PRIORITY HANDLING HARDWARE AND SOFTWARE FOR THE HP 2100A COMPUTER

A Thesis

Presented to

the Faculty of the Department of Electrical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

---

by

Rashmikant B. Patel

Aug 1974

ACKNOWLEDGMENT

A great deal of credit for this thesis goes to my brothers, for their help and encouragement.

I would like to gratefully thank Dr. J. D. Bargainer for his guidance and enthusiasm.

A special thank you goes to Mr. Bill Price for his help and instruction and to Ms. Michele Maes for her alacrity and efficiency.

PRIORITY HANDLING HARDWARE AND SOFTWARE FOR THE HP 2100A COMPUTER

An Abstract of a Thesis

Presented to

the Faculty of the Department of Electrical Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

---

by

Rashmikant B. Patel

Aug 1974

ABSTRACT

Although it is general purpose and highly efficient data processor, the HP 2100A Computer lacks efficiency for fast computational process, if it is not equipped with the high speed input/output peripherals. It is designed for input/output flexibility with plug-in interfaces as the main feature. In order to have an efficient computational processing system, a high speed paper tape reader and a high speed paper tape punch were interfaced with the HP 2100A Computer. The high speed reader, which operates on photo sense principle, reads at a rate of 300 characters-per-second. The high speed punch punches data at a rate of 60 characters-per-seconds. A selecting feature was added to facilitate the full use of a high speed reader and a high speed punch, as common devices between the HP 2100A Computer and the SDS-92 Computer.

In a process control environment there would be several tasks to be performed, parameters to be input and messages to be printed to the operator. It would be desirable to have the priority for these tasks which could be controlled by the operator. This function was achieved by writing a priority handler.

A priority handler was written partly in FORTRAN and partly in assembly language. An optional feature was created for interrupting a task service subroutine by one of the higher priority service subroutines. A queue table was generated to keep track of all task subroutines asking for service. Also, a capacity for nesting all interrupts from different subroutines was developed. The indication for the errors were generated by

the error messages.

A priority table was developed for the priority of the different task service subroutines, which could be controlled by the operator by re-configuring it. The priority handler was written to accept the request for servicing task subroutines by setting bits of a word on a particular interface card. The programming problems, due to the existing HP software were also resolved.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

The HP 2100 Computer is a small, general purpose data processor. It is designed for input/output flexibility with plug-in interfaces as the main feature. This computer can accept 14 interrupts from device interfaces. An additional 31 interrupt channels can be interfaced by using an extender.

1.1 The HP 2100A Computer Interface

Interfacing a peripheral device includes both hardware and software development. Hardware interfacing requires designing a printed-circuit card which fits into the input/output slots in the computer, and the necessary cable connection to the device. Software interface involves re-configuring the Basic Control System. A driver which handles the particular device interrupt, may be required if it is not available.

1.2 I/O Priority Structure

The priority of the device is determined by the physical location of the slot in which device interface card is installed. The lower order channel (10B) has highest priority among the I/O devices. The address called "select code", is the number of the slot into which the interface is connected. Since many device interfaces will be requesting service at random times, it is necessary to grant service in an orderly sequence, starting from highest priority. I/O priority is established

by an enabling line running in series through all interface cards. Figure 1.1 shows the priority structure and the device connections to the computer. The device can interrupt the com-

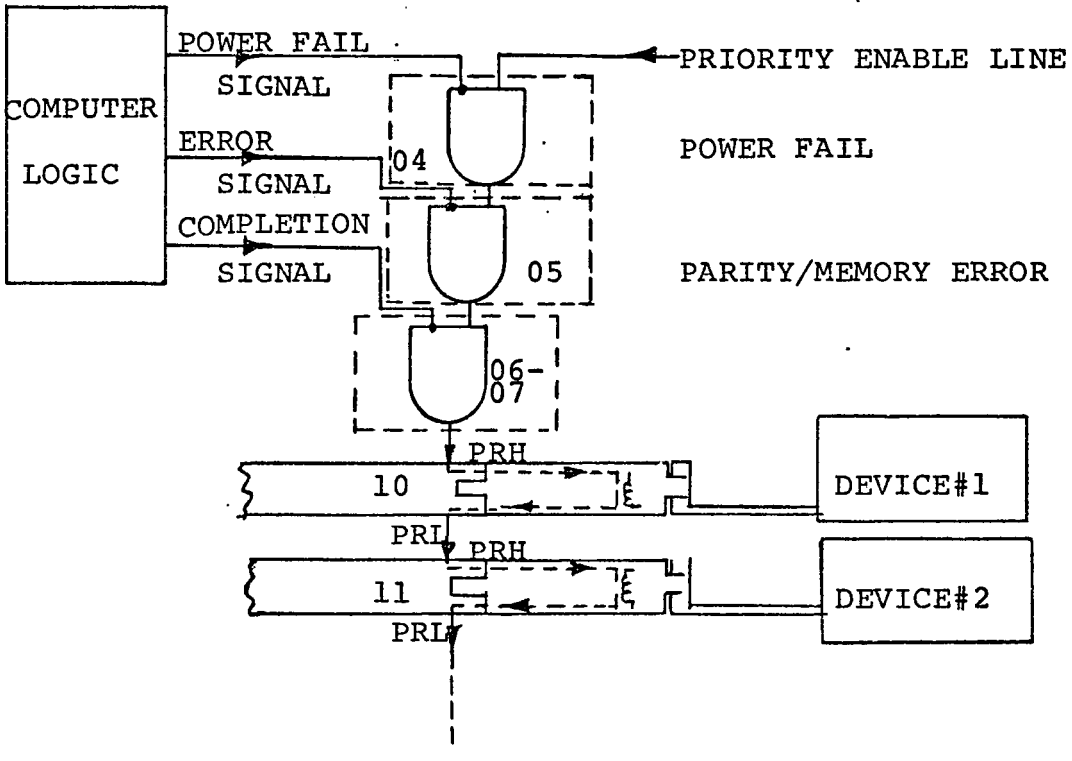


Figure 1.1 The I/O Priority Structure

puter for the service by breaking the priority enable line (represented by relay). The higher priority can break the line, such as Power Fail, at any time and temporarily disable the device that was being served. The interrupted transfer will continue when the higher priority device finishes. Thus due to the series linkage of the priority line, the lower priority device will be disabled when any of the higher priority needs service. The priority line should not be broken or there should

not be any open slots otherwise the lower priority devices will be disabled. The intermediate empty slots must be replaced by a jumper card.

1.3 Interface Interrupt Mechanism

There are three signals which determine whether an interrupt can occur or not. Figure 1.2 shows the block diagram of the essential signals. The signals PRH, Flag and Control are logically AND-ed as shown, to give an interrupt request signal (IRQ). IRQ is the signal which requests interrupt to the computer. PRH is the signal corresponding to the priority enabling line which is the input to the interface card from a device of higher priority. PRL is the signal to the next lower priority device. The Flag indicates that the device needs service and the Control signal enables the device. Thus when all the signals (PRH, Flag, Control) are logically true, then IRQ causes interrupt to the computer program. After the interrupt is granted, the interrupt flip-flop is cleared by the interrupt Acknowledge (IAK) signal. Flag flip-flop is set by the instruction "STF", or by the device itself, and can be cleared by the instruction "CLF". The device is enabled by setting the control with the STC instruction, and disabled with the CLC instruction. Upon interrupt to the computer, a Jump Subroutine (Indirect) to the word holding the absolute address for the entry point of the Interrupt Processor is executed. The interrupt location is reserved in the low core area of the memory. The Interrupt Processor processes the interrupt request and resumes the in-

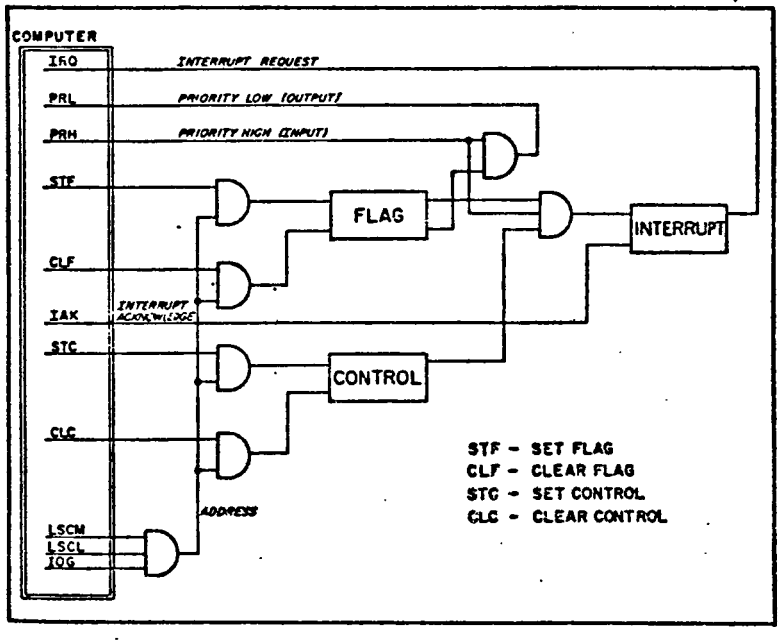


Figure 1.2 Flag and Control

terrupted program at the completion of interrupt procession.

1.4 Input/Output Transfer

INPUT. The basic block diagram for the input transfer of data is shown in Figure 1.3. The particular interface card is enabled when its address (LSCM/LSCL) is selected by the computer. The Flag signal, from the device, is used as a strobe to put data into the interface register. The device Flag in addition to strobing data, sets the Flag flip-flop of the interface card (Figure 1.5) to interrupt the computer. When the computer interrupts, it transfers data from the interface register to the A or B register of the computer through the IOBI lines. The instructions for loading A or B register from the interface register are LIA, LIB, or MIA, and MIB. These instructions essentially provide IOI and IOG signals which are used actually to transfer data as shown in Figure 1.3. After one operation is completed, the computer has to be interrupted in a same manner for the next single operation.

OUTPUT. The Figure 1.4 shows the output of data from computer to the interface card. The card is enabled by STC as was done for input. After the card is selected, the output instructions (OTA/OTB) provide the IOO and IOG signals necessary to transfer the data from the A and B registers (IOBO lines) into the interface register. The device will accept the data when a signal from the control flip-flop (Figure 1.5) indicates that the data is ready on an interface card.

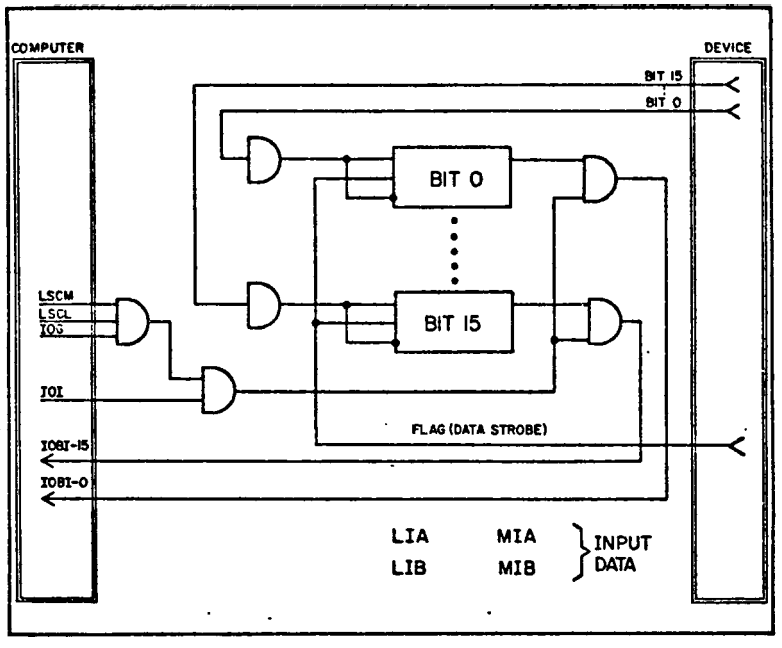


Figure 1.3 Input Transfers

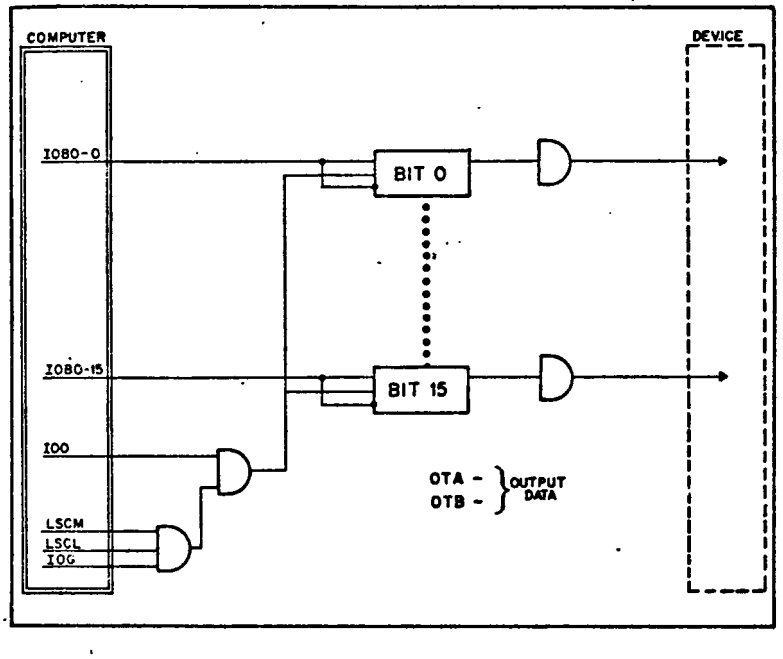


Figure 1.4 Output Transfers

1.5 I/O Device Commands

The device command for input/output operation is as shown in Figure 1.5. On output operation, STC instruction from the computer sets the control flip-flop which provides an Encode signal to indicate that data is ready on the output lines. The device Flag is cleared after the data is transferred to the device and resultant interrupt indicate that a single output operation is completed. On input operation the Encode signal indicates to the device that the interface card is ready to accept data. The device Flag strobes data into the interface register, and interrupts the computer to indicate that the data is ready in the interface register. This data is transferred into the memory by driver subroutine. The device Flag also clears the control flip-flop.

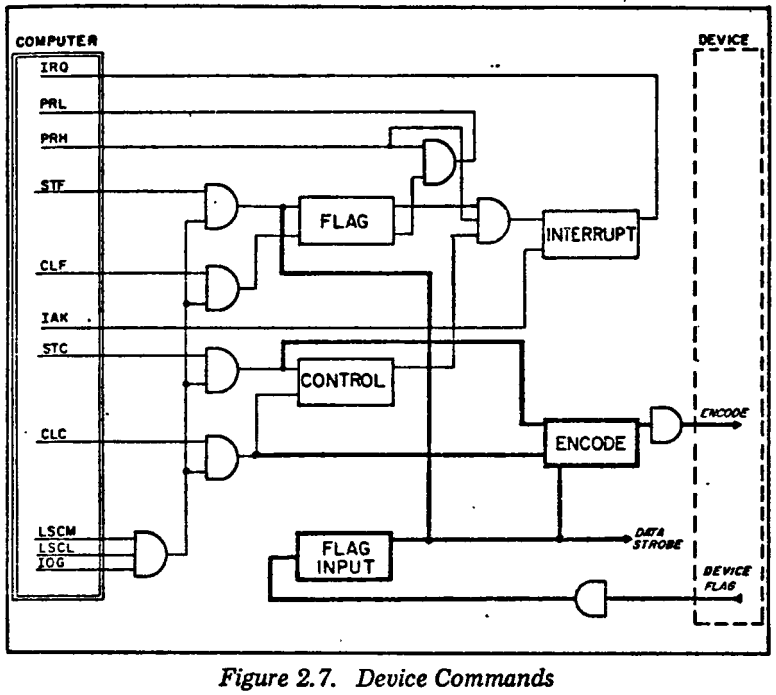


Figure 2.7. Device Commands

Figure 1.5 Device Commands

CHAPTER II

HIGH SPEED PAPER TAPE READER

The high speed paper tape reader was interfaced with the SDS 92 computer. The TTY reader reads the paper tape at the speed of 10-characters-per-second, while the high speed reader reads at a rate of 300 characters-per-second. So it was decided to interface the high speed reader with both the HP 2100 and to the SDS 92 computer, with a switch to select the computer. Thus the high speed reader is a common device shared by the two computers. The detail of the interface is discussed in the following sections.

2.1 REMEX 9131 Perforated Tape Reader

The REMEX Perforated Tape Reader consists of the following components. The tape transport mechanism consists of a pinch roller that pulls the perforated tape through the reader at a constant velocity, and a brake system used to halt the tape movement through the reader. The tape passes over a photocell block which senses the tape perforations. The output from the photocell is amplified by the HK61 Photo Sense Amplifier (in the SDS chassis) then routed to the reader output receptacle and then transmitted to the SDS coupler by means of a cable.

The drive system operating in the forward direction only, consists of a constant velocity drive roller shown in Figure 2.1, and a dc electromagnetically operated pinch roller (Jam Roller.) The pinch roller is operated by the application

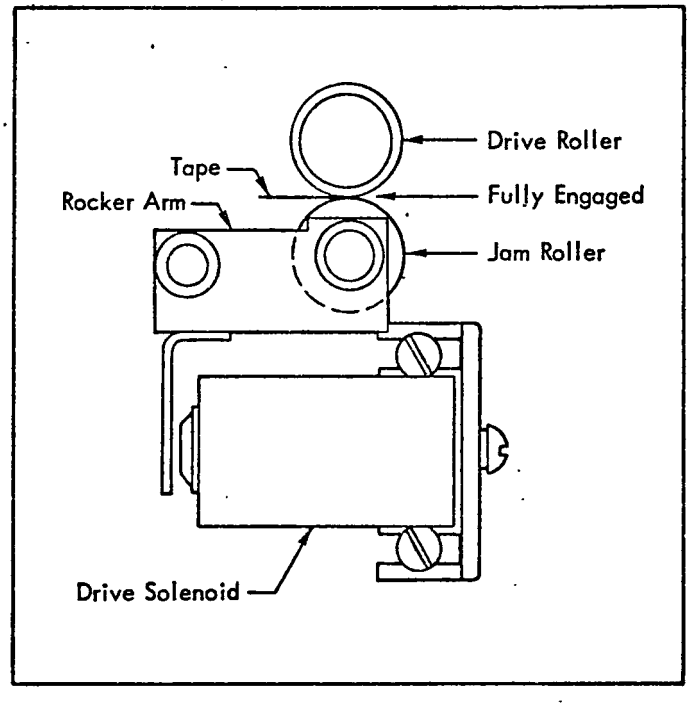


Figure 2-1 Tape Transport Drive System

of a +50 volt, 600ma dc drive signal to the drive solenoid.* When activated, it forces the pinch roller, and hence the tape, against the constant velocity drive roller. This causes the tape to be pulled in a forward direction. This dc signal can be applied by the computer or by the spring-loaded FEED switch on the reader front panel of the SDS-92 computer. Removal of this dc signal de-energizes the drive solenoid, releasing the rocker arm and pinch roller, and energizes the brake solenoid causing the tape movement to halt.

2.2 9131 Perforated Tape Reader Coupler System

A simplified functional block diagram of the coupler system is shown in Figure 2.2. All the necessary power supply for the coupler, and the reader is derived as shown in the diagram from the SDS computer.

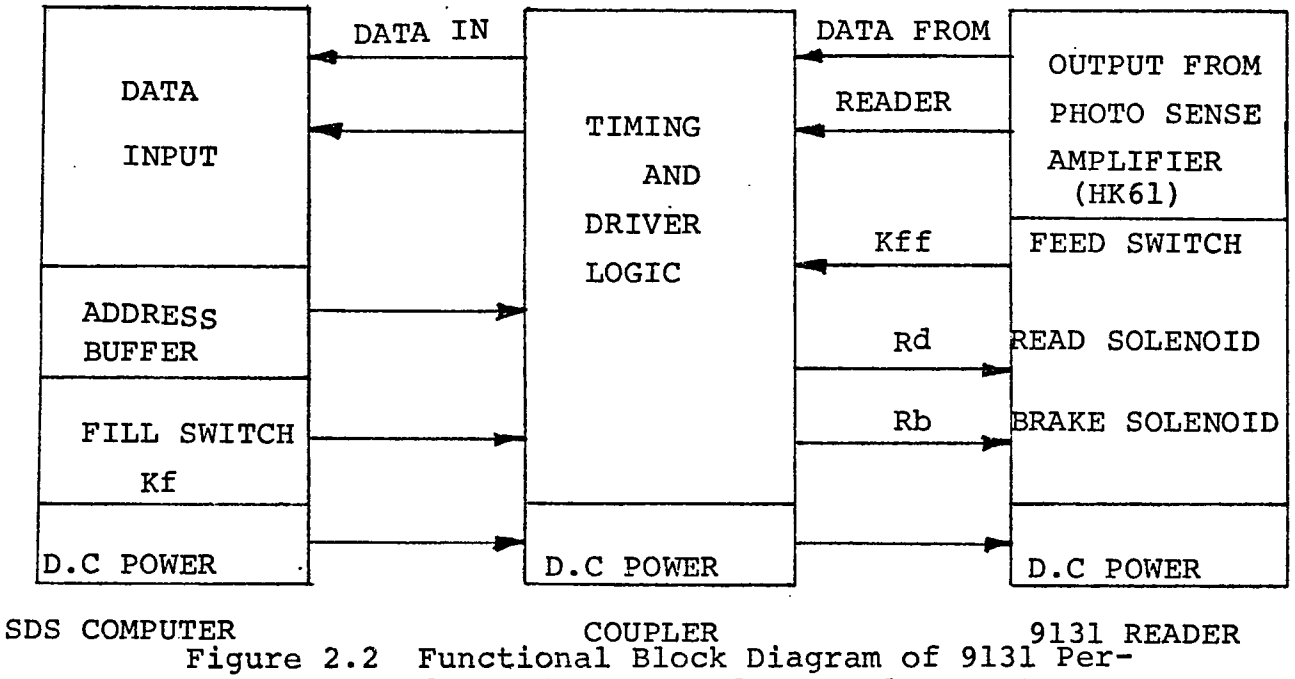


Figure 2.2 Functional Block Diagram of 9131 Perforated Tape Reader Coupler System

*See Reference 1 (Figure 5.9) for Schematic Diagram.

Data to the SDS-92 computer is transferred in parallel from the 9131 Reader through the coupler. Channels 1 through 8 is the data buss from the Photo Sense Amplifier. Whenever there is a hole on the paper tape, the corresponding channel data is made high and converted to the SDS positive logic level by the Photo Sense Amplifier. (+8 volts for logic 'True', representing 1, and 0 volt for logic 'False' representing 0). Only seven channels are used for the SDS computer, but provision for the eighth channel is provided, and its output is available at the output terminal of Photo Sense Amplifier.

For the SDS computer there is no ready status test for the reader, it is always ready for operation. When addressed by the computer W Buffer, the output of the address decoder enables the reader to start. The reader starts sending data within approximately one character time (1/300 of second). Three methods are available for enabling the reader in the SDS computer system:

1. Computer to reader control
2. Manually
3. By computer FILL switch

when the reader is addressed during a program execution, output of address decoder enables the reader. Manual enabling of the reader is accomplished by depressing the FEED switch (signal K_{ff}) on the reader front panel. The FILL switch (signal K_f) on the computer control panel may also be used to enable the reader. When K_f is logically true it enables the reader. The

purpose of FILL switch is to load the first few instructions of a program into the computer and initiate I/0 operation. The effect of K_f signal is to enable Re, which in turn enables the reader. When reader is addressed, the Re signal, to enable reader, goes true.* In its true state, Re causes the coupler relay driver circuit controlling the reader pinch-roller solenoid to function (i.e. switch its output to ground). As the low end of the pinch-roller solenoid is logically connected to the relay driver output, this grounds the low end of the pinch-roller solenoid. When this occurs, the pinch-roller solenoid (Rd) is energized, the brake solenoid (R_b) is de-energized, the reader is enabled and begins to transport and read perforated tape. Data read from the tape is transferred to the SDS computer via the coupler. The reader may be energized in a same way if manual FEED switch is pressed down (K_{ff} = 1). Thus simplified logic system for two signals R_b and Rd is as shown in Figure 2.3.

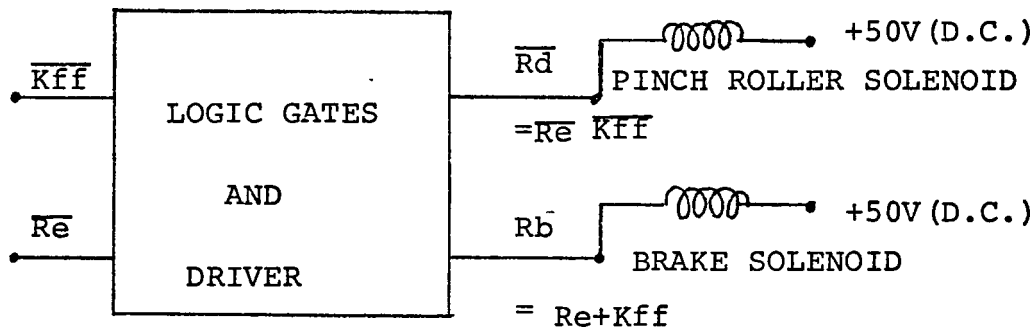


Figure 2.3 Coupler System

Normally K_{ff} and Re are low (Logic 0) so the brake solenoid is engaged. When any one of the two signal goes high (Logic 1)

*See Reference 1 (p. 3-11) for Logical Signal Path for Rb,Rd and Re.

the drive solenoid is engaged, which causes the paper tape to move.

2.3 Design Requirement for the Interface Card

After studying the reader and coupler system, it was decided to use the SDS driver, and SDS power supply for interfacing the reader. The design requirements for the HP 2100 interface card for the reader were as follows:

1. The reader should be compatible with all the HP 2100 hardware and software.
2. The reader should be used as a common device for both the computers.
3. The reader should be enabled, when FEED switch is pressed down, for manual feed operation by any of the two computers.

The first requirement can be satisfied by designing the interface card for the HP 2100 to meet requirements discussed in Section 1.4 for the input transfer of the data. The reader should be able to enable, i.e. should be able to read one character whenever a STC XX, instruction (XX = select code) is given. There should be a buffer register on the interface card, and the device flag has to be generated from the device in order to store data into the interface register and to interrupt the computer.

The second requirement can be met by putting a selectable switch for the use of the reader as a common device for both the computers. The reader can be enabled manually by

keeping the logical connection for the signal K_{ff} in the same position as it is.

2.4 Design of the HP 2100A Interface Card for the Reader

The design requirements for interfacing the reader are as mentioned in the Section 2.3. The reader is enabled manually by FEED switch on the SDS computer reader panel. In order to enable the reader manually, when it is used by HP computer, the signal (K_{ff}) coming from the reader to the coupler driver is used as it was for the SDS computer. This is shown in the Figure 2.4. Thus signal K_{ff} in its true state enables the reader, when FEED switch is pressed down. The source of the Re signal is either the SDS or the HP machine and is determined by the switch s.

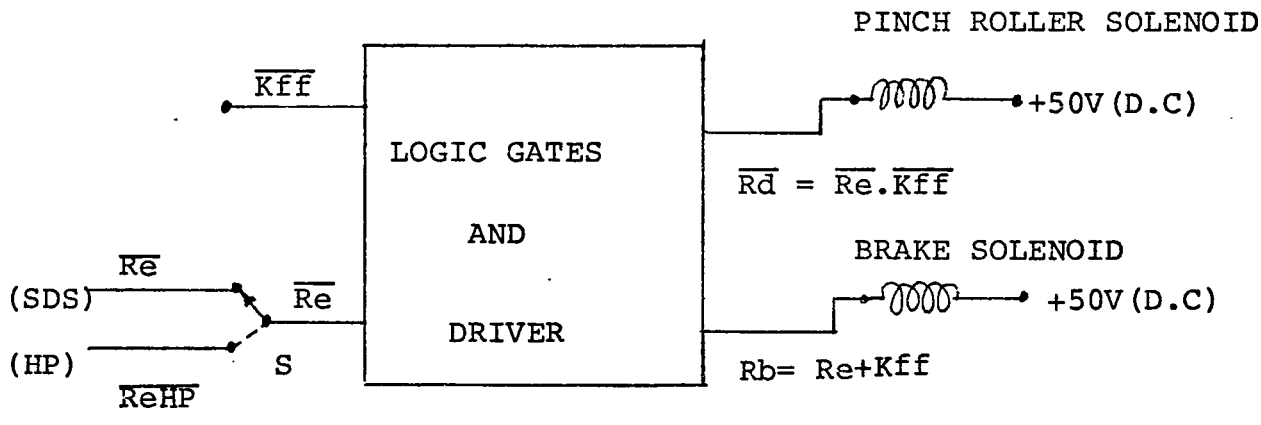


Figure 2.4 Modified Coupler System

When the switch s is in the SDS position, the reader is enabled by SDS computer. When it is in the HP position it is enabled by the HP computer. The signal $\overline{ReHP}$ (from HP computer) goes to ground whenever a SIC XX, instruction is executed to enable the reader.

The design of the interface card starts with the requirement of flag and interrupt logic. The Standard HP Interface Breadboard contains TTL Flag and Interrupt Logic. This board was available and was used for the Flag and Interrupt logic. The detail logic diagram of the Flag and Interrupt Logic is presented in Figure 2.5.*

The mechanism for the input transfer of data is as explained in Section 1.4. The output from the Photo Sense Amplifier is at the SDS logic level (+8 volt for logic 1). Since the interface card is design for TTL logic level, the SDS logic level is converted to the TTL by means of 3.0 volt zenner diode in series with the signal. The data lines are tapped off from the output of the Photo Sense Amplifier, (which comes to the chassis 45 s of the SDS computer), and bought to the interface card (B_0-B_7) through the chassis 31 s of the SDS computer. Figure 2.6 shows the detail logic diagram of the interface design. The signal from the sprocket (P_{sp}) is used to generate the device flag by triggering a monostable multivibrator on the trailing edge of the signal. External capacitor and resistor of the monostable multivibrator (D5) are adjusted so that the output signal pulse width is 200 ns long. This signal is used to strobe the input data, and the data are latched into the latch 7475 (C2,C3) which is used as an interface storage register. In addition to strobing the data, the invert of the device flag is used to set the flag buffer flip-flop (at TP1) to interrupt

*See Reference 3 (Section 2.21) for the Detail Explanation.

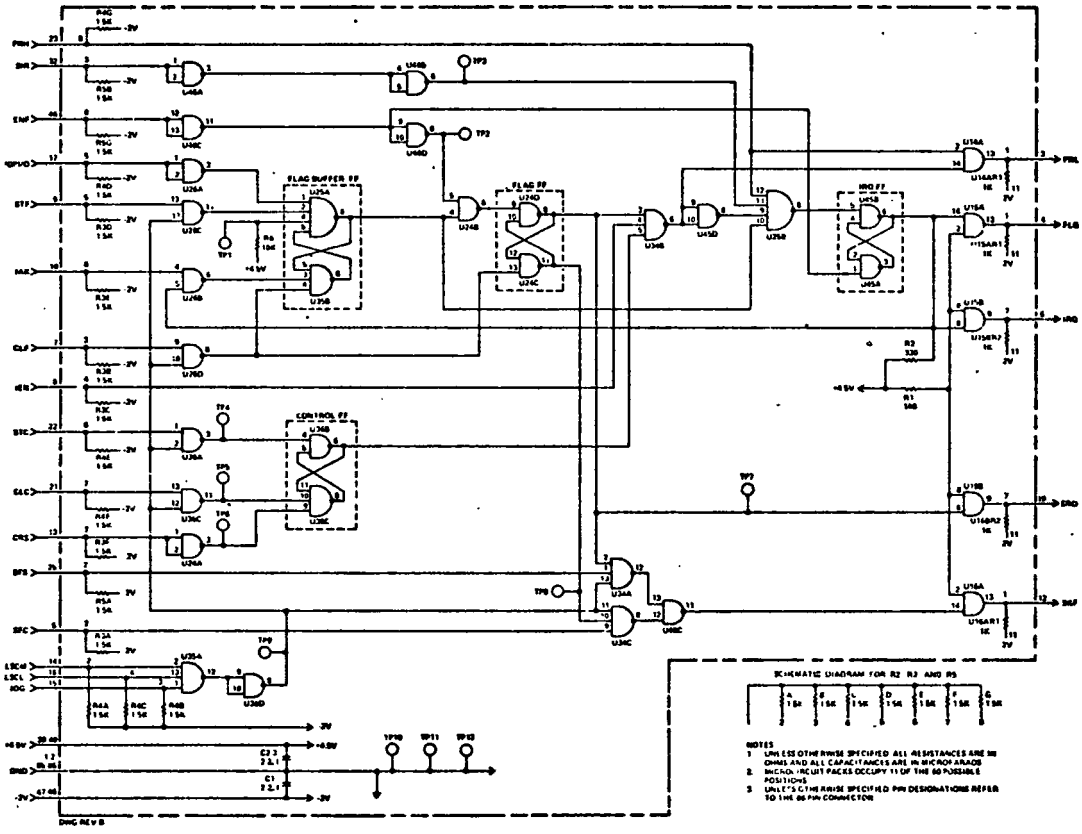


Figure 2.5 Breadboard Flag and Interrupt Logic

the computer. Command flip-flop is used to generate the signal $\overline{\text{ReHP}}$, which enables the reader as explained in Section 2.2. The STC XX instruction sets the control flip-flop in the flag interrupt logic circuit, and the output of the control flip-flop (TP4 as in Figure 2.5) is used as an input to the command flip-flop to set it. A logic level of +8 volt for true state is necessary for the SDS computer. Hence the output of the command flip-flop is used as an input to the 7403, and the output of the 7403 is tied to +8 volt through the 1.3K ohms resistor, which converts TTL logic level to SDS logic level for the signal $\overline{\text{ReHP}}$. This command flip-flop should be cleared by the Device Flag, CLC XX instruction and the CRS signal. When it is cleared, the reader is disabled (drive solenoid de-energized, and the brake solenoid energized). The CRS signal occurs at power turn-on, when PRESET button on the HP computer panel is pressed, or a CLC 00 instruction is executed. Since the NAND gate R-S flip-flop is used as a command flip-flop, the actual inputs ($\overline{\text{FLAG}}$, TP5, TP6) for resetting this flip-flop are the compliment of the above three signals, (TP5 and TP6 are from Flag Interrupt Logic) which are as shown in Figure 2.6.

The Device Flag notifies the flag and interrupt logic (by setting the flag buffer flip-flop at TP1) that data is now available to the computer. As explained in Section 1.4, data should be transferred to the A or B registers (via IOBI lines) with an input instruction (which provides IOI and IOG signals). and the proper address. Any signal to the TTL logic on the interface card requires a resistor pull to -2V.* The IOI signal

*Refer to Appendix C for Legal Receiver.

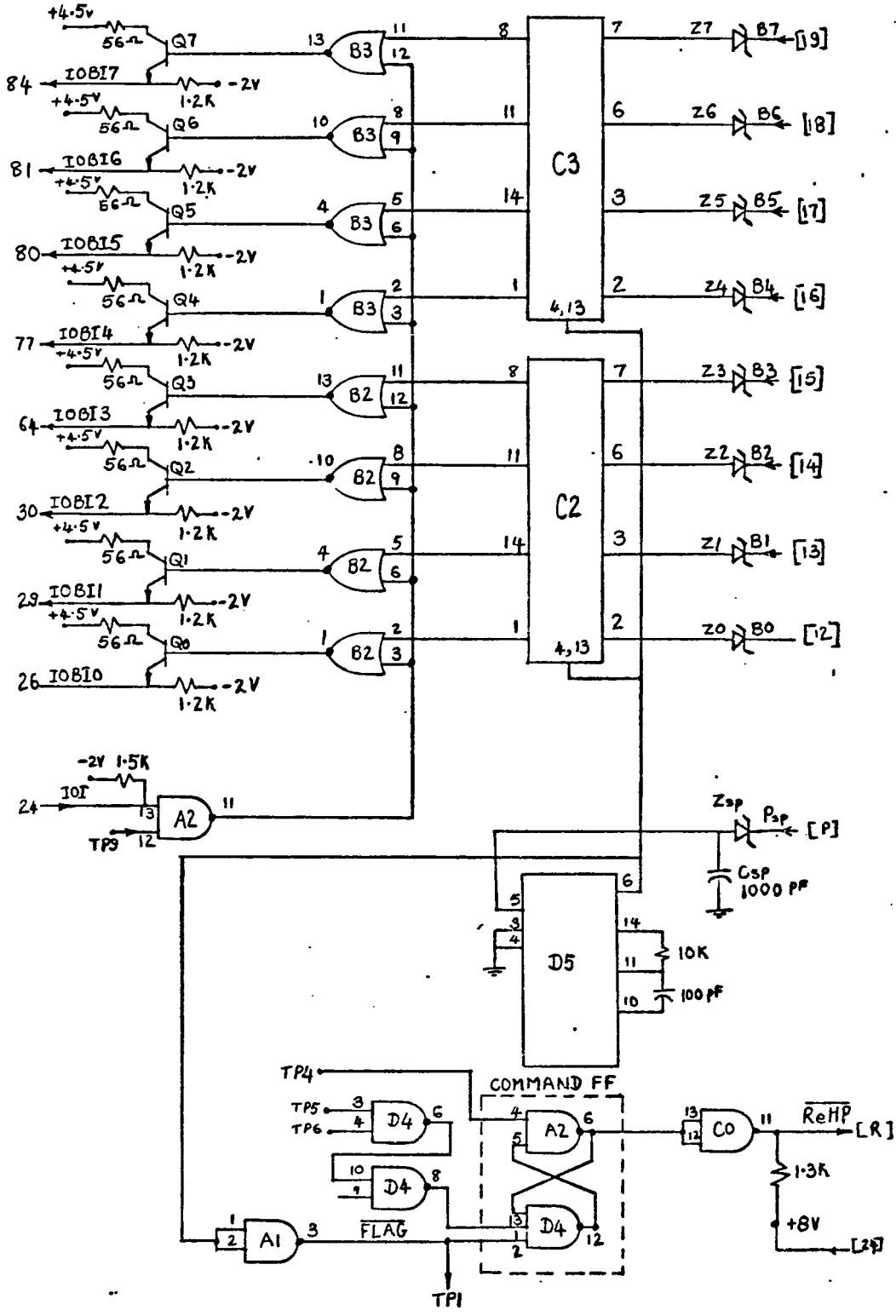


Figure 2.6 The Detail Logical Diagram of the Reader Interface

Note : (1) The Pin Assignment Is Given In Reference 3(Table 2.42)
(2) [YY] The Pin Number YY, Connected to the Device.

(with pull down resistor to -2V) and TP9 (which is output of the AND gate with LSCM, LSCL and IOG as inputs in Figure 2.5) are NAND'ed together. The output of the NAND gate and the inverted data are NOR'ed together, and this output signal is used as input to the emitter followers (Q_0-Q_7) with a emitter resistor tied to -2V. The outputs from the emitter of the 2N3643 are the IOBI input data lines going into the computer. Signals going to the backplane (computer) require a CTML 9956 driver, with a 1.5K resistor pull to -2V. The above discrete circuit is used in place of CTML 9956 driver, and the resultant circuit is as shown in Figure 2.6. To bypass noise from the signals, appropriate capacitors are used as shown in the Figure 2.6. The layout of the Interface card and the list of components used and the necessary change in connection on the SDS chassis is given in the Appendix D. After the Interface card was built, an interrupt test was executed with a Interrupt Test Program and various signals were checked with a small program loop.

The software requirement of the interface, which is an input driver (to handle input from the reader) was supplied by Hewlett-Packard and is used for the system input from the High Speed Reader. The Basic Control System was reconfigured to include the high speed reader driver using the special purpose program (Prepare Control System which is a part of system software package). Thus interface design was completed for interfacing the High Speed Reader to the HP 2100 computer.

CHAPTER III

HIGH SPEED PAPER TAPE PUNCH

The high speed paper tape punch, which was a perephreal of the SDS 92 computer, punches a tape at a speed of 60 characters-per-second. Like high speed reader, it was decided to interface the high speed punch with the HP 2100 computer. Thus the punch is used as a common device shared by both the computers and can be connected to any one of them at a time by means of a switch. The following sections discuss the detail about the interface.

3.1 Tape Perforator

The Tape Perforator is an electrically operated medium speed unit capable of perforating paper of varying widths from five to eight channels, at a rate of 60 characters per second. The unit is asynchronous, and can be operated at any repetition rate below the maximum, since each character is initiated by a separate, independent pulse. The perforator consists essentially of a perforator mechanism, a capstan drive mechanism and a 1/12 HP motor. The motor drives both the perforator which punches holes in the tape when the coils inside it are energized, and the capstan drive mechanism which moves the tape when energized. Both the mechanisms require dc pulses of 4.5 ±0.5 milliseconds duration, at 48 volts. Thus one pulse must be supplied for each movement of the tape and one pulse is required for each character to be punched. Punched pulses for all channels and the pulse for the advancement must be derived

from a common pulse, and the interval for the drive pulse should not be less than 16.7 milliseconds. (Leading edge to leading edge) These 4.5 ms pulses at 48 volts are obtained by tying one end of the solenoids, which energize the movement for punching and advancing, to the +48 volt and another end to the output of the driver. When a 4.5 ms pulse is applied as an input to the driver, the output of the driver goes to ground which in turn grounds the another end of the solenoid and energizes it. The output remains open when the input to the driver is at logic 0. The necessary power supply for operating the performator is obtained from the SDS computer power supply.

3.2 Tape Perforator and Coupler System

Figure 3.1 represents the simplified functional block diagram for the Tape Perforator and the Coupler System. The DC power supply is obtained from the SDS computer, and the inputs, R_1-R_6, are the bits of the character to be punched from the SDS computer buffer to the coupler system. R_p is the input for

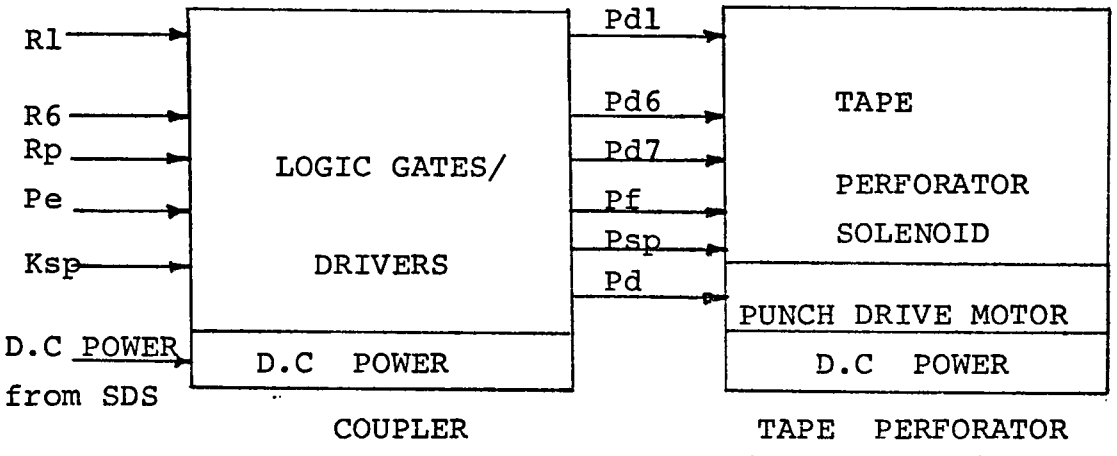


Figure 3.1 Functional Block Diagram for the Coupler System and Perforator

*See Reference 2, for the detail explanation.

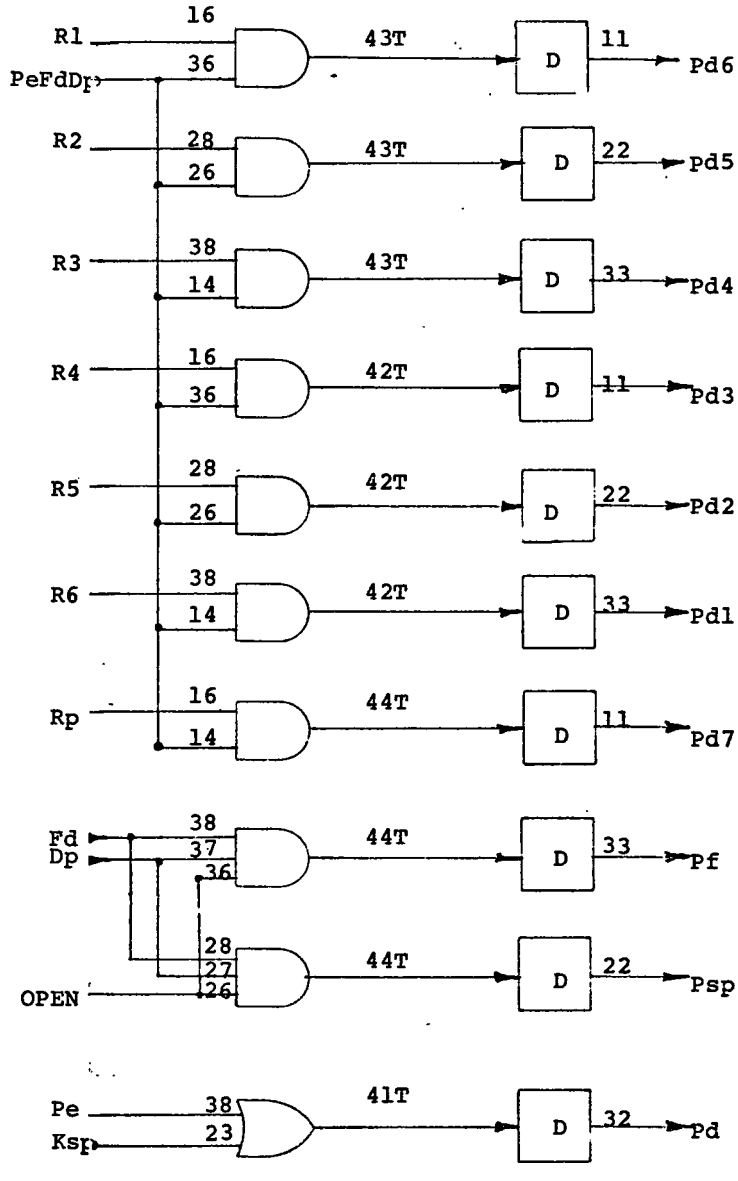
the parity bit, and the signal Pe goes true when the punch is addressed by the SDS computer which starts the motor in the perforator. The signal from the FEED button on the SDS front panel is K_{sp}, and is used for the same purpose as Pe. The output of the driver, Pd1-Pd7, are tied to one end of the punch solenoids, and the driver outputs P_f and P_{sp} are tied to the feed solenoid and sprocket solenoid for feeding the tape and punching the sprocket hole respectively. The output of the driver, Pd, enables the Punch Drive Motor by closing the relay in the Perforator which, in turn, closes the ac circuit to the punch motor. The following paragraph explains the manual feed operation of the tape.

When the feed button on the punch is manually depressed, the signal, K_{sp}, goes true. The relay driver Pd closes a relay in the punch assembly which, in turn closes the ac circuit to the punch motor ($Pd = K_{sp}+\ldots$) Figure 3.2 represents the simplified logic diagram for the punch coupler.* When K_{sp} goes true, the signal Fd which remains at logic 1 after some delay is generated. The signal Dp which is a series of pulses of width 4.5 msec and of the duration 16.5 msec is also generated after some delay.** The delay is provided in order to allow the motor to get up to the speed. The feed solenoid driver and the sprocket hole solenoid driver are now actuated and tape leader is generated as long as the button remains depressed.

---

* See Reference 5, for the detail logic layout for the punch.

** See Reference 4, for the detail explanation for the generation of the signals Fd and Dp.



NOTE: D- Solenoid Driver.
See Appendix B for Logical Symbol.

Figure 3.2 Simplified Logic Diagram for the Punch Coupler

$P_f = P_d D_p$ (feed solenoid driver)

$P_{sp} = F_d D_p$ (sprocket solenoid driver)

A toggle switch (on perforator front panel) can be set in either the 'auto' or 'run' position. If it is placed in the 'auto' position, the punch motir is turned on only when the SDS computer has addressed the punch or the feed button has been manually depressed. In the 'auto' position, there is an automatic delay each time the punch is addressed, in order to allow the motor to get up to speed. In run position the motor operates continuously.

3.3 Design Requirement for the Punch Interface

The SDS coupler can handle only seven data channels and a sprocket hole channel, but the perforator can handle eight data channels. After studying the coupler system and the perforator, the design requirements for interfacing the high speed punch were determined to be as follows:

1. A new driver should be designed in order to handle the capacity for eight channels since the HP 2100 computer punches ASCII characters. The seven drivers already built in the SDS coupler could be used for the another seven channels.
2. The punch should be used as a common device between the two computers.
3. The manual feed operation of the paper tape by pressing the FEED switch could be done when the punch is used by any of the two computers.

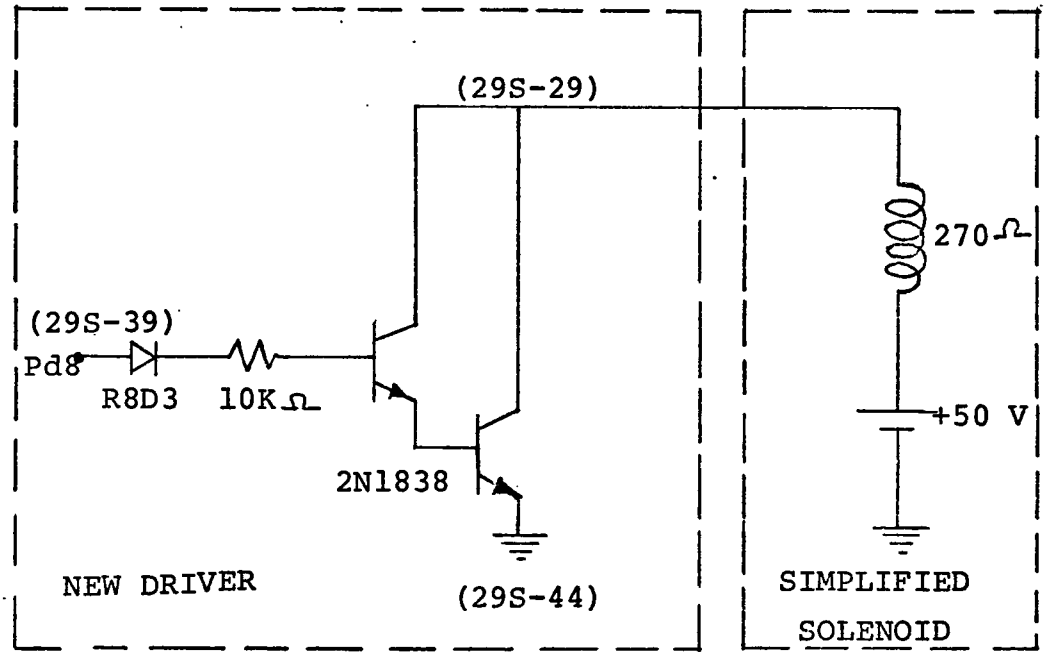
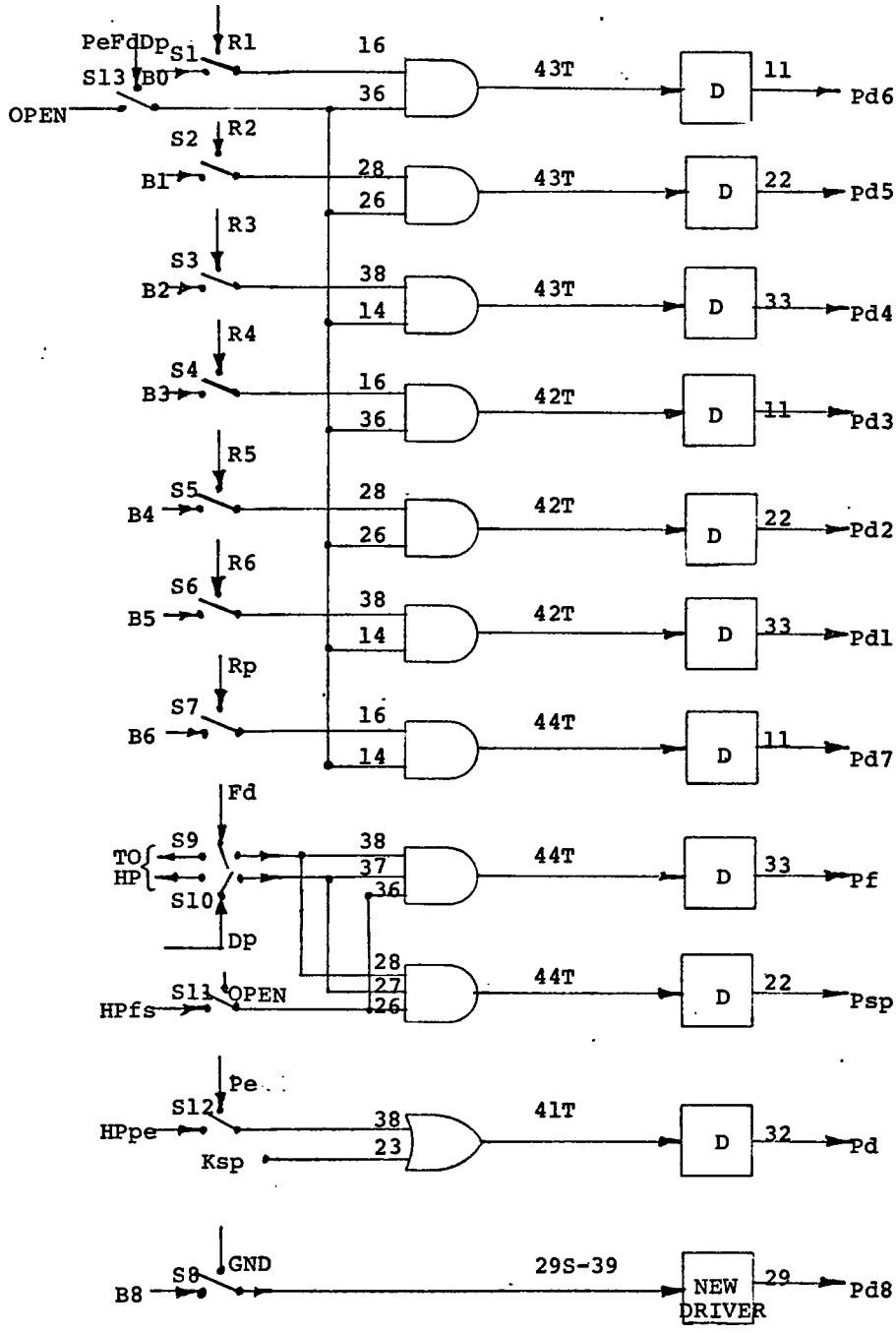


Figure 3.3 Darlington Pair for New Driver

4. The punch should be compatible with all the HP software and hardware requirements.

3.4 Design of the HP 2100A Interface Card for the Punch

The first requirement of having a new driver (for eighth channel) can be satisfied by designing a Darlington pair circuit with a R8D3 diode in series with a base resistance of 10K and 2N1838 transistors to meet the solenoid current of 0.3 amp, when the output of the driver connected to the one end of the solenoid is connected to the ground. The circuit diagram is as shown in Figure 3.3. (This is mounted on SDS card 29S). The requirement to operate the punch as a common device is met by putting a rotary switch which connects all the connections in the coupler back to their original position as they were before the interface when it is in the SDS position. The switch, when put to the HP position, connects the necessary signals to the HP 2100A. The simplified logic diagram, Figure 3.2, is modified to show the switch connections and the resultant new diagram is as shown in Figure 3.4. The switch when put to the SDS position connects the vertical connections, (Figure 3.4) and when in the HP position connects the horizontal connections. The output bits B0-B7 of the character to be punched from the HP computer replaces the data driver inputs (R1-R7 from SDS) when switch is in the HP position, with B7 as an input (parity check) to the new driver. When FEED switch on the SDS front panel is pressed, the series of pulses of width 4.5 ms are generated as mentioned in Section 3.2. In order to operate the punch manually when switch is in the HP positions, signals Fd



NOTE: 1) S1-S13 Are Switches on Rotary Switch.
The Signals from HP Interface Are Shown In Figure 3.6
2) D- Driver In SDS Coupler.

Figure 3.4 Simplified Logic Diagram of Drivers with Switch Connection.

and Dp are brought to the interface card and ORed with the 4.5 ms pulse from the interface card. By keeping K_{sp} in the same way the motor can be enabled manually, with the switch in either of the two positions. The actual wiring connection for the switch is given in the Appendix E.

Regarding compatibility of the interface with the HP computer, the interface card should be designed as explained in Section 1.4 for the output transfer. The data from A or B register is transferred by the output (OTA-OTB) instruction, which generates IOO signal. The output bits (IOB00-IOB07) are latched into the interface register (SN7475) by the enabling signal, which is the output of the AND gate with the IOO and TP9 (TP9 is as shown in Figure 2.5 of flag and interrupt logic) as inputs. The flag and interrupt logic was built as shown in Figure 2.5 for this interface card. As discussed in section 3.2 the 200 ms delay should be provided whenever the punch is first enabled and afterwards, to punch the data a 4.5 ms pulse with a period of 16.5 ms is required for each character to be punched and for each movement of the tape. The NAND gated RS flip-flop is used as a command-flip-flop. The schematic diagram of the design is as shown in the Figure 3.5. The $\overline{STC}$ (TP4) sets this flip-flop with the $\overline{CRS}$ (TP6), $\overline{CLC}$ (TP5) and Device $\overline{FLAG}$ as the reset inputs. The timing diagram for the different pulses is given in Figure 3.6. When command flip-flop is first set by the instruction STC XX, a pulse from M550 (MON) and another pulse from M200 are generated. The WAIT flip-flop (SN7474) was reset before the pulse from M550. The signal HPPe as shown in Figure

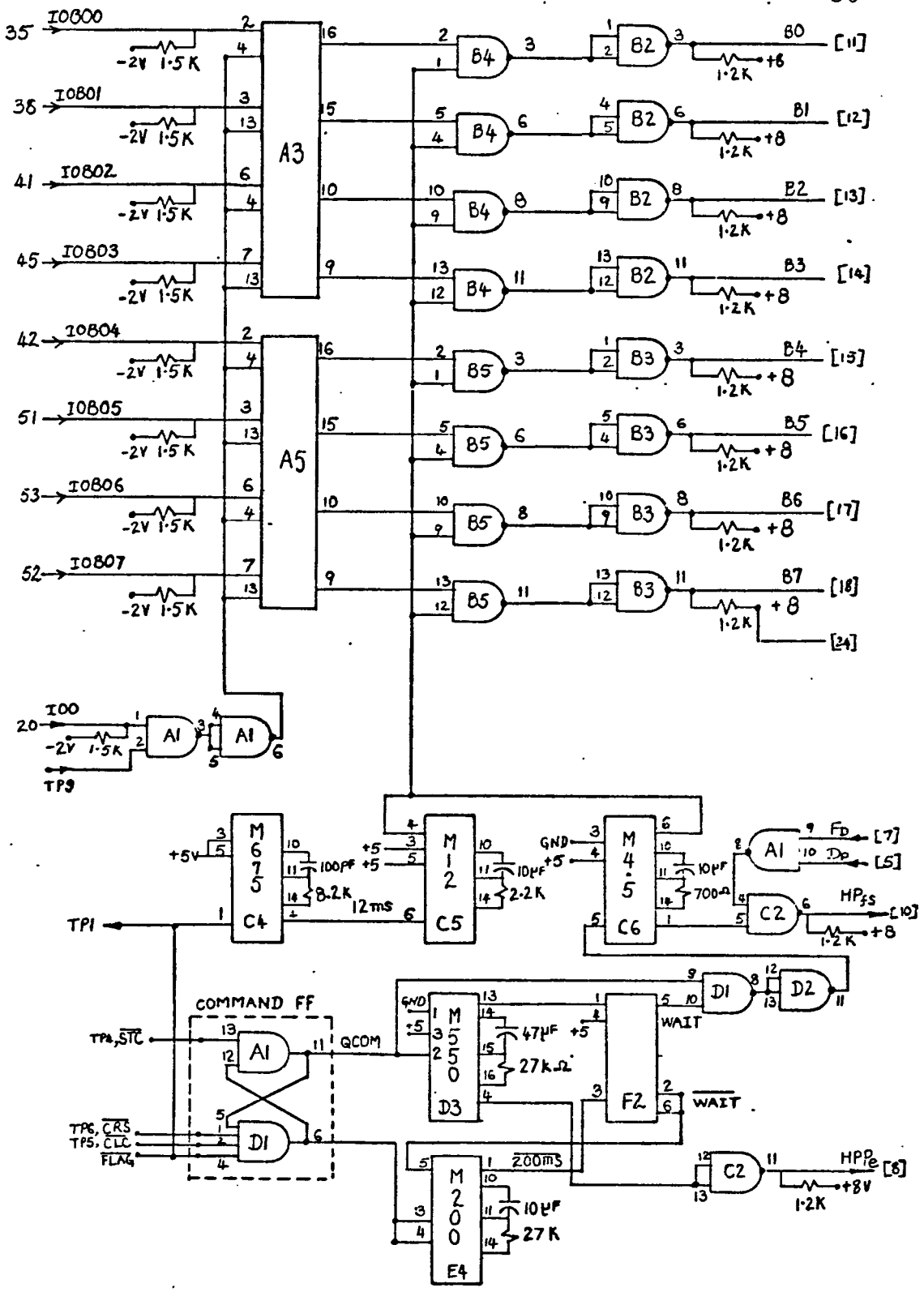
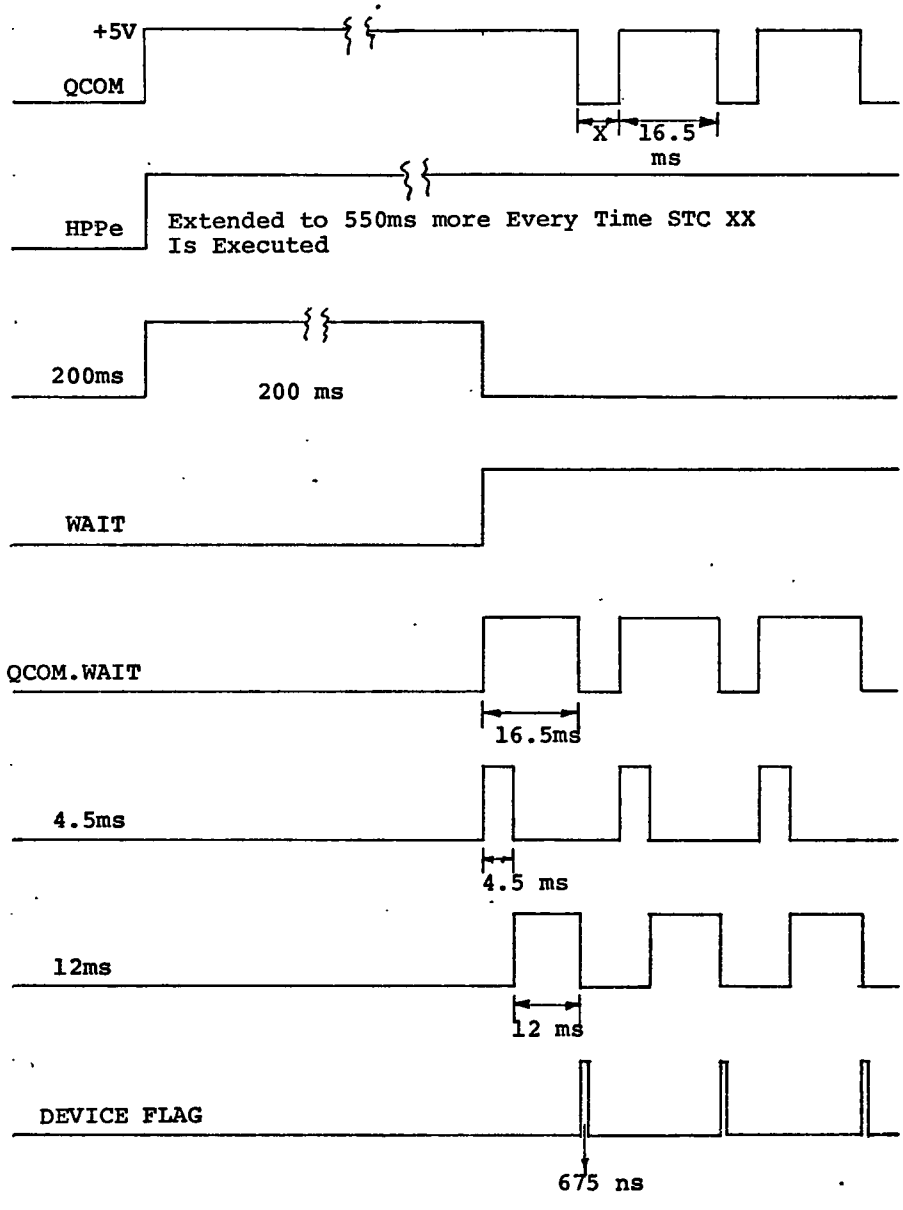


Figure 3.5 The Detail Logical Diagram of the Punch Interface

Note: (1) The Pin Assignment Is Given In Reference 3(Table 2.42)
(2) [YY] The Pin Number YY, Connected to the Device.



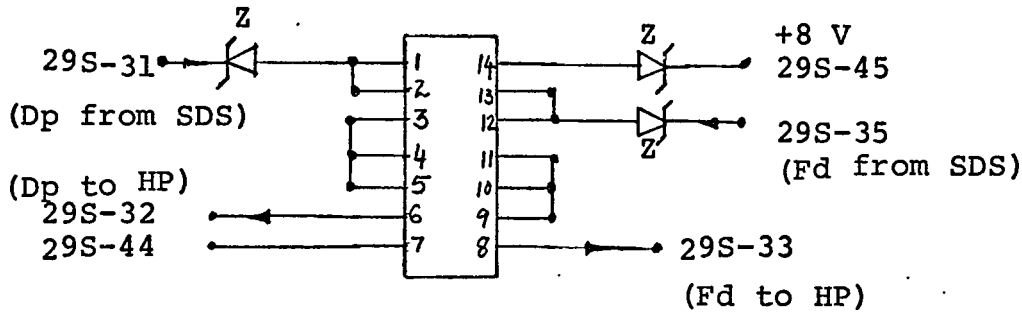
NOTE: X= Time Between the Interrupt of the Computer and the STC XX Instruction

Figure 3.6 Punching Data under HP Computer Control

3.5 enables the motor in the perforator. M550 monostable is connected to Rs input of the WAIT flip-flop, and the $\overline{Q}$ of M200 is connected as a clock input to the same flip-flop. During a pulse from M200, WAIT signal is false (logic 0) and the output of the command flip-flop is at logic 0. At the trailing edge of pulse from M200, the WAIT goes to its true state, which in turn enables the output of the command flip-flop (QCOM) and disable the M200 monostable from triggering. In addition, when the WAIT goes to its true state, the M4.5 monostable is triggered which enables all the drivers for the data (B0-B7) to be punched and for the movement of the tape (HP_{ff}) for the duration of 4.5 ms which is the requirement for the drive pulse. At the trailing edge of 4.5 ms pulse, M12 monostable is triggered in order to have a period of 16.5 ms for the drive pulse. The Device Flag signal is generated by triggering M675 monostable, at the trailing edge of the 12 ms pulse. The command flip-flop is cleared by the Device Flag and in addition, the computer is interrupted to indicate that the output transfer is completed. Thus one character is punched. If there are more to be punched then STC XX instruction is given individually for each character. If the pulse from M550 is at the logic level false (punch motor OFF) when the STC XX instruction is given for the next character, the above cycle of operation will be repeated. If the pulse from M550 is at the logic level true when the STC XX instruction is given, the pulse from the retriggerable monostable M550 is extended from the time the STC XX is given, to another 550 ms (motor on for another 550 ms). The output of the command

flip-flop is also enabled when STC XX instruction is given, and due to WAIT being in its true state, a M4·5 monostable is triggered. At the trailing edge of 4.5 ms pulse the M12 monostable is triggered, and Device Flag is generated at the trailing edge of the 12 ms pulse. Thus 4.5 ms, 12 ms and Device Flag together with the STC XX instruction form a counter to provide pulses for punching the data. Thus if next STC XX instruction is given within 550 ms (when the motor is ON) there will be no delay of 200 ms and data will be punched at every 16.5 ms interval for each instruction.

The line drivers (7400) are added as shown in Figure 3.7 to the SDS chassis to bring two signals Fd and Dp, in order to operate punch manually by the FEED switch. These signals are ANDed together and the output is ORed with the pulse from M4.5 to drive the feed and the sprocket drivers when the FEED bottom is depressed. The data (IOB00-IOB07) from TTL to SDS logic levels (B0-B7) are converted by open collector NAND gates with the output tied to +8 volts through 1.2K resistors as shown in Figure 3.5. The detail change in wiring on the SDS chassis is given in Appendix E. The cable connection for the Punch Interface is shown in Figure 3.8. The small program loop was executed to test the interface and the interrupt test was done by INTERRUPT TEST program. The layout of the interface card and the list of the components is given in Appendix E. The software part of the interface, adding a BCS High Speed Punch driver which was available and reconfiguring BCS system, was also done to operate the punch in the BCS environment. This completed the interface for the High Speed Paper Tape Punch.



NOTE: Z- Zenner Diode 1N746A,3V

Figure 3.7 Line Driver on 29S Card

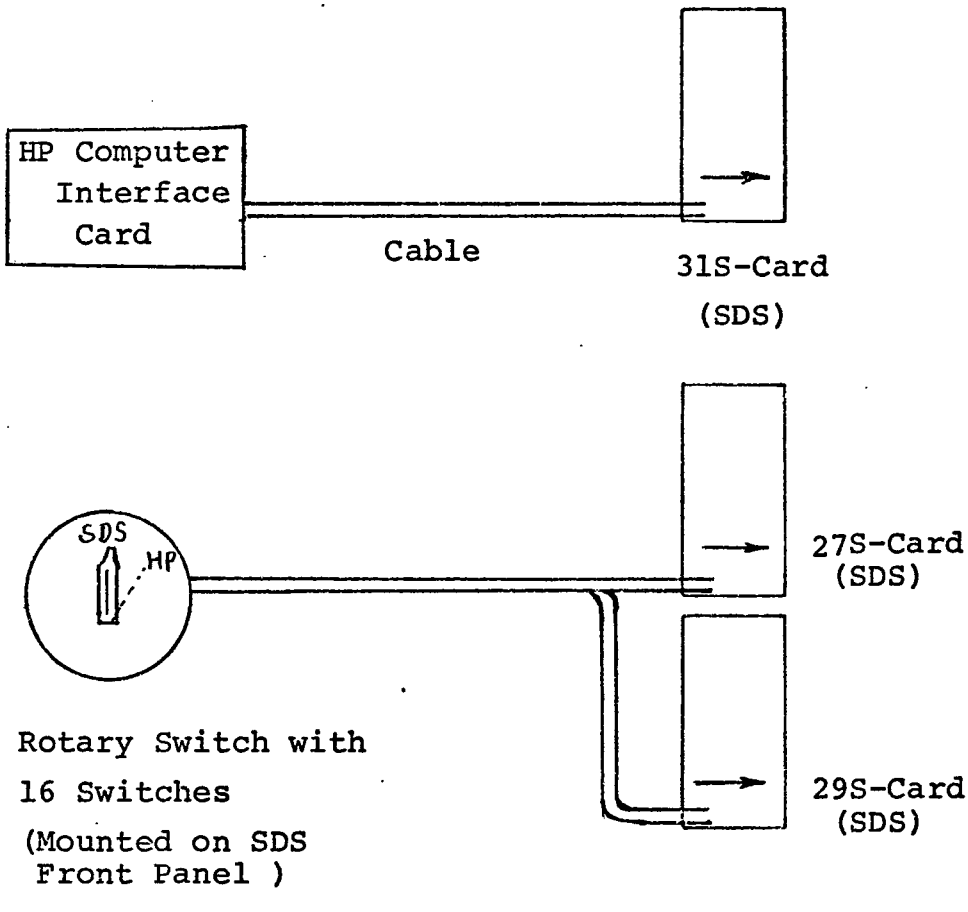


Figure 3.8 Cable Connection for the Punch Interface

CHAPTER IV

PRIORITY HANDLER

In a process control environment there would be several tasks to be performed by the computer. There would be status inputs to be checked, parameters to be input and messages to be printed to the operator. Each task would have a service subroutine. It would be desirable to have the priority for these tasks which would be controlled by the operator.

This is a function of a software priority handler to perform tasks according to a priority table entered by the operator. Such a handler should have the following characteristics.

1. Should allow service routines in FORTRAN.
2. Should allow an option to interrupt a subroutine by one of the higher priority.
3. Should keep track of all service subroutines asking for service.
4. Should allow nesting of interrupts.
5. Should be capable of logging error messages and setting different types of alarms for the errors.

The operational aspects of such a priority handler is discussed in the following sequences.

1. Start the program execution as given in the section OPERATING INSTRUCTION.
2. Enter priority table as given in Section 4.2, which sets up the priority for executing the service sub-

routines for different tasks.

3. Indicate for each service subroutine if it can be an interruptable or a non-interruptable routine. This can be indicated by the sign of the entry in a priority table as given in Section 4.2.

4. Request service routine by setting a bit of a word in a particular interface card. If a particular bit is one, the corresponding subroutine needs service. If a bit is zero, the corresponding subroutine doesn't need service. The service routine will be serviced according to its priority in the priority table. At the most there can be sixteen service routines asking for service at a time.

4.1 Introduction

The simplified Priority Handler functional flow chart is shown in Figure 4.1. It is written partly in FORTRAN and partly in assembly language. The Priority Handler system execution is started at program location 2_8. Like any other relocated program it is executed using the BCS system.* Loading of the priority table (IPRTB) is done by typing the elements of the table using TTY. If an error is made in configuring the priority table, the error messages are given and the priority table has to be loaded again. If no error is made, the interrupt location parameters are set in low cone area to link the interrupt from the Digital Input device and then the Digital** Input device is enabled by STC instruction. Afterward a wait

*See Reference 6, for the detail operation of the BSC system.
**See Appendix A, for the Digital Input Device.

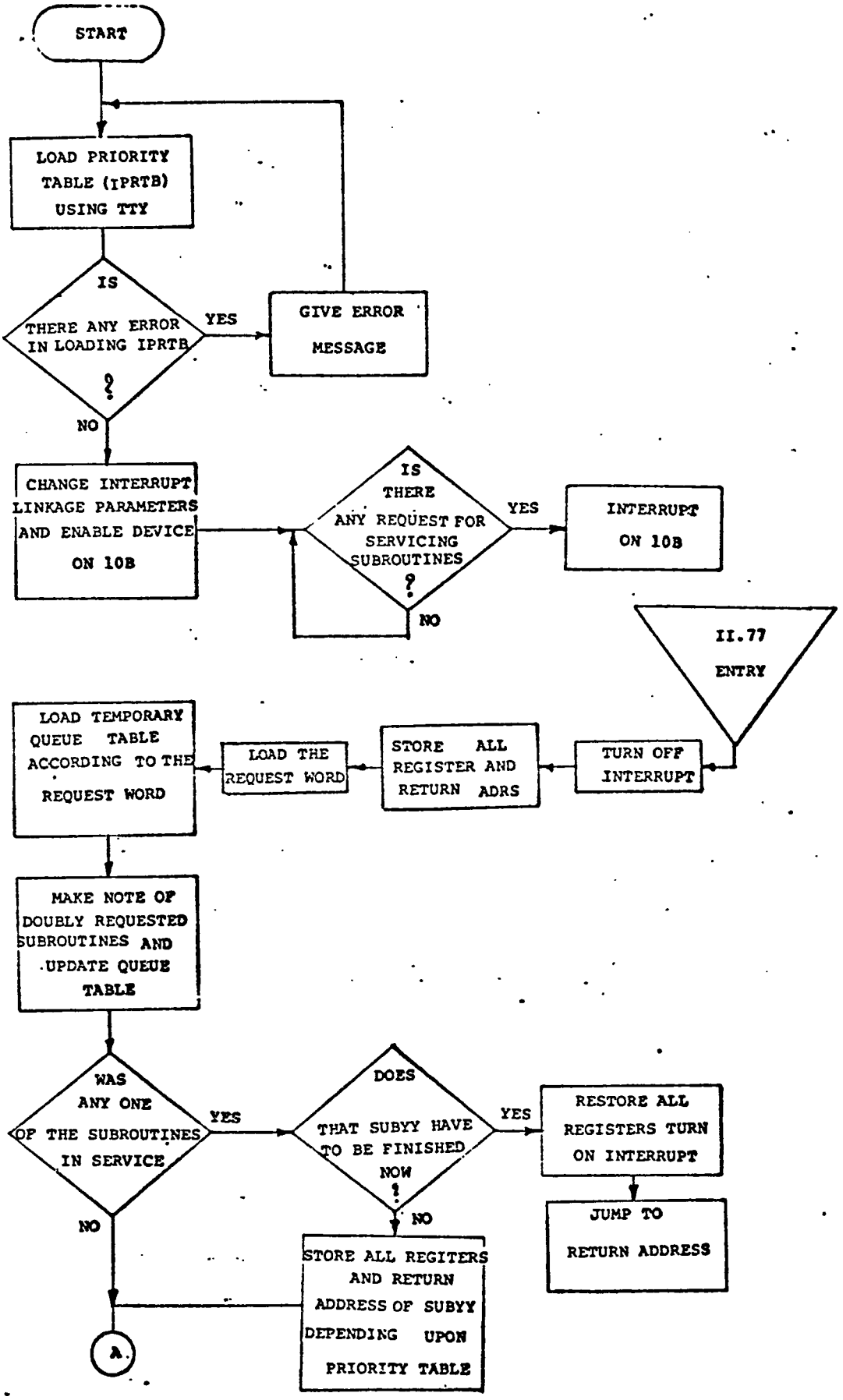


Figure 4.1 Functional Flowchart of Priority Handler

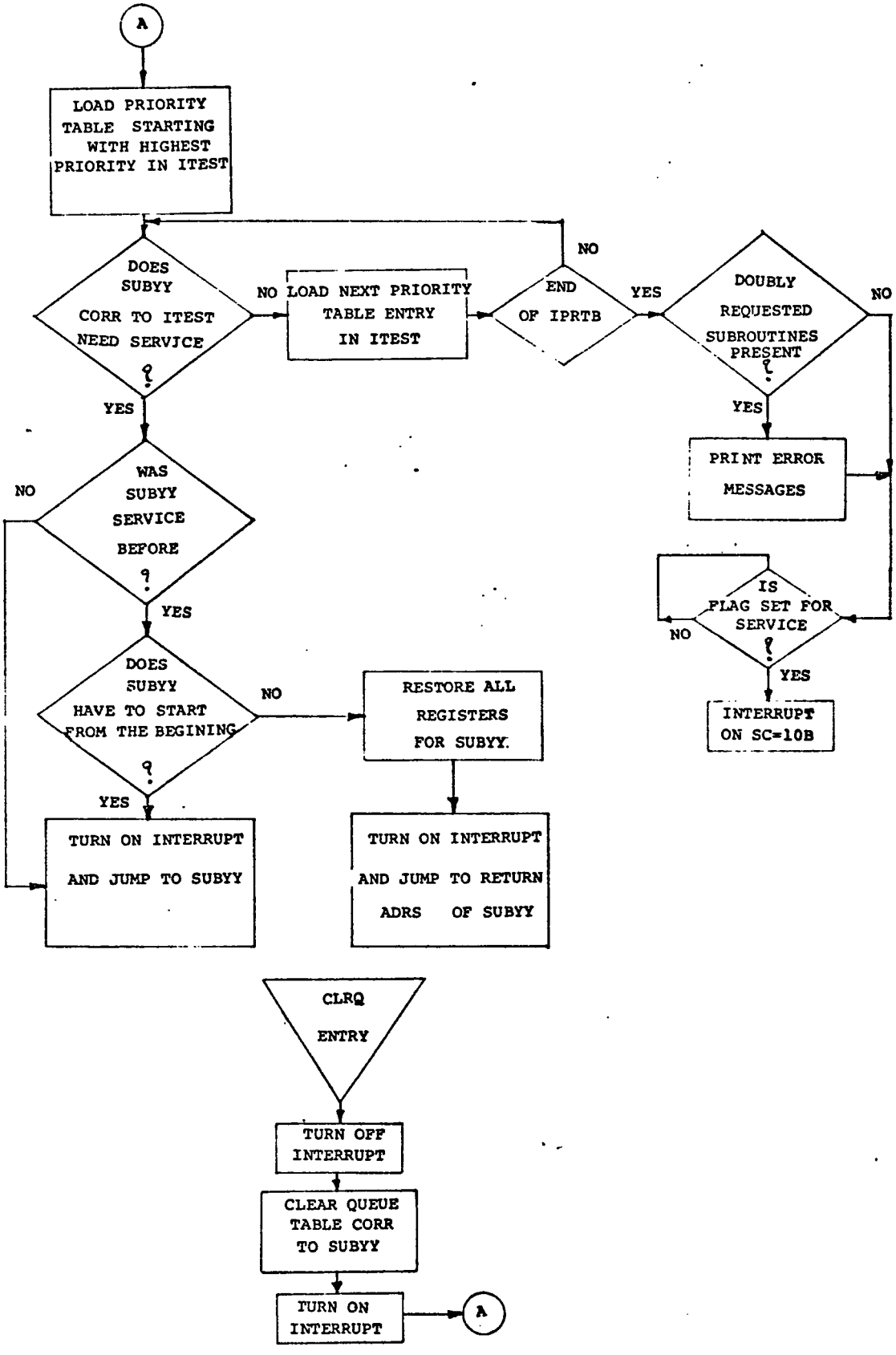


Figure 4.1 Functional Flowchart of Priority Handler (cont)

on flag set loop is created for the service request word. When the flag is set, the program is interrupted to process the request word for running the subroutines. The II.77 subroutine is executed as a result of the above interrupt. After the entry, the interrupt system is turned off so that no interrupt occurs during the processing of the requested word. The request word is loaded into a register after storing all working registers and the return address. Each bit of the request word corresponds to the request for running a particular subroutine out of 16 subroutines. If a bit is '0', the corresponding subroutine doesn't need service and if it is '1' the corresponding subroutine needs service.

These bits are loaded into the temporary queue table and then the permanent queue table (Q-table) is updated and the note for the doubly requested subroutines is made in order to print the error messages, when the computer is free. If any one of the subroutine was in service and interrupted, and if that subroutine (SUB YY) has to be finished immediately, (depending upon priority table) the return is made after restoring the working registers and setting the interrupt system active. If SUB YY does not have to be finished, the working registers and return address are stored into the storage area corresponding to the SUB YY depending upon the priority table. After this the Priority Handler processes as below. If none of the subroutines was in service or after being processed as above, the subroutines which are in queue according to the Q-table will be served starting from the highest priority subroutine to the lowest priority sub-

routine, depending upon the priority table configuration. The highest priority subroutine number is loaded in ITEST and check is made whether corresponding SUB YY (YY = ITEST) needs service or not by looking into the corresponding entry of the Q-table. If it does not need service, the next order priority subroutine is checked for the service in the same way. If none of the subroutines need service and the end of the priority table is reached, (computer is free), the check for the doubly requested subroutines, (if any of them were noted before) is made. If any of them were detected, error messages are given. After this or if no doubly requested subroutines are detected a loop on flag set (on s.c = 10B) is created in order to wait for the subroutine service request. When the flag is set, the program is interrupted by the interrupt on s.c = 10B, (Digital Input Device) which transfers control to the II.77 subroutine.

If any of the subroutines (SUB YY) needs service and if it was serviced before, depending upon priority table the subroutine is started from the beginning or from the point where it was interrupted. If it has to start from the point where it was interrupted, the working registers are restored from the corresponding storage area of the SUB YY and the interrupt system is enabled before the transfer is made to the SUB YY. If the SUB YY is running the first time or it has to be started from the beginning, the transfer is made to the beginning of the SUB YY after enabling the interrupt system.

The entry to the CLRQ section of the program is made by calling the CLRQ subroutine at the end of all the subroutine which are handled by the Priority Handler. After the entry, the

interrupt is turned off and the Q-table entry corresponding to SUB YY (which was in execution) is cleared indicating that the subroutine is serviced and the execution of this subroutine is finished. Transfer to the point A in flow chart is made to process the request for running the lower order priority subroutines after the interrupt system is enabled.

4.2 Priority Table

The Priority Table for running the subroutine has a typical element,

+/-NN, +/-YY

where,

NN = The order of loading the element of the Priority Table. Starting from 1 to 16 in the ascending order.

YY = The subroutine number in the name of the subroutine 'SUB YY'. YY equal to any number from 01 to 16.

The signs attached to NN and YY are for running the subroutine 'SUB YY'. The significants of the signs are,

1. +NN, SUB YY should be finished immediately when interrupted.

2. -NN, SUB YY does not have to be finished when interrupted.

3. +YY, when SUB YY is serviced, the execution is started from the point where it was interrupted.

4. -YY, when SUB YY is serviced, the execution is started from the beginning.

A sample of the Priority Table is illustrated as below.

```
-1,16
-2,-2
 3,3
 4,-4
-5,5
 6,6
-7,-7
 8,11
-9,13
10,-10
11,9
-12,-1
13,8
14,12
15,14
16,15
```

Sample Priority Table

In the above sample Priority Table, SUB 16 has highest priority, then SUB 02 and so on. SUB 15 has lowest priority. Now lets consider the entries for first four subroutines starting with the highest priority and following the sign conventions as discussed above, SUB 16 does not have to be finished when interrupted and whenever it is serviced execution starts from the point where it was interrupted. SUB 02 does not have to be finished and whenever it is serviced it starts from the beginning. The SUB 03 and SUB 04 have positive sign attached to number NN, so the sign attached to YY does not have any significant because whenever any one of them is interrupted, the execution of that subroutine starts back immediately.

4.3 Typical Subroutines

The subroutines handled by the Priority Handler can either be in FORTRAN or in assembly language. Following are two programs which illustrate the sequence of a typical subroutine handled by the Priority Handler.

FORTRAN SUBROUTINE.

The general form of the subroutine is:

```
SUBROUTINE SUBYY
COMMON  IDMMY(186),...*
:
:
:
:
:
CALL CL1OC
READ (....
CALL ST1OC
:
:
:
:
:
CALL CL1OC
WRITE (....
CALL ST1OC
:
:
:
:
:
CALL CLRQ
RETURN
END
END$
```

Before READ and WRITE statement CALL CL1OC, should always be specified, and CALL ST1OC should follow after them as illustrated above. The statement CALL CLRQ should always be placed before the 'RETURN' statement.

---

*If COMMON statement is specified in the program, the first array in COMMON statement should be IDMMY(186) which is a dummy array to reserve 186 common locations.

ASSEMBLY SUBROUTINE

The general form of the assembly subroutine is:

```
        NAM SUBYY              name of subroutine is SUBYY
        ENT SUBYY              YY = any number from
        EXT CL1OC,ST1OC,CLRQ   01,02,...,16
        COM IDMMY(186),...
  SUBYY NOP
        :
        JSB CL1OC
        DEF *+1
        :                      formatter calling
        :                      sequence
        JSB ST1OC
        DEF *+1
        :
        JSB CL1OC
        DEF *+1
        JSB  .IOC........      input/output calling
        :                      sequence
        JSB   ST1OC            in place of <normal return>
        DEF   *+1
        :
        JSB   CLRQ
        END
```

If the formatter or .IOC (for input/output operation) calling sequence is used, the CL1OC and ST1OC should be used before and after the calling sequence as shown in the general form. The 'JSB CLRQ' should be placed before the 'END' statement.

The Priority Hanlder may not work if above rules are not followed strictly. Care should be taken to see that the proper sequence is used.

4.4 Generation of the Priority Handler System

Like all relocatable programs, the Priority Handler system can be generated in two forms: (1) using BCS, relocate the code into core memory then execute it. (2) Using BCS, relocate

the code and punch an absolute tape which produces a permanent, runnable copy of the program.

OPERATING INSTRUCTIONS

1. Load a configured BCS into core with BBL or BBDL.

2. Set a starting address of 2_8.

3. Set all switch register bits off, then select the following options:

   Bit 15 on (suppress memory allocation listing)
   off (includes memory allocation listing)

   Bit 14 on (punch absolute tape copy of program)
   off (relocate into core, do not punch tape)

   If Bit 14 on and a teleprinter, is to be used for punching then,

   Bit 13 on (teleprinter is a 2754B and can print and punch separately; set teleprinter mode to KT)

   off (teleprinter cannot print and punch separately; BCS halts before and after each line of printing so that the operator can turn on/off punch unit to avoid punching list output, then punch absolute binary output).

4. Place the first relocatable program tape (one of the sixteen subroutines to be handled by the Priority Handler) into the reader. Press PRESET and RUN. BCS reads and relocates the binary code on the tape.

If switch register bit 14 is on, an absolute binary tape is punched. (Otherwise, BCS relocates the program in memory).

5. BCS halts after typing.

   *LOAD

   Load rest of the fifteen subroutines and the Priority Handler programs tapes as follows: set switch register bits 2-0 off. Place the tape in the reader. Set switch register bit 15 on (if desired) to suppress memory alocation listing. Press RUN. when tape has been read, BCS halts after typing.

   *LOAD

6. To read a library subroutine tape (and load only those subroutines which are necessary to resolve externals). Set switch register bit 2 on (bits 1 and 0 off). Place the relocatable library tape in the reader, in the following order one after another.

   a. B.O.S. library

   b. FORTRAN IV library

   c. BCS library

   Set switch register bit 15 on to suppress the memory allocation listing, if desired. Press RUN every time. When the tape has been read, BCS halts after indicating:

   No undefined externals

   *LST

(Set switch register bit 2 off and go to Step 9) or

Undefined externals

```
symbol
symbol
  ⋮
symbol
*LOAD
```

6a. To list undefined externals (or bypass further loading if there are no undefined externals), go to Step 7 or,

6b. To bypass further loading even if undefined externals remain, go to Step 8.

7. Set switch register bit 0 on (bits 1 and 2 off). Press RUN. BCS indicates whether undefined externals exist by printing either: No undefined externals

```
*LST
```

(set switch register bit 2 off and go to Step 9) or

Undefined externals

```
symbol
symbol
  ⋮
symbol
*LOAD
```

Return to Step 6a.

8. Set switch register bit 1 on (bits 2 and 0 off). Press RUN. BCS goes on to Step 10, even though undefined externals may still exist.

9. BCS has completed loading and is ready to print the Loader Symbol Table (LST), common bounds, and linkage area bounds. Set switch register bit 15 on to suppress listing of these items. Set bit 15 off to list them. If a 2754B Teleprinter is used, set the mode switch to "T" to enable the tape punch. Press RUN.

10. BCS completes listing (if requested by bit 15). If the program was relocated into core (bit 14 off), BCS prints

    *RUN

    Press RUN to execute the program.

11. If the program was punch onto paper tape (bit 14 on), BCS prints

    *END

12. Tear off the absolute tape output and wind. To execute the program:

    Load the tape with BBL or BBDL.

    Start the program at location 2_8.

4.5 Use and Configuration of the Priority Handler

The priority table is illustrated in Section 4.2. After starting the execution of the Priority Handler System, (as discussed in Section 4.4) the computer types,

*** LOAD PRIORITY TABLE ***

Load priority table entry one after another as decided. Terminate

each reply with a RETURN and LINEFEED. If an error is made while typing an entry of the priority table, press RUBOUT, RETURN and LINEFEED, then retype the entry. If no error is made in loading priority table, the computer prints,

*** NO ERROR ***

If an error is made, refer to the next section for the error messages and retype whole priority table again as corrected correction. After the computer types no errors message, load the input request word by means of the Digital Input Device Bit 1 corresponds to SUB01, and if it is 1, SUB01 needs service. If it is 0, SUB01 does not need service. The same correspondance is applied for rest of the fifteen subroutines.

4.6 Priority Table Error Messages

During the configuration of the priority table, error messages are printed on the TTY. Errors detected in the entry of the table are indicated by a numeric code inserted before and after the 'ERROR'. The formats are as follows:

1. LL *** ERROR-XX ***
2. YY ERRORS ****

LL	The entry line number in which error was detected, starting one from the top of the table.
XX	The error diagnostic code shown below.
YY	Number of errors totally occurred.

ERROR CODE

01 NN in the entry is not in the ascending order, starting from 1 to 16.

02 YY, greater than 16 is an invalid subroutine number.

03 YY=0 is an invalid subroutine number.

04 YY is a duplicate subroutine number.

CHAPTER V

SOFTWARE DEVELOPMENT

5.1 Software Development of the Priority Handler

The detail software development of the Priority Handler is discussed in this chapter. The simplified flow chart for the Priority Handler subroutines are illustrated for the explanation purpose. The complete listing of the Priority Handler is given in the Appendix A. The symbols for the common arrays are explained in the beginning of the program LPRTB. Dummy labels for the common arrays are used in some of the subroutines where only few common arrays are required. Each subroutine is discussed separately one after another.

a. LPRTB

Refer to Figure 5.1 and the complete listing in the Appendix A. This is the program written in FORTRAN to load correct priority table, IPRTB(J), array IFINS(J), and array IRSTR(J).

THE COMMON ARRAYS ARE,

```
IQTB(J)   - QUEUE TABLE ENTRY FOR SUBYY, J=YY.
J=1,16    = 0,  SUBYY DOES NOT NEED SERVICE
          = 1,   "    NEEDS SERVICE
IQTEM(J)  - TEMPORARY QUEUE TABLE FOR 'SUBYY', LOADED FROM DIGI-
J=1,16      TAL INPUT DEVICE.
          = 0., DOES NOT NEEDS SERVICE.
          = 1,  NEEDS SERVICE.
ISVTM(J)  - TEMPORARY STORAGE AREA FOR WORKING REGISTER A,B,E,O,
J=1,4       AND RETURN ADDRESS WHEN SUBROUTINE IS INTERUPTED.
INOW      - THE PRESENT SUBROUTINE NUMBER IN SERVICE, ANY FROM
            1 TO 16.
IFINS(J)  = +VE, IF ENTRY IS '+NN,+/-YY' FOR SUBYY, AND J=YY
J=1,16    = -VE, "    "    " '-NN,+/-YY'  "   "      "   "
            IF +VE, THEN SUBYY IS FINISHED IMMEDIATELY WHEN INTERRU-
            PTED.
```

```
            IF -VE, THEN SUBYY IS NOT FINISHED IMMEDIATELY WHEN
            INTERUPTED.
IREG(J)   - STORAGE AREA FOR A,B,O,E REGISTERS AND RETURN ADDRES
J=1,16      FOR 16 SUBROUTINE, FOUR FOR EACH SUBYY.
IRSTR(J)  - +VE, IF ENTRY IS '+/-NN,+YY' FOR SUBYY, J=YY
J=1,16    = -VE,  "    "    "  '+/-NN,-YY' "     "     "
            IF +VE, THEN WHENEVER SUBYY IS SERVED, IT SHOULD BE
            STARTED FROM THE POINT, WHERE IT WAS INTERUPTED.
            IF -VE, THEN SUBYY SHOULD BE STARTED FROM THE BEGINING
IPRTB(J)  - THE PRIORITY TABLE ENTRY, STARTING FROM HIGHEST.
J=1,16    = YY, IN THE ENTRY '+/-NN,+/-YY', AND J=NN.
ILOAD(J)  - TO RESTORE A,B,O,E REGISTERS AND RETURN ADDRESS FOR
J=1,4       THE INTERUPTED SUBROUTINE WHEN IT IS SERVED.
ICHAK(J)  = -VE,IF SUBYY YY=J WAS ASKED FOR THE SERVICE AGAIN
J=1,16      WHILE PREVIOUS REQUEST WAS NOT FINISHED.
            SUBYY IS PRINTED AS DOUBLY REQUESTED SUBROUTINE, IF
            ICHAK(J) =-VE, AND COMPUTER IS FREE.
ICHAK(17)= +VE, NONE SUBROUTINES WERE DOUBLY ASKED.
         = -VE, SOME SUBROUTINES WERE DOUBLY ASKED.
INUMM(J)  = NN,THE ORDER OF THE PRIORITY TABLE IN '+/-NN,+/-YY'
J=1,16      AND J=NN. THIS ARRAY ELEMENT ARE +VE IN THE BEGINING
            WHEN SUBYY IS SERVED, THEN INUMM(YY) IS MADE -VE TO
            INDICATE THAT SUBYY WAS SERVED. WHEN SUBYY IS SERVED
            COMPLETELY, IT IS MADE +VE.
```

These common arrays are initialized by calling the subroutine INITL. IEROR, which represents the number of errors occurred in the priority table, is also initialized. Then computer prints message for loading the priority table through the TTY. The priority table and the message are as given in the Section 4.2 and Section 4.5 respectively. The reading of the entry is done one after another. The DO loop is created to check the correct priority table. If INUMM(J) (which is equal to NN in the entry) is negative ,it is made positive and NEGNM is made negative. If INUMM(J) is not in the ascending order, starting from 1 to 16, error 01 is given and IEROR is incremented. If IPRTB(J) (Equal to +--YY in the entry) equal to zero, error 03 is indicated and IEROR is increased by 1. If IPRTB(J) is positive and if

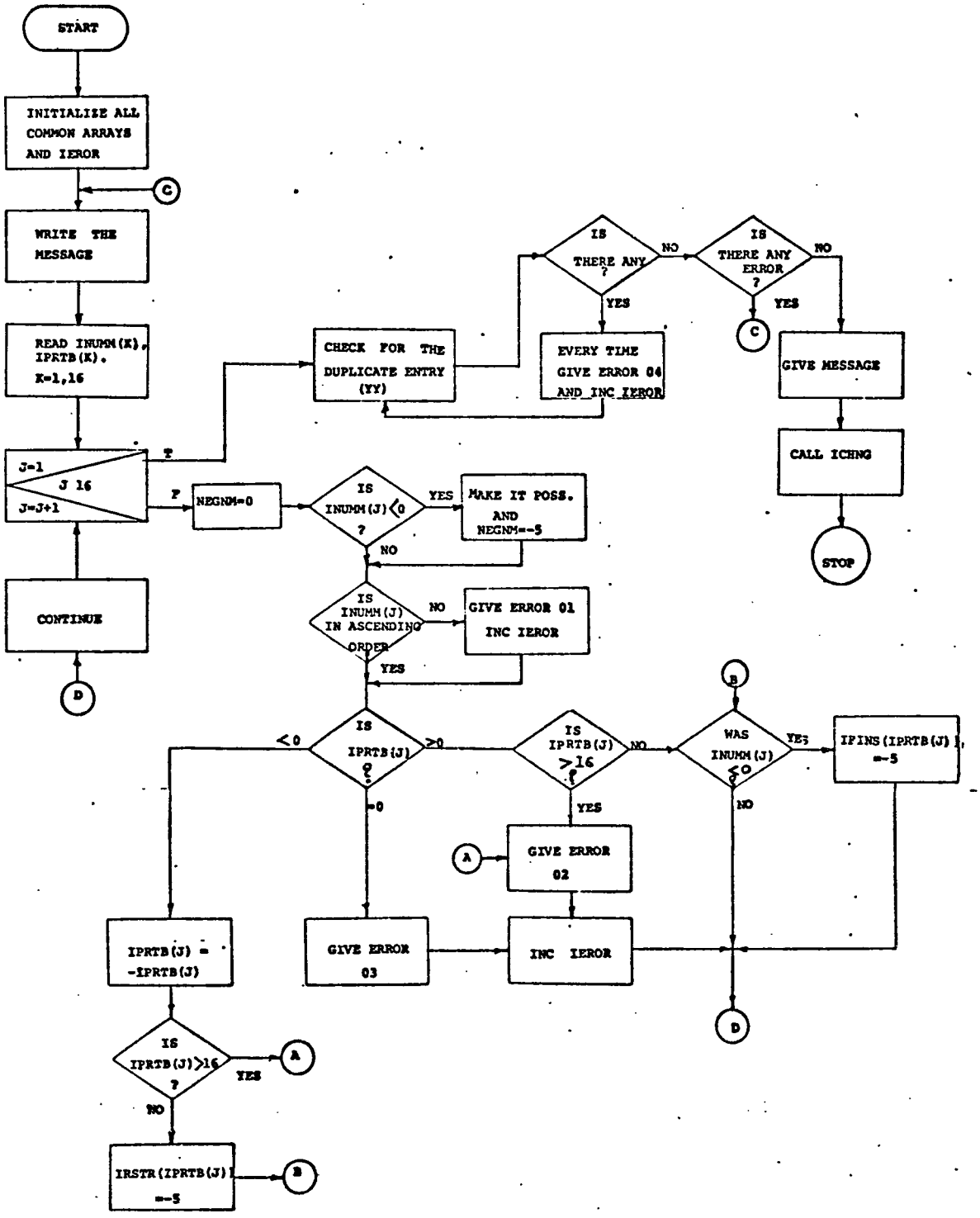


Figure 5.1 LPRTB

it is less than 16, INUMM(J) was negative or not is checked (By NEGNM). If it was negative, IFINS(IPRTB(J)) is made negative to indicate that the subroutine (SUBYY) need not be finished when interrupted. If INUMM(J) was positive, IFINS(IPRTB(J)) is not changed (which is zero) to indicate that the subroutine has to be finished immediately when interrupted. If IPRTB(J) is greater than 16, error 02 is given and IEROR is incremented. If IPRTB(J) is negative, it is made positive. If the absolute value of IPRTB(J) is greater than 16, transfer is made to A for the error 02. If negative and the absolute value is less than 16, IRSTR(IPRTB(J)) is made negative to indicate that whenever subroutine corresponding to IPRTB(J)(SUBYY) is served, it should be started from the beginning. Then transfer is made to B for restart condition as explained above. When above loop is finished, the check for the duplicate subroutine number is made. If any duplicate number is found, error 04 is indicated and IEROR is increased. Finally if any error is made in the priority table, the transfer is made to point C for loading the priority table again. If no error is involved, message for that is given and then subroutine ICHNG is called which does not transfer the control back to LPRTB.

b. ICHNG

Refer to the Appendix A. (The flow chart is not given). This assembly language subroutine sets up the interrupt linkage parameters in the base page to link the interrupt on channel 10. The wait on system bussy loop is created to finish

the printing message, then JSB 30B,I is loaded in location 10B and absolute address of II.77 is loaded in location 30B. The interrupt system is turned off and the wait on flag set (s.c.=10B) loop is created, after starting the device (Digital Input) when the flag is set, meaning request word for running subroutines needs to be processed, the interrupt system is turned on and JMP* is executed for the loop. The interrupt on s.c.=10B transfers the control to the subroutine II.77.

c. II.77

This subroutine is entered as a result of an interrupt generated for loading the service request word. It stores the working registers overflow and return address in the array ISVTM(J). It also loads request word into temporary queue table IQTM(J) and then jumps to subroutine INHAN. Refer to Figure 5.2 This section, II.77, is entered via a Jump Subroutine instruction stored in the interrupt location associated with the Digital Input device. When interrupt on s.c.=10B occurs, the return address is stored in the entry address II.77. Upon entry to the II.77, the interrupt system is disabled, in order to prevent further interrupt while processing of the request word is not finished. The flag on S.C.=10B is cleared and then all working registers and return address are stored into the temporary common array ISVTM(J). The request word from the Digital Input device is loaded into temporary common array IQTM(J). Finally transfer is made to the subroutine INHAN.

d. INHAN

This subroutine makes note of doubly requested sub-

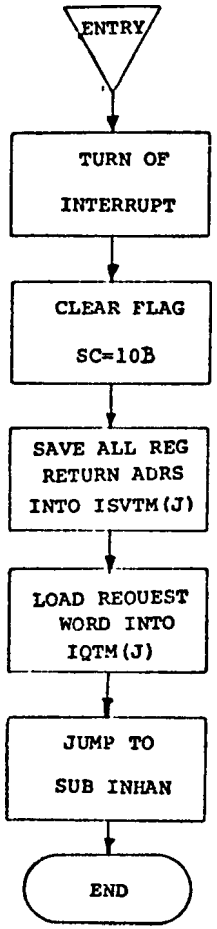


Figure 5.2 II.77

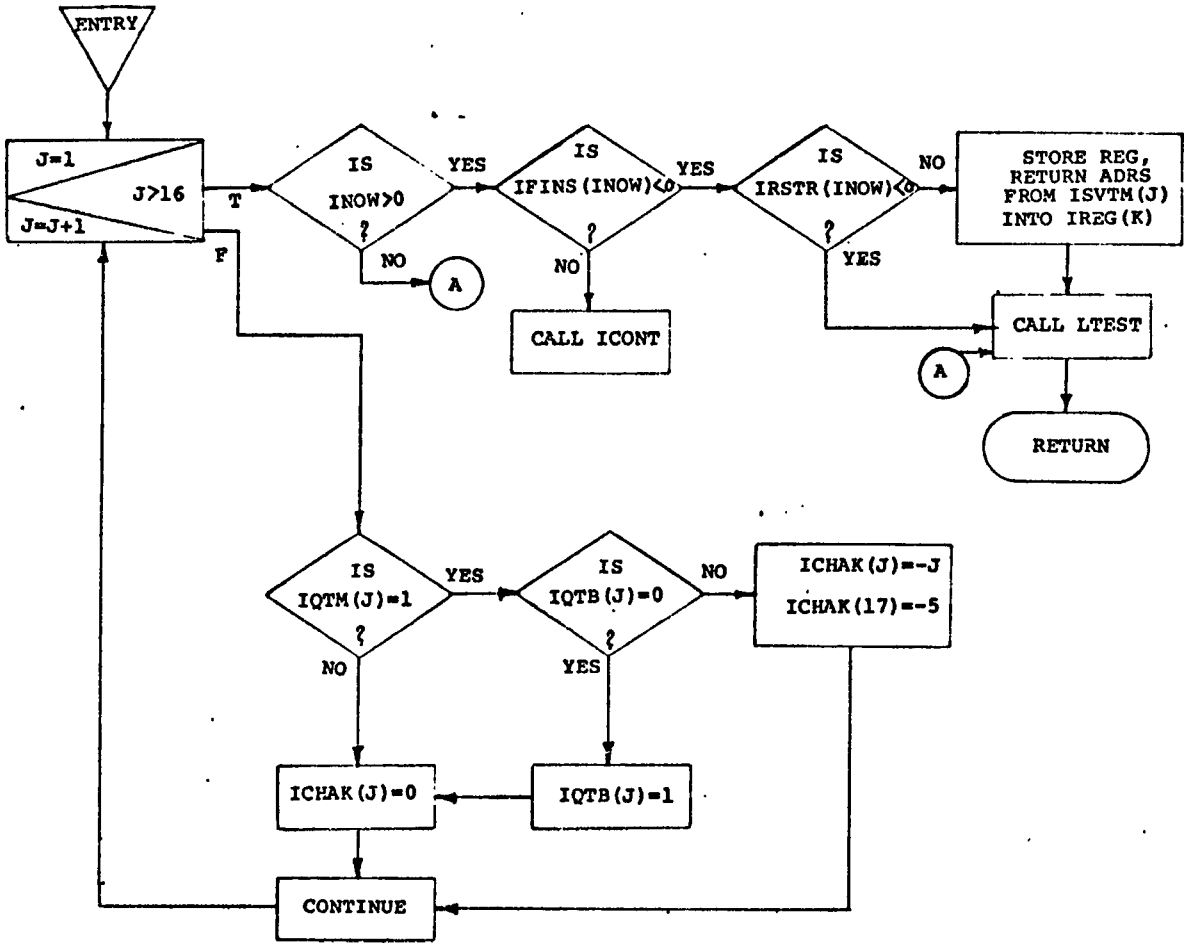


Figure 5.3 INHAN

routines. After this it transfers control to either subroutine ICONT or LTEST, depending upon certain conditions. If it transferscontrol to LTEST, the working registers, overflow and return address which were stored by II.77 are transferred to array IREG(K). Figure 5.3 represents the simplified flow chart of the subroutine INHAN. The request word which was loaded into the temporary queue table IQTEM(J) is processed and permanent queue table IQTB(J) is up-dated in this section. The DO loop is formed to check the request for all the subroutines as below. If IQTM(J) equal to 1, the corresponding subroutine (SUBYY,YY=J) needs service. If it is zero, the subroutine does not need service. So if IQTM(J) is zero, the array element ICHAK(J) is made zero which indicates that the subroutine is not doubly requested. If IQTM(J) equal to one and if the permanent queue table entry, IQTB(J), is zero, IQTB(J) entry is made one with ICHAK(J) equal to zero. If IQTB(J) equal to 1 (the corresponding subroutine SUBYY, YY=J, is already requesting service) and the subroutines is again requesting for the service (IQTM(J)=1), the error for the doubly requested subroutine is indicated by making ICHAK(J) negative. The ICHAK(M) is made negative to indicate that there are some doubly requested subroutines. Thus all the sixteen subroutines request is checked one after another.

After finishing the above loop, INOW is checked which indicates the subroutine number which was in service and interrupted. If INOW is greater than zero, some subroutine was in service when interrupt occurred. If INOW is zero, none of the subroutines was in service and the transfer is made to the point A as shown in the flow chart. If INOW is greater than zero and IFINS(INOW)

is not less than zero, the subroutine ICONT is called to continue the subroutine, SUByy (YY=INOW) which was in service. If IFINS (INOW) is negative (the subroutine which was in service does not have to be finished now), the IRSTR(INOW) is checked. If IRSTR (INOW) is negative, (the SUBYY, YY=INOW, should be started from the beginning whenever it is serviced) the subroutine LTEST is called. If the subroutine has to start from the address where it was interrupted, (IRSTR(INOW) greater than zero) all the working registers and return address, which are stored temporarily in the array ISVTM(J), are stored into the four storage areas of IREG(K). IREG(K) is the common array of length 64, four for each subroutine. Then subroutine LTEST is called.

e. LTEST

Refer to Figure 5.4. This subroutine processes the request for running the subroutines, which are waiting for the service according to queue table (IQTB(J)) which is updated by the subroutine INHAN. The subroutines are tested for the service, starting with the highest priority. If a subroutine needs service, condition for servicing it from the beginning or from the address where it was interrupted depending upon the priority table, is set up.

Upon entry to the LTEST, the DO loop is formed as shown in the flow chart, Figure 5.4. ITEST is loaded with the highest priority subroutine number from the priority table, which is IPRTB(L) for L=1. The Q-table entry (IQTB(ITEST)) corresponding to the subroutine number, ITEST, is checked for the service. If IQTB(ITEST) is not greater than zero, (the subroutine does

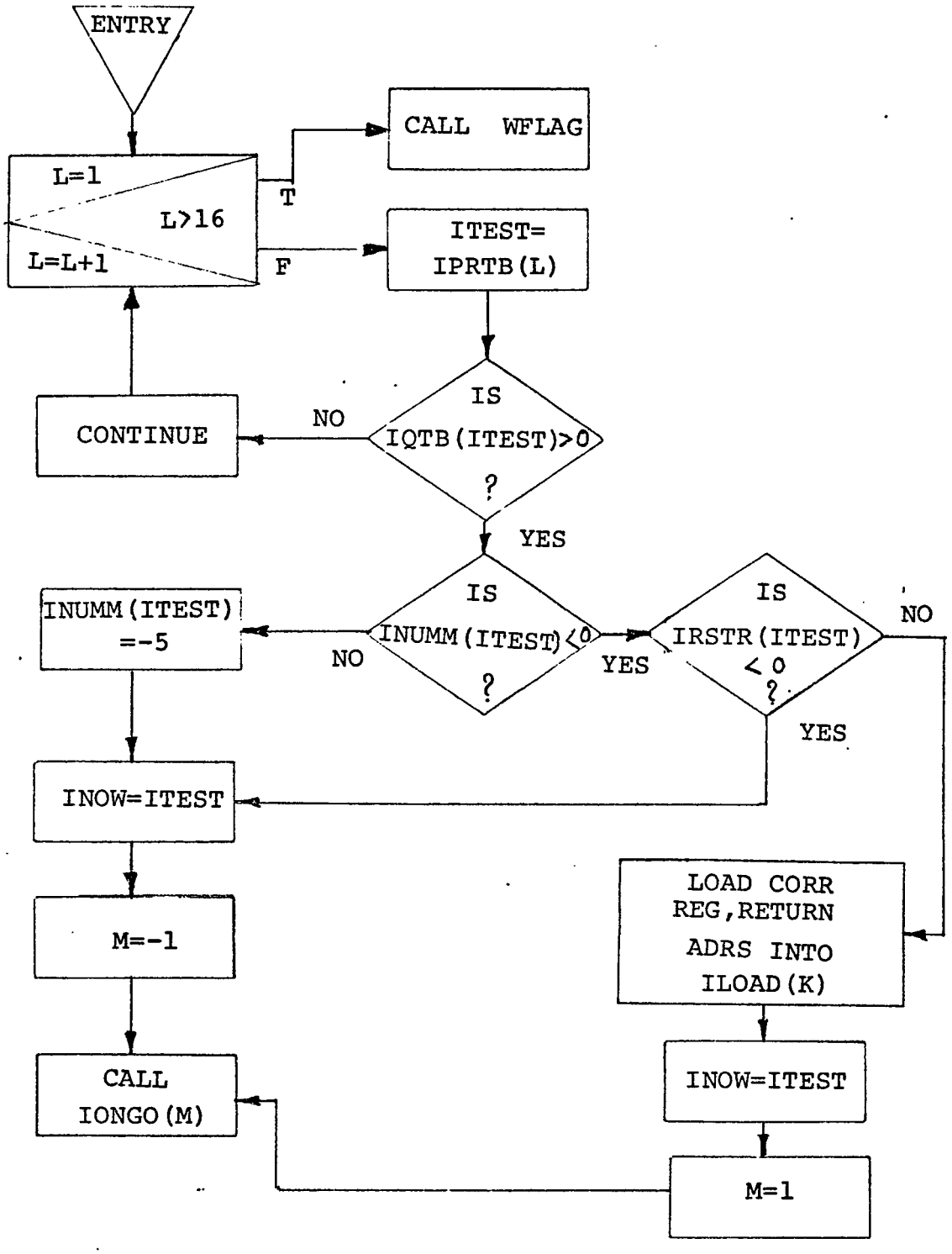


Figure 5.4 LTEST

not need service) the lower order priority subroutine is checked for service request. If IQTB(ITEST) is greater than zero, (the subroutine needs service) it is served as below.

The INUMM(ITEST) is checked. If it is zero, the subroutine is running for the first time, and if negative, the subroutine was serviced before but it is unfinished. So if INUMM(ITEST) is not negative, it is made negative and INOW, which indicates present subroutine in service, is loaded with the subroutine number ITEST. The subroutine IONGO(M) is called with M=-1 which indicates that the subroutine corresponding to INOW, has to start from the beginning. If INUMM(ITEST) is negative, the IRSTR(ITEST) is examined. The IRSTR(J) array is set up at the time of configuring the priority table. If IRSTR(J) is negative, the subroutine, (SUB YY, YY=J) should be serviced from the beginning. If IRSTR(J) is not less than zero, the SUB YY, YY=J, should be serviced from the address where it was interrupted. So if IRSTR(ITEST) is less than zero, INOW is made equal to ITEST and subroutine IONGO(M), with M=-1 is called to start the subroutine from the beginning. If IRSTR(ITEST) is not less than zero, the common array ILOAD(K) of length four is loaded to load the registers and return address, from the four storage area of IREG(K) corresponding to the SUB YY, YY.= ITEST. Then INOW is made equal to the ITEST and IONGO(M) is called with M=1 to start the subroutine from the point where it was interrupted.

f. ICONT

Refer to the Appendix A for the detail listing. This subroutine is called from the subroutine INHAN. Upon entry

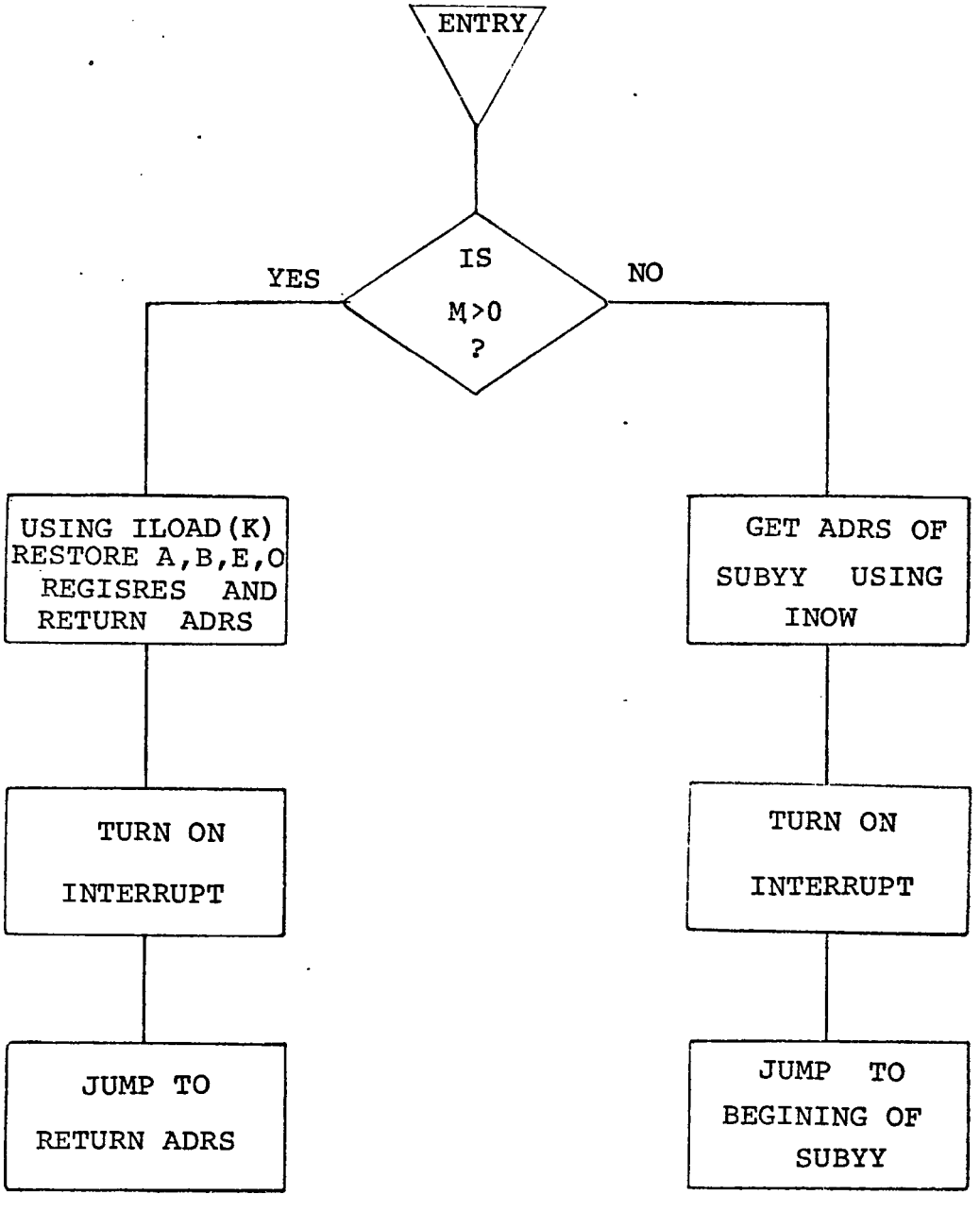


Figure 5.5 IONGO(M)

to the subroutine, the working registers (A,B,E and 0) are restored from the temporary storage area ISVTM(J). The transfer is made back to the address ISVTM+3 of the subroutine which was in service, after enabling the interrupt system.

g. IONGO(M)

Refer to the simplified flow chart, Figure 5.5. The subroutine IONGO(M) is called from the subroutine LTEST. This subroutine transfers the control to the beginning of a subroutine, for M=-1 or to the address from where it was interrupted when M is equal to 1. If M is not positive, the subroutine address for SUB YY, YY = INOW, is obtained and jump to the beginning of the subroutine is made after turning the interrupt system on. If M is positive, registers A,B,E and 0 are restored using ILOAD(X) and transfer to the return address for the subroutine which is going to be served is made. Thus subroutine corresponds to INOW is restarted from the point where it was interrupted, after the interrupt system is turned on.

h. WFLAG

Figure 5.6 represents the simplified flow chart for the subroutine WFLAG. When none of the subroutines need service, this subroutine is called to work for the new service request word. Besides waiting for the new request word, it calls subroutine WEROR to print the errors for the doubly requested subroutines if they exist.

Upon entry to the subroutine, INOW is cleared to indicate that none of the subroutines is in service. If ICHAK(17)

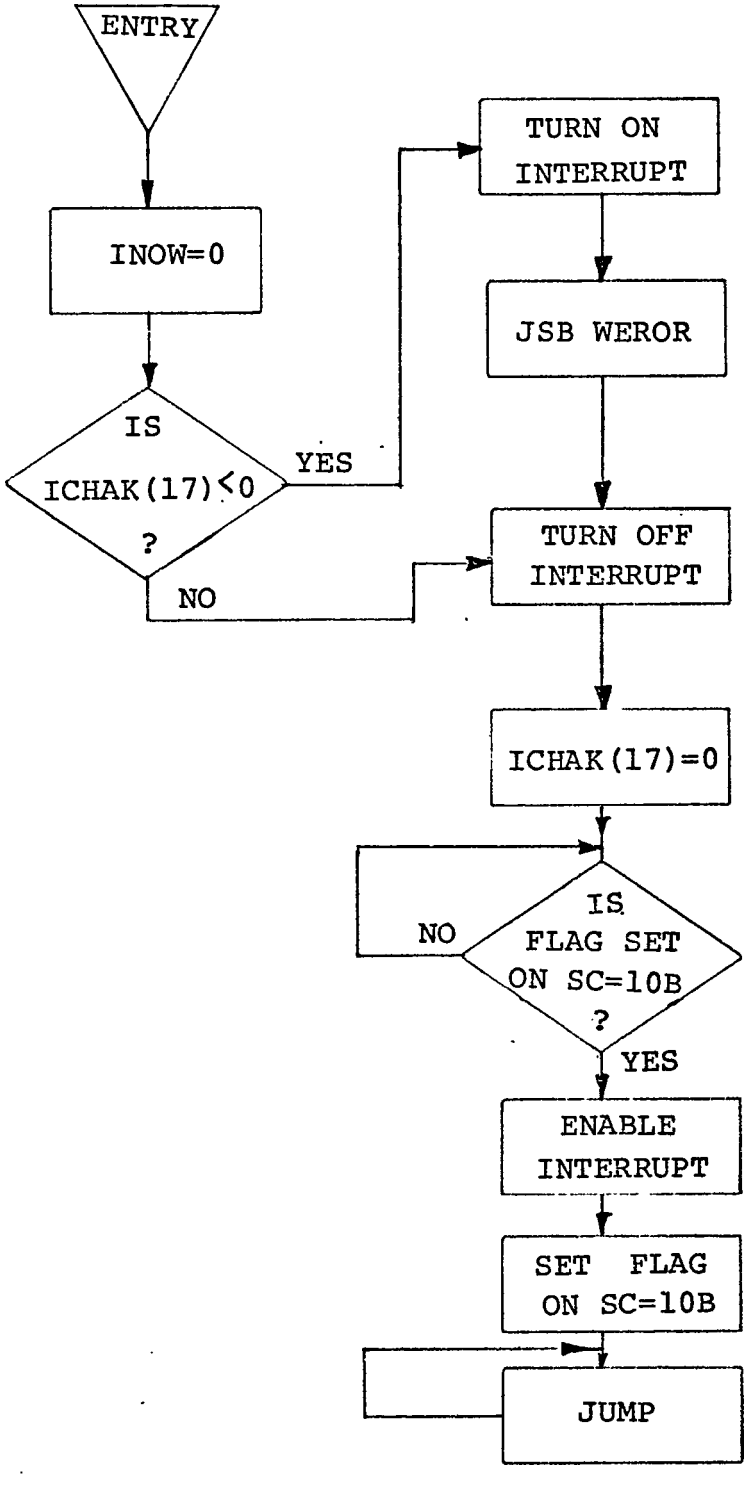


Figure 5.6 WFLAG

is negative, (doubly requested subroutine exist) the subroutine WEROR is called to print the error messages after enabling the interrupt system. When control returns, interrupt system is turned off. If ICHAK(17) is not less than zero, (doubly requested subroutines do not exist) the interrupt system is turned off. Then ICHAK(17) is made zero to indicate that doubly requested subroutines do not exist. The wait on flag set loop (Flag on s.c. = 10B) is created. The flag set means the request word needs to be processed. If the flag is set, interrupt system is enabled and jump to itself loop is created to wait. The interrupt on s.c. = 10B interrupts the computer, and control is transferred to II.77.

i. CLRQ

Figure 5.7 shows the simplified flow chart for the subroutine CLRQ. As shown in typical subroutine, Section 4.3, the CLRQ subroutine is always called at the end of a program. The subroutine in service calls CLRQ at the end, to clear the request for running this subroutine. (Which is represented by corresponding entry in IQTB(J)). Thus when subroutine is serviced completely, the corresponding entry in Q-table is cleared. Upon entry to this subroutine, interrupt system is turned off and IQTB(INOW) is cleared. Also NUMM(INOW) is cleared to indicate that the subroutine corresponding to INOW is serviced completely. Finally LTEST is called to process other request for running the subroutines which are waiting in a queue.

j. WEROR

The detail listing of this subroutine is given

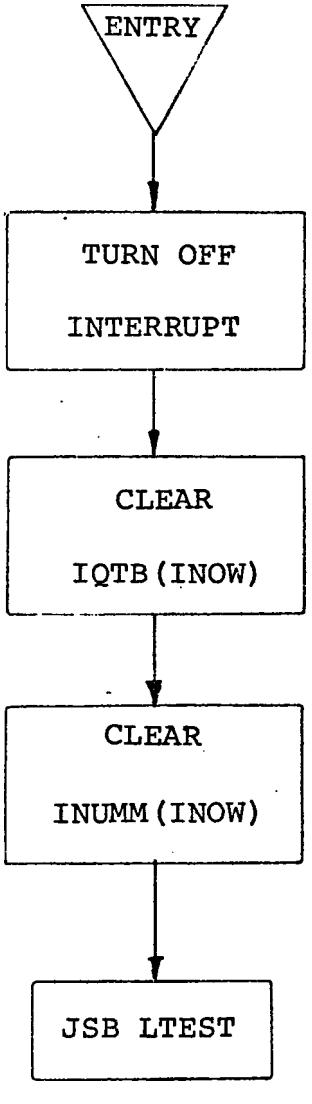


Figure 5.7 CLRQ

in the Appendix A. Upon entry to the subroutine WEROR, which writes errors for the doubly requested subroutines, the CL1OC is called before the loop. The ST1OC is called at the end, before return is made. As shown in the typical subroutine, Section 4.3, it was necessary to call these two subroutines. The reason for this is explained in the next section. The error for doubly requested subroutine is given, if ICHACK(J) corresponding to this subroutine is negative. The normal return is made at the end of the program.

The flow between the Priority Handler subroutines is shown in Figure 5.9. The names in the blocks are the entry name of the subroutines. A short description of each subroutine function is given below.

1. LDPRT: LDPRT loads correct priority table through TTY. Any error during generation of priority table is indicated and it waits for the corrected table.

2. INITL: INITL initializes the common arrays.

3. ICHNG: ICHNG sets up the interrupt linkage parameters in the base page to link the interrupt to subroutine II.77 and then generates the interrupt to jump to subroutine II.77.

4. II.77: II.77 is entered as a result of an interrupt. It loads request word in the temporary queue table after restoring the A,B,E and 0 registers, and return address.

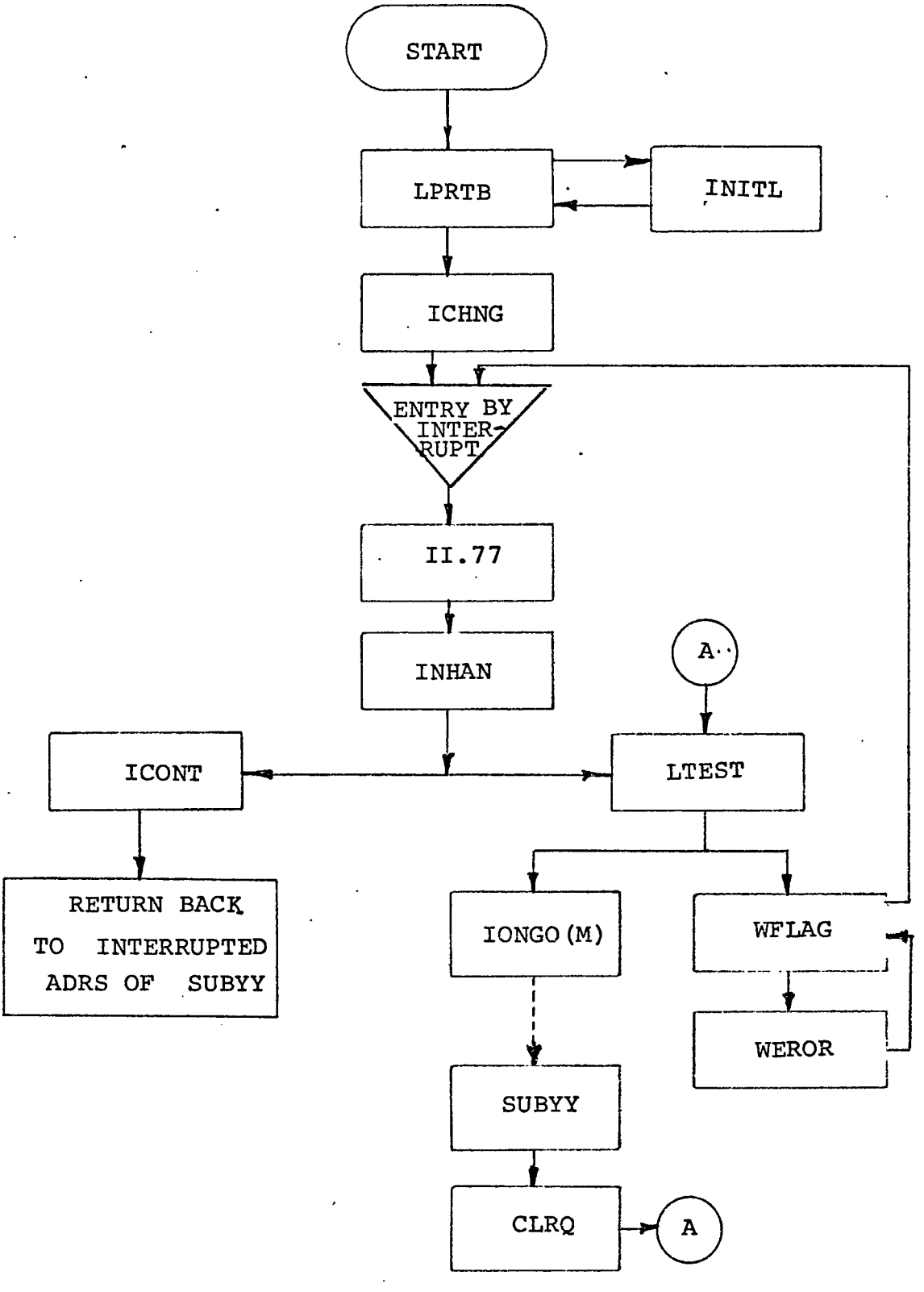


Figure 5.9 Flow Chart for the Priority Handler Subroutine

5. INHAN: INHAN makes note of doubly requested subroutines and updates the queue table. Then it transfers control to either subroutine ICONT or LTEST, depending upon certain conditions.

6. LTEST: LTEST tests the subroutines for the service, starting with the highest priority. A condition for running a subroutine from the beginning or from the address where it was interrupted depending upon the priority table, is set up.

7. ICONT: ICONT transfers control for continuing the execution of the subroutine, which was in service and interrupted, at the address where it was interrupted.

8. IONGO: IONGO transfers control to the beginning of a subroutine or to the address from where it was interrupted after turning on the interrupt system.

9. WFLAG: WFLAG waits for the new service request word when no more subroutine needs service.

10. WEROR: WEROR writes error messages for the doubly requested subroutines.

11. CLRQ: CLRQ clears the entry in the queue table corresponding to the subroutine from which it is called.

12. CL1OC: CL1OC clears the control flip-flop of the device on channel 10B.

13. ST1OC: ST1OC waits for any of the device being busy. When none of the device is busy, and if a new request word needs to be loaded in the queue table for the service, control is transferred to II.77 by means of an interrupt. Otherwise it enables the control flip-flop of the device on channel 10B.

5.2 The Problems and Solutions Involved in the Priority Handler

In the beginning the Priority Handler was written without the changes mentioned in this section. The problems involved without the changes are discussed with their solution. The changes are, (1) calling the CL1OC and ST1OC subroutines before and after the FORTRAN WRITE or READ statements, or any time the IOC subroutine is called for the input/output operation. (2) Addition of new B.O.S. library subroutine tape. (3) Changing the .ENTR subroutine in the B.O.S. library.

The change number one is introduced to inhibit the interrupt from channel 10B while input/output operation was being executed.

The problem was present due to the fact that the existing HP software system subroutines are not re-enterant. The re-enterant subroutine means, if the interrupted subroutine is used by some other program and if it is entered again from the interrupted address, that subroutine should work without any error. Thus

due to the existing HP software, the problem comes about is discussed by an example given below.

Let consider SUBO3 and SUB10 are two of the sixteen subroutines with SUBO3 having a higher priority than SUB10, written, only for printing messages on TTY. Also, assume that the priority table is set up in such a way that SUB10 is interruptable and will continue from the interrupted address whenever it is serviced. When the computer will be interrupted for loading the request word for the SUBO3 while printing the message for SUB10 is in progress, the transfer will be made to the beginning of the SUBO3 due to SUB10 being an interruptable subroutine. The request for printing the message for SUBO3 will be rejected by IOC (Input Output Control subroutine) because of TTY driver being busy for printing the message for the SUB10. Thus a loop will be created in SUBO3 for IOC request until the printing of the message for SUB10 is finished. When printing the message for SUB10 is completed, the request from SUBO3 will be granted and it will be serviced completely by the priority handler. Then the priority handler will transfer the control again back to the address of SUB10, from where it was interrupted for loading the request word for SUBO3, eventhough the SUB10 is finished. To solve this problem, the interrupt from 10B was inhibited while an input/output operation was being executed. This is done by calling CL10C and ST10C before and after any I/O operation. The CL10C and ST10C are written for this purpose which are discussed individually one after another.

CL1OC

Refer to the Appendix A for the detail listing of the CL1OC subroutine. Upon entry to the subroutine interrupt system is turned off and then the control flip-flop on channel 10B is cleared, (to which Digital Input is connected) so that the computer can not be interrupted for loading the request word even though the flag is set. The return is made back to the calling program after enabling the interrupt system.

ST1OC

Refer to Figure 5.8 for the simplified flow chart of the subroutine ST1OC. The complete listing is given in the Appendix A. This subroutine is called after the FORTRAN WRITE or READ statement or any time the IOC is called for the input/output operation is called. In the beginning the wait on system busy loop is created. When system is free, interrupt system is turned off and the return address is made indirect. If the flag on s.c. = 10B is set, the return address is stored in the entry II.77 after starting the device on channel 10B, and jump to II.77+1 is made to load and process the request word. If the flag on sc=10B is not set, the device on channel 10B (Digital Input) is started and the interrupt system is turned on. After this the normal return is made to the calling program.

The next change was to introduce the B.O.S. library subroutine tape which was available from the HP compony. This subroutine tape has almost all the arithmetic subroutines SIN, COS, ABS etc.) calling sequences. When any of the subroutine is called, the corresponding sequences in B.O.S. library replaces

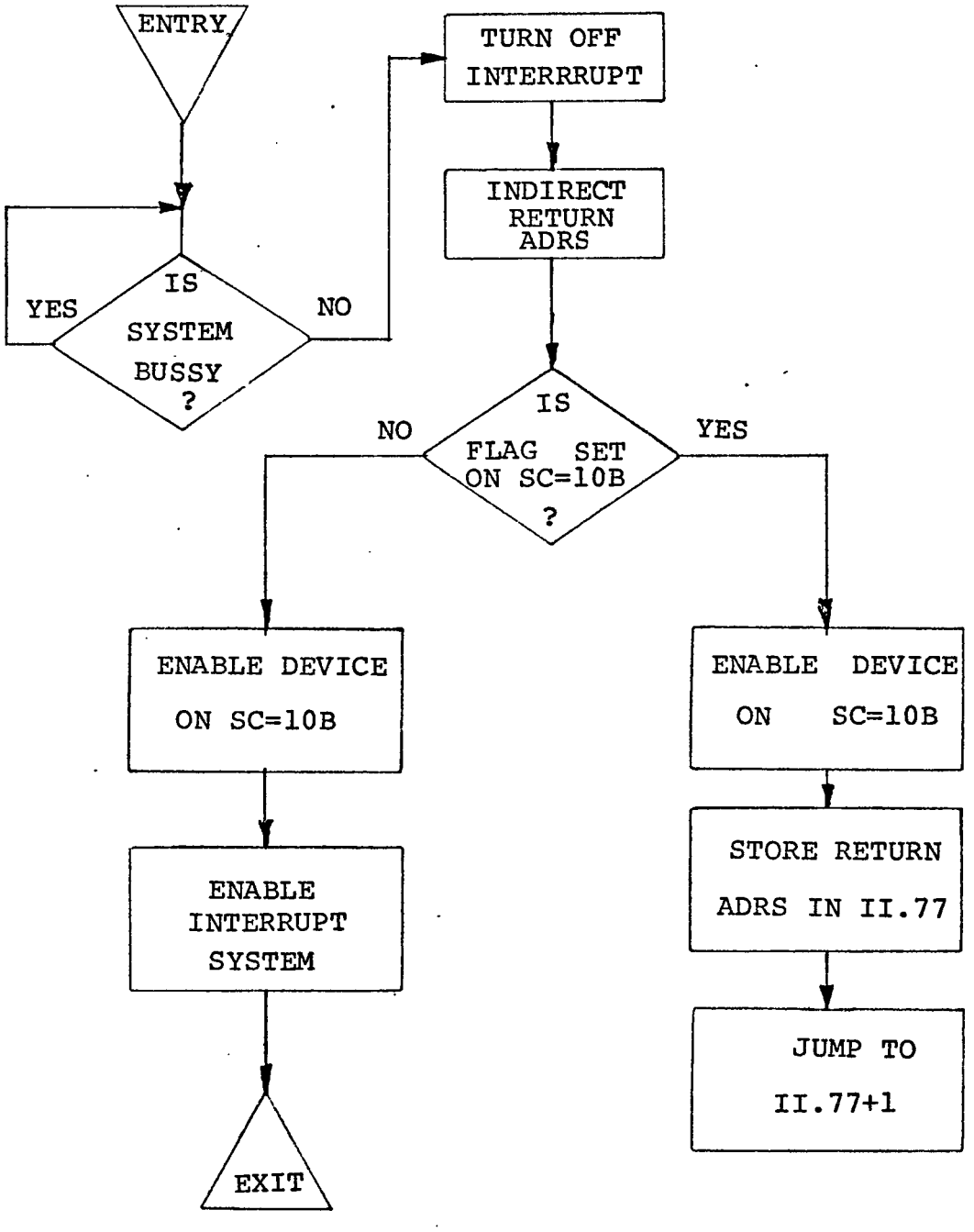


Figure 5.8 ST10C

that subroutine call. The calling sequences in B.O.S. library turns off the interrupt system, then it calls the arithmetic subroutine. When the transfer is returned back to the B.O.S. subroutine, the interrupt system is turned on. Thus the purpose of this library tape is to inhibit interrupting any of the arithmetic subroutine once it is started. The error could result if the B.O.S. library tape is not used and if a subroutine is in execution, is interrupted, is called by another program, and then re-entered again to the point from where it was interrrupted.

The third change of the subroutine .ENTR of the B.O.S. library was required due to the fact that the interrrupt should not be on, after any time the entry is made to II.77 and until the final decision of running the subroutine is made. The two subroutines, INHAN and LPRTBare written in FORTRAN language. Whenever FORTRAN subroutine is written, the compiler generates a calling sequence for subroutine.ENTR to obtain the direct address of the arguments, if any exist, and to set the correct return address. Since the sequence for subroutine .ENTR in the B.O.S. library disables the interrupt when entered and enables the interrupt at the end before return is made, the interrupt system remains on after the .ENTR in the above two subroutines (INHAN and LPRTB) is serviced. This violates the requirement of not turning the interrupt system on, whenever the decision of running the subroutine is in process as mentioned above. If the interrupt system is turned on, the error could result if, the computer is interrupted again for the new request word while the previous request word is not processed completely. For this

reason, the .ENTR subroutine in the B.O.S. library is replaced by new .ENTR subroutine which has NOP (no operation) instruction in place of STF 00 (turn on interrupt) instruction at the end. Thus new .ENTR subroutine does not turn on the interrupt system when finished so that the program is not interruptable any time the request word is being processed, and until the decision is made.

The disadvantage of solving the problem this way is the inability to interrupt a write sequence or arithmetic subroutines. A higher priority subroutine has to wait in a queue if there is a long write statement in the lower priority subroutine which is being executed. Same way if there is a long arithmetic subroutine in execution, the higher priority subroutine can not be executed until the execution of the arithmetic subroutine is finished. In order to minimize this effect, the long write statements should be split into a series of smaller write statements.

BIBLIOGRAPHY

[1] SDS Technical Manual, Perforated Tape Reader, Coupler. Models 9130, 9230, 9330.

[2] SDS Technical Manuals, Tape Perforator, Tally 1521, Instruction Manual.

[3] A Pocket Guide to Interfacing the HP 2100 Computer.

[4] SDS Technical Manual, KSR Keyboard Printer and Tape Punch Control Couplers.

[5] SDS Technical Manual, Keyboard Printer and Punch, Logic Layouts.

[6] A Pocket Guide to the 2100 Computer.

APPENDIX A

PRIORITY HANDLER LISTING

APPENDIX A

The detail listing of Priority Handler is given in the same order as it is discussed in Chapter 5. Detail listing includes the symbol table and then the assembly listing of all the assembly language subroutines. It also includes the listing of the FORTRAN programs.

The Digital Input Device was chosen as an input device, to load the request word for running the service subroutines, which was plugged into the highest priority slot of the HP computer (SC=10B).

```
FTN,B
      PRØGRAM LDPRT
C
C          ***  PRØGRAM  LDPRT   ***
C THIS IS THE PRØGRAM TØ LØAD CØRRECT PRIØRITY TABLE, IPRTB(J),    *
C ARRAY 'IFINS(J)', AND ARRAY 'IRSTR(J)'.                           *
C THE TYPICAL ENTRY FØR LØADING THE PRIØRITY TABLE,AFTER THE        *
C CØMPUTER PRINTS, "** LØAD PRIØRITY TABLE **" IS,                  *
C                  " +/-NN,+/-YY "                                  *
C WHERE,    NN-THE INCREASING ØRDER ØF THE PRIØRITY TABLE,          *
C           STARTING FRØM 1 TØ 16.                                  *
C           YY-THE SUBRØUTINE NUMBER IN 'SUBYY', SUBYY IS THE       *
C            NAME ØF ØNE ØF THE SIXTEEN SUBRØUTINES.                *
C           SAY, IF NN=1, AND YY=15, THEN SUB15 HAS HIGHTEST        *
C           PRIØRITY.                                               *
C THE CØMMØN ARRAYS ARE,                                            *
C                                                                   *
C  IQTB(J)- QUEUE TABLE ENTRY FØR SUBYY, J=YY.                      *
C  J=1,16   =0, SUBYY DØES NØT NEED SERVICE                         *
C           =1,  "    NEEDS SERVICE                                 *
C  IQTEM(J)- TEMPØRARY QUEUE TABLE FØR 'SUBYY', LØADED FRØM DIGI-  *
C  J=1,16    TAL INPUT DEVICE.                                      *
C           =0., DØES NØT NEEDS SERVICE.                            *
C           =1,  NEEDS SERVICE.                                     *
C  ISVTM(J)- TEMPØRARY STØRAGE AREA FØR WØRKING REGISTER A,B,E,Ø,   *
C  J=1,4     AND RETURN ADDRESS WHEN SUBRØUTINE IS INTERUPTED.      *
C  INØW    - THE PRESENT SUBRØUTINE NUMBER IN SRRVICE, ANY FRØM     *
C            1 TØ 16.                                               *
C  IFINS(J) =+VE, IF ENTRY IS '+NN,+/-YY' FØR SUBYY, AND J=YY       *
C  J=1,16   =-VE, "    "    " '-NN,+/-YY'  "    "      "   "        *
C            IF +VE, THEN SUBYY IS FINISHED IMMEDIATELY WHEN INTER*
C            PTED.                                                  *
C            IF -VE, THEN SUBYY IS NØT FINISHED IMMEDIATELY WHEN    *
C            INTERUPTED.                                            *
C  IREG(J)  -STØRAGE AREA FØR A,B,Ø,E REGISTERS AND RETURN ADDRES *
C  J=1,16   FØR 16 SUBRØUTINE, FØUR FØR EACH SUBYY.                 *
C  IRSTR(J) -+VE,IF ENTRY IS '+/-NN,+YY' FØR SUBYY, J=YY            *
C  J=1,16   =-VE, "    "    "  '+/-NN,-YY' "     "     "            *
C            IF +VE,THEN WHENEVER SUBYY IS SERVED, IT SHØULD BE     *
C            STARTED FRØM THE PØINT, WHERE IT WAS INTERUPTED.       *
C            IF -VE, THEN SUBYY SHØULD BE STARTED FRØM THE BEGINING
C  IPRTB(J)  - THE PRIØRITY TABLE ENTRY, STARTING FRØM HIGHEST.     *
C  J=1,16    =YY, IN THE ENTRY '+/-NN,+/-YY', AND J=NN.             *
C  ILØAD(J)  -TØ RESTØRE A,B,Ø,E REGISTERS AND RETURN ADDRESS FØR *
C  J=1,4      THE INTERUPTED SUBRØUTINE WHEN IT IS SERVED.          *
C  ICHAK(J)  =-VE,IF SUBYY YY=J WAS ASKED FØR THE SERVICE AGAIN     *
C  J=1,16     WHILE PREVIØUS REQUEST WAS NØT FINISHED.              *
C            SUBYY IS PRINTED AS DØUBLY REQUESTED SUBRØUTINE, IF    *
C            ICHAK(J) =-VE, AND CØMPUTER IS FREE.                   *
C  ICHAK(17) =+VE, NØNE SUBRØUTINES WERE DØUBLY ASKED.              *
C            =-VE, SØME SUBRØUTINES WERE DØUBLY ASKED.              *
C  INUMM(J)  =NN,THE ØRDER ØF THE PRIØRITY TABLE IN '+/-NN,+/-YY' *
C  J=1,16    AND J=NN. THIS ARRAY ELEMENT  ARE +VE IN THE BEGINING*
C            WHEN SUBYY IS SERVED, THEN INUMM(YY) IS MADE -VE TØ    *
C            INDICTE THAT SUBYY WAS SERVED. WHEN SUBYY IS SERVED *
C            CØMPLETELY, IT IS MADE +VE.                            *
```

```
      CØMMØN IQTB(16),IQTEM(16),ISVTM(4),INØW,IFINS(16),
     CIREG(64),IRSTR(16),IPRTB(16),ILØAD(4),ICHAK(17),
     CINUMM(16)
C IF ERRØR IS MADE IN LØADING TABLE, IERØR IS INCREASED FØR EACH  *
C ERRØR.                                                          *
 5    IERØR=0
C CALL 'INITL' TØ INITIATE ALL CØMMØN ARRAYS.                     *
 905  CALL INITL
 906  WRITE(2,210)
C READ ENTRY FØR PRIØRITY TABLE ØNE BY ØNE.
 907  DØ 10 K=1,16
 908  READ(1,*) INUMM(K),IPRTB(K)
 10   CØNTINUE
C CHECK WEATHER NN IS IN THE INCREASING ØRDER STARTING FRØM 1 TØ 16
C ØR NØT, IF IT IS NØT, ERRØR IS GIVEN.
C DEPENDING UPØN THE ENTRY +/-NN,+/-YY, THE ARRAYS IFINS(J),
C IRSTR(J) ARE LØADED, AND ELEMENTS ØF INUMM(J),IPRTB(J) ARE MADE
C PØSSITIVE IF THEY ARE NEGATIVE.
 50   DØ 40 J=1,16
 19   NEGNM=0
 20   IF( INUMM(J) )21,909
 21   NEGNM=-5
 23   INUMM(J) =-INUMM(J)
 909  IF( INUMM(J)-J)51,52,51
C WRITE ERRØR-01, FØR NN NØT IN THE INCREASING ØRDER, WITH
C THE ENTRY LINE IN WHICH ERRØR HAS ØCCURED.
 51   IE=01
 910  WRITE(2,240)J,IE
 911  IERØR= IERØR+1
C JUMP HERE IF NN IS IN INCREASING ØRDER, AND PRØCESS IPRTB(J)
C DEPENDING UPØN, -VE,0 ØR +VE.
 52   IF(IPRTB(J))60,70,80
C JUMP HERE IF IPRTB(J)<0,WRITE ERRØR-02 IF IPRTB(J)<-16,
C WITH THE ENTRY LINE.
 60   IPRTB(J) =-IPRTB(J)
 912  IF(IPRTB(J)-16)61,61,30
 61   M=IPRTB(J)
      IRSTR(M)=-5
      GØ TØ 82
C JUMP HERE IF IPRTB(J)=0,WRITE ENTRY LINE WITH THE ERRØR-03
C FØR YY=0, BEING INVALID SUBRØUTINE NUMBER.
 70   IE=03
 914  WRITE(2,240)J,IE
      GØ TØ 81
C JUMP HERE IF IPRTB(J)>0,WRITE ENTRY LINE WITH THE ERRØR-02
C IF IPRTB(J)>16.
 80   IF(IPRTB(J)-16)82,82,30
 82   IF(NEGNM)88,40
 88   M=IPRTB(J)
      IFINS(M)=-5
      GØ TØ 40
 30   IE=02
 916  WRITE(2,240)J,IE
 81   IERØR= IERØR+1
 40   CØNTINUE
```

```
C CHECK FØR YY BEING DUPLICATED.
C GIVE ERRØRR-04, IF DUPLICATE SUBRØUTINE NUMBER YY IS FØUND,
C WITH ENTRY LINE  NUMBER FØR WRØNG ENTRY.
 917   DØ 42 K=1,15
 918   L=K+1
 919   DØ 41 LL=L,16
 83    IF(IPRTB(LL)-IPRTB(K))41,84,41
 41    CØNTINUE
       GØ TØ 42
 84    IERØR=IERØR+1
 920   IE=04
       J=LL
 921   WRITE(2,240)J,IE
 42    CØNTINUE
C IF IERØR>0,THEN WRITE NUMBER ØF ERRØRS AND GØ TØ 5, TØ START AGAIN.
 922   IF(IERØR)90,91,90
 90    WRITE(2,260)IERØR
       GØ TØ 5
C IF IERØR=0, THEN WRITE 'NØ ERRØR ', AND CALL 'ICHNG', WHICH DØES
C NØT RETURN THE EXECUTIØN.
 91    WRITE(2,250)
       CALL  ICHNG
       GØ TØ 100
 210   FØRMAT(7(/),10X,"****  LØAD PRIØRITY TABLE  ****",/,/)
 240   FØRMAT(/,5X,I2,X,"*** ERRØR-",I2," ***")
 250   FØRMAT(/,/,10X,"***  NØ ERRØR   ***")
 260   FØRMAT(/,/,10X,I4,"ERRØRS  ****")
 100    STØP
       END
       END$
```

```
FTN,B
      SUBROUTINE INITL
      COMMON IDMI(186)
      DO 10 J=1,186
10    IDMI(J)=0
      RETURN
      END
      ENDS
```

PAGE 0001

```
0001                    ASMB,R,B,T,L
.177   R 000020
.IOC.  X 000002
ICHNG  R 000000
II.77  X 000001
**  NO ERRORS*
```

PAGE 0002 #01 **** SUBROUTINE ICHNG ****

```
0001                    ASMB,R,B,T,L
0003* THIS SUBROUTINE CHANGES THE LOCATION 10B AND 30B, TO LINK
0004* THE INTERUPT ON S.C.=10B FOR THE SUBROUTINE "II.77", AND
0005* TURNS ON THE DEVICE . THEN IT WAITES FOR FLAG. WHEN FLAG IS
0006* SET, INTERUPT SYSTEM IS TURN ON AND 'JMP *' IS EXECUTED
0007* TO MAKE LOOP.
0008*
0009*
0010* IN THE BEGINING SYSTEM REQUEST IS ASKED,(TO FINISH THE
0011* PRINTING FOR 'PRIORITY TABLE') AND IF BUSSY , WAIT ON BUSSY
0012* LOOP IS CREATED UNTIL PRINTING IS FINISHED.
0013*
0014  00000                NAM ICHNG
0015                       EXT II.77,.IOC.
0016                       ENT ICHNG
0017  00000 000000  ICHNG NOP
0018  00001 016002X       JSB .IOC.     JUMP TO SUB .IOC. FOR SYSTEM REQUEST
0019  00002 040000        OCT 40000   IF BUSSY, LOOP ON STATUS REQUEST.
0020  00003 002020        SSA
0021  00004 026001R       JMP *-3
0022  00005 062021R       LDA =B114030
0023  00006 070010        STA 10B
0024  00007 062020R       LDA .I77
0025  00010 070030        STA 30B
0026  00011 000000        NOP
0027  00012 103100        CLF 00      ****  TURN OFF INTERUPT SYSTEM  ****
0028  00013 103710        STC 10B,C         START DEVICE AND WAIT FOR FLAG
0029  00014 102310        SFS 10B
0030  00015 026014R       JMP *-1           WHEN FLAG IS SET
0031  00016 102100        STF 00            ** TURN ON INTERUPT
0032  00017 026017R       JMP *             AND INTERUPT TO II.77
0033*
0034*  ADDRESS AND CONSTANTS
0035*
0036  00020 000001X .I77 DEF II.77         ADDRESS OF II.77
      00021 114030
0037                       END
**  NO ERRORS*
```

```
PAGE 0001

0001                    ASMB,R,B,T,L
B        000001
.IQ16 R  000035
.IQTM R  000033
.NEXT R  000001
BUFFR R  000032
FINIS R  000027
II.77 R  000000
INHAN X  000001
IQTB  C  000000
IQTEM C  000020
ISVTM C  000040
LOOP  R  000016
M.01  R  000034
**  NO ERRORS*
```

```
0001                    ASMB,R,B,T,L
0003* THIS PROGRAM FIRST TURNS OFF THE INTERUPT SYSTEM
0004* AND AFTER STORING THE REGISTERS AND RETURN ENTRY POINT INTO
0005* THE TEMPORARY ARRAY "ISVTM", IT LOADS THE SUBROUTINE SERVICE
0006* REQUEST FROM THE INPUT CHANNEL INTO THE TEMPORARY Q-TABLE,
0007* IQTEM. THEN TO PROCESS THE REQUEST IT JUMPS TO SUB "INHAN"
0008*
0009* IQTEM - IS AN ARRAY OF LENGTH 16, TO LOAD TEMPORARY Q-TABLE
0010* ISVTM - IS AN ARRAY OF LENGTH 4, TO STORE ALL REGISTERS AND
0011* RETURN ENTRY POINT, IN COMMON AREA.
0012*
0013*
0014*  ENTERED BY JSB,I IN THE DEVICE INTERUPT LOCATION.
0015  00000                NAM II.77
0016                       COM IQTB(16),IQTEM(16),ISVTM(4)
0017                       ENT II.77,.NEXT
0018                       EXT INHAN
0019  00000 000000  II.77 NOP
0020  00001 103100  .NEXT CLF 00      *** TURN OFF INTERUPT ***
0021  00002 103110      CLF 10B
0022  00003 072040C     STA ISVTM       STORE A,B,E ,O
0023  00004 076041C     STB ISVTM+1     REGISTERS AND RETURN
0024  00005 001520      ERA,ALS         ENTRY ADDRESS INTO
0025  00006 102201      SOC             AN ARRAY -ISVTM-
0026  00007 002004      INA
0027  00010 072042C     STA ISVTM+2
0028  00011 062000R     LDA II.77
0029  00012 072043C     STA ISVTM+3
0030*
0031*  LOAD REQUEST FROM SC INTO IQTEM
0032*
0033  00013 102510      LIA 10B
0034  00014 072032R     STA BUFFR       STORE BITS OF REQUEST WORD
0035  00015 066033R     LDB .IQTM       'BUFFR', '1' OR '0' INTO
0036  00016 012034R LOOP AND M.01       IQTEM(I), I=1,16.
0037  00017 170001      STA B,I
0038  00020 006004      INB
0039  00021 056035R     CPB .IQ16
0040  00022 026027R     JMP FINIS
0041  00023 062032R     LDA BUFFR
0042  00024 001300      RAR
0043  00025 072032R     STA BUFFR
0044  00026 026016R     JMP LOOP
0045  00027 000000  FINIS NOP
0046  00030 016001X     JSB INHAN       JUMP TO SUB 'INHAN'.
0047  00031 000032R     DEF *+1
0048*
0049*  ADDRESS AND CONSTANTS
0050*
0051  00001          B EQU 1           B- REGISTER
0052  00032 000000  BUFFR OCT 0
0053  00033 000020C .IQTM DEF IQTEM    ADRS OF ARRAY IQTEM(I).
0054  00034 000001  M.01 OCT 1
0055  00035 000040C .IQ16 DEF IQTEM+16
0056                       END
**  NO ERRORS*
```

```
FTN,B
      SUBROUTINE INHAN
C             ****    SUBROUTINE  INHAN    ****
C
C THIS SUB UPDATES THE Q-TABLE, 'IQTB(J)', FROM TEMPORARY Q-TABLE
C 'IQTEM(J)'. IF SUB YY IS ASKED AGAIN FOR SERVICE, THE ERROR IS
C GIVEN WHICH WILL BE PRINTED LATER-ON,IF IT IS NOT CORRECTED.
C IF SOME SUB YY, IS INTERUPTED, AND IF IT HAS TO BE RESTARTED
C NOW, DEPENDING UPON 'IFINS(YY)', THEN SUB 'ICONT' IS CALLED.
C IF SUB YY CAN BE SERVED AFTERWARD THEN,DEPENDING UPON 'IRSTS(YY)'
C TEMPORARY SAVED REGISTERS ARE STORED INTO THE REGISTERS 'IREG(K)',
C FOR SUB YY. FINALLY IT CALLS SUBROUTINE 'LTEST'.
C
C
      COMMON IQTB(16),IQTEM(16),ISVTM(4),INOW,IFINS(16),
     CIREG(64),IRSTR(16),IPRTB(16),ILOAD(4),ICHAK(17),INUMM(16)
 20   DO 22 J=1,16
C IF IQTEM(J)=1,AND IQTB(J)=0, THEN  IQTB(J)=1 IS LOADED.
C IF IQTEM(J)=1, AND IQTB(J)=1, THEN IQTB(J) IS NOTB CHANGED,
C BUT ICHAK(J)=-VE IS LOADED FOR THE ERROR FOR THE DOUBLY
C REQUESTED SUBROUTINE.
C IF ERROR FOR DOUBLY REQUESTED SUBROUTINE OCCURES ICHAK(17) =-VE,
C IS LOADED.
 21   IF(IQTEM(J)-1)24,23,24
 23   IF(IQTB(J))25,26,25
 25   ICHAK(J)= -J
 29    ICHAK(17)=-5
      GO TO 22
 26   IQTB(J)=1
 24   ICHAK(J)=0
 22   CONTINUE
C IF INOW=0, NONE OF THE SUBROUTINE ARE IN SERVICE, SO CALL 'LTEST'.
C IF INOW>0, AND IF IFINS(INOW)=0, OR >0,THEN CALL 'ICONT'
C IF INOW>0, AND IF IFINS(INOW)=<0, AND IF IRSTR(INOW)=0
C OR >0, THEN  STORE 'ISVTM(J)'(TEMP. STORAGE AREA) INTO 'IREG(K)'
C CORROSPONDING TO SUBYY, HERE YY=INOW.
 32   IF(INOW)44,44,34
 34   IF(IFINS(INOW))35,36
 36   CALL ICONT
 35   IF(IRSTR(INOW))44,37
 37   IFRST=(INOW-1)
      IFRST=IFRST+IFRST+IFRST+IFRST
 38   ISTRT=IFRST+1
 39   IFINL=ISTRT+3
 40   DO 41 K=ISTRT,IFINL
      M=K-IFRST
 41   IREG(K)=ISVTM(M)
 44   CALL LTEST
      RETURN
      END
      END$
```

```
FTN,B
      SUBROUTINE LTEST
C
C        ****   SUBROUTINE  LTEST   ****
C THIS SUBROUTINE STARTS TESTING WEATHER 'SUBYY' NEEDS SERVICE
C OR NOT, STARTING WITH THE HIGHEST PRIORITY. IF IT NEEDS SERVICE
C (IQTB(YY)=1), AND IF INUMM(YY)> OR =Ø, THEN IT CALLS 'IONGO(M)'
C WITH M=-1. IT NEEDS SERVICE, AND INUMM(YY)<Ø(SUBYY WAS SERVED
C AND INTERUPTED), THEN IF IRSTR(YY)> OR =Ø, ARRAY 'ILOAD(J)' IS
C LOADED FROM STORAGE REGISTER AREA 'IREG(K)'(CORROSPONDING TO
C SUBYY). THEN 'IONGO(M)', WITH M=1 IS CALLED.
      COMMON IQTB(16),IQTEM(16),ISVTM(4),INOW,IFINS(16),
     CIREG(64),IRSTR(16),IPRTB(16),ILOAD(4),ICHAK(17),INUMM(16)
C START TESTING , STARTING WITH THE HIGHEST PRIORITY, FOR SERVICE
 1Ø   DO 2Ø L=1,16
 21   ITEST=IPRTB(L)
 23   IF(IQTB(ITEST))2Ø,2Ø,22
 2Ø   CONTINUE
      GO TO 5Ø
C IF INUMM(YY)<Ø,THEN SUBYY WAS SERVED AND INTERUPTED.
C IF INUMM(J)> OR =Ø, THEN SUBYY IS GETTING SERVICE FIRST TIME
 22   IF(INUMM(ITEST))222,25
 25   INUMM(ITEST) =-5
      GO TO 4Ø
 222  IF(IRSTR(ITEST))4Ø,3Ø
 3Ø   IFRST=(ITEST-1)
      IFRST=IFRST+IFRST+IFRST+IFRST
 31   ISTRT=IFRST+1
 32   IFINL=ISTRT+3
 33   DO 34 K=ISTRT,IFINL
      M=K-IFRST
 34   ILOAD(M)=IREG(K)
 35   INOW=ITEST
C CALL IONGO(M), WITH M=1, IF SUBYY HAS TO START FROM THE POINT
C WHERE IT WAS INTERUPTED.
 36   M=1
 37   GO TO 44
 4Ø   INOW=ITEST
C CALL IONGO(M),WITH M=-1,  IF SUBYY HAS TO START FROM THE BEGINING
 41   M=-1
  44  CALL IONGO(M)
 5Ø   CALL WFLAG
      RETURN
      END
      END$
```

PAGE 0001

```
0001                    ASMB,R,B,T,L
ICONT R 000000
IQTB  C 000000
IQTEM C 000020
ISVTM C 000040
**  NO ERRORS*
```

```
PAGE 0002 #01  ***   SUBROUTINE  ICONT   ***

0001                    ASMB,R,B,T,L
0003* THIS SUB RESORES WORKING REGISTERS, WHICH ARE IN TEMPORARY ARRAY
0004* 'ISVTM', TURNS ON THE INTERUPT SYSTEM, AND THEN JUMPS TO RETURN
0005* ADDRESS 'ISVTM+3'.
0006*
0007*
0008  00000              NAM ICONT
0009                     COM IQTB(16),IQTEM(16),ISVTM(4)
0010                     ENT ICONT
0011  00000 000000  ICONT NOP
0012  00001 062042C      LDA ISVTM+2       RESTORE A,B,E,O,AND
0013  00002 103101       CLO
0014  00003 000036       SLA,ELA           RETURN ADDRESS
0015  00004 102101       STF 1B
0016  00005 062040C      LDA ISVTM
0017  00006 066041C      LDB ISVTM+1
0018  00007 000000       NOP
0019  00010 102100       STF 00          ***  TURN ON INTERUPT  ***
0020  00011 126043C      JMP ISVTM+3,I     JUMP BACK TO SUB IN SERVICE
0021                     END
**  NO ERRORS*
```

```
PAGE 0001

0001                    ASMB,R,B,L,T
.ADD. R 000051
.ENTR X 000001
.SBAD R 000031
ADFNL R 000052
GETAD R 000020
IADMY C 000000
IBDMY C 000045
ILOAD C 000225
INOW  C 000044
INTON R 000026
IONGO R 000001
MADRS R 000000
SUB01 X 000002
SUB02 X 000003
SUB03 X 000004
SUB04 X 000005
SUB05 X 000006
SUB06 X 000007
SUB07 X 000010
SUB08 X 000011
SUB09 X 000012
SUB10 X 000013
SUB11 X 000014
SUB12 X 000015
SUB13 X 000016
SUB14 X 000017
SUB15 X 000020
SUB16 X 000021
**  NO ERRORS*
```

```
0001                    ASMB,R,B,L,T
0003* THIS SUB TURNS ON INTERUPT SYSTEM AND THEN JUMPS TO SUB YY, YY=INO
0004* IF M IS NEGATIVE, IT JUMPS TO THE BEGINING OF THE SUB. IF M IS
0005* POSSITIVE, IT JUMPS TO THE ENTRY, FROM WHERE IT WAS INTERUPTED
0006* AFTER RESTORING THE WORKING REGISTERS.
0007* IADMY(36), AND IBDMY(112) ARE DUMMY ARRAYS.
0008*
0009  00000                NAM IONGO
0010                       COM IADMY(36),INOW,IBDMY(112),ILOAD(4)
0011                       ENT IONGO
0012                       EXT .ENTR,SUB01,SUB02,SUB03,SUB04,SUB05,SUB06,SUB
0013                       EXT SUB10,SUB11,SUD12,SUB13,SUB14,SUB15,SUB16
0014  00000 000000  MADRS BSS 1
0015  00001 000000  IONGO NOP
0016  00002 016001X       JSB .ENTR
0017  00003 000000R       DEF MADRS
0018  00004 162000R       LDA MADRS,I        LOAD VALUE OF M
0019  00005 002020        SSA               IF M IS POSSITIVE, THEN JUMP TO E
0020  00006 026020R       JMP GETAD         POINT, FROM WHERE IT WAS INTERUPT
0021  00007 062227C       LDA ILOAD+2       AFTER RESTORING THE REGISTERS.
0022  00010 103101        CLO
0023  00011 000036        SLA,ELA
0024  00012 102101        STF 1B
0025  00013 062225C       LDA ILOAD
0026  00014 066226C       LDB ILOAD+1
0027  00015 000000        NOP
0028  00016 102100        STF 00            **  INTERUPT ON  **
0029  00017 126230C       JMP ILOAD+3,I
0030  00020 062051R GETAD LDA .ADD.         FOR M NEGATIVE, (.ADD.)+(INOW) GI
0031  00021 042044C       ADA INOW          THE ADRS OF THE SUB YY, YY=INOW I
0032  00022 072052R       STA ADFNL         GET THIS ADDRESS AND STORE IN 'AD
0033  00023 162052R       LDA ADFNL,I
0034  00024 072052R       STA ADFNL
0035  00025 000000        NOP
0036  00026 102100  INTON STF 00            **  TURN ON INTERUPT  **
0037  00027 116052R       JSB ADFNL,I
0038  00030 000031R       DEF *+1
0039  00031 000000  .SBAD BSS 16            RESERVE 16 LOCATION FOR .SBAD
0040  00031                ORG .SBAD        STORE ADDRESS OF SUB YY, YY=1 TO 1
0041  00031 000002X       DEF SUB01        INTO ARRAY '.SBAD'
0042  00032 000003X       DEF SUB02
0043  00033 000004X       DEF SUB03
0044  00034 000005X       DEF SUB04
0045  00035 000006X       DEF SUB05
0046  00036 000007X       DEF SUB06
0047  00037 000010X       DEF SUB07
0048  00040 000011X       DEF SUB08
0049  00041 000012X       DEF SUB09
0050  00042 000013X       DEF SUB10
0051  00043 000014X       DEF SUB11
0052  00044 000015X       DEF SUB12
0053  00045 000016X       DEF SUB13
0054  00046 000017X       DEF SUB14
0055  00047 000020X       DEF SUB15
0056  00050 000021X       DEF SUB16
0057  00051                ORR                RESETS ORIGIN AT .SBAD+16
0058*
```

```
PAGE 0003 #01   ****   SUBROUTINE  IONGO   ****

0059* ADDRESS AND CONSTANTS
0060*
0061   00051 000030R .ADD. DEF .SBAD-1     ADRS OF '.SBAD -1'
0062   00052 000000  ADFNL OCT 0          FINAL ADDRESS
0063                     END
**  NO ERRORS*
```

```
PAGE 0001

0001                    ASMB,R,B,L,T
.WRIT R 000007
IADMY C 000000
IBDMY C 000045
ICHAK C 000231
INOFF R 000012
INOW  C 000044
WEROR X 000001
WFLAG R 000000
**  NO ERRORS*
```

```
PAGE 0002 #01  ***   SUBROUTINE  WFLAG   ***

0001                    ASMB,R,B,L,T
0003* THIS SUB FIRST WRITES ERRORS FOR THE DOUBLY REQUESTED SUBROUTINS
0004* IF THERE EXIST, AND WHEN FINISHED WAITS FOR FLAG.
0005* INOW IS CLEARED FIRST, AND WHEN ALL ERRORS ARE
0006* PRINTED IN SUB 'WEROR',INTERUPT IS TURNED OFF AND IT WAITS FOR
0007* FLAG,. WHEN FLAG IS SET FOR SERVICE , INERUPT IS TURNED ON ,
0008* AND FLAG IS SET ON SC=10B TO JUMP TO SUBROUTINE 'II.77'
0009* ICHAK(17) IS CLEARED, WHEN PRINTING OF ERRORS IS FINISHED IN WEROR
0010*
0011  00000              NAM WFLAG
0012                     EXT WEROR
0013                     ENT WFLAG
0014                     COM IADMY(36),INOW,IBDMY(116),ICHAK(17)
0015  00000 000000  WFLAG NOP
0016  00001 002400       CLA
0017  00002 072044C      STA INOW        CLEAR 'INOW'
0018  00003 062251C      LDA ICHAK+16    IF 'ICHAK(17)' IS NEGATIVE,
0019  00004 002020       SSA
0020  00005 026007R      JMP .WRIT       WRITE ERRORS. WHEN PRINTING
0021  00006 026012R      JMP INOFF       IS FINISHED , WAIT FOR FLAG
0022  00007 102100  .WRIT STF 00         ** INTERUPT ON  **
0023  00010 016001X      JSB WEROR       JUMP TO SUB 'WEROR', TO WRITE
0024  00011 000012R      DEF *+1
0025  00012 103100  INOFF CLF 00         ERRORS. WHEN COMES BACK
0026  00013 002400       CLA             * TURN OFF INTERUPT
0027  00014 072251C      STA ICHAK+16      ICHAK(17)=0
0028  00015 000000       NOP
0029  00016 102310       SFS 10B
0030  00017 026016R      JMP *-1         WAIT FOR FLAG
0031  00020 103710       STC 10B,C
0032  00021 102100       STF  00          *: INTERUPT ON  **
0033  00022 102110       STF 10B
0034  00023 026023R      JMP *            SET FLAG ON SC=10
0035                     END
**  NO ERRORS*
```

```
PAGE 0001

0001                    ASMB,R,B,T,L
M1      R 000020
.ADRS   R 000021
ADFNL   R 000022
ADINM   R 000023
CLRQ    R 000000
FADIN   R 000024
IADMY   C 000045
INOW    C 000044
INUMM   C 000252
IQTB    C 000000
IQTEM   C 000020
ISVTM   C 000040
LTEST   X 000001
**  NO ERRORS*
```

```
PAGE 0002 #01  ****   SUBROUTINE  CLRQ   ****

0001                    ASMB,R,B,T,L
0003* THIS SUB CLEARS THE Q-TABLE ENTRY, IQTB(INOW), INUMM(INOW), CORROS
0004* TO 'INOW', AFTER SUB IS SERVED COMPLETELY. THEN IT JUMPS TO SUB
0005* 'LTEST'.
0006*
0007  00000                NAM CLRQ
0008                       COM IQTB(16),IQTEM(16),ISVTM(4),INOW,IADMY(133),I
0009                       ENT CLRQ
0010                       EXT LTEST
0011  00000 000000   CLRQ NOP
0012  00001 103100         CLF 00          **   TURN OFF INTERUPT   **
0013  00002 062021R        LDA .ADRS        GET ADRS OF 'IQTB(J)' AND
0014  00003 042044C        ADA INOW        CLEAR IT.
0015  00004 042020R        ADA M1
0016  00005 072022R        STA ADFNL
0017  00006 062023R        LDA ADINM        CLEAR INUMM(INOW), THEN
0018  00007 042044C        ADA INOW         JUMP TO SUB 'LTEST'.
0019  00010 042020R        ADA M1
0020  00011 072024R        STA FADIN
0021  00012 002400         CLA
0022  00013 172022R        STA ADFNL,I
0023  00014 172024R        STA FADIN,I
0024  00015 000000         NOP
0025  00016 016001X        JSB LTEST
0026  00017 000020R        DEF *+1
0027*
0028*  CONSTANTS AND ADDRESS
0029*
0030  00020 177777   M1 DEC -1
0031  00021 000000C .ADRS DEF IQTB      ADRS OF 'IQTB'
0032  00022 000000  ADFNL OCT 0         FINAL ADRS OF 'IQTB(INOW)'.
0033  00023 000252C ADINM DEF INUMM     ADRS OF 'INUMM'.
0034  00024 000000  FADIN OCT 0         FINAL ADRS OF 'INUMM(INOW)'.
0035                       END
**  NO ERRORS*
```

```
FTN,B
      SUBROUTINE WEROR
C THIS IS A SUBROUTINE TO WRITE ERRORS FOR DOUBLY REQUESTED
C SUBROUTINES.
      COMMON IDMMY(153),ICHAK(17)
      CALL CL10C
 1    WRITE(2,210)
C
C IF ICHAK(J) <0, WRITE CORROSPONDING SUB IN ERROR AND THEN MAKE
C ICHAK(J)=0.
C RETURN IS MADE WHEN FINISHED.
C
 20   DO 30 J=1,16
 21   IF(ICHAK(J))25,30
 25   WRITE(2,220) J
 26   ICHAK(J)=0
 30   CONTINUE
      CALL ST10C
 2    GO TO 230
 210   FORMAT(/,/,/,"**  THE FOLLOWING SUBROUTINES WERE ASKED,
     C  AGAIN FOR SERVICE **",/,/,/)
 220  FORMAT(5X,"*SUB",I2,/)
 230  RETURN
      END
      END$
```

```
PAGE 0001

0001                    ASMB,R,B,L,T
CL10C R 000000
M100  R 000010
**  NO ERRORS*
```

```
PAGE 0002 #01  ****   SUBROUTINE CL10C

0001                    ASMB,R,B,L,T
0003* THIS SUBROUTINE FIRST TURNS OFF THE INTERUPT, AND
0004* THEN CLEARS THE CONTROL, CLC 10B, FLIP-FLOP OF CHANNEL
0005* 10B. RETURN IS MADE AFTER ENABLING THE INTERUPT SYSTEM.
0006*
0007  00000                NAM CL10C
0008                       ENT CL10C
0009  00000 000000  CL10C NOP
0010  00001 103100        CLF 00     ** INTERUPT OFF  **
0011  00002 106710        CLC 10B       CLEAR CONTROL ON 10B
0012  00003 062000R       LDA CL10C
0013  00004 032010R       IOR M100      INDIRECT RETURN ADDRESS
0014  00005 072000R       STA CL10C
0015  00006 102100        STF 00       * INTERUPT ON *
0016  00007 126000R       JMP CL10C,I
0017  00010 100000  M100 OCT 100000
0018                       END
**  NO ERRORS*
```

```
PAGE 0001

0001                    ASMB,R,B,L,T
.IOC. X 000001
.NEXT X 000003
.STC  R 000016
II.77 X 000002
M100  R 000021
ST10C R 000000
STLUP R 000001
**  NO ERRORS*
```

PAGE 0002 #01 ** SUBROUTINE ST10C ***

```
0001                    ASMB,R,B,L,T
0003* THIS SUBROUTINE CHECKES THE STATUS OF THE SYSTEM.
0004* IF ANY OF THE DIVICE IS BUSSY, A LOOP ON BUSSY IS CREATED
0005* WHEN COMES OUT OF LOOP , IF FLAG IS SET ON 10B, THEN
0006* SETS CONTROL ON 10B, STORES RETURN ADDRESS INTO II.77, AND
0007* JUMP TO II.77+1, OTHERWISE SETS CONTROL WIHOUT
0008* SETTING THE FLAG, AND RETURN IS MADE WITH THE INTERUPT ON.
0009  00000                NAM ST10C
0010                       ENT ST10C
0011                       EXT .IOC.,II.77,.NEXT
0012  00000 000000  ST10C NOP
0013  00001 016001X STLUP JSB .IOC.   JUMP TO SUB .IOC., FOR SYSTEM STATU
0014  00002 040000       OCT 40000   IF BUSSY, LOOP ON STATUS REQUEST,
0015  00003 002020       SSA         UNTIL OPERATION IS COMPLETE.
0016  00004 026001R      JMP STLUP
0017  00005 103100       CLF 00      INTERUPT OFF
0018  00006 062000R      LDA ST10C   INDIRECT RETURN ADDRESS
0019  00007 032021R      IOR M100
0020  00010 072000R      STA ST10C
0021  00011 102310       SFS 10B
0022  00012 026016R      JMP .STC    IF FLAG ON 10B, IS SET
0023  00013 103710       STC 10B,C   SET CONTROL WITH FLAG CLEAR
0024  00014 072002X      STA II.77
0025  00015 026003X      JMP .NEXT   JUMP TO II.77+1
0026  00016 103710  .STC STC 10B,C
0027  00017 102100       STF 00      INTERUPT ON
0028  00020 126000R      JMP ST10C,I SAY BYE
0029  00021 100000  M100 OCT 100000
0030                       END
** NO ERRORS*
```

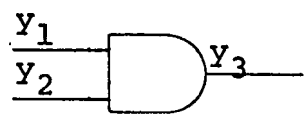
APPENDIX B

LOGIC SYMBOLS

APPENDIX B

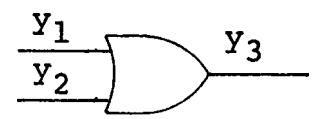
The logic symbols of the SDS Coupler system and HP interface are as shown below.

1) SDS Coupler Logic Symbols:



XXT
or XXS

AND gate



XXT
or XXS

OR gate

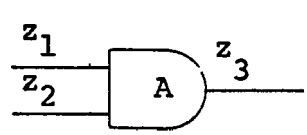
where,

y_n - The pin number of the particular location of the chassis.

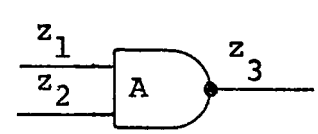
XXT - The card location XX on chassis T.

XXS - The card location XX on chassis S.

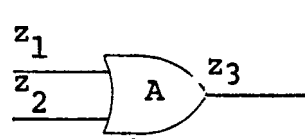
2) HP Interface Logic Symbols:



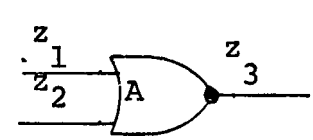
AND-gate



NAND-gate



OR-gate



NOR-gate

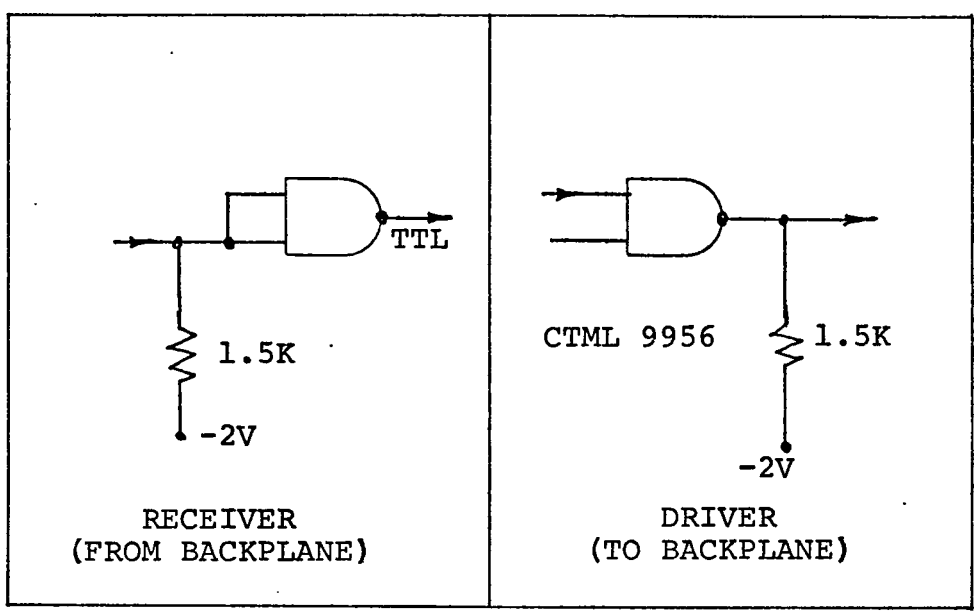
zn-pin number of an IC chip A.

APPENDIX C

BACKPLANE DRIVER/RECEIVERS

APPENDIX C

The I/O signals coming from the HP busses are CTL-driven, typically by a Fairchild type 9956. The interface capability of a 2100 Computer would be reduced to half if there are two loads per signal. The 9956 pulls positive to +2.5V. So a resistor pull to -2V is required when receiving into TTL. A resistor pull to -2V is also required for the driver to increase the speed. The receiver and driver are as shown below.



APPENDIX D

LAYOUT AND CONNECTION TABLE FOR THE HIGH SPEED READER

APPENDIX D

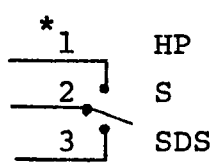
The layout of the high speed reader, interface card, the component list and the wiring connection are given here.

Table D.1 Wiring Connection for the Reader

From HP Reader Connection Pin#	Color Code Wire Base-Tracer	To SDS 31S Card
12	Brown-Yellow	31S-37
13	Orange-Red	31S-36
14	Brown-Violet	31S-35
15	Brown-Red	31S-33
16	Brown-White	31S-38
17	Orange-White	31S-32
18	Green-White	31S-31
19	Red-Gray	31S-30
20	Gray-Red	
21	Blue-White	31S-45
22	Red-Green	
23	Gray-Yellow	
24	Purple-Orange	31S-44
P	Yellow-Green	31S-34
R	Violet-Brown	31S-43
S	White-Brown	31S-43
T	Black-Brown	
U	Black-Blue	
V	White-Orange	
W	White-Green	
X	White-Blue	
Y	Green-Red	
Z	Yellow-Brown	
AA	Yellow-Gray	
BB	Blue-Purple	

Table D.2 Wiring Connection for the Reader

From SDS Chassis (Back Side)	Color Code Wire Base-Tracer	To SDS Chassis (Back Side)
31S-30	Green-Brown	45S-8
31S-31	Orange-Gray	45S-7
31S-32	Blue-Green	45S-6
31S-33	Orange-White	45S-4
31S-34	Green-White	45S-9
31S-35	Blue-Gray	45S-3
31S-36	Blue-Brown	45S-2
31S-37	Orange-Brown	45S-1
31S-38	Blue-Orange	45S-5
31S-43	Orange-Gray	45S-26
39S-31	Blue-Brown	45S-28
42S-43	Green-Brown	45S-30
Card P4-26	Cable	Card P19-26
Card P4-28	Cable	Card P19-28
Card P4-30	Cable	Card P19-30
Back Side P19-26	White-Green	1*
Back Side P19-28	Brown	2*
Back Side P19-30	Gray	3*

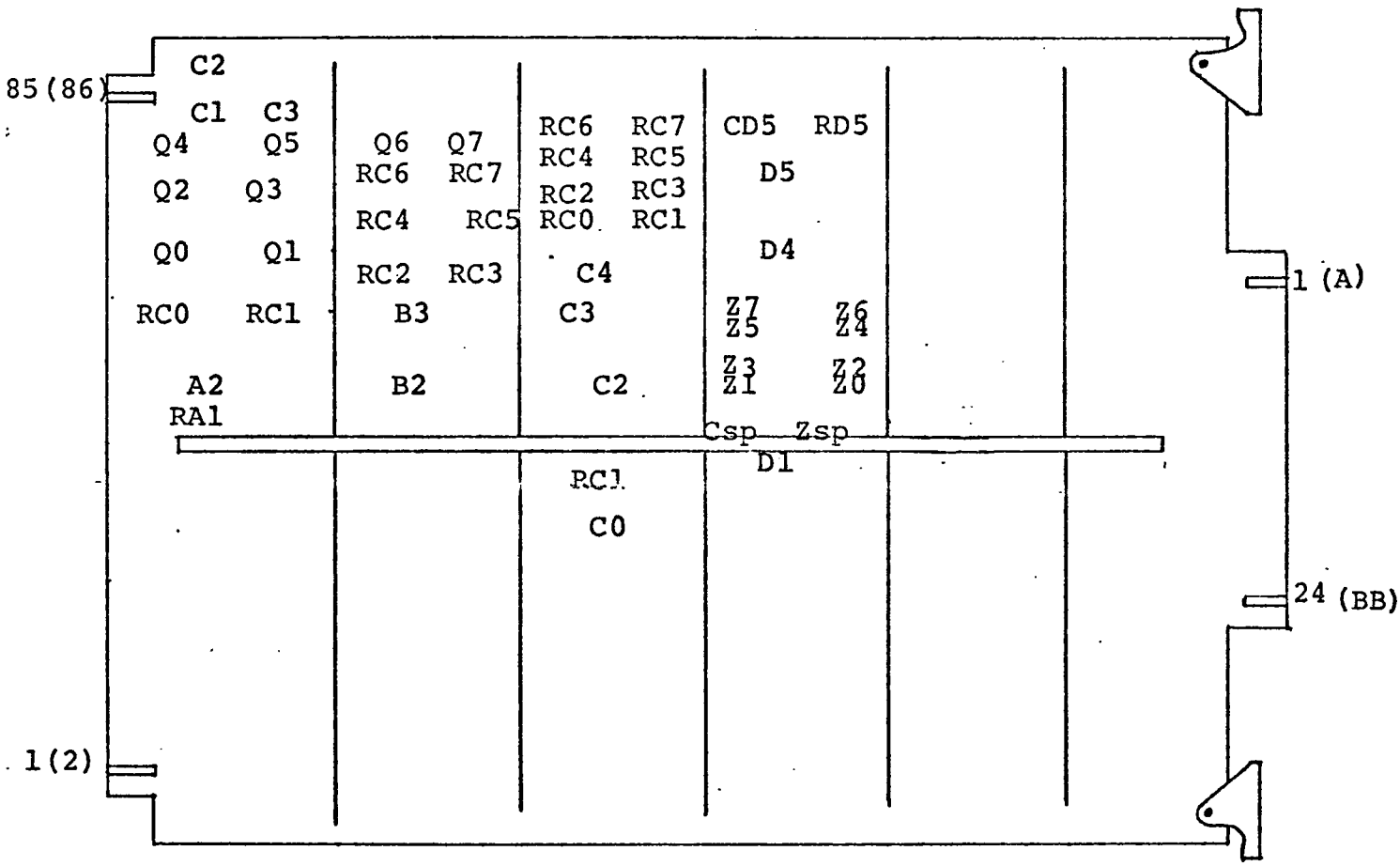


(one of the connection on the rotory switch mounted on SDS front panel)

Note: 45S Card (P4) is connected to cable plug module P19 Card.

COMPONENTS

1) A2-MC7400
2) B2,B3-SN7402N
3) C2,C3-SN7475N
4) CO-SN7403N
5) D4-MC7410P
6) D5-SN74121
7) Z_0-27, Z_{sp}-Zenner Diode 1N746A, +3V
8) Q_0-Q_7-2N3643
9) R_{C0}-R_{C7},R_{C1}-1.3K
10) R_{C0}-R_{C7}-50Ω
11) RA1-1.5K
12) RD5-10K
13) C1-C4-2.2μF
14) CD5-100PF
15) C_{sp}-1000PF



Layout of the Reader Interface Card

APPENDIX E

LAYOUT AND CONNECTION FOR THE HIGH SPEED PUNCH

APPENDIX E

This appendix is supplied for the wiring connection on rotary switch and the wiring from the HP Punch Interface card to SDS chassis. The layout of the punch interface card with the components list is also given.

Table E.1 Wiring for the Rotary Switch

From Rotary Switch	Color Code Wire Base-Tracer	To SDS 27S-Card, Pin#
S1	Yellow-Brown	13
	Green-Yellow	14
	Blue-Black	15
S2	Blue-Yellow	16
	Orange-Yellow	17
	Yellow-Gray	18
S3	Brown-Yellow	19
	Yellow-Green	20
	Gray-Black	21
S4	Yellow-Blue	22
	Gray-Yellow	23
	Black-Orange	24
S5	Yellow-Orange	25
	Black-Gray	28
	Green-Black	27
S6	Black-Brown	26
	Orange-Black	29
	Black-Blue	30
S7	Black-Green	31
	Brown-Black	32
	White-Green	33
S8	Red-Blue	34
	Orange-Red	35
	White-Gray	36
S9	Orange-White	37
	White-Blue	38
	White-Brown	39
S10	Blue-White	40
	Green-Red	41
	White-Orange	42
S11	(OPEN)	
	Green-White	13
	Red-Green	14
S12	Brown-Red	15
	Red-Gray	16
	Brown-White	17

From Rotary Switch	Color Code Wire Base-Tracer	To SDS 27S-Card, Pin#
S13	Gray-Red	18
	Red-Brown	19
	(OPEN)	

Note: When switch is in SDS position, all 1st and 2nd wires of all switches are connected, say pin 13 and 14 for S1. When it is in HP position, all 2nd and 3rd wires of all switches are connected, say pin 14 and 15 for S1.

Table E.2 Wiring from HP Punch Interface Card to SDS 31S Card

From HP Punch Connector Pin#	Color Code Wire Base-Tracer	To SDS 31S-Card, Pin#
1	Yellow-Orange	
2	Blue-Red	
3	Yellow-Blue	
4	Gray-Black	
5	Black-Green	
6	Red-Brown	13
7	Black-Brown	14
8	Red-Orange	15
9	Black-Gray	45
10	Purple-Green	17
11	Orange-Purple	18
12	Gray-Purple	19
13	Green-Black	20
14	Green-Yellow	21
15	Orange-Yellow	22
16	Gray-White	23
17	Purple-Gray	24
18	Brown-Black	25
19	Orange-Black	
20	Purple-Blue	
21	Blue-Green	
22	Red-Blue	
23	White-Gray	
24	Blue-Black	44
E	Green-Purple	

Table E.3 Wiring on SDS S and T Chassis

From SDS Chassis (Back Side)	Color Code Wire Base-Tracer	To SDS Chassis (Back Side)
27S-13	Gray-Red	40T-25
27S-14	Orange-Yellow	43T-16
27S-15	Red-Brown	31S-23
27S-16	Brown-Red	40T-4
27S-17	Yellow-Orange	43T-28
27S-18	Red-Gray	31S-22
27S-19	Yellow-Gray	40T-12
27S-20	Gray-Yellow	43T-38
27S-21	Brown-Yellow	31S-21
27S-22	Yellow-Brown	40T-18
27S-23	Blue-Yellow	42T-16
27S-24	Brown-Black	31S-20
27S-25	Black-Brown	40T-22
27S-28	Black-Blue	42T-28
27S-27	Blue-Black	31S-19
27S-26	Black-Orange	34T-43
27S-29	Orange-Black	42T-38
27S-30	Gray-White	31S-18
27S-31	Black-Gray	40T-28
27S-32	Gray-Black	44T-16
27S-33	Blue-Red	31S-24
27S-34	Red-Blue	27S-44
27S-35	Green-Yellow	29S-39
27S-36	Green-White	31S-25
27S-37	Brown-White	44T-38
27S-38	Red-Green	37T-39
27S-39	White	29S-35
27S-40	Green-Red	44T-27
27S-41	White-Gray	37T-29
27S-42	Purple-Orange	27S-31
29S-13	Orange-White	44T-26
29S-14	Red-Orange	31S-17
29S-15	Orange-Red	36T-25
29S-16	Green-Black	41T-38
29S-17	White-Gray	31S-15
29S-18	White-Orange	36T-6
29S-19	Green-Brown	42T-14
29S-32	Orange-Purple	31S-13
29S-33	White-Brown	31S-14
29S-35	White	27S-39
29S-31	Purple-Orange	27S-42
29S-39	Green-Yellow	27S-35
29S-29	Black-Green	45T-32
31S-13	Orange-Purple	29S-32
31S-14	White-Brown	29S-33
31S-15	White-Gray	29S-17

From SDS Chassis (Back Side)	Color Code Wire Base-Tracer	To SDS Chassis (Back Side)
31S-17	Red-Orange	29S-14
31S-18	Gray-White	27S-30
31S-19	Blue-Black	27S-27
31S-20	Brown-Black	27S-24
31S-21	Brown-Yellow	27S-21
31S-22	Red-Gray	27S-18
31S-23	Red-Brown	27S-15
31S-24	Blue-Red	27S-33
31S-25	Green-White	27S-36
31S-44	GROUND BUS	ON S-CHASSIS
31S-45	+8V BUS	ON S-CHASSIS

Table E.4 The Components List

Lable	Components
A1,B4,B5,D2	SN7400
B2,B3,C2	SN7403
A3,A5	SN7475N
D1	SN7420N
C4,C6,C6,E4	SN74121
D3	SN74123
CP1,CP2,CP3	1μF,15V
CP4-CP8	.1μF,10V,Ceramic
RO-R8,RA1	1.5K
R11-R18,RC1,RC2	1.2K
RC4	8.2K
RC5	2.2K
RC6	700 Ohm
RE4,RD3	27K
CC4	100Pf
CC5,CC6,CE4	10μF,Electrolyte
F2	SN7474

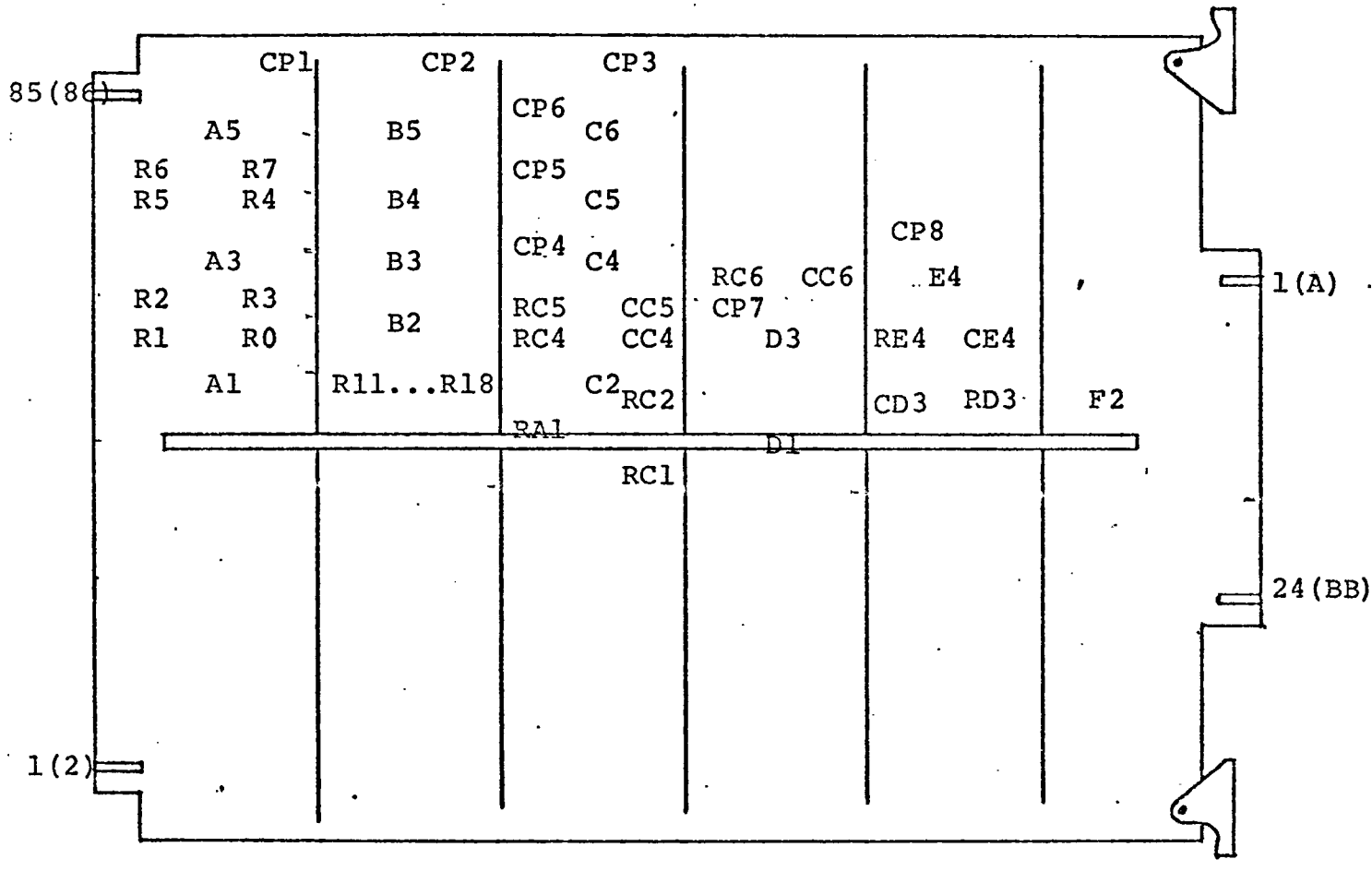


Figure E.5 The Layout of the Punch Interface Card