



Design of a microprocessor controlled data interface system  
by Dennis Ivan Smith

A thesis submitted in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE  
in Electrical Engineering  
Montana State University  
© Copyright by Dennis Ivan Smith (1975)

**Abstract:**

The subject of this thesis is the design of a microprocessor controlled data interface system. The system presented may be used as a remote data acquisition system or a type of industrial controller. Several of these systems may be connected together to create a complex data collection system.

The system requirements and specifications are relatively few. After the specifications of the data bus are given, several interface circuits are described. This is followed by a description of the controller requirements. A microprocessor was chosen and a controller designed around it. The programming techniques required by this specific system are presented as well as the operation of the entire system.

STATEMENT OF PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Montana State University, I agree that the Library shall make it freely available for inspection. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by my major professor, or, in his absence, by the Director of Libraries. It is understood that any copying or publication on this thesis for financial gain shall not be allowed without my written permission.

Signature Dennis Dean Juvik

Date May 23, 1975

DESIGN OF A MICROPROCESSOR CONTROLLED  
DATA INTERFACE SYSTEM

by

DENNIS IVAN SMITH

A thesis submitted in partial fulfillment  
of the requirements for the degree

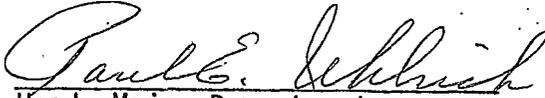
of

MASTER OF SCIENCE

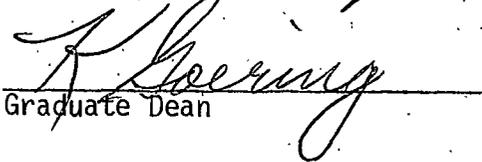
in

Electrical Engineering

Approved:

  
Head, Major Department

  
Chairman, Examining Committee

  
Graduate Dean

MONTANA STATE UNIVERSITY  
Bozeman, Montana

June, 1975

## ACKNOWLEDGMENT

The design and development of the data interface system presented in this thesis has involved several individuals. The writer would especially like to thank Mr. Dick Weaver of Western Telecomputing Corporation of Bozeman, Montana, for providing suggestions for improving the original design, and for designing several of the interface circuits. Of course, the author is especially grateful to Dr. Donald K. Weaver for his assistance and support of the project.

D. I. S.

## TABLE OF CONTENTS

Chapter	Page
VITA.....	ii
ACKNOWLEDGMENT.....	iii
TABLE OF CONTENTS.....	iv
LIST OF TABLES.....	vi
LIST OF FIGURES.....	vii
ABSTRACT.....	viii
I. INTRODUCTION.....	1
Background.....	1
Organization of Chapters.....	4
II. DESCRIPTION OF THE SYSTEM BUS.....	5
Description of the System Bus Lines.....	5
Description of the Data Transfer Operation.....	7
III. INTERFACE CIRCUITS.....	11
Counter-Timer.....	11
Memory.....	16
Serial Output.....	18
IV. SYSTEM CONTROLLER.....	20
Controller Requirements.....	20
Microprocessor Requirements.....	22
Intel 8008 Microprocessor Description.....	23
Description of the Microprocessor Controller.....	26

Chapter	Page
V. DESCRIPTION OF THE MICROPROCESSOR CONTROLLED SYSTEM.....	40
System Configuration.....	40
Program Considerations.....	40
Operation of the System.....	43
VI. CONCLUSION.....	46
REFERENCES.....	49

LIST OF TABLES

	Page
Table 1. Sample System Configuration.....	41

## LIST OF FIGURES

	Page
Figure 1. Bus Timing Diagram.....	8
Figure 2. Counter-Timer Circuit: Counter.....	12
Figure 3. Counter-Timer Circuit: Timer.....	13
Figure 4. Counter-Timer Circuit: Control.....	15
Figure 5. Controller Block Diagram.....	27
Figure 6. Controller Circuit: Input.....	32
Figure 7. Controller Circuit: CPU.....	33
Figure 8. Controller Circuit: Decoder.....	34
Figure 9. Controller Circuit: Channel Select.....	35
Figure 10. Controller Circuit: Transfer Control.....	36
Figure 11. Controller Circuit: Status Input & Bus Drivers..	37
Figure 12. Controller Circuit: Front Panel Control.....	38
Figure 13. Controller Circuit: Front Panel Address Register	39
Figure 14. Sample Data Transfer Program Segment.....	44

## ABSTRACT

The subject of this thesis is the design of a microprocessor controlled data interface system. The system presented may be used as a remote data acquisition system or a type of industrial controller. Several of these systems may be connected together to create a complex data collection system.

The system requirements and specifications are relatively few. After the specifications of the data bus are given, several interface circuits are described. This is followed by a description of the controller requirements. A microprocessor was chosen and a controller designed around it. The programming techniques required by this specific system are presented as well as the operation of the entire system.

## CHAPTER I

### INTRODUCTION

A data interface system is a term ascribed to various systems capable of obtaining information or data from various devices, such as instruments, sensors, transducers, or computer type memory, and routing this data to other devices, such as another memory or a teleprinter. Exactly how the data are obtained and where they are routed is dependent on the characteristics of a given systems. In one system, the data may be stored in memory for later use while other systems may require immediate use of the data, in which case the data would be routed to the waiting device immediately. The complexity of the system is dependent on the number and particular types of devices interfaced to it as well as the complexity of the controller.

This data interface system has been designed to provide an economical data acquisition system or a control unit. One particular requirement met by including a microprocessor in the controller is that of including the ability to easily preprocess raw data from the interface circuits before sending them to an external device.

### BACKGROUND

Many systems have been developed that could be classified as data interface systems. Computers could be classified as such since they interface memory to external peripherals. This interface may

be through the central processor or through such schemes as direct memory access or separate I/O processors used in conjunction with multi-port memory. Data acquisition systems could also be classified as data interface systems since they interface transducers and sensors to meters, counters, recorders, etc.

The data interface system described in this thesis has evolved from previous data acquisition systems and present techniques of digitally bussing data through a system. The initial objective of this project was to update and replace the present data acquisition system at Spring Creek. Refer to Williams (6) for a report on that particular system. That system was designed to measure various weather and stream parameters along Spring Creek and to transmit the corresponding data back to Montana State University using telephone lines. The system was remotely controlled by an HP2115A minicomputer located in a laboratory at MSU. All conversions of the raw data to reasonable units took place in the computer. The design of this data acquisition system made it very difficult to allow the computer to do much more than initiate the transmission of the current data. The control program in the computer initiated the system periodically to collect the data and store it in memory. When not collecting data, the program would enable a user to call the computer and access the data stored in memory. The data that could have been collected between system initiations were lost. If the data could be collected

continuously and preprocessed on site, the computer's program could be made more flexible as to when to access the system. The computer could then be allowed to access several systems. Many of these factors prompted the development of a new data acquisition system.

Several types of system architecture were investigated as possible structures for the new system. The architecture most applicable to the initial system requirements was a bus oriented structure. The Register Transfer Modules, designed and manufactured by the Digital Equipment Corporation (1), and the standard instrument interface system proposed by the International Electrotechnical Commission (3) were investigated. The resulting system is similar in many respects to each system. The Register Transfer Modules use a sixteen bit data bus in conjunction with a five bit control bus. Control is realized by a hardwired program using evoke modules or by a microprogrammed controller. The International Electrotechnical Commission's system utilizes an eight bit data bus, a five bit management bus, and a three bit handshake bus. The control functions are performed by one of the instruments connected to the bus. Commands are transmitted in the same manner that data are transmitted.

A bus organization was developed by combining several features of both of the above systems. This organization, described in Smith (4), used an eight bit data bus, a six bit control bus, and four nonbussed lines to each channel. Western Telecomputing Corporation

made several additions and changes to this initial bus organization to use in one of their new product lines called the WTC Data Interface System. This modified organization was used in this thesis because it was both convenient and functionally adequate to do so. This bus organization uses an eight bit data bus, an eleven bit control bus, and two non-bussed control lines to each channel. The initial design limits this system to sixteen channels. Each channel may have the capability of being both a source and a destination for data transfers. Like the Register Transfer Modules, there is only one channel acting as the source and one channel acting as the destination for each transfer. Control is realized by a centralized system controller which may range from a controller with a hard-wired sequence of commands to a microprocessor based controller. With the microprocessor controller, data may be preprocessed before being transmitted to the external world.

#### ORGANIZATION OF CHAPTERS

This thesis will discuss in greater detail the various aspects and parts of the data interface system. Chapter II will cover the bus lines and the data transfer sequences. The description of several interface circuits is contained in Chapter III. Controller development for the system will be covered in Chapter IV with a look at the overall system in Chapter V.

## CHAPTER II

### DESCRIPTION OF THE SYSTEM BUS

This chapter presents a description of the system bus lines and the sequence of signals necessary to perform the data transfers.

#### DESCRIPTION OF THE SYSTEM BUS LINES

Several lines are necessary to control the transfer of data between channels. Eleven bussed lines control the transfer operations. Two nonbussed lines from the controller to each of the sixteen channels select which channels are to participate in the data transfers. The data is transferred via a bidirectional eight bit data bus. All the lines are ground true to allow OR operations using open collector TTL.

Lines DB0 - DB7 (Data Bus) form the eight bit data bus. Any channel can be selected as the data source or the data destination. The selected source channel puts data on the bus, and the destination channel loads the data from the bus into one of its internal registers. Alternatively, the controller may also act as either the source or the destination.

After the controller has determined the source and destination channels, a single byte (eight bits) transfer is performed using the EDT, DAV, and DAC lines. The controller sets the EDT (Enable Data Transfer) line true to initiate the operation. The selected source then places eight bits of data on the data bus and sets DAV (Data

Available) true. When the destination detects that DAV has gone true, it transfers the data to its register and sets DAC (Data Accepted) true. The controller then sets EDT false, which in turn causes the source and destination channels to set DAV and DAC false respectively.

Generally, a source transmits a string of bytes to the destination and the handshake described above is performed for each byte. Normally the source determines how many bytes are to be transferred. The end of the byte string is usually indicated by setting the SKP (SKIP) line true. The SKP line may also be set true by the destination to indicate that it cannot accept further data. If the controller is programmable, it may use the SKP line for conditional branching.

Two of the bus lines, SC $\emptyset$  and SC1 (Source Channel), are used to address one of four possible separate source registers or to select one of four possible operating modes. The source may optionally ignore these lines or use them in any way that would improve the usefulness of the channel. Two other bus lines, DC $\emptyset$  and DC1 (Destination Channel), similarly control a destination. One of four destinations or one of four modes of operation can be selected.

Any channel can request service by setting SRQ (Service ReQuest) true. This line is OR tied to enable several channels to request service simultaneously. To service the request, the controller sets

ESC (Enable Status Check) true. This converts all the individual SSC and SDC lines (described below) into status lines to the controller. This allows the controller to identify which channels requested servicing and to take the appropriate action.

The remaining bussed line is the RST (ReSet) line. When this line is set true by the controller, all the channels are placed into a known state. This is done when the system is started or when recovering from a power failure.

The SSC (Select Source Channel) and SDC (Select Destination Channel) lines are not part of the bus structure, but are individual lines from the controller to each channel. These lines are used by the controller to select the data source and destination channels for the data transfers, and to bring status from the channels to the controller when ESC is true.

#### DESCRIPTION OF THE DATA TRANSFER OPERATION

Several operations occur simultaneously during each byte transfer. The following paragraphs describe the effect of the bus lines on the source and destination channels as well as the effects on the controller. A timing diagram for the bus lines involved in a byte transfer is given in Figure 1.

If the source is ready to transmit data, it will set its SSC line true when it detects that ESC is true. When both the channel's

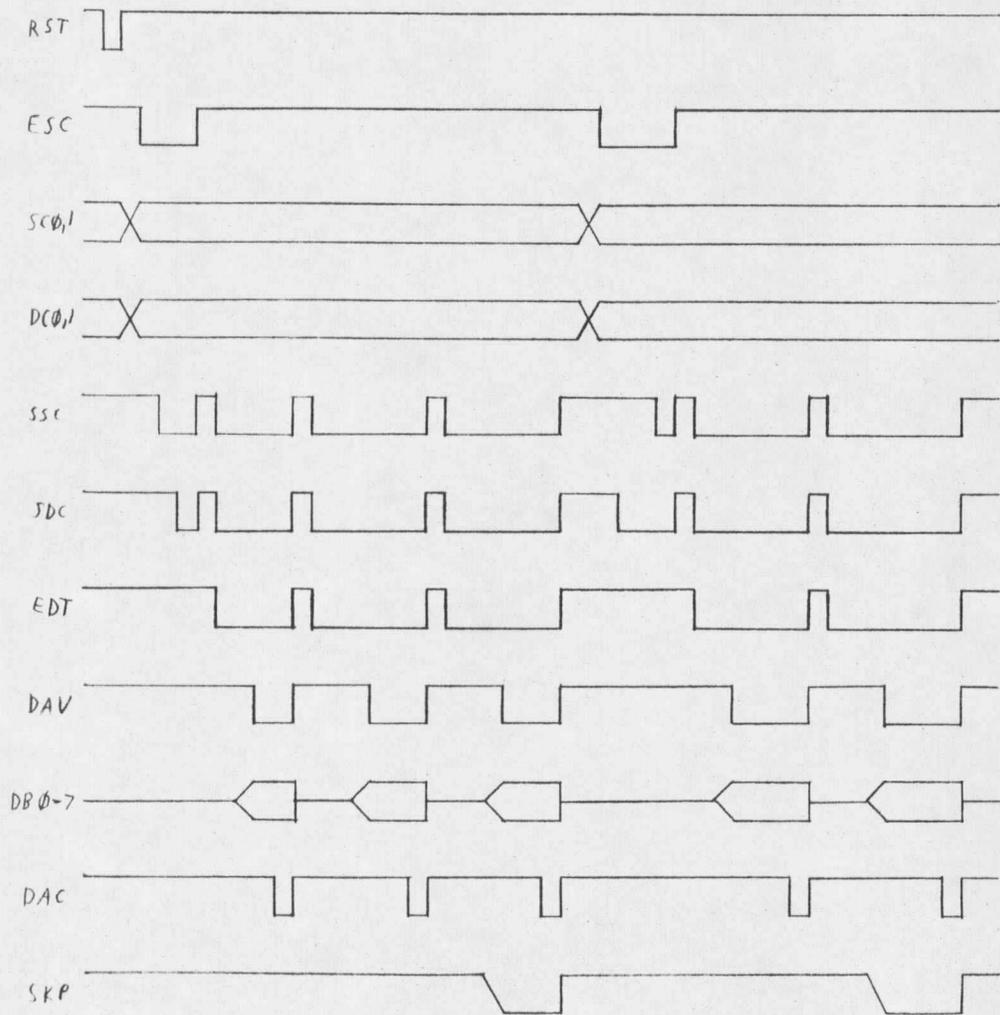


Figure 1. Bus Timing Diagram

SSC and the EDT lines are true, the source register selected by SC0 and SC1 will place its data on the bus. After an appropriate delay to allow the data to settle on the bus, DAV is set true to indicate that the source has placed its data on the bus. When EDT goes false signalling the end of the byte transfer, DAV is set false and the data is removed from the bus. On the transfer of the last byte, SKP is set true during the same interval that valid data is placed on the bus.

If the destination is ready to accept data, it will set its SDC line true when it detects that ESC is true. When both the channel's SDC and EDT lines are true, the destination register selected by DC0 and DC1 will be loaded with the data from the data bus when the destination channel detects that DAV has gone true. Concurrent with the transfer from the bus to a register, DAC will be set true to indicate that the destination has accepted the data from the bus. When EDT goes false at the end of a transfer, the destination sets DAC false. If the destination cannot accept any more data, it will set SKP true when it accepts its last possible byte. This feature must be included in several channels to prevent the destination from "hanging up" the system.

The controller checks the readiness of the channels by setting ESC true and checking the individual SSC and SDC lines. SC0, SC1, DC0, and DC1 may be used to check the status of any of the four

registers of each source and destinations. To enable a byte transfer, the controller selects the desired channels by setting the appropriate SSC, SDC, SC0, SC1, DC0 and DC1 lines. EDT is set true to initiate the transfer. When DAC is set true by the destination, the controller responds by setting EDT false.

## CHAPTER III

### INTERFACE CIRCUITS

This chapter describes several of the interface circuits that have been designed for the system. The first circuit, the counter-timer, will be described in greater detail than the subsequent circuits since the handshake and bus interface circuits are essentially the same for all the circuits and are of primary interest in this chapter.

#### COUNTER-TIMER

This interface circuit contains both an event counter and an interval timer. The event counter advances one count whenever an input pulse is received from an external source, such as a digital anemometer. The interval timer measures the time interval between any two consecutive pulses. In the case of the anemometer, the counter will hold the wind flow, and the timer will hold the reciprocal of the wind speed.

The counter portion of the circuit, shown in Figure 2, is a four digit decade counter. Whenever the data interface system wants to access its contents, the count is latched into Tri-state latches, which each place their digits onto the bus in turn, the most significant digit first.

The timer, shown in Figure 3, is much the same as the counter,

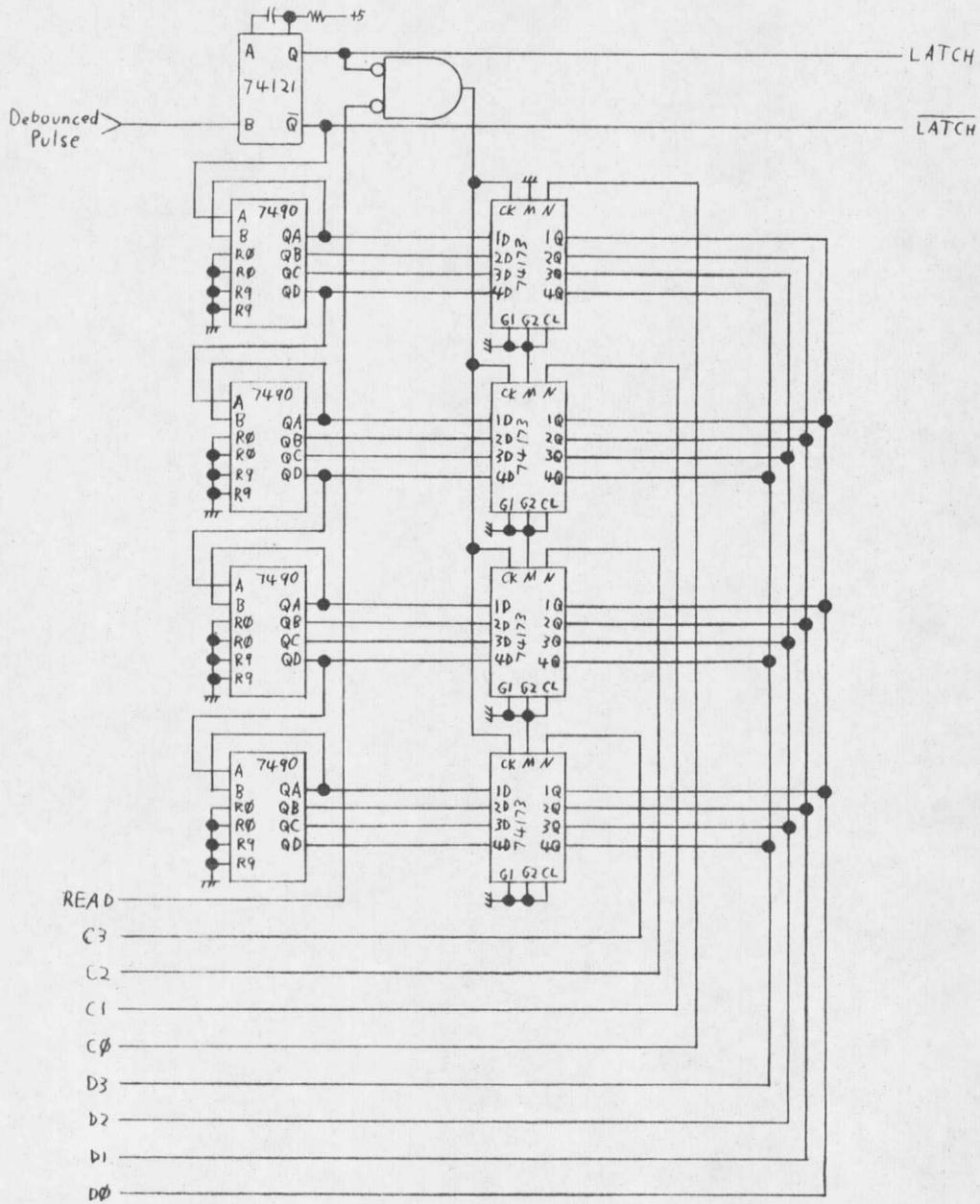


Figure 2. Counter-Timer Circuit: Counter

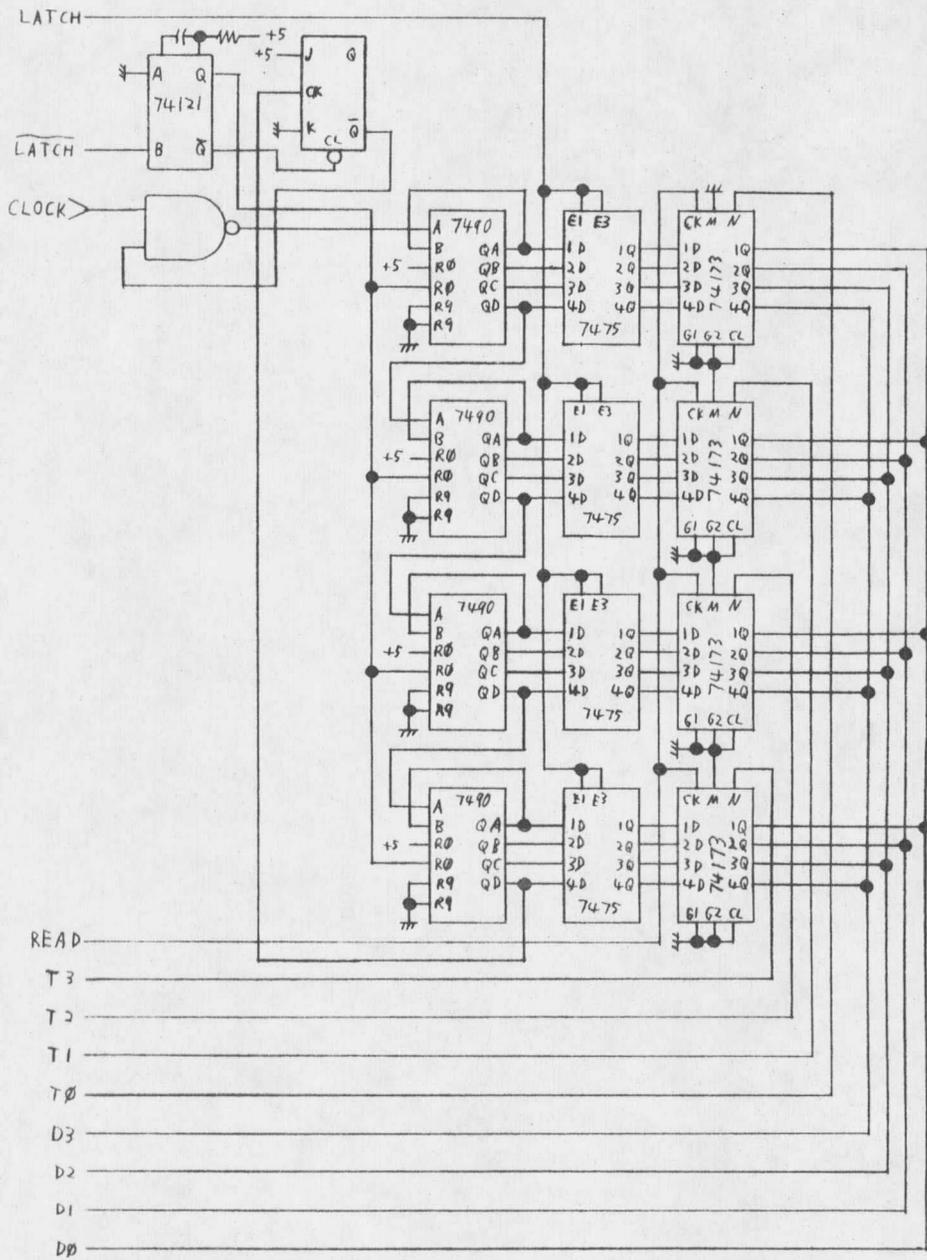


Figure 3. Counter-Timer Circuit: Timer

except an internal clock is counted. When an input pulse is received, the time counter contents are latched, the counter cleared, and counting is resumed. When the timer is accessed by the system, the contents of the latches are latched in Tri-state latches, which place their digits on the bus in turn, the most significant digit first. Provisions are included to inhibit further counting when the four digit decade counter overflows. Overflow is indicated by a 0000 output.

The control portion of this interface performs several functions. It must control whether the counter or the timer is to place data on the bus, identify which one is to be accessed, and fill out the digits from either to form ASCII characters. The control will always indicate that it is ready to transmit data whenever it is interrogated by the ESC line.  $SC\emptyset$  is used to select the counter or the timer.  $SC\emptyset$  is false for the counter and true for the timer. When the system requests that the channel transmit its data, the control first sends an identifying ASCII alphabetic character. This character is selected by inserting jumper diodes into the appropriate places of the circuit as shown in Figure 4. Then the four ASCII decimal digits are transmitted, the most significant digit first. Concurrent with the last digit, SKP is set true to signal the end of the byte string. SSC and EDT are ANDed together to detect when the system is requesting its data. This signal is used to enable

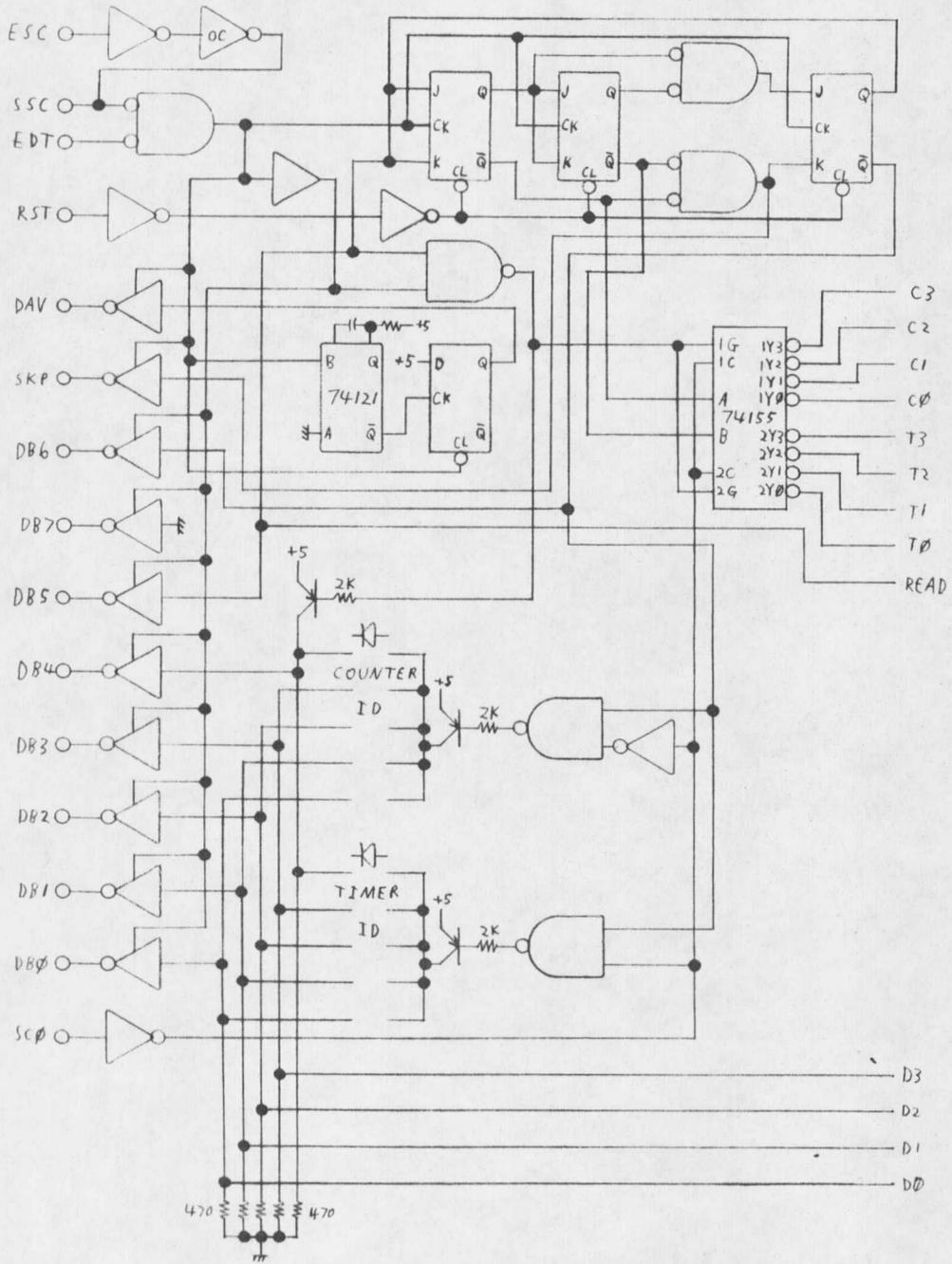


Figure 4. Counter-Timer Circuit: Control

bus drivers, and to advance the character counter. A delayed signal is used for DAV to allow the data to settle on the bus. This interface circuit does not use SRQ since very few circumstances require

An interface circuit very similar to the above counter-timer is the quad counter. This interface has four event counters. Operation is identical to the counter portion of the counter-timer. The control portion is much the same except that one of four sources must be selected and identified rather than one of two.

#### MEMORY

Memory is used in this data interface system for both data storage and control program storage. It is most easily implemented in multiples of 256 bytes since the data bus is eight bits wide. However, it is assumed here and in subsequent chapters to be 4096 bytes per channel. This allows sufficient space for a relatively complex control program as well as some data storage space. Besides being able to access the contents of the memory, it is necessary to specify which particular byte is to be accessed. A twelve bit address register is included to accommodate this. When using the memory for data storage, it is convenient to keep track of how many bytes have been stored and where the next byte is to be stored. A byte counter and an auto-increment address register are included to accommodate this feature.

The DC0 and DC1 lines are used to select the memory, the address register, or the byte counter as the destination. DC0 is set false and DC1 true to select the address register. Since 4096 bytes require twelve bits of address, two bytes must be received to completely specify the address. The first byte contains the highest four bits and the second byte the lower eight bits. The higher four bits of the first byte are ignored. The byte counter is accessed by setting both DC0 and DC1 true. This allows the counter to be set prior to reading from memory when in the data dump mode. Actual memory access has two possible modes. The mode used by the controller for its normal memory write operations is selected by setting both DC0 and DC1 false. Neither the address register nor the byte counter is altered in this mode. The other mode, selected by setting SC0 true and SC1 false, is used when storing a string of bytes in memory from one of the other system channels. Both the address register and the byte counter are incremented by one count for each byte stored.

Similar, SC0 and SC1 are used to select the memory, the address register, or the byte counter as the source. SC0 is set false and SC1 true to select the address register. The address is transmitted as two bytes. The higher four bits of the first byte are set to 0 since they are not used in the actual address. SKP is set true on the second byte. Both SC0 and SC1 are set true to access the byte counter. As in the destination mode, there are two modes of memory

access in the source mode. For normal memory read accesses by the controller, both SC0 and SC1 are set false. The other mode, selected by setting SC0 true and SC1 false, is used when dumping stored data to another channel. The address register is incremented and the byte counter decremented for each byte accessed. SKP is set true when the byte counter returns to zero.

#### SERIAL OUTPUT

The serial output interface designed by Western Telecomputing Corporation (5) interfaces EIA RS 232C compatible modems and serial printers to the data interface system. Ground true TTL compatible and twenty milliamperere compatible outputs are available in addition to the EIA levels. There is one control-out signal, one control-in signal, and the serial character data output line on the external side of the interface. The data from the interface are transmitted in bit serial form on the data output line. The lower seven bits from the bus are always transmitted. There are several options selected by jumpers for the eighth bit. This bit may be the even or odd parity bit of the lower seven bits, a mark, a space, or the eighth bit from the bus (DB7). Either one or two stop bits are transmitted, depending on another jumper of the interface. Several different data rates may be selected by yet another jumper and the choice of two clock crystal frequencies. The control-out line may be used to

control a modem carrier or a printer motor. The control-in line is used to enable the transmission of data from the interface and to control some of the interface operations.

This interface operates only as a destination channel. Either the control register or the data register may be addressed during a byte transfer by using DCØ. DCØ is set true to set the control register. DBØ sets the control-out line, 0 turning it off and 1 turning it on. DB1 is used to control the RQS operation of the channel. DB1 is 0 to enable the operation and 1 to inhibit it. The remaining six bits are ignored. A data byte can be placed in the data register by setting DCØ false during a byte transfer to this channel. The handshake will be completed when the transmission of the previous character has been completed and the control-in line is on. During a status check by the ESC line, this channel's SDC line will be false if the channel is busy transmitting a character or the control-in line is off. It will be true only when the channel is ready to accept another character and the control-in line is on. RQS will be set true, if enabled by the control bit described above, when the control-in line is on and the channel is not transmitting a character. The RST line will set the control register so that the control-out line is off and the RQS operation is enabled.

## CHAPTER IV

### SYSTEM CONTROLLER

This chapter presents a description of the basic requirements of the system controller and the minimum requirements for a microprocessor to be used in a complex controller. A microprocessor that fits these requirements was selected and designed into a system controller.

#### CONTROLLER REQUIREMENTS

The system controller is in charge of all data transfers performed on the bus. As described in Chapter II, the controller initiates and terminates each data transfer sequence. The status of each channel is used by the controller to decide which course of action it should follow.

One of the simplest controllers is one in which a string of data bytes is transmitted upon receipt of a SRQ signal. One pair of channels transfer a string of bytes until the SKP line is set true by the source. Then another pair of channels transfer another until SKP becomes true again. Upon receipt of the last SKP signal, the system enters an idle state until the next SRQ signal is detected. When SRQ is received again, the above sequence is repeated. The sequence of the pairs of channels participating in the data transfers can be modified by changing jumpers in the controller. This con-

troller does not check the status of the individual channels, but depends on the fact that the transfer handshake sequence will be completed only when both the source and destination channels are ready to do so.

A more sophisticated controller contains a microprocessor which chooses its course of action on the basis of the instructions of the control program. These instructions direct the controller to check status, initiate data transfers, and process the data collected. The control program will be located in a memory located either in one of the system channels or in the controller. In most cases, the speed of the system will be limited by the speed of the microprocessor. If the control program memory is located in one of the system channels, instruction fetching may be slightly slower than if the program memory was located in the controller. However, the memory located in one of the system channels should be more general purpose since it could also be used for data storage using channel to channel transfers.

Using the memory described in Chapter III, the instruction fetch or memory read operation would be executed as follows. The memory address would be sent to the memory channel address register, the highest four bits first followed by the lower eight bits. Then the contents of the addressed memory location would be sent from the memory channel to the controller.

The memory write operation would be executed as follows. The memory address would be sent to the memory channel address register in the same way it was sent for the memory read operation. Then the data to be stored in the addressed location would be sent from the controller to the memory channel.

Data transfers between channels are enabled during the execution of particular instructions, usually certain I/O instructions. Under normal operation, only one byte may be transferred per single execution of these instructions. There would usually be a few instruction fetch operations between data byte transfers between channels. If the destination channel or the source channel is one of the memory channels, it should be a different channel than the one instructions are currently being fetched from since the fetch operation would change the address register if they were both accessing the same channel.

#### MICROPROCESSOR REQUIREMENTS

The controller must be designed so the microprocessor can fetch its instructions from a memory channel, store data in a memory channel, and enable byte transfers between channels. Since the data bus is eight bits wide, it is only logical that the microprocessor utilize instructions and data that are eight bits wide. To allow enough programming space for relatively complex programs, and to be com-

patible with the memory presented in Chapter III, the microprocessor must be capable of addressing at least 4096 bytes of memory. The instruction set should be capable of allowing a relatively complex program to be easily realized. There should be a set of particular instructions to be used to set up and enable transfers between system channels.

Since the time required to access memory depends on the speed of the handshake sequences, which may vary considerably, the processor should have the capability of being held in an idle state until the operation being executed is completed. The instruction execution time should be short enough to allow the system to respond rapidly to any condition that needs immediate attention.

#### INTEL 8008 MICROPROCESSOR DESCRIPTION

The Intel 8008 microprocessor, described in detail in Intel (2), is the least expensive processor that fulfills the requirements for the system controller. It is an eight bit processor with forty-eight data oriented instructions. It can be used with any type and speed of memory. An address of fourteen bits allows addressing of up to 16,384 bytes of memory. By using the I/O instructions, a data transfer between channels can easily be set up and executed. The basic cycle time for the 8008 is twenty microseconds. A faster version, the 8008-1, has a basic cycle time of twelve and one half micro-

seconds. An internal stack allows nesting subroutines up to seven deep. Seven registers are accessible by the program. One of these is the accumulator while the other six are used for temporary data storage.

The processor communicates via an eight bit bidirectional bus. Time multiplexing of the bus allows control information, fourteen bit addresses, and data to be transferred between the processor and external circuits. Besides two inputs for the two phase non-overlapping clock, two inputs and four outputs are used for processor control. The state outputs (S0, S1, and S2) and the SYNC output indicate the state the processor is in during instruction cycles. The READY line is used to hold the processor in an idle state until it is set true. The 8008 has limited interrupt capability through the use of its INTERRUPT input. Due to the relative difficulty of saving the state of the processor during interrupt servicing, it is usually not used except when initially starting execution of the program.

There are eight states the processor may enter. Five of these are entered during the normal execution of a typical instruction. The lower byte of the address is transmitted during state T1. The higher six bits of the address and two control bits are transmitted during T2. The two control bits designate what type of instruction cycle is being executed. Both D6 and D7 are 0 for the instruction

fetch cycle. D6 is 0 and D7 is 1 for cycles in which additional bytes of data or address are fetched from memory. Both D6 and D7 are 1 for the memory write cycle. D6 is 1 and D7 is 0 for the I/O execution cycle. For the I/O cycle, the contents of the accumulator is transmitted during T1. The lower six bits of the byte transmitted during T2 contain the I/O device address and specify whether the instruction is an input or output instruction. During T3, the data from memory or an input device is received by the processor, or the data to be stored in memory is transmitted by the processor. Execution of the instructions occurs during states T4 and T5. Some instructions will skip one or both of these last states. State T1 is replaced by T1I when the processor has been interrupted to indicate to the external circuitry that the interrupt has been recognized. This is usually used to signal the circuitry to jam an instruction into the processor to replace the instruction normally fetched from memory. The program counter is not advanced during a cycle beginning with state T1I. The processor will enter state WAIT immediately after T2 if the READY line is low. The processor will remain in WAIT until READY is set high, enabling the processor to enter T3. The STOPPED state is entered after state T3 when a HALT instruction has been received by the processor.

## DESCRIPTION OF THE MICROPROCESSOR CONTROLLER

Basically, the configuration of the system with a microprocessor controller is that of a simple computer. Instructions are fetched from memory and executed, data is written in memory, and data transfers are set up and executed.

To realize a controller with the Intel 8008, the time multiplexed data on the processor's bus must be demultiplexed so that it can be used at the system's speed, which will be quite different than the speed of the processor. Circuitry must also be included to allow the input data from the bus, status registers, or interrupt instruction register to be received by the processor during T3. The byte transmitted by the processor during T2 is latched and decoded to determine what type of cycle is being executed. The following paragraphs describe the operations of the different cycles as executed by the design shown in the block diagram of Figure 5. The schematic diagram is Figures 6 through 12.

The T1 latch contains the low byte of the memory address for the instruction fetch or memory read cycle. The T2 latch contains the higher six bits of the memory address. The highest two bits of the address are used to select one of the first four system channels as the destination for the next two transfers and the source for the third transfer. The higher address byte is transferred from the T2 latch first, followed by the lower address byte from the T1 latch.

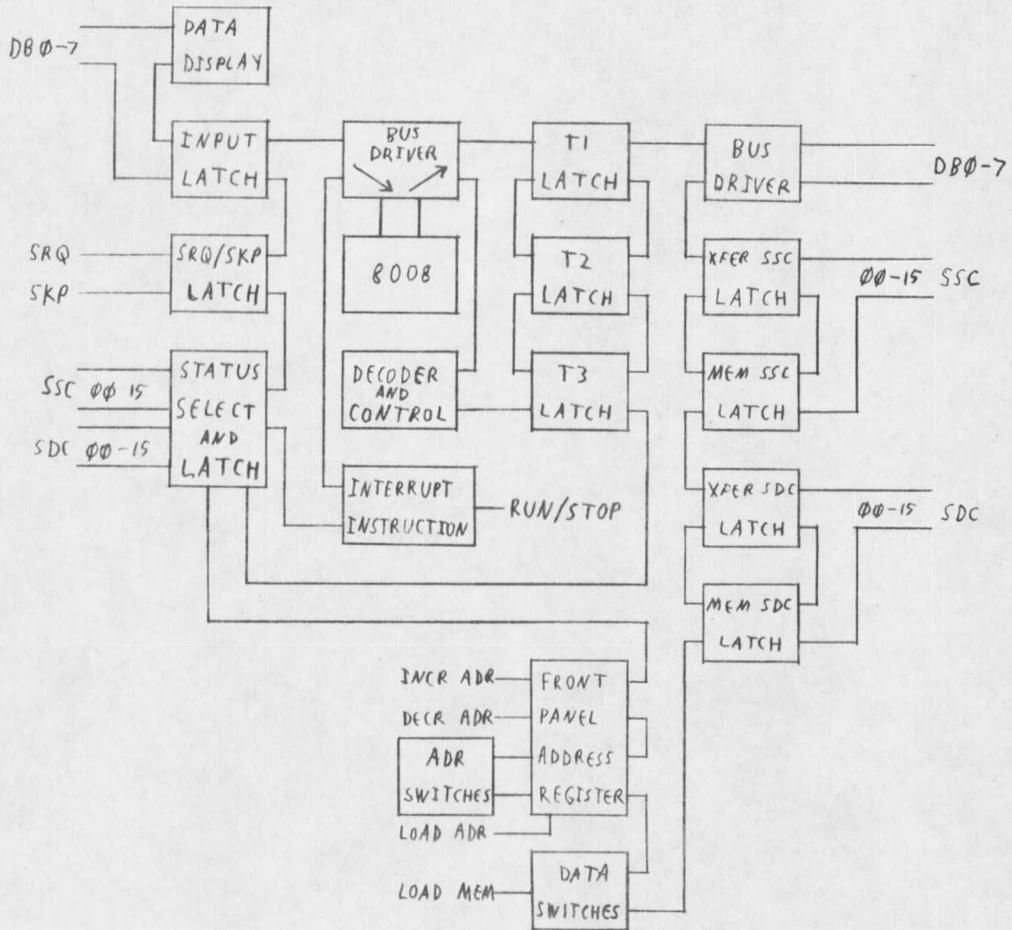


Figure 5. Controller Block Diagram

Then the selected channel is instructed to transmit the contents of the addressed memory location to the controller. This byte is latched in the INPUT latch. The contents of this latch are transferred at the proper time to the processor for instruction decoding and execution.

The T1 latch also contains the low byte of the memory address for the memory write cycle. The T2 latch contains the higher six bits of the address. The highest two bits of the address are used to select one of the first four system channels as the destination for the next three transfers. The higher address byte is transferred first from the T2 latch followed by the lower address byte from the T1 latch. Then the data byte contained in the T3 latch is transferred.

For the I/O instructions, the T2 latch contains the device address and an indication of whether an IN or OUT instruction is being executed. The contents of the T1 latch are the same as the contents of the processor's accumulator. The device address and the type of instruction determine how the controller executes the I/O instruction. An OUT 10B (the B indicates octal or base eight numbers) instruction will cause the contents of the T1 latch to be stored in the XFER SSC latch. This latch is used to select the source channel during a transfer between channels. If the highest bit is true, the controller is specified to be the source channel. An OUT 11B instruc-

tion will cause the contents of the T1 latch to be stored in the XFER SDC latch which is used to select the destination channel during a transfer between channels. If the highest bit is true, the controller is specified to be the destination channel. An IN 0 instruction enables the transfer of a data byte between the source and destination channels specified by the last OUT 10B and OUT 11B instructions. If the controller is specified to be the source channel, the contents of the T1 latch are placed on the data bus. When the byte has been transferred, SKP is latched into the SRQ/SKP latch and the data byte latched in the INPUT latch. The contents of the INPUT latch are then transmitted to the processor's accumulator. The IN 1 instruction is identical to the IN 0 instruction except that SKP is not latched. The IN 2 instruction is used to load the status of the channels into the processor's accumulator. The contents of the T1 latch are used to select which eight channels are to have their status sent to the processor. The IN 3 instruction causes the contents of the SRQ/SKP latch to be sent to the processor's accumulator.

In order for this controller to be useful, several operator controls must be provided. An operator should be able to start and stop the execution of the control program. Also, he should be able to examine and alter the contents of the program memory. This design provides these controls.

When the RUN/STOP switch is operated, an interrupt request is sent to the processor. If the processor was stopped, a RST 0 instruction is sent to the processor during the interrupt cycle to start the execution of the program at address 00000B. Whenever the processor is started, the front panel address register is cleared. If the processor was executing the program when the RUN/STOP switch was operated, a HALT instruction will be sent to the processor during the interrupt cycle. When the processor has been stopped, the contents of the front panel address register (cleared when the processor was started) will be transmitted to the channel specified by the highest two bits of the address. The contents of that memory location will then be transmitted to the controller where it will be displayed in the DATA DISPLAY. The DATA DISPLAY is designed to monitor the data bus, latching the data whenever DAC goes true.

The following controls will operate only when the processor has been stopped. The MANUAL RESET switch will set RST true momentarily to reset the logic of all the system channels. The LOAD ADR switch will cause the ADR switches to be loaded into the front panel address register. The contents of this register are displayed in the ADDRESS DISPLAY. INCR ADR will increment this register, and DECR ADR will decrement it. Any of these last three switches will cause the contents of the front panel address register to be sent to the channel specified by the highest two bits of the register. The channel will

then send the contents of the addressed memory location back to the controller to be displayed in the DATA DISPLAY. The LOAD MEM switch causes the DATA switches to be transmitted to the memory location specified by the last LOAD ADR, INCR ADR, or DECR ADR operation.

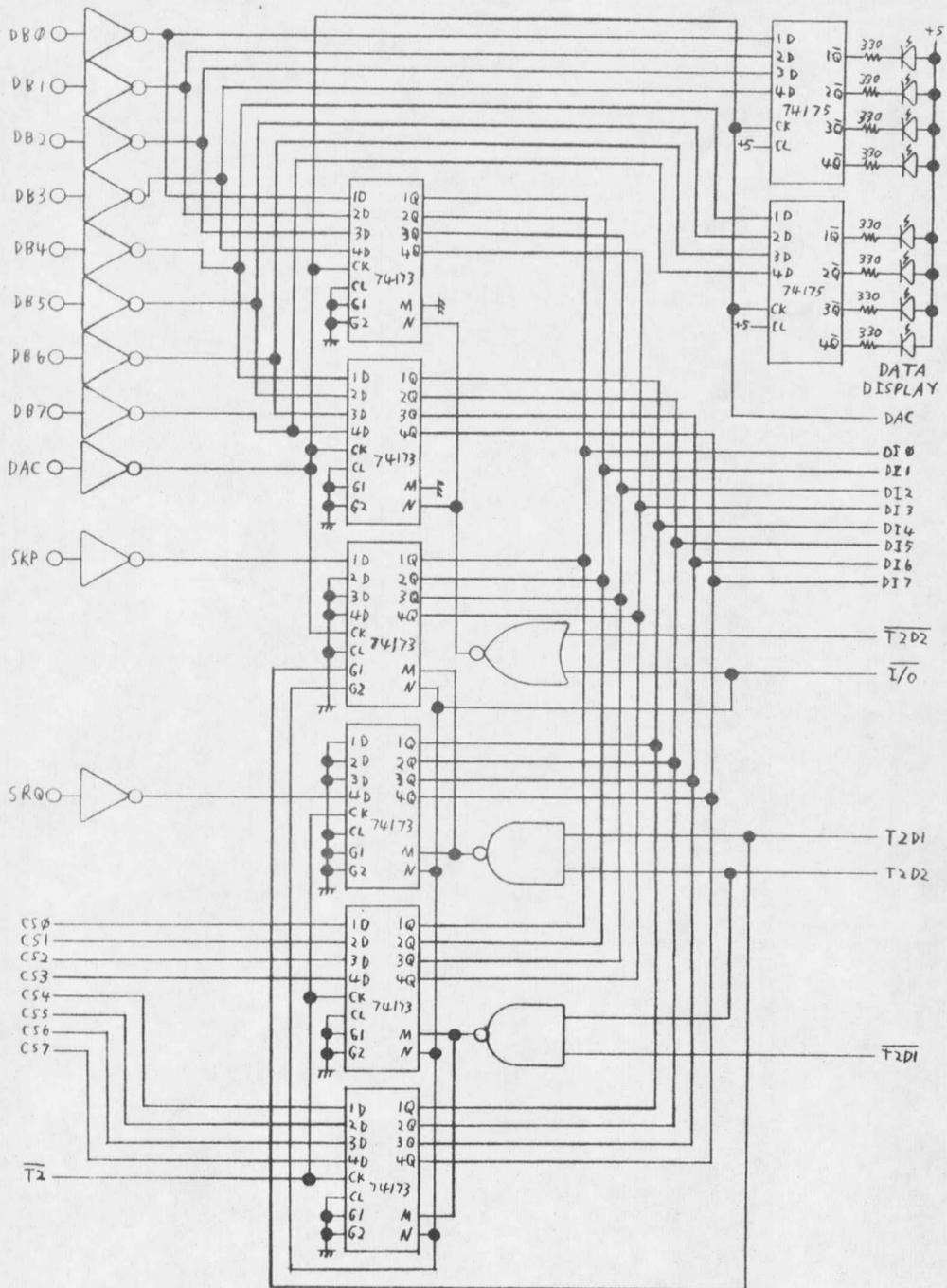


Figure 6. Controller Circuit: Input

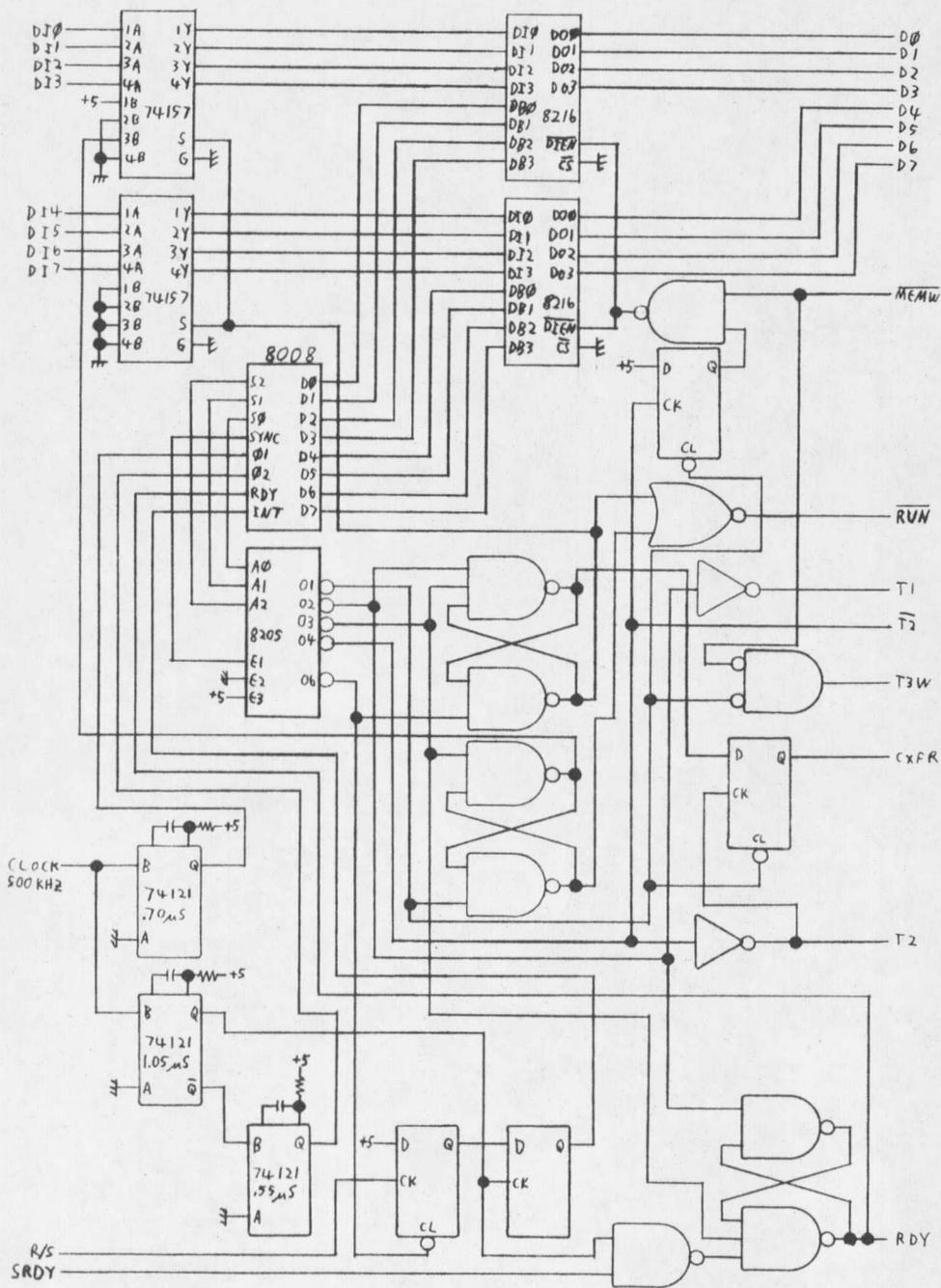


Figure 7. Controller Circuit: CPU

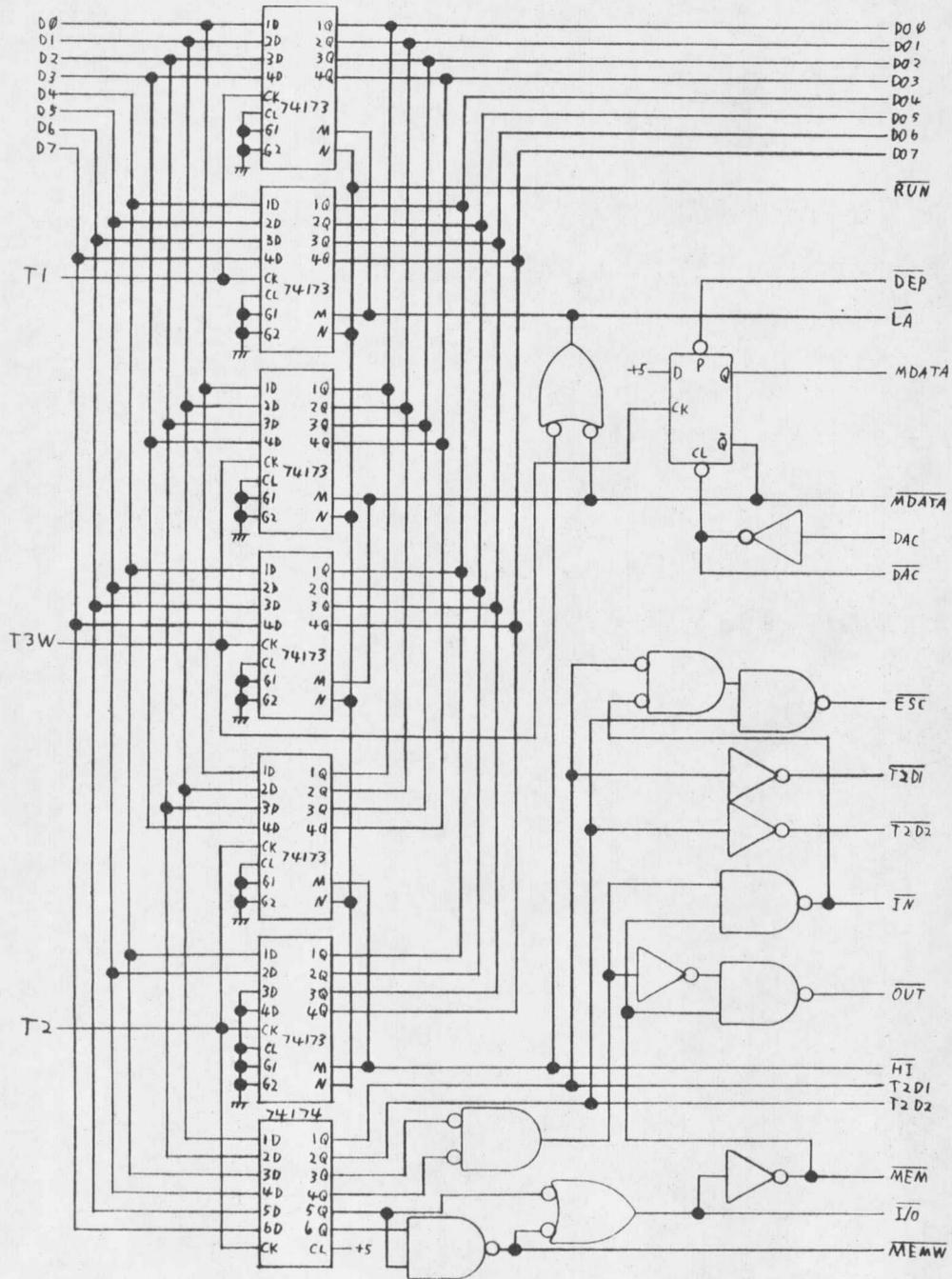


Figure 8. Controller Circuit: Decoder

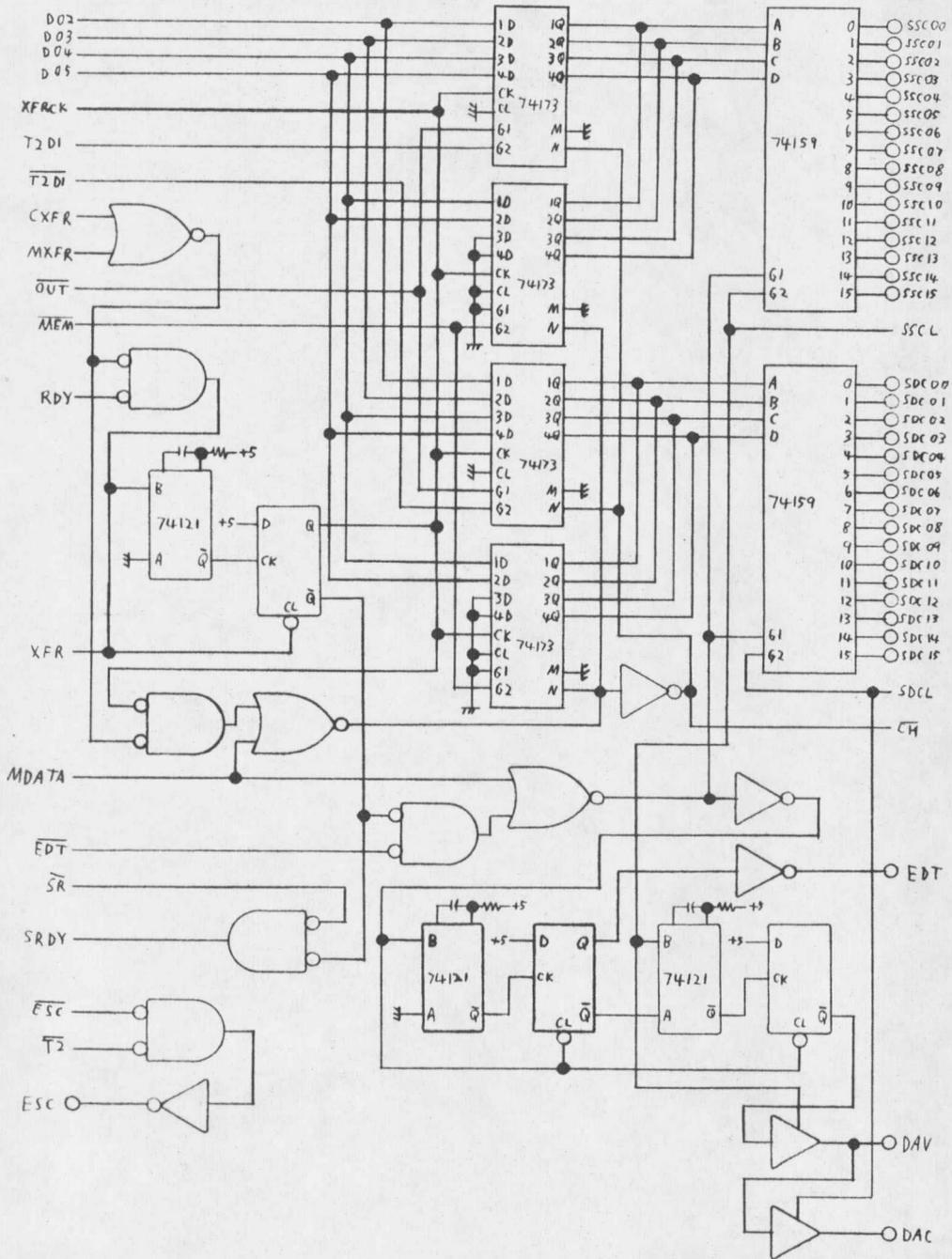


Figure 9. Controller Circuit: Channel Select

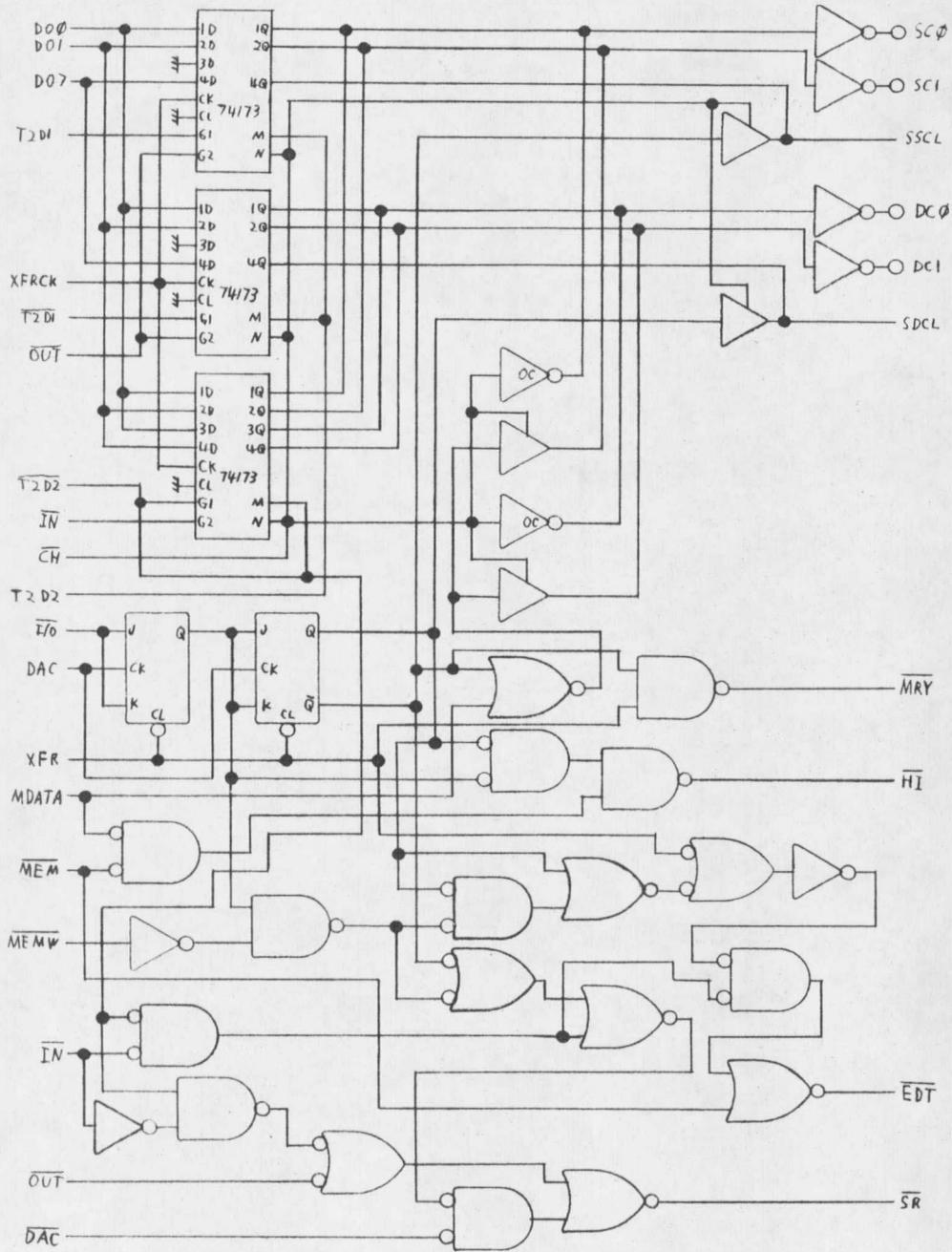


Figure 10. Controller Circuit: Transfer Control

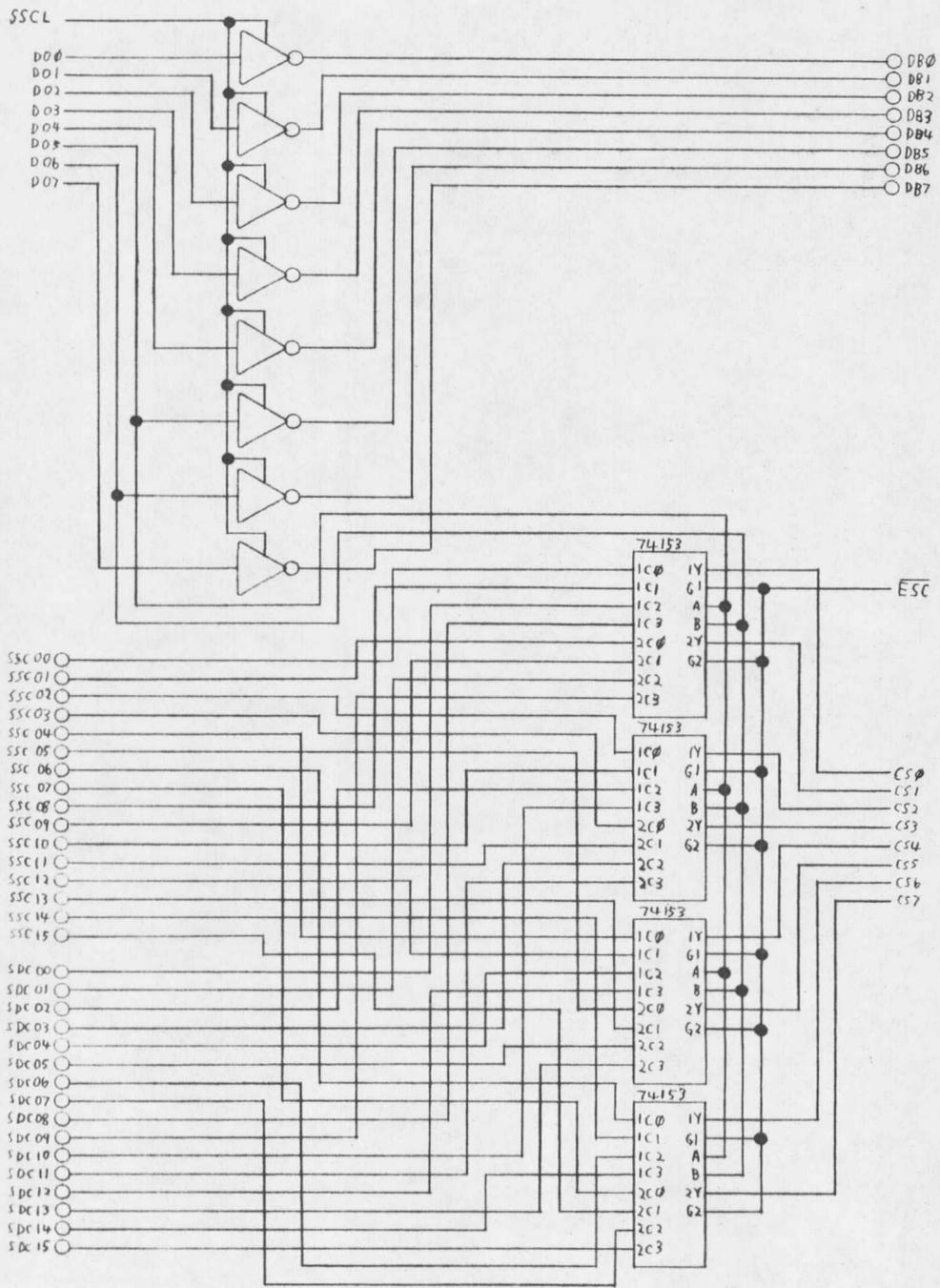


Figure 11. Controller Circuit: Status Input & Bus Drivers

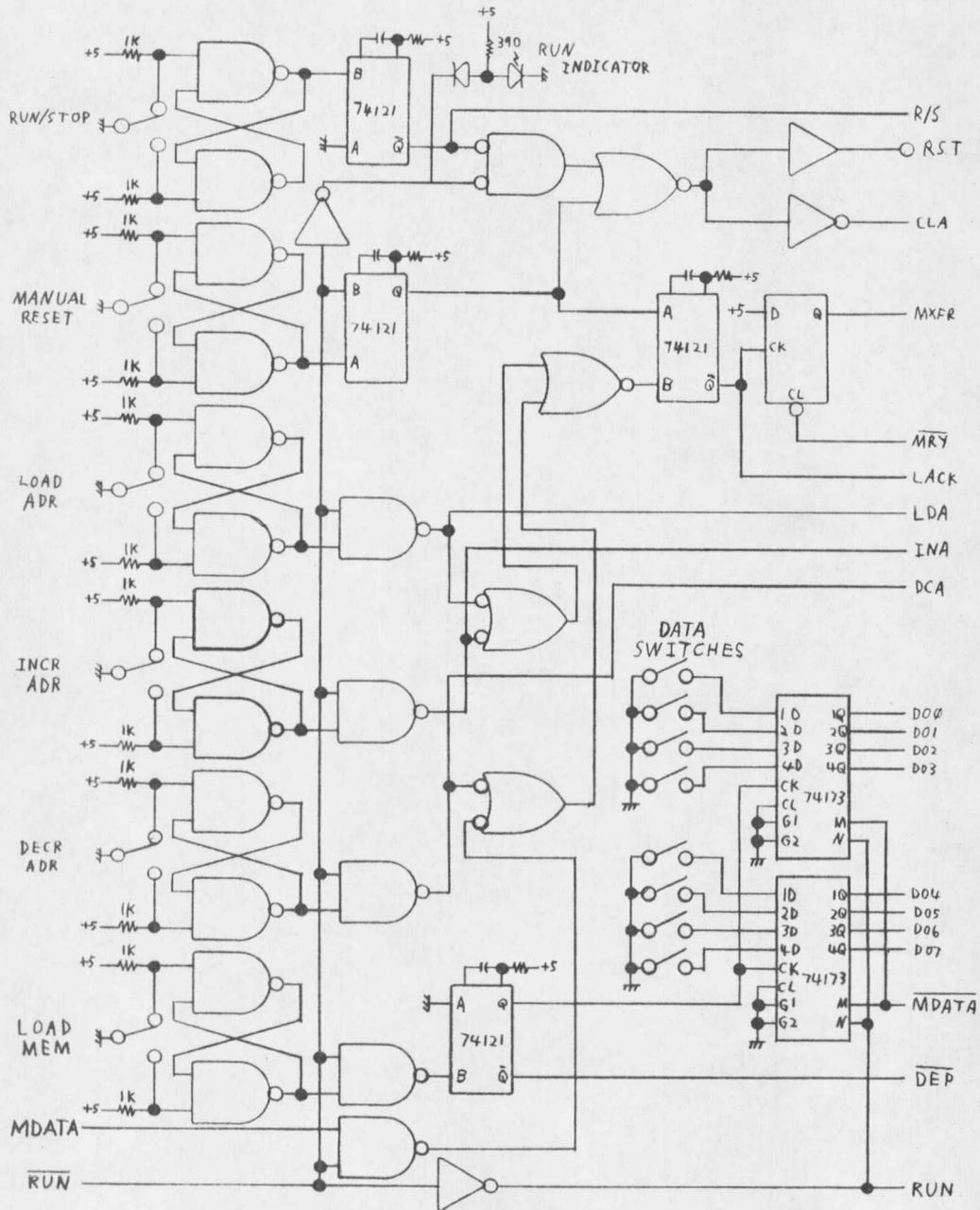


Figure 12. Controller Circuit: Front Panel Control

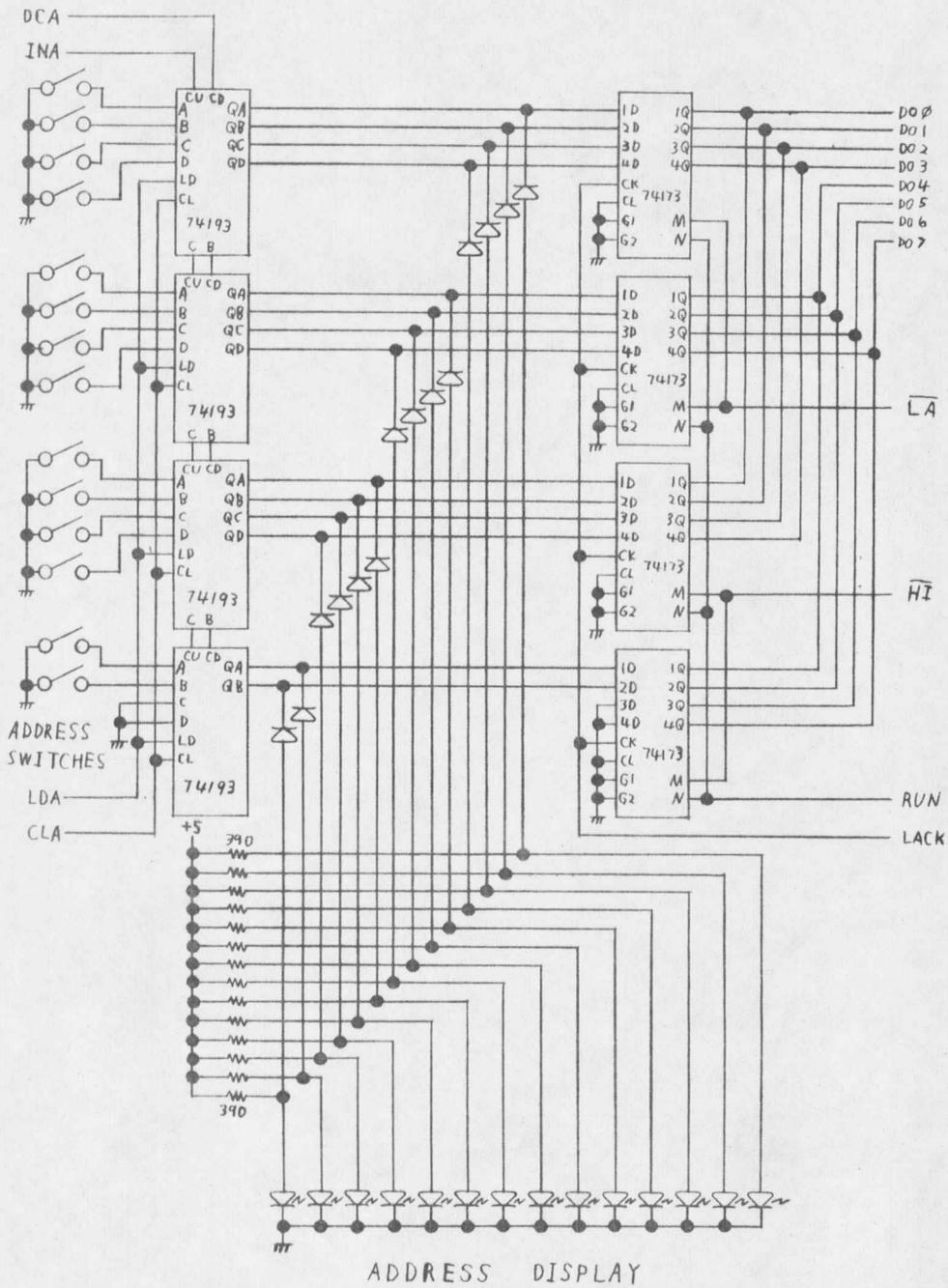


Figure 13. Controller Circuit: Front Panel Address Register

## CHAPTER V

### DESCRIPTION OF THE MICROPROCESSOR CONTROLLED SYSTEM

This chapter describes the requirements and operation of the complete microprocessor controlled data interface system.

### SYSTEM CONFIGURATION

The configuration of the system is almost entirely up to the system user. There is one constraint to be considered, however. The program memory must be located in the lowest numbered channels. If the memory is at the full 16,384 byte capacity of the 8008, the memory would be located in channels 0 through 3, 4096 bytes per channel. If only 4096 bytes of memory are to be used, it must be located in channel 0. Keeping the above restrictions in mind, a sample system is configured as shown in Table 1, assuming that 16,384 bytes of memory are to be used.

### PROGRAMMING CONSIDERATIONS

Developing the control program should be relatively straight forward. None of the 8008 instructions except the I/O instructions require any special considerations. The main considerations with the I/O instructions is the format of the data being input or output.

All the OUT instructions have the same format. If the address of the output device is 00B, the source channel is being specified,

Table I  
Sample System Configuration

Channel Number	Channel Use
0	Memory (00000B-07777B)
1	Memory (10000B-17777B)
2	Memory (20000B-27777B)
3	Memory (30000B-37777B)
4	Serial Output
5	Counter-Timer
6	Quad Counter
7	Quad Counter
8-15	Not used

and bits 1 and 0 of the accumulator specify  $SC1$  and  $SC0$  respectively. Bits 5 through 2 of the accumulator specify which one of the sixteen channels is to be the source channel on the next byte transfer between channels. If bit 7 is 1, the controller will be the source channel. If the device address is 11B, the contents of the accumulator specify which channel is to be the destination channel,  $DC1$ , and  $DC0$ . If bit 7 is 1, the controller will be the destination channel.

Some of the IN instructions require the accumulator to be pre-set. If the input address is 0 or 1 and the controller has been specified as the source channel, the accumulator must contain the data to be transmitted at the time the IN instruction is executed. The accumulator will always contain the byte that was transferred upon completion of the execution of the IN instruction. If the input address is 2, the accumulator must be set to specify which eight channels are to transmit their status to the controller. The lowest two bits of the accumulator specify  $SC1$  and  $SC0$ , and  $DC1$  and  $DC0$ . If bit 5 of the accumulator is 0, channels 0 through 7 will transmit their status to the controller. If bit 5 is 1, channels 8 through 15 will send their status to the controller. If bit 6 is 0, the source channels will have their status checked. If bit 6 is 1, the destination channels will have their status checked. The accumulator will contain the status of the selected eight channels after execution

of the IN 2 instruction is completed. The status is received in inverted form, that is when a channel is read, the corresponding bit of the accumulator is set false. The bit will be set true if the channel is not ready. The accumulator has no effect on the IN 3 instruction and does not need to be preset. The accumulator will contain the present SRQ and the latched SKP signals upon completion of the execution of the IN 3 instruction. SRQ will be stored in bit 7 and SKP in bit 0. The remaining bits will be 0.

Figure 14 presents a possible sequence of instruction to command the system to transfer the contents of the counter of the counter-timer channel to the serial output channel. The sample configuration given in Table 1 is assumed by the program.

#### OPERATION OF THE SYSTEM

While this system may be considerably slower than a system with a hardwired controller, it is much more flexible and powerful. The simple system just needs to be powered up and possibly reset to put the system into operation. The microprocessor controlled system requires that a program be loaded in the program memory before it can be started.

In the system presented in this thesis, loading the program requires loading the program by hand through the front panel switches or loading a bootstrap program through the front panel and using this

```
MVI A,200B      ;CONTROLLER
OUT 10B         ; IS SOURCE
MVI A,21B      ;SERIAL OUTPUT
OUT 11B        ; CONTROL IS DESTINATION
MVI A,3        ;TURN ON CONTROL-OUT
IN 1           ; AND DISABLE SRQ
MVI A,24B      ;SET COUNTER OF
OUT 10B        ; COUNTER-TIMER AS SOURCE
MVI A,20B      ;SET DATA REGISTER OF
OUT 11B        ; SERIAL OUTPUT AS DESTINATION
LOOP: MVI 100B  ;GET STATUS
IN 2           ; OF LOWER 8 DESTINATIONS
ANI 20B        ;ISOLATE SERIAL OUTPUT
JNZ LOOP       ;NOT READY YET
IN 0           ;READY: DO A TRANSFER
IN 3           ;GET SKP DATA
RAR           ;PUT IN CARRY BIT
JNC LOOP       ;SKP NOT SET: DO ANOTHER TRANSFER
              ;SKP SET: FALL OUT OF LOOP
```

Figure 14. Sample Data Transfer Program Segment

bootstrap program to load the control program. The system could be made easier to load if the bootstrap were contained in a read-only memory. In this case, only a JMP instruction and the bootstrap address would need to be loaded.

Once the program has been loaded by one of the above methods, all that is required to put the system into operation is to operate the RUN/STOP switch. This will cause the processor to start fetching and executing instructions starting at memory address 00000B. Unless the program has a HALT instruction included or the RUN/STOP switch is operated, the program will continue fetching and executing instructions. By using a battery system that is kept charged by the commercial mains or some other means, the system will be power fail safe.

## CHAPTER VI

### CONCLUSION

The microprocessor controlled data interface system is economically realizable using the design presented in this thesis. This data interface system can be operated with a number of benefits. Changing a hardwired controller is eliminated since only a minor program change is all that is required in the microprocessor controlled system. In addition, some data reduction may be performed in the processor before being transmitted to the external world. By designing special interface circuits, this system may communicate with another similar system or even with instruments connected to a bus such as the bus structure proposed by the International Electrotechnical Commission.

The ease of interfacing to the system presented here is also valuable. Only a small handful of components is required to perform the interface functions. All that is required of the device being interfaced is that it should be easily adapted to allow its data to be time multiplexed onto the bus.

The system should be powered by a power fail safe source. Present semiconductor memories will lose their contents if the power to them is interrupted. Since this entire system is solid state and operates on relatively low voltages, a possible source may be an automotive lead-acid battery kept charged by a trickle charger that is connected

to the commercial mains.

Cost has been one factor not discussed in detail in this thesis, but was very important in designing the controller for the system. Intel's 8008 microprocessor was the least expensive processor on the market that met the system requirements. This controller should be producible at a relatively small fraction of the total system cost, though it will probably be more expensive than a typical interface circuit. The most expensive interface circuit will probably be the memory channel due to the relatively high cost of semiconductor memories, though this is dropping fairly rapidly.

## REFERENCES

## REFERENCES

1. Digital Equipment Corporation, Register Transfer Modules, Digital Equipment Corporation, Dec. 1973.
2. Intel Corporation, Intel MCS-8 Users Manual, Intel Corporation, Nov. 1974.
3. International Electrotechnical Commission, Standard Interface Systems for Programmable Measuring Apparatus, Part 2: Byte-Serial-Parallel Interface Systems, International Electrotechnical Commission, March 1974.
4. Smith, Dennis, Digital Data Acquisition Systems Interface Specifications, Unpublished, June 1974.
5. Western Telecomputing Corporation, WTC-700, 100 A Serial Output Interface Card Specifications, Unpublished, April 1975.
6. Williams, Robert M., Development of the Spring Creek Data Acquisition System, Master's thesis, Montana State University, Electronics Research Laboratory Report 3174, August 1974.

