AFIT/GEO/ENG/92D-01

DESIGN OF A LABORATORY COMPUTER INTERFACE

THESIS

AFIT/GEO/ENG/92D-01 Douglas L. Durand
Capt USAF

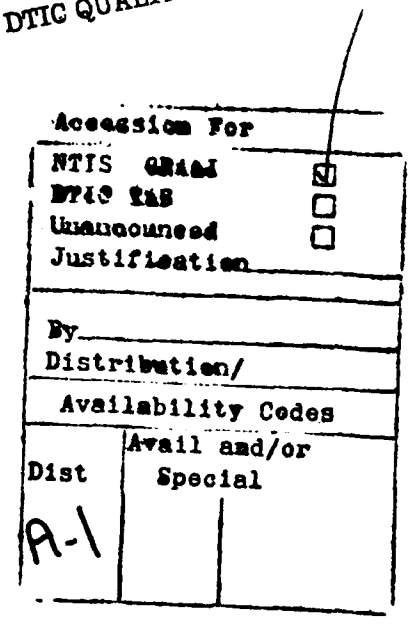
Approved for public release; distribution unlimited.

AFIT/GEO/ENG/92D-01

DESIGN OF A LABORATORY COMPUTER INTERFACE

THESIS

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



by
Douglas L. Durand, B.E.E.
Capt USAF
Graduate Electro-Optics
June 1992

Preface

The AFIT Physics Department has acquired an LSI-11 minicomputer system for use in their laser spectroscopy laboratory. The computer is to be used for data acquisition, data reduction, and equipment control. To perform these tasks, interface hardware and software is necessary. This thesis describes the design and implementation of this interface.

Contents

List of Figures

List of Tables

Abstract

An interface for the LSI-11 computer was designed and implemented so that the computer supports data acquisition, data reduction, and equipment control. The design includes both hardware and software and addresses both parallel and serial input/output (I/O).

The serial interface's hardware is simply a Serial Line Unit card. This card plugs into the LSI-11 bus and provides the signals necessary to interface EIA RS-232 compatible devices. A software utility was developed to allow communication with the serial device and to allow exchange of data files. Routines were written to allow serial I/O through a FORTRAN program.

The interface's parallel hardware includes the general purpose laboratory interface system (GPLIS) architecture. In addition, hardware modules were designed to convert certain device's signal levels to TTL levels. Software utilities were developed to acquire and store parallel data and routines were written to allow parallel I/O through a FORTRAN program.

1. INTRODUCTION

1.1 Background

In 1982, the AFIT Physics Department was in the process of setting up a laser spectroscopy laboratory to support both faculty and student in-house research. The facility was to provide the Air Force with a powerful set of state-of-the-art diagnostic tools in the tunable laser arena. One of the major pieces of equipment acquired for this facility was an LSI-11 minicomputer which was to be used for data acquisition, data reduction, and equipment control. This led the Physics Department to sponsor a thesis topic with the objective of developing the hardware and software necessary to interface the LSI-11 with a general class of laser spectroscopy experiments.

This thesis project was begun in 1982. During that year, the author completed the research, design, and implementation of the interface system. Completion of the thesis document itself, however, was delayed until 1992. This thesis document therefore is written from two time perspectives. The bulk of this document, from the remainder of this introduction through Appendix B, is written from the perspective of 1982. Appendix C brings the document up to date by addressing a design cycle from a 1992 perspective.

1.2 State of the Art--1982

Laboratory computer systems vary in cost and complexity. Digital Equipment Corporation (DEC) produces a

line of "MINC" computer systems which are small, PDP-11-based laboratory computers. With their various plug-in modules and powerful software, the MINC systems can perform data input/output (I/O), data manipulation, and control. The cost of such a system, however, can easily exceed $30,000 (according to a technical representative from Pioneer-Standard Electronics, Inc.).

IEEE Standard 583-1975, "Modular Instrumentation and Digital Interface System," describes an interface system which is sophisticated and complex. The Standard (Ref 1) describes a "crate" of plug-in modules. These modules allow various types of I/O under control of a "crate controller." The computer communicates with the modules through the crate controller.

Jerry G. Black describes an architecture for a general purpose laboratory interface system (GPLIS) (Ref 2). Like the IEEE 583-1975 system, GPLIS is modular. GPLIS, however, is much simpler and was designed specifically for the LSI-11. Its simplicity leads to low cost.

In addition to these systems, there are many interface cards manufactured for the LSI-11 (Ref 3). These cards plug into the LSI-11 bus and provide data lines and control lines for parallel and serial interfacing. A plug-in card alone may be all that is necessary to interface a certain device with the computer or the card may be part of a more complex interface.

The laboratory's existing LSI-11 computer system is a

DEC compatible version marketed by Heath. It includes a WJ-11-UL computer, a WH-27 dual disk drive, and a Z-19 video terminal. Also included is a Micro Peripherals Inc. Model 88G printer. Figure 1.1 shows a block diagram of the computer system plus the required interface and representative laboratory equipment (labelled Devices A through D). The laboratory equipment is further described in section 1.4 below.

1.3 Problem

The DEC MINC computer system and a system incorporating all the features described in IEEE 583-1975 are both too expensive for the laboratory to acquire. The problem is to develop economical hardware and software necessary to interface an LSI-11 with a general class of laser spectroscopy experiments. The computer will perform data acquisition, data reduction, and experiment control. In addition, an instruction manual is needed to explain to the users of the computer how to connect equipment to the interface, what programs are necessary, and how to operate the computer and interface. This manual must also explain how future equipment might be interfaced with the computer.

1.4 Laboratory Equipment

The laboratory equipment represented by Devices A through D in Figure 1.1 are of various types. The current experiments call for the use of serial devices (such as a cassette tape drive, a wave meter, and a modem), parallel devices (such as a paper tape reader and a multichannel

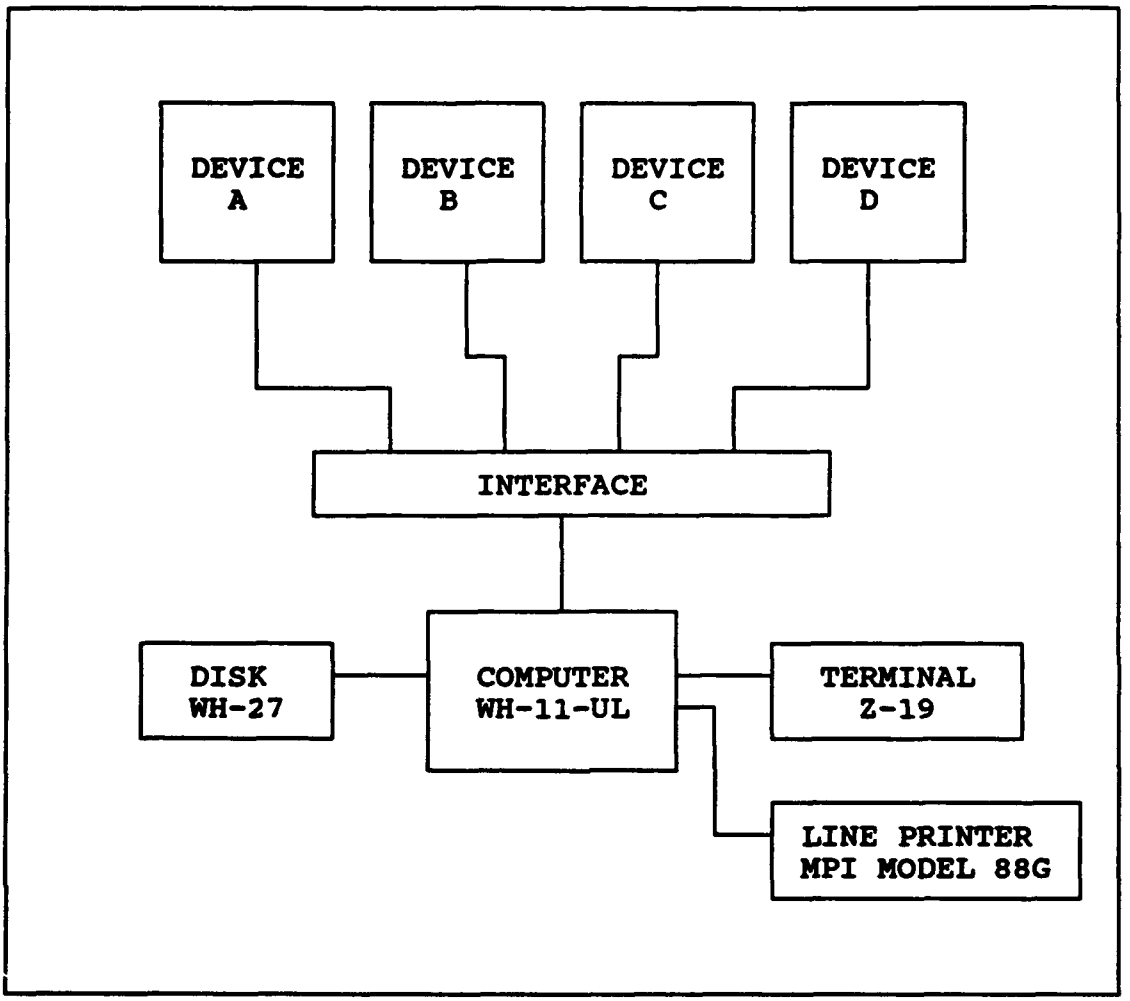


Figure 1.1 Computer/Interface System

analyzer) plus analog voltage sampling. In addition, future devices must also be considered.

The serial devices presently in use operate at various baud rates. A modem will be used to communicate with the on-base ASD CYBER at the currently available baud rate of 300 (although this rate could increase in the future). The cassette drive and the wave meter have switch-selectable baud rates. A future device could be expected to operate at any of the standard baud rates.

The parallel devices presently in use are the Hewlett Packard Model HP2737 paper tape reader and the Model 5400 multichannel analyzer. Their signal levels are not TTL compatible and therefore require conversion of their eight data lines and two control lines. Data transmission is controlled by handshaking thus allowing the computer to control the data rate. A future device could be expected to require signal conversion and some type of controlling signals.

Analog I/O is also required. These signals will be amplified external to the interface to levels suitable to the interface and the device. The sampling rates are expected to be slow--on the order of milliseconds.

1.5 Design Requirements

The interface must be flexible in that it must connect with a variety of equipment--some with standard interfaces and some without. It must also be adaptable to projected future equipment.

The present experiments have the following requirements:

1. Transfer of data via an RS-232 serial interface to the computer's disk storage.

2. Transfer of data from paper tape to the computer's disk storage.

3. Transfer of data from the computer's disk storage to a mainframe computer via a modem and phone lines.

4. Acquisition of data from equipment controllers and the transmission of control signals to these controllers through an RS-232 serial interface.

5. Acquisition of data through analog-to-digital (A/D) converters at timed intervals.

6. Transmission of signals through digital-to-analog (D/A) converters to provide a means of implementing a feedback loop of signal input, data processing, and signal output for control of specialized experiments.

Since the LSI-11's operating system is a single-user system and since it is expected that the computer will be used by one person at a time, timesharing need not be considered in this problem.

1.6 Design Approach

The interface system is one which allows the computer to communicate with devices which transfer data in serial, parallel, and analog form. The computer must receive data from these various devices, manipulate and store the data,

and transmit the data to the devices.

The first step to solving the interfacing problem was the familiarization with the computer system. This involved studying the reference manuals and operating the computer itself. A working knowledge of the computer's operating system, programming languages, architecture, and bus organization was necessary to determine what was possible and practical for an interface.

Next, it was necessary to determine what hardware and software were necessary to design an interface which satisfied the requirements listed earlier. The problem falls into two catagories--serial I/O and parallel I/O. The serial hardware must be RS-232 compatible and therefore is of standard type. The parallel hardware is not of any standard type. Each parallel device has signals which must be converted to TTL levels. This breaks the parallel interface design into modules--one for each device. Figure 1.2 shows a high-level block diagram of the hardware.

Software was likewise divided into two catagories--serial and parallel I/O. As shown in Figure 1.3, the hierarchy for software development for both serial and parallel I/O includes utilities and routines which in turn contain device drivers. The serial I/O with its standard hardware, requires development of only a single device driver. Parallel software is more complex in that there may be different device drivers for each device. Because device drivers are specific to the hardware to be

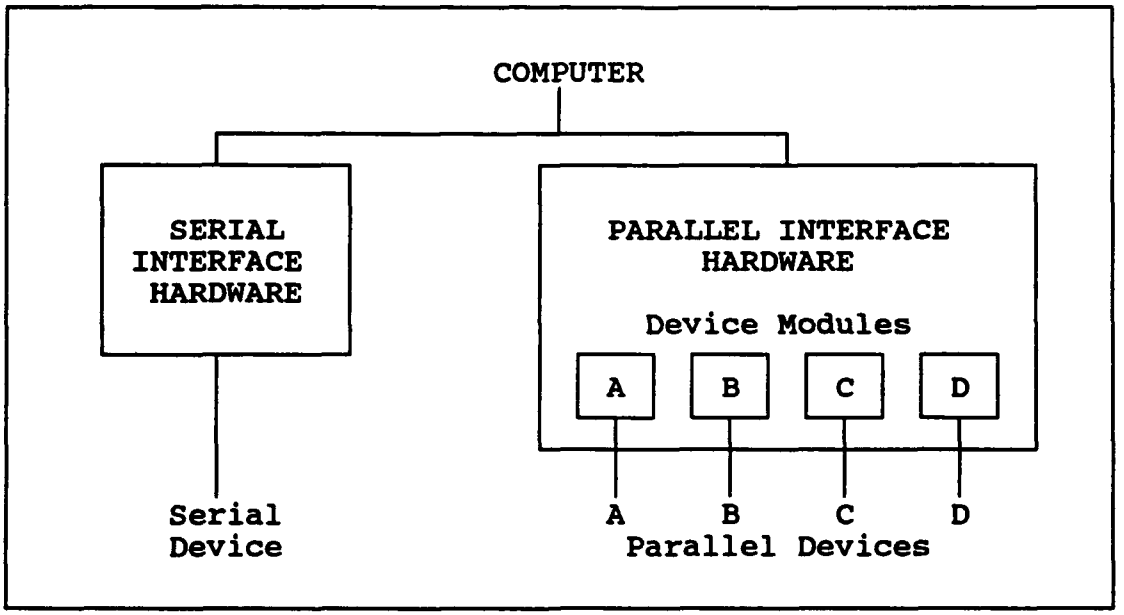


Figure 1.2 Interface Hardware

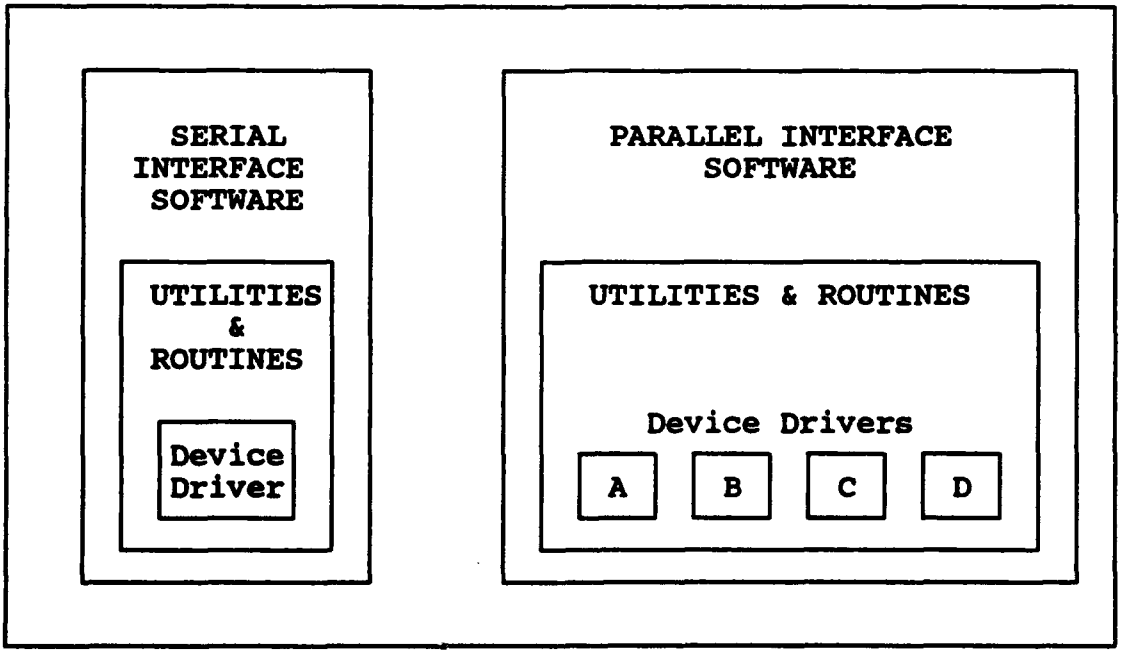


Figure 1.3 Interface Software

interfaced, they must be implemented in low level assembly language routines. Since a typical laboratory user may not have expertise in assembly language, however, any assembly language utilities and routines must be accessable to the user without the need to know their details. Actual operation of the interface will be through high-level language.

With the hardware and software thus broken into levels, the detailed design followed.

1.7 Summary

This chapter described the basic problem addressed by the thesis project (i.e., interfacing an LSI-11 computer to a general class of experiments). The background leading to the project was outlined and the laboratory equipment, design requirements, and design approach were discussed.

Chapter 2 describes the hardware of the interface from the plug-in cards to the different interface modules. Chapter 3 describes the software--addressing and interrupts along with a description of the utility programs and the subroutine package. Chapter 4 summarizes the results and conclusions. Appendix A is a User's Manual which has been written to help users of the computer/interface system. Appendix B contains a listing of the assembly language programs written for the interface. Appendix C, as noted earlier, addresses a design cycle from a 1992 perspective.

2. INTERFACE HARDWARE

2.1 Introduction

This chapter describes the hardware developed for the interface. The choice of existing hardware and the design of customized hardware depended not only on the design requirements, but also on the need to keep costs down.

2.2 Choice of Hardware

DEC's MINC system is a very powerful system for data acquisition, manipulation, and display. With its prewritten software routines and plug-in modules, it is extremely flexible and easy to use. Its $30,000 price tag, however, makes it unaffordable for the laser spectroscopy laboratory.

The GPLIS (Ref 2) is a simple, low cost interface architecture designed for the LSI-11. It is much simpler than the IEEE 583-1975 system which is more complex than required. The GPLIS provides exactly what is required--a multichannel parallel I/O interface. Using GPLIS as a base, modules can be added to suit the specific requirements of the devices to be used. This will be discussed further in section 2.4.

For serial I/O, an off-the-shelf plug-in serial I/O card was chosen. The card is the Heath WH-11-5 Serial Line Unit (SLU). It will be described in section 2.3 below.

Thus the hardware choice was composed of a combination of existing off-the-shelf hardware which could economically meet the requirements for standardized serial I/O plus a

customized design based on the GPLIS architecture to meet the requirements for the various parallel I/O devices. The GPLIS and the modules described below were breadboarded and tested separately before final assembly.

2.3 Serial Interface

To satisfy the requirement for an RS-232 serial I/O port, an off-the-shelf plug-in serial line card was chosen to economically interface with the serial devices which are all standardized. The card is the Heath WH-11-5 SLU. This SLU is compatible with the PDP-11 and LSI-11 (Ref 4:3).

The SLU card plugs into the LSI-11 bus and provides the necessary interface lines for the RS-232 serial interface. The card has jumper connections to select its memory address and interrupt vector. These will be discussed further in Chapter 3. The card has selectable baud rates of 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 4800, or 9600 (Ref 4:2).

By using the appropriate plugs, the card can be connected to data terminal equipment (DTE) such as terminals or to data communication equipment (DCE) such as modems or other computers.

Thus through this card, the computer can communicate with any serial device conforming to RS-232.

2.4 Parallel Interface

A parallel interface is necessary for devices which send and receive data more than one bit at a time. The GPLIS architecture provides a simple and economical basis

for development of an interface to the various I/O devices required.

The GPLIS first requires a plug-in card for the LSI-11 bus. This card is an off-the-shelf DEC DRV11 Parallel Line Unit (PLU). Like the SLU described above, the PLU is PDP-11 and LSI-11 compatible and plugs into the LSI-11 bus to provide the necessary interfacing lines (Ref 3:4-1). It too has jumper connections for selecting memory address and interrupt vector. These are described further in Chapter 3.

The PLU has three important 16-bit registers: (1) the input buffer (DRINBUF), (2) the output buffer (DROUTBUF), and (3) the control/status register (DRCSR) (Ref 3:4-38). The word formats for these registers are shown in Appendix A, Table A.1.

The PLU has four important control lines in addition to CSR1 and CSR0 from Table A.1. NEW DATA READY (NDR) and DATA TRANSMITTED (DATA TRANS) are positive-going pulses. NDR is used by GPLIS to latch data onto the output latches. DATA TRANS signals completion of data input and is used to acknowledge interrupt requests. INITIALIZE (INIT) is generated on power-up and is used to clear interrupt requests (Ref 3:4-39).

The PLU provides the connection from the computer bus to the GPLIS-based design. Together, they provide a multichannel 16-bit parallel interface. The following sections describe the modules that were designed and

implemented to form this parallel interface.

2.4.1 Clock Module. Because the clock module generates signals which can be used by other hardware modules, it was the first hardware module designed for this laboratory interface. It is used to generate interrupt requests. These requests set request bits in DRCSR and if the corresponding "enables" are set, the requests cause an interrupt. The module also has "acknowledge" outputs which are used by a user's device to signal that the computer has recognized the interrupt request.

The clock module can generate two different interrupt requests: one from its internal clock and one from an external clock. These are called, respectively, REQ A and REQ B. The internal clock can be used to generate interrupt requests at software-selectable, timed intervals of an integral number of milliseconds from one to 32,767. If a user requires interrupts either at intervals outside this range or at non-uniform intervals, the module will accept external clock signals. There are two external clock inputs: one for positive-going pulses and one for negative-going pulses. The user may choose either one depending on the device he is using.

The clock module requires one GPLIS input and one output channel. Currently, the input channel is only a dummy--it is needed only because DATA TRANS is used as the acknowledgment signal (DATA TRANS is generated when a GPLIS channel is read). A possible future use for this input

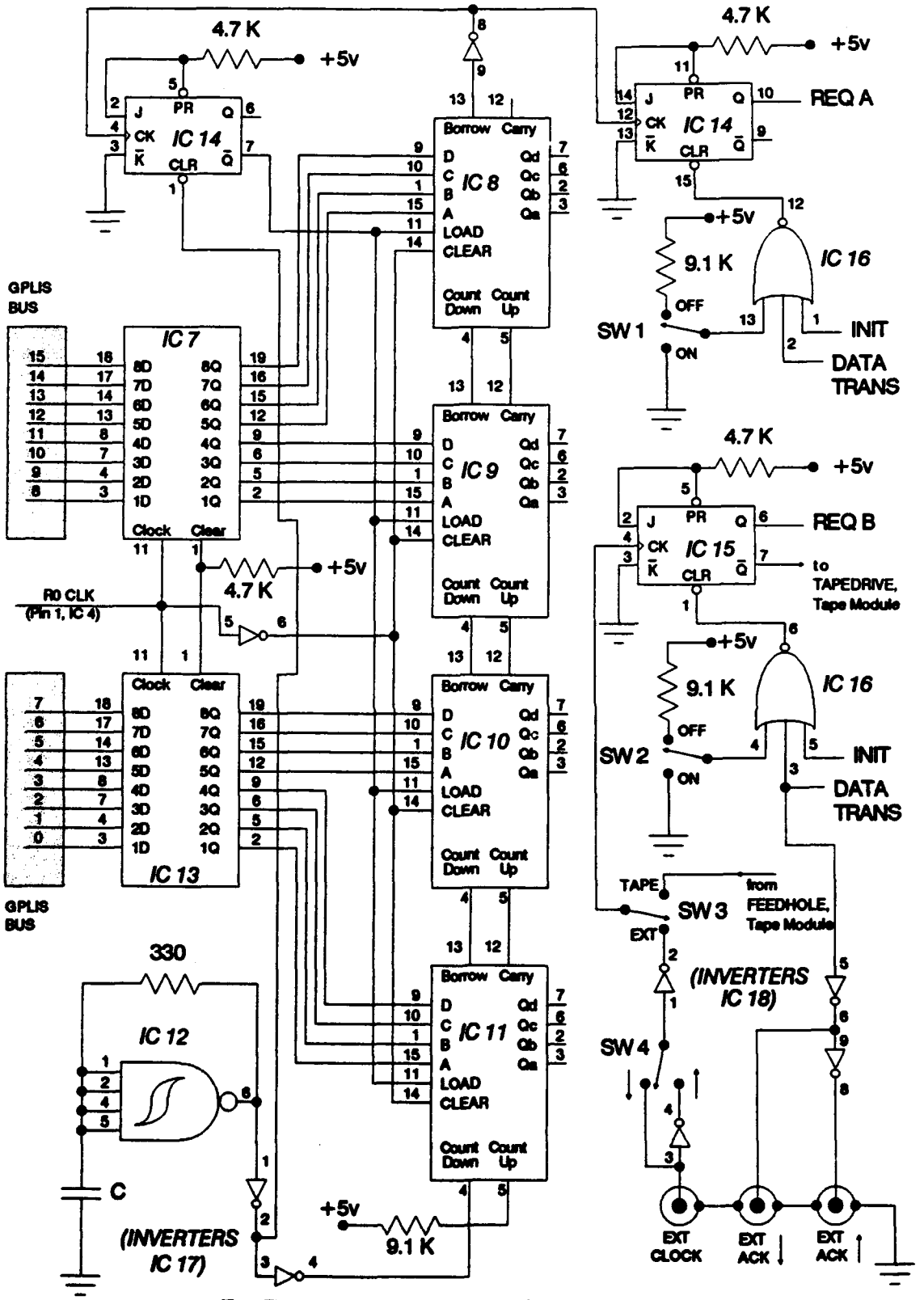
channel would be to read a device number which would be used by the input routine. In the following paragraphs, it is assumed that the clock module uses GPLIS input channel #0 and output channel #0.

As shown in Figure 2.1, the module is connected to the GPLIS bus and the ROCLK line. It is also connected to the PLU's REQ A, REQ B, INIT, and DATA TRANS lines. The module consists of a 74LS13 Schmitt Trigger, four cascaded 74LS193 synchronous 4-bit up/down counters, two 74LS109A dual J-K flip-flops, a 74LS27 tri 3-input NOR gate, two 74LS04 hex inverters, and resistors and capacitors.

The Schmitt Trigger is used as a monostable multivibrator set to 1000 Hz. With a 330 ohm resistor, $C + C' = 1/(390)x(1000)$ (Ref 5:239). With $C + C' = 2.564$ microfarads, the output waveform was a 300 microsecond wide pulse occurring once every millisecond. This waveform provides the basic set of pulses to the module's counters and flip-flops.

The 4-bit counters were cascaded to form a 16-bit counter. This 16-bit counter's data lines were connected in parallel to the output channel. This allows the counter to be loaded from the output channel, thus providing software-selectable interrupt intervals as explained in the examples below.

Two flip-flops were used for the interrupt request signals and one is necessary to produce the counter's LOAD signal.



(See Figure A.3 (Appendix A) for IC identification)

Figure 2.1 Clock Module

The sequence of a timed interrupt is as follows:

1. A 16-bit number is output to GPLIS channel #0.

2. The ROCLK latches this number on the output latches and also clears (sets to zero) the counter.

3. The next pulse from the Schmitt Trigger causes a "count down" which generates a BORROW because the count was zero.

4. The borrow is used to clock two different flip-flops:

   a. One flip-flop generates the LOAD signal causing the counter to be loaded from the output latches. This flip-flop is cleared by the next trigger pulse.

   b. The other flip-flop generates the REQ A signal. This flip-flop is cleared by the INIT or DATA TRANS or by the switch. While the switch is in the "off" position, no REQ A is generated even though the trigger and counter are still running.

5. The next count down pulse causes the newly loaded number to be decremented by one. This continues until the count is decremented to zero. Then the next count down generates a BORROW and the sequence continues from step 4.

As an example, assume the number 4 is output to channel #0. The number is latched and the counters are cleared by ROCLK. The next Schmitt Trigger pulse causes a BORROW which causes the number 4 to be loaded into the counter and also clocks the REQ A flip-flop. The next count down

decrements the count to 3; the next decrements the count to 2; then 1; then 0; and then BORROW--completing the cycle: 4, 3, 2, 1, 0, 4, 3, 2, 1, 0, ... A REQ A is generated every five counts. So to generate REQ A every T milliseconds, it is necessary to output (T-1) to channel #0. Figure 2.2 shows the example sequence.

The external interrupt is simply a flip-flop with inverters. The inverters allow the user to input either positive-going or negative-going pulses to generate REQ B.

To acknowledge either REQ A or REQ B, the computer performs a read from a GPLIS channel (dummy channel #0 or any other channel). This generates a DATA TRANS pulse on the PLU which is connected through NOR gates to the flip-flops' CLEARs. This turns off REQ A or REQ B (or both).

The flip-flops are also cleared by the INIT signals from the PLU. These signals are automatically generated on power-up (Ref 3:4-39). As mentioned earlier, REQ A can be disabled by a switch which forces the flip-flop to clear.

The external acknowledge is the DATA TRANS pulse. It is available as a positive-going or negative-going three microsecond pulse.

An example of a typical timed interrupt is the following: Assume a user wishes to input data from GPLIS channel #5 at 100 millisecond intervals. Meanwhile, the computer is to perform some other task. Solution--The clock module is set to generate interrupt requests at 100

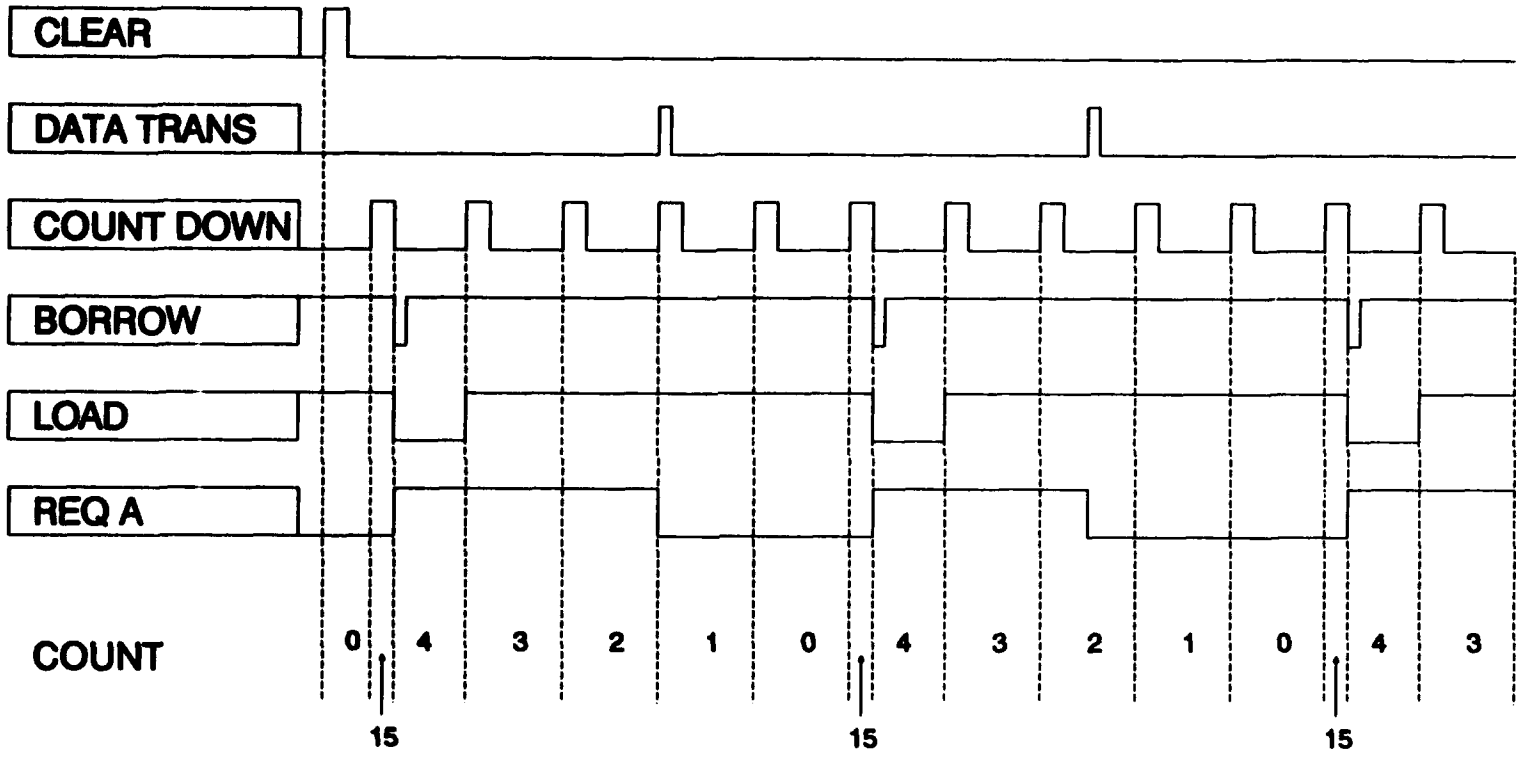


Figure 2.2 Example Count Sequence

millisecond intervals (output 99 to channel #0) and its interrupt enable is set (INT ENB A set to 1). Every 100 milliseconds, REQ A is generated which causes the computer to jump to the interrupt routine which in this case inputs data from channel #5. REQ A is turned off by DATA TRANS and the computer returns to its other task until the next REQ A.

An example of a typical external interrupt is the following: Assume a user wishes to input data from channel #6 whenever a certain laboratory device sends a positive-going pulse. Assume also that the user requires an acknowledgment pulse which is negative-going. No other tasks are required of the computer in this example. Solution--In this case, the user will use a routine which tests the interrupt request bit (REQ B) in DRCSR and performs the input when the bit is set. In this simplified example, rather than jumping to an interrupt routine, the computer simply executes a wait-loop until REQ B is set. The user's device can then request an interrupt, send data, and when it receives an acknowledgment, it will send further data; and so on until all data is sent.

The next two modules were also designed specifically for this laboratory interface. They use signals from the clock module just described and they provide the hardware interface to specific laboratory devices.

2.4.2 Paper Tape Module. The paper tape module converts the -12 volt signal levels from the HP2737 paper

tape reader to +5 volt signals used by the computer. The module also provides a TAPEDRIVE signal to the tape reader.

The tape reader's signals are positive logic (-12 volts = hole punched = logic 1 and 0 volts = no hole punched = logic 0). The module diagram and its level conversion circuit are shown in Figure 2.3. After level conversion, bits 0 through 7 are connected to a GPLIS input channel. The FEEDHOLE signal from the tape reader is connected to the clock module's external interrupt and is used to signal the computer to read the byte. The clock module's external acknowledge is connected to the tape reader's TAPEDRIVE signal which causes the tape to advance.

Data transfer is as follows:

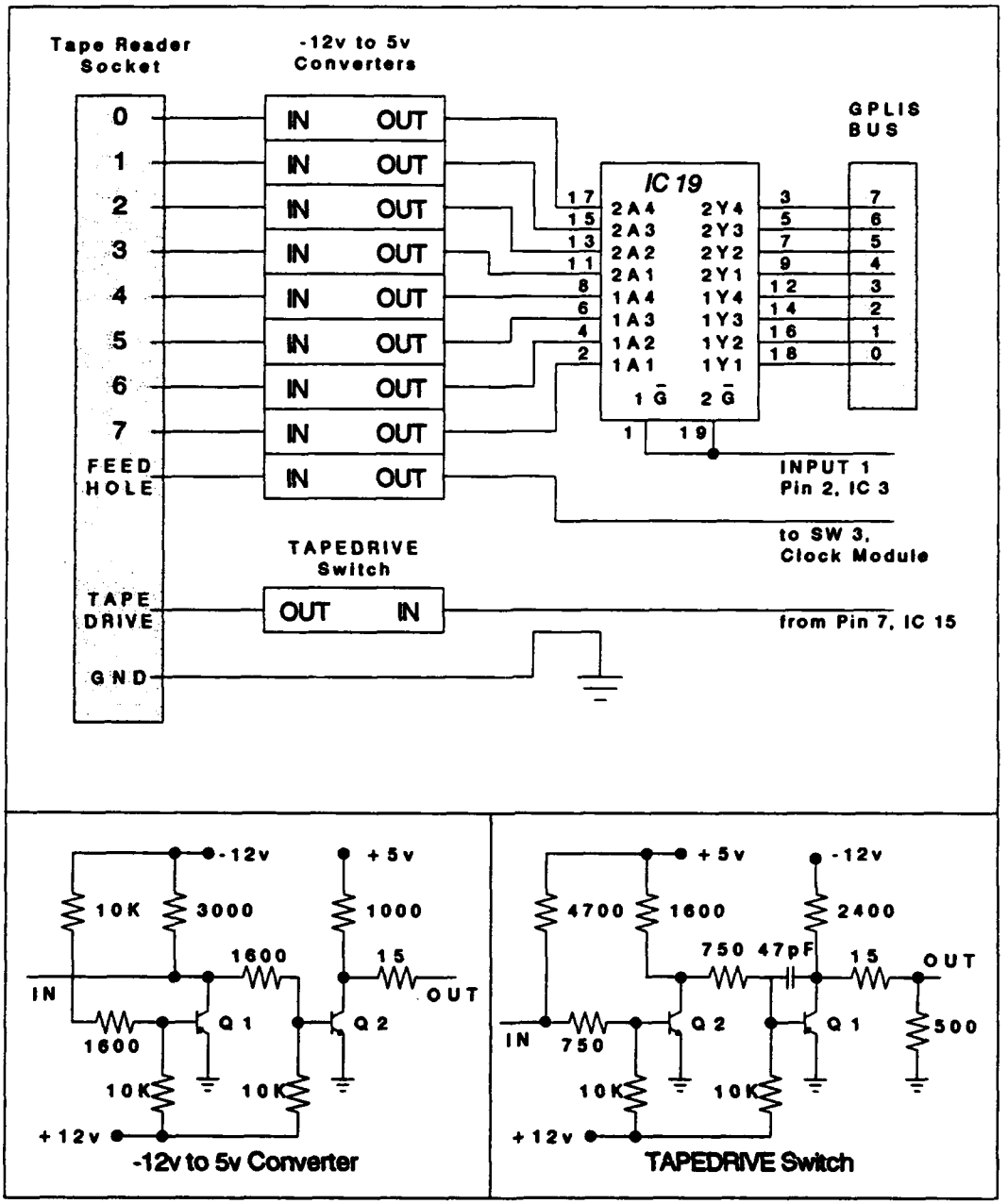
1. The tape feeds through the tape reader until the FEEDHOLE is under the read head. This signals the computer to read the byte.

2. The computer reads the byte and sends TAPEDRIVE causing the tape to advance until another FEEDHOLE is under the feed head.

3. This cycle repeats until the entire tape has run through the reader.

2.4.3 Multichannel Analyzer Module. Due to time constraints, the multichannel analyzer (MCA) module was not implemented as an interface. Its design is presented here for consideration as a future upgrade to the system.

The MCA module converts the +12 volt signals from the HP5400 multichannel analyzer to +5 volt signals used by the



Q1 = HP1853-0020 Q2 = HP1854-0071

(See Figures A.2 and A.3 (Appendix A) for IC identification)

Source: HP2737 User's Manual

Figure 2.3 Paper Tape Module

computer. The module also provides a +12 volt control signal used by the MCA.

The signals from the MCA were intended to operate a paper tape punch. They are negative logic (+12 volts = logic 0 and 0 volts = logic 1). The module diagram and its level conversion circuit are shown in Figure 2.4. After level conversion, bits 0 through 7 are connected to a GPLIS input channel. The MCA's PUNCH signal is connected to the clock module's external interrupt and is used to signal the computer to read the byte. The clock module's external acknowledge is connected to the MCA's FLAG signal which is used to signal for another byte.

With this module, the system emulates a paper tape punch. Instead of storing data on tape, however, the system stores the data on disk.

2.5 Summary.

This chapter described the interface hardware. Serial I/O is performed easily through a Serial Line Unit card. The card generates the signals required by the serial devices and by the LSI-11 bus. The Heath WH-11-5 SLU was chosen because it satisfies the requirements for serial I/O and because there was an unused WH-11-5 in the system. The parallel interface, on the other hand, required a custom design to provide for I/O to the various nonstandard devices to be used in the laboratory. The GPLIS architecture was chosen as a design basis to perform multichannel parallel I/O because it was specifically

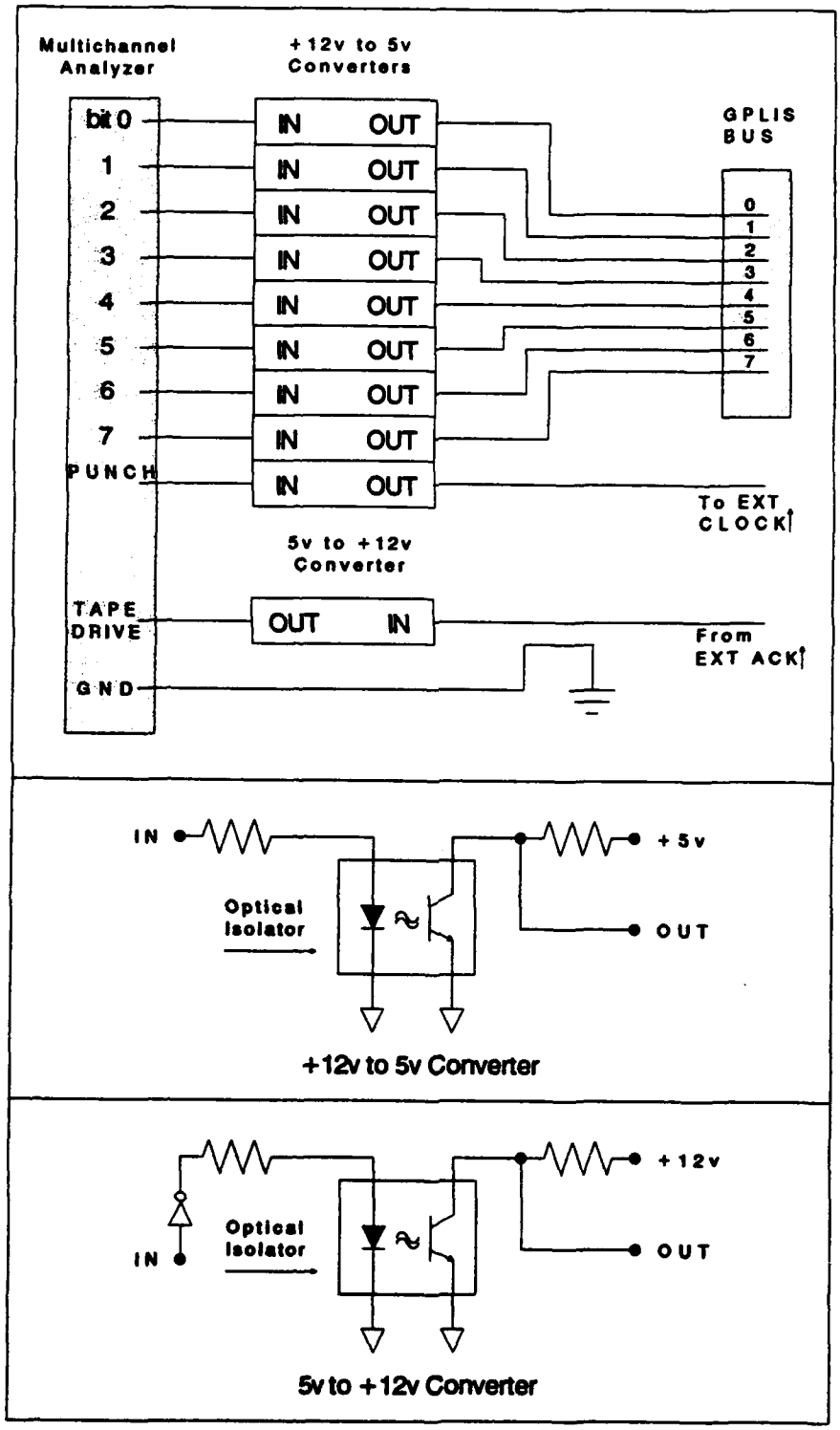


Figure 2.4 Multichannel Analyzer (MCA) Module

designed for the LSI-11 and because it is simple and inexpensive. The parallel interface design including its modules was described in detail. The next chapter will describe the software needed to run these components.

3. INTERFACE SOFTWARE

3.1 Introduction

This chapter describes the system's software from high level language to assembly language. At the assembly level, the addressing and interrupts are described. The utilities and subroutines which were developed to become the "prewritten programs" (from a future user's standpoint) are also described. At the high level, the choice of programming language is explained.

3.2 Choice of Software

The existing laboratory computer has three available programming languages: (1) assembly language, (2) BASIC, and (3) FORTRAN (Ref 5). Since the operating system recognizes only certain standard devices, software had to be developed to perform I/O for the laboratory interface. Because this I/O software had to manipulate low level information (e.g., registers and memory addresses) which depended on the detailed hardware design, a low level programming language was required. Assembly language programming was therefore selected as the available low level software. However, because the typical laboratory user may not have expertise in assembly language, operation of the interface must be thorough one of the available high level languages (BASIC or FORTRAN).

The requirement to use assembly language to manipulate low level information while still allowing a user to program through a high level language led to the need to

develop all assembly language I/O routines as prewritten packages (from a future user's standpoint). These assembly language routines would be available to the user through high level language subroutine calls or through the operating system. FORTRAN was chosen as the high level programming language because it allows subroutine calls to assembly language programs. The I/O routines were developed and compiled into a package which the user can link to his FORTRAN program. All a user needs to know are the routine names and their arguments.

3.3 Addressing

Before data I/O can occur, the location of the data's source or destination must be specified. The interface has two types of addressing. First, the Serial Line Unit (SLU) and Parallel Line Unit (PLU) have their addresses. Second, each GPLIS input and output channel has an "address" or channel number. GPLIS addressing is explained in Ref 2. The addressing for the SLU and PLU is explained below.

As mentioned in Chapter 2, the SLU and PLU have jumper connections for address selection. The SLU has four accessible registers and the PLU has three accessible registers. Each register is 16 bits (2 bytes) wide and has its own address which is treated like a memory address by the computer. Table 3.1 shows how the register addresses are allocated (the x's denote the jumper selectable portion of the address). (Ref 3:6-5, 6-13)

The allowable addresses for these registers can range

Table 3.1 Register Address Allocation

SERIAL LINE UNIT

Address (base 8)	Register (16-bit, 2 byte)
1xxxx0	RCSR (Receiver control/status)
1xxxx2	RBUF (Receiver data buffer)
1xxxx4	XCSR (Transmitter control/status)
1xxxx6	XBUF (Transmitter data buffer)

PARALLEL LINE UNIT

Address (base 8)	Register (16-bit, 2 byte)
1xxxx0	DRCSR (control/status)
1xxxx2	DROUTBUF (output data buffer)
1xxxx4	DRINBUF (input data buffer)

Source: Ref 3:6-5, 6-13.

from 160000 base 8 through 177777 base 8. Addresses were chosen for the SLU and PLU which did not conflict with reserved addresses as follows: SLU, 175610 base 8; PLU, 167770 base 8.

3.4 Interrupts

Whenever there is an interrupt, the CPU saves its current program address and program status word. It then loads a new address and status word and proceeds (Ref 2). In the LSI-11, these addresses and status words are stored in low memory and are pointed to by interrupt vectors. Each vector points to a 2 word (4 byte) data block. The low order word is the address of the interrupt routine and the high order word is the processor status word. The allowable vectors can range from 0 through 377 base 8 and are allocated as in Table 3.2. (Ref 3:6-6, 6-14)

Table 3.2 Vector Allocation

SERIAL LINE UNIT

Vector	Description
000xx0	Receiver interrupt vector
000xx4	Transmitter interrupt vector

PARALLEL LINE UNIT

Vector	Description
000xx0	Interrupt A
000xx4	Interrupt B

Source: Ref 3:6-6, 6-14.

One of the tasks of the prewritten initialization program (described below) is to store the appropriate addresses and status words in the locations pointed to by the vectors. These vectors were arbitrarily chosen to be 110 and 114 base 8 for the SLU; and 300 and 304 for the PLU.

3.5 Prewritten Programs

This section describes the prewritten assembly language programs developed for the interface. Again, these programs are "prewritten" from the standpoint of a future user; they were developed specifically for this laboratory interface. The programs include two utility programs, CONNECT and TAPEIN, and a package of subroutines, IOPACK. IOPACK is linked to the user's FORTRAN program to allow him to access the interface from his FORTRAN program. The programs were written, debugged, and tested separately. Appendix A has step-by-step instructions for the use of

these programs and Appendix B contains the software flow charts and program listings.

3.5.1 CONNECT Utility. The CONNECT utility is run from the computer's monitor and is used to access serial devices from the computer's terminal. While CONNECT is running, the system behaves as if the terminal were connected directly to the serial device. In addition, CONNECT offers some useful tools to the user.

The first tool allows the user to record data from the serial device on the computer's disk. To start recording, the user enters CONTROL-R. After this, all data from the serial device is stored on disk as it appears on the screen. The user stops recording by entering CONTROL-T or by exiting CONNECT (CONTROL-P). The record routine is double buffered allowing the program to record data at rates up to 1200 baud.

The second tool allows the user to transmit a file from the computer's disk to the serial device. The user enters CONTROL-E and a previously stored file is transmitted as it appears on the screen.

In addition, if the user desires to perform other operations with the computer, he may do so by exiting CONNECT (CONTROL-P). This does not disturb the device and the user may reenter CONNECT to pick up where he left off. From the device's point of view, it is as if the user had simply sat idle instead of having actually switched from the device to the computer and back to the device.

One typical use for CONNECT would be for communication with the on-base CYBER. In this case, the user plugs the modem into the interface, runs CONNECT, and initiates communication with the CYBER as if the terminal were connected directly to the modem. The user may read a file from the CYBER to his disk, switch attention to the computer and perhaps edit that file, and then switch back to the CYBER and replace the file with the updated version.

Another typical use for CONNECT would be for communication with a cassette tape drive. The Canberra tape drives can be operated from a terminal and therefore can be operated through CONNECT. The user could read data from the tape to disk or from disk to tape. Thus, CONNECT is a general purpose utility for exchanging data with serial devices.

3.5.2 TAPEIN Utility. The TAPEIN utility is run from the computer's monitor and is used to read data from a paper tape and store it on disk. To read a tape, the user loads the tape in the tape reader, plugs the reader into the interface, and then runs TAPEIN. The program reads the data through the paper tape module of the interface and stores the data on disk.

A typical use for the TAPEIN utility would be to read paper tapes produced by the Hewlett Packard multichannel analyzer (MCA). The data from the tapes could be stored on disk for later manipulation by a user-written program or for transmission to the CYBER for high-quality plotting on

a CALCOMP plotter.

3.5.3 MCAIN Utility. Due to time constraints, the multichannel analyzer (MCA) module was not implemented on the interface. It is outlined here for consideration as a future upgrade to the system.

The MCAIN utility is run from the computer's monitor and is used to read data from the HP5400 MCA and to store the data on disk. To read data, the user plugs the MCA into the interface, runs MCAIN, and operates the MCA as if he were punching a tape. The computer program reads the data through the MCA module and stores the data on disk. The user can thus avoid the use of paper tape altogether if he so desires.

3.5.4 IOPACK Subroutine Package. IOPACK is a set of assembly language subroutines which may be called from a user's FORTRAN program. The subroutines allow the user access to the interface through FORTRAN without having to know the details of how the subroutines work. The user needs to know only the routine name and its arguments.

The simple I/O routines are PARIN, PAROUT, SERIN, and SEROUT. PARIN and PAROUT are used, respectively, to input and output 16-bit words to a GPLIS channel. They have two arguments--a GPLIS channel number and the 16-bit data word. The arguments are both FORTRAN integers. An example of a subroutine call is the following: CALL PARIN(2,IVOLT). This would read GPLIS input channel #2 into the variable IVOLT where IVOLT might represent an

analog voltage.

SERIN and SEROUT are used, respectively, to input and output serial data. These routines have only one argument--the data byte which is a FORTRAN integer. An example subroutine call is shown in the following program segment:

```
      DO 10 I=1,6
      CALL SERIN(CHAR(I))
10    CONTINUE
```

In this segment, a six element array of ASCII characters is read. These could be later converted into a real number for further manipulation.

These first four routines allow simple I/O in either parallel or serial form. The remaining routines allow the user to take advantage of the interrupt capability of the clock module developed for the interface.

The first of these routines, INIT, sets up the interrupt vectors. INIT must be called from the user's FORTRAN program to enable the user to run interrupt routines. Two assembly language interrupt routines were developed: AINT and BINT. Each time one of the interrupt requests (REQ A or REQ B) is received, AINT or BINT increments its respective counter which can be tested by two other two subroutines, WAITA and WAITB. WAITA and WAITB allow the user to bring his FORTRAN program to a pause until either a REQ A or REQ B is received. Finally, two assembly language subroutines were developed to allow

the user to enable and disable the interrupts from within his FORTRAN program: INTON and INTOFF. The operation of these routines will be made clear in the example below.

A typical use of the interrupt capability would be to sample a GPLIS input channel at a timed interval. In this example, assume the input channel is #5 and the timed interval is 100 milliseconds. To accomplish this, the user would write a FORTRAN program which contains the following program segment:

```
      PAROUT(0,99)
      INIT
      DO 10 I=1,1000
      WAITA
      PARIN(5,SAMPLE(I))
10    CONTINUE
      INTOFF
```

In the above example, PAROUT is used to set the clock module so that it generates a REQ A signal every 100 milliseconds (see section 2.4.1). INIT initializes the interrupt capability and enables the interrupts. Each time a REQ A signal is received, the assembly language interrupt routine AINT increments a counter. WAITA checks that counter and causes the program to pause until REQ A is received. Then PARIN is used to input the sample data. After the DO-loop, the interrupts are disabled with INTOFF.

A warning message is generated if either of the following occurs: (1) two or more REQ A signals are received before WAITA is called or (2) WAITA is called after REQ A has been generated. These warning messages

alert the user to an error in his FORTRAN program which is causing too much time to elapse between WAITA calls.

3.6 Summary

This chapter described the software aspects of the interface. FORTRAN was chosen as the high level programming language for the user because it allows subroutine calls to the assembly language programs developed to operate the interface. These programs were described in detail as were the two utility programs. The goal of the software development was to allow the user to stay at as high a level as possible so he need not worry about the low level details. The next chapter summarizes the results of the hardware and software system design.

4. RESULTS

This chapter summarizes the results of the system design and of the test procedures.

4.1 Serial Interface

To satisfy the requirement for communication with RS-232 compatible serial devices, an off-the-shelf plug-in Serial Line Unit was used. The SLU allows communication with DCEs or DTEs at baud rates of 50 to 9600. Through the CONNECT utility developed for this interface, the user may communicate with a device through the computer's terminal and he may transmit and receive data files. Also, through the SERIN and SEROUT routines developed for this interface, the user may access a serial device through a FORTRAN program.

The hardware and software was tested by actually communicating with various serial devices. These devices included a microcomputer, a terminal, a cassette tape drive, a wave meter, and a modem which was connected to the on-base CYBER. Data was sent and received at the various baud rates at which these devices were able to communicate. With a microcomputer sending a continuous stream of data, it was found that the CONNECT utility can record data reliably at rates up to 1200 baud. Above that rate, the computer could not empty its buffers fast enough to avoid losing data.

4.2 Parallel Interface

The GPLIS architecture was used to provide parallel

data communication. Connection of the developed hardware to the computer's bus was through an off-the-shelf plug-in Parallel Line Unit. The GPLIS implementation plus the specific interface modules allowed parallel I/O with the various nonstandard laboratory devices. The clock module was developed to generate interrupts--either timed or external--and to send acknowledge signals. Two other modules were developed to convert signals to TTL levels so that the interface could accept data from a paper tape reader or a multichannel analyzer (as noted earlier, the MCA module was designed, but not implemented).

The software associated with the parallel interface includes the TAPEIN utility and the PARIN and PAROUT routines. The utility allows data input to disk and the routines allow parallel I/O through a FORTRAN program.

The clock module was tested and adjusted by monitoring its signals with an oscilloscope. By adjusting the variable capacitor, the COUNTDOWN signal was set at a frequency of 1000 Hz. Loading the timer with various values via software produced interrupt requests at various timed intervals as explained in section 2.4.1.

The software utility was tested through the actual recording of data from the paper tape reader. The files produced by this utility can be displayed on the terminal or printer or they may be used as an input file to a FORTRAN program.

Conclusions and recommendations are provided next.

5. CONCLUSIONS AND RECOMMENDATIONS

The interface system is a flexible, expandable system for data acquisition, data manipulation, communication, and control. It is flexible because it can be operated by a user via FORTRAN subroutine calls or operating system commands. The user can therefore customize operation to suit his application. It is expandable because of its modular architecture. Additional input or output channels may be connected to the GPLIS bus (see Ref 2). As presently configured, a total of 16 input and 16 output channels are possible.

Users will need to become familiar with the system by reading the reference manuals (Refs 6, 7, and 8) and the User's Manual (Appendix A). In general, the users need not know assembly language programming to use the system.

One member of the laboratory staff, however, should become familiar with PDP-11 assembly language programming (see Ref 9). This person would be designated the "System Programmer." The System Programmer would be the expert of the system and would be responsible for maintenance and modification of the system. He would also keep spare copies of the system's software disks to restore the system in case of accidental destruction of data on a system disk. In general, the System Programmer would be responsible for maintaining the system as a useful device in the laser spectroscopy laboratory.

BIBLIOGRAPHY

1. IEEE Std 583-1975. Modular Instrumentation and Digital Interface System (CAMAC). New York: The Institute of Electrical and Electronics Engineers, Inc. 1975.

2. Black, Jerry G. "Simple Laboratory Computer Interface System," Review of Scientific Instrumentation, 51 (5): pp 655-7, May 1980.

3. EK-LSI11-TM-003. LSI-11, PDP-11/03 User's Manual. Maynard, Mass., Digital Equipment Corp., 1976.

4. 595-2158-01. Serial Interface Module, Model WH11-5, Operation/Service Manual. Benton Harbor, Mich.: Heath Company, 1978.

5. Greenfield, Joseph D. Practical Digital Design Using IC's. New York: John Wiley & Sons, Inc., 1977.

6. 595-2225-04, Part A. Introductory Operations.
   Part B. BASIC User's Guide.
   Part D. Operating System.
   Benton Harbor, Mich.: Heath Company, 1978.

7. 595-2235-01, Part A. FORTRAN IV Introduction.
   Part B. FORTRAN IV User's Guide.
   Benton Harbor, Mich.: Heath Company, 1978.

8. DEC-11-LFLRA-C-D. PDP-11 FORTRAN Language Reference Manual. Maynard Mass.: Digital Equipment Corp., 1975.

9. Singer, Michael. PDP-11 Assembly Language Programming and Machine Organization. New York: John Wiley & Sons, Inc., 1980.

10. Kocher, Carl A. "A Laboratory Course in Computer Interfacing and Instrumentation," American Journal of Physics, 60 (3): pp 246-51, March 1992.

11. Bok, J., Barvik, I., Praus, P., Herman, P., and Cermakova, D. "Integrated Software Packages in the Physical Laboratory," Computer Physics Communications, 61 (1 & 2): pp 219-24, November 1990.

12. Petrini, M. F., Dwyer, T. M., Wall, M. A., Mansel, J. K., and Norman, J. R. "Communication Between the PC and Laboratory Instruments," Computer Applications in the Biosciences, 6 (3): pp 161-4, July 1990.

13. Hall, B. D. "A General-Purpose Interface System for the Laboratory," Computer Physics Communications, 61 (1 & 2): pp 239-45, November 1990.

APPENDIX A

USER'S MANUAL

APPENDIX A

USER'S MANUAL

Contents

1. Introduction

This User's Manual should be used as a supplement to the Heath Reference Manuals and to the thesis text. This manual does not describe all the details of the computer/interface system, but it should serve as an aid to the user. If additional information is desired regarding the assembly language routines developed for the interface, the user should refer to the thesis text (Chapter 3), the software flow charts and program listings (Appendix B), and an assembly language programming manual (e.g., Ref 9).

2. BOOTUP Procedure

a. Set switches to DC OFF, RUN, and LTC OFF.

b. Turn on AC power switches.

c. Set switch to DC ON. The computer will respond with "$".

d. Insert disks and close doors.

e. Enter "DX" (or "dx") on the keyboard. The computer will respond as follows:

```
HT-11   H01A-5
.SET USR NOSWAP
.SET TTY SCOPE
THE PREVIOUS DATE WAS 17-DEC-82          (date will vary)
CHANGE?
```

f. Enter the correct date or hit RETURN. The computer will respond as follows:

```
.ASSIGN DX1=DK
.
```

g. At this point, programs may be run.

3. FORTRAN Programming

a. Write a FORTRAN program through EDIT. Be sure it has a .FOR extension.

b. Enter "R FORTRA" to run the FORTRAN compiler. The computer responds with "*".

c. (The following steps assume your FORTRAN program is called UPROG.FOR.) Enter "UPROG,UPROG=UPROG" to compile UPROG.FOR (produces UPROG.OBJ and a list file UPROG.LST).

d. Exit the compiler with a CONTROL-C.

e. Enter "R LINK" to link the program. The computer responds with "*".

f. Enter "UPROG=UPROG,IOPACK/F". This links the program with the IOPACK routines to allow access to the interface.

g. Exit LINK with CONTROL-C.

h. Run the program by entering "RUN UPROG".

4. IOPACK

IOPACK is a set of assembly language subroutines which are linked to a user's FORTRAN program to allow the program to use the interface. The routines are called just like FORTRAN subroutines. Each is described below.

a. PARIN(arg1,arg2)

PARIN is used to input data from a GPLIS input channel. The first argument, arg1, is the GPLIS input channel number and must be an integer from 0 to 15. The second argument is the input data word and must be an integer from -32,768 to +32,767.

b. PAROUT(arg1,arg2)

PAROUT is used to output data to a GPLIS output channel. Its arguments are integers and are the channel number (0 to 15) and output data word (-32,768 to +32,767) respectively. PAROUT is used to set the timer. Forexample, PAROUT(0,T) will set the timer to generate REQ A once every T+1 milliseconds.

c. SERIN(arg1)

SERIN is used to input a data byte from the serial interface. Its argument is the input byte and will be an integer from 0 to 255.

d. SEROUT(arg1)

SEROUT is used to output a data byte to the serial interface. Its argument is the data byte and must be an integer from 0 to 255.

e. INIT

INIT sets up interrupt vectors. INIT must be called before a program can use the interrupt capability of the clock module. It should be called immediately before the program segment(s) which use the WAITA or WAITB subroutines described below. This routine has no arguments.

f. INTOFF

INTOFF disables the interrupts. INTOFF should be called immediately after the program segment(s) which use the WAITA or WAITB subroutines described below. This routine has no arguments.

g. INTON

INTON enables the interrupts. If desired, INTON may be used after INTOFF is called to re-enable the interrupts. It is not necessary to call INTON after calling INIT, however, because INIT automatically enables the interrupts.

h. WAITA

WAITA is used to pause a program until the interrupt request REQ A is received. REQ A is generated by the interface's internal clock ("Timer"). If two or more REQ A signals are received before WAITA is called or if WAITA is called after a single REQ A has been received, an error message will be generated. This alerts the user to an error in his FORTRAN program which is causing too much time to elapse between WAITA calls. This routine has no arguments.

i. WAITB

WAITB performs the same function as WAITA except that it works with REQ B. REQ B is generated by using the interface's external clock input.

5. CONNECT Utility

CONNECT is used to communicate with serial devices such as a cassette drive or a modem.

a. Start Up

(1) Set the four baud rate selector switches to the baud rate of the device. The chart on the side of the computer shows how to set the switches. WARNING--If data

will be recorded from the device, the baud rate should be no higher than 1200. If necessary, the baud rate of the device should be changed to 1200 or below.

(2) Plug the device into the male or female "D" connector.

(3) Turn on the device and enter "RUN CONNECT" on the keyboard.

(4) The bottom line of the screen will show a list of which CONTROL keys perform the following special functions:

```
Set FULL DUPLEX
Set HALF DUPLEX
Turn on RECORD
Turn off RECORD
TRANSMIT file
EXIT
```

(5) Set FULL or HALF DUPLEX as required.

(6) Now use the terminal as if it were connected directly to the device. The terminal will send all characters except those six control characters and it will receive all characters from the device.

b. Recording Data

To record data from a device, turn on RECORD. All data from the device received after this will be stored in a file called TAKEN.DAT. WARNING--Be sure to save any previous files named TAKEN.DAT under a different file name before turning on RECORD.

To stop recording data, either turn off RECORD or EXIT.

c. Transmitting Data

To transmit data, simply enter the TRANSMIT control key. This causes the file GIVEN.DAT to be sent to the device. WARNING--The file GIVEN.DAT must be created before attempting to transmit data. The transmitted data will appear on the screen as it is sent to the device.

SPECIAL NOTE--To transmit a file to the CYBER, enter the following after "COMMAND-" is given: COPYBF,INPUT,filename (where filename is any unused file name). Now enter the TRANSMIT control key to transmit the file. After the file is sent, enter "%EOF" to signal the CYBER that this is the end of file. To confirm transmission, first enter "REWIND,filename" and then "COPYSBF,filename". This will cause the CYBER to print the file on the screen.

d. EXIT

To leave CONNECT, simply enter the EXIT control key. This transfers control back to the computer's monitor. This will not disturb the device and it is possible to reenter CONNECT by entering "RUN CONNECT" as before. The terminal may thus be connected to the device or to the computer as desired.

6. TAPEIN Utility

TAPEIN is used to read paper tapes. The procedure is as follows:

a. Plug the paper tape plug into the socket marked TAPE READER.

b. With the tape reader's RUN/LOAD switch in the LOAD position, thread the tape into the reader.

c. If there is enough header on the tape, the take-up reel should be used to reel in the tape. Otherwise, the tape will simply feed through the reader and pile up on the table.

d. Set the RUN/LOAD switch to RUN and turn the tape reader on.

e. Enter "RUN TAPEIN" on the keyboard. The tape will feed through the reader and the data will be stored in the file TAPE.DAT. WARNING--Be sure to save any previous files named TAPE.DAT under a different file name before running TAPEIN.

7. Future Devices

This interface was designed to be easily adaptable to new devices. The assembly language programs have been developed such that the user can take advantage of their features without needing to know the details of assembly language. However, if for some reason it becomes necessary to modify or to add to the assembly language programs, an excellent reference which can serve as a programmer's manual is Ref 9.

The hardware is easily adaptable. Any serial device which conforms to RS-232 can be plugged into the interface. The available baud rates will almost surely match one of the device's baud rates. The parallel interface is also easily expandable. Additional input or

output channels may be connected to the GPLIS bus (see Ref 2). As presently configured, a total of 16 input and 16 output channels are possible.

The following drawing package (Figures A.1 through A.5) further documents the interface implementation and should be used as a reference for use and future modifications.

Following the drawing package is Table A.1 which provides the detailed description of the word formats for the three important Parallel Line Unit (PLU) 16-bit registers, DRINBUF, DROUTBUF, and DRCSR described in section 2.3 of the thesis text.

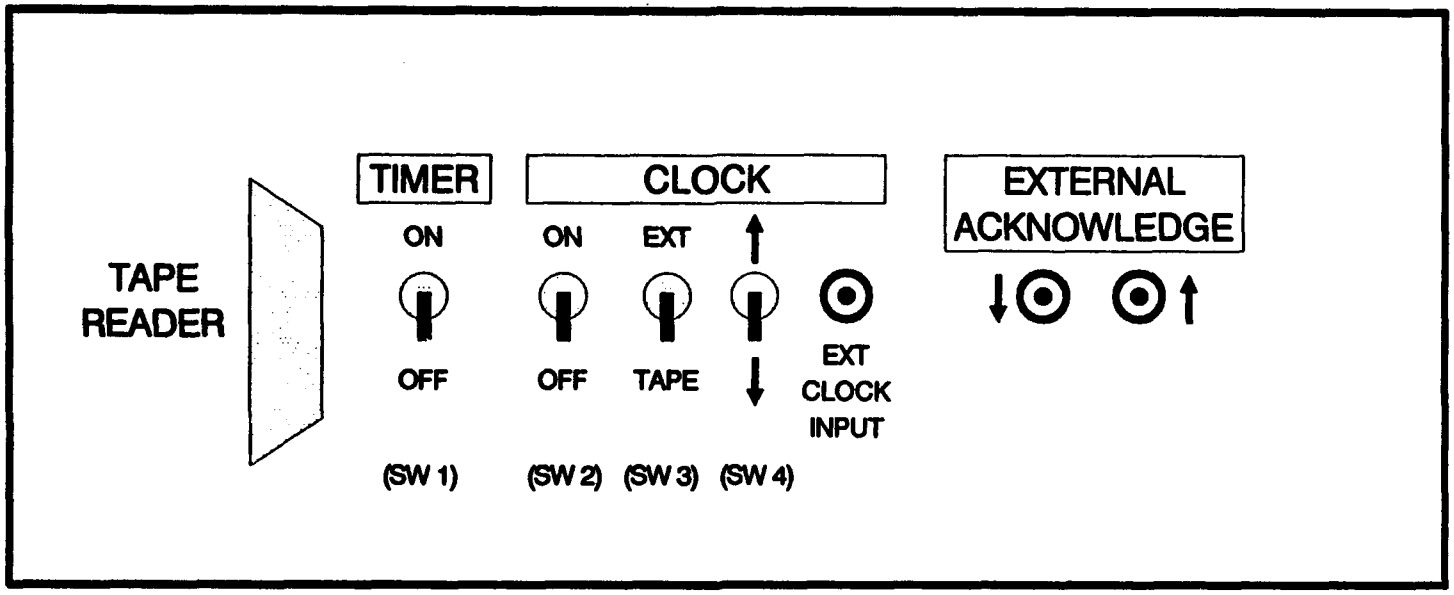


Figure A.1 Front Panel Drawing

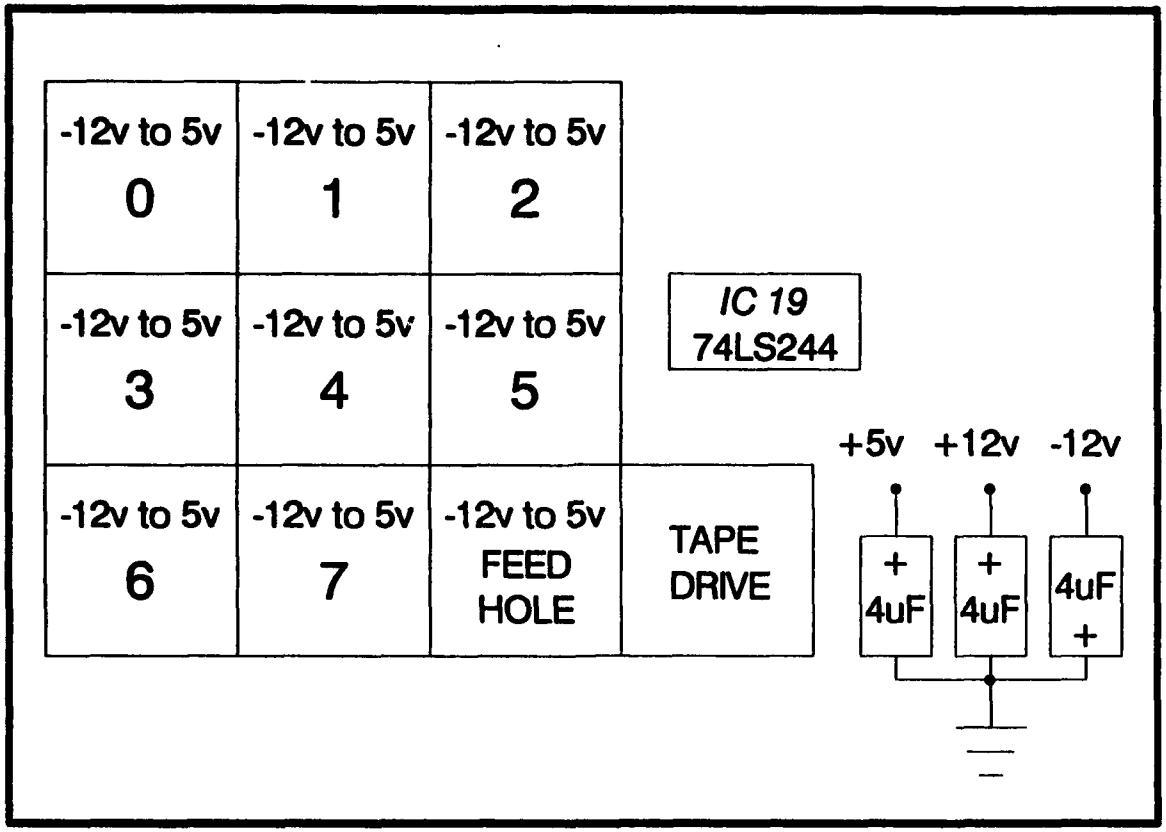


Figure A.2 Upper Board Drawing

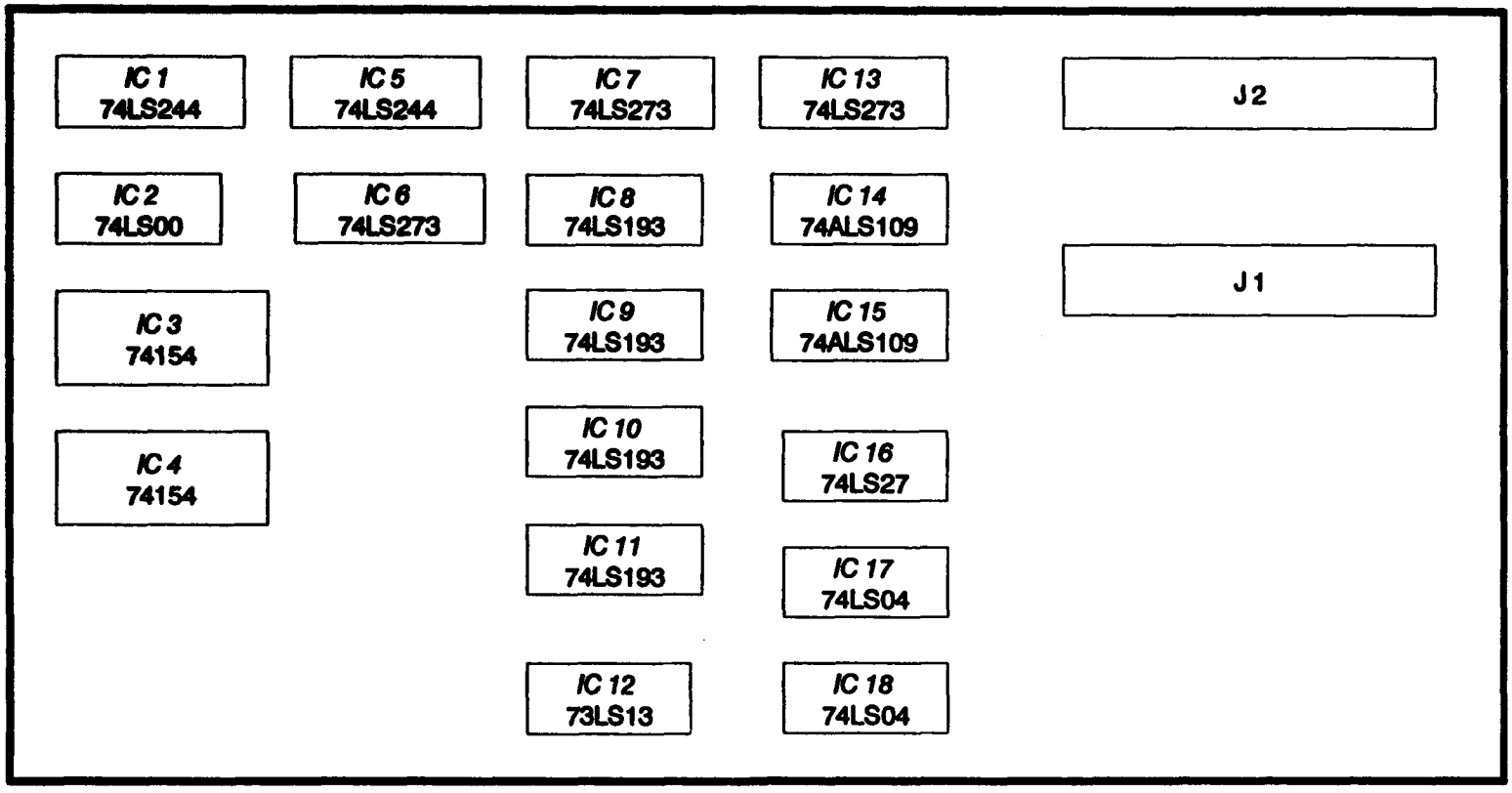
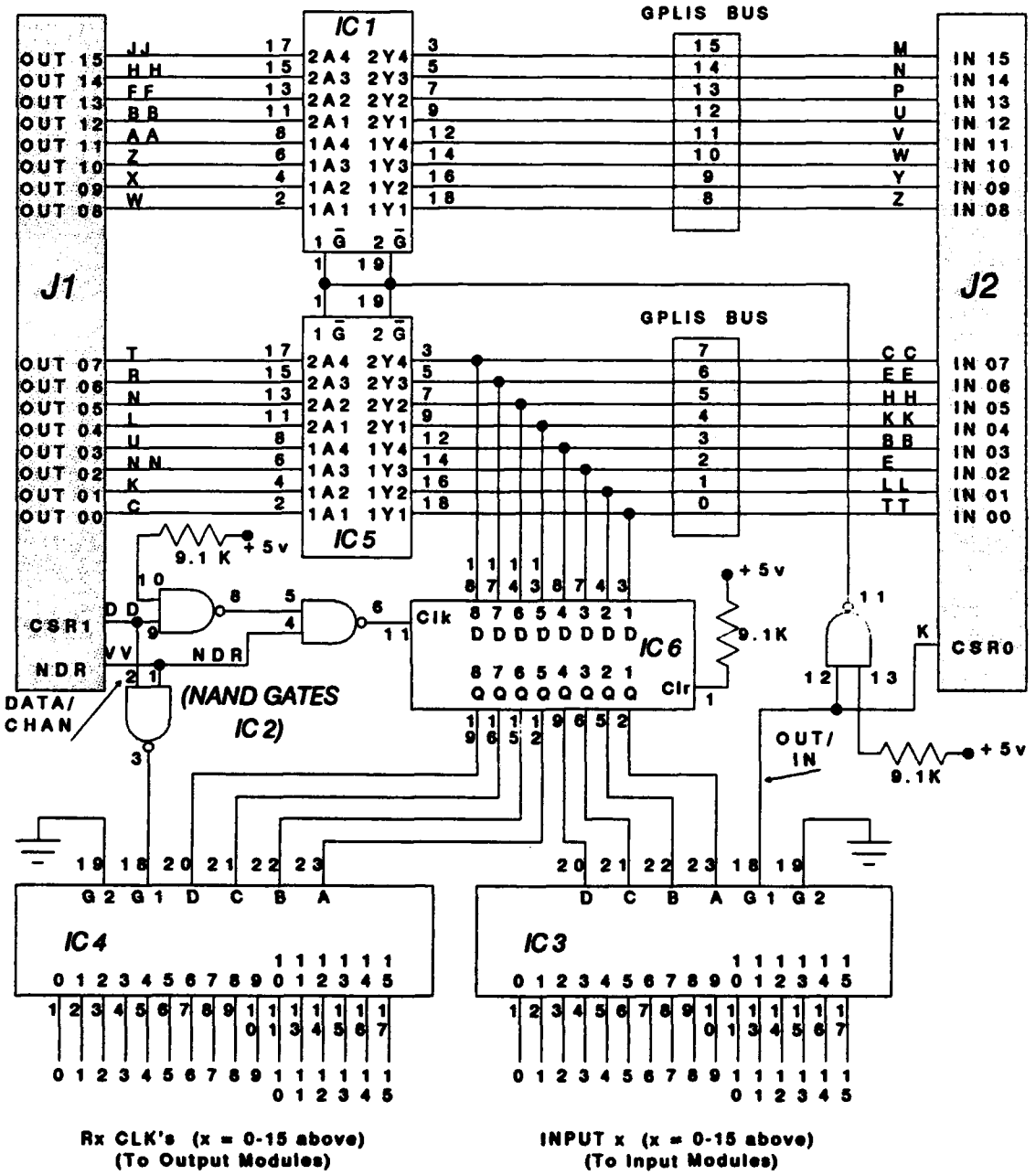


Figure A.3 Lower Board Drawing



(See Figure A.3 (Appendix A) for IC identifications)

Figure A.4 "GPLIS Master Logic Board" Detailed Drawing
(As Implemented)

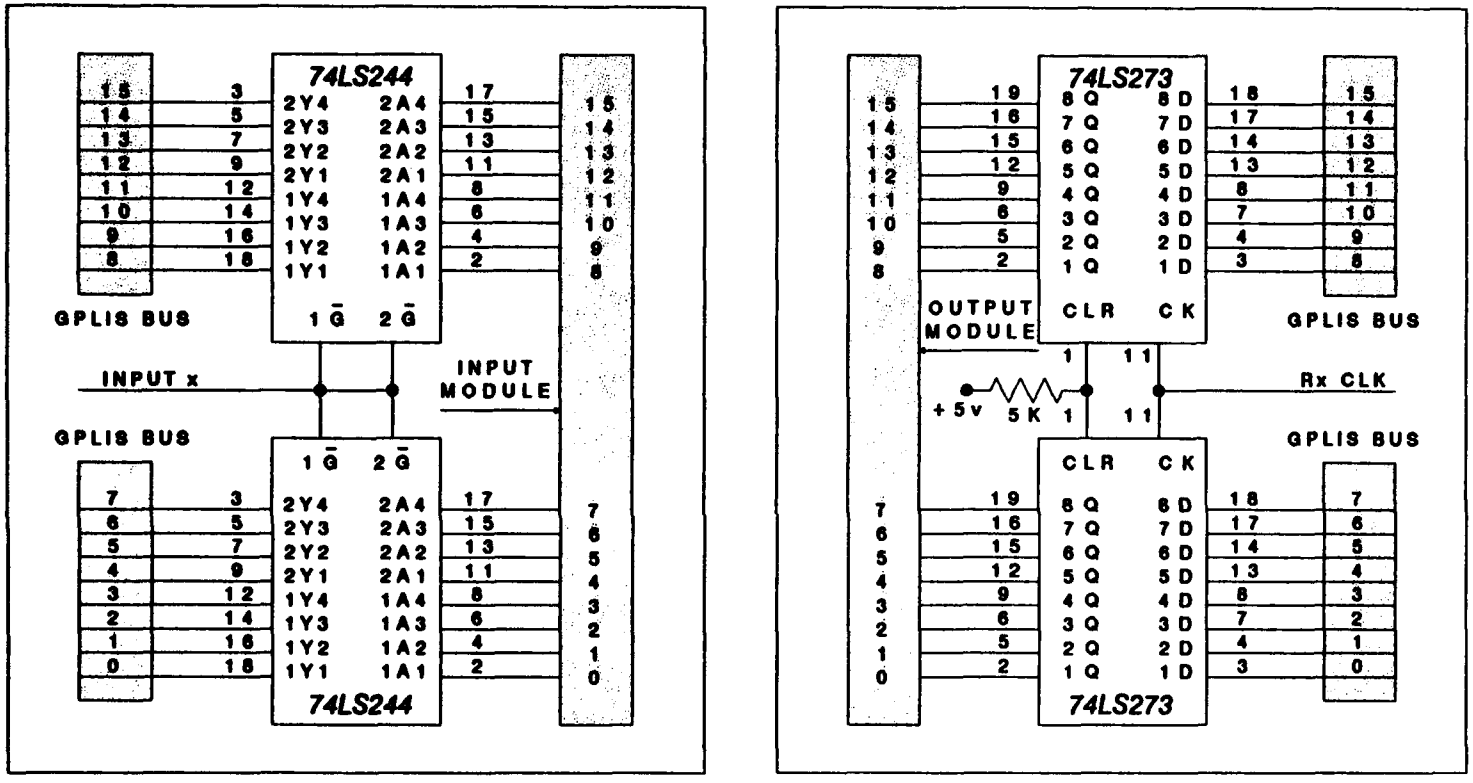


Figure A.5 Typical GPLIS Channel Implementation Drawing
(For Future Modules)

Table A.1 PLU Register Word Formats

Word	Bit(s)	Function
DRCSR	15	REQUEST B--This bit is under control of the user's device and may be used to initiate an interrupt sequence or to generate a flag that may be tested by the program. When used as an interrupt request, it is asserted by the external device and initiates an interrupt provided the INT ENB B bit (bit 05) is also set. When used as a flag, this bit can be read by the program to monitor external device status. When the maintenance cable is used, the state of this bit is dependent on the state of CSR1 (bit 01). This permits checking interface operation by loading a 0 or 1 into CSR1 and then verifying that REQUEST B is the same value. Read-only bit. Cleared by INIT when in maintenance mode.
	14-08	Not used. Read as 0.
	07	REQUEST A--Performs the same function as REQUEST B (bit 15) except that an interrupt is generated only if INT ENB A (bit 06) is also set. When the maintenance cable is used, the state of REQUEST A is identical to that of CSR) (bit 00). Read-only bit. Cleared by INIT when in maintenance mode.
	06	INT ENB A--Interrupt enable bit. When set, allows an interrupt request to be generated, provided REQUEST A (bit 07) becomes set. Can be loaded or read by the program (read/write bit). Cleared by INIT.

Table A.1 (continued) PLU Register Word Formats

Word	Bit(s)	Function
DRCSR (cont.)	05	INT ENB B--Interrupt enable bit. When set, allows an interrupt request to be generated, provided REQUEST B (bit 15) becomes set.
	04-02	Not used. Read as 0.
		Can be loaded or read by the program (read/write bit). Cleared by INIT.
	01	CSR1--This bit can be loaded or read (under program control) and can be used for a user-defined command to the device (appears only on Connector No. 1).
		When the maintenance cable is used, setting or clearing this bit causes an identical state in bit 15 (REQUEST B). This permits checking operation of bit 15 which cannot be loaded by the program.
		Can be loaded or read by the program (read/write bit). Cleared by INIT.
	00	CSR0--Performs the same functions as CSR1 (bit 01) but appears only on Connector No. 2.
		When the maintenance cable is used, the state of this bit controls the state of bit 07 (REQUEST A).
		Read/write bit. Cleared by INIT.
DROUTBUF	15-00	Output Data Buffer--Contains a full 16-bit word or one or two 8-bit bytes: High Byte = 15-8; Low Byte = 7-0.
		Loading is accomplished under a program-controlled DATO or DATOB bus cycle. It can be read under a program-controlled DATI cycle.

Table A.1 (continued) PLU Register Word Formats

Word	Bit(s)	Function
DRINBUF	15-00	Input Data Buffer--Contains a full 16-bit word or one or two 8-bit bytes. The entire 16-bit word is read under a program-controlled DATI bus cycle.

Source: Ref 3:6-15,6-16.

APPENDIX B

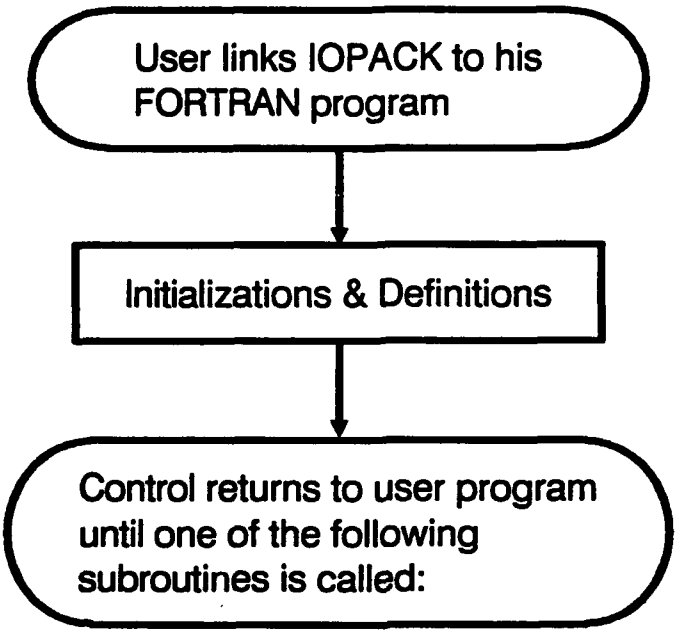
SOFTWARE FLOW CHARTS AND PROGRAM LISTINGS

APPENDIX B

SOFTWARE FLOW CHARTS AND PROGRAM LISTINGS

Contents

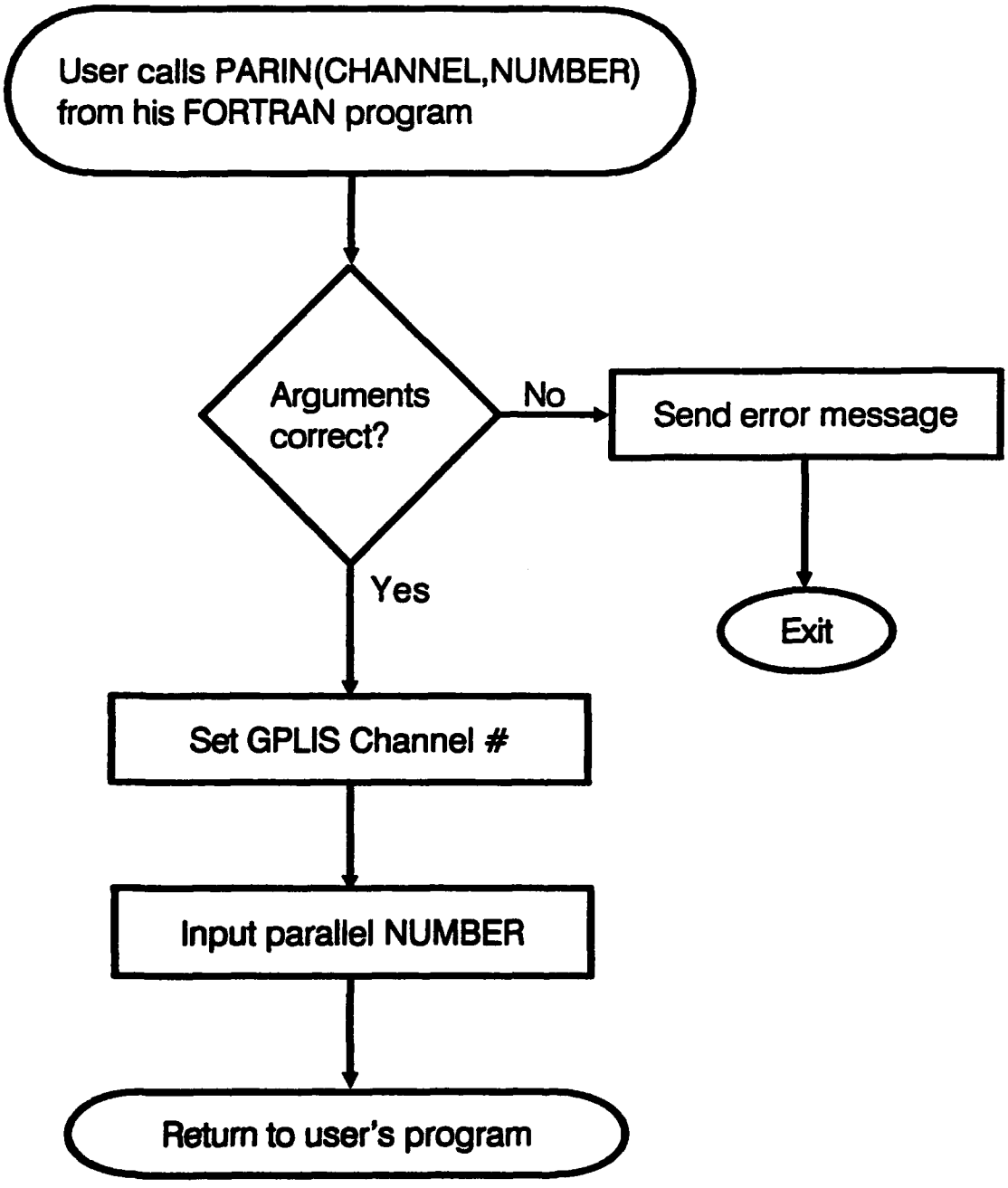
IOPACK Subroutine Package Top Level Flow Chart



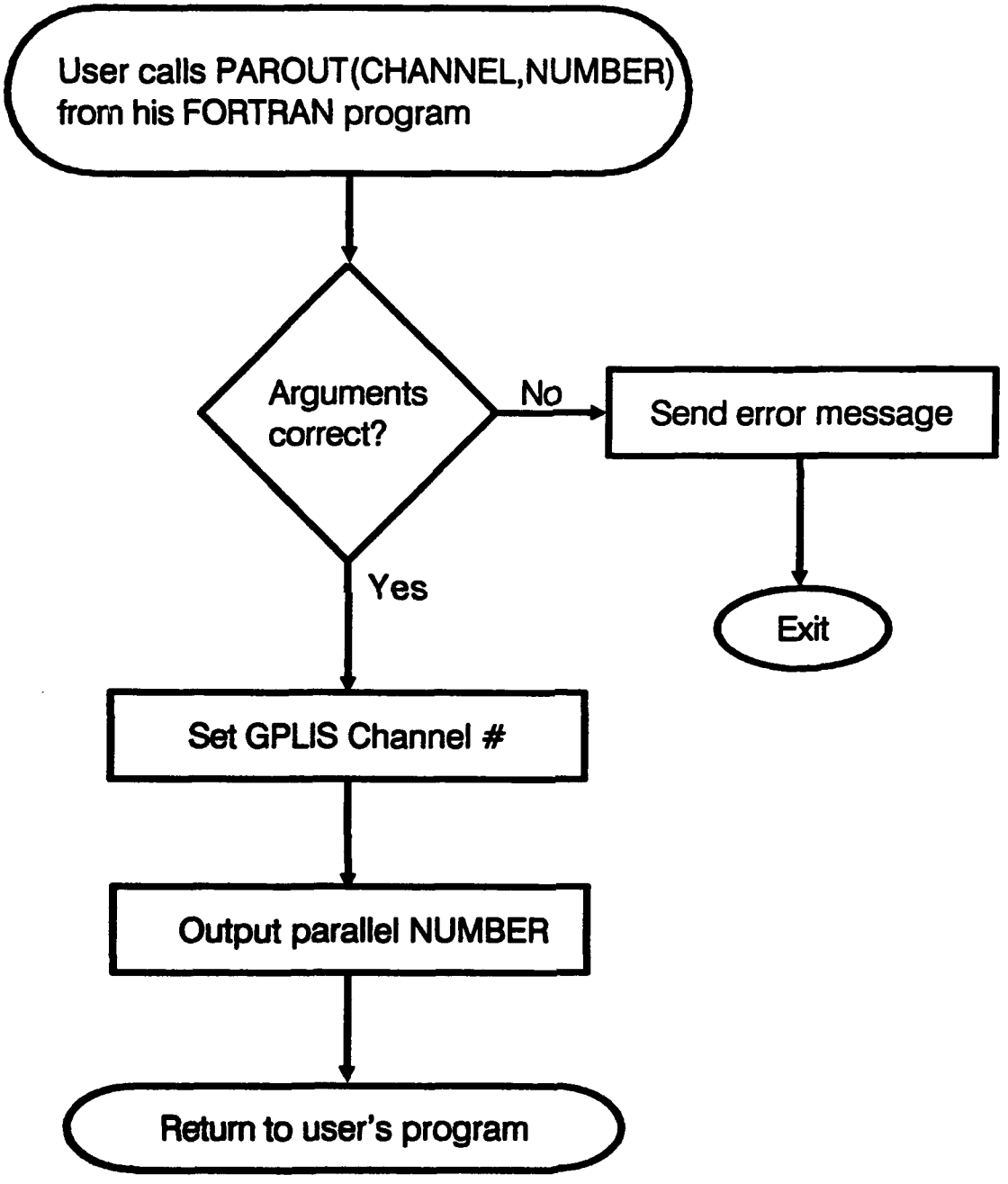
PARIN(CHANNEL,NUMBER)
PAROUT(CHANNEL,NUMBER)
SERIN(NUMBER)
SEROUT(NUMBER)
INIT
INTON
INTOFF
WAITA
WAITB

(Individual flow charts for each of the above are provided on the following pages)

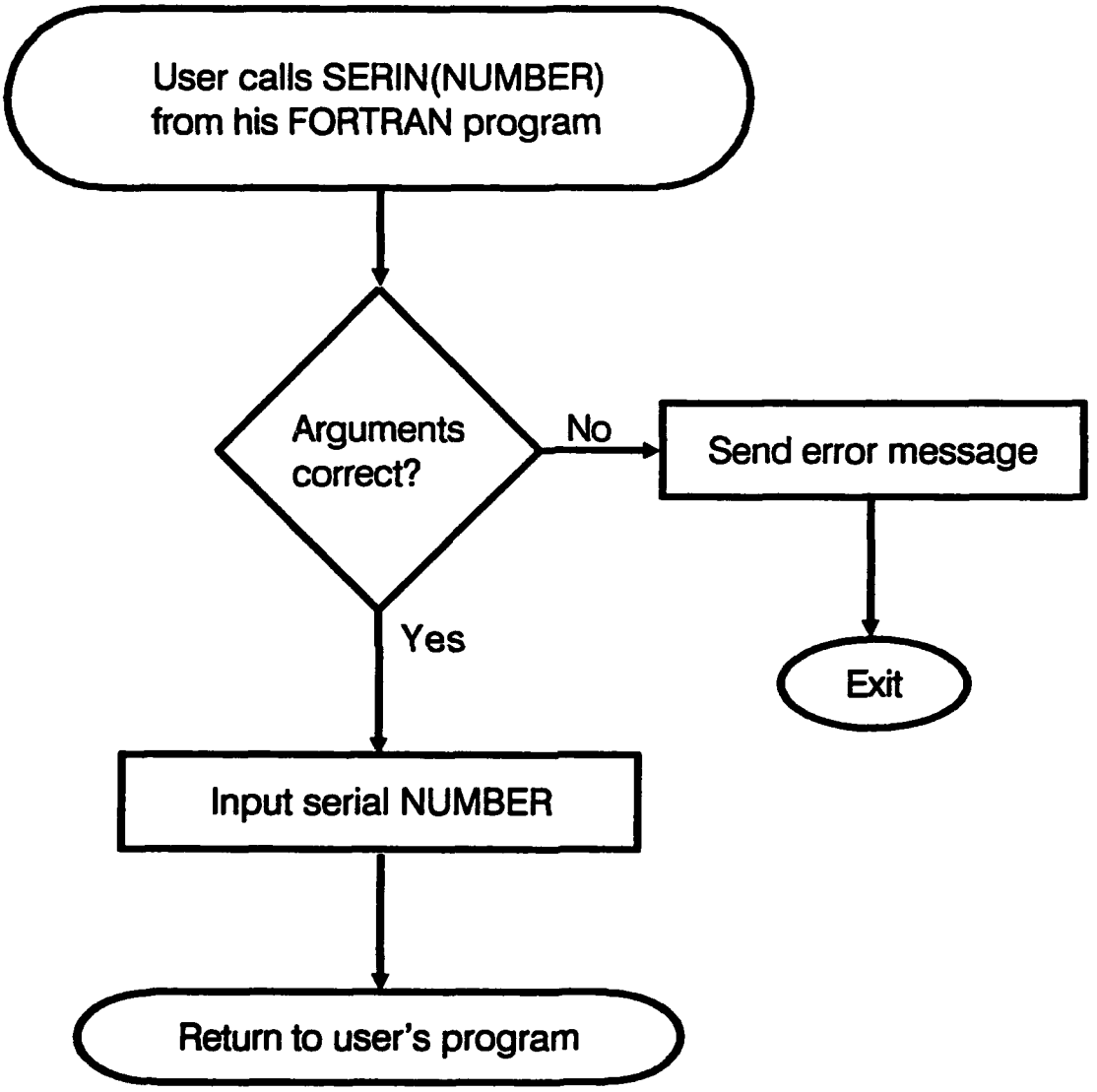
Subroutine PARIN Flow Chart



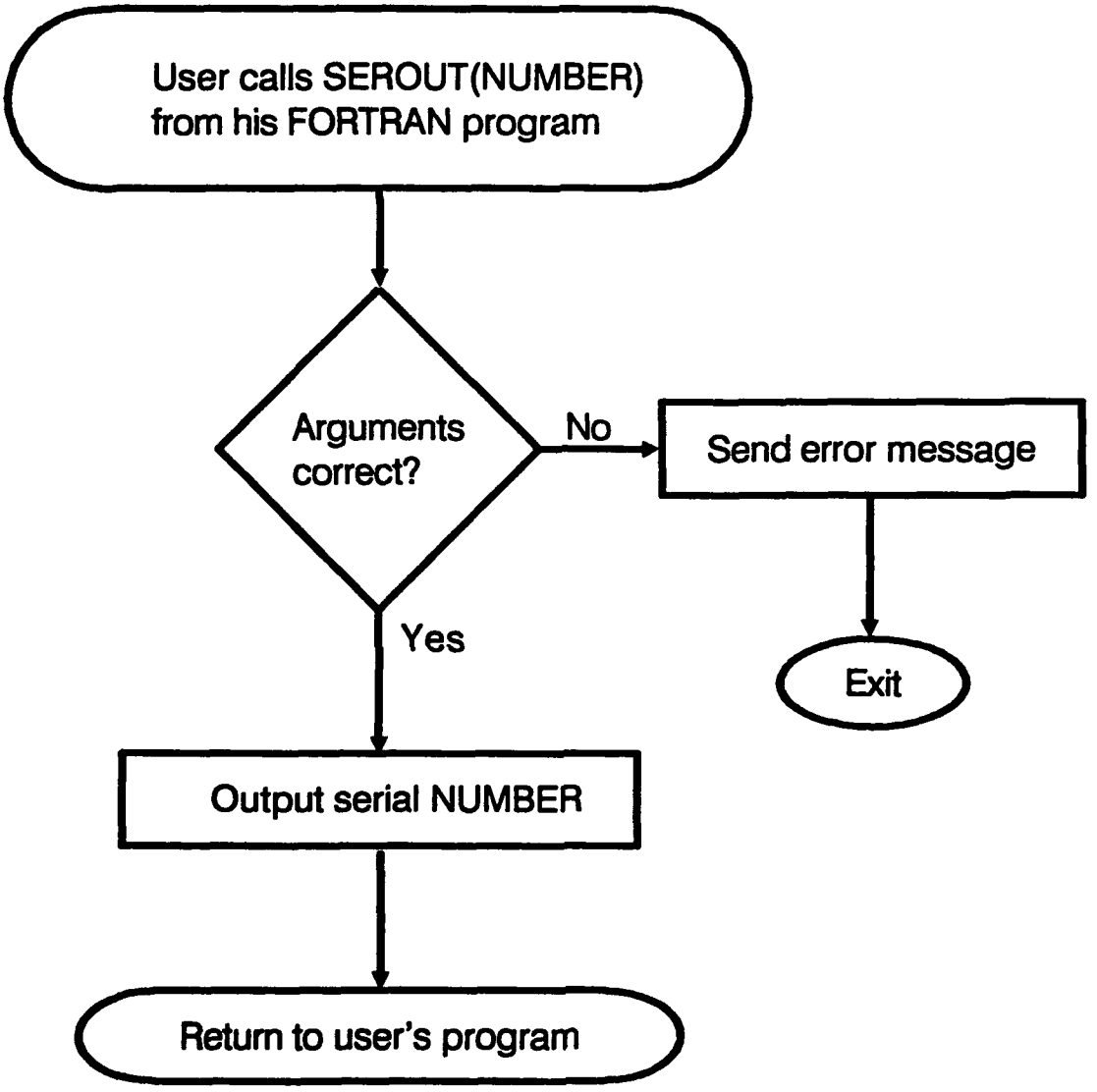
Subroutine PAROUT Flow Chart



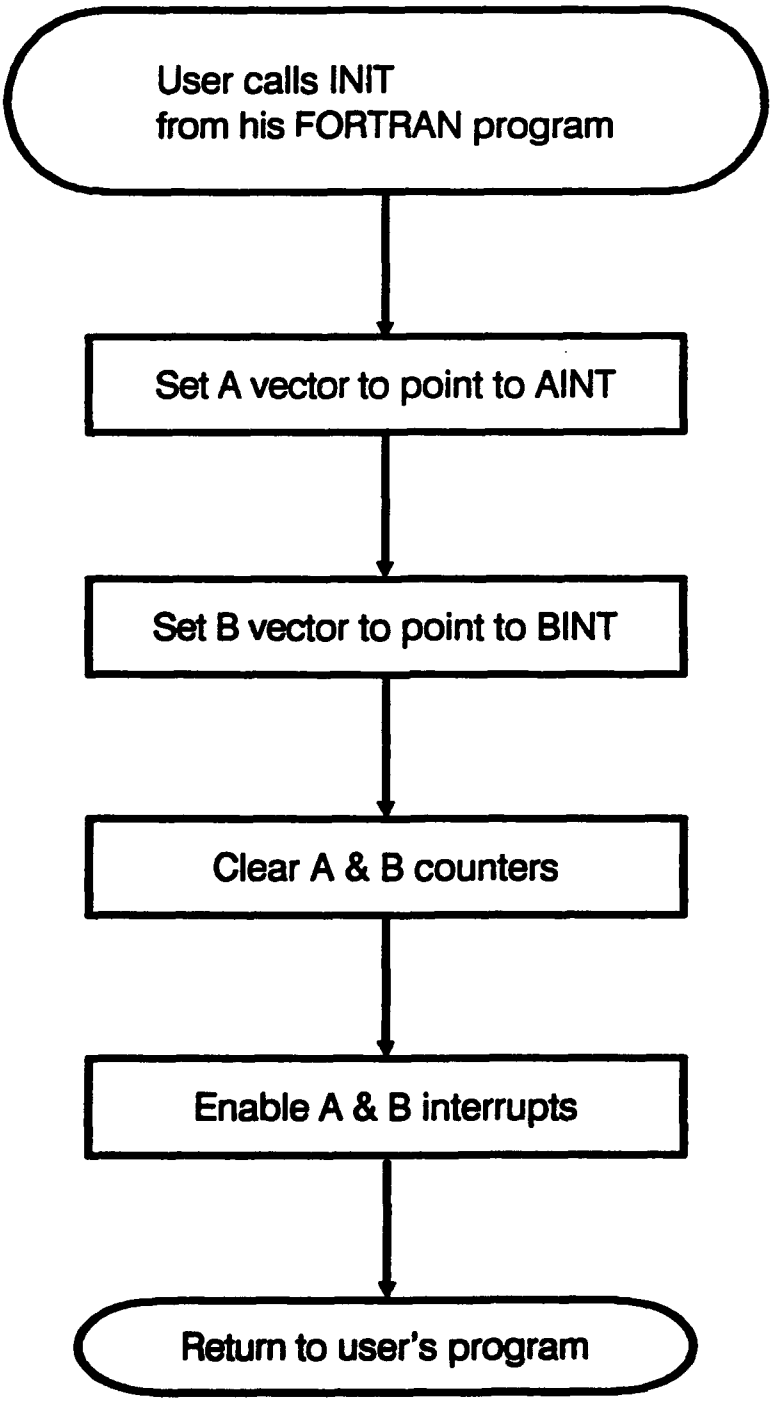
Subroutine SERIN Flow Chart



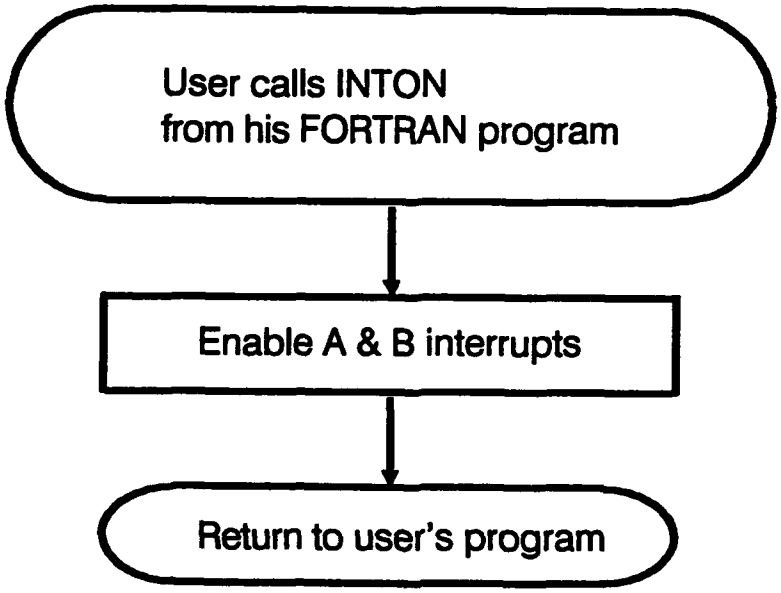
Subroutine SEROUT Flow Chart



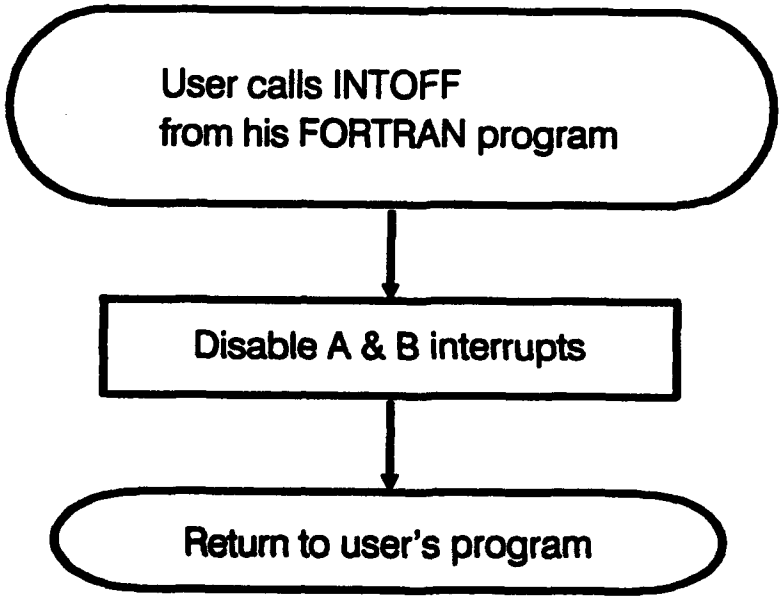
Subroutine INIT Flow Chart



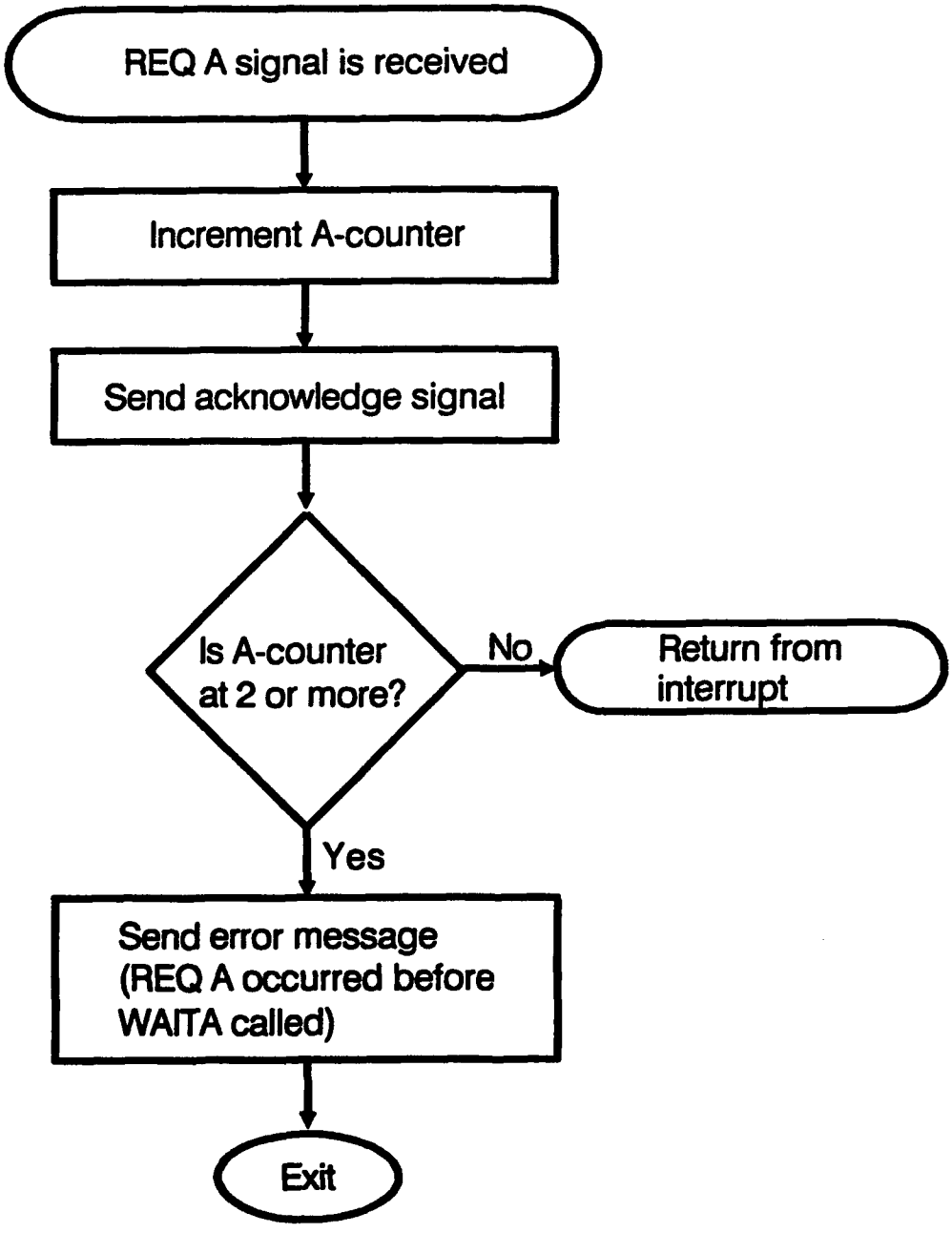
Subroutine INTON Flow Chart



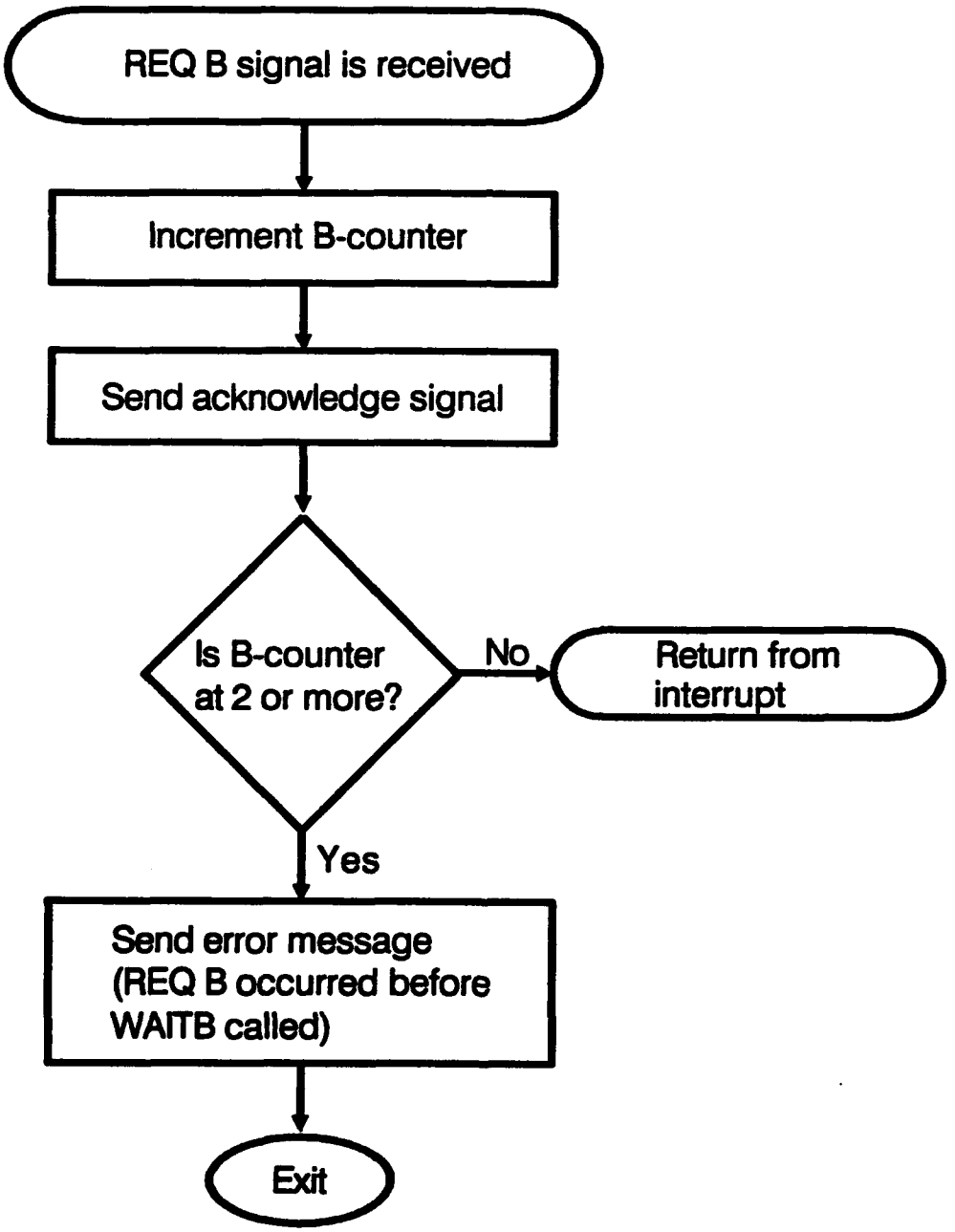
Subroutine INTOFF Flow Chart



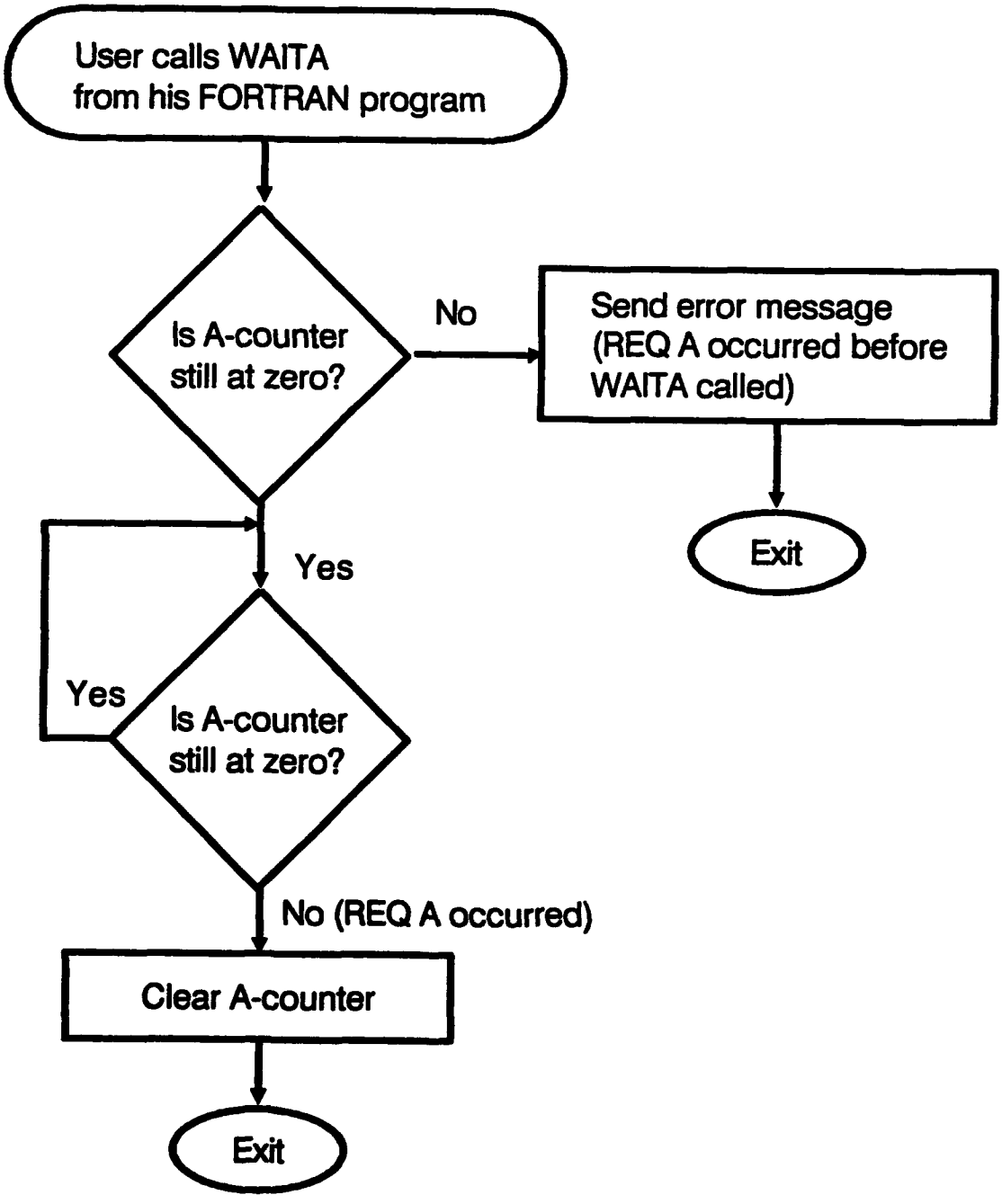
Interrupt Routine AINT Flow Chart



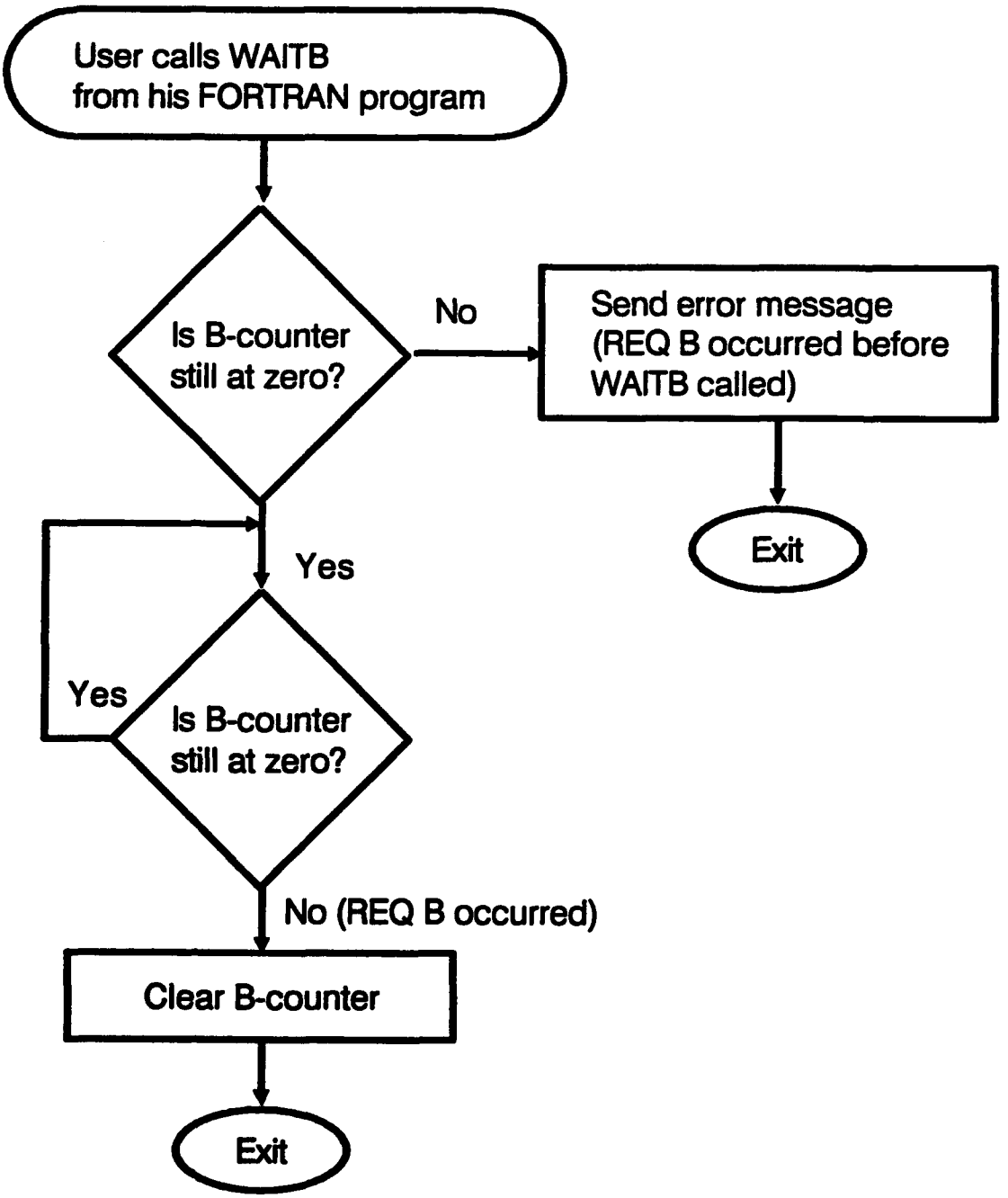
Interrupt Routine BINT Flow Chart



Subroutine WAITA Flow Chart



Subroutine WAITB Flow Chart



IOPACK Subroutine Package

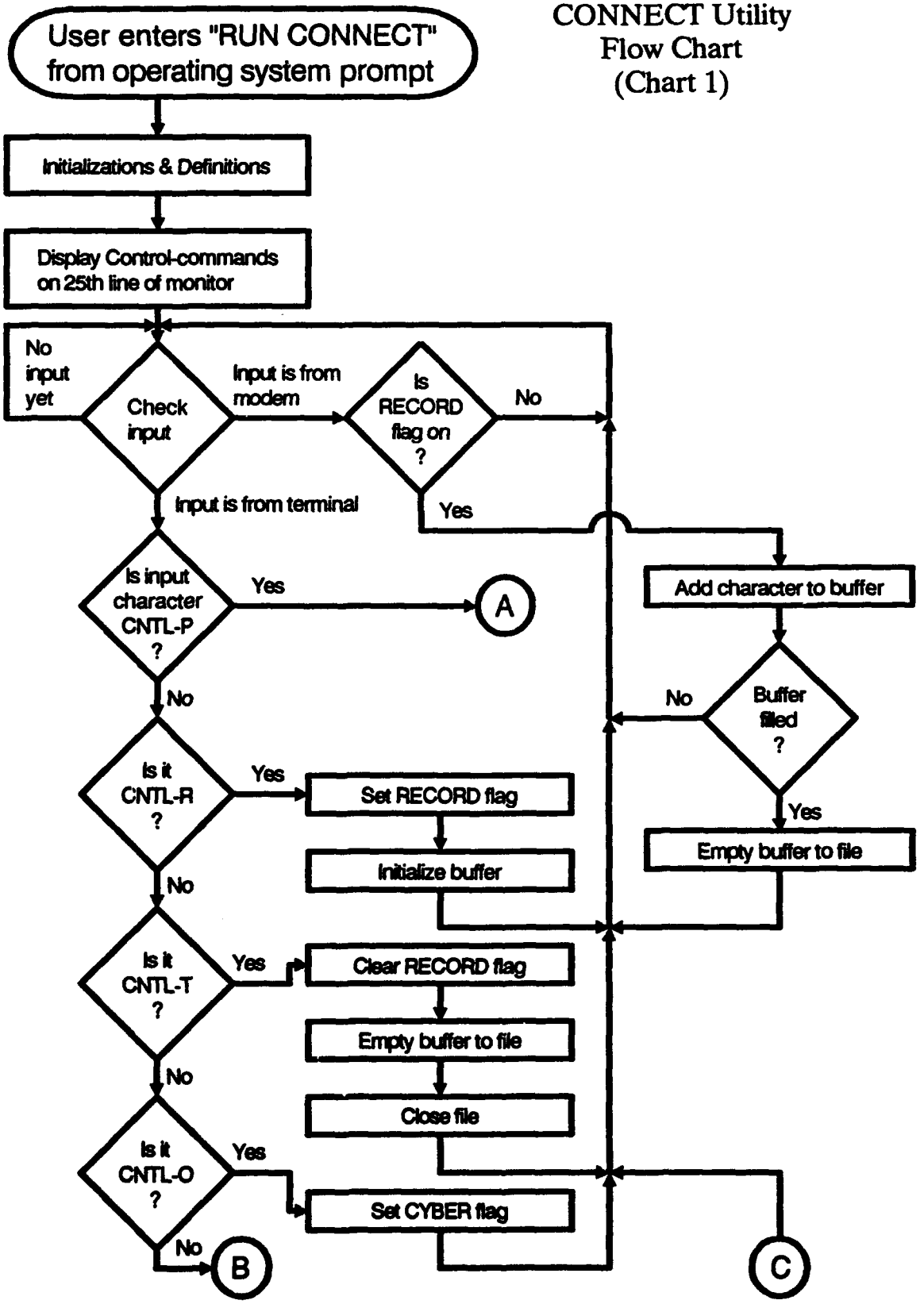
Program Listing

```
        .TITLE  IOPACK
        .MCALL  .PRINT,.EXIT
        .GLOBL  PARIN,PAROUT,SERIN,SEROUT
        .GLOBL  WAITA,WAITB,INIT,INTON,INTOFF
        DRCSR=167770
        DROUTB=167772
        DRINBU=167774
        MRCSR=175610
        MRBUF=175612
        MXCSR=175614
        MXBUF=175616
        R0=%0
        R1=%1
        R2=%2
        R5=%5
        PC=%7
START:  NOP
;       PARIN(CHANNEL,NUMBER)
PARIN:  JSR     PC,CHKARG
        BIS     #1,DRCSR                ;OUT/IN=1
        BIC     #2,DRCSR                ;DATA/CHAN=0 (SET CHANNEL)
        MOV     R0,DROUTB               ;WRITE CHANNEL
        BIC     #1,DRCSR                ;OUT/IN=0 (INPUT)
        MOV     DRINBU,@(R5)+           ;INPUT DATA
        RTS     PC
;       PAROUT(CHANNEL,NUMBER)
PAROUT: JSR     PC,CHKARG
        BIS     #1,DRCSR                ;OUT/IN=1
        BIC     #2,DRCSR                ;DATA/CHAN=0 (SET CHAN)
        ASL     R0                      ;SHIFT CHAN FOUR BITS
        ASL     R0                      ;  LEFT SO THE NUMBER
        ASL     R0                      ;  IS INTERPRETED AS
        ASL     R0                      ;  AN OUTPUT CHAN
        MOV     R0,DROUTB               ;WRITE CHANNEL
        BIS     #3,DRCSR                ;OUT/IN=1,DATA/CHAN=1
        MOV     @(R5)+,DROUTB           ;OUTPUT DATA
        RTS     PC
;
CHKARG: CMP     (R5)+,#2                ;ARE THERE 2 ARGS?
        BNE     ERARG                   ;IF NO THEN ERROR
        MOV     @(R5)+,R0               ;R0=CHANNEL
        TST     R0                      ;CHANNEL < 0 ?
        BLT     ERCHAN                  ;IF YES THEN ERROR
        CMP     R0,#17                  ;CHANNEL > 15 DECIMAL?
        BGT     ERCHAN                  ;IF YES THEN ERROR
        RTS     PC
;
ERARG:  .PRINT  #MSARG
        .EXIT
MSARG:  .ASCII  /WRONG NUMBER OF ARGUMENTS/
        .ASCIZ  /IN CALL TO PARIN OR PAROUT/<07>
        .EVEN
ERCHAN: .PRINT  #MSCHAN
        .EXIT
MSCHAN: .ASCII  /CHANNEL NUMBER OUT OF THE/
        .ASCIZ  /RANGE OF 0 TO 15/<07>
        .EVEN
;
INIT:   MOV     #AINT,300               ;SET A VECTOR TO AINT
        MOV     #0,302
```

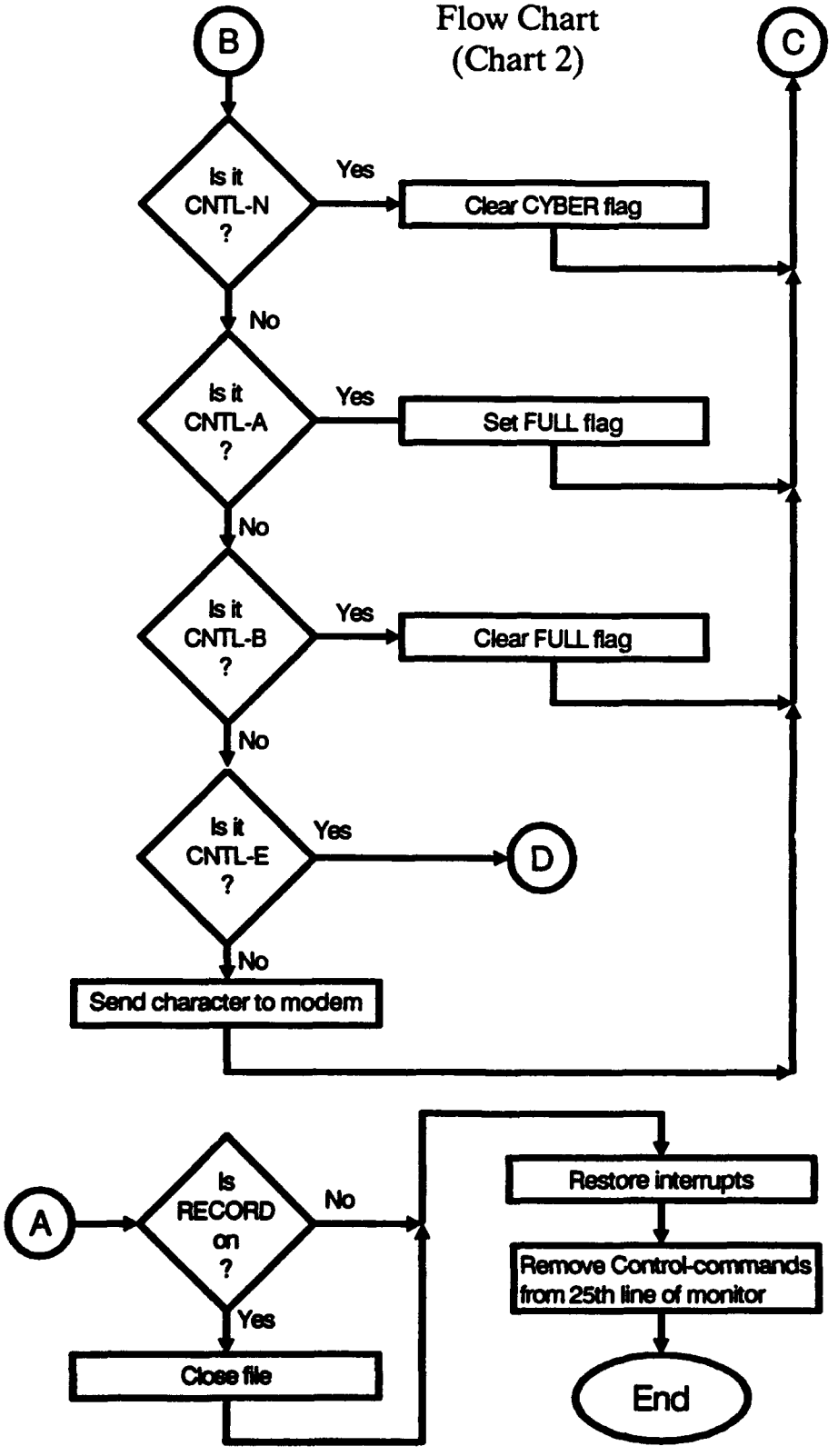
```
        MOV     #BINT,304               ;SET B VECTOR TO BINT
        MOV     #0,306
        CLR     ACOUNT                  ;CLEAR INTERRUPT COUNTER A
        CLR     BCOUNT                  ;CLEAR INTERRUPT COUNTER B
        BIS     #140,DRCSR              ;ENABLE A AND B INTERRUPTS
        RTS     PC
;       INTERRUPT SERVICE ROUTINES
AINT:   INC     ACOUNT                  ;INC INT COUNTER A
        JSR     PC,ACK                  ;ACKNOWLEDGE REQ A
        CMP     ACOUNT,#2               ;HAVE TWO REQ A'S GONE BY?
        BGE     A1                      ;IF YES THEN WE MISSED ONE
        RTI
A1:     .PRINT  #MISSA
        .EXIT
WAITA:  TST     ACOUNT                  ;HAS REQ A OCCURRED ALREADY?
        BEQ     A2                      ;IF NO GOTO A2
        .PRINT  #MISSA                  ;IF YES THEN WE MISSED IT
        .EXIT
A2:     TST     ACOUNT                  ;WAIT FOR REQ A
        BEQ     A2
        CLR     ACOUNT                  ;RESET
        RTS     PC                      ;RETURN
MISSA:  .ASCII  /REQ A OCCURRED BEFORE YOU CALLED WAITA/<15><12>
        .ASCIZ  /CORRECT YOUR PROGRAM/<07><15><12>
        .EVEN
ACOUNT: .WORD   0                       ;INTERRUPT A COUNTER
BINT:   INC     BCOUNT                  ;INC INT COUNTER B
        JSR     PC,ACK                  ;ACKNOWLEDGE REQ B
        CMP     BCOUNT,#2               ;HAVE TWO REQ B'S GONE BY?
        BGE     B1                      ;IF YES THEN WE MISSED ONE
        RTI
B1:     .PRINT  #MISSB
        .EXIT
WAITB:  TST     BCOUNT                  ;HAS REQ B OCCURRED ALREADY?
        BEQ     B2                      ;IF NO GOTO B2
        .PRINT  #MISSB                  ;IF YES THEN WE MISSED IT
        .EXIT
B2:     TST     BCOUNT                  ;WAIT FOR REQ B
        BEQ     B2
        CLR     BCOUNT                  ;RESET
        RTS     PC
MISSB:  .ASCII  /REQ B OCCURRED BEFORE YOU CALLED WAITB/<15><12>
        .ASCIZ  /CORRECT YOUR PROGRAM/<07><15><12>
        .EVEN
BCOUNT: .WORD   0                       ;INTERRUPT B COUNTER
ACK:    BIS     #1,DRCSR                ;READ GPLIS CHAN 0
        BIC     #2,DRCSR                ;TO ACK INTERRUPT
        MOV     #0,DROUTB
        BIC     #1,DRCSR
        MOV     DRINBU,DUMMY
        RTS     PC
DUMMY:  .WORD   0
;       SERIN(NUMBER)
SERIN:  JSR     PC,ARGCHK
S1:     TSTB    NRCSR                   ;WAIT FOR INPUT
        BPL     S1
        MOVB    NRBUF,@(R5)+            ;INPUT DATA
        RTS     PC
;       SEROUT(NUMBER)
SEROUT: JSR     PC,ARGCHK
```

```
S2:     TSTB    MXCSR                   ;WAIT FOR READY
        BPL     S2
        MOVB    @(R5)+,MXBUF            ;OUTPUT DATA
        RTS     PC
;
ARGCHK: CMP     (R5)+,#1                ;IS THERE ONE ARG?
        BNE     ERSAR                   ;IF NO THEN ERROR
        RTS     PC
;
ERSAR:  .PRINT  #MSSAR
        .EXIT
MSSAR:  .ASCII  /WRONG NUMBER OF ARGUMENTS/
        .ASCIZ  /IN CALL TO SERIN OR SEROUT/<07>
        .EVEN
;       INTON   TURNS INTERRUPT ENABLES ON
INTON:  BIS     #140,DRCSR              ;ENABLE A AND B INTERRUPTS
        RTS     PC
;       INTOFF  TURNS INTERRUPT ENABLES OFF
INTOFF: BIC     #140,DRCSR              ;DISABLE A AND B INTERRUPTS
        RTS     PC
        .END    START
```

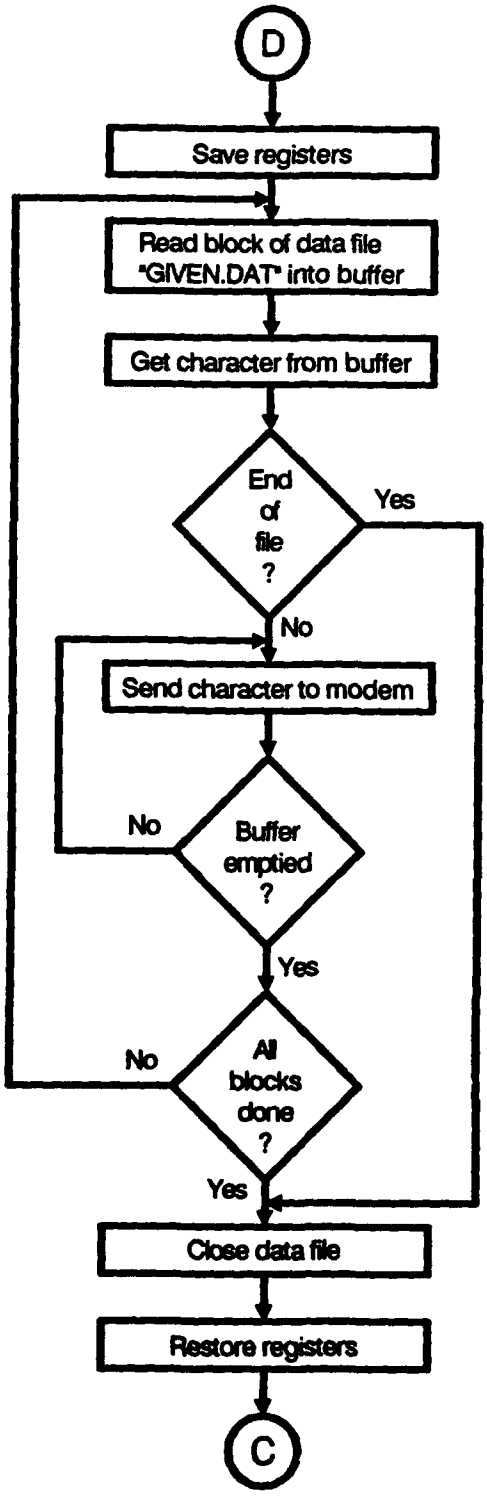
CONNECT Utility
Flow Chart
(Chart 1)



CONNECT Utility
Flow Chart
(Chart 2)



CONNECT Utility Flow Chart
(Chart 3)



CONNECT Utility

Program Listing

```
        .TITLE  CONNECT
;
;       THIS PROGRAM WILL WORK RELIABLY AT 1200 BAUD OR LESS
;
        .MCALL  .ENTER,.FETCH,.WRITE,.CLOSE,.EXIT,.PRINT
        .MCALL  .LOOKUP,.READW,.WRITW
        MRCSR=175610            ;MODEM RCSR
        MRBUF=175612            ;MODEM RBUF
        MXCSR=175614            ;MODEM XCSR
        MXBUF=175616            ;MODEM XBUF
        TRCSR=177560            ;TERMINAL RCSR
        TRBUF=177562            ;TERMINAL RBUF
        TXCSR=177564            ;TERMINAL XCSR
        TXBUF=177566            ;TERMINAL XBUF
        R0=%0                   ;USED BY MACRO
        R1=%1                   ;CHARACTER STORAGE
        R2=%2                   ;COUNTER
        R3=%3                   ;ADDRESS POINTER
        R4=%4                   ;COUNTER
        R5=%5                   ;COUNTER
        SP=%6                   ;STACK POINTER
        PC=%7                   ;PROGRAM COUNTER
;
;       INITIALIZATIONS
;
START:  BIC     #100,TRCSR      ;DISABLE TERM INTERRUPT
        BIC     #100,MRCSR      ;DISABLE MODEM INTERRUPT
        MOV     #INBUF2,HEAD1   ;POINT HEAD1 TO INBUF2
        MOV     #INBUF1,HEAD2   ;POINT HEAD2 TO INBUF1
        CLR     FULFLG          ;SET FOR HALF DUPLEX
        CLR     RECFLG          ;SET RECORD OFF
        MOV     #1,CYBFLG       ;SET FOR CYBER
        .PRINT  #LABELS         ;DISPLAY COMMANDS
        BR      INCHK
LABELS: .ASCII  <33><152>       ;SAVE CURSOR POSITION
        .ASCII  <33><170><61>   ;ENABLE 25TH LINE
        .ASCII  <33><131><70><40>       ;MOVE TO 25TH LINE
        .ASCII  /CNTL- A(FUL) B(HAF) R(REC ON) /
        .ASCII  /T(REC OFF) E(TRNS FILE) P(EX)/
        .ASCII  / O(CYBER) N(NOT CYBR)/
        .ASCIZ  <33><153>       ;RETURN TO SAVED POSITION
        .EVEN
;
;       TERM/MODEM INPUT CHECK LOOP
;
INCHK:  TSTB    TRCSR           ;TERMINAL INPUT?
        BMI     TERMIN
        TSTB    MRCSR           ;MODEM INPUT?
        BMI     MODIN
        BR      INCHK
;
;       TERMINAL INPUT HANDLER
;
TERMIN: MOVB    TRBUF,R1        ;CHAR TO R1
        CMP     R1,#20          ;CNTL-P  EXIT
        BEQ     FINISH
        CMP     R1,#22          ;CNTL-R  RECORD ON
        BEQ     RECON
        CMP     R1,#24          ;CNTL-T  RECORD OFF
        BEQ     RECOF1
```

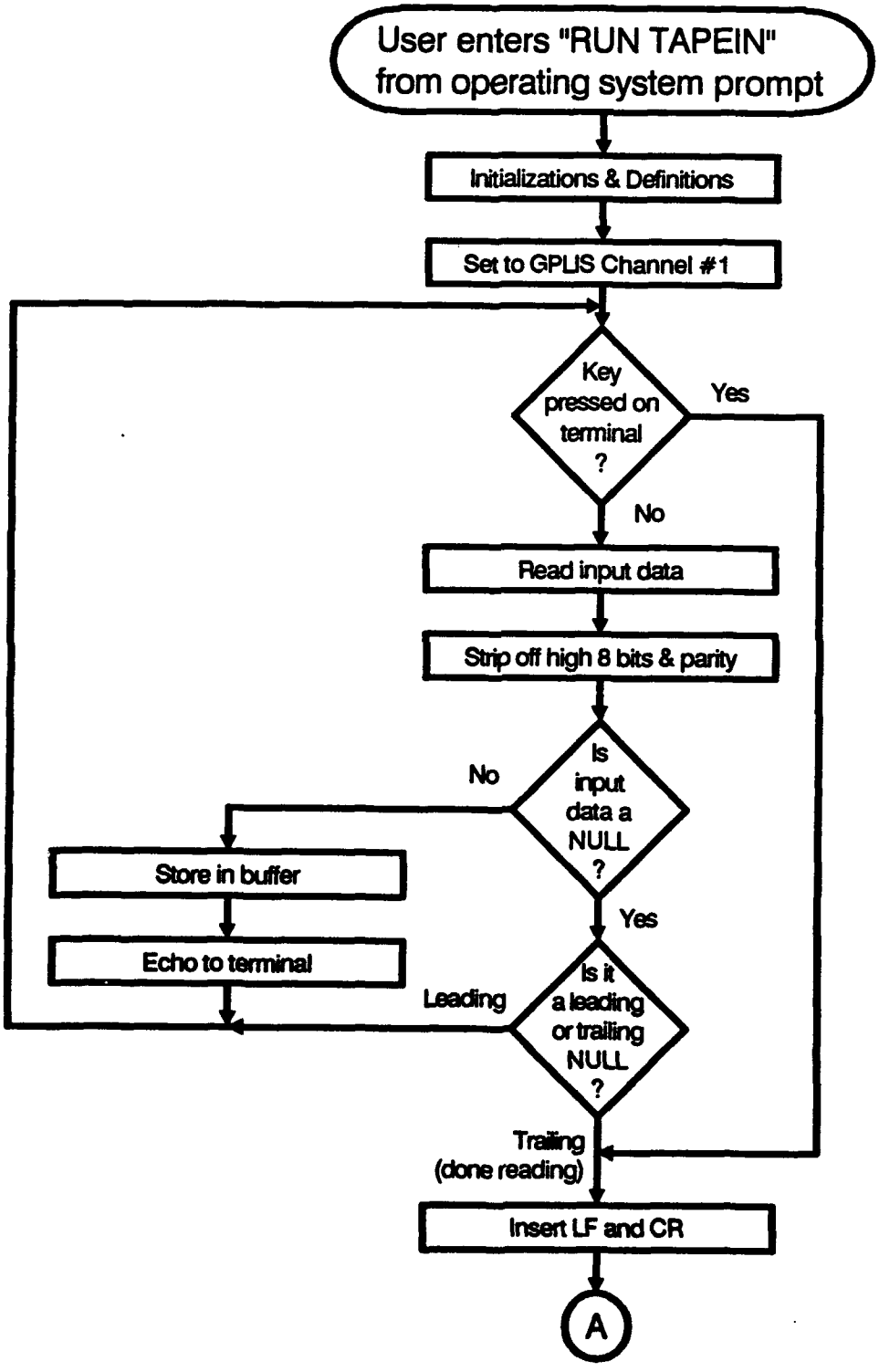
```
        CMP     R1,#17          ;CNTL-O  CYBER
        BEQ     CYBER
        CMP     R1,#16          ;CNTL-N  NOT CYBER
        BEQ     NOTCYB
        CMP     R1,#1           ;CNTL-A  SET FULL DUPLEX
        BEQ     FULSET
        CMP     R1,#2           ;CNTL-B  SET HALF DUPLEX
        BEQ     HAFSET
        CMP     R1,#5           ;CNTL-E  TRANSMIT FILE
        BEQ     TRANS1
TER1:   TSTB    MXCSR           ;WAIT FOR MODEM READY
        BPL     TER1
        MOVB    R1,MXBUF        ;SEND CHAR TO TERM
        TST     FULFLG          ;0 = HALF DUPLEX
        BNE     INCHK
TER2:   TSTB    TXCSR           ;WAIT FOR TERM READY
        BPL     TER2
        MOVB    R1,TXBUF        ;ECHO TO TERM
        BR      INCHK
RECOF1: JMP     RECOFF
TRANS1: JMP     TRANS
;
;       MODEM INPUT HANDLER
;
MODIN:  MOVB    MRBUF,R1        ;CHAR TO R1
MOD1:   TSTB    TXCSR           ;WAIT FOR TERM READY
        BPL     MOD1
        MOVB    R1,TXBUF        ;SEND TO TERM
        TST     RECFLG          ;RECORD ON?
        BEQ     INCHK           ;IF NO, BR INCHK
        JMP     RECORD          ;IF YES, RECORD
;
;       CYBER FLAG ROUTINES
;
CYBER:  MOV     #1,CYBFLG       ;SET FOR CYBER
        BR      INCHK
NOTCYB: CLR     CYBFLG          ;SET FOR NOT CYBER
        BR      INCHK
;
;       DUPLEX ROUTINES
;
FULSET: MOV     #1,FULFLG       ;SET FULL DUPLEX
        BR      INCHK
HAFSET: CLR     FULFLG          ;SET HALF DUPLEX
        BR      INCHK
;
;       EXIT ROUTINE
;
FINISH: TST     RECFLG          ;RECORD ON?
        BEQ     FIN1
        JSR     PC,OFF          ;CLOSE FILE
FIN1:   BIS     #100,TRCSR      ;REENABLE TERM INTERRUPT
        BIS     #100,MRCSR      ;REENABLE MODEM INTERRUPT
        .PRINT  #OFFLAB         ;TURN OFF LABELS
        .EXIT
OFFLAB: .ASCIZ  <33><171><61>   ;DISABLE 25TH LINE
        .EVEN
;
;       RECORD ON ROUTINE
;
```

```
RECON:  TST     RECFLG          ;IS RECORD ALREADY ON?
        BNE     ALON
        MOV     #1,RECFLG       ;TURN RECORD ON
        .ENTER  #IOBLK0,#0,#FILNAM,#-1
        BCS     ERENT           ;ERROR IN ENTER
        MOV     #INBUF1,R3      ;POINT R3 TO INPUT BUFFER 1
        CLR     R5              ;SET BLOCK NUMBER TO ZERO
        MOV     R3,R4           ;POINT R4 TO INPUT BUFFER
        CLR     R2              ;SET CHAR COUNT TO ZERO
        JMP     INCHK
ALON:   .PRINT  #ALON1
        JMP     INCHK
ALON1:  .ASCIZ  /RECORD IS ALREADY ON!/<07>
        .EVEN
ERENT:  .PRINT  #ERENT1
        BR      FINISH
ERENT1: .ASCIZ  /FATAL--ERROR IN ENTER/<07>
        .EVEN
FILNAM: .RAD50  /DK/
        .RAD50  /TAKEN DAT/
;
;       RECORD OFF ROUTINE
;
RECOFF: TST     RECFLG          ;IS RECORD ALREADY OFF?
        BEQ     ALOFF
        CLR     RECFLG          ;TURN RECORD OFF
        JSR     PC,OFF
        JMP     INCHK
OFF:    INC     R2
        ROR     R2              ;R2 IS NOW WORD COUNT
        .WRITW  #IOBLK0,#0,R3,R2,R5
        BCS     ERWRI
        .CLOSE  #0
        RTS     PC
ALOFF:  .PRINT  #ALOFF1
        JMP     INCHK
ALOFF1: .ASCIZ  /RECORD IS ALREADY OFF!/<07>
        .EVEN
;
;       RECORD CHARACTER
;
RECORD: TSTB    R1              ;NULL?
        BNE     REC1            ;IF NO, THEN RECORD IT
        JMP     INCHK           ;IF YES, THEN SKIP IT
REC1:   MOVB    R1,(R4)+        ;PUT CHAR IN BUFFER
        INC     R2              ;INC CHAR COUNT
        CMP     R2,#1000        ;BUFFER FILLED?
        BGE     WRIBUF          ;IF YES, THEN WRITE BUFFER
        JMP     INCHK           ;IF NO, GET MORE CHARS
WRIBUF: .WRITE  #IOBLK0,#0,R3,#400,R5
        BCS     ERWRI           ;ERROR IN WRITE
        INC     R5              ;INC BLOCK NUMBER
        MOV     -2(R3),R3       ;POINT R3 TO OTHER BUFFER
        MOV     R3,R4           ;POINT R4 TO OTHER BUFFER
        CLR     R2              ;SET CHAR COUNT TO ZERO
        JMP     INCHK           ;GET MORE CHARS
ERWRI:  .PRINT  #ERWRI1
        JMP     FINISH
ERWRI1: .ASCIZ  /FATAL--ERROR IN WRIT(E)(W)/<07>
        .EVEN
```

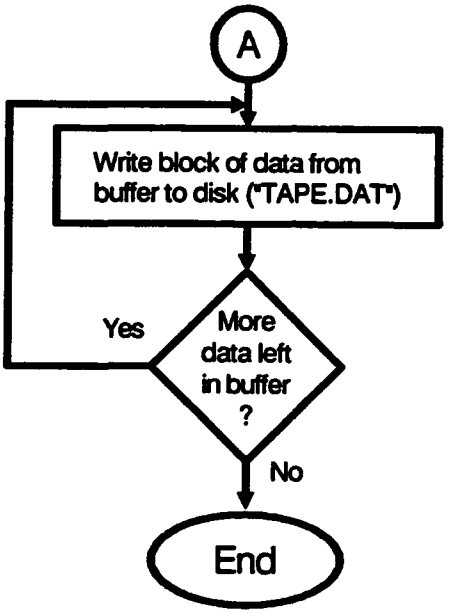
```
IOBLK0: .BLKW   10              ;MACRO'S SCRATCH SPACE
;
;       TRANSMIT ROUTINE
;
TRANS:  MOV     R1,R1STO        ;SAVE REGISTERS
        MOV     R2,R2STO
        MOV     R3,R3STO
        MOV     R4,R4STO
        MOV     R5,R5STO
        CLR     R5              ;SET BLOCK NUMBER TO ZERO
        .LOOKUP #IOBLK1,#1,#FILOUT
        BCS     ERLOO           ;ERROR IN LOOKUP
        MOV     R0,R1           ;R1= # BLOCKS IN FILE
        BPL     READ            ;IF R1<0 THEN EMPTY FILE
        JMP     EMPTY
READ:   .READW  #IOBLK1,#1,#INBUF1,#400,R5
BCC     TRA1
        JMP     ERREA           ;ERROR IN READ
TRA1:   INC     R5              ;INC BLOCK NUMBER
        MOV     #INBUF1,R4      ;POINT R4 TO INBUF1
        CLR     R2              ;SET CHAR COUNT TO ZERO
WRITE:  CMPB    (R4),#0         ;NULL?
        BEQ     EOF
        CMPB    (R4),#200       ;NULL? (WITH PARITY)
        BEQ     EOF
        CMPB    (R4),#12        ;LINEFEED?
        BEQ     LF
        CMPB    (R4),#212       ;LINEFEED? (WITH PARITY)
        BNE     TRA2
LF:     TST     CYBFLG          ;SKIP OVER LF ONLY IF CYBER
        BEQ     TRA2
        TST     CRFLG           ;WAS LAST CHAR CR?
        BEQ     TRA2            ;IF NO DON'T SKIP OVER
        BR      TEST            ;IF YES SKIP OVER THIS LF
TRA2:   TSTB    MXCSR           ;WAIT FOR MODEM READY
        BPL     TRA2
        MOVB    (R4),MXBUF      ;SEND CHAR TO MODEM
        TST     FULFLG          ;0 = HALF DUPLEX
        BEQ     TRA4
TRA3:   TSTB    MRCSR           ;WAIT FOR ECHO
        BPL     TRA3
        MOVB    MRBUF,(R4)      ;PUT ECHO IN (R4)
TRA4:   TSTB    TXCSR           ;WAIT FOR TERM READY
        BPL     TRA4
        MOVB    (R4),TXBUF      ;ECHO TO TERM
CRTST:  CMPB    (R4),#15        ;CARRIAGE RETURN?
        BEQ     WAITPR
        CMPB    (R4),#215       ;CR? (WITH PARITY)
        BEQ     WAITPR
        CLR     CRFLG           ;NOT A CR, SO CLEAR FLAG
TEST:   INC     R4              ;POINT TO NEXT CHAR
        INC     R2              ;INC CHAR COUN
        CMP     R2,#1000        ;BUFFER EMPTIED?
        BLT     WRITE
        CMP     R5,R1   ;ALL BLOCKS DONE?
        BLT     READ
EOF:    .CLOSE  #1
        MOV     R1STO,R1        ;RESTORE REGISTERS
        MOV     R2STO,R2
        MOV     R3STO,R3
```

```
        MOV     R4STO,R4
        MOV     R5STO,R5
        JMP     INCHK
WAITPR: TST     CYBFLG          ;WAIT FOR PROMPT ONLY IF CYBER
        BEQ     TEST
        MOV     #1,CRFLG        ;SET CRFLG
TRA5:   TSTB    MRCSR           ;WAIT FOR PROMPT
        BPL     TRA5
        MOVB    MRBUF,R3        ;PROMPT TO R3
TRA6:   TSTB    TXCSR           ;WAIT FOR TERM READY
        BPL     TRA6
        MOV     R3,TXBUF        ;SEND PROMPT TO TERM
        BR      TEST
ERLOO:  .PRINT  #ERLOO1
        JMP     FINISH
ERLOO1: .ASCII  /FATAL--ERROR IN LOOKUP/<15><12>
        .ASCIZ  /BE SURE FILE "GIVEN.DAT" EXISTS/<07>
        .EVEN
EMPTY:  .PRINT  #EMPTY1
        JMP     INCHK
EMPTY1: .ASCIZ  /FILE "GIVEN.DAT" IS EMPTY/<07>
        .EVEN
ERREA:  .PRINT  #ERREA1
        JMP     FINISH
ERREA1: .ASCIZ  /FATAL--ERROR IN READW/<07>
        .EVEN
IOBLK1: .BLKW   10              ;MACRO'S SCRATCH SPACE
FILOUT: .RAD50  /DK/
        .RAD50  /GIVEN DAT/
R1STO:  .WORD   0               ;REGISTER TEMPORARIES
R2STO:  .WORD   0
R3STO:  .WORD   0
R4STO:  .WORD   0
R5STO:  .WORD   0
;
;
HEAD1:  .WORD   0
INBUF1: .BLKW   400
HEAD2:  .WORD   0
INBUF2: .BLKW   400
FULFLG: .WORD   0
RECFLG: .WORD   0
CYBFLG: .WORD   0
CRFLG:  .WORD   0
        .END    START
```

TAPEIN Utility Flow Chart
(Chart 1)



TAPEIN Utility Flow Chart
(Chart 2)



TAPEIN Utility

Program Listing

```
        .TITLE  TAPEIN
        .MCALL  .ENTER,.FETCH,.WRITW,.CLOSE,.EXIT,.PRINT
        DRCSR=167770
        DROUTB=167772
        DRINBU=167774
        TKS=177560          ;TERMINAL KEYBOARD STATUS
        TKB=177562          ;TERMINAL KEYBOARD BUFFER
        TPS=177564          ;TERMINAL PRINT STATUS
        TPB=177566          ;TERMINAL PRINT BUFFER
        R1=%1               ;INPUT ADDRESS POINTER
        R2=%2               ;COUNTER
        R3=%3               ;COUNTER
        R4=%4               ;TEMP STORAGE
        R5=%5               ;NULL FLAG
;
;       FIRST FILL INPUT BUFFER
;
START:  BIC     #140,DRCSR      ;ENSURE INTERRUPTS DISABLED
        MOV     #INPUT,R1       ;POINT R1 TO INPUT BUFFER
        MOV     #1,R5           ;SET NULL FLAG ON
        BIC     #100,TKS        ;DISABLE TERMINAL INTERRUPT
        BIS     #1,DRCSR        ;OUT/IN=1
        BIC     #2,DRCSR        ;DATA/CHAN=0 (SET CHAN)
        MOV     #1,DROUTB       ;WRITE CHAN #1
        BIC     #1,DRCSR        ;OUT/IN=0 (INPUT)
        CLR     R2              ;SET CHAR COUNTER TO ZERO
        INC     R2              ;SET TO ONE
L1:     TSTB    TKS             ;KEY PRESSED ON TTY?
        BMI     DONEIN          ;IF YES THEN DONE WITH INPUT
        TST     DRCSR           ;WAIT FOR INPUT
        BPL     L1
        MOV     DRINBU,R4       ;READ INPUT
        BIC     #177600,R4      ;REMOVE HIGH 8 BITS AND PARITY
        TSTB    R4              ;NULL?
        BNE     L11             ;IF NOT THEN IT'S A CHAR
        TST     R5              ;NULL FLAG OFF?
        BEQ     DONEIN          ;IF YES THEN THIS IS A TRAILING NULL (END!
        BR      L1              ;IF NO THEN LEADING NULL--SKIP & CONTINUE
L11:    CLR     R5              ;CLR NULL FLAG (CHAR WAS READ)
        INC     R2              ;CHARACTER COUNT
        MOVB    R4,(R1)         ;READ CHAR INTO BUFFER
L2:     TSTB    TPS             ;WAIT FOR TERMINAL READY
        BPL     L2
        MOVB    (R1)+,TPB       ;ECHO TO TERMINAL
        BR      L1
DONEIN: MOV     DRINBU,R4       ;READ NEXT CHAR
        CMPB    R4,#377         ;END OF TAPE?
        BNE     DONEIN          ;IF NO THEN DO AGAIN
        MOVB    #12,(R1)+       ;INSERT LINE FEED
        INC     R2
        MOVB    #15,(R1)+       ;INSERT CARRIAGE RETURN
        INC     R2
        ROR     R2              ;R2 IS NOW WORD COUNT
;
;       NOW WRITE INPUT BUFFER TO DISK
;
        .ENTER  #IOBLK,#0,#FILNAM,#-1
        BCS     ERENT           ;ERROR IN ENTER
        MOV     #INPUT,R1       ;POINT R1 TO INPUT BUFFER
        INC     R1              ;SKIP OVER LEADING LF,CR
```

```
        INC     R1
        DEC     R2
        CLR     R3              ;SET BLOCK NUMBER TO ZERO
L3:     CMP     R2,#400         ;ENOUGH WORDS TO FILL BUFFER?
        BLT     PART            ;IF NO, THEN PART
        .WRITW  #IOBLK,#0,R1,#400,R3
        BCS     ERWRI           ;ERROR IN WRITW
        INC     R3              ;SET FOR NEXT BLOCK NUMBER
        ADD     #1000,R1        ;POINT TO NEXT BLOCK OF INPUT
        SUB     #400,R2         ;DECREASE WORD COUNT BY ONE BLOCK
        BR      L3
PART:   .WRITW  #IOBLK,#0,R1,R2,R3
        BCS     ERWRI           ;ERROR IN WRITE
        .CLOSE  #0
FINISH: BIS     #100,TKS        ;REENABLE TERMINAL INTERRUPT
        .EXIT
;
;       ERROR ROUTINES
;
ERENT:  .PRINT  #ERENT1
        BR      FINISH
ERWRI:  .PRINT  #ERWRI1
        BR      FINISH
ERENT1: .ASCIZ  /ERROR IN ENTER/
        .EVEN
ERWRI1: .ASCIZ  /ERROR IN WRITE/
        .EVEN
DK:     .RAD50  /DK/
FILNAM: .RAD50  /DK/
        .RAD50  /TAPE  DAT/
IOBLK:  .BLKW   10              ;SCRATCH SPACE
INPUT:  .WORD   0               ;INPUT BUFFER
        .END    START
```

APPENDIX C

DESIGN CYCLE FOR CURRENT (1992) TECHNOLOGY

APPENDIX C

DESIGN CYCLE FOR CURRENT (1992) TECHNOLOGY

Contents

DESIGN CYCLE FOR CURRENT (1992) TECHNOLOGY

1. Introduction

Since the time this project was begun (1982), the state of the art has advanced and the author's understanding of a formal design cycle has matured. This appendix describes (1) how recent advances in computer technology would affect this project and (2) a design process which would provide a disciplined, well-documented approach to requirements analysis, design, implementation, and testing.

2. Current Technology

2.1 Background. When this project was begun, an LSI-11 minicomputer had already been acquired for use in the laser spectroscopy laboratory. The Physics Department had decided to use this computer to perform data acquisition, data reduction, and experiment control. The required task was to design and implement an interface system for that computer. If this project were begun today, however, newer technology might lead the Physics Department to choose a different computer.

According to Kocher (Ref 10:246), "Advances in integrated-circuit electronics have revolutionized the possibilities for laboratory applications of small computers." He also notes that the IBM PC "has become a *de facto* standard . . . [and that] an abundance of inexpensive software is available for it." (Ref 10:246) Bok *et al* (Ref 11:219) describe IBM PC's as having "been accepted as industrial standards for the automation of experiments."

Various built-in cards and the standard RS-232C and HP-IB interfaces allow one to "build a powerful measurement and control unit." (Ref 11:219) Furthermore, computer software is now on the market which is directly applicable to a laboratory application (Ref 11:219).

If this thesis project were to be started today, the Physics Department might select an IBM PC instead of an LSI-11 for use in their laser spectroscopy laboratory. Furthermore, according to Petrini et al (Ref 12:161), "Many manufacturers now make available instruments already packaged with hardware and software that allow personal computers to control the machines, and to collect, store, analyze and display data without burdening the investigator with the computer details." Thus, technology advances in computers as well as in the laboratory instruments themselves would affect an interface design.

2.2 The State of the Art. An interface project, if started today, would need to consider the current state of the art. A sampling of the current literature uncovered several recent articles which would have potential application to an IBM-PC-based laser spectroscopy laboratory interface system.

For example, serial communication using RS-232 devices is described by Petrini et al (Ref 12). He addresses general laboratory applications for data I/O and equipment control. In addition, Hall (Ref 13) notes that IEEE-488 is a widely used, well known standard for computer

peripherals. He describes an interface system based on the IEEE-488 bus and provides two laboratory interfacing examples. Two other recent articles, highlighted below, would also have potential application to the project.

Bok et al (Ref 11) describe the automation of an ultraviolet-visible spectrometer and a single-proton counting apparatus. The hardware configuration employed used standard RS-232C and HP-IB interfaces plus MetraByte CTM-05 and PIO-12 cards to provide TTL levels for data I/O and device control. ASYST was the commercially-available software package selected.

Kocher (Ref 10) describes an instructional laboratory at the Oregon State University's Physics Department. A standard IBM serial/parallel I/O interface board was modified (see Ref 10 for details) and incorporated into an IBM PC-AT laboratory setup. The setup also included a 60-MHz general-purpose oscilloscope, digital multimeter, function generator, multi-output power supply, and prototyping breadboard. Microsoft QuickBASIC was selected as the programming language. The laboratory course covered the following topics: parallel I/O, serial I/O, digital-to-analog conversion, analog-to-digital conversion, closed-loop experiments, fast data sampling, and signal averaging.

2.3 Summary. If the laser spectroscopy laboratory were being established today, the AFIT Physics Department might select an IBM PC instead of an LSI-11 as one of the

major pieces of laboratory equipment. With today's technology, many of the remaining pieces of laboratory equipment could be selected which were manufactured to be compatible with the IBM PC. The equipment selection would, of course, affect the design of an interface system. In fact, a custom-built interface might not be needed at all.

If a custom-built interface was required, however, a sampling of the current literature indicates that several articles exist which could potentially relate to a laser spectroscopy application. The next section describes the process to analyze both the interface requirements of the actual laboratory equipment and the applicable literature as part of the overall formal design cycle.

3. Design Cycle

3.1 Background. Since this project was begun, the author's experience with the Air Force acquisition process has led to a maturing of his understanding of a formal design cycle. If this project were begun today, a more disciplined and better documented approach would be used as outlined below.

3.2 Design Cycle Details.

3.2.1 Requirements Analysis. The first step of the design cycle would be to document the user's top-level requirements. These requirements would be obtained either by interviewing Physics Department personnel or by reviewing any existing documentation relating to the anticipated use of the laser spectroscopy laboratory--or

both. The following information would need to be obtained: types of experiments anticipated; types of computers and laboratory equipment anticipated; specifics regarding interfaces, timing, purposes, data rates and volume, and experiment durations. The anticipated constraints on the project would also have to be obtained; particularly maximum cost and desired schedule.

Once obtained, these requirements would be organized into a system requirements document. A system requirements review would then be conducted to obtain concurrence from the user (and thesis advisor) of the validity of the requirements. The review could be either a formal meeting or simply a review of the requirements document by the individuals involved. In either case, written concurrence by the user and thesis advisor would be obtained.

The next step would be to analyze the system requirements to derive lower-level hardware and software requirements. The current state of the art would need to be examined to determine if commercially-available products could satisfy some or all of the requirements. If commercially-available products could not meet all the requirements or if they exceeded the stated cost constraints, then a literature search would be conducted to collect applicable information for a custom-designed system. A preliminary design would then be prepared and presented to the user in a design review. Design options might also be presented to the user for a system which

might not meet all his requirements, but which could be completed more quickly or for less cost. Again, written concurrence would be obtained.

3.2.2 Detailed Design and Implementation. Assuming that commercially-available products alone would not meet all the user's requirements, the next step would be to develop a detailed design. The preliminary design would be decomposed into hardware and software modules which could be individually tested. The requirements would be refined into a set of specifications for the system. Test procedures would be written to document how the system would be tested to ensure it meets its specifications. A detailed design review would be conducted to again obtain user concurrence.

Following detailed design, hardware modules would be breadboarded and individually tested. Tests would be conducted using the test procedures noted above and the results would be documented in test logs. Likewise, software modules would be written and individually tested.

Integration of the hardware and software modules would then follow with testing as noted above as modules are incrementally added. Finally, an overall system test would be performed to ensure all user requirements are met.

Once a working breadboard had been demonstrated, implementation of the final hardware configuration could begin. Assuming that a wire-wrapped implementation is acceptable, individual modules would be constructed and

tested as they were in the breadboarding phase. Test results would be compared to the breadboard tests to ensure defects were not introduced. Integration and testing of modules would proceed until a completed, functioning system successfully passed all tests.

Before the system could be turned over to the user, a user's manual would have to be written which documented the operation and maintenance of the system.

The final step would be to perform a comprehensive acceptance test to demonstrate to the user that the system met all his stated requirements.

3.3 Summary. The design cycle outlined above provides a structured approach to requirements analysis, design, implementation, and testing. The approach ensures that each step is well documented and provides for frequent feedback opportunities to ensure the project is on track. The focus is on strong interaction with the user from the beginning, when his requirements are first determined, to the end, when the system that meets those requirements is ultimately delivered.

4. Conclusion

If this project were begun today, the design approach and the design itself would both be different. Advancing technology has made personal computers and compatible equipment a routine part of the laboratory environment. Existing off-the-shelf hardware and software might be adequate to meet the needs of a laser spectroscopy

laboratory. If, however, a custom-designed interface system was still required, the author's current understanding of a formal design cycle would allow him to use a more structured, better documented approach for the project.

VITA

Douglas Letson Durand was born on 28 October 1958 at Wright-Patterson AFB, Ohio. He graduated from high school in Dunwoody, Georgia in 1977. He attended Georgia Institute of Technology with an ROTC scholarship and received the degree of Bachelor of Electrical Engineering in June 1981. Upon graduation, he received a commission in the USAF and immediately entered the School of Engineering, Air Force Institute of Technology.

After his period of residence at AFIT, he was assigned to Air Force Systems Command, Space Systems Division, Los Angeles AFB, California, from January 1983 to July 1989. At Space Systems Division, he began his systems acquisition career first as a project engineer for the NATO III-D communications satellite and later as a program manager for a highly-classified advanced space-related system.

He continued his systems acquisition career with an assignment to Headquarters Air Force Systems Command, Andrews AFB, Maryland, from July 1989 to the present. He is currently serving as Manager, Advanced Space Systems, where he focuses headquarters support functions to a broad array of classified space-related programs.

Permanent address: 4912 Olde Village Court
Dunwoody, Georgia 30338

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1992	MS Thesis

4. TITLE AND SUBTITLE
DESIGN OF A LABORATORY COMPUTER INTERFACE

5. FUNDING NUMBERS

6. AUTHOR(S)
Douglas L. Durand
Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology (AFIT-EN)

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GEO/ENG/92D-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

An interface for the LSI-11 computer was designed and implemented so that the computer supports data acquisition, data reduction, and equipment control. The design includes both hardware and software and addresses both parallel and serial input/output (I/O).

The serial interface's hardware is simply a Serial Line Unit card. This card plugs into the LSI-11 bus and provides the signals necessary to interface EIA RS-232 compatible devices. A software utility was developed to allow communication with the serial device and to allow exchange of data files. Routines were written to allow serial I/O through a FORTRAN program.

The interface's parallel hardware includes the general purpose laboratory interface system (GPLIS) architecture. In addition, hardware modules were designed to convert certain device's signal levels to TTL levels. Software utilities were developed to acquire and store parallel data and routines were written to allow parallel I/O through a FORTRAN program.

14. SUBJECT TERMS
Computer Interface, General Purpose Laboratory Interface System, GPLIS

15. NUMBER OF PAGES
105

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL