THE DESIGN AND IMPLEMENTATION OF A DOWN-LINE LOADER

THESIS

Presented to the Graduate Council of
Southwest Texas State University
in Partial Fulfillment of
the Requirements

For the Degree of
MASTER OF SCIENCE

By

Donald Dee Druckenbrodt
San Marcos, Texas
December 1977

TABLE OF CONTENTS

PREFACE

This thesis will discuss the design and implementation of a down-line loader. My motivation in selecting this topic has come from the desire to involve as many different facets of computer science as possible in one project. It has also provided the opportunity to investigate computer communications and to investigate two vastly different computer systems. But most important of all it has posed many problems from the use of a wide range of hardware and has required the use of varied programming techniques to solve these problems.

In many instances computer hardware is not used to its fullest potential or is acquired unnecessarily without first surveying all possible alternatives. It is the goal of this thesis to investigate a particular hardware related problem and propose possible solutions. Each solution will be described in detail and the advantages and disadvantages of each will be discussed.

While researching and writing this thesis there were many problems and setbacks encountered. I would like to sincerely thank those who have helped me to overcome the many obstacles to make this project possible. Their contributions, large and small, are very much appreciated.

I would especially like to thank my parents for their encouragement and continuous support and my wife for her endless patience and love.

D.D.D.

Southwest Texas State University
San Marcos, Texas
August, 1977

CHAPTER I

PRELIMINARY CONCEPTS

INTRODUCTION

The problem we will investigate involves the loading of a PDP-8/E which acts as a concentrator for a DEC-10 computer system. The PDP-8/E and DEC-10 are linked by a direct synchronous line over a distance of approximately one half mile. Acting as a concentrator, the operating system of the PDP-8/E places sixteen independent devices in contact with the Dec-10 over a single communications line. The operating system of the PDP-8/E periodically requires reloading due to system crashes or programming changes. Due to hardware constraints this operating system is currently being loaded by paper tape with an ASR-33 teletype which is quite slow and inconvenient to use. The only other input device available to the PDP-8/E is a card reader. Loading by cards has been performed in the past but was found to be inconvenient because the installation does not have a card punch capability and periodic software changes will not allow for a permanent card deck to be used. Acquisition of additional input devices for loading purposes will be ruled out due to cost.

The alternative methods of loading involve down-line loading schemes which do not require additional hardware and which provide excellent performance.

We intend then to discuss the design and implementation of down-line loaders, each implementation being limited to the use of existing hardware.

In the interest of notational simplification all decimal constants in the following text will be distinguished by the presence of a decimal point. All constants appearing without decimal points are to be considered octal constants.

THE DOWN-LINE LOADER

In general a loader is a program which accepts object code, places it in memory and initiates execution. Here, the loaders to be described are different in that they obtain object code over a communication link from another computer system. The phrase "down-line loader" implies that a particular situation exists. First, that communication exists between a larger host processor and a smaller target processor which has been dedicated to a specific task or is being used in a distributed data processing network. Second, communication has been interrupted due to the destruction of the operating system within the target processor and that loading is to occur from the host processor to the target processor which will reestablish the operating system of the target processor and restore normal

communications between the two.

Down-line loading requires that the target processor first be loaded by some means to reestablish communications. This initial program loading or bootstrapping should enable the target processor to communicate with the host processor and receive and load object programs sent by the host processor. Once the target processor's operating system has been restored, normal operation can resume.

The down-line loading idea can also be applied in other situations where any independent target processor may not have a desirable program storage capability. In many instances processors are used to monitor or control mechanical devices under varying circumstances. In some cases the infrequent need for loading software can not justify the cost to acquire additional hardware for this purpose. If communications could be established with a host processor a loading procedure could be developed which would allow the target processor to utilize the storage capability of the host and in many cases decrease the time required to access and load programs. Therefore, the target processor would not require any auxiliary storage capability other than that required for the initial program loader, which could take the form of a hardware bootstrap loader or a PROM. All other software such as assemblers, editors, or utilities could reside with the host and be loaded only as required.

OVERVIEW

In the following chapters two down-line loading schemes will be discussed. The first involves establishing communications between the two central processors over a telephone line using the host's dial-up capability and a standard telephone with an acoustic coupler. In normal use the dial-up feature of the DEC-10 is used to provide communication with a remote terminal device by telephone which is dialed to establish contact with the DEC-10. The telephone receiver is then placed in the acoustic coupler, which is used to convert analog signals on the telephone line to serial digital pulses. The terminal device, which requires serial digital pulses, can be plugged into the acoustic coupler's output and placed in contact with the DEC-10. The host will output a requested object code file over the telephone line to the target processor which is connected to the coupler and programmed to receive, format, and load the object code it receives.

The second scheme involves the use of the direct synchronous line as a communications link between processors. The DEC-10 uses the DS-10 synchronous line unit to handle the transmission and reception of serial data over one full-duplex, synchronous communications line at speeds of 600. to 20,000. bits per second. The DS-10 transfers full data words between itself and the DEC-10 under program control via the I/O bus. On the PDP-8/E end of the synchronous line the DP8-E synchronous modem interface is

the counterpart of the DS-10 for the transmission and reception of data. The DP8-E can also access memory in the PDP-8/E directly without program control. For example, the transmission of a message requires only the initialization of the first address of the message and the number of characters to be sent. When placed in transmit mode the DP8-E automatically references the message directly from memory to the synchronous line without program intervention.

The second scheme, is concerned primarily with the transmission and the reception of data through the DS-10 and the DP8-E .

THE PDP-8/E

Before we can discuss the details of the loading schemes it will be necessary to describe some particular features in the architecture of the PDP-8/E. The organization of memory will be the primary concern.

The memory of the PDP-8/E is divided into 4K (4096. byte) fields. A three bit register is used to specify which particular field is being accessed at the current time. This allows up to eight fields or 32.K words of memory. There are two special three bit registers used to specify fields. The instruction field register determines the memory field from which instructions are executed and the memory field from which the operand of a directly addressed instruction is taken. The data field register determines the memory field from which an indirectly addressed operand

is taken.

Whenever it becomes necessary to change the instruction or data fields, a field change instruction must be executed to accomplish the change. To change the current instruction field, we must execute an instruction field change instruction (CIF). The current instruction field remains in effect until a jump indirect instruction is executed, at which time the program counter is set to the specified operand within the new field. To change data fields, we must execute a data field change instruction (CDF). The current data field is changed immediately upon execution of the CDF.

Each of the fields is divided into 32. logical pages, each page containing 200 locations. The purpose behind the page structure can be seen when examining the format of the memory reference instructions (MRI). Memory reference instructions are twelve bits in length as are all other PDP-8/E instructions. The first three bits of an MRI are used to specify the op code. The fourth bit is used to indicate the use of direct or indirect addressing. The fifth bit is used to indicate whether the operand is on the current page or on page zero which are the only pages that may be referenced directly; all others must be referenced indirectly. The last seven bits are page address bits which allow for a relative address from 0 to 177 on each page. The paging scheme then allows for the referencing of 4K of memory using a nine bit operand rather than a twelve bit

operand. Therefore without the paging scheme the word size would be fifteen bits rather than twelve.

In addition to memory reference instructions there are operate microinstructions which allow the programmer to manipulate or test data stored in a twelve bit general purpose register referred to as the accumulator and a one bit register referred to as the link. The link is logically attached to the beginning of the accumulator.

CHAPTER II

TELEPHONE LINE LOADERS

Our first loading scheme involves communication between the two central processors over a telephone line at a rate of 300. baud. Here, as described earlier, we connect the PDP-8/E to the DEC-10 via a telephone link. We will have the DEC-10 output data to the PDP-8/E as if the PDP-8/E were a remote terminal programmed to receive, format, and load data as required.

We establish contact with the DEC-10 by dialing in on the telephone. Using the acoustic coupler and a terminal we can login and reach monitor level. We then execute a program on the DEC-10 which is used to access disk files that contain PDP-8/E object code and to output these files to the PDP-8/E. Upon execution, this program waits for an ACK (positive acknowledgement) character from the PDP-8/E before output of the desired object code begins. At this point the terminal is disconnected from the coupler in order to connect the PDP-8/E which is assumed to be in an unloaded state. Therefore the loader to be used to accept object code from the DEC-10 must itself be loaded into memory. Fortunately, the PDP-8/E has an M18-E hardware bootstrap loader which has a capacity of 32. twelve bit words. The

M18-E can load an initial memory address and instruction field to be used, deposit 32. words sequentially into memory, load a memory start address, and begin execution of the loaded program. Due to the size restriction placed on the loader by the M18-E, it is not possible to encode a 32. word loader capable of acknowledging the DEC-10 and receiving and loading binary-formatted object code with the attendant checksumming and handshaking. It is possible, however, to write a loader to send an ACK to the DEC-10 and receive and load RIM (read in mode) formatted object code without the checksumming and handshaking being done.

RIM format requires that each instruction in object code be preceded by the address at which the instruction is to be loaded. Binary format is similar to RIM format except that only the initial address in a series of consecutive instructions is specified. The binary loader must then recognize new addresses or increment the current address when loading binary-formatted files. RIM format allows for a simpler loader because of the regular address/instruction order. However, the additional addresses added to the code being loaded will increase load time considerably. It is desirable for the RIM format loader to load another larger loader capable of loading binary formated files. We refer to this first RIM format loader as a TRIM (telephone read in mode) loader.

TRIM LOADER

The TRIM loader must satisfy the following requirements. First it must send an ACK to the host to initialize output. As input is received from the host, the TRIM loader must distinguish between an address, an instruction to be loaded, a data field change, or a leader-trailer, and must take the necessary steps to see that each instruction is loaded at the proper address and in the correct data field. Leader-trailer codes are placed at the beginning and the end of the object code produced by the assembler and will be discarded if received by the TRIM loader.

Each PDP-8/E address and instruction is twelve bits in length. Data sent over the telephone line by the DEC-10 is in groups of eight bits requiring each PDP-8/E address or instruction to be sent in two parts, each containing two identification bits and six data bits. A field change or a leader-trailer requires only six bits and may be sent in one part.

The two identification bits are used only in the first half of the instruction or address that is sent. The first two bits of a required second half are always off. Each eight bit group has the following identification bit representation.

TYPE	ID BITS
Instruction	0 0
Address	0 1
Leader-trailer	1 0
Field change	1 1

As each first half is received it is identified. If an address is identified the second half is received and the full address is formed and placed in a location reserved for the temporary storing of current addresses. If an instruction is received the full instruction is formed after receiving the second half and the instruction is loaded indirectly on the current address. When a field change is encountered a data field change instruction to field one is always executed. Data field zero is initialized by the M18-E. If size restrictions had not been placed on the TRIM loader, the six data bits received with the field change identification bits would be used to form a CDF instruction so that a change to any specified field could be made. Any leader-trailer that is received is ignored by the TRIM loader.

The following is a flowchart of the TRIM loader. Note that the algorithm has no logical end. All RIM object code that is loaded overlays the TRIM loader to cause a transfer to the start address of the loaded object code.

TRIM FLOWCHART

(T0) Send an ACK to the host

(T1) If a character has been received from the host THEN (T2) ELSE (T1)

(T2) Place the character in the AC (accumulator)

(T3) Place the character in the low half of the AC and the first ID bit in the LINK (rotate the AC left 5 bits)

(T4) IF bit 0 is set THEN (T5) ELSE (T10)

(T5) IF the link is set THEN (T13) ELSE (T6)

(T6) Execute change to data field 1

(T7) GO TO (T1)

(T10) IF LINK is set THEN (T11) ELSE (T13)

(T11) This character is a leader-trailer code so disregard

(T12) GO TO (T1)

(T13) Place the second ID bit in the LINK (rotate AC left one bit)

(T14) IF a character has been received from the host THEN (T15) ELSE (T14)

(T15) Place the character in the high half of the AC

(T16) IF LINK is set THEN (T17) ELSE (T21)

(T17) TEMP = AC

(T20) GO TO (T1)

(T21) LOC(TEMP) = AC

(T22) GO TO (T1)

THE BINARY LOADER

After the TRIM loader has been placed in the memory of the PDP-8/E by the M18-E the PDP-8/E is able to load the binary loader. The TRIM loader, although used here to load the binary loader, could be used to load any RIM formatted file without having to use the binary loader. Loading the binary loader in order to load a binary-formatted file may not be as fast as loading that same file in RIM format using the TRIM loader. A relatively short file could be loaded much faster in RIM format. The loading of a large file however, would be much slower using RIM format due to the time required to output and process the additional addresses. This time exceeds the time required to load the binary loader and then load the same file using binary format. Also the possibility of extraneous characters being placed in the transmission line, referred to as line noise, is increased as load time increases and could result in an improper load using the TRIM loader.

Earlier it was mentioned that the host processor would execute a program to output desired object code to the target processor. The TRIM loader by design will generally load the binary loader. The host program, referred to as RIMPRT must then access and output the binary loader in RIM format when an ACK is received from the TRIM loader. For convenience, all PDP-8/E object code is assumed to be in binary format which will require RIMPRT to change binary-formatted code to RIM format before output. This

requirement is easily implemented and does not increase load time due to the slow transmission rate of the telephone line. TRIM and RIMPRT are always used together to load RIM formatted code.

RIMPRT FLOWCHART

(R0) Open channel for input

(R1) Open channel for output

(R2) Strip off leader-trailer codes from object code

(R3) Read next character

(R4) IF character is leader-trailer THEN (R22) ELSE (R5)

(R5) IF character is field change THEN (R6) ELSE (R10)

(R6) Output character to target

(R7) GO TO (R3)

(R10) IF character is a address THEN (R11) ELSE (R13)

(R11) Read second character and form current address

(R12) GO TO (R3)

(R13) Separate current address into two bytes

(R14) Output both address bytes

(R15) Output current character

(R16) Read second half of the instruction

(R17) Output second half

(R20) Increment current address

(R21) GO TO (R3)

(R22) END

The binary loader must be able to request and load any binary formatted file that the host may have stored on disk. To enable the user to request files in a convenient manner, the binary loader should also include a device driver to allow for user I/O with the host. The binary loader should also perform error checking and have the ability to request retransmission of data when errors are detected. If larger programs are to be loaded, the binary loader should be as small as possible.

Because of these constraints the binary loader, which is referred to as IMPORT, has two modes of operation: communications mode and load mode. Communications mode provides a device driver that allows the user to use a terminal device connected to the PDP-8/E to access the host via the telephone link. Any character typed on the user's console is accepted by IMPORT and sent to the host. The host receives the character and generates an echo of that character. IMPORT receives the character that has been echoed and sends it to the user's console. During communications mode the user has full use of the facilities of the host processor.

At some point the user can request that a binary file be loaded from the host. This is done by the execution of a host program similar to RIMPRT. This program, called EXPORt, can access binary-formatted object code and output it in that form. When the particular object code file has been determined, EXPORT sends a control character that

IMPORT will recognize as the beginning of a load sequence and will enter load mode. When load mode has been initiated IMPORT will not output to the user's console, but will accept all data as object code to be loaded. Since error checking is to be done, EXPORT sends data in blocks containing a checksum and a word count in addition to the object code. Should an error in transmission occur, IMPORT requests EXPORT to retransmit the last block of data until it is received correctly or until a predetermined error count is exceeded. EXPORT sends data in the same manner as RIMPRT, eight bits at a time. The first two bits are used for identification, the last six bits for data. Since object code is sent in blocks containing control information, IMPORT always considers the first eight bits of each block to be a checksum, the second eight bits to be a word count, and the remainder to be object code to be identified by their identification bits.

There may be cases that require object code to be loaded into memory where IMPORT is currently executing. A buffer area the size of a particularly vulnerable part of IMPORT has been set aside in the upper portion of the second 4K field. Code that is to be loaded in the same memory locations as IMPORT is placed in the corresponding relative locations in this temporary buffer.

At the conclusion of each load EXPORT sends another control character to terminate load mode, writes a load summary, and places the user terminal back into

communication with the host. At this point the user may load additional object code, use the facilities of the host, or dump the temporary buffer overlaying IMPORT and begin execution of the loaded code. To dump the temporary buffer the user must type a control G, which IMPORT will recognize from communications mode to cause a transfer of control to a dump routine located near the buffer. This routine will dump the buffer and print a "load completed" message on the user's console.

The following are flowcharts of IMPORT and EXPORT.

IMPORT FLOWCHART

(I0) Initialize variables

(I1) IF an interrupt occurs THEN (I2)
ELSE (I1)

(I2) IF an interrupt from the host to user device occurs THEN (I7)

(I3) IF an interrupt from the user's keyboard occurs THEN (I5)

(I4) IF an interrupt from the user's printer occurs THEN (I15) ELSE (I1)

(I5) Read character from user keyboard and send to the host

(I6) GO TO (I1)

(I7) Read character from the host

(I10) IF in load mode THEN (IL0)

(I11) IF character is dump control character THEN (ID0)

(I12) IF load mode control character THEN (I13)
ELSE (I15)

(I13) Set load mode

(I14) GO TO (I1)

(I15) Write character to buffer

(I16) IF output to user console is in progress THEN (I1) ELSE (I5)

(I17) Output a character from the buffer to the user printer

(I20) GO TO (I1)

(IL0) IF byte flag set THEN (IL26)

(IL1) IF word count has been received then (IL6) ELSE (IL2)

(IL2) IF checksum has been received THEN (IL5)

(IL3) Initialize checksum

(IL4) GO TO (I1)

(IL5) Initialize word count

(IL6) GO TO (I1)

(IL7) IF the character is the first half of an instruction THEN (IL10) ELSE (IL14)

(IL10) Move the character to low half of AC and store

(IL11) Increase checksum

(IL12) Set byte flag

(IL13) GO TO (I1)

(IL14) IF the character is the first half of an address THEN (IL15) ELSE (IL17)

(IL15) Set address flag

(IL16) GO TO (IL10)

(IL17) IF the character is a field change THEN (IL20) ELSE (IL21)

(IL20) Set new data field

(IL21) GO TO (IL45)

(IL22) IF the character is a terminate load code THEN (IL23) ELSE (IL25)

(IL23) Clear load mode

(IL24) GO TO (I1)

(IL25) The character is trash so ignore

(IL26) GO TO (I1)

(IL27) Turn off byte flag

(IL30) Increase checksum

(IL31) Join bytes to form word

(IL32) IF address flag set THEN (IL33)
ELSE (IL36)

(IL33) Clear address flag

(IL34) Set CA (current address)

(IL35) GO TO (IL45)

(IL36) IF this data should be buffered THEN (IL37)
ELSE (IL41)

(IL37) Store the word in the buffer

(IL40) GO TO (IL44)

(IL41) Select the current data field

(IL42) LOC(CA) = WORD

(IL43) Restore original data field

(IL44) Increment current address

(IL45) Increment word count

(IL46) IF this is the last word THEN (IL47)
ELSE (I1)

(IL47) IF checksum is correct THEN (IL50)
ELSE (IL52)

(IL50) Send an ACK to the host

(IL51) GO TO (I1)

(IL52) Send retransmit to the host

(IL53) Clear line

(IL54) IF line is clear THEN (I1) ELSE (IL53)

(ID0) Dump the buffer

(ID1) Print "LOAD COMPLETE"

(ID2) GO TO the initial starting location of the loaded object code.

EXPORT FLOWCHART

(E0) Open channel for input

(E1) Open channel for output

(E2) Strip off leader-trailer codes from object code

(E3) Read next character and store

(E4) Read second half of address and store

(E5) Initialize checksum and word count

(E6) Read next character

(E7) IF character is a leader-trailer THEN (E30)

(E10) IF character is an address THEN (E11) ELSE (E16)

(E11) Send current buffer

(E12) IF reply from target is an ACK THEN (E13) ELSE (E11)

(E13) Store address half

(E14) Read second half of address, form full address, and store

(E15) Initialize checksum and word count

(E16) Strip off consecutive addresses and re-initialize if necessary

(E17) GO TO (E6)

(E20) IF word count is equal to the buffer size THEN (E21) ELSE (E25)

(E21) Send the current buffer

(E22) IF reply from target is ACK THEN (E23) ELSE (E21)

(E23) Store current address

(E24) Re-initialize checksum and word count

(E25) Store character in buffer

(E26) Increment word count

(E27) GO TO (E6)

(E30) Output current buffer

(E31) Terminate load mode

(E32) Output load summary

CHAPTER III

SYNCHRONOUS LINE LOADER

Our second loading scheme involves the use of both the telephone line communications link and a direct synchronous line communications link. Because the M18-E has been configured to receive telephone line communication, it is again used to initially load a larger binary loader. Therefore, the TRIM loader and RIMPRT are used in the same manner as described earlier. In this case however, the TRIM loader is to load a binary loader which communicates with the host processor via a synchronous communications line at a rate of 9600. baud. Due to the high speed data tranfer this binary loader can load object code from the host 32. times faster than IMPORT.

THE BINARY LOADER

The use of this high speed communication link has caused this second binary loader, called SYNC, to be quite different from IMPORT. Because of the data transfer speeds involved, characters sent from the host are not processed as they are received. The PDP-8/E is only able to receive data over the synchronous line through the use of the DP8-E synchronous modem interface. The DP8-E accesses the memORY

of the PDP-8/E directly by taking control of the CPU when a character is to be sent or received over the synchronous line. When transmitting or receiving a character, the DP8-E references two particular memory locations that have been initialized under program control. These two locations, current address (CA) and word count (WC), are used as registers. When transmitting, the CA register is used as a pointer to the address of the next character to be referenced and sent out on the line. When receiving, the CA register is used as a pointer to the address of the next location to be filled by an incoming character. In both cases the CA register is first incremented by the M18-E and then used as an address. Therefore the CA register must be initialized to point to the first location before the message buffer. Under program control the WC register is set to the two's complement form of the number of characters to be received or sent. As each character is transmitted on the line, the WC register is incremented by one by the DP8-E. When the WC register becomes zero a flag is set by the DP8-E that is detected by SYNC.

SYNC has been designed around the operation of the DP8-E. Incoming blocks of data are received in their entirety and then processed. The construction of data blocks sent by the host must be modified due to the requirement to initialize the WC register. SYNC receives data blocks in two parts. The first part is called the "header". Each header is of the same fixed length so that

the WC register can be initialized consisently. The CA register is always set to the memory location one address before the first location of the message buffer. Contained within the header is a word count which is used to reinitialize the WC register to enable SYNC to receive the variable length block to follow, referred to as "text." Also included in the header is the current data field, the current load address, and a checksum for the header. After receiving the header the WC register is reinitialized so that the following text block can be recived. The text block consists of the binary-formatted object code followed by a checksum for the text block.

In an effort to decrease the size of this loader and to make the loading of a single file faster and easier, the communications mode used in IMPORT is not implemented in SYNC. In the case when only a single file is to loaded and the facilities of the host, other than loading functions, are not to be used the communications mode becomes an unnecesary step in the loading process. Currently communications mode allows the user to execute EXPORT in order to output a specified object file. By combining the functions of RIMPRT and EXPORT into one program the object file can be specified prior to loading and eliminate the need to pause between the execution of RIMPRT and EXPORT. Using SYNC we first make contact with the host by telephone using the ascoustic coupler and a terminal. We then execute the counterpart of the combined programs, RIMPRT and EXPORT.

This program, called LDSYNC, requests the user to specify the object file to be loaded and then waits for an ACK from the TRIM loader. At this point the terminal is disconnected from the coupler and the PDP-8/E is plugged into the coupler. The M18-E is started causing the TRIM loader to be brought into memory, and the TRIM loader execution begins with the RIMPRT portion of LDSYNC. Once SYNC is brought into memory by the TRIM loader and begins execution, the counterpart of EXPORT within LDSYNC is already waiting for a load request from SYNC to begin output of the specified object file.

LDSYNC will output the object file in two-part blocks via the DS-10 synchronous line unit. Data is transferred between LDSYNC and the DS-10 in 36. bit words via the I/O bus. Since each character unit is eight bits in length, four characters are contained in each word. The remaining four bits per word are ignored by the DS-10. Therefore, under program control LDSYNC must group four eight bit characters into one word from the object file and output the word to the DS-10. The DS-10 in turn receives the word from LDSYNC, sends it out on the synchronous line, and signals LDSYNC when it is ready for the next word. When messages are received from the PDP-8/E, the same process occurs in the reverse direction. LDSYNC continues to send two-part blocks to SYNC until the entire file has been sent. Should a transmission error occur during the loading of a particular block, SYNC will detect the error through

checksumming and request LDSYNC to retransmit the last block of data until it is received correctly or until an error count is exceeded. The last instruction sent by LDSYNC overlays a SYNC instruction and causes a tranfer to the starting address of the loaded object file.

Under normal system operation the DS-10 is accessed only by the DEC-10 monitor and is restricted from user access. The DS-10 can only be accessed if changes are made to the monitor code. Due to the complexity of the monitor operating system and the expertise required to make specific changes where needed, the synchronous loader has not been implemented. The source listings for the algorithms discussed have been written, though, and appear in the appendix. Routines for the host which involve I/O with the DS-10 have been omitted.

The following is a flowchart of SYNC. A flowchart of LDSYNC will not be given due to its close similarities to RIMPRT and EXPORT.

SYNC FLOWCHART

(S0) Send a load request to host

(S1) Set header WC and CA

(S2) Receive the header block

(S3) Calculate the header checksum

(S4) IF header is correct THEN (S6)
ELSE (S17)

(S5) Send ACK to the host

(S6) Receive the text block

(S7) Calculate the text checksum

(S10) IF the text is correct THEN (S11) ELSE (S17)

(S11) Set the current data field

(S12) Set the current load address

(S13) Form instructions from the adjacent bytes and load into memory

(S14) IF finished processing the buffer THEN (S15) ELSE (S13)

(S15) Send ACK to the host

(S16) GO TO (S1)

(S17) Send NAK to the host

(S20) GO TO (S1)

CHAPTER IV

CONCLUSIONS

In summary, we have discussed the design and implementation of two down-line loading systems. The need for these systems came from the requirement of a target processor to be loaded by a more efficient means, without acquiring additional hardware. The existing hardware environment provided communication facilities to a host processor and therefore the basic elements existed for the development of a down-line loader.

Chapter I described the current operating environment of the target processor. Here it was suggested that the use of a down-line loader would provide the most acceptable solution for the loading requirement of the target processor. After the preliminary concepts of a down-line loader were defined, the two individual implementations were outlined. Chapter I concluded with a description of certain features of the PDP-8/E which would be required later.

Chapter II dealt with the detailed explanation of the first implementation of the loader using the telephone line communication link. Also included was a description of the use of the M18-E hardware bootstrap loader to resolve the initial program load problem as well as a description of the

algorithms used to accomplish loading and the data transfer between processors.

Chapter III illustrated the use of the TRIM loader to load a larger binary loader. The binary loader described here, however, was of a much different design due to the use of the direct synchronous line communication link. The elimination of a communications mode in the SYNC and the combination of RIMPRT and EXPORT into one program to improve the loading process was also detailed.

If further information is required regarding the DEC-10, the PDP-8/E, or general information about loaders the reader is referred to the sources listed in the bibliography.

APPENDIX A

PROGRAM LISTINGS

(1) TRIM.PAL

```
          TTYTLS=6656
          TTYKCC=6642
          TTYKRS=6644
          TTYTSF=6651
          TTYKRB=6646
          TTYKSF=6641
          CDF1=6211

          FIELD 0

*7740
START,    TAD TMP          /GET ACK
          TTYTLS           /SEND ACK
X0,       TTYKCC           /CLEAR FLAG AND AC
X1,       TTYKSF           /SKIP IN FLAG SET
          JMP X1
          TTYKRB           /CLR AC, READ FIRST HALF
          CLL RTL          /ROTATE
          RTL              /5 BITS
          RAL              /LEFT.
          SMA              /SKIP MINUS AC
          JMP X2           /   POSS. FLD OR ADDR.
          SNL              /SKIP NON-ZERO LINK
          JMP X3
          CDF1             /CHANGE TO DATA FIELD ONE
          JMP X0
X2,       SZL              /SKIP ZERO LINK :DATA
          JMP X0           /  ELSE LEADER TRAILER
X3,       RAL              /ROTATE ONE BIT LEFT
          TTYKSF           /SKIP IN FLAG SET
          JMP.-1
          TTYKRS           /READ SECOND HALF STATIC
          SNL              /SKIP IF NOT ADDRESS
          DCA I TMP        /MOVE AC TO ADDRESS TMP
          DCA TMP          /MOVE AC TO TMP
          JMP X0           /GET NEXT WORD
TMP,       25
$
```

(2) IMPORT.PAL

```
          FIELD 0
*0
          0
          JMP I 2
          SKPCH
          HALT=7402
          CTYKSF=6031      /DEFINE IOTS
          CTYTCF=6042
          CTYKRB=6036
          CTYTLS=6046
          CTYTSF=6041
          TTYKSF=6641
          TTYTCF=6652
          TTYKRB=6646
          TTYTSF=6651
          TTYTLS=6656
          CDF0=6201
          CDF1=6211
          CIF1=6212
          CIF0=6202
START,    CLA
          TAD INI
          DCA NC
          TAD INI
          DCA FS
          DCA OUTFLG
          ION
SELF,     JMP SELF
SKPCH,    TTYKSF           /LOCATE INTERRUPT
          SKP
          JMP TTYMD
          CTYTSF
          SKP
          JMP CTYOUT
          CTYKSF
          SKP
          JMP CTYMD
          TTYTCF
          JMP EXIT
CTYMD,    CTYKRB           /SERVICE CTY
          TTYTLS
          JMP EXIT
TTYMD,    TTYKRB           /SERVICE TTY
          DCA I TAC
          TAD LOAD         /IS LOAD MODE SET?
          SZA CLA
          JMP I LDMDD      /YES, GO TO LOAD
          TAD BELL         /RECEIVED BELL?
          TAD I TAC        /TYPE ^G TO
          SZA CLA          /DUMP BUFFER
          JMP NDMP
GDMP,     CIF1
```

```
        JMP I DMP2      /JUMP TO DUMP
NDMP,   TAD I TAC       /ELSE
        TAD I LTT       /IS IT LEADER TRAILER?
        SZA CLA
        JMP X
        IAC             /YES, SET LOAD MODE
        DCA LOAD
        JMP EXIT
X,      TAD I TAC       /NO, PUT CHAR IN BUFFER
        DCA I FS
        ISZ FS          /INC FREE SPACE PTR.
        TAD FS
        CIA
        TAD END         /COMPARE FS TO END
        SZA CLA
        JMP CLR
        TAD CTRLS       /IF FULL SEND CTRL-S
        TTYTLS
        JMP EXIT
CLR,    TAD OUTFLG
        SNA
        JMP OUTSET
        JMP EXIT
OUTSET, CLA CMA
        DCA OUTFLG
CTYOUT, CLA             /OUTPUT THE CTY BUFFER
        TAD INI
        CIA
        TAD FS          /ARE THERE CHAR'S
        SZA CLA         /WAITING IN CTY BUFFER?
        JMP WRITE
        CLA
        DCA OUTFLG
        CTYTCF
        JMP EXIT
WRITE,  TAD I NC        /OUTPUT CHAR
        CTYTLS
        ISZ NC          /INC POINTER
        CLA
        TAD NC
        CIA
        TAD FS          /DONE?
        SZA
        JMP EXIT
        TAD INI
        DCA NC
        TAD INI         /INITIALIZE POINTERS
        DCA FS
        TAD CTRLQ       /SEND CTRL-Q
        TTYTLS
EXIT,   ION             /TURN ON INTERRUPT
        JMP I 0         /RETURN
TAC,    AC
LOAD,   0
LDMDD,  LDMD
```

```
OUTFLG, 0
NC,     BUFFER
FS,     BUFFER
INI,    BUFFER
END,    BUFEND-10
CTRLS,  23
CTRLQ,  21
DMP2,   DUMP
BELL,   -7
LTT,    LT
BUFFER, 0
        *177
BUFEND, 0


*200
LDMD,   TAD BYTFLG
        SZA CLA          /IS BYTFLG SET?
        JMP LOADWD
        TAD WRDCNT       /HAS WRDCNT BEEN SET
        SNA CLA
        JMP LOADINI      /NO, THEN INITIALIZE
        TAD AC           /ELSE
        AND (300
        SNA              /IS IT DATA?
        JMP HALF
        AND (200
        SNA CLA          /IS IT ADDRESS?
        JMP ADDR
        TAD AC
        AND (100         /IS IT FIELD CHANGE?
        SZA CLA          /IS IT LEADER TRAILER?
        JMP FLDCHG
        TAD AC           /TERMINATE LOAD?
        AND (77
        SZA
        JMP EXIT         /TRASH! SO DISMISS
        DCA LOAD         /GO INTO COMMUN. MODE
        JMP START
FLDCHG, TAD AC
        AND (77
        DCA CCDF         /SET CURRENT FIELD
        JMP CHECK+1
ADDR,   IAC
        DCA ADRFLG       /SET ADDRESS FLAG
HALF,   TAD AC
        TAD SUM          /INCREASE CKSUM
        DCA SUM
        TAD AC
        AND (77
        CLL RTL
        RTL
        RTL
        DCA BYTE         /PUT AC IN HIGH BYTE
```

```
        IAC
        DCA BYTFLG      /TURN ON BYTE FLAG
        JMP EXIT
LOADINI,TAD CKSUM       /HAS CKSUM
        SNA CLA         /BEEN ENTERED
        JMP CK
        TAD AC
        CIA
        DCA WRDCNT      /ENTER WORD COUNT
        JMP EXIT
CK,     TAD AC
        TAD (100        /MAKE CKSUM NON-ZERO
        DCA CKSUM       /ENTER CHECK SUM
        JMP EXIT
LOADWD, DCA BYTFLG      /TURN OFF BYTE FLAG
        TAD AC
        TAD SUM         /INCREASE SUM
        DCA SUM
        TAD AC
        TAD BYTE        /JOIN BYTES TO
        DCA BYTE        /FORM WORD
        TAD ADRFLG      /IS ADDRESS FLAG SET?
        SNA CLA
        JMP DATA
        TAD BYTE        /SET LOCATION
        DCA LC          /COUNTER TO NEW ADDRESS
        DCA ADRFLG      /CLEAR ADDRESS FLAG
        JMP CHECK
DATA,   TAD LC          /DOES DATA NEED
        AND (7400       /TO BE BUFFERED
        SZA CLA
        JMP TEST
        TAD CCDF
        SNA CLA
        JMP STORE
TEST,   TAD CCDF
        TAD CODE
        DCA OP          /SELECT DATA FIELD
OP,     0
        TAD BYTE
        DCA I LC        /LOAD WORD
        CDF0            /RESTORE DATA FIELD 0
        JMP NEXT
STORE,  TAD LC
        TAD (ZDUMP-1    /DETERMINE
        DCA LC          /BUFFER ADDRESS
        TAD BYTE
        CDF1
        DCA I LC        /PUT IT IN BUFFER
        CDF0
        TAD (ZDUMP-1
        CIA
NEXT,   TAD LC          /INCREMENT LC
        IAC
        DCA LC
```

```
CHECK,  ISZ WRDCNT      /INC AND CHECK
        ISZ WRDCNT      /WORD COUNT
        JMP EXIT
        CLA CLL
        TAD SUM         /COMPARE CHECKSUM
        AND (77
        DCA SUM
        TAD CKSUM
        AND (77
        CIA
        TAD SUM
        DCA SUM         /PLACE DIFFERENCE IN SUM
        CIF1
        JMP REPLY
BYTFLG, 0
CKSUM,  0
ADRFLG, 0
SUM,    0
WRDCNT, 0
LC,     0
BYTE,   0
LT,     -201
AC,     0
CODE,   6201
DMP,    DUMP
CCDF,   0
        FIELD 1
*7200
ZDUMP,  0


                        /THIS ROUTINE CLEARS
                        /THE LINE WHEN NOISE
                        /IS DETECTED.
*7600
REPLY,  TAD I XSUM
        SNA CLA         /IS CKSUM RIGHT?
        JMP ACK
CLEAR,  TAD NAKK        /GET NAK
        TTYTLS          /SEND NAK
        TTYKSF          /READY ?
        JMP.-1
        TTYKRB          /READ
        CIA
        TAD SYNC        /HAS SYNC
        SZA CLA         /BEEN READ
        JMP CLEAR+2     /NO
        JMP NR          /YES, RETURN
SYNC,   26
NAKK,   25
RET,    EXIT
ACK,    TAD ACKIT       /GET ACK
        TTYTLS          /SEND ACK
NR,     CLA CLL
```

```
        CDF0            /DATA FIELD 1
        DCA I XCKSUM    /CLEAR CKSUM
        DCA I XWRDCT    /      WRDCNT
        DCA I XSUM      /      SUM
        CIF0            /INSTR. FIELD 0
        JMP I RET       /RETURN
ACKIT,  6
XCKSUM, CKSUM
XWRDCT, WRDCNT
XSUM,   SUM

DUMP,   CDF1            /DATA FIELD 1
        TAD I PTR
        ISZ PTR         /THIS ROUTINE DUMPS
        ISZ LOC         /THE BUFFER AT *7200
        CDF0            /TO PAGES 0 AND 200.
        DCA I LOC       /DUMP INSTR.
        ISZ SIZE
        JMP DUMP
        CDF1            /DATA FIELD 1
        CTYTLS          /PRINT MESSAGE TO
XX,     TAD I DONE      /CTY AFTER DUMP
        CTYTSF
        JMP.-1
        CTYTLS
        ISZ DONE
        CLA CLL
        ISZ CNT
        JMP XX
        CDF0
        CIF0
        JMP I DC72
PTR,    ZDUMP
LOC,    0
CNT,    -10
MSG,    314             /L O A D E D <CR> <LF>
        317
        301
        304
        305
        304
        215
        212
DONE,   MSG
SIZE,   -400
DC72,   200
$
```

(3) RIMPRT.BLI

```
MODULE RIMPRT(STACK(300),TIMER=EXTERNAL(SIX12))=
BEGIN

EXTERNAL OPEN,WRITE,WRITES,READ,FORCEOUT,LOOKUP,
         CLOSE,ENTER;

REQUIRE TTCALL.BLI;

BIND TTCHAN=1,  ICHAN=2, ACK=6, NAK=#25,
        MAXNAK=100, MAXTRASH=100;

MACHOP JRST=#254, CALLI=#047;

MACRO HALT=JRST(4)$, RESET=CALLI(0)$;

OWN     OBUF[3],
        IBUF[3],
        MESSAGE[260],
        FILESPECS[4],
        REPLY,
        NAKCNT,
        TRASHCNT,
        HIGH,
        LOW,
        HALF,
        ADDR;

BIND CKSUM=MESSAGE[0], WRDCNT=MESSAGE[1];

REGISTER WORD;

BIND    ERRTMN=0,
        ERRTMT=1;



ROUTINE OUTN(NUM,BASE,REQD)=
BEGIN
        OWN N,B,RD,T;
        ROUTINE XN=
                BEGIN LOCAL R;
                IF .N EQL 0 THEN RETURN
        (DECR I FROM (.RD-.T-1) TO 0 DO
                 WRITE(TTCHAN,"0"));
                R_.N MOD .B; N_.N/.B; T_.T+1; XN();
                WRITE(TTCHAN,.R+"0")
                END;

        IF .NUM LSS 0 THEN WRITE(TTCHAN,"-");
        B_.BASE; RD_.REQD; T_0; N_ABS(.NUM); XN()
        END;
```

```
LABEL   LOOP,LOOP2;

!!! MAINLINE !!!

RESET;
IF NOT OPEN(ICHAN,#10,SIXBIT 'DSK',IBUF<0,0>)
THEN (OUTS('???GCOULD NOT OPEN  DEVICE');CRLF;HALT);
FILESPECS[0]_SIXBIT 'IMPORT';
FILESPECS[1]_SIXBIT 'BIN';
FILESPECS[2]_FILESPECS[3]_0;
IF NOT LOOKUP(ICHAN,FILESPECS)
    THEN (OUTS('???GLOOKUP FAILED');CRLF;HALT);
IF NOT OPEN(TTCHAN,#210,SIXBIT 'TTY', OBUF^18)
    THEN (OUTS('???GCOULD NOT OPEN TTY');CRLF;HALT);

WHILE (WORD_INCHRW) NEQ #33 DO VREG_0;

WHILE (WORD_READ(ICHAN) EQL #200) DO VREG_0;

WORD_READ(ICHAN);
LOOP: WHILE 1 DO
        BEGIN
        LOOP2:  WHILE 1 DO
                BEGIN
                IF (HALF_READ(ICHAN)) EQL #200 THEN
                        LEAVE LOOP;
                IF (.HALF AND #300) EQL #300 THEN
                        BEGIN
                        IF (.HALF AND #77) EQL 0 THEN
                                BEGIN
                                .HALF_0;
                                LEAVE LOOP2;
                                END;
                        WRITE(TTCHAN,.HALF);
                        FORCEOUT(TTCHAN);
                        LEAVE LOOP2;
                        END;
                IF (WORD_.HALF AND #300) EQL #100 THEN
                BEGIN
                ADDR_((.HALF AND #77)^6+READ(ICHAN));
                LEAVE LOOP2
                END;
                HIGH_(((.ADDR)^(-6)) + #100) ;
                LOW_(.ADDR AND #77);
                WRITE(TTCHAN,.HIGH);
                WRITE(TTCHAN,.LOW);
                WRITE(TTCHAN,.HALF);
                HALF_READ(ICHAN);
                WRITE(TTCHAN,.HALF);
                FORCEOUT(TTCHAN);
                ADDR_.ADDR+1;
```

```
                END;
        END;
WHILE 1 DO VREG_0;
CLOSE(TTCHAN)
END ELUDOM
```

(4) EXPORT.BLI

```
MODULE DC72(STACK(300),TIMER=EXTERNAL(SIX12))=
BEGIN

EXTERNAL OPEN,WRITE,WRITES,READ,FORCEOUT,LOOKUP,
         ENTER,CLOSE;

REQUIRE TTCALL.BLI;

BIND TTCHAN=1, ICHAN=2, ACK=6, NAK=#25, MAXNAK=100,
        MAXTSH=100,
        SYNC=#26;

MACHOP JRST=#254, CALLI=#047;

MACRO HALT=JRST(4)$, RESET=CALLI(0)$;

OWN     OBUF[3],
        IBUF[3],
        MESSAGE[300],
        FILESPECS[4],
        REPLY,
        NAKCNT,
        TRASHCNT,
        ACKCNT;

BIND CKSUM=MESSAGE[0], WRDCNT=MESSAGE[1];

REGISTER WORD,ADDR;

LABEL   LOOP;

BIND    ERRTMN=0,
        ERRTMT=1;


ROUTINE OUTN(NUM,BASE,REQD)=
        BEGIN OWN N,B,RD,T;
        ROUTINE XN=
        BEGIN LOCAL R;
        IF .N EQL 0 THEN RETURN
        (DECR I FROM (.RD-.T-1) TO 0 DO
                WRITE(TTCHAN,"0"));
        R_.N MOD .B; N_.N/.B; T_.T+1; XN();
        WRITE(TTCHAN,.R+"0")
        END;

        IF .NUM LSS 0 THEN WRITE(TTCHAN,"-");
        B_.BASE; RD_.REQD; T_0; N_ABS(.NUM); XN()
        END;
```

```
ROUTINE ERROR(ERR)=
        BEGIN
        WRITE(TTCHAN,#200);
        CASE .ERR OF
        SET
WRITES(TTCHAN,PLIT(ASCIZ '???GTOO MANY NAK''S')<36,7>);
WRITES(TTCHAN,PLIT(ASCIZ '???GTOO MUCH NOISE')<36,7>);
        TES;
        CLOSE(TTCHAN);
        HALT
        END;


ROUTINE SENDBUFF=
        BEGIN

        LABEL SEND, CONFIRM;

        CKSUM_.CKSUM AND #77;
        SEND: WHILE 1 DO
                BEGIN
                CLRBFI;
                INCR I FROM 0 TO .WRDCNT+1 DO
                WRITE(TTCHAN,.MESSAGE[.I]);
                FORCEOUT(TTCHAN);
                CONFIRM: WHILE 1 DO
                        BEGIN
                        IF (REPLY_INCHRW) EQL ACK THEN
                                BEGIN
                                ACKCNT_.ACKCNT+1;
                                LEAVE SEND;
                                END;
                        IF .REPLY EQL NAK THEN
                                BEGIN
                                WRITE(TTCHAN,SYNC);
                                NAKCNT_.NAKCNT+1;
                                IF .NAKCNT GTR MAXNAK
                                THEN ERROR(ERRTMN);
                                LEAVE CONFIRM
                                END
                            ELSE
                                BEGIN
                                TRASHCNT_.TRASHCNT+1;
                                IF .TRASHCNT GTR MAXTSH
                                THEN ERROR(ERRTMT)
                                END;
                        END;
                END;
        END;


!!! MAINLINE !!!
```

```
RESET;
IF NOT OPEN(ICHAN,#10,SIXBIT 'DSK',IBUF<0,0>)
THEN (OUTS('???GCOULD NOT OPEN  DEVICE');CRLF;HALT);
FILESPECS[0]_SIXBIT 'DWNLIN';FILESPECS[1]_SIXBIT 'BIN';
FILESPECS[2]_FILESPECS[3]_0;
IF NOT LOOKUP(ICHAN,FILESPECS)
       THEN (OUTS('???GLOOKUP FAILED');CRLF;HALT);
IF NOT OPEN(TTCHAN,#210,SIXBIT 'TTY', OBUF^18)
       THEN (OUTS('???GCOULD NOT OPEN TTY');CRLF;HALT);

WRITE(TTCHAN,#201);

WHILE (WORD_READ(ICHAN)) EQL #200 DO .VREG_0;
MESSAGE[2]_CKSUM_.WORD;
MESSAGE[3]_READ(ICHAN);
WRDCNT_2;
WHILE (WORD_READ(ICHAN)) NEQ #200 DO
        BEGIN
        IF (.WORD AND #700) EQL #100 THEN
                BEGIN
                SENDBUFF();
                MESSAGE[2]_CKSUM_.WORD;
                CKSUM_.CKSUM+(MESSAGE[3]_READ(ICHAN));
                WRDCNT_2;
        LOOP:   WHILE 1 DO
                        BEGIN
                IF((WORD_READ(ICHAN))AND #700) NEQ #100
                  THEN LEAVE LOOP
                    ELSE BEGIN
                MESSAGE[2]_CKSUM_.WORD;
                CKSUM_.CKSUM+(MESSAGE[3]_READ(ICHAN));
                WRDCNT_2;
                        END;
                        END;
                END;
         IF .WRDCNT EQL #202 THEN
                BEGIN
                SENDBUFF();
                ADDR_((.MESSAGE[2])^6+.MESSAGE[3])
                        +((.WRDCNT-2)/2);
                MESSAGE[2]_((.ADDR)^(-6));
                MESSAGE[3]_(.ADDR AND #77);
                CKSUM_.MESSAGE[2]+.MESSAGE[3];
                WRDCNT_2;
                END;
        IF (.WORD AND #700) NEQ #300
                THEN CKSUM_.CKSUM+.WORD;
        WRDCNT_.WRDCNT+1;
        MESSAGE[.WRDCNT+1]_.WORD;
        END;
SENDBUFF();WRITE(TTCHAN,1);WRITE(TTCHAN,1);
WRITE(TTCHAN,#200);FORCEOUT(TTCHAN);
WRITES(TTCHAN,CRLFSTR);
WRITES(TTCHAN,PLIT ASCIZ 'NAK COUNT = ');
        OUTN(.NAKCNT,10,1);
```

```
WRITES(TTCHAN,CRLFSTR);
WRITES(TTCHAN,PLIT ASCIZ 'TRASH COUNT = ');
        OUTN(.TRASHCNT,10,1);
WRITES(TTCHAN,CRLFSTR);
WRITES(TTCHAN,PLIT ASCIZ 'ACK COUNT = ');
        OUTN(.ACKCNT,10,1);
WRITES(TTCHAN,CRLFSTR);
CLOSE(TTCHAN)
END ELUDOM
```

(5) SYNC.PAL

```
/   PDP-8/E SYNC. LOADER


NEG=CML CMA IAC
BSW=7002
SOH=201
STX=202
ETX=003
REP=005
SYN=226
MRK=376
IDLE=210



/  RECEIVE MESSAGE FORMAT
/  HEADER
/  SYN SYN SYN SYN SOH WC FLD ADR ADR LRC LRC
/
/  TEXT
/  MSG MSG MSG .. .. LRC LRC MRK
/
/  TRANSMIT MESSAGE FORMAT
/  SYN SYN SYN SYN SOH MSG MRK


/PDP-8/E SYNC. LINE INTERFACE IOTS


SGTT=6405        /TRANSMIT GO
SGRR=6404        /RECEIVE GO
SSCD=6400        /SKIP IF CHARACTER DETECTED
SCSD=6406        /CLEAR SYNC. DETECT
SSRO=6402        /SKIP IF RECEIVE WORD COUNT
                 /OVERFLOW
SCSI=6401        /CLEAR SYNC. INTERFACE
SRTA=6407        /READ TRANSFER ADDRESS REGISTER
SLCC=6412        /LOAD CONTROL
SSRG=6410        /SKIP IF RING FLAG
SSCA=6411        /SKIP IF CARRIER/AGC FLAG
SRS2=6414        /READ STATUS 2
SRS1=6415        /READ STATUS 1
SLFL=6413        /LOAD FIELD
SSBE=6416        /SKIP ON BUS ERROR
SRCD=6417        /READ CHARACTER DETECTED (IF AC0=0)
                 /MAINTENANCE INSTRUCTION (IF AC0=1)
SSTO=6403        /SKIP IF TRANSMIT WORD COUNT
                 /OVERFLOWS
```

FIELD 1

```
*7200
TOP,    CLA CLL          / ** MAIN LINE **
        TAD (REP         /INITIALIZE LOAD
        DCA LRM+5        /REQUEST BUFFER
        TAD (1100        /SET FIELDS
        SLFL
        JMS LDRQ         /REQUEST LOAD
LOOP,   JMS RVMS         /RECEIVE MESSAGE
        JMS MSGCHK       /MESSAGE CHECK
        JMP NAKSND       /ERROR RETURN
        JMS LDBF         /LOAD BUFFER
ACKSND, CLA CLL
        TAD ACK          /GET ACK CODE
        DCA LRM+5        /PUT IT IN MESSAGE
        JMS LDRQ         /SEND IT
        JMP LOOP
NAKSND, CLA CLL
        TAD NAK          /GET NAK CHARACTER
        DCA LRM+5        /PUT IT IN MESSAGE
        JMS LDRQ         /SEND IT
        JMP LOOP
ACK,    6
NAK,    225



LDRQ,   0                /THIS SUBROUTINE WILL
                         /SEND REPLY MESSAGES
        SCSI             /CLEAR SYNC. DETECT
        TAD (5400
        SLCC             /TERMINAL RDY, ENABLE,
        CLA CLL          /REQUEST
        TAD (LRM-1       /POINT TO MESSAGE
        DCA TCA          /SET CURRENT ADDRESS
        TAD (LRM-LRME    /SET LENGTH OF MESSAGE
        DCA TWC          /SET WORD COUNT
        SGTT             /SEND MESSAGE
        SSTO             /SKIP IF DONE
        JMP .-1
        JMP I LDRQ
LRM,    SYN;SYN;SYN;SYN;SOH;0;MRK
LRME,   0
*7630
RVMS,   0                /THIS SUBROUTINE WILL
        CLA              /RECEIVE MESSAGES
        TAD     (7400    /SET TIMER
        DCA TIME1
        DCA TIME2
RSTART, CLA
        SCSD             /CLEAR SYNC. DETECT
AGAIN,  CLA CLL CMA      /GET -1
        DCA RWC          /SET WORD COUNT
        TAD (BUFF-1
```

```
        DCA RCA          /SET CURRENT ADDRESS
        SGRR             /START RECEIVER
GSOH,   SSRO             /SKIP IF DONE
        JMP TSOH         /TIME CHECK
        TAD BUFF         /GET THE FIRST CHARACTER
        TAD (-226        /IS IT SYNC.?
        SNA
        JMP AGAIN        /YES, READ ANOTHER
        TAD (31          /IS IT SOH ?
        SZA CLA
        JMP RSTART       /NO, RESTART
        TAD HSIZE
        DCA RWC          /SET WORD COUNT
        SGRR             /START RECEIVER
GHDR,   SSRO             /SKIP IF DONE
        JMP THDR         /TIME CHECK
        CLA CLL
        TAD BUFF+1       /GET DATA LENGTH
        DCA MSIZE        /STORE MESSAGE SIZE
        TAD MSIZE
        NEG
        DCA RWC          /SET WORD COUNT
        SGRR             /READ DATA
GTXT,   SSRO             /SKIP IF DONE
        JMP TTXT
        JMP I RVMS
TSOH,   TAD (GSOH
        JMP TMLP
THDR,   TAD (GHDR
        JMP TMLP
TTXT,   TAD (GTXT
TMLP,   DCA RET          /RETURN ADDR.
        ISZ TIME2
        JMP I RET
        ISZ TIME1
        JMP I RET
        JMP NAKSND       /TOO LONG, SEND NAK
TIME1,  0
TIME2,  0
RET,    0
*7245
MSGCHK, 0
                /THIS SUBROUTINE PROVIDES A
                /LRC ON BUFFER CONTENTS AND
                /COMPARES IT TO THE RECEIVED LRC
        CLA CLL
        TAD HSIZE        /GET HEADER SIZE
        DCA SIZE         /STORE
        TAD HBEG
        DCA BEG          /GET HEADER BEGINNING
        TAD BUFF+5       /GET UPPER HALF LRC
        BSW              /SWAP HALVES
        TAD BUFF+6       /GET LOWER HALF
        NEG
        DCA LRC          /GET HEADER LRC
```

```
        JMS CLCLRC      /CALCULATE LRC
        JMP NAKSND      /NAK THIS MESSAGE
        CLA CLL
        TAD MSIZE       /GET MESSAGE SIZE
        DCA SIZE        /STORE
        TAD MBEG        /GET MESSAGE BEGINNING
        DCA BEG         /STORE
        TAD SIZE+7      /POINT TO LRC
        DCA MLRC
        TAD I MLRC      /GET MESSAGE LRC
        BSW             /SWAP HALVES
        ISZ MLRC
        TAD I MLRC      /GET LOW BYTE LRC
        NEG             /NEGATE AND
        DCA LRC         /STORE
        JMS CLCLRC      /CALCULATE LRC
        JMP NAKSND      /ERROR RETURN
        JMP I MSGCHK
SIZE,   0
BEG,    0
HSIZE,  -6
HBEG,   BUFF
MSIZE,  0
MBEG,   BUFF+7
LRC,    0
MLRC,   0


CLCLRC, 0               /THIS SUBROUTINE CALCULATES
                        /LRC AND MAKES COMPARISON
LRCLP,  TAD I BEG       /GET CHARACTER
        ISZ BEG         /NEXT CHARACTER
        ISZ SIZE        /SKIP IF DONE
        JMP LRCLP
        TAD LRC         /ADD NEGATIVE RECEIVED LRC
        SNA CLA         /SKIP NON-ZERO AC
        ISZ CLCLRC      /INC FOR NORMAL RETURN
        JMP I CLCLRC
*7400
BUFF,   0               /RECEIVE MESSAGE BUFFER


*7600
LDBF,   0               /THIS SUBROUTINE WILL LOAD
                        /BUFFER INTO THE PROPER
                        /MEMORY LOCATIONS
        CLA CLL
        TAD BUFF+3      /GET HIGH ADDR BYTE
        BSW             /SWAP HALVES
        TAD BUFF+4      /GET LOW ADDR BYTE
        DCA ADR         /STORE ADDR.
        TAD BUFF+2      /GET CURRENT FIELD
        TAD CDFINS      /MAKE INSTRUCTION
```

```
        DCA LDF
LDF,    0               /EXECUTE DATA FIELD CHANGE
        TAD (BEG
        DCA LBEG        /POINT TO MESSAGE BEGINNING
LOAD,   TAD LBEG        /GET FIRST HALF
        BSW             /SWAP HALVES
        ISZ LBEG
        TAD LBEG        /GET SECOND HALF
        DCA I ADR       /LOAD WORD
        ISZ LBEG        /INC POINTERS
        ISZ ADR
        ISZ MSIZE       /SKIP IF DONE
        ISZ MSIZE
        JMP LOAD
        JMP I LDBF
ADR,    0
LBEG,   0
CDFINS, 6201

                        /*7720-7723 MAY CONTAIN
                        /FOUR TEST CHARACTERS

*7724
RWC,    0               /RECEIVE WORD COUNT
RCA,    0               /        CURRENT ADDR.

*7727
TWC,    0               /TRANSMIT WORD COUNT
TCA,    0               /         CURRENT ADDR.
$
```

(6) LDSYNC.BLI

```
MODULE DC72(STACK(300),TIMER=EXTERNAL(SIX12))=
BEGIN

EXTERNAL OPEN,WRITE,WRITES,READ,FORCEOUT,LOOKUP,
         CLOSE,ENTER;

REQUIRE TTCALL.BLI;

BIND TTCHAN=1,  ICHAN=2, ACK=6, NAK=#25,
        MAXNAK=100, MAXTRASH=100;

MACHOP JRST=#254, CALLI=#047;

MACRO HALT=JRST(4)$, RESET=CALLI(0)$;

OWN     OBUF[3],
        IBUF[3],
        MESSAGE[260],
FILESPECS[4],
        REPLY,
        NAKCNT,
        TRASHCNT,
        HIGH,
        LOW,
        HALF,
        ADDR;

BIND CKSUM=MESSAGE[0], WRDCNT=MESSAGE[1];

REGISTER WORD;




ROUTINE OUTN(NUM,BASE,REQD)=
BEGIN
        OWN N,B,RD,T;
        ROUTINE XN=
                BEGIN LOCAL R;
                IF .N EQL 0 THEN RETURN
        (DECR I FROM (.RD-.T-1) TO 0 DO
                 WRITE(TTCHAN,"0"));
                R_.N MOD .B; N_.N/.B; T_.T+1; XN();
                WRITE(TTCHAN,.R+"0")
                END;

        IF .NUM LSS 0 THEN WRITE(TTCHAN,"-");
        B_.BASE; RD_.REQD; T_0; N_ABS(.NUM); XN()
        END;
```

```
LABEL   LOOP,LOOP2;

!!! MAINLINE !!!

RESET;
IF NOT OPEN(ICHAN,#10,SIXBIT 'DSK',IBUF<0,0>)
THEN (OUTS('???GCOULD NOT OPEN  DEVICE');CRLF;HALT);
FILESPECS[0]_SIXBIT 'IMPORT';
FILESPECS[1]_SIXBIT 'BIN';
FILESPECS[2]_FILESPECS[3]_0;
IF NOT LOOKUP(ICHAN,FILESPECS)
    THEN (OUTS('???GLOOKUP FAILED');CRLF;HALT);
IF NOT OPEN(TTCHAN,#210,SIXBIT 'TTY', OBUF^18)
    THEN (OUTS('???GCOULD NOT OPEN TTY');CRLF;HALT);

WHILE (WORD_INCHRW) NEQ #33 DO VREG_0;

WHILE (WORD_READ(ICHAN) EQL #200) DO VREG_0;

WORD_READ(ICHAN);
LOOP: WHILE 1 DO
        BEGIN
        LOOP2:  WHILE 1 DO
                BEGIN
                IF (HALF_READ(ICHAN)) EQL #200 THEN
                        LEAVE LOOP;
                IF (.HALF AND #300) EQL #300 THEN
                        BEGIN
                IF (.HALF AND #77) EQL 0 THEN
                                BEGIN
                                .HALF_0;
                                LEAVE LOOP2;
                                END;
                        WRITE(TTCHAN,.HALF);
                        FORCEOUT(TTCHAN);
                        LEAVE LOOP2;
                        END;
        IF (WORD_.HALF AND #300) EQL #100 THEN
                BEGIN
        ADDR_((.HALF AND #77)^6+READ(ICHAN));
                LEAVE LOOP2
                END;
                HIGH_(((.ADDR)^(-6)) + #100) ;
                LOW_(.ADDR AND #77);
                WRITE(TTCHAN,.HIGH);
                WRITE(TTCHAN,.LOW);
                WRITE(TTCHAN,.HALF);
                HALF_READ(ICHAN);
                WRITE(TTCHAN,.HALF);
                FORCEOUT(TTCHAN);
                ADDR_.ADDR+1;
                END;
        END;
```

```
CLOSE(TTCHAN);
BEGIN


BIND    TTCHAN=1,       ICHAN=2,        ACK=#225,
        MAXNAK=100,     MAXTRASH=100,   SYNC=#226,
        NAK=5,          REP=5,          IDLE=#210,
        MRK=#376;


OWN     OBUF[3],
        IBUF[3],
        MESSAGE[300],
        FILESPECS[4],
        REPLY,
        NAKCNT,
        LRC3,
        LRC4,
        TRASHCNT,
        ACKCNT;

BIND    SOH=MESSAGE[0],
        WRDCNT= MESSAGE[1],
        FLD=MESSAGE[2],
        ADR1=MESSAGE[3],
        ADR2=MESSAGE[4],
        LRC1=MESSAGE[5],
        LRC2=MESSAGE[6];

REGISTER WORD,ADDR;

LABEL   LOOP;



ROUTINE SENDBUFF=
        IF .WRDCNT NEQ 0 THEN
        BEGIN
        WRDCNT_.WRDCNT+2;
        LRC2_.LRC2 + .WRDCNT ;
        LRC1_(((.LRC2)^(-6)) AND #77);
        LRC2_(.LRC2 AND #77);
        LRC3_(((.LRC4)^(-6)) AND #77);
        LRC4_(.LRC4 AND #77);
        END;

        DS-10 I/O ROUTINES


ROUTINE STARTBUFF=
        BEGIN
        MESSAGE[3]_LRC2_(.WORD AND #77);
        LRC2_.LRC2 + (MESSAGE[4]_READ(ICHAN));
```

```
        WRDCNT_0;
        LRC4_0;
        END;


ROUTINE INTERRUPT=
        BEGIN
        SENDBUFF();
        ADDR_((.MESSAGE[3])^6+
        +.MESSAGE[4])+((.WRDCNT-2)/2);
        MESSAGE[3]_((.ADDR)^(-6));
        MESSAGE[4]_(.ADDR AND #77);
        LRC2_.MESSAGE[3]+.MESSAGE[4];
        LRC4_0;
        WRDCNT_0;
        END;

!!! MAINLINE !!!

RESET;
IF NOT OPEN(ICHAN,#10,SIXBIT 'DSK',IBUF<0,0>)
THEN (OUTS('???GCOULD NOT OPEN INPUT DEVICE');
        CRLF;HALT);
FILESPECS[0]_SIXBIT 'DWNLIN';
 FILESPECS[1]_SIXBIT 'BIN';
FILESPECS[2]_FILESPECS[3]_0;
IF NOT LOOKUP(ICHAN,FILESPECS)
        THEN (OUTS('???GLOOKUP FAILED');
                CRLF;HALT);

WHILE (WORD_READ(ICHAN)) EQL #200 DO .VREG_0;
STARTBUFF();
WORD_0;
WHILE (WORD_.WORD+1) NEQ 4 DO READ(ICHAN);
SOH_#201;
FLD_0;
WHILE (WORD_READ(ICHAN)) NEQ #200 DO
        BEGIN
        IF (.WORD AND #700) EQL #100 THEN
                BEGIN
                SENDBUFF();
                STARTBUFF();
        LOOP:   WHILE 1 DO
                        BEGIN
                IF((WORD_READ(ICHAN)) AND #700)
                        NEQ #100
                           THEN LEAVE LOOP
                           ELSE BEGIN
                                STARTBUFF();
                                END;
                        END;
                END;
        IF .WRDCNT EQL #144 THEN
                BEGIN
                INTERRUPT();
```

```
                END;
        IF (.WORD AND #700) EQL #300 THEN
                BEGIN
                INTERRUPT();
                FLD_(.WORD AND #77);
                LRC2_.LRC2 + .FLD;
                END
        ELSE
                BEGIN
                WRDCNT_.WRDCNT+1;
                LRC4_.LRC4 + .WORD;
                MESSAGE[.WRDCNT+6]_.WORD;
                END;
        END;

SENDBUFF();

END;
END ELUDOM
```

APPENDIX B

PROCEDURES

(1) TELEPHONE LINE LOADERS

(T0) PLUG IN THE ASCOUTIC COUPLER AND TURN IT ON

(T1) Plug in the ADM-1 terminal and turn it on

(T2) Dial up the DEC-10

(T3) Place the telephone receiver in the coupler and plug the ADM-1 into the coupler

(T4) LOGIN and type EX RIMPRT.BLI,IO.BLI

(T5) Unplug the ADM-1 from the coupler after execution begins

(T6) Plug PDP-8/E port 07 into the coupler using the reverse EIA adapter

(T7) Halt the PDP-8/E and lift the SW switch up

(T10) Press the bootstrap loader button, press enable, press the SW switch down and then lift it up

(T11) Panel lights will flash for approximately 40. seconds while IMPORT is being loaded.

(T12) After IMPORT has been loaded the word "READY" will appear on the console

(T13) Type control C twice on the console and when at monitor level type EX EXPORT.BLI,IO.BLI

(T14) After eight minutes a load summary will be typed on the console

(T15) Type control G and the DC72 will initialize

(2) SYNCHRONOUS LINE LOADER

(S0) Plug in the ascoutic coupler and turn it on

(S1) Plug in the ADM-1 terminal and turn it on

(S2) Dial up the DEC-10

(S3) Place the telephone receiver in the coupler and plug the ADM-1 into the coupler

(S4) LOGIN and type EX LDSYNC.BLI,IO.BLI

(S5) Unplug the ADM-1 from the coupler after execution begins

(S6) Plug PDP-8/E port 07 into the coupler using the reverse EIA adapter

(S7) Halt the PDP-8/E and lift the SW switch up

(S10) Press the bootstrap loader button, press enable, press the SW switch down and then lift it up

(S11) After loading occurs the DC72 will initialize

BIBLIOGRAPHY

Digital Data Communication Message Protocol, Maynard Massachusetts:Digital Equipment Corporation, 1974.

Donovan, John J. Systems Programming, New York:Mcgraw-Hill Book Company, 1972.

Eckhouse, Richard H., Jr., Minicomputer Systems: Organization and Programming, Englewood Cliffs, N.J.:Prentice-Hall, Inc., 1975.

External Bus Optional Maintenance Manual, Maynard Massachusetts:Digital Equipment Corporation, 1974.

Introduction to Programming, Maynard Massachusetts: Digital Equipment Corporation, 1970.

Small computer Handbook, Maynard Massachusetts:Digital Equipment Corporation, 1973

System Reference Manual, Maynard Massachusetts:Digital Equipment Corporation, 1975