the XBUF needs service, a flag causes the XBUF service routine to be entered, which takes all buffers off the XQUE and converts them to logical messages and puts them into the XBUF for transmission to the host processor.

The Initalize routine (INIT) is the routine which initalizes all necessary variables, initalizes all queues, sets up the stack pointer, sets up the I/O controller, starts an input cycle to all devices and enables the interrupts.

The Clock and I/O set up routine (CIO) is called by the INIT routine to set up the clock and timer interrupts, to set up the I/O device controllers, and to set up the priority encoder for device service requests.

COMMAND PROCESSOR

The Command Processor routine (CMDPRC) executes all system commands. It receives these commands from all devices in the form of buffers. The buffers are analyzed by the command processor and the appropriate action is taken. All commands start with a 'CTL C' followed by the command.

The commands are as follows:

ASSIGN - Consists of an 'A', followed by a one or two digit number. This command assigns the device, designated by the number, as the destination of the device issuing the command. All further inputs from the sorce device are routed to this destination device.

ATTACH - Consists of 'CTL A'. This command does two things: first it assigns the Host processor to the source device and second, it sends a select message to the Host to connect the source device to the Host.

DETACH - Consists of a 'D'. This command causes a detach message to be sent to the Host, which disconnects the source device from the Host. (Causes an immediate 'CP DISCONNECT')

The Command Input routine (CMDIN) is entered from DEVRD when a 'CTL C' is detected. It sets up a buffer to receive the command and pass it to the command processor routine.

DEVICE I/O ROUTINES

The Device Input routine (DEVIN) sets up the parameters in the device DCB for input from the device. It first acquires a block from the buffer pool and stores its address in the DCB. It then sets up the source and destination in the buffer header, initalizes the other parameters in the DCB for input from the device, and clears the read mask bit for this device in the I/O service mask.

The Device Output routine (DEVOUT) sets up the parameters in the device DCB for output to the device. On entry the buffer address is stored in the DCB. It then gets the byte count from the buffer and stores it in the DCB, initializes the other parameters in the DCB for output to the device, and clears the write mask bit for this device in the I/O service mask.

The Interrupt routine (INTR) is the device interrupt handler. This routine is entered once every millisecond from a timer interrupt. On entry it checks first for the Host needing service and then for any device needing service. If no service is needed, an exit is taken. However, if the Host needs service, the Host service routine is called. Additionally, if any device has a service request bit set and the device is not masked, the routine finds the correct DCB for that device, loads the registers with parameters from it, and then calls the service routine to service that device.

The Device Read routine (DEVRD), which is called by INTR, reads a character from the device designated by the parameters in the registers. It then stores this character in the designated buffer, updates the parameters, checks for the end of line character, checks for the last character in the present block or checks for a command character.

If the character read indicates the end of the line, then the buffer is closed and the device placed in idle mode. If the character read is the lst in the present block, then the block is closed and a new one linked to the present one, and if the character read is a command character ('CTL C') then the present buffer is aborted and a command buffer is initiated by CMDIN.

The Device Write routine (DEVWR), which is called by INTR, writes a character to the device designated by the parameters in the registers. It then updates the parameters, checks for the end of the buffer, checks for the end of the present block, or checks for the extended buffer.

If this is the last character in the buffer, the routine restores the block to the buffer pool and places the device in the idle mode, if it is only the end of the present block it restores this block to the buffer pool and gets the address of the next block, and if it is an extended buffer, it restores this block to the buffer pool and then sets the device up to receive another output buffer.

HOST I/O

The Start Receive Buffer routine (STRBUF) is called by RBFSRV or GENXBF to start the next RBUF input cycle. It sets up the necessary parameters in the Host Control Block and unmasks the interrupt for input from the HOST.

The Start Transmit Buffer routine (STRXBF) is called by RBFSRV or GENXBF to start the next XBUF output cycle. It sets up the necessary parameters in the HCB and unmasks the interrupt for output to the HOST.

The Host Receive routine (HOSTR) is entered from an interrupt, and if the Host I/O controller is in sync, a byte is read from the Host and placed in the RBUF. If it is an end of message character the full flag is set, the buffer is closed, and the interrupt mask set.

The Host Transmit routine (HOSTX) is entered fron an interrupt, and it transmits the next byte of XBUF to the Host. If it is the end of the buffer, the empty flag is set and the interrupt mask set.

HOST BUFFER CONVERSION

The Receive Buffer Service routine (RBFSRV) is called by MAIN to service the RBUF. It first strips the link message from the RBUF, analyzes the link message and uses it to set the RBUF status flags. Next, depending on the flag setting, this routine will retransmit the old XBUF, transmit a service message, generate and transmit a new XBUF or convert all logical messages in the RBUF to output buffers and put them on the RQUE. The routine then starts reception of the next RBUF.

The Strip Link routine (STRLNK) is called by RBFSRV to strip the link message off the RBUF. Depending on the data in the header, it will set or clear the flags in the RBUF status byte.

The Service Message routine (SRVMSG) is called by RBFSRV to analyze RBUF service messages and either send back the correct acknowledge message or initiate the appropriate action.

The Get Message routine (GETMSG) is called by RBFSRV to get the next logical message off RBUF. It also calculates the length of the message and stores the address in a save area.

The Get Buffer routine (GETBUF) is called by RBFSRV to get a buffer to store the logical message. It gets enough blocks off the AQUE to hold the logical message and links them together as one buffer.

The Convert Logical Messages routine (CONMSG) is called by RBFSRV to convert the logical message to an output buffer and store the data in the buffer. It also puts the necessary header data into the buffer header.

The Generate XBUF routine (GENXBF) is called by RBFSRV to generate the next XBUF for output to the Host. It first generates a new link header for the XBUF using information from the present RBUF. It then gets buffers off the XQUE, converts them to logical messages and puts them into the XBUF. When the buffer is full or there are no more buffers on the XQUE, it starts transmission of the new XBUF.

The <u>Generate Link</u> routine (GENLNK) is called by GENXBF to generate a link message for the new XBUF. It analyzes the data in the present RBUF link header, generates the new link header and puts it in XBUF.

The <u>Convert To Logical Messages</u> routine (CONLOG) is called by GENXBF to convert input buffers to logical messages and put them into XBUF. It first generates a logical message header, puts it in the XBUF, and then moves the data from the input buffer to XBUF.

<u>QUEUE HANDLERS</u>

The queues are set up as simple linear lists with pointers to the top and bottom stored in the header. The header also holds the status byte, and the top and bottom buffer pointers

The <u>Put On Queue</u> routine (PUT) is called by many routines to put a value on a queue. It stores the two byte value, in the HL register, at the location pointed to by the top queue pointer. It then updates the pointer and,if the queue is full, sets the flag.

The <u>Get From Queue</u> routine (GET) is called by many routines to get a value from a queue. It gets the value pointed to by the bottom queue pointer and places that value in the HL register. It then updates the pointer and if the queue is empty, sets the flag.

CONCLUSION

This project has clearly shown the ability to produce a useful microprocessor development system with a bare minimum of hardware. The total cost of all hardware for this system came to less than $1,000. The FORTRAN cross assembler,however, which normally sells for $250.00, was donated by MOSTEK. And the cost of the console terminal is not considered because it was already owned by the university. But even including the prices of these items, this system still compares favorably with stand alone systems selling for up to $10,000.

This system would be perfect for an application such as microprocessor training for a number of students. All programs could be written and assembled on normal time-share terminals and then loaded into the development system for testing. The low price means several units could be accquired for the same price as one expensive stand alone system.

While the program discussed in the second part of the thesis was written and tested on the microprocessor development system, it was never completely tested with the HOST system. This was due to the lack of availability of an RNP line on the HOST system. And the effort necessary to implement such a line on the HOST was too extensive to be completed in the allotted time. However, the software routines were all tested locally and worked well with test programs.

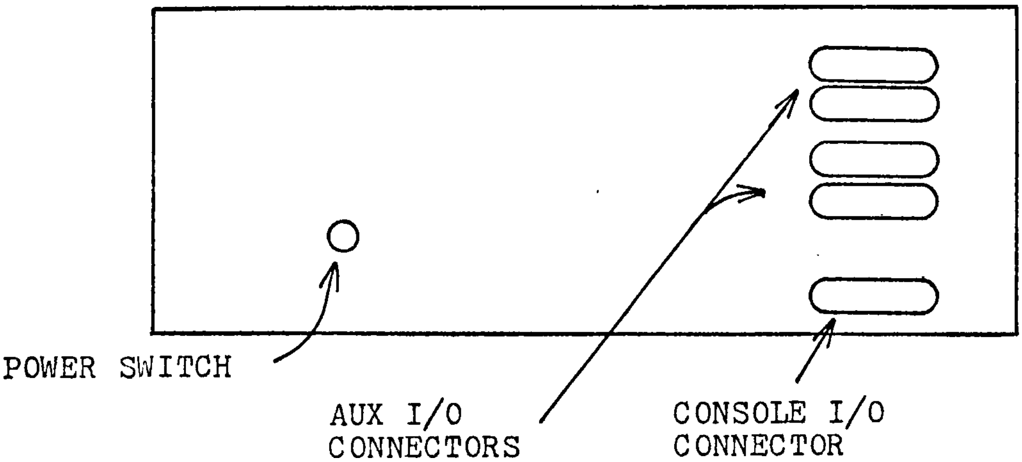
Although the emulator program should be very useful as it is, with a few changes to the harware and the software it could be made much more versatile and efficient. For example, the number of devices which it could service could be increased considerably by adding a different interrupt scheme and a DMA capability to the HOST I/O line, although as it stands, it could probably handle up to 16 low speed terminals (300 baud).

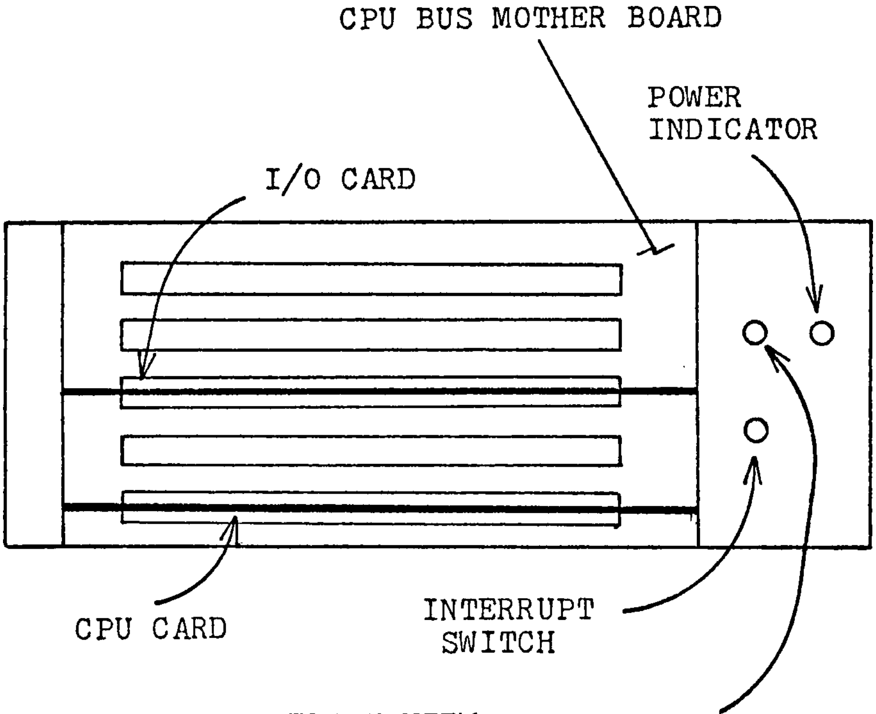
In addition, the emulator could be used with a different HOST machine just by changing the routines which determine the protocol. However, the basic framework of the emulator and most of the subroutines would remain the same as they are presently.

APPENDIX A

HARDWARE DIAGRAMS

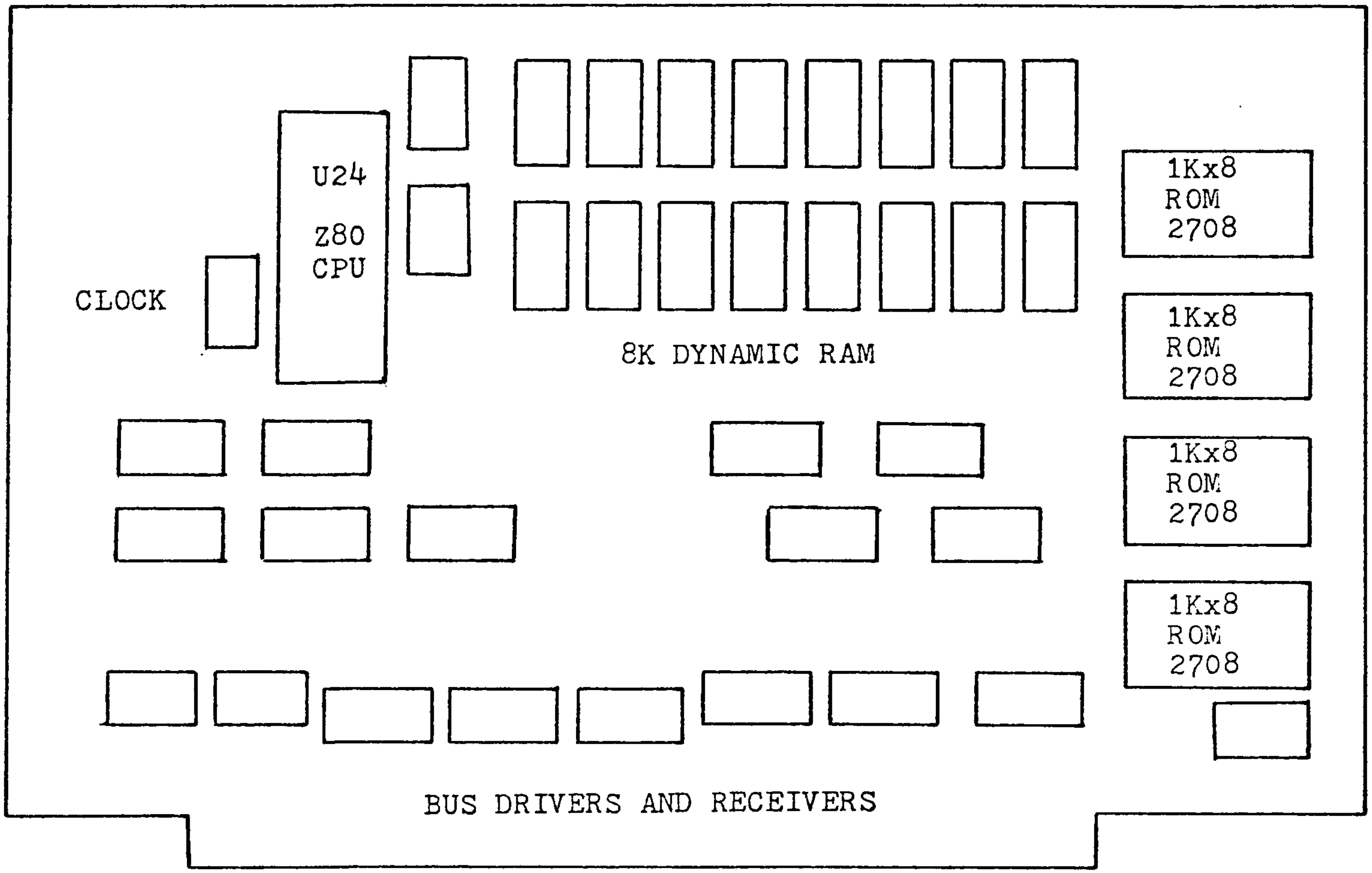
REAR VIEW



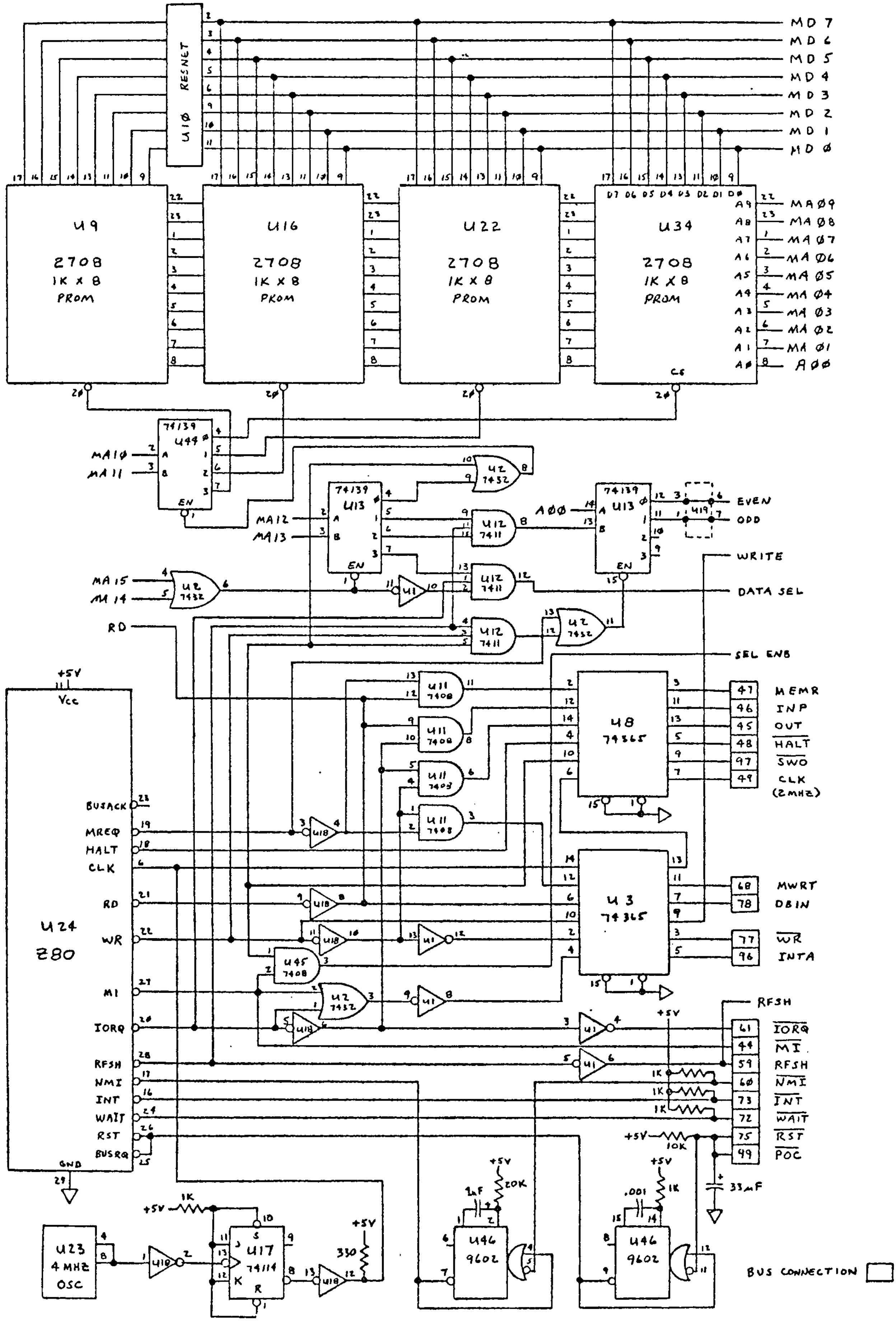


FRONT VIEW

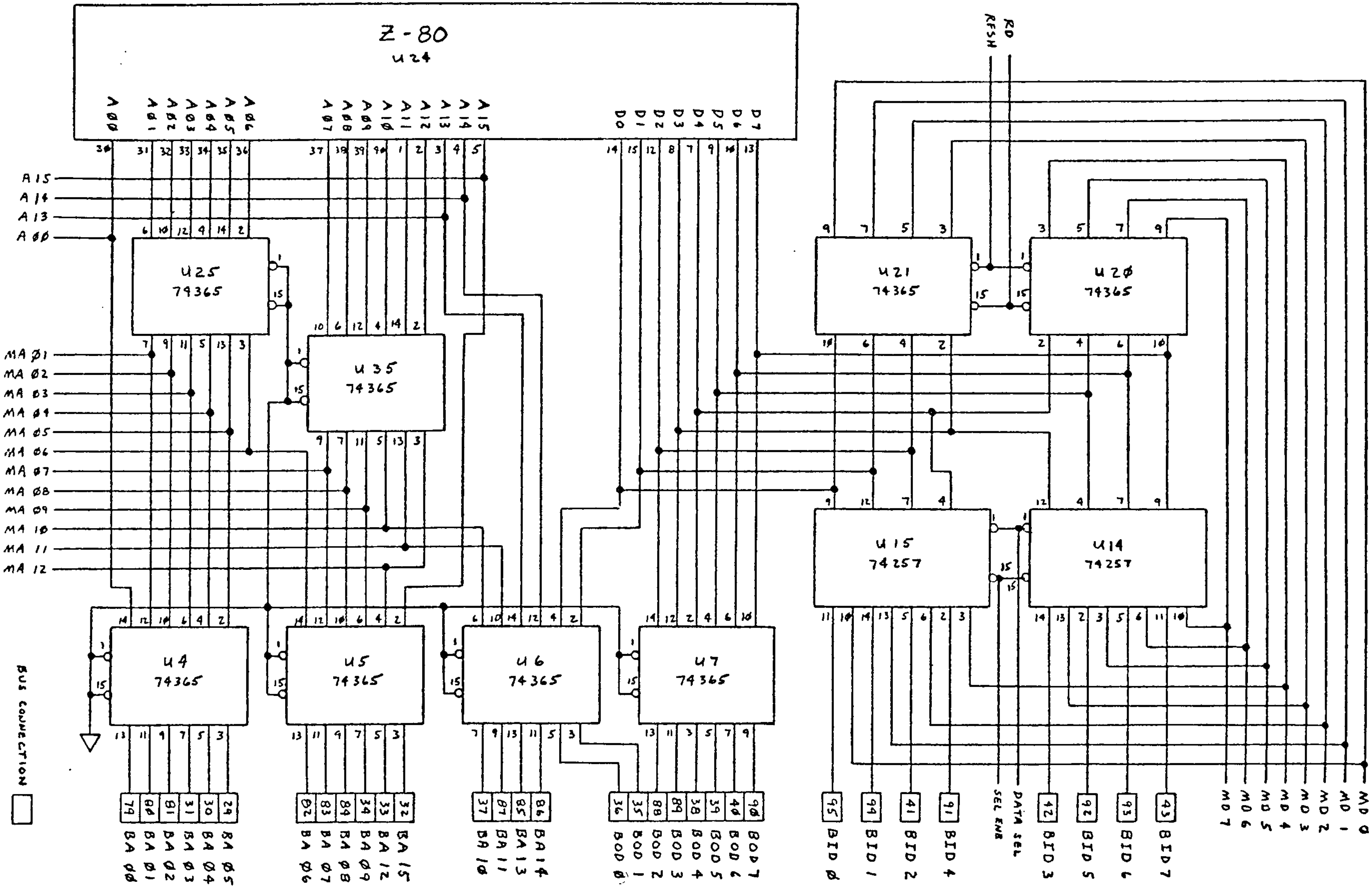
SYSTEM FRONT AND REAR VIEW



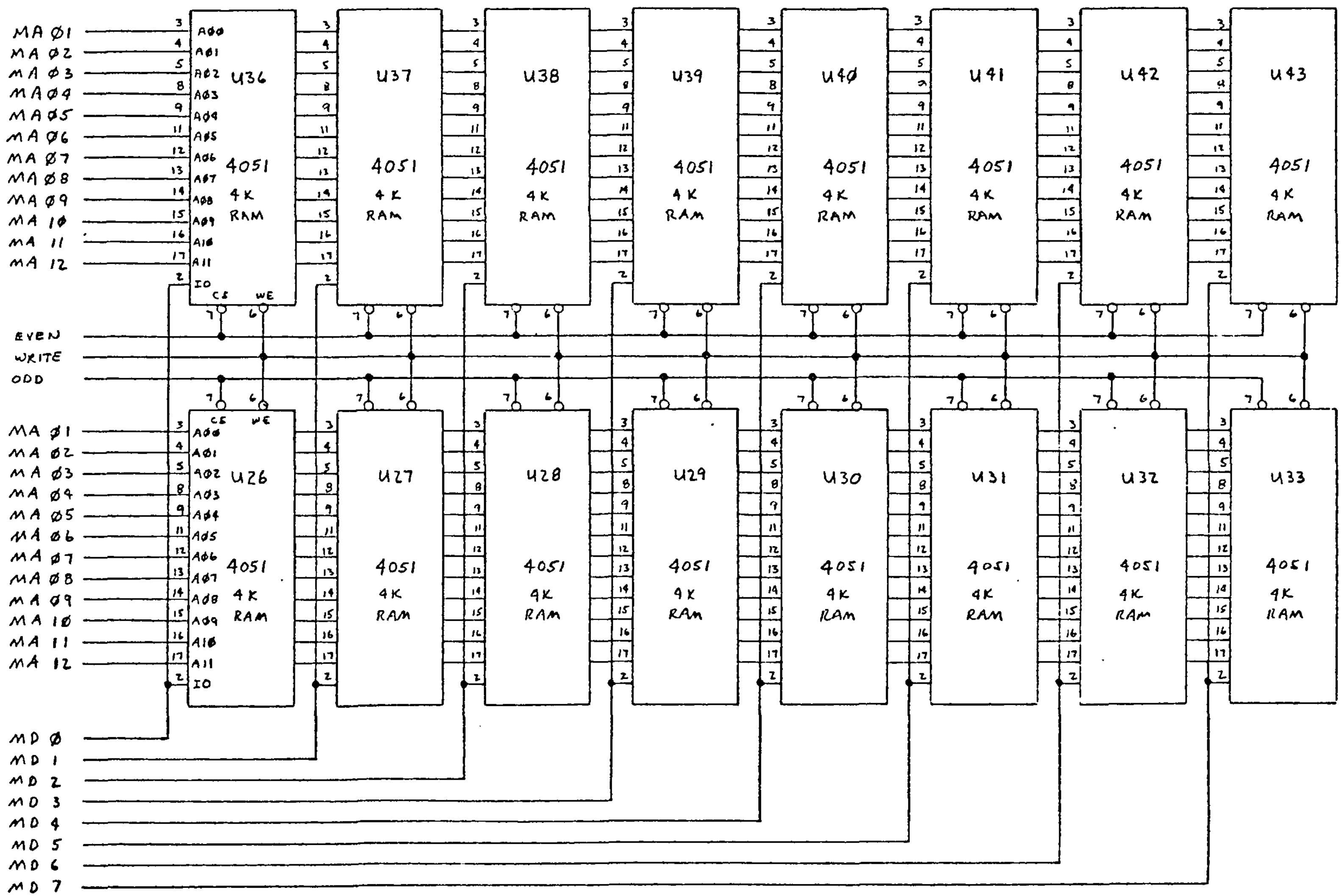
CPU CARD PHYSICAL LAYOUT



CPU CARD (LOGIC CIRCUITS)
Sheet 1 of 3



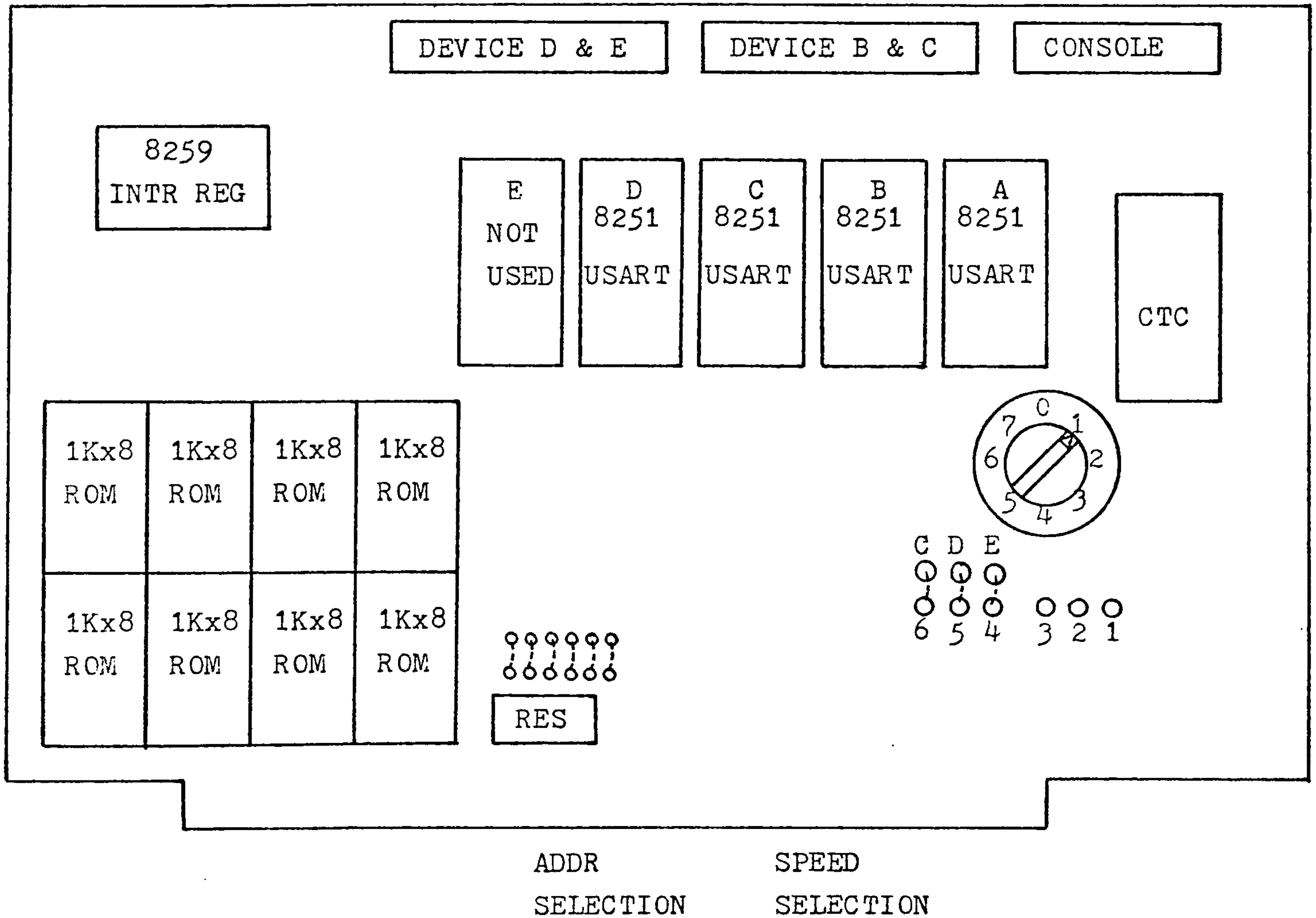
CPU CARD (BUS INTERFACE)
Sheet 2 of 3



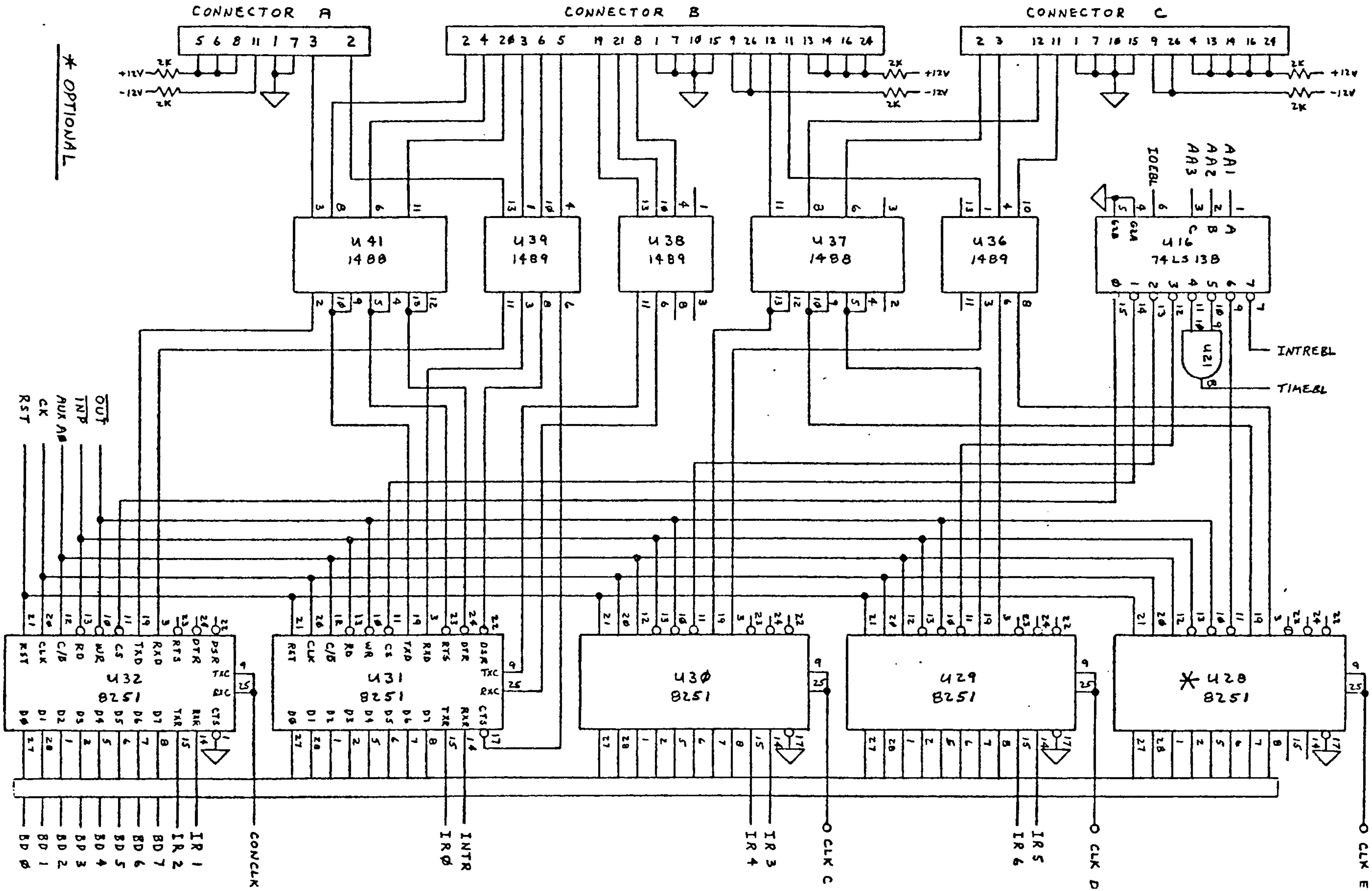
CPU CARD (RAM STORAGE)
Sheet 3 of 3

DEVICE	ADDR
A	10H
B	12H
C	14H
D	16H
E	NOT USED

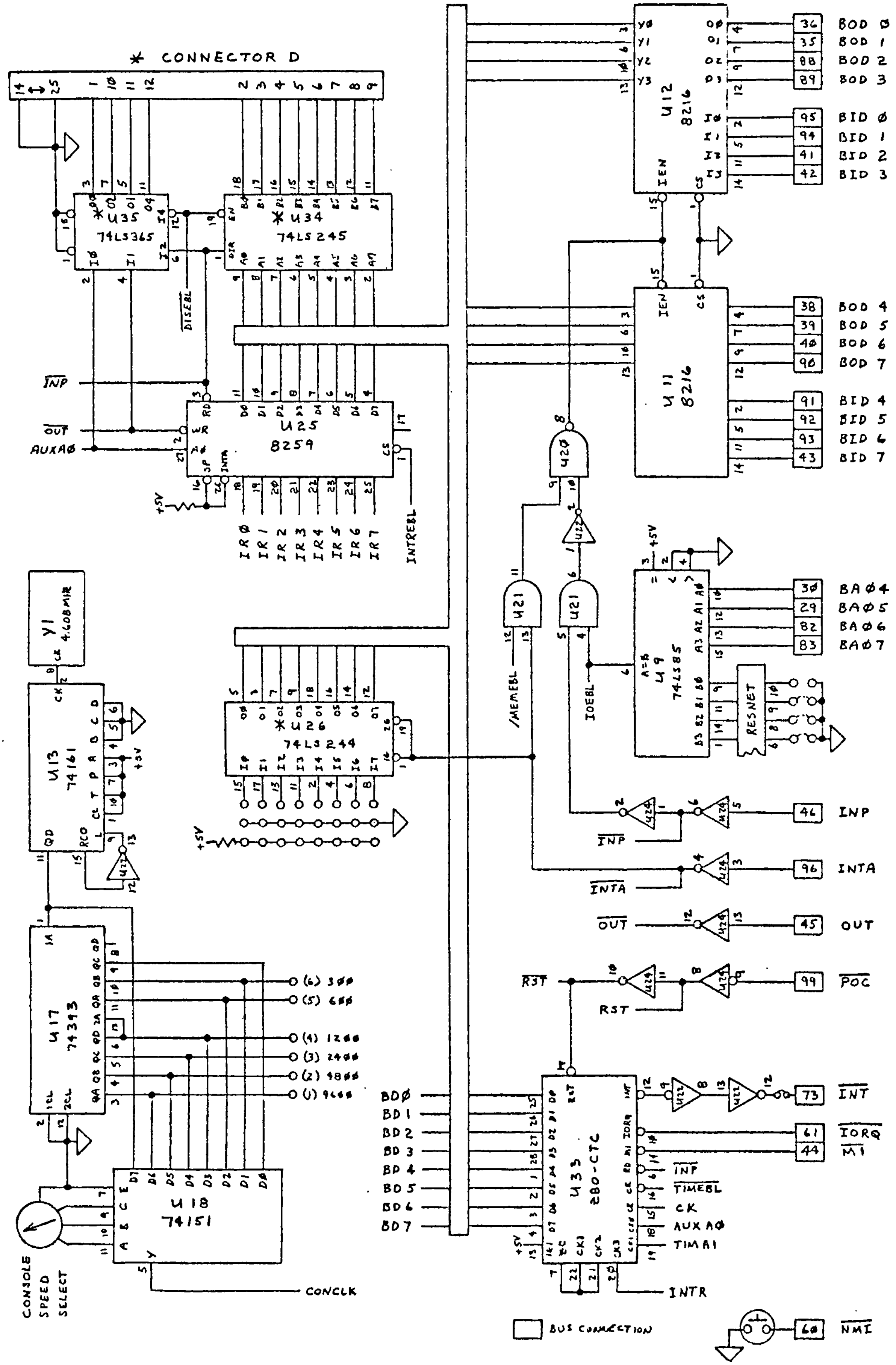
SWITCH POS	BAUD
1	9600
2	4800
3	2400
4	1200
5	600
6	300
7	150
0	---



I/O CARD PHYSICAL LAYOUT

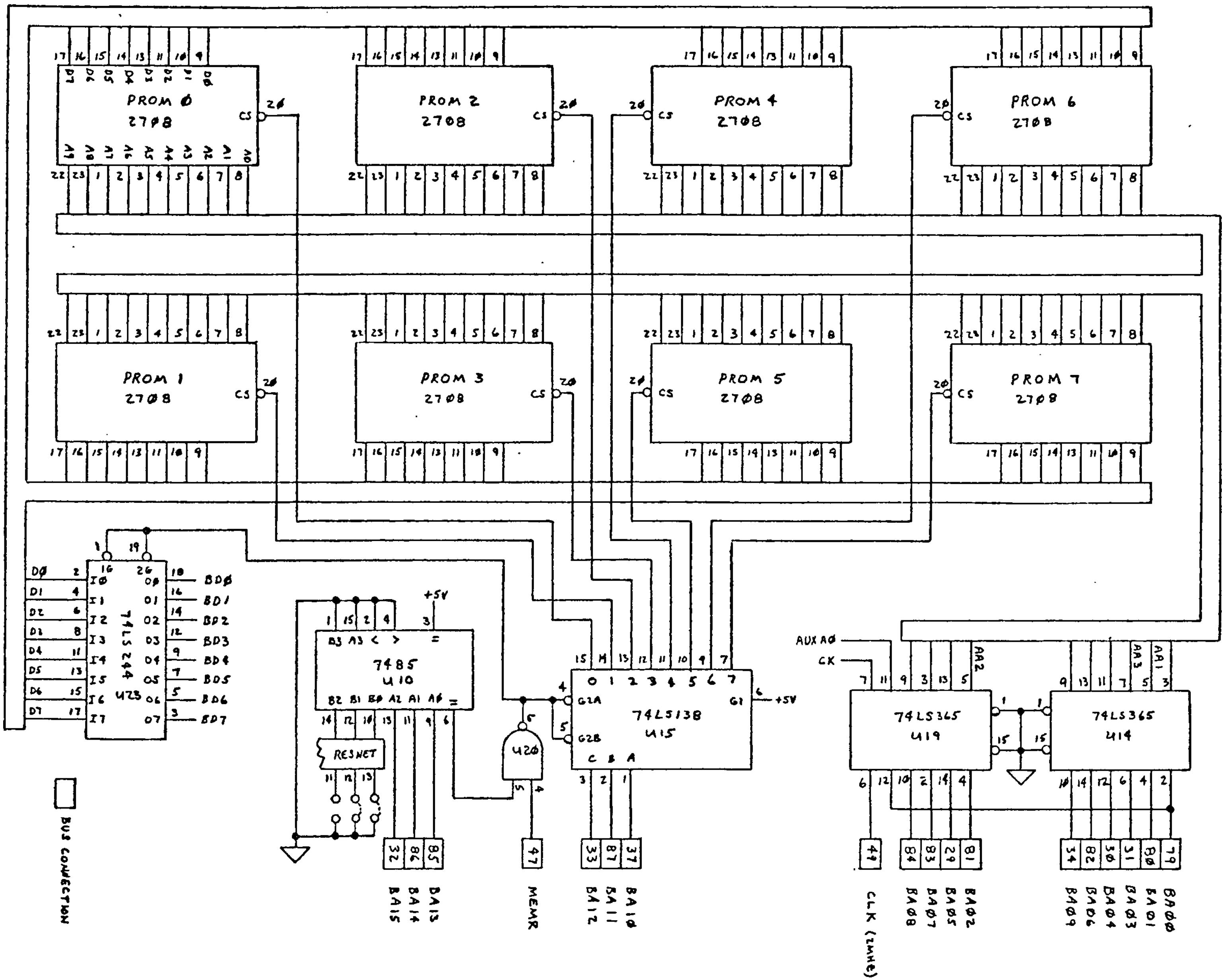


I/O CARD (SERIAL PORTS)
Sheet 1 of 3



I/O CARD (INTERRUPTS AND CLOCK)
Sheet 2 of 3

I/O CARD (PROM STORAGE)
Sheet 3 of 3



APPENDIX B

ZAPPLE MONITOR COMMANDS

NOTE - The text in Appendix B was copied from the ZAPPLE MONITOR OPERATIONS MANUAL

by

Roger Amidon, Technical Design Labs

APPENDIX B

COMMANDS

The following is a list of commands for the Zapple Monitor. Precise definitions and usage notes are covered in the next section.

A - ASSIGN reader, punch, console or list device options from the console.
B - BYE (system shut down).
C - COMPARE the contents of memory with the reader input and display any differences.
D - DISPLAY the contents of any defined memory area in Hex.
E - END OF FILE statement generator.
F - FILL any define area of memory with a constant.
G - GOTO an address and execute. With breakpointing.
H - HEX MATH. Gives the sum and difference of two Hex numbers.
I - VERIFY ROM. Verifies contents of ROM against memory.
J - JUSTIFY MEMORY - a non-destructive test for hard memory failures.
K - Jump to HOST I/O Routine.
L - LOAD a binary file.
M - MOVE a defined memory area to another starting address.
N - NULLS to the punch device.
O - PROGRAM ROM. Programs ROM from memory.
P - PUT ASCII characters into memory from the keyboard.
Q - QUERY I/O ports - may output or input any value to or from any I/O port.
R - READ a Hex file. Performs checksum, relocating, offsetting, etc.
S - SUBSTITUTE and/or examine any value at any address (in hex).
T - TYPES the contents of a defined memory block in their ASCII equivalent.
U - UNLOAD a binary tape to the punch device.
V - VERIFY the contents of a defined memory block agains that of another block and display the differences.
W - WRITE a checksummed hex file to the punch device.
X - eXAMINE and/or modify any or all registers including the special Z-80 registers.
Y - "Yis there". Search memory for defined byte strings and display all addresses where they are found.
Z - "Z end". Locate and display the highest address in memory.

COMMAND SET USAGE

The following section lists the commands, and describes their format and their use. It should be noted that the Zapple Monitor recognizes both upper and lower case letters for its commands, and that in general, a command which is printing can be stopped with a CONTROL C, which is checked during a carrage return - line feed sequence. The following EXAMPLES show a comma (,) as a delimiter between parameters, however a space may also be used. If an error is made while inputting a command from the keyboard, it may be terminated by a rubout and the command re-typed. An asterisk is displayed indicating an ABORT of some kind.

COMMAND   DESCRIPTION

A   ASSIGNMENT OF I/O DEVICE: The monitor system is capable of supporting up to 4 logical devices, these being: the CONSOLE, the READER, the PUNCH, and the LIST DEVICE. To these may be connected 4 different actual I/O devices, for a total of 16 direct combinations of I/O device and function. The specific permutations are:

LOGICAL DEVICE	ASSIGNED DEVICES
CONSOLE	TTY CRT BATCH USER (user defined)
READER	TTY CASSETTE PAPER (HIGH SPEED READER user written) USER (user defined)
PUNCH	TTY CASSETTE PAPER (HIGH SPEED PUNCH user written) USER (user defined)
LIST DEVICE	TTY CRT LINE PRINTER (user written) USER (user defined)

The default mode for each logical device is always the teleprinter.

Assignments are made using the following format;

EXAMPLE: AC=C(cr)

assigns the console equal to the Crt (video terminal) device. similarly:

EXAMPLE: AR=T(cr)

assigns the reader device to be the teleprinter.

While performing a command which requires a reader input (C,L,R), if the assigned reader is the Teleprinter, the software will look for a character from the TTY input. If a character is not recieved within a few seconds, it will ABORT, printing an asterisk (*) and return to the command mode. Similarly, if the assigned reader is the Cassette device, and you WISH to abort for some reason, changing the position of any of the SENSE switches will force an ABORT. On the external reader routines, returning with the carry set indicates an abort (or OUT OF DATA) conditinn.

When assigning a device, only the first letter initial of its name is required.

The Monitor itself is set-up to support the TTY, CRT, and Cassette routines. The other assignments require the addition of user's routines. These are addressed via the commands, which vector to starting addresses.

EXAMPLE; AL=L(cr)

assigns the list device to be the line printer. It vectors to (start address) +812H, or 12H above the end of the monitor. That would be the address for the line printer routine. For details of these arrangements, see the Source Documentation.

Within the above, the assign console equals batch "AC=B(cr)" deserves further mention. In BATCH mode, the READER is made the Keyboard input, and the LIST DEVICE is made the console output. This allows the running of a job directly from the reader input, with the result being output to the list device.

A typical use of this assignment would be the reconstruction of a lengthy text editing job where the text and your editing commands have all been saved on paper tape. With the BATCH MODE, you may assign the reader equals the TTY, the List device equals the TTY, and Console equals BATCH. Running the tape through the reader is the same as you redoing the entire text editing by hand, and the output will go to the TTY and be printed. On a very lengthy job, you could even start the process, and go away until it's done. Its usefullness is limited only by your imagination.

B BYE. This command completely shuts down the system. It is useful where children might have access to the system, where a telephone communications link is established under remote control, or anytime when the operator wishes to make the system inaccessible to unauthorized use.

EXAMPLE: B

completely kill the keyboard, Recovery from the shut-down is accomplished simply by inputting a CONTROL-SHIFT N from the keyboard. (ASCII equivalent is a Record Separator - "RS"; HEX character is a 1EH.) The monitor will sign on and print a greater-than sign (>), however, the register storage area will not be cleared.

C COMPARE the reader input with memory. This command is useful for verifying correct loads, verifying that a dumped tape matches with its source, etc.

EXAMPLE: C1000,2000(cr, start reader)

compares the memory block 1000H to 2000H with the input from the reader device.

For those with automatic readers, the operation is very simple. Assign the Reader equal to the device you wish to enter the data against, type C (starting address), (ending address)(cr), and the reader will start. The first character read by the reader will be the one matched with the starting address. If any discrepencies are encountered, the reader will stop, and the address (in hex) of the error will be printed on the display. The reader will restart, and continue in this fashion until the entire tape is compared.

If your reader cannot operate automatically, start the reader manually. If an error is encountered, however, while the incorrect address is being printed, the reader will continue, and get "out of sync" with the compare action. Therefore, it is necessary to manually stop the reader if an error is encountered, and manually reposition the tape to the byte following the error. (An excellent article on how to convert ASR33 type readers to automatic operation was recently presented in INTERFACE magazine.)

D DISPLAY memory contents. This command displays the contents of memory in Hex. Memory is displayed 16 bytes per line, with the starting address of the line given as the first piece of data on the line.

EXAMPLE: D100,1FF (cr)

will display in hex the values contained in the memory block 100H to 1FFH.

E END OF FILE. This command generates the end of file pattern for the checksum loader. It is used after punching a block of memory to the punch device using the "W" command. An address parameter for the end of file may be given if so desired.

EXAMPLE: E(cr)

will generate an "end of file marker".

EXAMPLE: E100(cr)

generates the EOF marker with the address parameter "100H". When loading such a file, upon completion, the address contained in the End of File will be placed in the "P" register. Execution of the program may then be initiated by typing "G(cr)".

F FILL command. This command fills a block of memory with a specific value. It is quite handy for initializing a block to a specific value (such as for tests, zeroing memory when starting up, etc.) *NOTE: Avoid doing this over the monitor's stack area. This area may be determined as being between the value you get when typing the Z command, and the value in the S register upon sign-on. It is approximately 60H bytes below the "Top of memory"(Z).

The format for the command is:

EXAMPLE: F100,1FF,FF

fills memory block 100H to 1FFH with the value FFH.

G GOTO command. This command allows the user to cause the processor to GOTO an address and execute the program from that address. In the actural performing of the G command, a program, which has been placed in the stack area during the sign-on of the monitor, is executed. This program will first take all of the values in the register storage area (displayed with the X command), and stuff them in their correct registers in the CPU, and finally JMP to the program address being requested by the operator. If this short program up in the stack has been destroyed (as a result of a"blow-up", or the F or M commands, etc.) the monitor will not be able to GO anywhere, and a manual restart of the monitor will be required. Whenever the monitor is restarted at the initialization point (first address I.E. 0F000H), the contents of the registers are set to ZERO with the exception of the S (stack), which contains a valid stack address. This actual value depends on the amount of memory in the system, etc. In its simplest form, the letter "G" accompanied by a parameter causes the processor to go to that address and start execution.

EXAMPLE: G1000

would cause the processor to goto address 1000(H) and execute from that address.

Additionally, one or two breakpoints may be set.

EXAMPLE: G1000,1005,1010

would cause the program to start execution at address 1000H, and IN THE EVENT that the program gets to address 1005, OR 1010, the program will stop execution and return to the monitor, printing an "at" sign, and the address of the breakpoint that was executed. (ie. @1010). It then prints the ">"prompt, awaiting further instructions. This action also cancels any breakpoints previously set.

Breakpoints must be set at locations containing an instruction byte. This is a SOFTWARE breakpoint system, and requires either RAM at RST 7 (restart 7, addr. 0038H), or if using ROM, a permanent JMP to the monitor TRAP address (0F01EH) at 0038H. Remember, this is a SOFTWARE breakpoint system, and the program being debugged must be in non-protected Read/Write memory.

EXAMPLE:

```
*C2   JNZ    1234H
 34
 12
*3E   MVI    A,CR
 0D
*21   LXI    H,1000H
 00
 10
*77   MOV    M,A
*23   INX    H
*CD   CALL   5678H
 78
 56
```

The asterisks (*) mark the bytes that may be used as breakpoints.

H HEX MATH. This command allows the execution of hexidecimal arithmetic directly from the console. It will give the sum and difference of any two hex numbers entered.

```
EXAMPLE:   H1000,1010(cr)
           2010 FFF0
           >
```

2010H being the sum, and FFF0 being the difference of the two hex values.

J The J command is a non-destructive memory test. The command reads any given byte, complements it, writes into the location the complement, compares the complement with the accumulator, and rewrites the original byte into the location. The command is used with two parameters, delineating the block of memory to be checked.

```
EXAMPLE:   J1000,1FFF
           >
```

would perform the above test on the block 1000H to 1FFFH.

If errors are detected, the address at which the error is found and the error are displayed on the console before the test is continued.

```
EXAMPLE:   J1000,1FFF(cr)
           1F00  00001000
           >
```

would indicate that the 4th bit (D3) at location 1F00H did not correctly complement itself.

This test is useful for the discovery of hard memory failures, and also serves as a quick check for accidentally protected memory. A fully protected memory block would print out as entirely "1s". (11111111)

L LOAD BINARY FILE. This command loads a binary file from either a cassette or paper tape.

```
EXAMPLE:   L1000(cr)
```

would load the tape at address 1000H. This would require that the program be an absolute program, designed for address 1000H. The start-of-file mark (automatically generated by the "U" command) is a series of 8 0FFH's (rubouts). When this is detected at the start of file, the bell will ring on the TTY to indicate the start of the load process. When the end-of-file is detected (again, a series of 8 rubouts) the load is terminated, and the address of the NEXT location that would have been loaded is printed on the console. There are two constraints on this type of file system. The middle of the program can not contain more than 6 OFFs (11111111) in a row (an unusual occurence), and if OFFH is the LAST data byte in the file, it will be ignored. This too is unusual, and only a minor inconvienience.

Binary programs loaded at other than their design address will not run. The "L" command does not perform checksum functions, and cannot handle relocatable files. This is a pure and simple byte-for-byte binary loader (see "U" command.)

M MOVE COMMAND. This command is used to move a block of memory from one location to another. The original block is NOT affected by the move, remaining intact so long as the block moved into does not overlap with the block currently occupied. This command, like the "F" command should be used with some caution as moving a block into an area occupied by the stack, or the program or the monitor will cause unpredictable results.

EXAMPLE: M1000,1FFF,2000(cr)

moves the contents of memory contained in the block 1000H to 1FFFH to a starting address of 2000H. The new block has the limits 2000H to 2FFFH.

This command is very useful for working on programs without destroying the original, verifying blocks of memory loaded with existing memory, etc.

N NULL. This command punches nulls to the punch device. 72 nulls are punched whenever the command is used. It may be used repetitively for any desired leader length.

EXAMPLE: (N)
*Note: the "N" or "n" will NOT echo, so as to not spoil the paper tape.

It will punch 72 nulls to the punch device.

P PUT ASCII characters into memory. This command allows ASCII characters to be written directly into memory. It is useful for placing labels in files, etc.

EXAMPLE: P1000(cr)

activates the command, and any further inputs via the keyboard would be placed into memory in their ASCII equivalent. The command is terminated by a CONTROL D character, with the address of the location following the last entry printed on the console (the Control-D is NOT stored). Recovery of the input data is affected by use of the "T" or "U" command.

Q QUERY INPUT/OUTPUT PORTS. This command allows any value to be output to any I/O port, and allows the value in binary on any I/O port to be read on the console.

EXAMPLE: Q01,7(cr)

would output an ASCII "7" to I/O PORT 1. (ASCII seven is a "bell" so on a TTY, the bell would ring.)

EXAMPLE: QI1(cr) 00001101

inputs the value at port 1, in the illustration above, we see that bits 0,2 and 3 are high, the others low. This is useful for observing the condition of status bits and other diagnostic activities.

R READ A CHECKSUMMED HEX FILE. This command reads checksummed hex files in the INTEL format, as well as being capable of loading the relocatable TDL files at any selected address and bias offset. When reading an ABSOLUTE file (INTEL format), there may be only a BIAS added. These files cannot be relocated. The format is: R(bias),(relocation)(cr).

If a checksum error or a failure to write the data to memory occurs, the loading process is stopped, an asterisk is printed (indicating some error condition), and the address that was attempting to be written will be displayed on the console device. This is to assist in determining the failure.

EXAMPLE: R(cr, start reader)

will load a hex file at its absolute address.

EXAMPLE: R,1000(cr, start reader)

will load a TDL relocatable hex file at address 1000H and modify the program to run at address 1000H.

EXAMPLE: R1000,100 (cr, start reader)

loads the file set up to run at 100H, but with a positive BIAS of 1000H added to it. Thus, the file, set up to run at 100H will be loaded at 1100H.

EXAMPLE: R1000(cr)

will load the file, set up to run at address 0000H, at address 1000. In other words, using the TDL relocating format, you may load any program, to execute anywhere in memory, anywhere in memory. (Think about it......)

S SUBSTITUTE and examine. This command allows any address in memory to be examined directly, and allows substitution of one value for another at that address if desired.

EXAMPLE: SF810(sp)00-(sp)1A-(sp)C3-(sp)(cr)
>

In this case the "S" command examines address F810H. The hitting of the space bar (sp) displays the value of that address. (assuming value 00H at that address.) Hitting the space bar again displays the NEXT location in memory (F811H), and so forth. Simply typing S(sp) starts display from address 0000H. By repetitive typing of (sp), all of memory could be displayed one address at a time.

EXAMPLE: SF810(sp)00-(kb)FF(cr)

This command examines address F810H, showing the value 00H at that address. Immediately typing in FFH from the keyboard SUBSTITUTES FFH for 00H at that address. Repeating the example above would show:

EXAMPLE: SF810(sp)FF-

When an address is being examined, the address being examined may be moved BACKWORD by entering a backarrow (ba) or SHIFT-O, or underline, depending on the terminal used.

EXAMPLE: SF810(sp)00-(ba)AA-

shows that at address F80FH, the value AA exists. Typing a space bar will examine F810H again.

T TYPE ASCII characters from memory. This command allows the contents of memory to be displayed in their ASCII equivalents. All non-printing characters will be displayed as periods (.). It may be used to display the results of the 'P' command which allows keyboard entry of ASCII characters directly into memory. Also useful for finding text strings and messages in software. The initial address if first displayed, then the first 64 characters, the next address, etc. until the upper limit has been reached.

EXAMPLE: T1000,2000(cr)

displays the ASCII equivalents of memory locations 1000H to 2000H. If the 'P' command had been used to place a 'message' into memory somewhere in that memory block, it would soon be apparent on the console display.

U UNLOAD BINARY. This command simply dumps core to the punch device. It may be used with a cassette system as well, with no start-up problems. It does not generate a checksum. The format which is generated will be a leader, eight 0FFHs, binary data, eight 0FFHs, and a trailer. The 0FFHs are 'rubouts' and are called files ques. These are detected and counted to determine the start and the end of files.

EXAMPLE: U00,FF (cr, start reader)

will generate a binary tape, formated as discribed above, of the values contained in memory locations 00H to FFH.

V VERIFY. This command allows the user to verify the contents of one memory block agains the contents of another memory block. This is very useful for functions such as verifying that a file generated from a program is a duplicate of the actual program, etc.

EXAMPLE: V1000,2000,3000

will compare the contents of the memory block 1000H to 2000H against the contents of the memory block commencing at 3000H and extending to 4000H. Any differences will be displayed.

```
EXAMPLE:   V1000,2000,3000
            100F 00 FF
```

indicated that the contents of address 100FH is a 00 while that at 300FH is an FF.

W WRITE Hex file. This command dumps memory to the punch device in the standard 'Intel-style' hex file format. Both start and end of file parameters are required. The proper 'end of file' (EOF) is generated by the 'E' command.

```
EXAMPLE:   W00,FF(cr,start punch)
           (after punching)
           E(cr)
```

will generate a checksummed hex file of the values in the memory block 00H to FFH. If the assigned punch and console are the same, the program will pause and wait for the operator to turn on the punch (ASR33, etc.). Use of the 'N' command at either the beginning and/or end of the file is optional, but recommended.

X eXAMINE REGISTERS. The "X" command allows the user to examine and/or modify all of the Z80 registers.

A - Accumulator
B,C,D,E,H,L - CPU REGISTERS
M - Memory (pointed to by H & L)
P - Program Counter (PC)
S - Stack Pointer (SP)
I - Interrupt Register
X - Index (IX)
Y - Index (IY)
R - Refresh Register

EXAMPLE: X(cr)

displays the contents of MAIN registers A, B, C, D, E, F, H, L, M, P, S and I, in hex.

EXAMPLE X'(cr)

displays the contents of PRIME registers A, B, C, D, E, F, H, L, M, X, Y and R.

Typing the letter "X" (or X'), followed by a specific register letter will display the contents of that register. Entering a new value via the keyboard (kb) will substitute the new value in the specific register. Hitting the space bar will display the next register in which you may then perform substitutions, etc. In the unique case of the "M" register, you may modify the 16 bit pointer (H&L) to that memory location.

EXAMPLE;
```
XA 00-(kb)FF(cr)
XA FF-(sp)00-(kb)FF(cr)
XA FF-(sp)FF-(cr)
>
```

first examines the contents of register "A" (00H), then substitutes an FF. In the next line, the FF is displayed, a space character displays the next register (again a 00H), and substitutes an FF for this value. The last line displays both registers as containing FFHs.

Y SEARCH. This command allows unique byte strings, from one up to 255 bytes to be searched for in memory, and the addresses where they are found to be displayed. It is advisable to search for unique patterns rather than single bytes. The search operation may be stopped with a control-C.

```
EXAMPLE:    YC3,21,F3,01(cr)
            0081
            00B2
            0F08
            >
```

indicates that the byte string (in hex) C3, 21, F3, 01, is found in memory at locations 0081H, 00B2H and 0F08H. This routine will search all 65-K of memory for a unique sequence of bytes in less that one second.

Z Z TOP OF MEMORY. This command locates and gives the highest address of available memory in your system.

```
EXAMPLE:    Z
            7FFF
            >
```

indicates that the highest available memory is at address 7FFFH. Note that NO carriage return is required. Also, if only one 1K board were in the system, and it was addressed to have its top byte at address 7FFFH, the Z command would so indicate regardless of the absence of lower memory.

Additional Functions

The following functions are not part of the original Zapple Monitor. One of the functions,(K), cause a jump to the HOST I/O routine. The other two, (I and O), are used to program and verify PROMS using an auxillary PROM programming card.

K - JUMP TO HOST I/O.
EXAMPLE: K
will go to the HOST I/O routine and start execution.

I - VERIFY ROM. This command will verify that a 1K block of data has been correctly written into ROM by the 'O' function.
EXAMPLE: I1000
will compare the program in the ROM with the data in the block from 1000H to 1400H. Any differences will be displayed.
EXAMPLE: I1000
10F0 00FF
indicates that the contents of location F0H in the ROM is 00H, while that at 10F0H in memory is FFH.

O - PROGRAM ROM. This command will write data into a PROM from a 1K block of memory.
EXAMPLE: O1000
writes the data from 1000H to 1400H into the PROM.

NOTE - The above two commands (I and O) require a special card which contains a ROM programmer for 1K by 8 programmable ROMs.

Appendix C

RMC MESSAGE FORMATS

XMIT BLOCK

Q-FRAMES

SYN	SYN	SOH	EOT

(Implies NAK)

SERVICE MSG

LINK MSG	EOT

(Service Request in LINK MSG)

DATA MSG

LINK MSG	LOGICAL MSG #1	----	LOGICAL MSG #n	EOT

$\leftarrow$ Total length $\leq$ 1024 Char $\rightarrow$

($1 \leq n \leq$ # Devices)

LINK MSC

SYN	SYN	SYN	SYN	SOH	FC	SC	AC	OC	IC	STX	ETX	BCC

$\leftarrow$ Header $\rightarrow$ (SOH through IC)

(BCC = Block)
(Check)
(Char)

LINK MSG INFORMATION

SYN - All XMIT BLOCKS must start with 2 or more SYN characters. SYN characters may also appear anywhere within the XMIT BLOCK but are disregarded.

SOH - Start Of Header character.

FC	- Format Code	110_8	Transmission ACK/NAK.
		102_8	Service message (RFD, DIS).
SC	- Sequence Code	101_8,102_8	Alternates on each new XMIT BLOCK.
AC	- Address Code	100_8	Alway this value in single RNP networks

OC	- Operation Code	$1xy_8$	x= 0 ACK 1 NAK y= 0 No Instruction 3 A Call (accept all calls) 4 Ready for Disconnect (RFD) 5 N Call (accept no calls) 6 Disconnect (DIS) 7 Reserved
IC	- Identification Code	$1xx_8$	xx= # of messages in XMIT BLOCK. xx≤ 63
STX	- Start Of Text character		Text may follow this character, but not normally found in LINK.
ETX	- End Of Text character		Follows text, if present.
BCC	- Block Check Character		X-OR of all characters from SOH to ETX not including SOH or any SYN characters in msg.

LOGICAL MSG

SOH	FC	SC	AC	C1	C2	IC	STX	TEXT 0-232 Char	ETX	BCC

|← Header →| (SOH through IC)

LOGICAL MSG INFORMATION

SOH	- Start of Header		
FC	- Format Code	Bits 5-6=1	Bits 0-4 Indicate Mode
SC	- Sequence Code	101_8, 102_8	Set but not checked.
AC	- Address Code	$1xx_8$	xx= Destination Device #
OC1	- Operation Code 1	101_8	Indicates OC2 to be used.

OC2 - Operation Code 2	$1xy_8$	x= 0	ACK
		1	Break ACK
		2	DIS ACK
		7	NULL
		y= 0	No Request
		1	Break
		2	DIS
		3	SELECT (Connect Term)
		4	Bad Parity
		5	Logical DIS

IC - Identification Code

101_8	Remote Computer	
110_8	TTY	110 baud
111_8		150 baud
112_8		300 baud
130_8	VIP	

STX - Start Of Text — Text of logical message follows.

TEXT- This area may contain 0-232 ASCII characters excluding the following special characters:
(SOH, ETX, ETB, STX, ACK, NAK, ENQ, US, DLE, EOT)

ETX - End Of Text — Follows text, can also be ETB character for messages which are longer than 232 char.

BCC - Block Check Character — Same as for LINK MSG including all text characters.

```
;I/O PROGRAM FOR MODEM TO Z80 SYSTEM
;
ZEE:      EQU  0800H          ;MAIN MONITOR PROGRAM
TRAP:     EQU  081EH          ;TRAP RETURN ADDR
LFADR:    EQU  0C70H          ;PRINT CRLF & HL
EXPR1:    EQU  0D33H          ;GETS 2 BYTE PARAMETER
IOSTS:    EQU  101BH         ;I/O STATUS BYTE
LOD0:     EQU  0A28H         ;TAPE READ ROUTINE
CO:       EQU  0C78H         ;CONSOLE OUT ROUTINE
CI:       EQU  0E13H         ;CONSOLE IN ROUTINE
FILE:     EQU  1020H         ;FILE NAME BUFFER
TOM1:     EQU  0C40H         ;MESSAGE PRINT ROUTINE
CRLF:     EQU  0D04H         ;CR & LF TO CONSOLE
;
ENTER:    CALL INIT          ;RESET I/O
          LD   A,0DH         ;CR
          LD   (FILE+0AH),A  ;PUT AT END OF FILE BUFF
          CALL CRLF          ;CR & LF
          LD   B,05H         ;CHAR COUNT
          LD   HL,MSG0       ;PROGRAM NAME
          CALL TOM1          ;PRINT IT
;
START:    LD   C,'+'         ;PROMPT CHAR
          CALL CO            ;PRINT IT
INCO:     IN   A,(11H)       ;READ CONSOLE STATUS
          AND  02            ;INPUT DATA
          JR   Z,INM-$       ;NO, CHECK MODEM
          CALL CI            ;YES, READ CHAR
;
          CP   1BH           ;= ESC
          JR   Z,CONBRK-$    ;YES, SEND BREAK
          CP   02H           ;= 'CNTL-B'
          JR   Z,HOSTLD-$    ;YES, LOAD FROM MODEM
          CP   03H           ;= 'CNTL-C'
          JR   Z,INM-$       ;IGNORE IT...
          CP   0EH           ;= 'CNTL-N'
          RET  Z            ;GO TO MONITOR
;
OUTM:     CALL OUTMOD        ;OUTPUT TO MODEM
;
INM:      IN   A,(15H)       ;READ MODEM STATUS
          AND  02            ;INPUT DATA RDY
          JR   Z,INCO-$      ;NO, CHECK CONSOLE
          CALL INMOD         ;YES, READ CHAR
          LD   C,A           ;
          CALL CO            ;PRINT ON CONSOLE
          JR   INCO-$
;
INMOD:    IN   A,(15H)       ;READ MODEM STATUS
          AND  02            ;INPUT DATA RDY
          JR   Z,INMOD-$     ;NO, WAIT
          IN   A,(14H)       ;YES, READ CHAR
          AND  7FH           ;MASK PARITY
          RET
```

```
;
OUTMOD:   PUSH AF             ;SAVE A
          IN   A,(15H)        ;READ MODEM STATUS
          AND  01             ;XMIT RDY
          JR   Z,OUTMOD+1-$   ;NO, WAIT
          POP  AF             ;YES, GET DATA
          OUT  (14H),A        ;SEND DATA
          RET
;
CONBRK:   CALL BREAK          ;SEND BREAK TO MODEM
          JR   START-$
;
BREAK:    LD   A,3DH          ;LOAD BREAK CODE
          OUT  (15H),A        ;OUTPUT TO MODEM USART (8251)
          LD   A,200          ;WAIT COUNT (200 MS)
          CALL WAIT           ;WAIT...
          LD   A,35H          ;LOAD NORMAL OPERATION CODE
          OUT  (15H),A        ;OUTPUT TO USART
          LD   C,'<'          ;PRINT < ON CONSOLE
          JP   CO             ;
;
WAIT:     LD   B,090H         ;DELAY COUNT
          DJNZ WAIT+2-$       ;WAIT 1 MS
          DEC  A              ;DEC WAIT COUNT, = 0
          JR   NZ,WAIT-$      ;NO, DELAY AGAIN
          RET                 ;YES, RETURN
;
RDMOD:    CALL INMOD          ;READ CHAR
          PUSH AF             ;SAVE AF
          PUSH BC             ;SAVE BC
          LD   C,A            ;
          CALL CO             ;PRINT ON CONSOLE
          POP  BC             ;RESTORE BC
          POP  AF             ;RESTORE AF
          OR   A              ;CLEAR CARRY
          RET
;
HOSTLD:   LD   C,'>'          ;PROMPT CHAR
          CALL CO             ;PRINT ON CONSOLE
          CALL EXPR1          ;GET BIAS, IF ANY
          LD   A,B            ;LOOK AT DELIMITER
          SUB  0DH            ;IF = CR
          LD   B,A            ;RELOCATION = 0
          LD   C,A            ;
          POP  DE             ;DE=BIAS
          JR   Z,R0-$         ;CR ENTERED
          CALL EXPR1          ;GET RELOCATION
          POP  BC             ;BC=RELOCATION
R0:       EX   DE,HL          ;
          EXX                 ;HL'=BIAS, BC'=RELOCATION
          CALL CRLF           ;
;
          LD   B,0AH          ;CHAR COUNT
          LD   HL,MSG2        ;LOAD POINTER
```

```
          CALL TOM1          ;PRINT MSG2
          LD   HL,FILE       ;FILE NAME BUFFER
          LD   B,0AH         ;MAX LENGTH
H1:       CALL CI            ;READ NAME
          LD   (HL),A        ;PUT IN BUF
          CP   0DH           ;CARRIAGE RET
          JR   Z,H2-$        ;YES, LAST CHAR
          CP   1BH           ;= ESC
          JR   Z,CONBRK-$    ;YES, SEND BREAK
          INC  HL            ;INC POINTER
          DJNZ H1-$          ;CHECK MAX LNGTH
H2:       CALL CRLF          ;CRLF TO CONSOLE
          LD   B,5           ;CHAR COUNT
          LD   HL,MSG1       ;LOAD POINTER
          CALL MOUT          ;OUTPUT MESSAGE TO MODEM
          LD   B,0BH         ;MAX FILE LENGTH
          LD   HL,FILE       ;FILE NAME POINTER
          CALL MOUT          ;OUTPUT TO MODEM
;
          LD   A,(IOSTS)     ;GET I/O STATUS
          PUSH AF            ;SAVE OLD STATUS
          AND  0F3H          ;CLEAR READER STATUS
          OR   04H           ;SET NEW STATUS
          LD   (IOSTS),A     ;PUT IN IOBYT
          LD   A,0C3H        ;JUMP CODE
          LD   (1006H),A     ;PLACE IN USER ROUTINE AREA
          LD   HL,RDMOD      ;MODEM I/O DRIVER
          LD   (1007H),HL    ;PLACE IN USER ROUTINE AREA
          CALL LOD0          ;GO READ TAPE
          POP  AF            ;RECALL I/O STATUS
          LD   (IOSTS),A     ;RESTORE OLD STATUS
          JP   START
;
MOUT:     LD   A,(HL)        ;GET CHAR FROM BUFFER
          CALL OUTMOD        ;OUTPUT TO MODEM
          LD   C,A           ;
          CALL CO            ;PRINT IT
          INC  HL            ;INC POINTER
          CP   0DH           ;= CARRIAGE RET
          JR   Z,M01-$       ;NO, CHECK COUNT
          DJNZ MOUT-$        ;= MAX COUNT
M01:      RET
;
MSG0:     DEFM 'ZIO'
          DEFW 0A0DH         ;CR & LF
MSG1:     DEFM 'LIST '       ;MESSAGE TO HIS
MSG2:     DEFM 'FILE'
          DEFM ' NAM'
          DEFM 'E-'
;
;         SET JP I/O CHANNELS
;
INIT:     XOR  A             ;CLEAR ACC
          LD   BC,4011H      ;RESET USART, I/O CHAN(10,11)
```

```
        LD    DE,0FA35H       ;SET MODE & FUNCTIONS
        CALL  USET            ;OUTPUT COMMANDS
        LD    C,15H           ;I/O CHAN(14,15)
        CALL  USET            ;OUTPUT COMMANDS
        LD    C,17H           ;I/O CHAN(16,17)
        CALL  USET            ;OUTPUT COMMANDS
        RET                   ;
;
USET:   OUT   (C),A           ;CLEAR COMMAND REG
        OUT   (C),B           ;RESET USART
        OUT   (C),D           ;SEND MODE COMMAND
        OUT   (C),E           ;SEND FUNCTION COMMAND
        RET                   ;
;
        END   0000H
```

APPENDIX E
RNP PROGRAM LISTINGS

```
;  FILE MAIN
;
;         MAIN RNP SERVICE ROUTINE
;*************************************************************
;
START:    CALL INIT              ;INITALIZE SYSTEM
;
MAIN:     LD   A,(HCB)           ;GET RNP STATUS
          BIT  ONLN,A            ;RNP ON LINE
          JR   Z,CHKRQ-$         ;NO, CHECK RQ FOR OUTPUT
;
          CALL RBFSRV            ;YES, SERVICE RBUF
;
CHKRQ:    LD   IX,RQUE           ;IX=RQUE
          CALL GET               ;GET ADDR OFF RQ
          JR   C,CHKEQ-$         ;IF EMTY, CHECK EQ
          CALL DSTDCB            ;HL=DST DCB, IY=BUF ADDR
          CALL DEVOUT            ;START BUF OUTPUT
          JR   NC,CHKEQ-$        ;DCB(BSY)=1 NO, CHECK EQ
;
          PUSH IY                ;YES, MOVE BUF ADDR
          POP  HL                ;TO HL
          CALL PUT               ;PUT ON RQ
;
CHKEQ:    LD   IX,EQUE           ;IX=EQUE
          CALL GET               ;GET ADDR OFF EQ
          JR   C,CHKWQ-$         ;IF EMTY, CHECK WQ
          POP  IX                ;IX=QUEUE ADDR, HL=BUF ADDR
          CALL PUT               ;PUT BUF ON Q
          JR   NC,CHKWQ-$        ;IF OK, CHECK WQ
          CALL FULERR            ;ELSE PUT BACK ON EQ
;
CHKWQ:    LD   IX,WQUE           ;IX=WQUE
          CALL GET               ;GET ADDR OFF WQ
          JR   C,MAIN-$          ;IF WQ EMTY, GO TO MAIN
          CALL DSTDCB            ;HL=DST DCB, IY=BUF ADDR
          BIT  ACK,(HL)          ;DEV(ACK)=1
          PUSH IY                ;MOVE BUF ADDR
          POP  HL                ;TO HL
          JR   Z,RSWQ-$          ;NO, PUTBACK ON WQ
          LD   IX,RQUE           ;YES, PUT ON RQ
RSWQ:     CALL PUT               ;PUT ON Q
          JR   NC,MAIN-$         ;NOT FULL, BACK TO MAIN
          LD   IX,WQUE           ;FULL, PUT BACK ON WQ
          JR   RSRQ-$            ;
;
;  FILE RBFSRV
;         HOST RECEIVE BUFFER SERVICE ROUTINE
;*************************************************************
;
;         ENTER-
;         EXIT-  RBUF SERVICED, IF ERROR-CARRY SET
;
RBFSRV:   LD   HL,RBUF           ;CHECK NEXT RBUF
```

```
         BIT   BSY,(HL)         ;RBUF(BSY)=1
         RET   NZ               ;YES, RETURN
         BIT   FULL,(HL)        ;NO, RBUF(FULL)=1
         JR    Z,RBMT-$         ;NO, CHECK RBUF(EMTY)
;
         CALL  STRLNK           ;YES, STRIP OFF LINK MSG
         BIT   ERR,(HL)         ;RBUF(ERR)=1
         JR    Z,CKSV-$         ;NO, CHECK SERV MSG
;
RERR:    LD    IX,XBUF          ;IX=XBUF
         BIT   ACTV,(IX+0)      ;XBUF(ACTV)=1
         JR    NZ,CKNAK-$       ;YES, GO CHECK NAK
         LD    IX,SBUF          ;NO, IX=SBUF
         BIT   ACTV,(IX+0)      ;SBUF(ACTV)=1
         JR    Z,NKMG-$         ;NO, SEND NAK MSG
;
CKNAK:   BIT   3,(IX+0CH)       ;OC(NAK)=1
         JR    NZ,STRBF-$       ;YES, GO START BUF
         SET   3,(IX+0CH)       ;NO, SET OC(NAK)=1
         LD    A,08H            ;RECALC BCC
         XOR   (IX+10H)         ;
;
STRBF:   PUSH  HL               ;SAVE RBUF ADDR
         PUSH  IX               ;MOVE BUF ADDR TO HL
         POP   HL               ;
         CALL  STRXBF           ;REXMIT OLD XBUF
         POP   HL               ;RESTORE RBUF ADDR
         JR    SRBF-$           ;START NEXT RBUF
;
NKMG:    CALL  NAKMSG           ;SEND NAK MSG
         JR    SRBF-$           ;START NEXT RBUF
;
CKSV:    BIT   SVM,(HL)         ;RBUF(SVM)=1
         JR    NZ,SRMG-$        ;YES, SEND SERVICE MSG
         BIT   LOG,(HL)         ;RBUF(LOG)=1
         JR    NZ,CKSC-$        ;YES, CHECK SC
         BIT   ACK,(HL)         ;NO, RBUF(ACK)=1
         JR    Z,RXMT-$         ;NO, REXMIT OLD XBUF
         PUSH  HL               ;SAVE HL
         CALL  GENXBF           ;YES, START NEW XBUF
         POP   HL               ;RESTORE HL
         JR    SRBF-$           ;START NEXT RBUF
;
RBMT:    BIT   EMTY,(HL)        ;RBUF(EMTY)=1
         JR    NZ,SRBF-$        ;YES, START NEXT RBUF
         JR    NXMSG-$          ;NO, CONTINUE PROCESSING RBUF
;
SRMG:    CALL  SRVMSG           ;XMIT SERVICE MSG
         JR    C,RERR-$         ;IF CARRY, ERROR
         JR    SRBF-$           ;GO START NEXT RBUF
;
CKSC:    BIT   NSC,(HL)         ;RBUF(NSC)=1
         JR    NZ,NXMSG-$       ;YES, PROCESS RBUF
         BIT   ACK,(HL)         ;RBUF(ACK)=1
```

```
        JR    NZ,AKMG-$       ;YES, SEND ACK MSG
RXMT:   CALL  STRXBF          ;REXMIT OLD XBUF
        JR    SRBF-$          ;GO START NEXT RBUF
;
NXMSG:  PUSH  HL              ;(SP)=RBUF
NXM:    CALL  GETMSG          ;GET NEXT MSG
        JR    NZ,RERR-$       ;IF NZ, ERROR
        JR    C,PTRQ-$        ;BUF SAVED, IF CARRY
        CALL  GETBUF          ;GET NEXT AVAILABLE BUF
        JR    C,SVMG-$        ;IF EMPTY, SAVE MSG
PTRQ:   LD    IX,RQUE         ;IX=RQUE
        CALL  PUT             ;PUT ADDR ON RQ
        JR    C,SVMG-$        ;IF FULL, SAVE MSG
        CALL  CONMSG          ;CONVERT TO OBUF
        JR    NC,NXM-$        ;IF NOT 'EOT', NEXT MSG
;
AKMG:   CALL  ACKMSG          ;SEND ACK MSG
;
        POP   HL              ;HL=RBUF
        RES   FULL,(HL)       ;RBUF(FULL)=0
        SET   EMTY,(HL)       ;RBUF(EMTY)=1
;
SRBF:   CALL  STRBUF          ;START NEXT RBUF
        RES   ACK,(HL)        ;RBUF(ACK)=0
        SET   BSY,(HL)        ;RBUF(BSY)=1
        RET                   ;
;
SVMG:   INC   C               ;SET SAVE FLAG
        LD    (SAVMSG),BC     ;SAVE LENGTH & FLAG
        LD    (SAVMSG+2),DE   ;SAVE MSG ADDR
        LD    (SAVMSG+4),HL   ;SAVE BUF ADDR
        POP   HL              ;HL=RBUF
        RES   FULL,(HL)       ;RBUF(FULL)=0
        RES   EMTY,(HL)       ;RBUF(EMTY)=0
        RET                   ;
;
;;; FILE STRLNK
;
;       ROUTINE TO STRIP LINK MSG AND SET RBUF FLAGS
;*************************************************************
;       ENTRY- HL=RBUF ADDR
;       EXIT- STATUS FLAGS IN RBUF SET AS INDICATED
;       IN LINK MSG
;
STRLNK: PUSH  HL              ;MOVE RBUF TO IX
        POP   IX              ;
        LD    (IX+0),00H      ;CLEAR ALL FLAGS
        LD    HL,RBUF+04H     ;SET UP BUFFER POINTER
        LD    A,(HL)          ;GET CHAR
        CP    01H             ;=SOH
        JP    NZ,LNKERR       ;NO, ERROR
        INC   HL              ;YES, CHECK FC
        LD    A,(HL)          ;NEXT CHAR
        CP    48H             ;=ACK\NAK MSG
```

```
        JR    Z,CHKSC-$        ;YES, CHECK SC
        CP    42H              ;NO, =SRVMSG
        JR    NZ,LNKERR-$      ;NO, ERROR
        SET   SVM,(IX+0)       ;YES, RBUF(SVM)=1
CHKSC:  INC   HL               ;CHECK SC
        LD    C,A              ;START BCC
        LD    A,(HL)           ;NEXT CHAR
        CP    41H              ;=41H
        JR    Z,CHKLSC-$       ;YES, CHECK LAST SC
        CP    42H              ;NO, =42H
        JR    NZ,LNKERR-$      ;NO, ERROR
CHKLSC: LD    B,A              ;B=SC
        LD    A,(SAVSC)        ;GET LAST SC
        CP    B                ;=LAST SC
        JR    Z,CHKAC-$        ;YES, CHECK AC
        LD    A,B              ;GET NEW SC
        LD    (SAVSC),A        ;SAVE NEW SC
        SET   NSC,(IX+0)       ;RBUF(NSC)=1
CHKAC:  INC   HL               ;CHECK AC
        XOR   C                ;CALC BCC
        LD    C,A              ;SAVE BCC
        LD    A,(HL)           ;NEXT CHAR
        CP    40H              ;=40H
        JR    NZ,LNKERR-$      ;NO, ERROR
        INC   HL               ;YES, CHECK OC
        XOR   C                ;CALC BCC
        LD    C,A              ;SAVE BCC
        LD    A,(HL)           ;NEXT CHAR
        AND   70H              ;CLEAR LOWER BYTE
        CP    40H              ;=40H
        JR    NZ,LNKERR-$      ;NO, ERROR
        LD    A,(HL)           ;YES, GET OC
        BIT   3,A              ;NAK BIT SET
        JR    NZ,SVOC-$        ;YES, SAVE OC
        SET   ACK,(IX+0)       ;NO, RBUF(ACK)=1
SVOC:   LD    (SAVOC),A        ;SAVE PRESENT OC
        INC   HL               ;CHECK IC
        XOR   C                ;CALC BCC
        LD    C,A              ;SAVE BCC
        LD    A,(HL)           ;NEXT CHAR
        BIT   6,A              ;BIT 6 SET
        JR    Z,LNKERR-$       ;NO, ERROR
        AND   3FH              ;CLEAR MSB
        LD    (SAVIC),A        ;SAVE IC
        INC   HL               ;CHECK STX
        XOR   C                ;CALC BCC
        LD    C,A              ;SAVE BCC
        LD    A,(HL)           ;NEXT CHAR
        CP    02H              ;=STX
        JR    NZ,LNKERR-$      ;NO, ERROR
        LD    B,00H            ;CLEAR B
CHKETX: DEC   B                ;B=B-1
        JR    Z,LNKERR-$       ;B>256, NO ETX FOUND
        INC   HL               ;FIND ETX
```

```
        XOR   C              ;CALC BCC
        LD    C,A            ;SAVE BCC
        LD    A,(HL)         ;NEXT CHAR
        CP    03H            ;=ETX
        JR    NZ,CHKETX-$    ;NO, CHECK NEXT CHAR
        INC   HL             ;YES, CHECK BCC
        XOR   C              ;CALC BCC
        LD    C,A            ;SAVE BCC
        LD    A,(HL)         ;NEXT CHAR
        CP    C              ;C=BCC
        JR    NZ,LNKERR-$    ;NO, ERROR
        INC   HL             ;YES, CHECK FOR EOT
        LD    A,(HL)         ;NEXT CHAR
        CP    04H            ;=EOT
        JR    NZ,LNKRET-$    ;YES, RETURN
        SET   LOG,(IX+0)     ;NO, RBUF(LOG)=1
        LD    (SAVMSG+2),HL  ;SAVE LOCATION POINTER
        LD    HL,0000H       ;
        LD    (SAVMSG),HL    ;COUNT=00,SAVE FLAG=0
        JR    LNKRET-$       ;RETURN
LNKERR: SET   ERR,(IX+0)     ;SET ERR BIT
LNKRET: PUSH  IX             ;MOVE RBUF TO HL
        POP   HL             ;
        RET                  ;RETURN
;;;  FILE SRVMSG
;
;       ROUTINE TO ANALYZE SERVICE MSG
;       ENTRY- SAVOC=PRESENT VALUE OF OC
;       EXIT- DECODE OC SERVICE MSG, ON ERROR,
;       SET CARRY & RETURN, ELSE, TAKE INDICATED ACTION
;
SRVMSG: LD    A,(SAVOC)      ;GET PRESENT OC
        AND   07H            ;CLEAR UPPER 5 BITS
        JR    Z,NOINST-$     ;=0, NO INSTRUCTION
        SUB   03H            ;
        JR    Z,ACALL-$      ;=3, ACCEPT ALL CALLS
        DEC   A              ;
        JR    Z,RFD-$        ;=4, READY FOR DISCONNECT
        DEC   A              ;
        JR    Z,NCALL-$      ;=5, ACCEPT NO CALLS
        DEC   A              ;
        JR    Z,DIS-$        ;=6, DISCONNECT
;
        SCF                  ;ERROR, SET CARRY
        RET                  ;RETURN
;
NOINST: JP    ACKMSG         ;SEND ACK MSG
;
ACALL:  LD    HL,HCB         ;GET RNP STATUS ADDR
        SET   ACPT,(HL)      ;HCB(ACPT)=1
        JP    ACKMSG         ;SEND ACK MSG
;
RFD:    LD    A,44H          ;A=OC(ACK,RFD)
        LD    (SBOC),A       ;PUT IN SRVMSG BUFFER
```

```
          LD    A,(SBCC)        ;GET BCC
          XOR   04H             ;CALC NEW BCC
          LD    (SBCC),A        ;PUT IN SBUF
          JP    STRSBF          ;SEND SERVICE MSG
;
NCALL:    LD    HL,HCB          ;GET RNP STATUS ADDR
          RES   ACPT,(HL)       ;HCB(ACPT)=0
          JP    ACKMSG          ;SEND ACK MSG
;
DIS:      LD    HL,HCB          ;GET RNP STATUS ADDR
          RES   ONLN,(HL)       ;HCB(ONLN)=0
          RET                   ;
;;;  FILE GETMSG
;
;         ROUTINE TO GET NEXT MSG OFF RBUF
;*************************************************************
;         ENTRY- SAVMSG+2=MSG ADDR, SAVMSG+4=SAVED BUF
;         EXIT- B=BYTE COUNT OF MSG, DE=MSG ADDR,
;         HL=BUF ADDR, CARRY IF BUFFER SAVED, NZ IF ERROR
;
GETMSG:   LD    A,(SAVMSG)      ;GET STATUS
          OR    A               ;CLEAR FLAGS
          JR    NZ,SAVED-$      ;IF A>0, DATA SAVED
          LD    A,03H           ;NO, GET BYTE COUNT
          LD    HL,(SAVMSG+2)   ;HL=START OF MSG
          LD    BC,00FFH        ;NO, SET BC=255
          CPIR                  ;'ETX' FOUND
          RET   NZ              ;NO, ERROR-RET WITH NZ
;
;         CALC BYTE COUNT
          LD    HL,00FFH        ;HL=255
          SBC   HL,BC           ;HL=BYTE CNT
          LD    B,L             ;B=BYTE CNT
          LD    DE,(SAVMSG+2)   ;DE=MSG ADDR
          XOR   A               ;SET ZERO
          SCF                   ;SET CARRY
          RET                   ;
;
SAVED:    SCF                   ;SET CARRY
          DEC   A               ;STATUS>1
          JR    Z,NOBUF-$       ;NO, BUF NOT SAVED
          XOR   A               ;CLEAR CARRY & SET Z
          LD    HL,(SAVMSG+4)   ;YES, GET BUF ADDR
;
NOBUF:    LD    DE,(SAVMSG+2)   ;GET MSG ADDR
          RET                   ;
;
;
;         ROUTINE TO GET 1-4 LINKED BUFFERS OFF AQUE
;*************************************************************
;         ENTRY- B=BYTE COUNT
;         EXIT- HL=BUF ADDR OF FIRST BUF IN LINKED SET
;         LINK BIT SET IN ALL BUT LAST BUFFER AND
;         POINTER TO NEXT BUF IN LAST 2 BYTES
```

```
;
GETBUF:   PUSH BC              ;SAVE COUNT
          PUSH DE              ;SAVE MSG ADDR
          LD   A,B             ;A=BYTE COUNT
          LD   B,00H           ;B=0
          LD   IX,AQUE         ;IX=AQUE
NUMBUF:   INC  B               ;B=B+1
          CALL GET             ;GET BUF OFF AQ
          JR   C,RSAQ-$        ;IF EMPTY, RESTORE OTHER BUFS
          PUSH HL              ;SAVE BUF ADDR
          SUB  3AH             ;A=A-58, A<=0
          JR   NC,NUMBUF-$     ;NO, NEXT BUF
;
          POP  HL              ;GET BUF ADDR
          LD   (HL),00H        ;CLEAR STATUS
          DEC  B               ;B=B-1, B=0
          JR   Z,LSTBUF-$      ;YES, ONLY BUF
;
LNKBUF:   POP  IX              ;NO, LINK BUFFERS
          LD   (IX+62),L       ;LOWER BYTE NEXT BUF ADDR
          LD   (IX+63),H       ;UPPER BYTE NEXT BUF ADDR
          PUSH IX              ;MOVE IX TO HL
          POP  HL              ;
          SET  LNK,(HL)        ;LINKED TO NEXT BUF
          DJNZ LNKBUF-$        ;LAST BUF
;
LSTBUF:   XOR  A               ;CLEAR CARRY
          POP  DE              ;RESTORE MSG ADDR
          POP  BC              ;RESTORE COUNT
          RET                  ;
;
RSAQ:     LD   IX,AQUE         ;LOAD QUEUE ADDR
RSQ:      SCF                  ;SET CARRY
          DEC  B               ;B=B-1, B=0
          JR   Z,LSTBUF+1-$    ;YES, RETURN
          POP  HL              ;GET BUF ADDR
          CALL PUT             ;PUT BUF BACK
          JR   RSQ-$           ;NEXT BUF
;;;  FILE CONMSG
;
;         ROUTINE TO CONVERT LOG MSG'S FROM RBUF TO OBUF'S
;************************************************************
;
;         ENTER- HL=OBUF, DE=RBUF MSG ADDR,
;                (SAVMSG+1)=BYTE COUNT
;         EXIT-  CARRY SET IF 'EOT'
;
CONMSG:   PUSH HL              ;SAVE OBUF POINTER
          EX   DE,HL           ;DE=OBUF, HL=MSG
          LD   A,(HL)          ;GET CHAR
          CP   01H             ;='SOH'
          JP   NZ,LOGERR2      ;NO, ERROR
          INC  HL              ;
          LD   A,(HL)          ;GET FC
```

```
        CP    60H             ;=FC
        JP    NZ,LOGERR2      ;NO, ERROR
        LD    C,A             ;START BCC
        INC   HL              ;GET SC
        LD    A,(HL)          ;NEXT BYTE
        XOR   C               ;CALC BCC
        LD    C,A             ;
        INC   HL              ;GET AC
        LD    A,(HL)          ;NEXT BYTE
        PUSH  AF              ;SAVE AC
        XOR   C               ;CALC BCC
        LD    C,A             ;
        INC   HL              ;GET OC1
        LD    A,(HL)          ;NEXT BYTE
        CP    41H             ;=OC1
        JR    NZ,LOGERR1-$    ;NO, ERROR
        XOR   C               ;CALC BCC
        LD    C,A             ;
        INC   HL              ;GET OC2
        LD    A,(HL)          ;NEXT BYTE
        PUSH  AF              ;SAVE OC2
        XOR   C               ;CALC BCC
        LD    C,A             ;
        INC   HL              ;GET IC
        LD    A,(HL)          ;NEXT BYTE
        CP    48H             ;=IC
        JR    NZ,LOGERR-$     ;NO, ERROR
        XOR   C               ;CALC BCC
        LD    C,A             ;
        INC   HL              ;GET STX
        LD    A,(HL)          ;NEXT BYTE
        CP    02H             ;=STX
        JR    NZ,LOGERR-$     ;NO, ERROR
        XOR   C               ;CALC BCC
;
        INC   HL              ;HL=POINTER TO TEXT
        LD    (SAVMSG+2),HL   ;SAVE RBUF POINTER
        LD    (SAVBCC),A      ;SAVE BCC
;
        POP   HL              ;HL=OC2
        POP   AF              ;A=AC (LU#)
        PUSH  HL              ;SAVE OC2
        CALL  DEVDCB          ;GET DCB ADDR (A=DEV#, HL=DCB)
        POP   IX              ;IX=DCB
        POP   AF              ;GET OC2
        LD    (IX+4),A        ;PUT OC2 IN DCB
        EX    DE,HL           ;HL=OBUF, DE=DCB
;
        LD    A,(SAVMSG+1)    ;A=BYTE COUNT
        INC   HL              ;
        LD    (HL),A          ;PUT IN OBUF
        INC   HL              ;
        LD    (HL),E          ;PUT DCB IN OBUF
        INC   HL              ;
```

```
          LD     (HL),D             ;
          INC    HL                 ;HL=OBUF DATA POINTER
          EX     DE,HL              ;DE=OBUF DATA POINTER
          LD     HL,(SAVMSG+2)      ;HL=RBUF MSG ADDR
;
PUTMSG:   LD     A,(SAVMSG+1)       ;A=BYTE COUNT
          LD     C,A                ;
          SUB    3AH                ;A=A-58
          JP     M,LTBF             ;IF A>0, MOVE 58 BYTES
          JR     Z,NOBF-$           ;IF A=0 NO MORE DATA
;
          LD     (SAVMSG+1),A       ;SAVE REMAINING BYTE COUNT
          LD     BC,003AH           ;SET BC=58 BYTES
          CALL   MOVMSG             ;MOVE BLOCK OF DATA
;
          EX     DE,HL              ;HL=OBUF POINTER
          LD     E,(HL)             ;DE=LINKED BUF ADDR
          INC    HL                 ;
          LD     D,(HL)             ;
          LD     HL,(SAVMSG+2)      ;HL=RBUF POINTER
          INC    DE                 ;
          INC    DE                 ;
          INC    DE                 ;
          INC    DE                 ;DE=OBUF DATA POINTER
          JR     PUTMSG-$           ;PUT DATA IN OBUF
;
LTBF:     LD     B,00               ;NO, MOVE REMAINING BYTES
          CALL   MOVMSG             ;MOVE BLOCK OF DATA
NOBF:     LD     A,(SAVBCC)         ;GET SAVED BCC
          LD     C,(HL)             ;GET MSG BCC
          CP     C                  ;A=C
          JR     NZ,LOGERR2-$       ;NO, ERROR
;
          XOR    A                  ;A=0
          LD     (SAVMSG),A         ;CLEAR STATUS
          INC    HL                 ;
          LD     A,(HL)             ;GET NEXT BYTE
          CP     04H                ;='EOT'
          SCF                       ;SET CARRY
          RET    Z                  ;YES, RETURN WITH CARRY SET
          XOR    A                  ;CLEAR CARRY
          POP    HL                 ;
          RET                       ;
;
LOGERR:   POP    HL                 ;
LOGERR1:  POP    HL                 ;
LOGERR2:  POP    HL                 ;
          SET    ERR,(HL)           ;SET OBUF(ERR)=1
          XOR    A                  ;CLEAR CARRY
          RET                       ;
;
MOVMSG:   LD     A,(SAVBCC)         ;GET BCC
MVMG:     XOR    (HL)               ;CALC BCC
          LDI                       ;MOVE BYTE
```

```
          JP    PE,MVMG         ;IF BC NOT 0, NEXT BYTE
          LD    (SAVBCC),A      ;SAVE BCC
          LD    (SAVMSG+2),HL   ;SAVE RBUF MSG POINTER
          RET                   ;
;
;;;  FILE GENXBF
;
;         ROUTINE TO GENERATE NEXT XBUF
;***************************************************************
;
GENXBF:   LD    HL,XBUF         ;HL=XBUF
          BIT   BSY,(HL)        ;XBUF(BSY)=1
          RET   NZ              ;YES, RETURN
          BIT   FULL,(HL)       ;NO, XBUF(FULL)=1
          JR    NZ,STXB-$       ;YES, START XMIT OF XBUF
          BIT   EMTY,(HL)       ;NO, XBUF(EMTY)=1
          JR    Z,GNMG-$        ;NO, NEXT LOG MSG
;
          CALL  GENLNK          ;GENERATE LINK MSG
          RES   EMTY,(HL)       ;XBUF(EMTY)=0
;
GNMG:     PUSH  HL              ;(SP)=XBUF
GNI:      LD    IX,XQUE         ;IX=XQUE
          CALL  GET             ;GET NEXT IBUF OFF XQ
          JR    C,RTXB-$        ;IF XQ EMPTY, RETURN
          CALL  CONLOG          ;CONVERT IBUF TO LOG MSG
          JR    NC,GNI-$        ;IF XBUF NOT FULL, GET NEXT IBUF
;
          CALL  PUT             ;PUT BACK ON XQ
          POP   HL              ;HL=XBUF
          SET   FULL,(HL)       ;XBUF(FULL)=1
;
STXB:     LD    A,(RBUF)        ;A=RBUF STATUS
          BIT   BSY,A           ;RBUF(BSY)=1
          RET   NZ              ;YES, RETURN
          CALL  STRXBF          ;NO, START XMIT XBUF
          SET   BSY,(HL)        ;XBUF(BSY)=1
          RET                   ;
;
RTXB:     POP   HL              ;RESTORE HL
          RET                   ;
;
;         ROUTINE TO GENERATE LINK MSG IN XBUF
;***************************************************************
;
;         ENTRY- HL=XBUF
;         EXIT- LINK SET UP IN XBUF, (SAVLOC)=END OF LINK
;
GENLNK:   PUSH  HL              ;SAVE BUF ADDR
          LD    HL,XBUF         ;CHECK XBUF STATUS
          BIT   ACTV,(HL)       ;XBUF(ACTV)=1
          JR    NZ,GXSC-$       ;YES, GENERATE NSC
;
          LD    HL,SBUF         ;NO, CHECK SBUF STATUS
```

```
        BIT   ACTV,(HL)      ;SBUF(ACTV)=1
        JR    Z,NACTV-$      ;NO, SEND ACK MSG
        LD    A,(SBSC)       ;YES, GET OLD SC
        RES   ACTV,(HL)      ;SET SBUF(ACTV)=0
        JR    GNSC-$         ;GEN NSC
;
NACTV:  LD    A,42H          ;START NSC
        JR    GNSC-$         ;
;
GXSC:   LD    A,(XBSC)       ;GET OLD SC
        RES   ACTV,(HL)      ;SET XBUF(ACTV)=0
;
GNSC:   XOR   03H            ;GENERATE NEW SC
        POP   IX             ;IX=BUF ADDR
        LD    (IX+0AH),A     ;PUT NSC IN BUF
        LD    C,(IX+09H)     ;GET FC
        XOR   C              ;START NEW BCC
        LD    C,(IX+0CH)     ;GET OC
        XOR   C              ;CALC BCC
        LD    (IX+10H),A     ;PUT NEW BCC IN XBUF
        PUSH  IX             ;MOVE BUF ADDR TO HL
        POP   HL             ;
        SET   ACTV,(HL)      ;SET BUF(ACTV)=1
        LD    BC,0011H       ;
        ADD   HL,BC          ;HL=END OF LINK
        LD    A,04H          ;A='EOT'
        LD    (HL),A         ;PUT IN XBUF
        LD    (SAVLOC),HL    ;SAVE END POINTER
        RET                  ;
;  FILE CONLOG
;       ROUTINE TO CONVERT IBUFS TO LOG MSGS AND
;       PLACE THEM IN XBUF
;**************************************************************
;
;       ENTRY-  HL=IBUF
;       EXIT-   RETURN WITH CARRY IF ERROR,
;       (SAVLOC)=END OF MSG
;
CONLOG: PUSH  HL             ;SAVE IBUF
        SCF                  ;SET CARRY
        BIT   BSY,(HL)       ;IBUF(BSY)=1
        RET   NZ             ;YES, RETURN
        BIT   ERR,(HL)       ;IBUF(ERR)=1
        RET   NZ             ;YES, RETURN
        SET   BSY,(HL)       ;SET IBUF(BSY)=1
;
        INC   HL             ;GET BYTE COUNT
        LD    C,(HL)         ;PUT IN C
        LD    B,00H          ;B=0
        LD    HL,(SAVLOC)    ;GET XBUF POINTER
        PUSH  HL             ;SAVE XBUF POINTER
        ADD   HL,BC          ;ADD BYTE COUNT
        LD    DE,XBFN-0CH    ;GET XBUF END - HEADER COUNT
        CALL  HLDE           ;RETURN CARRY IF HL>DE
```

```
          JR    C,FULRET-$        ;XBUF FULL YES, RETURN
;
          POP   DE                ;GET XBUF
          PUSH  DE                ;SAVE FOR LATER
          LD    BC,000BH          ;BC=BYTE COUNT
          LD    HL,LOGHDR         ;HL=LOG MSG HEADER BUF
          LDIR                    ;MOVE HEADER TO XBUF
          LD    (SAVLOC),DE       ;SAVE NEW XBUF LOC
;
          POP   IX                ;IX=START OF HEADER
          POP   HL                ;HL=IBUF
          CALL  DSTDCB            ;HL=DST DCB, IY=IBUF
          PUSH  IY                ;MOVE DCB TO IY
          PUSH  HL                ;
          POP   IY                ;IY=DCB
;
          LD    C,29H             ;START BCC
          LD    A,(IY+3)          ;GET AC FROM DCB
          OR    40H               ;
          LD    (IX+5),A          ;PUT IN XBUF
          XOR   C                 ;CALC BCC
          LD    C,A               ;
;
          LD    A,(IY+4)          ;GET OC2 FROM DCB
          OR    40H               ;
          LD    (IX+7),A          ;PUT IN XBUF
          XOR   C                 ;CALC BCC
          LD    C,A               ;
;
          LD    A,(IY+5)          ;GET IC FROM DCB
          OR    40H               ;
          LD    (IX+8),A          ;PUT IN XBUF
          XOR   C                 ;CALC BCC
          LD    C,A               ;SAVE BCC
          POP   HL                ;HL=IBUF
;
MVBF:     LD    A,(HL)            ;GET IBUF STATUS
          BIT   LNK,A             ;IBUF(LNK)=1
          JR    Z,LSBF-$          ;NO, LAST BUFFER
;
          INC   HL                ;
          LD    B,(HL)            ;B=BYTE COUNT
          CALL  MOVBLK            ;MOVE BLOCK OF DATA
          PUSH  DE                ;SAVE XBUF LOC
          LD    E,(HL)            ;GET NEXT BUF ADDR
          INC   HL                ;
          LD    D,(HL)            ;
          EX    DE,HL             ;HL=NEXT BUF ADDR
          POP   DE                ;DE=XBUF LOC
          JR    MVBF-$            ;MOVE NEXT BLOCK
;
LSBF:     INC   HL                ;HL=BYTE COUNT POINTER
          LD    B,(HL)            ;B=BYTE COUNT
          LD    (DE),A            ;PUT BCC IN XBUF
```

```
        LD    A,04H           ;A='EOT'
        INC   DE              ;
        LD    (DE),A          ;PUT IN XBUF
        LD    (SAVLOC),DE     ;SAVE XBUF LOC
;
        XOR   A               ;CLEAR CARRY
        RET                   ;
;
FULRET: SCF                   ;SET CARRY
        POP   HL              ;ADJUST STACK
        POP   HL              ;RESTORE HL
        RET                   ;
;
MOVBLK: INC   HL              ;
        INC   HL              ;
        INC   HL              ;HL=DATA LOC
;
MVBT:   LD    A,(HL)          ;GET DATA
        LD    (DE),A          ;PUT IN XBUF
        XOR   C               ;CALC BCC
        LD    C,A             ;SAVE BCC
        INC   HL              ;INC POINTERS
        INC   DE              ;
        DJNZ MVBT-$           ;CONTINUE TILL B=0
        RET                   ;
;
LOGHDR: DEFW 1616H            ;SYN,SYN
        DEFB 01H              ;SOH
        DEFB 60H              ;FC
        DEFB 41H              ;SC
        DEFB 40H              ;AC (40H+DEV#)
        DEFB 41H              ;OC1
        DEFB 40H              ;OC2 (40H+ACK+REQ)
        DEFB 48H              ;IC (48H=TTY)
        DEFB 02H              ;STX
        DEFB 03H              ;ETX
;
;;; FILE DEVOUT
;
;       ROUTINE TO SET UP OUTPUT TO A DEVICE
;***************************************************************
;
;       ENTER- HL=DST DCB, IY=OBUF ADDR
;       EXIT-  IF CARRY, DEVICE IS BUSY OR RD IS SET
;              IF BAD COUNT OR DEV NOT ON LINE, OBUF
;              PUT ON AQ
;              ELSE, PARAMETERS STORED IN DCB (IX=DCB)
;
        LIST X
DEVOUT: SCF                   ;SET CARRY
        BIT   BSY,(HL)        ;DCB(BSY)=1
        RET   NZ              ;YES, RETURN WITH CARRY
        BIT   RD,(HL)         ;NO, DCB(RD)=1
        RET   NZ              ;YES, RETURN WITH CARRY
```

```
        BIT   ONLN,(HL)      ;DCB(ONLN)=1
        JR    Z,IOERR-$      ;NO, DEV NOT ON LINE
        SET   BSY,(HL)       ;SET DCB(BSY)=1
        RES   ERR,(HL)       ;SET DCB(ERR)=0
        RES   ETB,(HL)       ;SET DCB(ETB)=0
        LD    A,(IY+3)       ;GET DST AC
        JP    Z,CMDPRC       ;A=0 YES, PROCESS COMMAND
;
        PUSH  IY             ;
        EX    (SP),HL        ;HL=OBUF
        POP   IX             ;IX=DCB
        LD    (IX+8),H       ;PUT OBUF ADDR IN DCB
        LD    (IX+7),L       ;
        BIT   ETB,(HL)       ;OBUF(ETB)=1
        JR    Z,6            ;NO, SKIP NEXT INST
        SET   ETB,(IX+0)     ;YES, SET DCB(ETB)=1
        LD    E,(HL)         ;E=PRESENT BUF STATUS
        LD    C,(IX+2)       ;C=DEVICE BUS ADDR
        INC   HL             ;
;
        BIT   LNK,E          ;OBUF(LNK)=1
        JR    Z,NLNK-$       ;NO, GET BYTE COUNT
        LD    B,3AH          ;YES, SET B=58 BYTES
        LD    A,(HL)         ;A=TOTAL BYTE COUNT
        SUB   B              ;A=REMAINING BYTE COUNT
        JR    C,IOERR-$      ;IF A<0, ERROR IN BYTE COUNT
        LD    D,A            ;D=REMAINING BYTES
        JR    3              ;SKIP NEXT INST
;
NLNK:   LD    B,(HL)         ;B=TOTAL BYTE COUNT
        INC   HL             ;
        INC   HL             ;
        INC   HL             ;HL=OBUF DATA POINTER
;
        CALL  SAVPR4         ;SAVE ALL PARAMETERS IN DCB
;
        CALL  SETMSK         ;SET I/O MASK BIT
        XOR   A              ;A=00
        OUT   (C),A          ;
        RET                  ;
;
IOERR:  RES   BSY,(IX+0)     ;SET DCB(BSY)=0
        PUSH  IY             ;
        POP   HL             ;HL=OBUF
PUTAQ:  LD    IX,AQUE        ;
        CALL  PUT            ;RESTORE BLOCK TO AQ
        XOR   A              ;CLEAR CARRY
        BIT   LNK,(HL)       ;OBUF(LNK)=1
        RET   Z              ;NO, RETURN
        LD    L,(IY+62)      ;YES, MOVE NEXT BLOCK
        LD    H,(IY+63)      ;HL=BLOCK ADDR
        PUSH  HL             ;
        POP   IY             ;IY=BLOCK ADDR
        JR    PUTAQ-$        ;RESTORE BLOCK
```

```
;
;;; FILE DEVIN
;
;         ROUTINE TO SET UP INPUT FROM A DEVICE
;**************************************************************
;
;         ENTER- HL=SRC DCB
;         EXIT-  IF CARRY, DEVICE IS BUSY OR NOT ON LINE
;                OR RD NOT SET OR NO BUFFERS LEFT IN AQ
;                ELSE, PARAMETERS STORED IN DCB (IX=DCB)
;
DEVIN:    SCF                   ;SET CARRY
          BIT  BSY,(HL)         ;DCB(BSY)=1
          RET  NZ               ;YES, RETURN WITH CARRY
          BIT  ONLN,(HL)        ;DCB(ONLN)=1
          RET  Z                ;NO, RETURN WITH CARRY
          BIT  RD,(HL)          ;DCB(RD)=1
          RET  Z                ;NO, RETURN WITH CARRY
          SET  BSY,(HL)         ;SET DCB(BSY)=1
          RES  ERR,(HL)         ;SET DCB(ERR)=0
          RES  ETB,(HL)         ;SET DCB(ETB)=0
;
          PUSH HL               ;(SP)=DCB
          LD   IX,AQUE          ;IX=AQUE
          CALL GET              ;GET IBUF OFF AQ
          POP  IX               ;IX=DCB
          RET  C                ;IF EMPTY, RETURN WITH CARRY
          PUSH HL               ;
          POP  IY               ;IY=IBUF
;
          XOR  A                ;A=00
          LD   E,A              ;CLEAR E
          LD   (HL),A           ;CLEAR IBUF STATUS
          LD   (IX+7),L         ;PUT IBUF ADDR IN DCB
          LD   (IX+8),H         ;
          INC  HL               ;
          LD   (HL),A           ;CLEAR BYTE COUNT
          LD   D,3AH            ;D=58 BYTES
          LD   C,(IX+2)         ;C=DEVICE BUS ADDR
          INC  HL               ;
          LD   A,(IX+3)         ;GET SRC AC
          LD   (HL),A           ;PUT IN IBUF
          INC  HL               ;
          LD   A,(IX+4)         ;GET DST AC
          LD   (HL),A           ;PUT IN IBUF
          INC  HL               ;HL=OBUF DATA POINTER
;
          OR   A                ;DST AC=0
          JR   Z,SETHST-$       ;YES, SKIP ID HEADER
;
          BIT  IDM,(IX+0)       ;DCB(IDM)=1
          JR   Z,SETCNT-$       ;NO, SKIP ID HEADER
;
          LD   A,(IX+3)         ;GET SRC AC
```

```
        LD   (HDRID),A       ;SAVE
        CALL IDHDR           ;PUT ID HEADER ON IBUF
        JR   FININ-$         ;
;
SETHST: SET  HOST,(IY+0)     ;SET IBUF(HOST)=1
SETCNT: LD   B,D             ;INITAL BYTE COUNT
;
FININ:  CALL SAVPR4          ;SAVE PARAMETERS IN DCB
        CALL SETMSK          ;SET I/O MASK BIT
        IN   A,(C)           ;CLEAR DEVICE STATUS
        EI                   ;ENABLE INTR
        RET                  ;
;;
;       ID HEADER ADDED TO BUFFER- [XX] , XX=LU#
IDHDR:  LD   A,5BH           ;START OF ID HEADER,  '['
        LD   (HL),A          ;PUT IN IBUF
        INC  HL              ;
        LD   A,(HDRID)       ;GET AC VALUE
        PUSH AF              ;SAVE
        SRL  A               ;GET UPPER 4 BITS
        SRL  A               ;
        SRL  A               ;
        SRL  A               ;
        CALL HEXASC          ;CONVERT TO ASCII
        LD   (HL),A          ;PUT IN IBUF
        INC  HL              ;
        POP  AF              ;GET LOWER 4 BITS
        CALL HEXASC          ;CONVERT TO ASCII
        LD   (HL),A          ;PUT IN IBUF
        INC  HL              ;
        LD   A,5DH           ;END OF ID HEADER,  ']'
        LD   (HL),A          ;PUT IN IBUF
        INC  HL              ;
        LD   A,' '           ;SPACE
        LD   (HL),A          ;PUT IN IBUF
        INC  HL              ;
        LD   B,35H           ;B=53
        RET                  ;
;;;  FILE INTR
;
;       INTERRUPT HANDLER
;**************************************************************
;
;       ENTER- INTERRUPT FROM LEVEL 1
;       EXIT-  INTERRUPT DEVICE SERVICED
;
INTR:   PUSH AF              ;SAVE REG'S
        IN   A,(13H)         ;GET HOST STATUS
        BIT  01H,A           ;READ=1
        JP   NZ,HOSTR        ;YES, GO READ CHAR
        PUSH BC              ;
        LD   A,0AH           ;
        OUT  (1EH),A         ;
        IN   A,(1EH)         ;INTR REQUEST REG
```

```
          LD    B,A            ;SAVE IN B
          LD    A,(MASK)       ;I/O MASK
          AND   B              ;CLEAR MASKED REQUESTS
          JR    Z,FIN-$        ;RETURN IF NO REQ'S
          SRL   A              ;BIT 1 SET
          JP    C,SYNWR        ;YES, SYNC WRITE
;
          PUSH  DE             ;NO, SAVE REMAINING REG'S
          PUSH  HL             ;
          PUSH  IX             ;
          PUSH  IY             ;
          LD    HL,INTTAB+2    ;HL=INTR TABLE ADDR+2
LSB:      SRL   A              ;LSB=1
          JR    C,DCBA-$       ;YES, GET DCB ADDR
          INC   HL             ;NO, HL=HL+2
          INC   HL             ;
          JR    LSB-$          ;NEXT BIT
;
DCBA:     LD    E,(HL)         ;E=LOWER BYTE DCB ADDR
          INC   HL             ;
          LD    D,(HL)         ;D=UPPER BYTE DCB ADDR
;;
;         ROUTINE TO HANDLE IO INTERRUPTS
;
          PUSH  DE             ;
          POP   IX             ;IX=DCB
          LD    A,(DE)         ;DEVICE STATUS BYTE
          BIT   BSY,A          ;DEVICE BSY
          JR    Z,INTERR-$     ;NO, GOTO ERROR ROUTINE
          LD    B,(IX+9)       ;GET BYTE COUNT
          LD    C,(IX+2)       ;GET DEV BUS ADDR
          LD    H,(IX+13)      ;GET DATA POINTER
          LD    L,(IX+12)      ;
          BIT   RD,A           ;DCB(RD)=1
          JR    Z,WRITE-$      ;NO, GO WRITE BYTE
;
;         READ DATA BYTE
          CALL  DEVRD          ;READ BYTE
          JR    NC,SVPR-$      ;SAVE PARAMETERS, IF NC
          BIT   ETB,(IX+0)     ;DCB(ETB)=1
          JR    Z,FINIO-$      ;NO, END OF READ
          BIT   ERR,(IX+0)     ;YES, DCB(ERR)=1
          JR    Z,FINIO-$      ;NO, END OF READ
          PUSH  IX             ;YES, INPUT NOT FINISHED
          POP   HL             ;HL=DCB ADDR
          RES   BSY,(IX+0)     ;SET DCB(BSY)=0
          RES   IDM,(IX+0)     ;SET DCB(IDM)=0
          CALL  DEVIN          ;START NEW READ CYCLE
          JR    SVPR+3-$       ;END OF THIS READ
;
WRITE:    CALL  DEVWR          ;WRITE BYTE
          JR    NC,SVPR-$      ;SAVE PARAMETERS, IF NC
;
FINIO:    RES   BSY,(IX+0)     ;SET DCB(BSY)=0
```

```
         JR    5               ;SKIP NEXT INST
;
SVPR:    CALL  SAVPR2          ;SAVE PARAMETERS
         POP   IY              ;RESTORE REG'S
         POP   IX              ;
         JP    RRET            ;RETURN FROM INTR
;
INTERR:  SET   ERR,(IX+0)      ;SET DCB(ERR)=1
         JR    FINIO-$         ;
;;
;
SAVPR4:  LD    (IX+11),D       ;SAVE 4 PARAMETERS IN DCB
         LD    (IX+10),E       ;
SAVPR2:  LD    (IX+9),B        ;SAVE 2 PARAMETERS IN DCB
         LD    (IX+13),H       ;
         LD    (IX+12),L       ;
         RET                   ;
;;
;
SETMSK:  LD    B,(IX+1)        ;GET DEV MASK
         BIT   RD,(IX+0)       ;DEV(RD)=1
         JR    NZ,4            ;YES, SKIP NEXT INST
         SLA   B               ;ADJUST FOR WRITE
         LD    A,(MASK)        ;GET I/O MASK
         OR    B               ;SET BIT
         LD    (MASK),A        ;RESTORE MASK
         RET                   ;
;;
;
CLRMSK:  LD    A,(IX+1)        ;GET DEV MASK
         BIT   RD,(IX+0)       ;DCB(RD)=1
         JR    NZ,4            ;YES, SKIP NEXT INST
         SLA   A               ;ADJUST FOR WRITE
         CPL                   ;INVERT
         LD    B,A             ;MOVE TO B
         LD    A,(MASK)        ;GET PRESENT I/O MASK
         AND   B               ;CLEAR BIT
         LD    (MASK),A        ;RESTORE MASK
         RET                   ;
;
;;; FILE DEVWR
;
;        ROUTINE TO WRITE A BYTE TO A DEVICE
;************************************************************
;
;        ENTER- HL=OBUF DATA POINTER, IX=DST DCB
;               B=PRESENT REMAINING BYTES
;               C=DEVICE BUS ADDR
;        EXIT-  CHAR MOVED TO DEVICE, PARAMETERS UPDATED
;
DEVWR:   LD    A,(HL)          ;GET CHAR FROM OBUF
         OUT   (C),A           ;WRITE TO DEVICE
         INC   HL              ;UPDATE POINTER
         DJNZ  SVWR-$          ;CONTINUE TILL B=0
```

```
;
        PUSH HL             ;SAVE POINTER
        PUSH IX             ;SAVE DCB ADDR
        LD   H,(IX+8)       ;GET OLD OBUF ADDR
        LD   L,(IX+7)       ;
        LD   IX,AQUE        ;IX=AQUE
        CALL PUT            ;PUT ON AQ
        CALL  C,FULERR      ;IF FULL, PUT ON EQ
        POP  IX             ;IX=DCB
        POP  HL             ;HL=POINTER
;
        BIT  LNK,(IX+10)    ;OLD OBUF(LNK)=1
        JR   NZ,NXLK-$      ;YES, NEXT BUF
        CALL CLRMSK         ;CLEAR I/O MASK BIT
        BIT  ETB,(IX+0)     ;DCB(ETB)=1
        JR   NZ,6           ;YES, SKIP NEXT INST
        SET  RD,(IX+0)      ;SET DCB(RD)=1
        SCF                 ;SET CARRY
        RET                 ;END OF OUTPUT
;
NXLK:   LD   E,(HL)         ;GET NEXT OBUF ADDR
        INC  HL             ;
        LD   D,(HL)         ;
        EX   DE,HL          ;HL=NEW OBUF
        LD   (IX+8),H       ;SAVE NEW OBUF ADDR IN DCB
        LD   (IX+7),L       ;
;
        LD   D,(IX+11)      ;GET TOTAL REMAINING BYTES
        LD   E,(HL)         ;E=NEW BUF STATUS
        BIT  LNK,E          ;OBUF(LNK)=1
        JR   Z,NOLNK-$      ;NO, LAST BLOCK
        LD   B,3AH          ;B=58 BYTES
        LD   A,D            ;A=TOTAL REMAINING BYTES
        SUB  B              ;A=TOTAL-58
        JR   C,BDLK-$       ;IF A<0, BAD LNK BIT
        LD   (IX+11),A      ;SAVE IN DCB
        JR   NOLNK+1-$      ;SKIP NEXT INST
;
BDLK:   RES  LNK,E          ;SET OBUF(LNK)=0
;
NOLNK:  LD   B,D            ;B=TOTAL REMAINING BYTES
        LD   (IX+10),E      ;SAVE PRESENT OBUF STATUS
        INC  HL             ;
        INC  HL             ;
        INC  HL             ;
        INC  HL             ;HL=NEW OBUF DATA POINTER
;
SVWR:   XOR  A              ;CLEAR CARRY
        RET                 ;
;
FULERR: PUSH IX             ;SAVE QUEUE ADDR
        EX   (SP),HL        ;HL=QUEUE ADDR, (SP)=BUF ADDR
        LD   IX,EQUE        ;IX=EQUE
        CALL PUT            ;PUT ON EQ
```

```
         RET   C                  ;RETURN IF FULL
         POP   HL                 ;HL=BUF ADDR
         CALL  PUT                ;PUT ON EQ
         CALL  C,GET              ;IF FULL, REMOVE QUEUE ADDR
         RET                      ;
;
;;; FILE DEVRD
;
;        ROUTINE TO READ A BYTE FROM A DEVICE
;*************************************************************
;
;        ENTER- HL=IBUF DATA POINTER, IX=SRC DCB
;               B=PRESENT REMAINING BYTES
;               C=DEVICE BUS ADDR
;        EXIT-  CHAR MOVED TO IBUF, PARAMETERS UPDATED
;
DEVRD:   IN    A,(C)              ;READ CHAR
         CP    03H                ;A='CNTL C'
         JP    Z,CMDIN            ;YES, SET CMD INPUT BUF
         LD    (HL),A             ;PUT IN IBUF
         INC   HL                 ;UPDATE POINTER
         LD    D,A                ;MOVE CHAR TO D
         LD    A,(ENDCHR)         ;GET BUFFER TERMINATOR
         CP    D                  ;D=END CHARACTER
         JR    Z,FINRD-$          ;YES, END OF LINE
         DJNZ  SVRD-$             ;NO, CONTINUE TILL B=0
;
         LD    A,0AEH             ;A=3*58, 3 BLOCKS
         CP    (IX+11)            ;TOTAL COUNT <= A
         JR    NC,NXBK-$          ;YES, NEXT BLOCK
         SET   ERR,(IX+0)         ;NO, SET DCB(ERR)=1
         JR    EMTAQ+3-$          ;CLOSE BUFFER
;
NXBK:    PUSH  IX                 ;SAVE SRC DCB ADDR
         PUSH  HL                 ;SAVE POINTER
         LD    IX,AQUE            ;
         CALL  GET                ;GET NEW BUF OFF AQUE
         JR    C,EMTAQ-$          ;IF CARRY, AQ EMPTY
         POP   DE                 ;
         EX    DE,HL              ;DE=NEW BUF , HL=LINK LOC
         LD    (HL),E             ;PUT LINK ADDR IN OLD BUF
         INC   HL                 ;
         LD    (HL),D             ;
         PUSH  HL                 ;
         POP   IY                 ;IY=OLD BUF POINTER
         SET   LNK,(IY-63)        ;SET OLD BUF(LNK)=1
;
         POP   IX                 ;RESTORE SRC DCB ADDR
         LD    A,3AH              ;A=58
         LD    B,A                ;B=58
         ADD   A,(IX+11)          ;INCREASE TOTAL COUNT BY 58
         LD    (IX+11),A          ;STORE IN SRC DCB
         EX    DE,HL              ;HL=NEW BUF ADDR
         XOR   A                  ;A=0
```

```
          LD    (HL),A          ;CLEAR STATUS
          INC   HL              ;
          LD    (HL),A          ;CLEAR COUNT
          INC   HL              ;
          LD    (HL),A          ;CLEAR SRC AC
          INC   HL              ;
          LD    (HL),A          ;CLEAR DST AC
          INC   HL              ;HL=NEW BUF DATA POINTER
;
SVRD:     XOR   A               ;CLEAR CARRY
          RET                   ;
;
EMTAQ:    POP   HL              ;HL=BUF POINTER
          POP   IX              ;IX=DCB
          SET   ETB,(IX+0)      ;SET SRC DCB(ETB)=1
          JR    3               ;SKIP NEXT INST
;
FINRD:    DEC   B               ;ADJUST B
          LD    A,(IX+11)       ;GET TOTAL BYTE COUNT
          SUB   B               ;SUBTRACT UNUSED BYTES
          LD    L,(IX+7)        ;GET MAIN IBUF ADDR
          LD    H,(IX+8)        ;HL=IBUF
          INC   HL              ;HL=BYTE CNT ADDR
          LD    (HL),A          ;PUT FINAL BYTE COUNT IN IBUF
          DEC   HL              ;HL=IBUF ADDR
          PUSH  IX              ;SAVE SRC DCB ADDR
          INC   C               ;DEVICE STATUS ADDR
          IN    A,(C)           ;GET STATUS BYTE
          AND   08H             ;BAD PARITY BIT
          JR    Z,CKHOST-$      ;NO, CHECK HOST BIT
          SET   PAR,(HL)        ;YES, SET IBUF(PAR)=1
CKHOST:   BIT   HOST,(HL)       ;IBUF(HOST)=1
          JR    Z,CKCMD-$       ;NO, CHECK FOR COMMAND
;
          LD    IX,XQUE         ;IX=XQUE, HL=IBUF
          JR    IBPT-$          ;PUT ON QUEUE
CKCMD:    LD    A,(IX+6)        ;GET DST AC
          SCF                   ;SET CARRY
          OR    A               ;A=0
          CALL  Z,CMDPRC        ;YES, PROCESS CMD
          JR    NC,IBPT+6-$     ;IF CARRY, SAVE BUF
;
PUTWQ:    LD    IX,WQUE         ;GET WQUE ADDR
IBPT:     CALL  PUT             ;PUT IBUF ON QUEUE
          CALL  C,FULERR        ;QUEUE FULL PUT ON EQ
;
          POP   IX              ;RESTORE SRC DCB ADDR
          CALL  CLRMSK          ;CLEAR I/O MASK BIT
          BIT   ETB,(IX+0)      ;DCB(ETB)=1
          JR    Z,6             ;NO, SKIP NEXT 2 INST
          SET   ETB,(HL)        ;YES, SET IBUF(ETB)=1
          JR    NZ,6            ;SKIP NEXT INST
          RES   RD,(IX+0)       ;SET DCB(RD)=0
          SCF                   ;SET CARRY
```

```
;         RET                   ;
;
;;; FILE HOSTIO
;
;         ROUTINE TO SET UP HOST READ
;**************************************************************
;
;         ENTER- ;;
;         EXIT- ;;
;
STRBUF:   LD    HL,RBUF         ;GET RBUF ADDR
          BIT   EMTY,(HL)       ;RBUF(EMTY)=1
          RET   Z               ;NO, RETURN
;
          PUSH  HL              ;SAVE RBUF
          LD    HL,HCB          ;GET HCB ADDR
          BIT   BSY,(HL)        ;HCB(BSY)=1
          RET   NZ              ;YES, RETURN
          BIT   RD,(HL)         ;HCB(RD)=1
          RET   Z               ;NO, RETURN
;
          IN    A,(13H)         ;GET STATUS
          BIT   DSR,A           ;HOST(DSR)=1
          JR    Z,NODSR-$       ;
;
          SET   BSY,(HL)        ;SET HCB(BSY)=1
          ;
          LD    BC,400H-4       ;BC=BYTE COUNT
          LD    DE,RBUF+4       ;DE=DATA POINTER
          LD    (HCB+2),BC      ;SAVE IN HCB
          LD    (HCB+4),DE      ;
          XOR   A               ;CLEAR CARRY
          RET                   ;
;
;;;       ROUTINE TO READ A CHAR FROM HOST CHAN
;**************************************************************
;
;         ENTER- ALL INTR'S DISABLED
;         EXIT- PUT CHAR FROM HOST IN RBUF
;
HOSTR:    PUSH  BC              ;SAVE REG'S
          PUSH  DE              ;
          PUSH  HL              ;
;
          LD    BC,(HCB+2)      ;GET COUNT
          LD    DE,(HCB+4)      ;GET POINTER
          LD    HL,HCB          ;HL=HCB ADDR
;
          BIT   SYNC,(HL)       ;MODEM IN SYNC
          JR    NZ,RDAT-$       ;YES, READ CHAR
          BIT   SYNC,A          ;NO, MODEM SYNC BIT SET
          JR    Z,RRET-$        ;NO, RET
          SET   SYNC,(HL)       ;YES, SET HCB(SYNC)=1
;
```

```
RDAT:     IN    A,(12H)         ;READ CHAR
          CP    SYN             ;=SYNC CHAR
          JR    Z,RRET-$        ;YES, SKIP CHAR
          LD    (DE),A          ;NO, SAVE IN RBUF
          INC   DE              ;UPDATE POINTER
          DEC   BC              ;UPDATE COUNT
          CP    EOT             ;=EOT CHAR
          JR    Z,FINSR-$       ;YES, END OF RBUF
;
          LD    A,B             ;
          ADD   A,C             ;BC=0
          JR    Z,FINSR-$       ;YES, END OF BUFFER
;
SRET:     LD    (HCB+2),BC      ;SAVE COUNT
          LD    (HCB+4),DE      ;SAVE POINTER
;
RRET:     POP   HL              ;RESTORE REG'S
          POP   DE              ;
FIN:      POP   BC              ;
ARET:     POP   AF              ;RESTORE AF
          EI                    ;
          RETI                  ;
;
FINSR:    RES   RD,(HL)         ;SET HCB(RD)=0
          RES   BSY,(HL)        ;SET HCB(BSY)=0
          RES   SYNC,(HL)       ;SET HCB(SYNC)=0
          LD    HL,RBUF         ;
          SET   FULL,(HL)       ;SET RBUF(FULL)=1
          LD    HL,400H-4       ;CALCULATE BYTE COUNT
          SBC   HL,BC           ;
          LD    (RBUF+1),BC     ;PUT IN RBUF
          JR    RRET-$          ;RETURN
;
;;;      ROUTINE TO SEND CHAR TO HOST
;***************************************************************
;
;        ENTER- ALL INTR'S DISABLED
;        EXIT - CHAR SENT TO HOST
;
HOSTW:    PUSH  DE              ;SAVE DE
          LD    BC,(HCB+2)      ;GET COUNT
          LD    DE,(HCB+4)      ;GET POINTER
          LD    HL,HCB          ;HL=HCB
          LD    A,(DE)          ;GET CHAR
          OUT   (12H),A         ;SEND TO HOST
          INC   DE              ;INC POINTER
          DEC   BC              ;DEC COUNT
          CP    EOT             ;='EOT'
          JR    Z,FINSW-$       ;YES, END OF BUFFER
          LD    A,B             ;
          ADD   A,C             ;A=0
          JR    Z,FINSW-$       ;YES, END OF BUFFER
          LD    (HCB+2),BC      ;SAVE COUNT
          LD    (HCB+4),DE      ;SAVE POINTER
```

```
          JR    RRET-$           ;RETURN
;
FINSW:    SET   RD,(HL)          ;SET HCB(RD)=1
          RES   BSY,(HL)         ;SET HCB(BSY)=0
          LD    HL,XBUF          ;HL=XBUF
          SET   EMTY,(HL)        ;SET XBUF(EMTY)=1
          JR    RRET-$           ;RETURN
;
;;; FILE CMDPRC
;
;         ROUTINE TO SET UP COMMAND BUFFERS
;**************************************************************
;
;         ENTER- IX=DCB
;         EXIT- BUFFER SET UP TO RECEIVE COMMAND
;
CMDIN:    LD    L,(IX+7)         ;GET BUF ADDR
          LD    H,(IX+8)         ;
          PUSH  IX               ;SAVE DCB ADDR
          CALL  IOERR            ;RESTORE PRESENT BUF TO AQ
          POP   IX               ;
          LD    A,(IX+6)         ;GET PRESENT AC
          PUSH  AF               ;SAVE AC
;
          LD    (IX+6),00H       ;PUT CMD PROCESSOR # IN DCB
          CALL  DEVIN            ;SET UP INPUT TO CMD PROC
          POP   AF               ;GET OLD AC
          LD    (IX+6),A         ;RESTORE TO DCB
          SCF                    ;
          RET                    ;
;
;;        ROUTINE TO HANDLE COMMANDS
;**************************************************************
;
;         ENTER- IX=DCB, HL=BUF
;         EXIT- COMMAND PROCESSED
;
CMDPRC:   PUSH  HL               ;
          POP   IY               ;IY=BUF
          INC   HL               ;
          INC   HL               ;
          INC   HL               ;
          INC   HL               ;HL=DATA POINTER
          LD    A,(HL)           ;GET FIRST CHAR
          CP    'A'              ;=A
          JR    Z,ASGN-$         ;YES, ASIGN # TO DCB(SRC AC)
          CP    'D'              ;= 'D'
          JR    Z,DDIS-$         ;YES, DISCONNECT DEVICE FROM HOS
          CP    01H              ;= 'CNTL A'
          JR    Z,DSEL-$         ;YES, SEND SELECT TO HOST & ASGN
;
FINC:     XOR   A                ;
          RET                    ;
;
```

```
;;
ASGN:   CALL AHBYTE         ;CONVERT ASCII CHAR TO HEX BYTE
        JR   C,FINC-$       ;CARRY, IF ERROR
        OR   A              ;A=HOST
        JR   Z,CKHST-$      ;YES, CHECK HOST
        CP   04H            ;A<04H
        JR   NC,NONE-$      ;NO, NO SUCH DEVICE
        PUSH AF             ;SAVE AC
        CALL DEVDCB         ;GET DCB ADDR
        POP  HL             ;HL=DCB
        BIT  ONLN,(HL)      ;DCB(ONLN)=1
        JR   Z,NODEV-$      ;NO, DEV NOT ON LINE
        POP  AF             ;YES, GET AC
PTAC:   LD   (IX+6),A       ;PUT NEW AC IN DCB
        RET                 ;
;
CKHST:  LD   HL,HCB         ;GET HOST DCB ADDR
        BIT  ONLN,(HL)      ;HCB(ONLN)=1
        JR   NZ,PTAC-$      ;YES, HOST IS ON LINE
        PUSH IY             ;
        POP  HL             ;
        INC  HL             ;
        INC  HL             ;
;
        LD   B,3AH          ;B=REMAINING BYTES IN BUF
        LD   A,00H          ;A=ERROR MSG #
        JR   ERROUT-$       ;SEND ERROR MSG
;
NODEV:  PUSH IY             ;
        POP  HL             ;
        INC  HL             ;
        INC  HL             ;
        LD   (HDRID),A
        CALL IDHDR          ;PUT HEADER ON BUF
        LD   A,01H          ;A=ERROR MSG #
        JR   ERROUT-$       ;SEND ERROR MSG
;
NONE:   PUSH IY             ;
        POP  HL             ;
        LD   (HDRID),A
        CALL IDHDR          ;PUT HEADER ON BUF
        LD   A,03H          ;A=ERROR MSG #
        JR   ERROUT-$       ;SEND ERROR MSG
;
DDIS:   CALL AHBYTE         ;CONVERT ASCII CHAR TO HEX BYTE
        JR   C,FINC-$       ;
        OR   A              ;A=0
        RET  Z              ;YES, RETURN
        CP   04H            ;A<04H
        RET  NC             ;NO, RETURN
        CALL DISGEN         ;GENERATE DISCONNECT MSG
        LD   IX,XQUE        ;
        CALL PUT            ;PUT MSG ON QUEUE
        XOR  A              ;
```

```
          RET                    ;
;
DSEL:     CALL AHBYTE            ;CONVERT ASCII CHAR TO HEX BYTE
          JR   C,FINC-$          ;
          OR   A                 ;A=0
          RET  Z                 ;YES, RETURN
          CP   04H               ;A<04H
          RET  NC                ;NO, RETURN
          CALL SELGEN            ;GENERATE SELECT MSG
          LD   IX,XQUE           ;
          CALL PUT               ;PUT MSG ON QUEUE
          XOR  A                 ;
          RET                    ;
;
;;
AHBYTE:   LD   B,0               ;B=0
          JR   GTNBL-$           ;GET FIRST NIBBLE
;
NXNBL:    ADD  A,B               ;ADD PREVIOUS VALUE*4
          ADD  A,B               ;
          LD   B,A               ;SAVE IN B
;
GTNBL:    INC  HL                ;GET CHAR FOR NIBBLE
          LD   A,(HL)            ;
          CALL ASCHEX            ;CONVERT TO HEX NIBBLE
          JR   NC,NXNBL-$        ;CARRY, IF NON-HEX CHAR
;
          LD   A,0DH             ;
          CP   (HL)              ;=CR
          SCF                    ;
          RET  NZ                ;NO, RET WITH CARRY
          XOR  A                 ;YES, CLEAR CARRY
          LD   A,B               ;PUT VALUE IN A
          RET                    ;
;
;;
ASCHEX:   SUB  30H               ;A<'0'
          RET  C                 ;YES, RET WITH CARRY
          CP   17H               ;A>'F'
          CCF                    ;
          RET  C                 ;YES, RET WITH CARRY
          CP   0AH               ;0<=A<=9
          CCF                    ;
          RET  NC                ;YES, RET NO CARRY
          SUB  07H               ;ADJUST CHAR
          CP   0AH               ;NO CARRY IF, 'A'<=A<='F'
          RET                    ;
;
;;;       ROUTINE TO OUTPUT ERROR MESSAGES
;***********************************************************
;
;         ENTER- HL=BUF DATA POINTER, IY=BUF ADDR
;                A=ERROR CODE, B=REMAINING BYTES, IX=DST DCB
;         EXIT-  ERROR MSG PUT IN BUF AND DEVOUT ENTERED
```

```
;
ERROUT:   PUSH BC             ;SAVE BYTE COUNT
          PUSH HL             ;SAVE DATA POINTER
          LD   HL,ERRTAB      ;HL=ERROR TABLE ADDR
          ADD  A,A            ;A=A+A
          LD   B,00           ;
          LD   C,A            ;BC=ERROR CODE OFFSET
          ADD  HL,BC          ;HL=ERRTAB+OFFSET
          LD   E,(HL)         ;
          INC  HL             ;
          LD   D,(HL)         ;DE=ERROR MSG ADDR
;
          LD   A,(DE)         ;A=MSG BYTE COUNT
          LD   C,A            ;BC=ERR MSG BYTE COUNT
          INC  DE             ;DE=ERR MSG POINTER
          POP  HL             ;HL=BUF DATA POINTER
          EX   DE,HL          ;
          LDIR                ;MOVE ERROR MSG TO BUF
;
          POP  BC             ;GET REMAINING BYTES
          SUB  B              ;
          ADD  A,3AH          ;CALCULATE TOTAL BYTE COUNT
          LD   (IY+1),A       ;PUT IN BUF
          RES  BSY,(IX+0)     ;SET DCB(BSY)=0
          RES  RD,(IX+0)      ;SET DCB(RD)=0
          PUSH IX             ;
          POP  HL             ;HL=DST DCB
          JP   DEVOUT         ;OUTPUT MESSAGE
;
ERRTAB:   DEFW ERR00          ;ERROR MSG 00
          DEFW ERR01          ;ERROR MSG 01
          DEFW ERR02          ;ERROR MSG 02
          DEFW ERR02          ;ERROR MSG 02
          DEFW ERR02          ;ERROR MSG 02
          DEFW ERR02          ;ERROR MSG 02
          DEFW ERR02          ;ERROR MSG 02
          DEFW ERR02          ;ERROR MSG 02
;
ERR00:    DEFB 13H            ;BYTE COUNT
          DEFM 'HOST'         ;
          DEFM ' NOT'         ;
          DEFM ' ON '         ;
          DEFM 'LINE'         ;
          DEFM '.'            ;
          DEFW 0A0DH          ;LF,CR
;
ERR01:    DEFB 0FH            ;BYTE COUNT
          DEFM ' NOT'         ;
          DEFM ' ON '         ;
          DEFM 'LINE'         ;
          DEFM '.'            ;
          DEFW 0A0DH          ;LF,CR
;
ERR02:    DEFB 12H            ;BYTE COUNT
```

```
        DEFM ' NOT'         ;
        DEFM ' DEF'         ;
        DEFM 'INED'         ;
        DEFM '... '         ;
        DEFW 0A0DH          ;LF,CR
;
;  FILE INIT
;       ROUTINE TO INIT SYSTEM
;************************************************************
;
;       SET UP RESTART JUMPS
;
        LD   SP,2FFFH       ;STACK=TOP OF MEMORY
;
        LD   A,03CH         ;CODE FOR JUMP
        LD   BC,START       ;JUMP ADDR FOR RST 00H
        LD   (00),A         ;STORE JUMP
        LD   (01H),BC       ;STORE ADDR
        LD   BC,MAIN        ;JUMP ADDR FOR RST 66H
        LD   (66H),A        ;STORE JUMP
        LD   (67H),BC       ;STORE ADDR
;
        CALL CIO            ;SET UP I/O,INTR & TIMERS
        RET
;
;        ROUTINE TO SET UP INTERRUPTS
;        SET UP OF CLOCK DEVICE (CTC)
;        & INTR HANDLER (8259)
;************************************************************
;
CIO:     LD   C,18H         ;CHANNEL #0
         LD   DE,951FH      ;(TIMR,.25MS,INTR)
         OUT  (C),D         ;
         OUT  (C),E         ;
         INC  C             ;CHANNEL #1
         LD   DE,953EH      ;(TIMR,.5MS,INTR)
         OUT  (C),D         ;
         OUT  (C),E         ;
         INC  C             ;CHANNEL #2
         LD   DE,957DH      ;(TIMR,1MS,INTR)
         OUT  (C),D         ;
         OUT  (C),E         ;
         INC  C             ;CHANNEL #3
         LD   DE,0C5FAH     ;(CNTR,CNT=250,INTR)
         OUT  (C),D         ;
         OUT  (C),E         ;
         LD   BC,IVEC       ;INTR VECTOR ADDR
         LD   A,B           ;
         LD   I,A           ;SET UP UPPER BYTE
         LD   A,C           ;
         OUT  (18H),A       ;SET UP LOWER BYTE
;
         LD   A,12H         ;SET UP INTR DEVICE (8259)
         OUT  (1EH),A       ;
```

```
         XOR  A              ;CLEAR A
         OUT  (1FH),A        ;
         LD   (MASK),A       ;INIT MASK
         CPL                 ;INVERT A
         OUT  (1FH),A        ;
;
         IM   2              ;SET INTR MODE 2
;
;        SET UP I\O CHANNELS
;
        LD   BC,4011H       ;RESET USART, I\O CHAN(10,11)
        LD   DE,0FA35H      ;SET MODE & FUNCTIONS
        CALL MSET           ;OUTPUT COMMANDS
        LD   C,13H          ;I\O CHAN(12,13)
        CALL MSET           ;OUTPUT COMMANDS
        LD   C,15H          ;I\O CHAN(14,15)
        CALL MSET           ;OUTPUT COMMANDS
        LD   C,17H          ;I\O CHAN(16,17)
        CALL MSET           ;OUTPUT COMMANDS
        CALL CIO            ;SET UP INTR & TIMERS
        RET                 ;
;
MSET:   OUT  (C),B          ;OUTPUT B
        OUT  (C),D          ;OUTPUT D
        OUT  (C),E          ;OUTPUT E
        RET                 ;
;
INT0:    NOP                 ;INTERRUPT LEVEL 0
         NOP                 ;
         NOP                 ;
         EI                  ;
         RETI                ;
;
INT1:    EQU  INTR           ;INTERRUPT LEVEL 1
;
INT2:    NOP                 ;INTERRUPT LEVEL 2
         NOP                 ;
         NOP                 ;
         EI                  ;
         RETI                ;
;
INT3:    NOP                 ;INTERRUPT LEVEL 3
         NOP                 ;
         NOP                 ;
         EI                  ;
         RETI                ;
;
;;; FILE SUBS
;
;        FILE CONTAINING FLAG BIT ASSIGNMENTS AND
;        SMALL SUBROUTINES USED BY MORE THAN ONE
;        OTHER ROUTINE.
;*************************************************************
;
```

```
BSY:      EQU  0               ;QUEUE OR BUFFER BUSY
FULL:     EQU  1               ;FULL QUEUE OR BUFFER
RD:       EQU  1               ;DEVICE READ BIT
EMTY:     EQU  2               ;EMPTY QUEUE OR BUFFER
ACPT:     EQU  2               ;RNP ACCEPTING CALL
ACK:      EQU  3               ;PREVIOUS MSG ACKNOWLEDGED
HOST:     EQU  3               ;BUFFER TO OR FROM HOST
NSC:      EQU  4               ;NEW SEQUENCE COUNT
ONLN:     EQU  4               ;RNP OR DEVICE ON LINE
PAR:      EQU  4               ;BAD PARITY IN BUFFER
SVM:      EQU  5               ;SERVICE MESSAGE REQUEST
ACTV:     EQU  5               ;BUFFER PRESENTLY ACTIVE
ETB:      EQU  5               ;END OF BUF, NOT END OF TEXT
LOG:      EQU  6               ;LOG MSG IN RBUF
LNK:      EQU  6               ;BUF LINKED TO NEXT BUF
IDM:      EQU  6               ;ID MESSAGE REQUEST
SYNC:     EQU  6               ;SYNC RD/WR IN SYNC
ERR:      EQU  7               ;ERROR IN BUFFER
;
;;
;
DSTDCB:   PUSH HL              ;MOVE BUF ADDR
          POP  IY              ;TO IY
          LD   A,(IY+3)        ;GET DST AC
          CALL DEVDCB          ;GET DCB, (SP)=DCB
          POP  HL              ;HL=DST DCB
          RET                  ;
;;
;
INITAQ:   LD   IX,AQUE         ;IX=AQUE (AVAILABLE BUFFERS)
          LD   BC,40H          ;BC=64 (SIZE OF EACH BUF)
          LD   HL,ABUF         ;HL=ABUF (START OF BUF AREA)
;
PTBF:     CALL PUT             ;PUT ADDR ON AQUE
          RET  C               ;RETURN WHEN AQUE FULL
          ADD  HL,BC           ;NEXT BUF
          JR   PTBF-$          ;CONTINUE TILL AQUE FULL
;;
;
DEVDCB:   ADD  A,A             ;A=A+A
          LD   C,A             ;C=OFFSET
          LD   B,00H           ;
          LD   HL,DCBTAB       ;HL=DCBTAB ADDR
          ADD  HL,BC           ;HL=DCBTAB+OFFSET
          LD   C,(HL)          ;DCB L
          INC  HL              ;
          LD   B,(HL)          ;DCB U
          POP  HL              ;GET RETURN ADDR
          PUSH BC              ;SAVE DCB ADDR
          PUSH HL              ;RESTORE RETURN ADDR
          RET                  ;
;;
;
HLDE:     LD   A,E             ;LOWER BYTE
```

```
        SUB   L                ;
        LD    A,D              ;UPPER BYTE
        SBC   A,H              ;CARRY SET IF HL>DE
        RET                    ;
;;
;
HEXASC: AND   0FH              ;CLEAR UPPER 4 BITS
        ADD   A,90H            ;A=A+90H
        DAA                    ;DECIMAL ADJUST
        ADC   A,40H            ;A=A+40H+CARRY
        DAA                    ;DECIMAL ADJUST
        RET                    ;ASCII VALUE IN A
;
;;;  FILE QUE
;        INPUT-OUTPUT QUEUE ROUTINES
;        VERSION 1.0  REV C
;
;        ROUTINE TO PUT AN ADDRESS INTO QUEUE.
;*************************************************************
;        ENTRY- IX CONTAINS QUEUE CONTROL BLOCK ADDR
;        AND HL CONTAINS DATA TO BE PUT ON QUEUE.
;        EXIT- IF QUEUE IS FULL, RETURN WITH CARRY SET
;        NORMAL RETURN IS WITH CARRY CLEAR.
;
PUT:    SCF                    ;SET CARRY FLAG
        BIT   BSY,(IX+0)       ;TEST FOR BSY QUEUE
        RET   NZ               ;YES, RETURN
        BIT   FULL,(IX+0)      ;TEST FOR FULL QUEUE
        RET   NZ               ;YES, RETURN
        SET   BSY,(IX+0)       ;SET BSY BIT
        RES   EMTY,(IX+0)      ;CLEAR EMPTY FLAG
        EX    DE,HL            ;DE=ADDR
        LD    L,(IX+3)         ;LOWER BYTE BQP
        LD    H,(IX+4)         ;UPPER BYTE BQP
;
        LD    (HL),E           ;PUT LOWER BYTE
        INC   HL               ;NEXT BYTE
        LD    (HL),D           ;PUT UPPER BYTE
        INC   HL               ;HL=BQP+2
;
        CALL  CHKBB            ;IF BQP=BB, SET BQP=TB
;
        LD    A,(IX+1)         ;LOWER BYTE TQP
        CP    L                ;= L
        JR    NZ,PUTR-$        ;NO, QUEUE NOT FULL
        LD    A,(IX+2)         ;YES, UPPER BYTE TQP
        CP    H                ;= H
        JR    NZ,PUTR-$        ;NO, QUEUE NOT FULL
        SET   FULL,(IX+0)      ;YES, SET QUEUE FULL
;
PUTR:   LD    (IX+3),L         ;LOWER BYTE BQP
        LD    (IX+4),H         ;UPPER BYTE BQP
        EX    DE,HL            ;HL=ADDR
        XOR   A                ;CLEAR CARRY FLAG
```

```
        RES   BSY,(IX+0)    ;RESET BSY FLAG
        RET
;;
;
;       ROUTINE TO GET ADDRESS FROM QUEUE
;*************************************************************
;       ENTRY- IX CONTAINS QUEUE CONTROL BLOCK ADDR
;       EXIT- HL CONTAINS DATA REMOVED FROM QUEUE,
;       IF QUEUE IS EMPTY, RETURN WITH CARRY SET
;       NORMAL RETURN IS WITH CARRY CLEAR.
;
GET:    SCF                 ;SET CARRY FLAG
        BIT   BSY,(IX+0)    ;TEST FOR BSY QUEUE
        RET   NZ            ;YES, RETURN
        BIT   EMTY,(IX+0)   ;TEST FOR EMPTY QUEUE
        RET   NZ            ;YES, RETURN
        SET   BSY,(IX+0)    ;SET BSY FLAG
        RES   FULL,(IX+0)   ;CLEAR FULL FLAG
        LD    L,(IX+1)      ;LOWER BYTE TQP
        LD    H,(IX+2)      ;UPPER BYTE TQP
;
        LD    E,(HL)        ;GET LOWER BYTE
        INC   HL            ;NEXT LOCATION
        LD    D,(HL)        ;GET UPPER BYTE
        INC   HL            ;HL=TQP+2
;
        CALL  CHKBB         ;IF TQP=BB, SET TQP=TB
;
        LD    A,(IX+3)      ;LOWER BYTE BQP
        CP    L             ;= L
        JR    NZ,GETR-$     ;NO, QUEUE NOT EMPTY
        LD    A,(IX+4)      ;YES, UPPER BYTE BQP
        CP    H             ;= H
        JR    NZ,GETR-$     ;NO, QUEUE NOT EMPTY
        SET   EMTY,(IX+0)   ;YES, SET QUEUE EMPTY
;
GETR:   LD    (IX+1),L      ;LOWER BYTE TQP
        LD    (IX+2),H      ;UPPER BYTE TQP
        EX    DE,HL         ;HL=ADDR
        XOR   A             ;CLEAR CARRY FLAG
        RES   BSY,(IX+0)    ;RESET BSY FLAG
        RET
;;
;
CHKBB:  LD    A,(IX+7)      ;LOWER BYTE BB
        CP    L             ;= L
        RET   NZ            ;NO, RETURN
        LD    A,(IX+8)      ;YES, UPPER BYTE BB
        CP    H             ;= H
        RET   NZ            ;NO, RETURN
        LD    L,(IX+5)      ;LOWER BYTE TB
        LD    H,(IX+6)      ;UPPER BYTE TB
        RET                 ;
;;;   FILE HBUFS
```

```
;         BUFFER STORAGE AND QUEUES
;*************************************************************
          ORG  2000H
;
;         HOST XMIT AND RECEIVE BUFFERS
;
RBUF:     EQU  $                  ;RECEIVE BUFFER (FROM HOST)
          DEFB 04H                ;STATUS BYTE
          DEFW 0000H              ;BYTE COUNT
          DEFS 400H-03H           ;BUFFER STORAGE LOCATIONS
RBFN:     EQU  $                  ;END OF RBUF
;
;
XBUF:     EQU  $                  ;XMIT BUFFER (TO HOST)
          DEFB 04H                ;STATUS BYTE
          DEFW 0000H              ;BYTE COUNT
          DEFB 00H                ;
          DEFW 1616H              ;SYN,SYN
          DEFW 1616H              ;SYN,SYN
          DEFB 01H                ;SOH
          DEFB 48H                ;FC (ACK/NAK MSG)
XBSC:     DEFB 41H                ;SC (41H OR 42H)
          DEFB 40H                ;AC
XBOC:     DEFB 40H                ;OC (ACK,NO INST)
          DEFB 41H                ;IC (41H+#MSG)
          DEFB 02H                ;STX
          DEFB 03H                ;ETX
XBCC:     DEFB 00H                ;BCC (BLOCK CHECK CHAR)
          DEFW 1616H              ;SYN,SYN
LOG1:     DEFB 01H                ;SOH (START OF FIRST LOGICAL MSG
          DEFS 400H-14H           ;BUFFER STORAGE LOCATIONS
XBFN:     EQU  $                  ;END OF XBUF
;
;
SBUF:     EQU  $                  ;SERVICE MESSAGE BUFFER
          DEFB 04H                ;STATUS BYTE
          DEFW 000EH              ;BYTE CNT
          DEFB 00H                ;
          DEFW 1616H              ;SYN,SYN
          DEFW 1616H              ;SYN,SYN
          DEFB 01H                ;SOH
          DEFB 42H                ;FC (SRV MSG)
SBSC:     DEFB 41H                ;SC (41H OR 42H)
          DEFB 40H                ;AC
SBOC:     DEFB 40H                ;OC (40H+ACK/NAK+SVM)
          DEFB 41H                ;IC (1 MSG)
          DEFB 02H                ;STX (NO TEXT)
          DEFB 03H                ;ETX
SBCC:     DEFB 42H                ;BCC (42H+SC+OC)
          DEFB 04H                ;EOT
;
QFRM:     EQU  $                  ;Q-FRAME BUFFER
          DEFB 00H                ;STATUS BYTE
          DEFW 0006H              ;
```

```
         DEFB 00H          ;
         DEFW 1616H        ;SYN,SYN
         DEFW 1616H        ;SYN,SYN
         DEFB 01H          ;SOH
         DEFB 04H          ;EOT
;;;  FILE QUES
;       QUEUE CONTROL AND STORAGE BUFFERS
;***************************************************************
;
        ORG  281CH
RQUE:   EQU  $             ;RECEIVE QUEUE
        DEFB 04H           ;RQUE FLAG
        DEFW TRB           ;TOP RQUE POINTER
        DEFW TRB           ;BOTTOM RQUE POINTER
        DEFW TRB           ;TOP RQUE BUFFER
        DEFW TRB+20H       ;BOTTOM RQUE BUFFER
TRB:    DEFS 20H           ;RQUE BUFFER
;
XQUE:   EQU  $             ;TRANSMIT QUEUE
        DEFB 04H           ;XQUE FLAG
        DEFW TXB           ;TOP XQUE POINTER
        DEFW TXB           ;BOTTOM XQUE POINTER
        DEFW TXB           ;TOP XQUE BUFFER
        DEFW TXB+20H       ;BOTTOM XQUE BUFFER
TXB:    DEFS 20H           ;XQUE BUFFER
;
AQUE:   EQU  $             ;AVAILABLE BUFFERS QUEUE
        DEFB 04H           ;AQUE FLAG
        DEFW TAB           ;TOP AQUE POINTER
        DEFW TAB           ;BOTTOM AQUE POINTER
        DEFW TAB           ;TOP AQUE BUFFER
        DEFW TAB+20H       ;BOTTOM AQUE BUFFER
TAB:    DEFS 20H           ;AQUE BUFFER
;
WQUE:   EQU  $             ;WAIT QUEUE
        DEFB 04H           ;WQUE FLAG
        DEFW TWB           ;TOP WQUE POINTER
        DEFW TWB           ;BOTTOM WQUE POINTER
        DEFW TWB           ;TOP WQUE BUFFER
        DEFW TWB+20H       ;BOTTOM WQUE BUFFER
TWB:    DEFS 20H           ;WQUE BUFFER
;
EQUE:   EQU  $             ;ERROR QUEUE
        DEFB 04H           ;EQUE FLAG
        DEFW TEB           ;TOP EQUE POINTER
        DEFW TEB           ;BOTTOM EQUE POINTER
        DEFW TEB           ;TOP EQUE BUFFER
        DEFW TEB+20H       ;BOTTOM EQUE BUFFER
TEB:    DEFS 20H           ;EQUE BUFFER
;
;;;  FILE STRG
;    GENERAL STORAGE AREA
;***************************************************************
;
```

```
HDRID:   DEFB 00H         ;HEADER ID SAVE AREA
;
SAVSC:   DEFB 00H         ;SAVE PRESENT SC (RBUF)
SAVIC:   DEFB 00H         ;SAVE PRESENT IC (RBUF)
SAVOC:   DEFB 00H         ;SAVE PRESENT OC (RBUF)
SAVMSG:  DEFW 0000H       ;SAVE MSG LENGTH & FLAG (RBUF)
         DEFW 0000H       ;SAVE MSG LOCATION
         DEFW 0000H       ;SAVE BUF LOCATION
;
SAVLOC:  DEFW 0000H       ;SAVE LOCATION POINTER (XBUF)
SAVBCC:  DEFB 00H         ;SAVE NEW BCC (XBUF)
;
MASK:    DEFB 0FFH        ;LEVEL 3 INTR MASK
ENDCHR:  DEFB 0DH         ;BUFFER TERMINATING CHAR <CR>
;
         ORG  2900H       ;
ABUF:    EQU  $           ;AVAILABLE BUFFER AREA
         DEFS 10H*40H     ;16 BUFS * 64 BYTES PER BUF
;
IVEC:    EQU  $             ;INTERRUPT VECTOR TABLE
         DEFW INT0          ;INTERRUPT LEVEL 0
         DEFW INT1          ;INTERRUPT LEVEL 1
         DEFW INT2          ;INTERRUPT LEVEL 2
         DEFW INT3          ;INTERRUPT LEVEL 3
;
INTTAB:  DEFW DCB00         ;HOST DCB (HCB)
         DEFW DCB01         ;CONSOLE DCB
         DEFW DCB01         ;CONSOLE DCB
         DEFW DCB02         ;RMC1 DCB
         DEFW DCB02         ;RMC1 DCB
         DEFW DCB03         ;RMC2 DCB
         DEFW DCB03         ;RMC2 DCB
         DEFW DCB01         ;EXTERNAL INTR DCB
;
DCBTAB:  DEFW DCB00         ;DEV00 DCB (HOST)
         DEFW DCB01         ;DEV01 DCB (CONSOLE)
         DEFW DCB02         ;DEV02 DCB (RMC01)
         DEFW DCB03         ;DEV03 DCB (RMC02)
         DEFW DCBXX         ;DEV04 DCB
         DEFW DCBXX         ;DEV05 DCB
         DEFW DCBXX         ;DEV06 DCB
         DEFW DCBXX         ;DEV07 DCB
;
DCBXX:   EQU  0FFFFH        ;DUMMY DCB
;
HCB:     EQU  $             ;HOST DCB = DCB00
DCB00:   DEFB 02H           ;DEVICE STATUS FLAGS
         DEFW 0000          ;SAVE BYTE COUNT
         DEFW 0000          ;SAVE BUFFER POINTER
;
DCB01:   DEFB 02H           ;DEVICE STATUS FLAGS
         DEFB 02H           ;DEVICE MASK
         DEFB 10H           ;DEVICE #
         DEFB 01H           ;AC (LU#)
```

```
        DEFB 00H            ;DST AC (READ ONLY)
        DEFW 0000           ;SAVE MAIN BUF ADDR
        DEFB 0000           ;SAVE REMAINING BYTES
        DEFW 0000           ;SAVE DE (STATUS, TOTAL BYTES)
        DEFW 0000           ;SAVE HL (POINTER)
;
DCB02:  DEFB 02H            ;DEVICE STATUS FLAGS
        DEFB 08H            ;DEVICE MASK
        DEFB 14H            ;DEVICE #
        DEFB 02H            ;AC (LU#)
        DEFB 00H            ;DST AC (READ ONLY)
        DEFW 0000           ;SAVE MAIN BUF ADDR
        DEFB 0000           ;SAVE REMAINING BYTES
        DEFW 0000           ;SAVE DE (STATUS, TOTAL BYTES)
        DEFW 0000           ;SAVE HL (POINTER)
;
DCB03:  DEFB 02H            ;DEVICE STATUS FLAGS
        DEFB 20H            ;DEVICE MASK
        DEFB 16H            ;DEVICE #
        DEFB 03H            ;AC (LU#)
        DEFB 00H            ;DST AC (READ ONLY)
        DEFW 0000           ;SAVE MAIN BUF ADDR
        DEFB 0000           ;SAVE REMAINING BYTES
        DEFW 0000           ;SAVE DE (STATUS, TOTAL BYTES)
        DEFW 0000           ;SAVE HL (POINTER)
;
        END  0000           ;END OF RNP...
```

APPENDIX F

I. Getting on line with TSS

The modem line can be in any one of several states, depending on what has happened since it was last used. The following routine will usually get the time sharing system connected, regardless of the state the line is presently in.

1) Turn on power to CPU and CONSOLE. (Green light on CPU should light if power is ok.)

2) Press lower push button on CPU panel to reset CPU. (Illustrated in App. A) The following message should appear on the console: Zee OS V1.2. If it does not appear, press button again.

3) You are now in the monitor program (described in Chapter 2). Next press "K" on the console keyboard. This should cause the CPU to enter the HOST I/O routine (described in Chapter 3). The following message should appear on the console: ZIO+. If it does not appear, go back to step 2.

4) You are now ready to connect to the TSS system. Press "CR" on the console. The TSS system may answer with the sign-on message. If so, continue as if you were on a dial-up terminal. (Refer to Honeywell Time-Sharing System Pocket Guide, BS12, page 13, TSS Terminal Operation.)

5) If there is no response from the previous input, press a "CTL A". The system may respond with: Program Name? If so, answer with TSS, and continue like a dial-up terminal.

6) If there is still no response, press the "ESC" key on the console and go back to step 4. This serves as the "break" key on this system.

7) If, after repeating the above procedure (from 4 to 6) several times, there is still no response from the HOST system, disconnect the power to the data modem and reconnect (this should reset the data line to the HOST system). Then go back to step 4 and start over.

8) If none of the above procedures get a response from the HOST system, then there is probably something wrong with the HOST system. In this case, you should check with the Computing Center on the status of TSS.

II. Loading and debugging programs

Since the HOST system does not require continous I/O to remain connected, you can jump from the I/O routine back into the monitor and then back into the I/O program and still remain connected to TSS. This capability allows you to load programs from the HOST, jump to the monitor for debugging and then jump back to the I/O program without disconnecting from TSS. However, you must remember that TSS will time out if it sees no input within 10 minutes. Therefore, you should go into the IDLE mode if you plan to be off longer, or sign off completely.

To load a program from the HOST, the program must be in absolute HEX format. (This format is explained in the Zapple Monitor Operating Manual.)

The following shows how to load a program from the HOST and then go to the monitor.

1) You must be in the I/O program and connected to TSS.
2) Press "CTL B" on the console and the console will return a ">".
3) You must now type in an offset value in HEX, or if no offset is required, just press "CR".
4) You now type in the name of the file which holds the assembled program, followed by a "CR". The console will then print <u>LIST <file>,</u> after which the HOST system will start sending the program code. The program code is both placed in memory and printed on the console so you can tell if an error occurs. (ERROR recovery will be explained later.)
5) After the program has been loaded, you may press "CTL N" to return to the monitor.
6) While in the monitor you may run the program just loaded, display the code, modify the code, etc. (All monitor functions are explained in Appendix B and the Zapple Monitor Operating Manual.)
7) When finished with the monitor, you press "K", which will go to the I/O program, and you can use the HOST system without signing back on to TSS.

ERROR recovery - if an error is detected during the the file transfer, the program will exit to the monitor routine and print "*" to indicate an error. Any non-Hex character detected during the transfer of a block or a bad checksum will cause an error exit. If an error is indicated it means you have returned to the monitor program. Therefore, you must re-enter the HOST I/O routine by entering a "K" from the keyboard before you can continue with TSS operations.

III. Creating and assembling microprocessor programs.

Programs for the microprocessor are written on the HOST system using the standard text editor. They must be written in standard ZILOG or MOSTEK Z-80 neumonics and include only the pseudo ops given in the assembler instruction manual (MOSTEK XFOR-80 CROSS ASSEMBLER MANUAL).

The file created with the editor is used as the input file to the assembler (XFOR-80), which is run using the standard FORTRAN system on the HIS 66/60. The input file number is 05, the line printer output file is 06, and the assembled Hex code file is 03. Two temporary files, 02 and 04, are also generated, but are of no use and need not be saved.

The following command is the format for assembling a file called MYSRC (Z80 SOURCE CODE), putting the listing into a file called MYLP (LINE PRINTER LISTING) and putting the assembled code into a file called MYCODE (Z80 HEX CODE).

```
RUNY Z80#MYSRC"05";MYLP"06";MYCODE"03"
```

The results of the assembly can be displayed on the console, using LIST or EDIT, to check for errors, etc.

Naturally, since the assembler is written in FORTRAN, it can also be run under batch and the I/O files can be any medium acceptable to the FORTRAN system. (For detailed information on the FORTRAN system, refer to Honeywell FORTRAN Pocket Guide, DD82.)

IV. Console Baud Rate Considerations

Since the I/O program is run in real time and uses no buffering, the console must not be run slower than the modem link to the HOST system. In fact, to prevent loss of characters, it should be faster than the modem speed (ie. if the modem is set at 1200 baud, run the console at 2400 baud).

The console baud rate is selected on the CPU by the rotary switch on the AUX card (illustrated in App. A). The setting here should match the setting on the rear of the console.

BIBLIOGRAPHY

1. Abrams, Blanc and Cotton, Computer Networks - A Tutorial, IEEE, 1975.

2. Amidon, Roger, The Zapple Monitor, Technical Design Labs, 1976.

3. Basket, F., "Open, Closed and Mixed Nets of Queues", Journal of ACM, COM 22, No. 2, April '75, pp. 248-260.

4. Booth, Taylor L., Digital Networks and Computer Systems, John Wiley & Sons, NY, 1971.

5. Brant, G. J.,"IEEE Transactions on Communication Techniques", Proceedings of IEEE, COM 17, No. 3, June '69, pp. 340-349.

6. Carlson, D. E., ADCCP, IEEE COMPCON 75, pp. 110-113.

7. Chang, J. H., IEEE Transactions on Communication, COM 20, No. 3, part II, June 72, pp. 619-629.

8. Coit, Kenneth T., "Programmable Multiline Communications Processor Provides Front-End Flexibility", Computer Design, May 1977, pp. 99-102.

9. Denning, Peter J. "Operating System Principles for Data Flow Networks", Computer, July '78, pp. 86-96.

10. Fratta, L. M., Networks, John Wiley & Sons, NY, 1973, pp. 97-133.

11. Gear, William C., Computer Organization and Programming, McGraw-Hill, Inc., 1969.

12. Gerla, M., PhD Dissertation, Department of Computer Science, UCLA, 1973.

13. Hirsch, Abe, "Minis Used as Data Interfaces Merit Multilevel Considerations", EDN, Jan. 5, 1978, pp. 61-67.

14. Honeywell Information System, FORTRAN Pocket Guide, DD 82, 1975.

15. Honeywell Information Systems, RNP/FNP Interface, DB 92 A, 1974.

16. Honeywell Information Systems, Time-Sharing System Pocket Guide, BS 12, 1974.

17. Lesea, Austin and Urkumyan, Nishan, "Multiplexer System Reduces Cost of Terminal Interfacing", Computer Design, Aug. 1977, pp. 109-113.

18. Leventhal, Dr. Lance A., "Cut Your Processor Computation Time", Electronic Design, Aug. 16, 1977, pp. 82-89.

19. Madnick, Stuart E. and Donovan, John J., Operating Systems, McGraw-Hill, Inc., 1974.

20. Mills, David L., "Executive Systems and Software Development for Minicomputers", Proceedings of IEEE, Nov. 1973, Vol. 61, No. 11, pp. 1556-1652.

21. Mostek Corporation, Mostek Z80 Technical Manual, 1977.

22. Mostek Corporation, XFOR-80 FORTRAN IV Cross Assembler, 1977.

23. Mostek Corporation, Z80 Programming Manual, 1977.

24. Muller, Donald J., "Microcomputers Decentralize Processing in Data Communication Networks", Computer Design, Oct. 1977, pp. 81-88.

25. Schoeffler, James D., Tutorial: Minicomputer Realtime Executives, COMPCON, Fall '74.

26. Schwartz, Mischa, Computer Communication Network Design and Analysis, Prentis-Hall, 1977.

27. Scrupski, Stephen E., "Communications Data - Handling Gains Flexibility", Electronics, July 11, 1974, pp. 88-91.

28. Ulrickson, Robert W., "Real-time Systems Often Use Interrupts", Electronic Design, May 10, 1977, pp. 80-84.

29. Villasener, Tony, "Need a multi-terminal interface? Try a microprocessor network", EDN, Oct. 5, 1977, pp. 63-68.