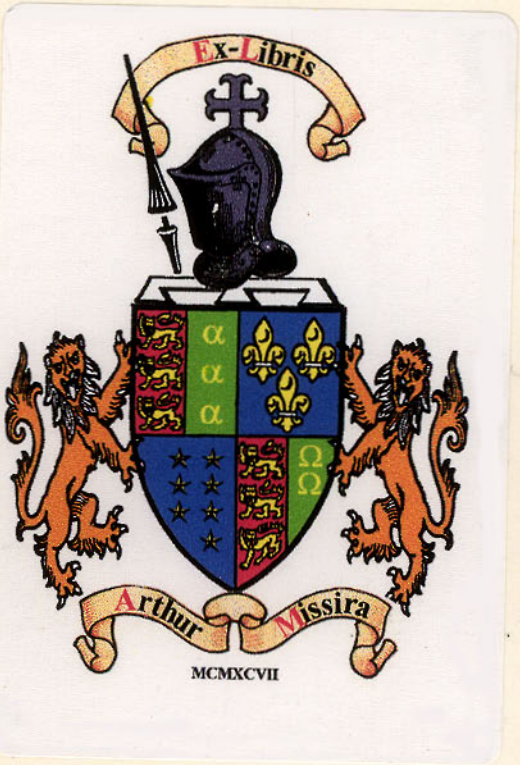
MICROPROCESSOR INTERFACING TECHNIQUES

AUSTIN LESEA
RODNAY ZAKS

SYBEX

Microprocessor INTERFACING Techniques


Ex-Libris
α α α
Ω Ω
Arthur Missira
MCMXCVII

MICROPROCESSOR INTERFACING TECHNIQUES

AUSTIN·LESEA

RODNAY ZAKS

SYBEX

Published by: SYBEX Incorporated
2161 Shattuck Avenue
Berkeley, California 94704

In Europe: SYBEX-EUROPE
313 rue Lecourbe
75015-Paris, France

DISTRIBUTORS
L. P. ENTERPRISES
313 KINGSTON ROAD
ILFORD, Essex. IG1 1PJ
Tel: 01-553 1001

$9.95 (USA)
FF66 (Europe)

FOREWARD

Every effort has been made to supply complete and accurate information. However, Sybex assumes no responsibility for its use; nor any infringements of patents or other rights of third parties which would result. No license is granted by the equipment manufacturers under any patent or patent rights. Manufacturers reserve the right to change circuitry at any time without notice.

In particular, technical characteristics and prices are subject to rapid change. Comparisons and evaluations are presented for their educational value and for guidance principles. The reader is referred to the manufacturer's data for exact specifications.

Library of Congress Card Number: 77-20627
ISBN Number: 0-89588-000-8
Printed in the United States of America
Printing 10 9 8 7 6 5 4 3 2 1

CONTENTS

TABLE OF CONTENTS

PREFACE

Computer interfacing has traditionally been an art, the art to design and implement the required control electronics for connecting a variety of peripherals to the main processor.

With the advent of microprocessors, and of LSI chips, since 1976, microprocessor interfacing is no longer an art. It is a set of techniques, and in some cases just a set of components. This book presents the techniques and components required to assemble a complete system, from a basic central processing unit, to a system equipped with all usual peripherals, from keyboard to floppy-disk.

Chapters two and three are a recommended reading for every designer who has not had the experience of designing a basic system. Chapter two presents the construction of a basic CPU, in the case of popular microprocessors such as the Intel 8080, 8085, and the Motorola 6800. Chapter three presents the set of input-output techniques used to communicate with the external world, and a brief survey of the existing chips which facilitate the implementation of these techniques.

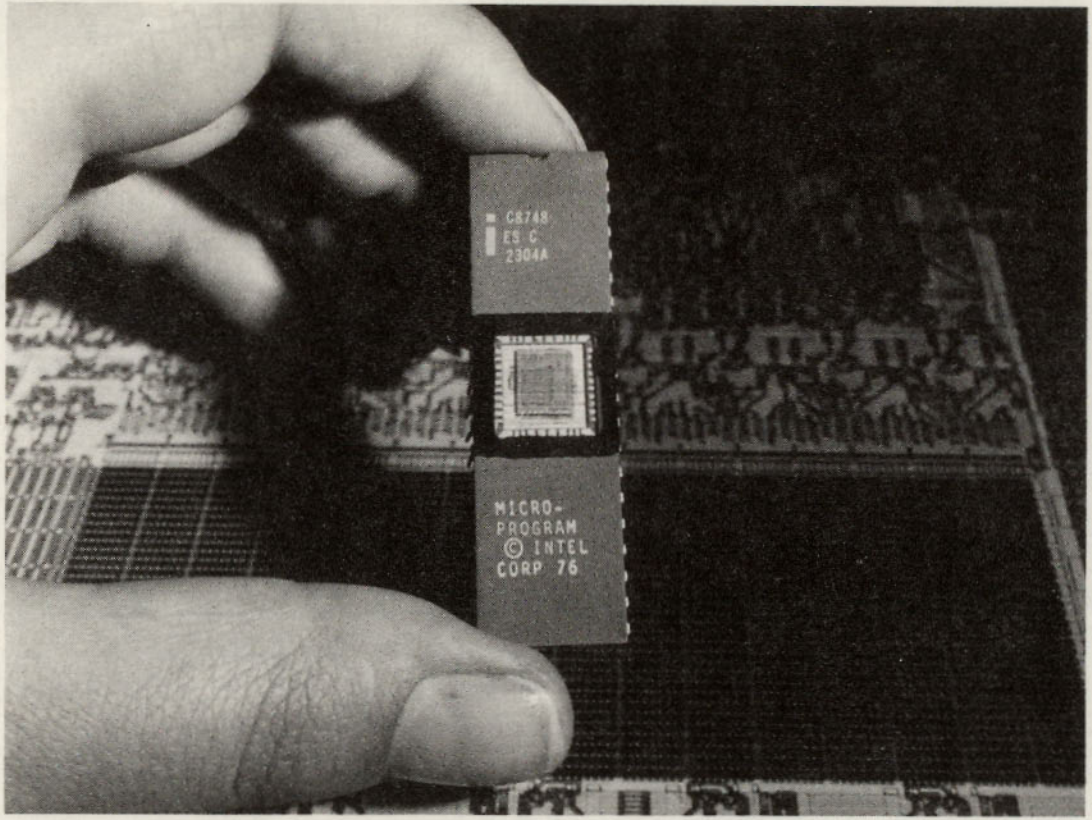
Chapter four is an essential chapter: the microprocessor-based CPU will be successively interfaced to every major peripheral: keyboard, LED, teletype, floppy-disk, CRT display, tape-cassette.

The following chapters then focus on specific interfacing problems and techniques, from industrial design (analog-to-digital conversion) (chapter five) to communication with the outside world (busing, including S-100 and other bus standards), in chapter six.

Chapter seven presents a detailed case study, which incorporates the interfacing principles presented in the previous chapters: the design of a real 32-channel multiplexer.

Finally, chapter eight presents the basic techniques and tools for trouble-shooting microprocessor systems.

This book assumes a basic understanding of microprocessor systems, equivalent to the level of book **C201 - Microprocessors: from chips to systems.**


C8748
ES C
2304A
MICRO-
PROGRAM
© INTEL
CORP 76

CHAPTER 1
INTRODUCTION

OBJECTIVE

The objective of this book is to present the complete set of techniques required to interface a microprocessor to the external world. Because of the availability of new LSI interface chips, which implement most techniques in hardware, it will be shown that interfacing has become simple.

FROM ART TO TECHNIQUE

Microcomputer interfacing has traditionally been the art of designing complex boards of logic managing the data transfers and the synchronization signals necessary for the processor to communicate with external devices. The processor itself has traditionally required one or more boards of logic. Each I/O interface has traditionally required on or more boards of boards. Such Multi-board implementations are obsolete today in most cases. *Large scale integration* (LSI) has now resulted in the implementation of a complete (or almost complete) CPU in *a single chip*. The new market created by microprocessors has introduced, in turn, the necessity for manufacturers to provide the required support components. Most of the boards required to assemble a complete system have now been shrunk into LSI chips. Since 1976, even device-controller interface chips exist. They do for interface design what the microprocessor has done for CPU design.

A complete interface board, or most of it, is today shrunk into a few LSI chips. The price paid, just like in the case of a microprocessor, is that the architecture is frozen inside the LSI chip.

It is now possible to implement a complete microcomputer system, including interfaces, in a small number of LSI chips. *If you are still implementing your interfaces on one or more boards of logic, your design might be obsolete!*

Microprocessor interface-chips have not reached their maturity yet. They are still "dumb" chips. In other words, they can execute only a very few commands. It can be predicted that in view of the very low cost of a processing-element, most microprocessor interface chips will become fully programmable in the near future. They will become "processor-equipped", and be capable of sophisticated programmed sequencing. They will become "intelligent" interfaces.

Although this next step has not been reached yet, all the techniques presented within this book should retain their validity in the future. There is always a trade-off between software and hardware implementation. The balance will change with the introduction of new components, and with the trade-offs involved in each specific system design.

THE HARDWARE/SOFTWARE TRADE-OFF

Detailed techniques will be presented to solve all the common interfacing problems. As usual in computer design, most of these techniques may be implemented either by *hardware* (by components), or by *software* (by programs), or by a combination of both. It is always up to the system designer to strike a reasonable compromise between the efficiency of hardware, and the lower component count of a software implementation. Examples of both will be provided.

THE STANDARD MICROPROCESSOR SYSTEM

Throughout this book, reference will be made to a "standard microprocessor." The "standard" microprocessor today is the *8-bit microprocessor*. Examples are the Intel 8080, 8085, the Zilog Z-80, the Motorola 6800, the Signetics 2650, etc. In view of the pin number limitation on DIP's (dual-in-line packages), the 8-bit microprocessor has become the norm. The reason is simple:

The number of pins is limited to 40 (or 42) by economic considerations. Industrial testers required to test components having more than 40 pins are either not available, or would be extremely expensive. All standard testers will accept only up to 40 or 42 pins. In addition, naturally, the cost of the package itself increases rapidly over 40 pins.

Because of the limitation of the densities which can be achieved with the MOS LSI process, it is not yet possible to integrate the complete memory, plus I/O facilities directly on the microprocessor chip. In the standard system, the microprocessor itself (abbreviated MPU), and perhaps the clock, reside on a single chip. The memory (ROM, or Read-Only Memory, and RAM, or Random-Access Memory) are external. Because memory and I/O chips are external to the microprocessor, a selection mechanism must be provided to address the components: a microprocessor must be equipped with an *address-bus*. The standard width of the address-bus is 16 bits, permitting the addressing of 64 K locations (where K = 1,024: 2^{16} = 64K).

An 8-bit microprocessor will transfer 8-bit data. It must be equipped with an 8-bit *data-bus*. This requires 8 additional pins.

At least two pins must be provided for power, and two more for connection to an external crystal or oscillator. Finally 10 to 12 control lines must be provided to provide the coordination of data transfers in the system (the control-bus). The total number of pins used is 40. No pins are left unused.

Because of this pin-number limitation, a 16-bit microprocessor cannot provide at the same time a 16-bit address-bus, and a 16-bit data-bus. One of the buses must be *multiplexed.* This results in turn into a slower operation, and in the necessity of external components to multiplex and de-multiplex the buses.

It can be expected that the progress of integration will soon introduce a new standard microprocessor, the *16-bit microcomputer-on-a-chip.* A microcomputer-on-a-chip is a microprocessor-plus-clock-plus-memory (ROM + RAM) on a single chip. Since the memory is directly on the chip, there is no longer the necessity to provide an external address-bus. 16 pins become available. In such a system, *at least 24 lines become available for data transfers.* They are general-purpose I/O lines. The disadvantage of current microcomputers is that, for the time being, the quantity of memory which may be implemented directly on the microcomputer-chip is limited. The current limitation is 2048 words for the ROM, and 512 words for the RAM. Adding external memory involves complex multiplexing and de-multiplexing, and is usually not worth it. However, if a system can be implemented in the near future with a significantly larger memory, it can be expected that it will become the next standard design.

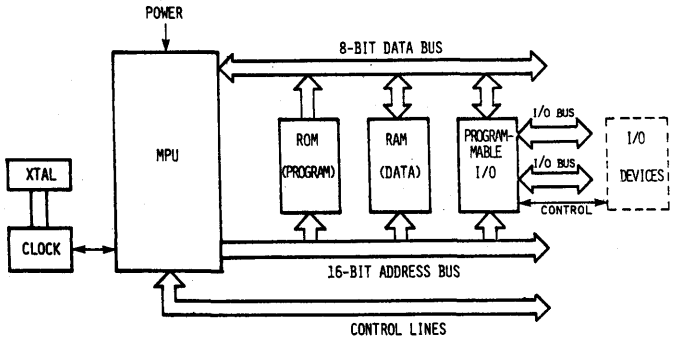


Fig. 1.1 Standard Microprocessor System

For the time being, the 8-bit microprocessor is indeed the standard design used for "powerful" and flexible applications, and will be referenced as

such. The basic diagram showing the architecture of a standard system appears on Fig. 1-1. The microprocessor itself, labeled MPU, appears on the left of the illustration. On most standard systems, up to 1976, the clock was external to the MPU. It appears here on the far left at the illustration. Since 1976, the clock circuitry has been incorporated in the microprocessor chip itself and all recent products do not require this external clock. However they always require an external *crystal* or oscillator. It appears here, connected to the clock.

The microprocessor creates *three buses:*

The 8-bit bi-directional *data-bus* (implemented in tri-state logic to allow the use of a direct-memory-access controller, or DMAC).

A 16-bit mono-directional *address-bus,* connected internally, within the microprocessor, to the address-pointers, and in particular to the program-counter (PC). The address-bus is also implemented in tri-state logic in order to allow the use of a DMAC.

Finally, a 10 to 12-line *control-bus*, which carries the various synchronization signals to and from the microprocessor. Control lines are not necessarily tri-state.

All the usual system components are directly connected to these three buses. The three basic components appear on the illustration. They are respectively the ROM, the RAM, and the PIO. The *ROM* is the Read-Only Memory. It stores the *programs*. The *RAM* is the Random-Access-Memory. It is a read-write MOS memory which stores the *data*. The PIO is a programmable input-output chip which multiplexes the data-bus into two or more input-output ports. It will be studied in more detail in chapter three. These ports may be connected directly to input-output devices, or to device-controllers, or may require the use of interface-circuits.

The interface-circuits or *interface-chips* required to interface this basic system to actual I/O devices will be connected to these buses, whether the microprocessor buses or the input-output buses created by the PIO, or by other chips.

Interfacing techniques are precisely those techniques required to connect this basic system to the various input-output devices. The basic interfacing techniques required to connect any microprocessor system to input-output devices are essentially identical. They will be described in detail in chapters three, four, and five. At the level of the microprocessor itself, the logical and electrical interface required is simple. All standard microprocessors have essentially the same data-bus and the same address-bus. The essential difference is the *control-bus*. It is the specific characteristics of the control-bus which make input-output interface chips compatible or incompatible from one microprocessor to the next. As an example of basic interfacing characteristics, the basic 8080, 6800, and SC/MP, interfacing characteristics appear on Fig. 1-2. A more detailed listing of signal equivalences

appears on Fig. 1-3.

	8080	PACE	6800	SC/MP
ADDRESS WORD LENGTH	16-BIT	16-BIT DATA/ ADDRESS BUS	16-BIT	12- OR 16-BIT
DATA WORD LENGTH	8-BIT		8-BIT	8-BIT
ADDRESS & DATA POLARITY	ALL CHIPS ARE 1 = TRUE; HOWEVER, IF INTEL SYSTEM BUS DRIVERS AND RECEIVERS ARE USED, 8080 SYSTEM DATA AND ADDRESS BITS ARE Ø = TRUE.			
ADDRESS STROBE	NONE	NADS = Ø TO SET MEMORY ADDRESS LATCHES	VMA = 1 (CONCURRENT WITH)	NADS = Ø
MEMORY READ STROBE	MRDC = Ø TO READ DATA	IDS = 1 TO INPUT DATA	R/W = 1 TO READ DATA	NRDS = Ø TO READ DATA
MAXIMUM CLOCK RATE	2MHz	2MHz	1MHz	1MHz

Fig. 1.2 Basic Interfacing Characteristics

Interfacing input-output devices requires the understanding of two basic techniques:

1. The assembly of a complete CPU, using a microprocessor chip. This topic will be addressed in chapter 2.
2. The fundamental input-output techniques used to communicate between the microprocessor and the external world. This topic will be addressed in chapter 3.

MICROPROCESSOR CONTROL SIGNALS

It has been shown that a standard MPU creates three busses: the 8-bit bi-directional data bus, the 16-bit mono-directional address bus, and a control-bus of varying width, depending on the microprocessor. The data-bus is essentially identical for all microprocessors. It is 8-bit-bi directional bus, normally implemented in tri-state logic. Similarly, the address-bus is almost universally a 16, or sometimes 15-bit mono-directional bus, used to select a device external to the MPU. The actual use and interconnect of the address-bus and the data-bus will be presented in the next chapter. The third bus is the only complex one. It carries the microprocessor control signals or "interface signals."

APPROXIMATE BUS SIGNALS EQUIVALENCES

	8080 & 8228	8085	Z80	M6800	MC56502
ADDRESS	AØ to A15	ADØ to AD7 + A8 to A15 and ALE	AØ to A15	AØ to A15	ABØ to AB15 not tristate
DATA	DØ to D7	ADØ to AD7 and $\overline{ALE}$	DØ to D7	DØ to D7	DBØ to DB15
CONTROL	HLDA HOLD Ø2 INT INTE WAIT READY RESET SYNC INTA MEMR MEMW I/O R I/O W BUSEN STATUSSTR	HLDA HOLD CLK INTR -- -- READY RESET IN -- INTA RD and IO/M WR and IO/M RD and IO/M WR and IO/M -- --	BUSAK BUSRQ -- INT -- -- WAIT RESET M1 M1 and IORQ RD and MREQ WR and MREQ RD and IORQ WR and IORQ -- --	BA and VMA HALT Ø2 stretched IRQ -- -- -- RESET -- VMA and (AØ-A15)=FFF8 R/W and Ø2 R/W and Ø2 R/W and Ø2 R/W and Ø2 HALT --	-- RDY Ø2 stretched IRQ -- -- RDY RESET SYNC -- R/W and Ø2 R/W and Ø2 R/W and Ø2 R/W and Ø2 -- --
OTHER CONTROL SIGNALS	-- -- -- -- -- -- -- -- -- -- --	RST 5.5 RST 6.5 RST 7.5 TRAP RESET OUT SID SOD ALE -- -- -- -- --	-- -- -- NMI -- -- -- -- RFSH HALT -- -- --	-- -- -- NMI -- -- -- -- -- -- -- TSC DBE --	-- -- -- NMI -- -- -- -- -- -- -- -- -- SO

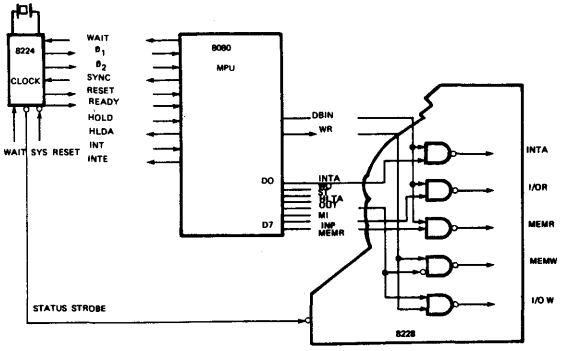
Fig. 1.3 Signal Equivalences

The control bus provides four functions:

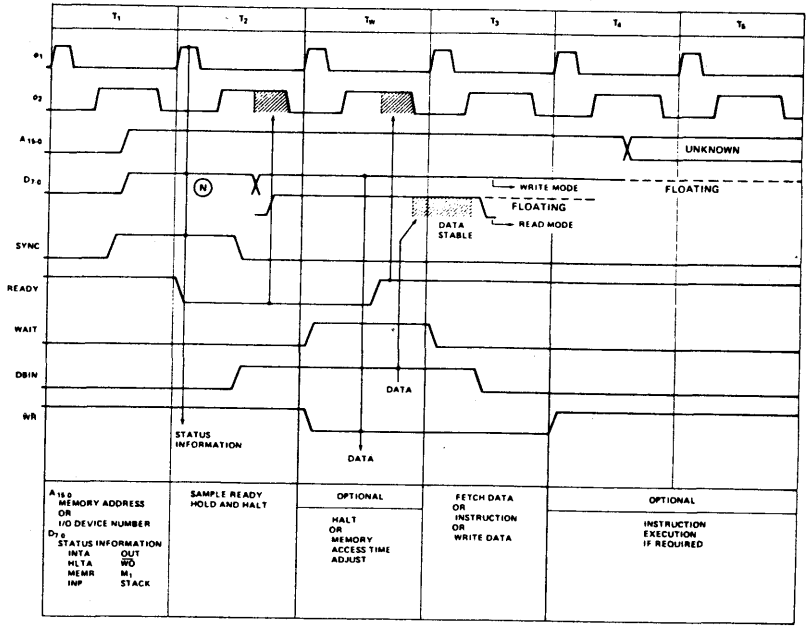
1. memory synchronization
2. input-output synchronization
3. MPU scheduling -- interrupt and DMA
4. utilities, such as clock and reset.

Memory and input-output synchronization are essentially analogous. A hand-shake procedure is used. In a "read" operation, a "ready" status or signal will indicate the availability of data. Data will then be transferred on the data-bus. In the case of some input-output devices, an "acknowledge" is generated, to confirm the receipt of data. For "write" operation, the availability of the external device is verified through a status-bit or signal, and the data is then deposited on the data-bus. Here also an "acknowledge" might be generated by the device to confirm the receipt of data.

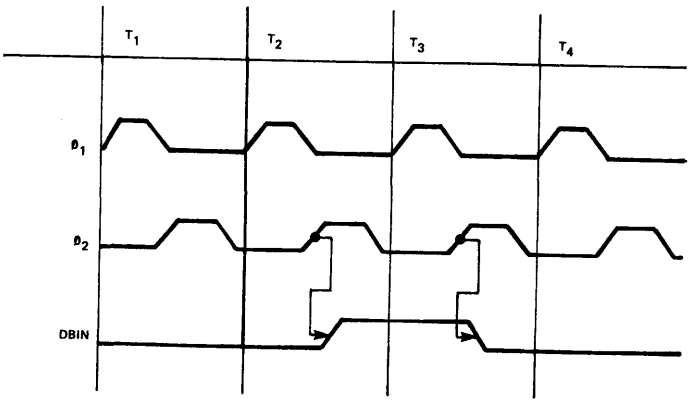
The generation, or non-generation, of an "acknowledge" is typical of the use of the synchronous procedure versus an asynchronous one. In a synchronous procedure, all events take place within a specified period of time. In this case there is no need to acknowledge. In an asynchronous system, an acknowledge must be generated. The choice of a synchronous versus an asynchronous communication philosophy is basic to the design of a control bus. A synchronous design has a potential for a higher speed and a lower number of control lines. However it imposes speed constraints on the external devices. An asynchronous design will require an additional acknowledge, and somewhat more logic, but allows the use of components of varying speeds in the same system.



8080 Control Signals

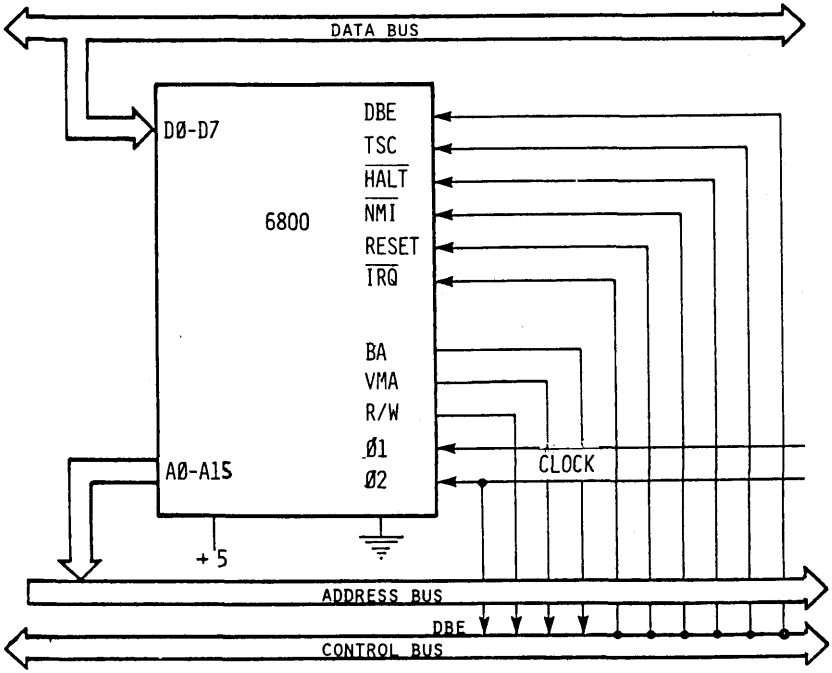


Basic 8080 Instruction Cycle

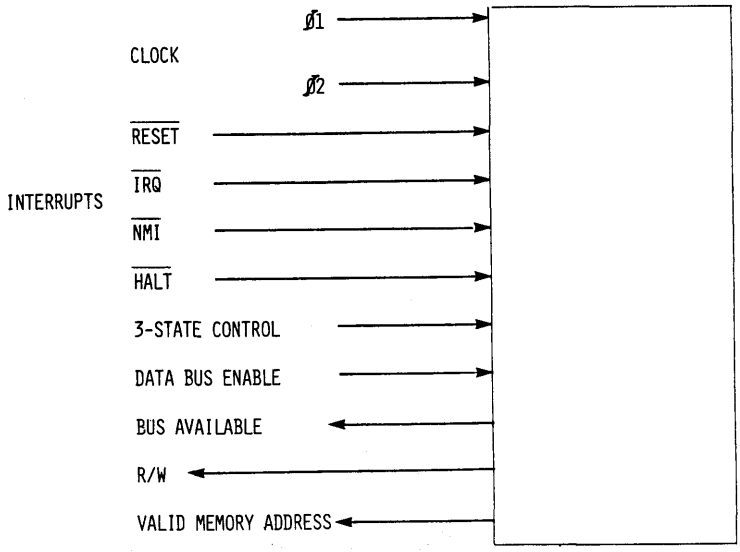


DBIN IS TRIGGERED BY Φ_2

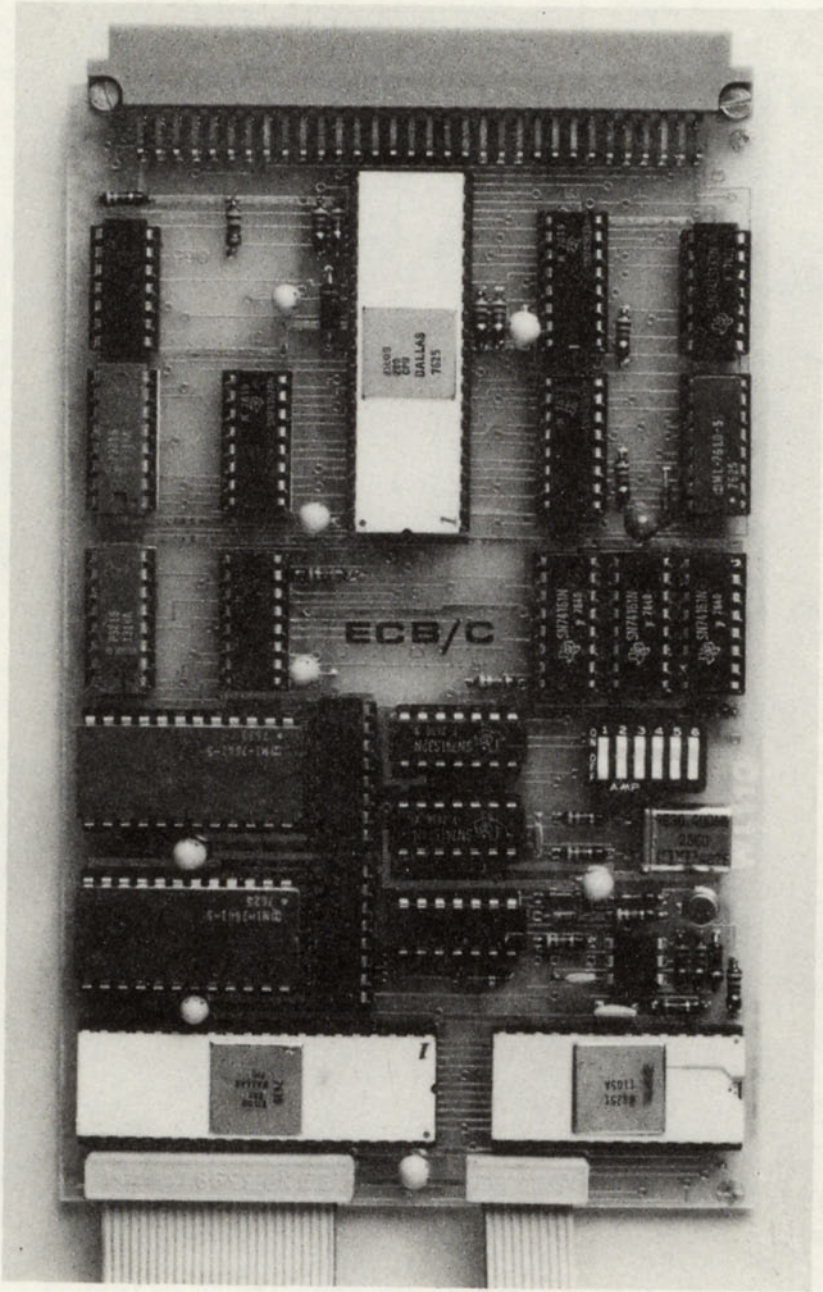
DBIN Timing



6800 Bus Signals



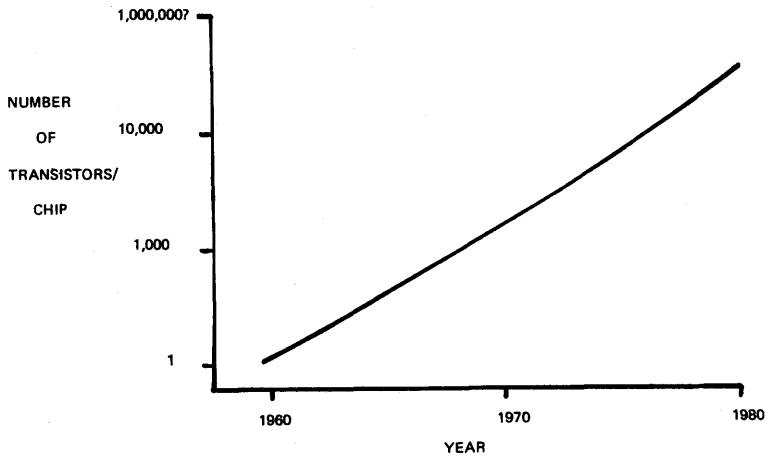
Detail: 6800 Bus Control


ECB/C
DALLAS
7625
AMP

CHAPTER 2
ASSEMBLING THE CENTRAL PROCESSING UNIT

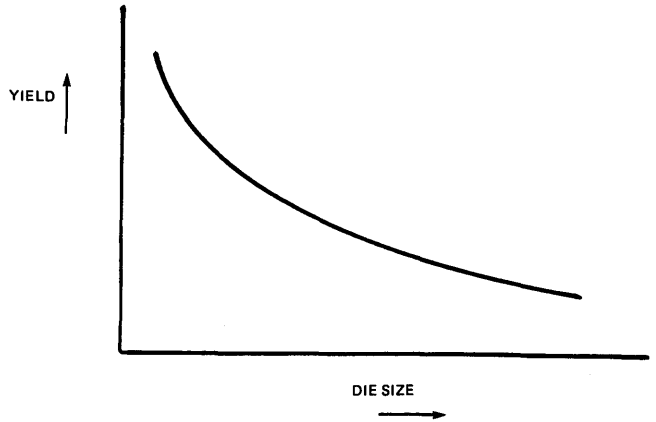
INTRODUCTION

The heart of any microprocessor system is the *central processing unit or CPU*. A CPU includes the microprocessor, plus any additional components it may require. Memory devices, buffers, decoders, clock-drivers are all included in the typical central processing unit. Many of these circuits are now being integrated on the same chip as the processor. In fact, since 1976, one-chip microcomputers are a reality. Yet, even with the advent of one-chip microcomputers, there still exist certain limitations on integrated circuit fabrication. There are three basic limits of the present LSI technology: *yield* limits the number of transistors per chip, *packaging* limits the number of pins on the package, and *substrate material* prevents some devices from being integrated.



2-1 Devices Integrated Versus Time

At first, only single transistors were made on each chip. Later, differential pairs, and simple logic gates made their appearance. Present technology allows for up to 30,000 devices to be integrated on a chip. A graph of devices integrated versus time appears in Fig. 2-1. One factor has remained constant throughout this process: process defects limit the maximum size of the individual die. *Yields* are higher for smaller die sizes. (The yield is the number of good devices per batch). In the design of any LSI chip, the "real-estate" (chip-area) becomes an all important factor affecting the cost of the final device. Fig. 2-2 illustrates the trade-off between yield and die-size. Yields also increase with manufacturing experience—this is called the "learning-curve": costs decrease with higher quantities, because of improved yield.



2-2 Yield versus Die Size

A less obvious factor is the *packaging* of LSI devices. Present testing equipment cannot handle packages with more than 40 pins. Future test systems may overcome this limitation, but, for now, the scarcity of package-pins may require the use of multiplexing techniques: the data-bus may also be used to carry *address* or *control* information so that pins may be conserved (ex: 8080, 8085).

How does the substrate material limit LSI technology? Certain components require a different physical material. The simplest example is the

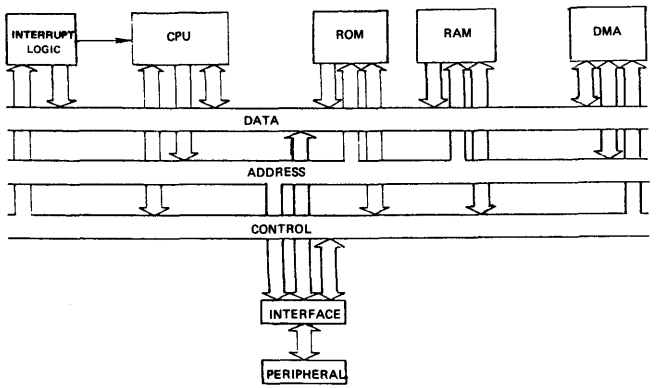
crystal required for timing. A crystal is cut from quartz. The integrated circuit is made from silicon. All systems requiring accurate timing will require a crystal. Because of its bulk, the crystal is external.

In addition to the fact that limitations of LSI technology partition our system into multiple components, additional devices are often needed for system expansion. Large microprocessor systems require a significant amount of "support-logic".

This chapter will present the concepts, techniques, and components required to build a complete CPU: from system architecture to support logic. Four typical systems will be presented, using the 8080, 6800, Z-80, and 8085 microprocessors.

SYSTEM ARCHITECTURE

Fig. 2-3 presents the block-diagram of a typical microprocessor system. All standard microprocessors, such as the 8080 or the 6800, have a similar architecture. Three buses connect the systems' components: data; address; and control-bus.



2-3 Typical System Architecture

The *data-bus* carries information to and from the processor element. It carries the instructions fetched from memory, the data-input from input devices, the data stored into memory, and the data-output going to the output-devices.

To specify where the data are going, or where they are coming from, the

address-bus is used. It selects a location in memory or a register of an input-output device.

The control bus is used to control the sequencing and nature of the operation being performed. The control-bus indicates in particular the type of operation to be performed: "read from memory to the processor," "write to memory from the processor," "read from an input-device to the processor," or "write to an output-device from the processor." Additionally, interrupt, direct memory access, and other control functions are carried by lines of the control bus to implement the scheduling and synchronization of events.

Our standard microprocessor has 8 data-lines, 16 address-lines, and at least 8 control-lines. 8 data-bits form a *byte*. The byte is the basic unit of information in our standard system. Half of a byte is sometimes known as a *nibble*. The 16 address-lines allow for addressing of 65,536 (2^{16}) different memory locations or bytes. Two methods are used for selecting a memory location, or a device-register: linear selection, and fully-decoded selection.

Linear Selection

In the microprocessor-world, memory is partitioned into read-only-memory (ROM) for programs and fixed data tables, and random-access-memory (RAM) for data storage and temporaries, because of the volatility of MOS RAM's.

When more than one type of memory is used, the two types of memory are generally in separate packages. Also, the size of each will be considerably less than the full 65,536 possible locations available to our system. We must place each device in its proper place in our *memory map*. A memory map is the addressing plan for the address bus bits.

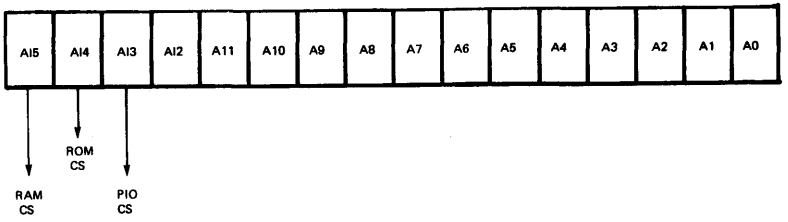
Initially, each device, RAM and ROM for our system, will have 256 locations. This implies that eight address lines will be needed to select one of the 256 possible locations in each chip. Besides these eight lines, the processor must be able to select one device at a time. RAM and ROM devices have, in addition to their address inputs, at least one "chip-select" (CS). This select-line, when activated, allows the operation to be performed on the device (Read or Write).

Two basic techniques are used to implement the chip selection: *Linear-selection* connects individual address lines to individual chip-select inputs. For example, if the most-significant address bit (bit 15) is tied to a chip-select, that chip is selected whenever the most-significant-bit is a one. This occurs for half of the total memory locations. Assume that our ROM is

selected by this most-significant-bit being "0" and the RAM by this bit being "1". To address the 256 locations available inside each device, we will connect lines A0 to A7 of the address-bus.

The essential advantage of linear-selection is simplicity: no special logic is necessary to select chips. Each new chip is selected by a dedicated address-line. This is, indeed, the approach used in all small microprocessor systems.

For example, a 1K × 8 ROM chip will be used and a 512 × 8 RAM, plus 3 peripheral chips. The 1K ROM requires 10 lines for address-selection: A0 - A9, plus one line for the chip-select: A14. The RAM will use A0 - A8 for address-selection, and A15 for the chip-select. Lines A12, A13, A14, A15 may be used for additional devices.



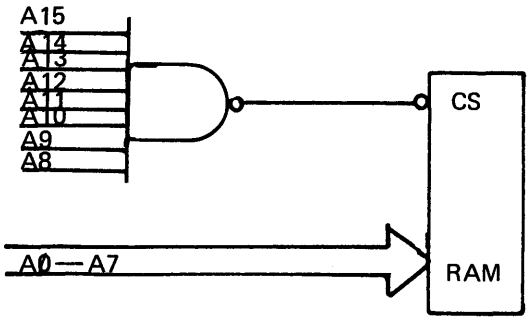
2-4 Linear Selection

However linear selection divides the available memory in half every time a separate address line is used. If the need exists to select more devices than there are available address lines, another method must be used: *fully-decoded addressing.*

Fully-Decoded Addressing

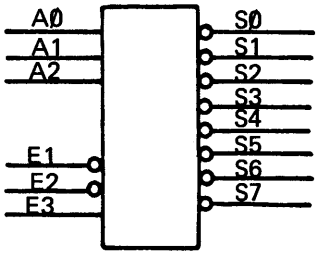
The goal of fully-decoded addressing is to provide a complete 64K addressing capability.

In our example, the 256-location RAM will reside in the last 256 locations of the memory. Expressed in binary, this is addresses 1111111100000000_2 to 1111111111111111_2. Grouping into four-bit groups and converting to hexadecimal this is: FF00 to FFFF. (See appendix for hexadecimal conversion table). We see that the RAM chip should be enabled when the 8 high-order address bits are equal to "1". "ANDing" these bits together would form our chip select. Fig. 2-5 illustrates the decoding for our example.



2-5 Fully Decoded Selection

Instead of using AND gates for every device, there exist general-purpose gating devices known as *decoders*. An example is the 8205 or 74LS138 three-to-eight decoder. The 8205 has three inputs to select one of eight mutually exclusive outputs, in function of three enable inputs. When the three enable inputs are in their proper states, one of the outputs will be active depending on the three select lines. Examples using the 8205 will be presented in the hardware section to clarify full-decoding schemes.



$$S0 = (\overline{A0} \cdot \overline{A1} \cdot \overline{A2}) \cdot (\overline{E1} \cdot \overline{E2} \cdot E3)$$

$$S1 = (A0 \cdot \overline{A1} \cdot \overline{A2}) \cdot (\overline{E1} \cdot \overline{E2} \cdot E3)$$

$$S7 = (A0 \cdot A1 \cdot A2) \cdot (\overline{E1} \cdot \overline{E2} \cdot E3)$$

2-6 8205 Decoder

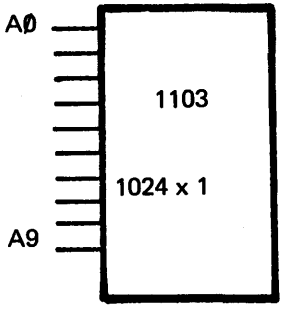
Complete-decoding selects devices without wasting available address space. A contiguous memory may be built where addresses pass from one device to the next without large areas of nonexistent or overlapping memory. The disadvantage of this approach is the cost of decoding. Most systems implement a mix of linear selection and partial decoding.

Storage Chips

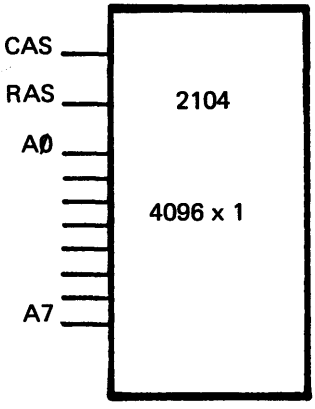
The basic devices for storing information now used are the RAM and the ROM. The ROM contains permanent information and *cannot be changed* by the system. The RAM allows for temporary storage and retrieval of information. The program information is usually kept in a non-volatile ROM since it does not change, and the data and intermediate results are stored in RAM.

"RAM" usually refers to a semiconductor device, but is also used for other storage media such as core memory.

A RAM chip may contain from 256 to 16,384 cells, each cell representing a bit of the information-word or byte. Each cell may consist of a flip-flop type structure—in which case it is a *static device*, or it may consist of a capacitor structure—in which case it is a *dynamic device*. The static RAM will retain information as long as power is present, whereas the dynamic device must be refreshed every few milliseconds in order to renew the stored charge in each cell. This means that dynamic memory will undergo a refresh cycle one to five percent of the time. This may be important in some



1103 uses 10 pins
18- PIN DIP



2104 user 10 pins
16- PIN DIP

Intel Dynamic RAMs

Address is multi-plexed

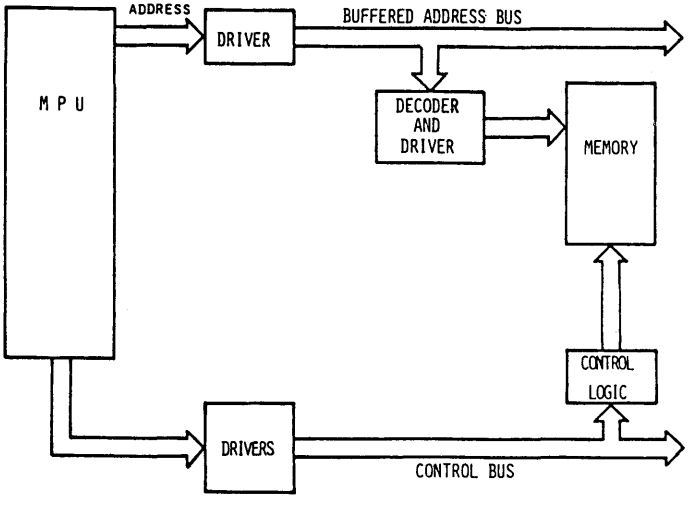
real-time applications as memory will be "busy" and unavailable for use as long as a refresh cycle is in progress.

ROM will refer here to an LSI device, but may also be used to denote other types of read-only memories. Several types of ROM's are available. The masked-ROM is "programmed" by the *manufacturer* and will stay programmed for the life of the chip. It cannot be altered. The PROM is programmed by the *user* and may either be of the fusible-link type, where a bit is programmed by blowing a microscopic fuse, or it may be a stored-charge type that will retain the pattern for tens of years. The latter type is also known as an EPROM because it can be erased by ultraviolet light and reused. The EAROM is electrically erasable and could be considered as RAM except that it takes 100 milliseconds or longer (typically) to erase the device. This makes it inconvenient to use as a scratchpad for calculations or data manipulations. The use of EAROM's has been restricted so far to military applications.

Buffering the Buses

Each input of a device presents a load on the output driving it. Most components drive anywhere from one to twenty other components. Every component must be checked for its input and output loading and driving characteristics.

The microprocessor's buses must connect to every memory and peripheral input-output chip in a system. All MOS microprocessors lack the

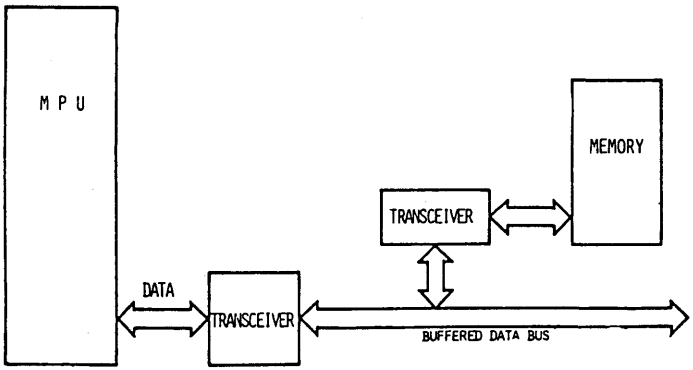


2-7 Buffering Address and Control Lines

output drive needed for a large system. Because of this, *buffers* or *drivers* are used to boost the driving power of the buses. There are bus *transmitters* for driving the bus, and bus *receivers* for listening to the bus and driving the processor.

Fig. 2-7 illustrates the use of transmitters to buffer the address and control-buses. The lines on the address and control buses are *unidirectional*: the data flows in one direction.

Fig. 2-8 illustrates the use of bus *transceivers* for the data-bus. Data must pass in both directions so both transmitters and receivers are used. The *bidirectional* data-bus will receive data and transmit data, depending on the function being performed.



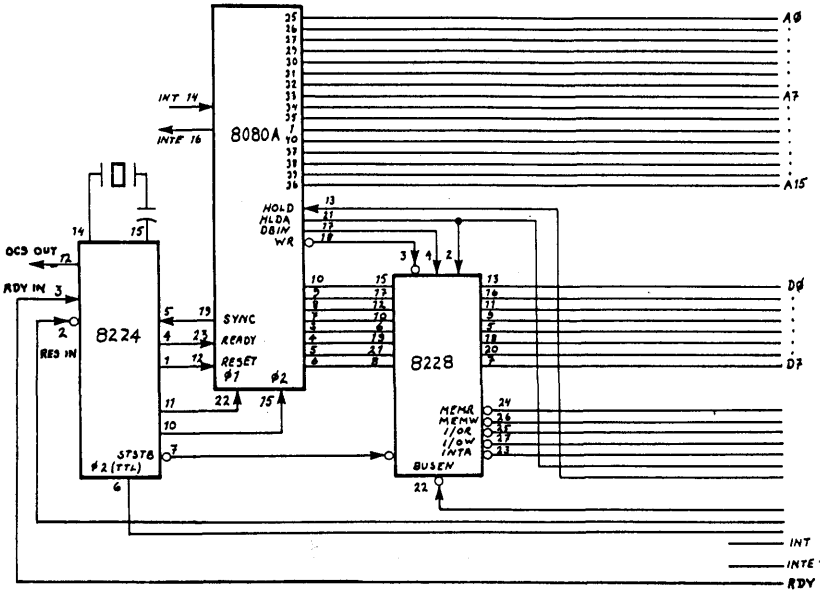
2-8 Buffering The Data Bus

The concept of a system architecture will be expanded and completed in Chapter 3 on input and output techniques. To clarify the concepts presented so far, four real systems will now be assembled: an 8080, a 6800, a Z-80 (with dynamic RAM), and an 8085 system.

THE 8080 SYSTEM

Intel's 8080 has been the most widely used "standard"-architecture-microprocessor. The 8080 is a popular processor also used in many hobby microcomputers. We will assemble the complete central processing module for a typical 8080 computer system. The connection of the: clock, system

controller, RAM, and ROM will be presented. The input-output will be covered in detail in Chapter 3.



2-9 8080 Completed CPU

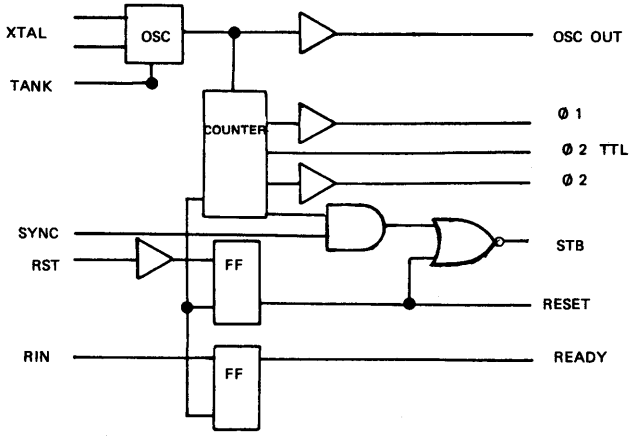
The Clock

The 8080 requires a two phase non-overlapping clock. This clock must swing between +11 volts and +0.3 volts. The clock is therefore not TTL-compatible. Initially, clock-drivers were made from discrete components or special-driving integrated circuits. Intel introduced the 8224 clock chip to reduce parts-count and simplify the clock interface problem. One merely connects the crystal to the 8224, the 8224 to the 8080, and all clock interfacing is complete.

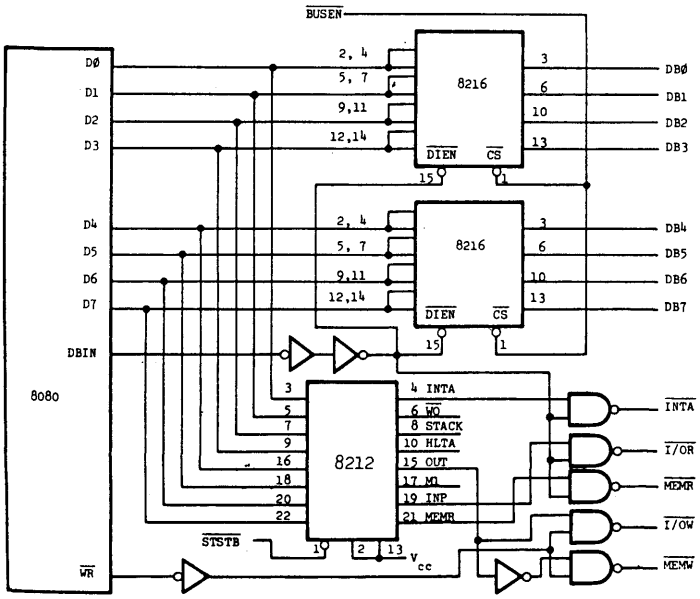
The connection of the 8224 appears in Fig. 2-9, and the structure of the 8224 itself appears in Fig. 2-10.

The System Controller

When designing the 8080, the lack of pins became a major limitation. In order to gate out the required control signals, pins have to be multiplexed.



2-10 8224 Schematic



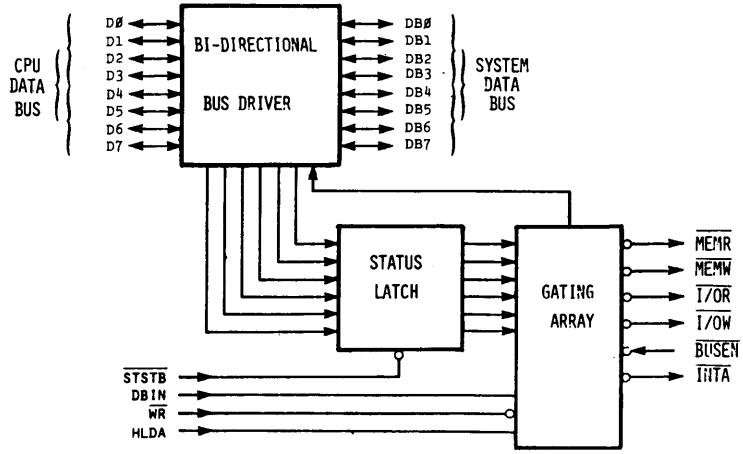
2-11 System Controller Using 8212 and 8216's

Control or address functions would have to share lines with the data-bus. In this case, the designers chose to multiplex control information or *status* on the data-bus. This status byte may be latched for use at the time of the SYNC signal. The lack of pins is essentially due to the early technology

used for the 8080, which required three power levels, using four pins.

Early processor designs used latches and random-logic to capture these status signals. In fact, this is why the actual S100 bus still retains what is known as the *old 8080 status signals*. The design of what became known as the *system-controller* appears in Fig. 2-11. The latch holds the status information and the gates decode the status along with the other 8080 control lines into control signals for the memory and input-output devices.

Intel, realizing early that the system-controller function should be integrated into a single-chip, introduced the 8228 chip, shown in Fig. 2-12. This device latches the status and drives the control bus. In addition, it buffers the data bus, i.e. includes a data-bus driver.



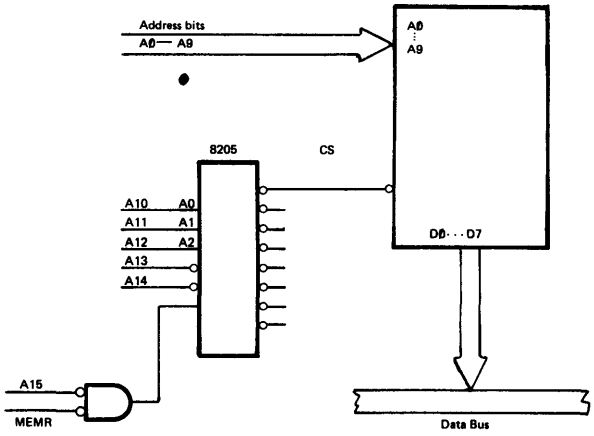
2-12 8228 System Controller

The trio of 8224, 8228, and 8080 now completes the central processor function. The only other component required is the crystal. To complete the CPU we need to add the program memory and the random-access memory (ROM and RAM).

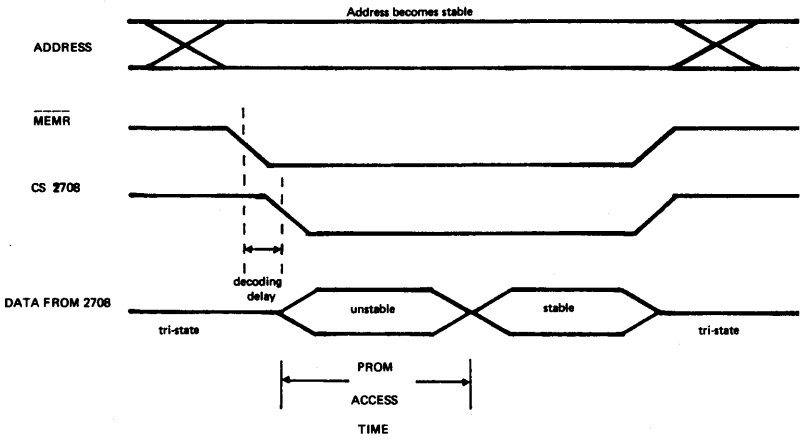
Connecting the ROM

Read-only memories come in two essential varieties: programmable and masked. The programmable ROM's may be programmed once at the time they are to be used, (such as fusible link ROM's or PROM's); or they may be programmed, used, and erased, (such as ultraviolet erasable ROM's or EPROM's). The mask-ROM's are programmed at the time of manufacture

and are used only in production systems. The erasable or fusible link ROM's are used for prototyping.



2-13 2708 Selection Using 8205



2-14 PROM Timing

A typical erasable ROM appears connected to our 8080 buses in Fig. 2-13. This device, a 2708 EPROM, contains 1024 bytes of memory. In order to address 1024 bytes, 10 address lines are needed, (2^{10} = 1024). In addition, the chip must be selected at its proper place in the memory map. We will

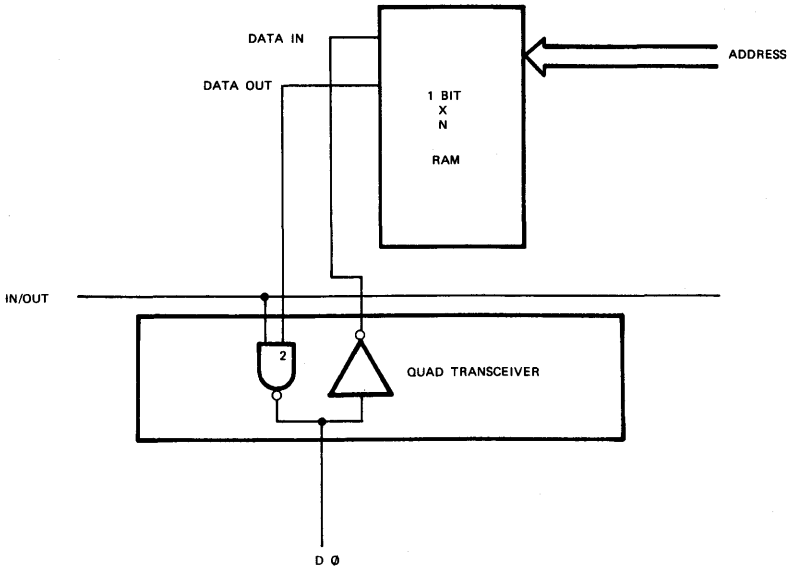
choose to put this memory at locations 0000 through 03FF hexadecimal. In order to decode this address space, an 8205 is used in addition to some other selection logic for controlling the memory read condition. Note that it can select up to seven additional, contiguously located, 2708's, if required. The data-bus connects directly to the data lines of the 8228 system-controller. The only control-line required is the memory read line. The timing of a memory read appears in Fig. 2-14.

The address and memory-read lines activate the 2708. After a period of time called access-time, the data byte fetched appears on the data-bus. The processor reads this byte and executes the instruction.

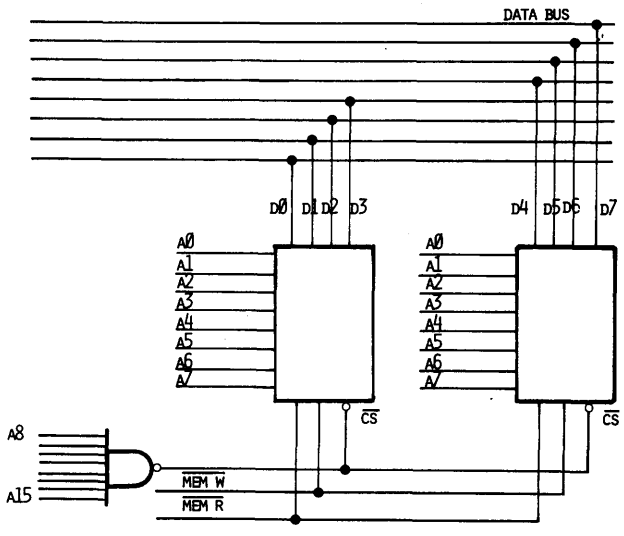
Connecting the RAM

A convenient size for the economical manufacture of ROM's is 1K by 8 bits (1K = 1024). RAM's, however, come in different sizes. The most inexpensive configuration is 1K by 1 bit (least number of pins). We need eight bits for a byte, so that eight devices are needed—one for each bit. Another popular size is 256 by 4 bits. This type of RAM is interfaced here.

256 by 4 implies that two devices are needed to complete the byte. The schematic for the 256 by 4 memories, interfaced to the 8080 bus, appears in Fig. 2-15.

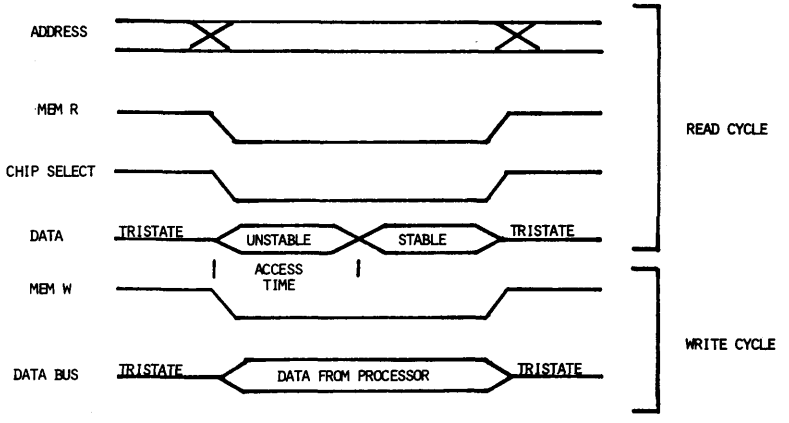


Buffing RAM data Lines Using a Bus Transceiver



2-15 Connecting the 2111 RAM

The address-bus lines needed to specify the address are connected to each RAM chip. The eight address-lines will select one of the 256 bytes in each RAM chip. The unused eight address lines are decoded by an eight input NAND gate. As per our earlier discussion, the RAM will be located from FF00 to FFFF hexadecimal. The data bus splits in two, with four bits going to each of the 256 by 4 bit RAM's. Control lines are needed to enable the memories for reading and writing as well as controlling the timing of the



2-16 RAM Timing

writing operation. The 2111 RAM's used here have a number of extra enable inputs, as well as a read/write line. The two signals: "memory-read," and "memory-write," are used to control the RAM's. "Memory-read" enables the output drivers of the chips to drive the data-bus. At all other times, the chip is in a read-mode, but will not place information on the bus. "Memory-write" enables the RAM to perform a write cycle and gates data presented on the data-bus into the RAM's. Timings of these operations are illustrated in Fig. 2-16.

When the address becomes stable and "memory-read" is brought low, the chip is enabled to drive the data-bus. After the byte is accessed, it remains on the bus until fetched by the processor and "memory-read" returns high. The write-cycle is similar, except that, this time, "memory-write" is brought low, forcing the data-bus contents to be written into the RAM's.

Integrating the processor and memory into an assembled module requires only that we draw them all on the same schematic.

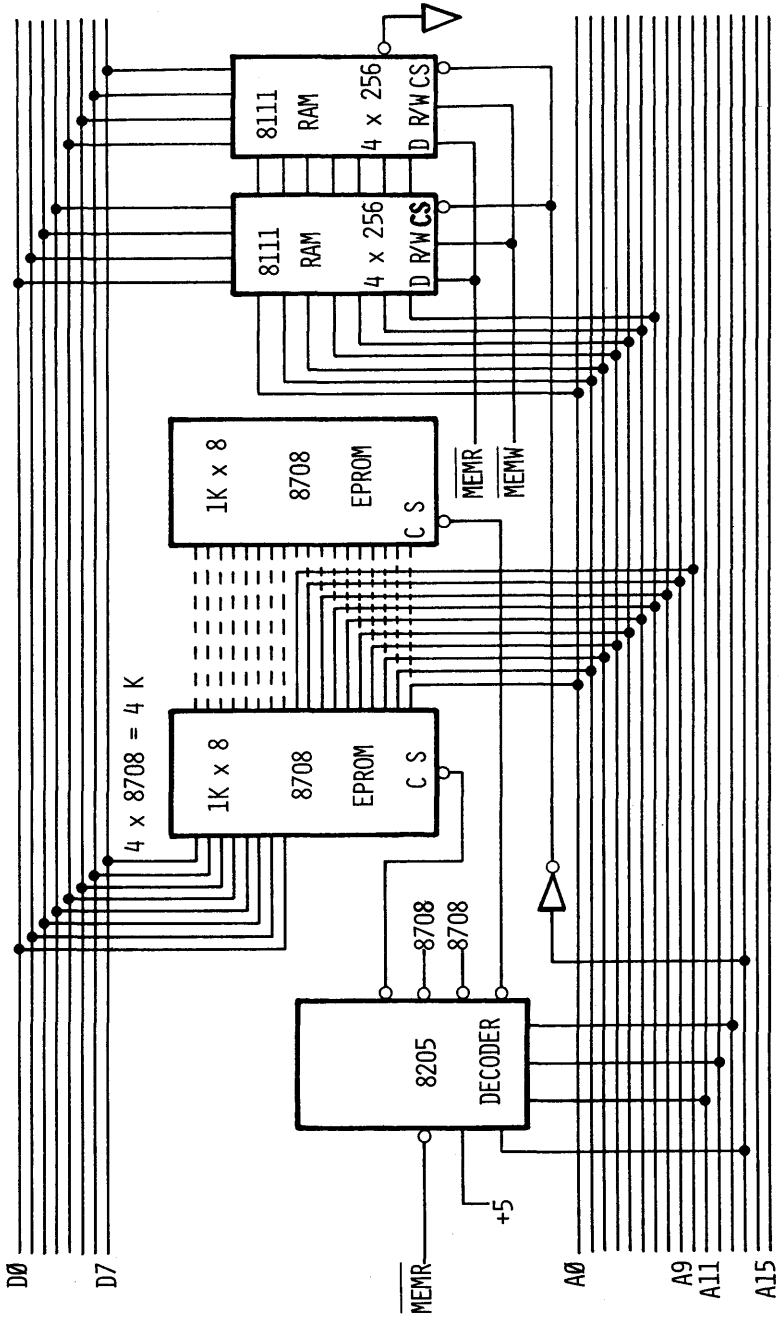
The Complete 8080 System

To make life more interesting, the system module presented here contains only *partial decoding* for the PROM's and *linear-selection* for the RAM's. The memory module appears in Fig. 2-17. The PROM's will occupy locations 0000 through 0FFF hexadecimal. The RAM will be at 2000 through 20FF hexadecimal. It will also be addressed for all addresses of the form: 0XX1XXXXXXXXXXXX binary—where X is a one or a zero (don't care condition). The PROM is addressed for: XX0000000000000 through XX0111111111111 binary. We cannot add any other memory to this system without further decoding.

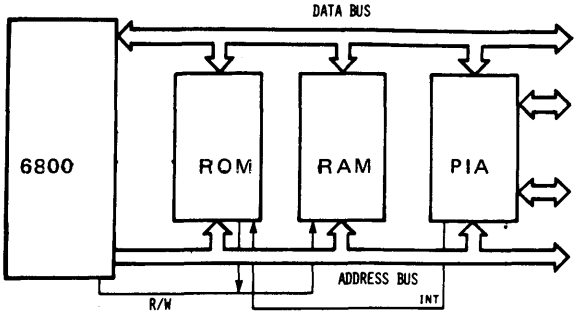
The central-processor module will be the same as in Fig. 2-9. As an exercise, the central processor assembly of Chapter 8 could be examined at this time and the reader should verify his/her understanding of address-decoding and buffering techniques.

THE 6800 SYSTEM

Developed by Motorola, the 6800 is also a popularly used "standard" type of microprocessor. In comparison to Intel's device, the 6800 implements some design philosophy differences. The most obvious are the lack of pin-multiplexing and the single power-supply requirement. Other differences lie in the instruction set, internal architecture, and control signals. Overall, both devices are essentially similar. Fig. 2-18 shows a schematic of a 6800 system.



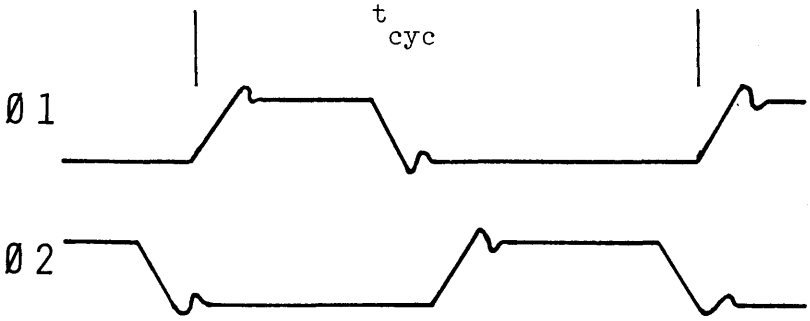
2-17 Complete 8080 System Memory



2-18 6800 System Block Diagram

The Clock

The 6800 requires a non-TTL compatible clock-generator. Since no other useful functions are needed for the two-phase clock generation, either simple discrete clock circuits, or integrated drivers are used. Motorola produces a hybrid device which contains the crystal and conveniently provides the necessary clock phases. Fig. 2-19 details the 6800 clock requirements.



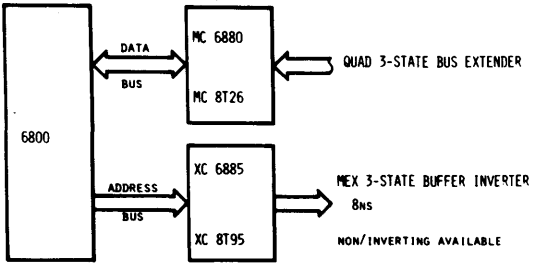
2-19 6800 Non-Overlapping Clock Signals

6800 Buses

The 6800 architecture uses memory-mapped input-output (see Chapter 3) and requires only a single power-level, versus three for the 8080. As a result, no multiplexing is required to gate the control signals. However, the buses need to be buffered in any large system, making the parts count

essentially equal between 8080 and 6800 systems. (The 8228 system controller includes a data-bus driver).

The data-bus is a bidirectional 8-bit bus. It requires buffering for most applications. The suggested Motorola components appear in Fig. 2-20.



2-20 Buffing 6800 Buses - Suggested Devices

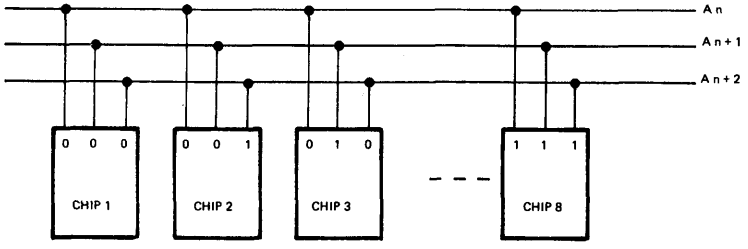
The address and control-buses are unidirectional with respectively 16 address lines, and ten control lines. Fig. 2-21 illustrates the 6800 bus signals. For memory interfacing, the R/$\overline{W}$, Φ, and VMA signals are required. They are the read/write control, phase two of the clock, and valid-memory-address control line.

- TSC HIGH FORCES ADDRESS BUS AND R/W INTO HIGH-IMPEDANCE MODE
- DBE LOW FORCES DATA BUS INTO HIGH-Z MODE
- R/W MPU IS IN READ MODE WHEN LOW
- VMA IS VALID MEMORY ADDRESS. A HIGH ENABLES RAM, PIA, ACIA
- $\overline{IRQ}$ IS INTERRUPT REQUEST LINE. PC IS LOADED FROM FFF8, FFF9
- $\overline{RESET}$ STARTS THE 6800 FROM POWER-DOWN. PC IS LOADED FROM FFFE, FFFF. 8 CYCLES ARE REQUIRED BEFORE
- $\overline{NMI}$ IS NON-MASKABLE INTERRUPT. PC IS LOADED FROM FFFC, FFFD
- $\overline{HALT}$ ALLOWS PROGRAM EXECUTION BY EXTERNAL SOURCE AND STEPPING
- BA (HALT OR WAIT) INDICATES THAT ADDRESS BUS IS AVAILABLE

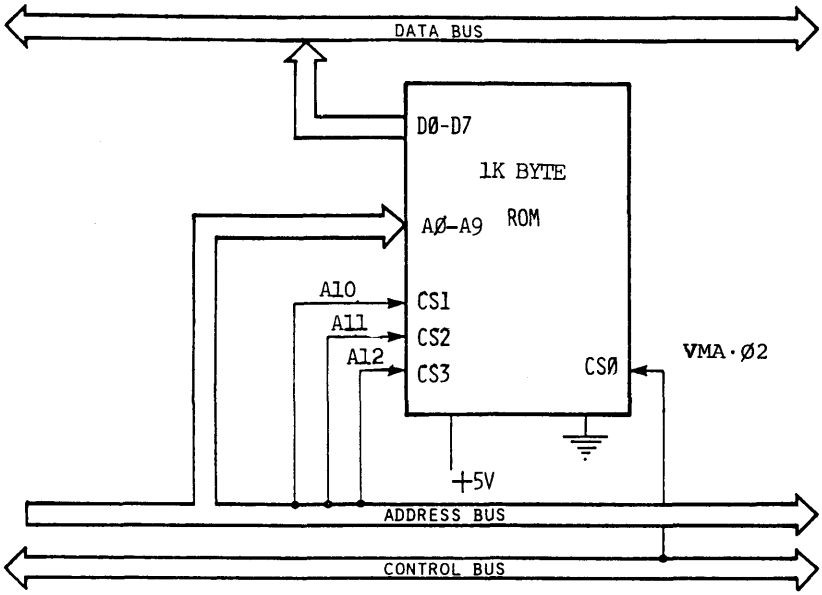
2-21 6800 Control Signals

The ROM

Motorola manufactures a line of 6800 compatible products which facilitate the interface requirements in small or medium-size systems. Their 1K by 8-bit mask ROM includes *four chip-select lines* for selecting the ROM.



3-Chip Selects Allow Connection of up to 8 Devices



2-22 6800 ROM Connection

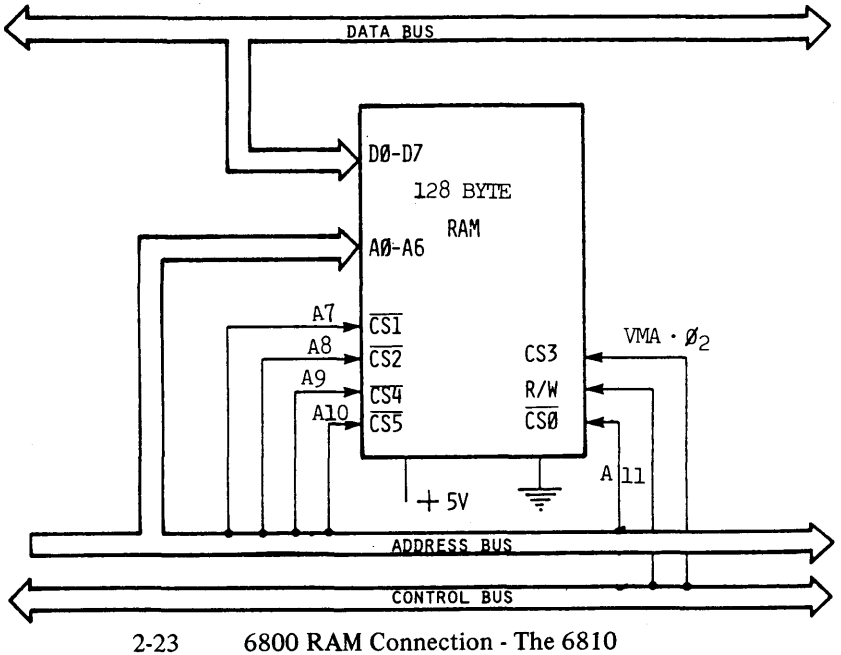
In the example of Fig. 2-22, the chip-selects are connected to three of the high-order address bits, and to the VMA signal ANDed with the Φ2 signal. In this way, the ROM is selected for any valid memory address cycle from

1C00 to FFFF hexadecimal. Of course, the ROM is only 1024 bytes, so the large area it takes up is due to the "don't cares" or the undecoded address bits: A15, A14, and A13. The essential advantage of providing the three Chip-Selects is to allow the possibility of connecting up to 8 devices to only 3 address lines: no external decoder is needed (see Fig. 2-24).

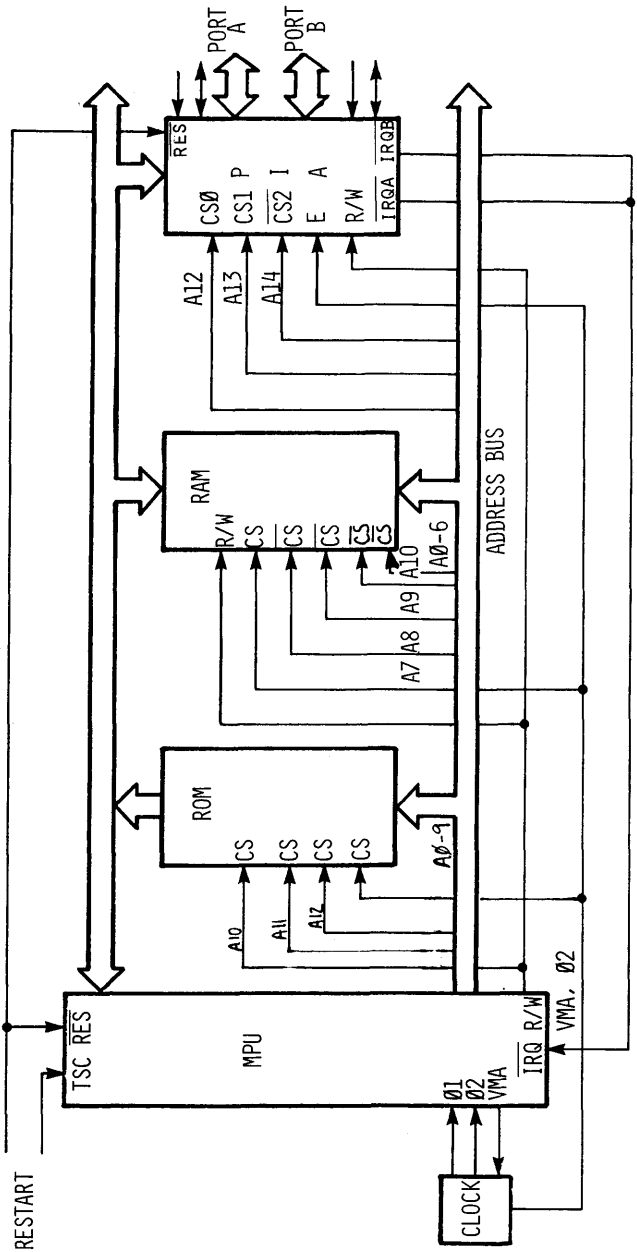
The RAM

Motorola is one of the few manufacturers that makes a 128 by 8-bit RAM. This is a convenient size for small systems. The interface to the 6810 RAM is aided by the large number of decoded chip-selects that are provided on the chip.

The interface of the RAM appears in Fig. 2-23. Note that only seven address lines are needed to select one of the 128 RAM bytes. The other 9 address lines must be used in some combination to select the chip. In this example, RAM is selected when A11 through A7 are all low. This would be address 0000 through 00FF hexadecimal. Since the highest four address bits are not fully decoded, the memory is also enabled for addresses 1000 through 10FF. Similarly, it is enabled for 2000 through 20FF, and so on, ending with F000 through F0FF.



2-23 6800 RAM Connection - The 6810



2-24 Completed 6800 System

In order to use our RAM with our ROM, we must select those places where the two do not overlap. One example is ROM from FC00 through FFFF and RAM from 0000 through 00FF.

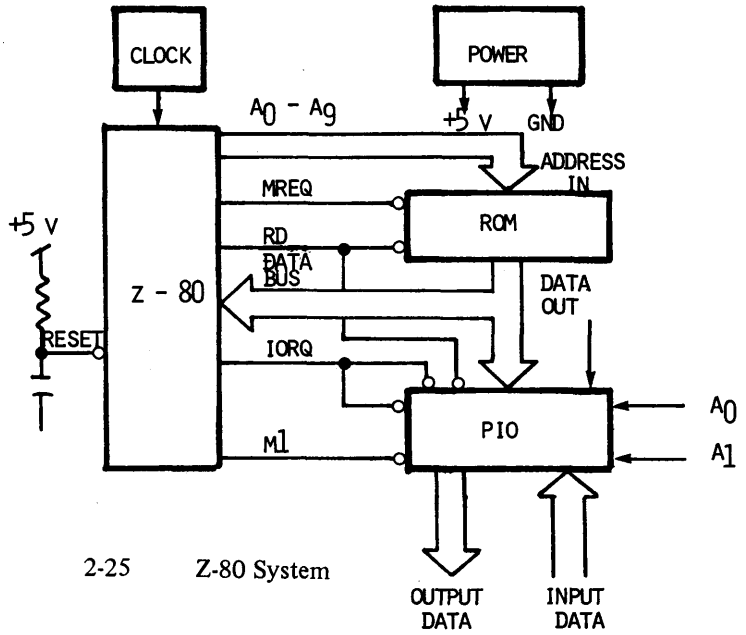
The VMA and Φ2 signals select the device for the memory cycle, and "read/write" controls the function: fetching or storing.

The Complete 6800 System

In Fig. 2-24, the complete 6800 system is presented. Note that an input-output device is included here. This will be explained in Chapter 3.

THE Z-80

Up to this point, the processors used were developed at about the same time. Zilog, created by the designers of the Intel 8080, was determined to improve the power of the original device. The Z-80 is software-compatible with the 8080. (In addition, it has some additional instructions and registers which improve its processing capability.) In particular, the Z-80 provides the necessary signals to interface with the larger *dynamic memory* devices. A Z-80 system appears in Fig. 2-25.



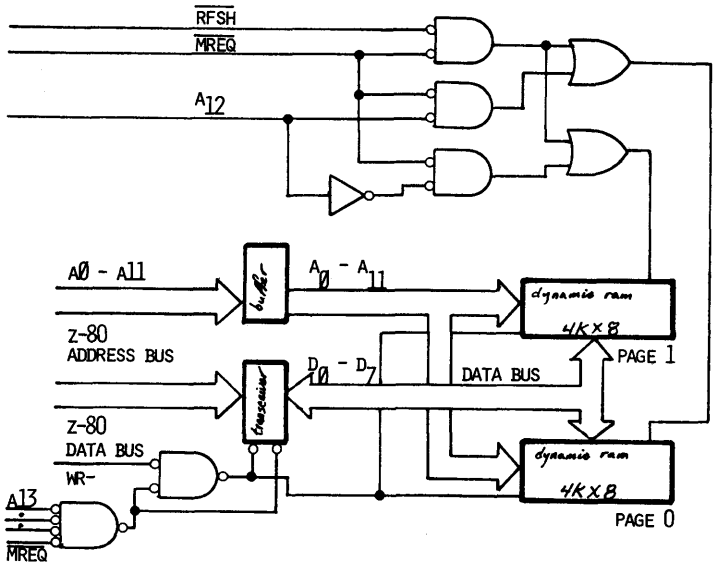
2-25 Z-80 System

Dynamic RAM Interface

In our previous examples, the memory devices used were *static* RAM's. With static RAM's data are retained as long as power is applied. *Dynamic* RAM's need to be *refreshed* periodically. A dynamic RAM stores information in a FET capacitor. Such a device can only retain its charge for a few milliseconds. The cell must be accessed every few milliseconds, in order to renew, or "refresh" the cell. The Z-80 provides the refresh address using a design trick.

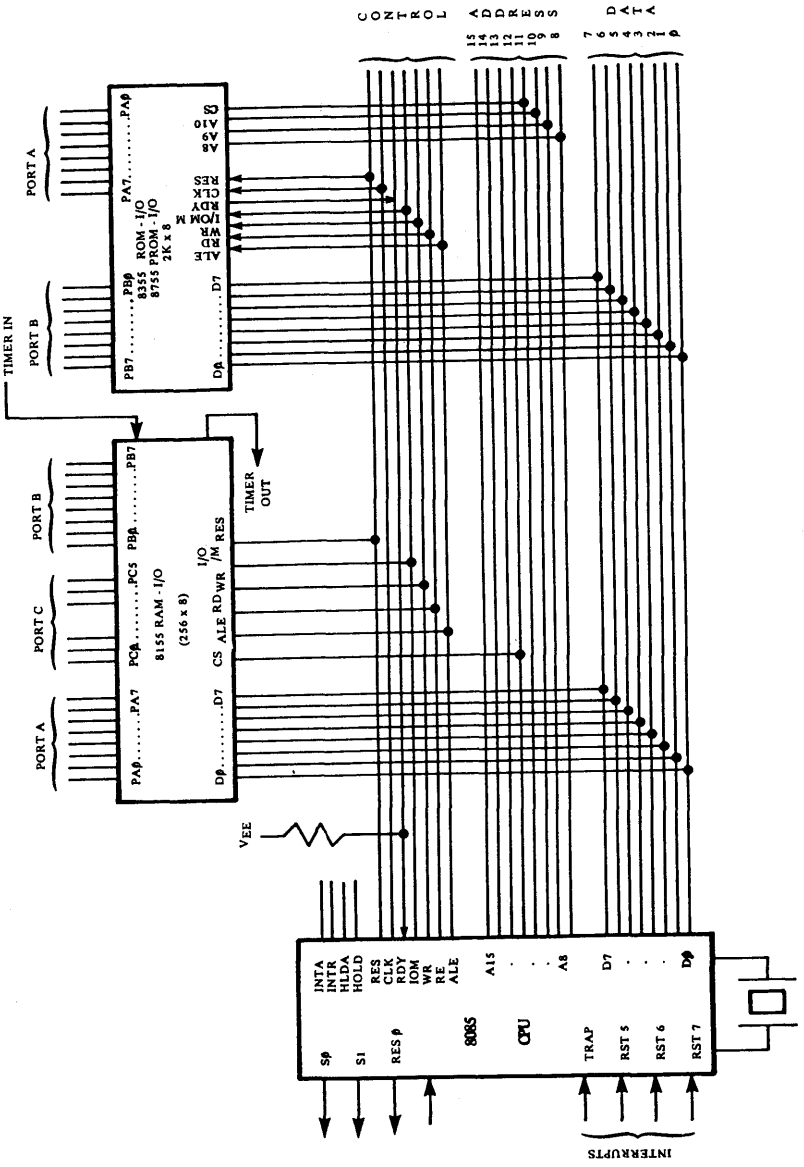
After an instruction is fetched, the address bus no longer needs to remain stable. Instead of wasting this time, the Z-80 outputs on the lower 7 address bits a *refresh address*. This address increments once each instruction-cycle. With this method of "stealing" a refresh cycle in every instruction cycle, and with the additional internal refresh register dynamic memories may be interfaced easily to the Z-80.

Otherwise, the processor would have to wait while a separate circuit, called the *refresh-controller*, stepped through the dynamic memory rows refreshing the cells.



2-26 Z-80 Dynamic Memory Interfacing

The dynamic memory interface appears in Fig. 2-26. Mostek, which second-sources Zilog, produces a single-board CPU, with 16K bytes of RAM, 20K bytes of ROM and various input-output ports. The RAM bank



2-27 8085 System

consists of eight 16K by 1-bit dynamic memories, and the ROM bank of five 4K by 8-bit ROM's. This one board uses few chips to implement a powerful processor. Compared to the 8080, the chip-count reduction is due to the elimination of the 8224 clock, 8228 system controller, and refresh logic.

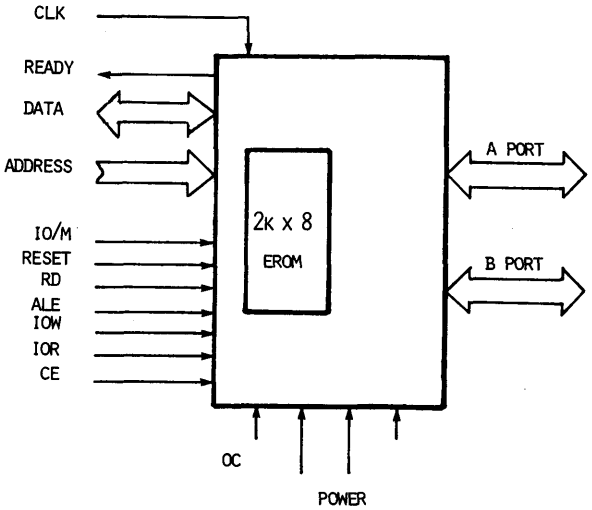
The 8085

Intel naturally also had to improve the 8080 design. The 8085 reduces the parts count of an 8080 system while increasing the speed. Essentially, it integrates the 8080, the 8224, and the 8228 into a single-chip.

This time, to provide expanded control functions, 16 address lines and 8 data lines, the decision was made to multiplex the low eight address bits. At the beginning of every instruction cycle, the low eight address-lines appear on the data-bus. To be used, they need to be latched. The multiplex-control line ALE ("address-latch enable") is used to latch and hold the lower address bits.

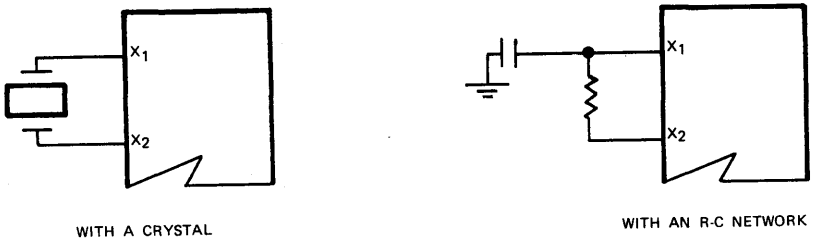
Fig. 2-27 shows the 8085 system. Right away it should be apparent that *no latch is used for the low address bits*! Intel has created a new line of special RAM, ROM, PROM, and input-output chips which *contain the low-address latch.* Thus, the 8085 bus has 8 data; 8 address; and 11 control-lines.

The special peripheral chips contain combinations of RAM, PROM, and input-output. In this way, complete systems with as few as three LSI chips may be built. An 8277 PROM I/O chip is presented in Fig. 2-28.



2-28 8277 PROM + I/O

The clock circuitry has also been built into the 8085. The connection of a crystal to two pins finishes the interface for the basic CPU.



Clocking The 8085

SUMMARY

The standard microprocessor architecture, with its three buses, controls the assembly of our complete microcomputer. The memory devices, RAM and ROM, are easily connected to the standard microprocessor buses. Small systems use partial or linear decoding to select the memory. Larger systems use full address-decoding. The 8080, 6800, Z-80, and 8085 systems were presented to illustrate the simplicity of CPU assembly. Future processors will contain almost everything—except for the crystal, making CPU assembly obsolete. The only task remaining will be signal buffering and input-output interfacing. The basic input-output techniques will now be presented, before the interfacing of actual peripherals.

CONVERSION TABLE

BINARY	DECIMAL	HEXADECIMAL	OCTAL
0000	Ø	Ø	Ø
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	10
1001	9	9	11
1010	10	A	12
1011	11	B	13
1100	12	C	14
1101	13	D	15
1110	14	E	16
1111	15	F	17

APPENDIX CHAPTER 2

CHAPTER 3
BASIC INPUT-OUTPUT

INTRODUCTION

Now that the processing section of our microcomputer is complete, the next step is to communicate with the peripherals. Information about the outside world must be gathered and processed. Once processed, the information must be displayed, and sent to control the various devices. This chapter will present the input-output techniques, and illustrate them with design examples. This will be done in two steps.

Basic input and output interfacing will first be described: Serial input-output, and parallel input-output. The concepts will first be presented, then the chips which implement the algorithms.

The scheduling techniques required for sequencing the input-output devices will then be presented: polling, interrupts, and direct-memory-access.

A terminology problem will first be clarified. Larger computers have been equipped traditionally with memory-type instructions, and with I/O-type instructions. *This distinction is obsolete for microprocessors.*

MEMORY VS. I/O MAPPING OF INPUT-OUTPUT DEVICES

The traditional implementation of computers distinguishes I/O and memory instructions:

Memory-Mapped I/O

Memory-mapped I/O refers to the use of memory-type instructions to access I/O devices. Memory-mapped input-output allows the processor to use the same instructions for memory transfers as it does for input-output transfers. An I/O port is treated as a memory location. The advantage is that the same powerful instructions used for reading and writing memory can be used to input and output data. In a traditional computer, there are usually many more memory instructions than I/O instructions. For example, in memory-mapped I/O, arithmetic may be performed directly on an input or output latch, without having to transfer the contents in and out of temporary registers.

What are the disadvantages? First, each I/O port used in this way makes one less location available for memory. Thus, if all 65,536 memory locations are needed as memory, memory-mapped I/O should not be used. Clearly,

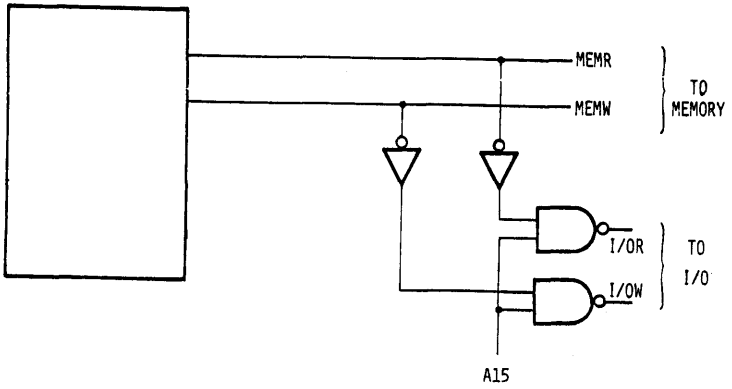
this is virtually never the case in a microprocessor system. Second, instructions that operate on the memory normally require three bytes to address the location of the port (there can be 65,536 locations, which require 16 bits of address), whereas special I/O instructions may need only eight bits to specify a port). Third, memory-mapped I/O instructions may take longer to execute than special I/O instructions because of the need for extra address bytes. This problem is usually solved by allowing "short addressing", i.e., the use of 2-byte memory instructions.

I/O Mapped Input-Output

In I/O-mapped input-output, the processor sends control signals indicating that the present cycle is for input or output only—not for memory. Two special lines are supplied for I/O read, and I/O write. Fewer address-lines may be used to select input-output ports, since systems need less input-output ports than memory locations.

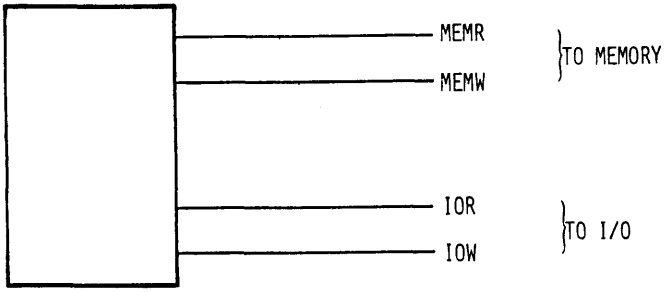
There are three advantages to I/O-mapped input-output. One, since separate I/O instructions are used, they can be easily distinguished from a memory-reference instruction while programming, a convenience. Two, because of shorter addressing, less hardware is necessary for decoding. Three, the instructions are shorter. The disadvantages are two: One loses the processing power of memory-mapped I/O, but, most important, two control pins must be "wasted" for I/O read and I/O write. For this reason, this technique is almost never used with microprocessors (except the 8080).

Fig. 3-1 shows a memory-mapped input-output system, where the control signal, which determines whether the address is for memory or I/O, depends on the state of A15. If A15 is high, then all addresses on bits A14 through A0 specify an I/O device. If A15 is low, A14 through A0 specify a memory location.

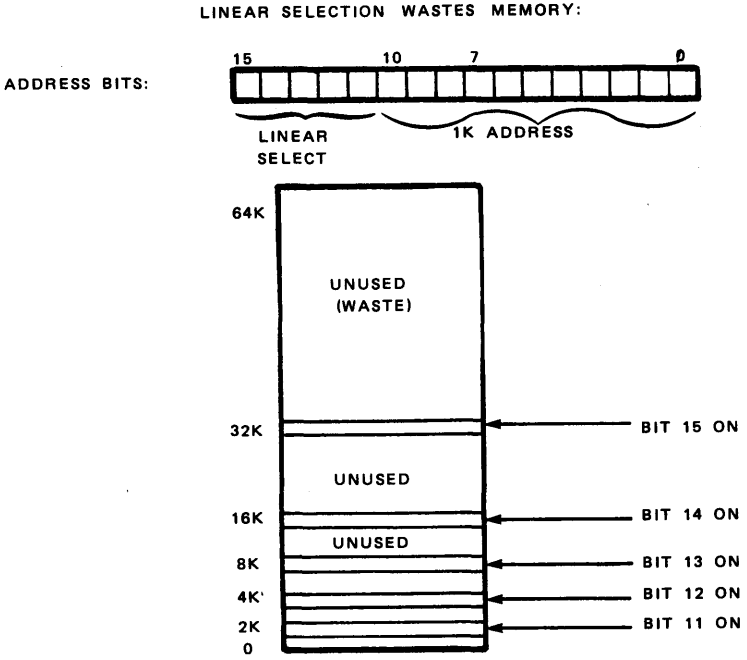


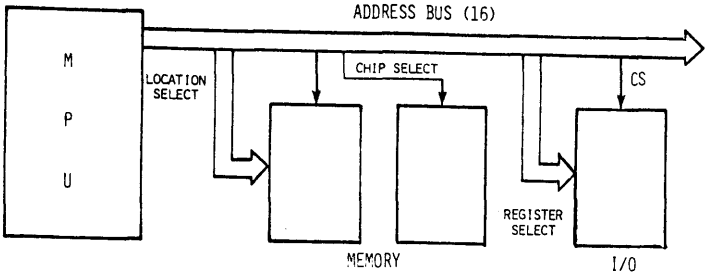
3-1 Memory-Mapped Input-Output

Fig. 3-2 shows an I/O-mapped input-output system with separate control lines for memory and I/O-control functions. The address bus will select a device and a register or location within the device. This is illustrated in Fig. 3-3. The control-bus will specify the operation to be performed. This is the standard design in most every microprocessor system.

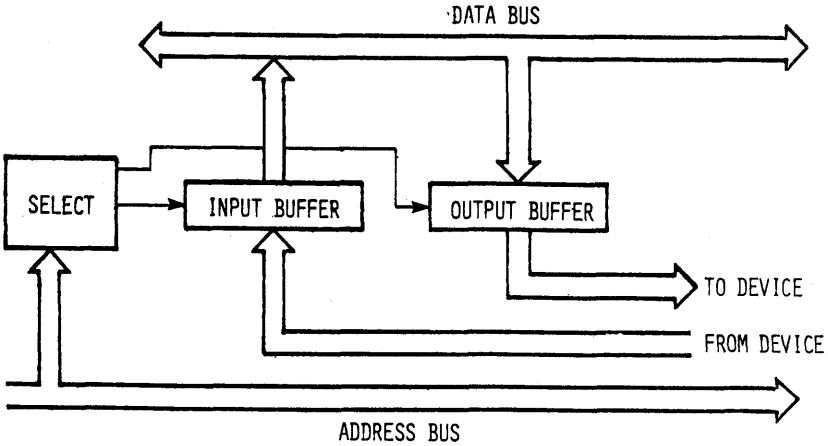


3-2 Input/Output Mapping





3-3 Selection of an I/O Port

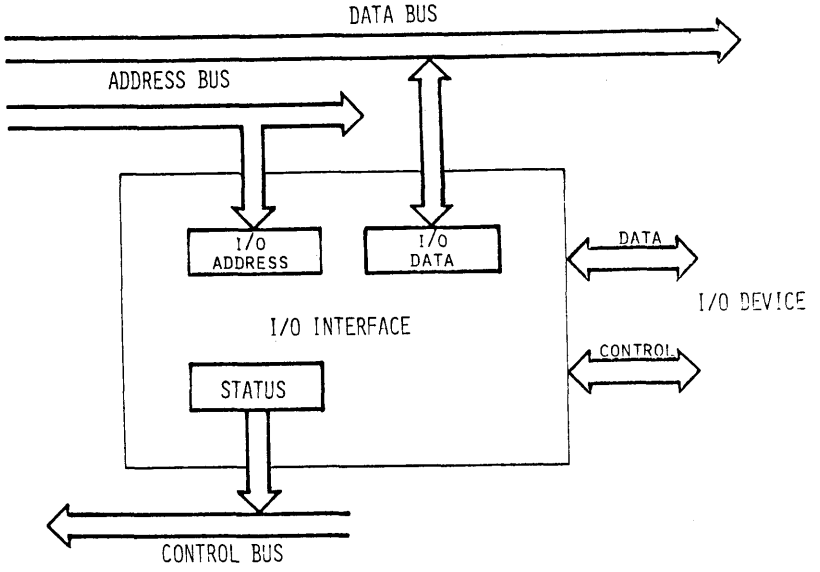


3-4 Basic I/O Port

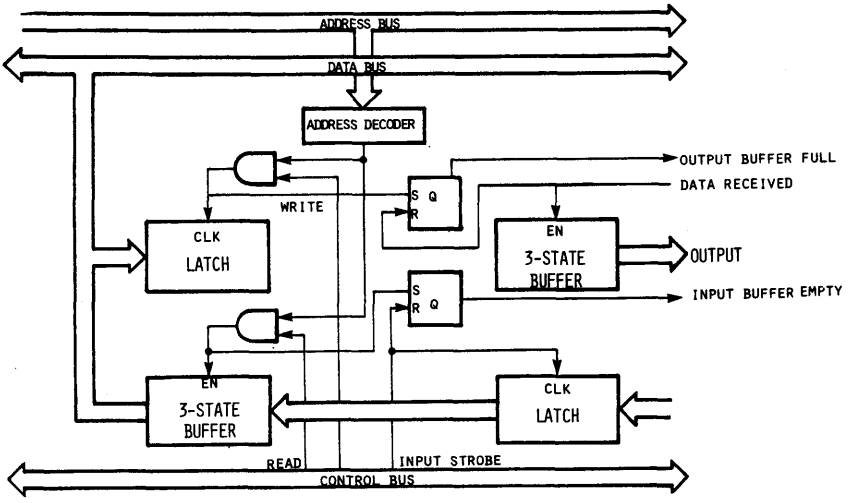
PARALLEL INPUT-OUTPUT

A minimum parallel interface requires *latches* and *bus-drivers*. Let us look at a basic LSI input-output port. On Fig. 3-4, a port is equipped with an input-buffer, which latches input signals from a device, and holds them stable, until the microprocessor requires that information, and with an output-buffer to latch microprocessor data, to hold them as long as the external device requires. In addition, there must be a *selection* mechanism and *read/write control* for the registers or ports. Figs. 3-5 and 3-6 illustrate conceptually what a simple I/O port requires.

This device has: an input-latch that can hold external information until the system reads it; an output-latch to hold data from the system stable until output, and bus-buffers to receive and drive the data-bus. Additionally,



3-5 Simple I/O Port (I)



3-6 Simple I/O Port (II)

there should be an internal status-register indicating if there are data to be read, or whether the data has been output. Although such ports can be constructed from discrete devices, a new component, the PIO, has made them essentially obsolete.

Programmable Parallel Input-Output Device

The programmable parallel LSI input-output device (PIO) will perform the following functions: address-decoding, data input-output buffering and multiplexing, status for "handshaking", and other control functions, to be described.

The address-decoder will select one of the internal registers to be read or written. These registers may be the input-latch, output-latch, direction-register, or status-register. Usually, three address-bits, as well as the chip-select, will be required for 6 to 8 internal registers. In addition, *the PIO is "programmable"*.

The new concept is the use of a "data-direction register": it is possible, on a bit-by-bit basis, to define a port as having the first three bits configured as inputs and the last five as outputs, or any other combination.

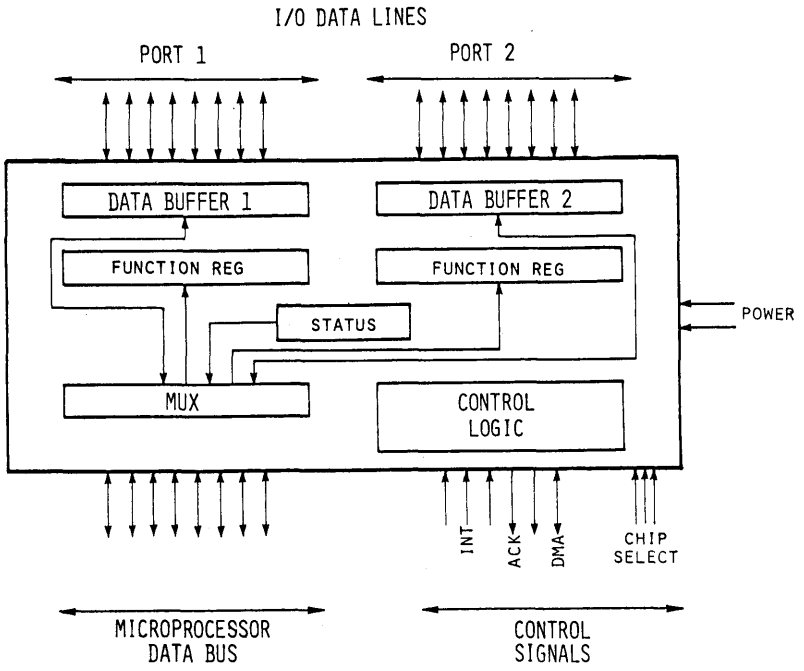
The direction of every line of the PIO ports is programmable in direction. Each bit of the "data-direction register" specifies whether the corresponding bit of the PIO port will be an input or an output. Typically, a "0" is the data-direction register specifies an input, while a "1" specifies an output. A PIO is programmable in other ways. Each PIO has one or more command-registers which specify other options, such as the configuration of the ports, and the operation of the control logic.

Finally, each PIO multiplexes its connection to the microprocessor data-bus into 2 or more 8-bit-ports. The maximum is 3, including control lines for the I/O device, because of the 40-pin limitation on the package. A typical PIO appears on Fig. 3-7. In this case, the device has two ports equipped each with its own direction register. In addition, a status-register is used to indicate the status of each port.

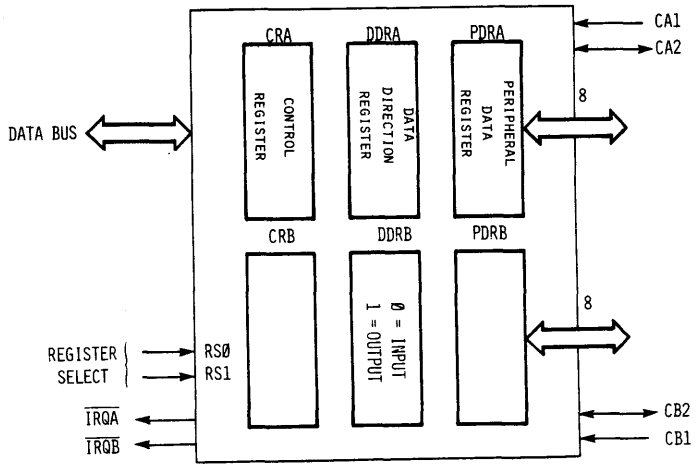
Example 1: the Motorola 6820 PIA

The internal diagram of the 6820 appears on Fig. 3-8. It has six registers, two sets of three register per port. One set is for port A and the other is for port B.

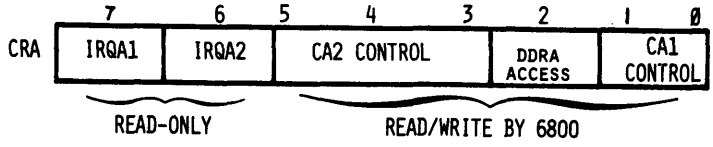
Let us examine the control register. Its format is shown on Fig. 3-9. Bit 7 indicates a transition of the CA1 input. It is used as an interrupt-flag. The same is true of bit 6, except that it monitors the CA2 pin of CA2 used as an input. Bits 5, 4, and 3 establish the eight different modes of the device, and the function of the CA2 pin. Bit 2 indicates whether the direction-register or



3-7 Typical PIO



3-8 6820 PIA



3-9 6820 Control Register Format

data-register is to be selected, as they have the same address. Bits 1 and 0 are the interrupt enable/disable control bits.

A clarification is needed here: Motorola's PIA has 6 registers and only two register-select (RS) pins, because of the 40-pin limitation. The DR and the DDR in each port share the same address! They are differentiated by the value of bit 2 of the control register, a programming nuisance.

Fig. 3-10 indicates how the registers are selected by use of the RS1 and RS0 pins, and the state of the internal bit 2 of the control register.

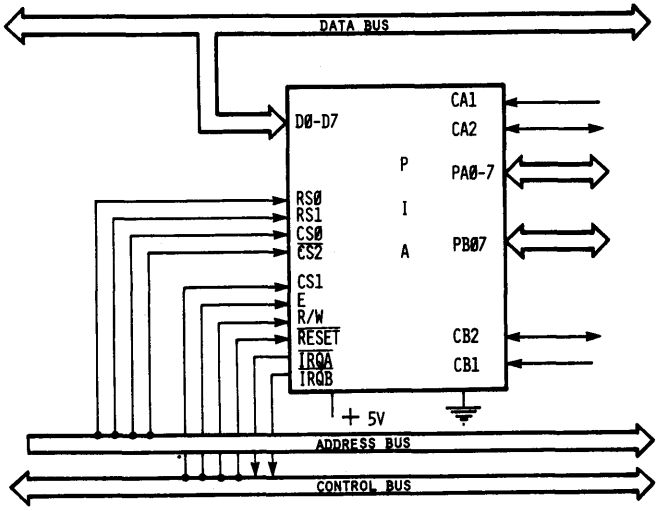
SELECTING PIA REGISTERS USES 2 LINES (RSØ, RSJ), PLUS BIT 2 OF CR:

RS1 = Ø SELECTS PORT A REGISTER
RS1 = 1 SELECTS PORT B REGISTER
RSØ = 1 SELECTS CONTROL REGISTER (A OR B)
RSØ = Ø SELECTS DATA DIRECTION OR BUFFER REGISTER

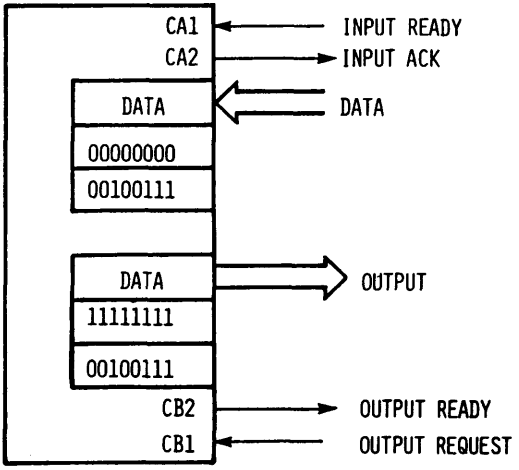
RS1	RSØ	CRA(2)	CRB(2)	REGISTER	
0	0	0	-	DATA DIRECTION REGISTER	A
0	0	1	-	BUFFER REGISTER	
0	1	-	-	CONTROL REGISTER	
1	0	-	0	DATA DIRECTION REGISTER	B
1	0	-	1	BUFFER REGISTER	
1	1	-	-	CONTROL REGISTER	

3-10 6820 Register Selection

Fig. 3-11 shows the connection to the 6800 buses, and Fig. 3-12 illustrates a typical application with the bits shown for the control and data direction registers.

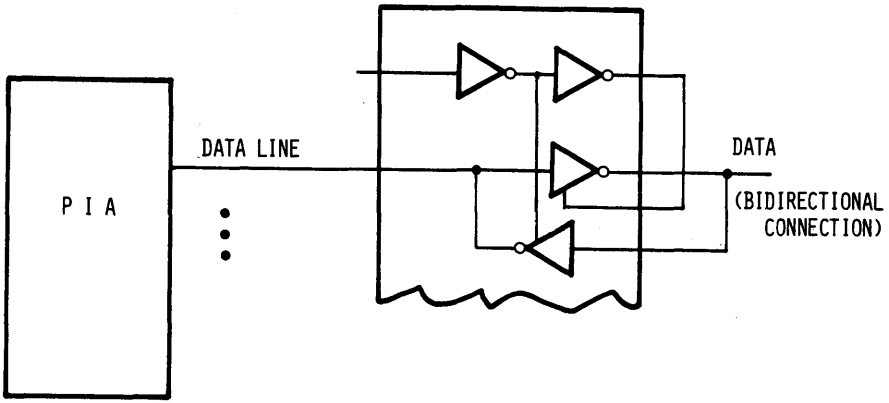


3-11 6820 and 6800 Interface



3-12 6820 Application

As a last note on the 6820, it is a good idea to buffer the data bus to this chip as it cannot drive a heavily loaded data bus. Fig. 3-13 gives a suggested buffering arrangement for the data lines.

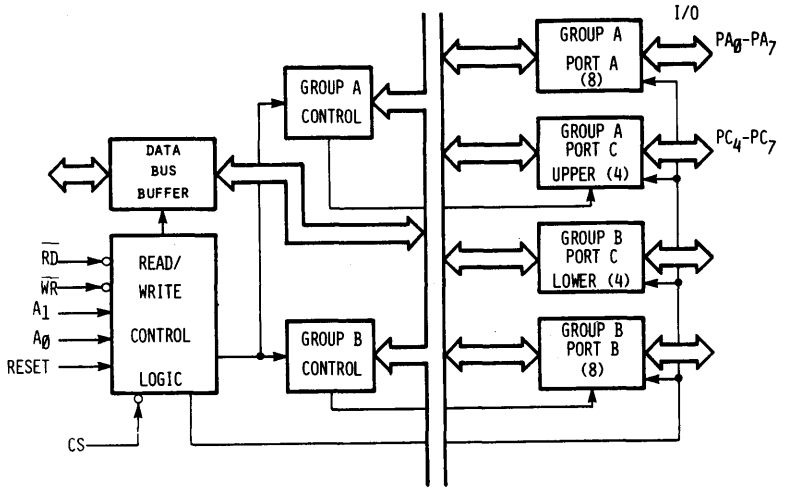


3-13 Data Bus Buffering

Example 2: Intel 8255 PPI

The 8255 contains four ports, two with eight bits each, and two with four bits each. Each port can be programmed via the mode control register to be either all inputs, all outputs, or a special function. The 8255 appears on Fig. 3-14.

Table 3-15 indicates how the ports are addressed. There are several modes of operation, where each half of port C are used for interrupt flag inputs or handshaking signals. The Intel device is not programmable by bit, but offers 4 more lines for control. Overall, the functions performed are essentially analogous. In fact, a PIA can be used on an 8080 system, and conversely. Each major microprocessor manufacturer has its own version of a programmable parallel interface. Their function is essentially similar.



3-14 8255 Addressing

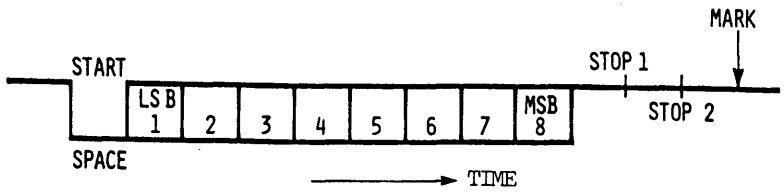
$\overline{CS}$	A1	A0	$\overline{RD}$	$\overline{WR}$	OPERATION	
0	0	0	0	1	PORT A TO DATA BUS	MPU READ (A,B,C)
0	0	1	0	1	PORT B TO DATA BUS	
0	1	0	0	1	PORT C TO DATA BUS	
0	0	0	1	0	DATA BUS TO PORT A	MPU WRITE
0	0	1	1	0	DATA BUS TO PORT B	
0	1	0	1	0	DATA BUS TO PORT C	
0	1	1	1	0	DATA BUS TO CONTROL	
0	1	1	0	1	ILLEGAL	
1	-	-	-	-	DATA BUS TO 3-STATE	(DISABLE)

3-15 8255 Addressing

SERIAL INPUT-OUTPUT:

Several devices require serial communication: teletype (TTY), tape, disk.

Instead of latching eight bits of parallel data, we could pass each bit in the byte to a single line one at a time. Known as bit-serial interfacing, there are serial standards that cover this kind of transmission. Such standards are discussed in Chapter 6. The format of the serial input-output to a teletype is shown in Fig. 3-16.



3-16 Serial Character Format

Since microcomputers are parallel systems, we need to convert an eight bit byte of data to serial form before output, and from serial form to input. There are two ways to perform this conversion: by software, or with a *UART* (universal asynchronous receiver-transmitter).

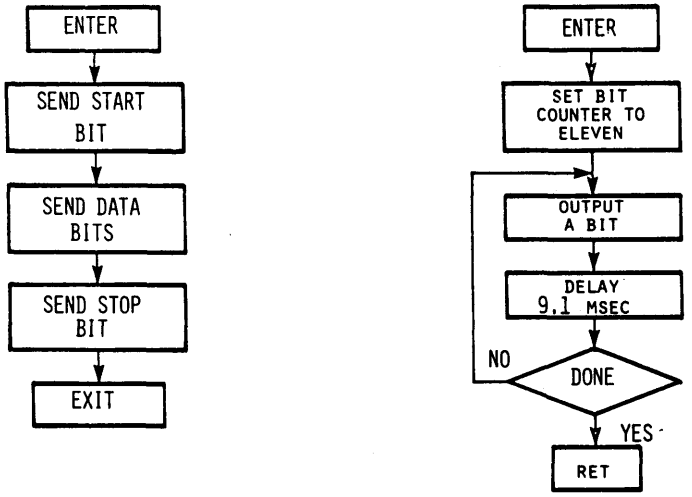
Software Serial I/O:

In software, a program can simply accomplish the serialization-deserialization. On input, the program will wait until it senses a start bit, then sample at the proper times to read the data bits. On output, the program will send the series of ones and zeroes to a single line, with a programmed delay between each bit.

An example of a teletype output program appears in the flowchart of Fig. 3-17 and the 8080 program listing on Fig. 3-18.

It will be described in Chapter 4. The principles of a serialization routine is to assemble an 8 (or more)-bit word in the accumulator, and to shift it out, one bit at a time, at the proper frequency. The simplest way is to output the contents of the accumulator to an output port which is connected only to line 0. The accumulator is then shifted right, by one bit position, a delay is implemented, and the next bit is output. After 8 (or more) outputs, the initial parallel data has been serialized.

Conversely, assembling serial data into parallel form by program is just as simple. Bit 0 is read into the accumulator. The accumulator is shifted left. After a specified delay, bit 0 is read again. After eight shifts, a byte has been assembled.



3-17 Flowchart for Serial Conversion

```
;
;  THIS SUBROUTINE ENTERED WITH CHARACTER TO BE OUTPUT IN THE C REGISTER
;
TYOUT:  MVI     B,11    ;  SET COUNTER FOR 11 BITS
        MOV     A,C     ;  CHARACTER TO ACCUMULATOR
        ORA     A       ;  CLEAR CARRY-FOR START BIT
        RAL             ;  MOVE CARRY TO A(0)
MORE:   OUT     2       ;  SEND TO TTY
        CALL    DELAY   ;  KILL TIME
        RAR             ;  POSITION NEXT BIT
        STC             ;  SET CARRY-FOR STOP BITS
        DCR     B       ;  DECREMENT BIT COUNTER
        JNZ     MORE    ;  DONE?
        RET             ;  YES
;
;  9 MSEC DELAY (ASSUME NO WAIT STATES)
;
DELAY:  MVI     D,6
DLO:    MVI     E,2000
DL1:    DCR     E       ;  1.5 MSEC
        JNZ     DL1     ;  INNER LOOP
        DCR     D
        JNZ     DLO
```

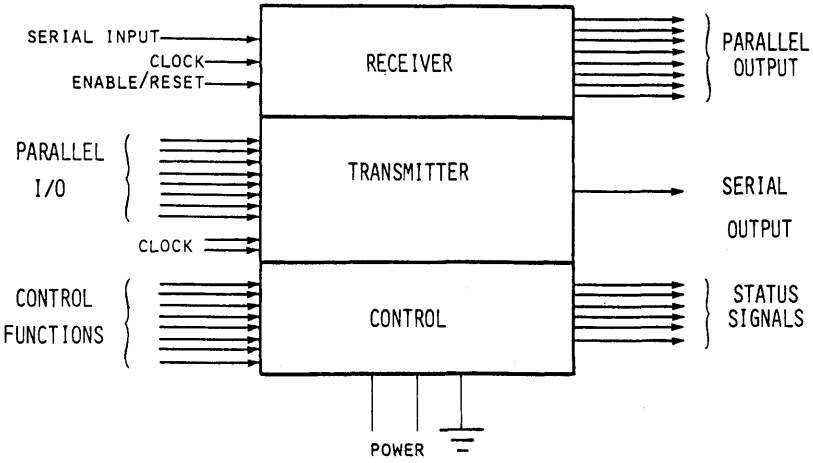
3-18 8080 Serial Conversion Program

The advantage of a programmed implementation is simplicity and the elimination of external hardware. However, it is slow, and might impair the microprocessor's performance. Also, no reliable delays can be implemented in a system using interrupts. A hardware implementation is required.

UART and USART:

One of the earliest standard LSI devices was the UART. A UART is a serial-to-parallel and parallel-to-serial converter. The UART has two functions: to take parallel data and convert it to a serial bit stream with start, parity, and stop characters, and to take a serial bit stream and convert it to parallel data.

The functional block-diagram of the UART appears on Fig. 3-19. Each UART has 3 sections: a transmitter, a receiver, and a control section. Almost all the manufacturers have a pin-compatible or "improved" version of the standard UART.

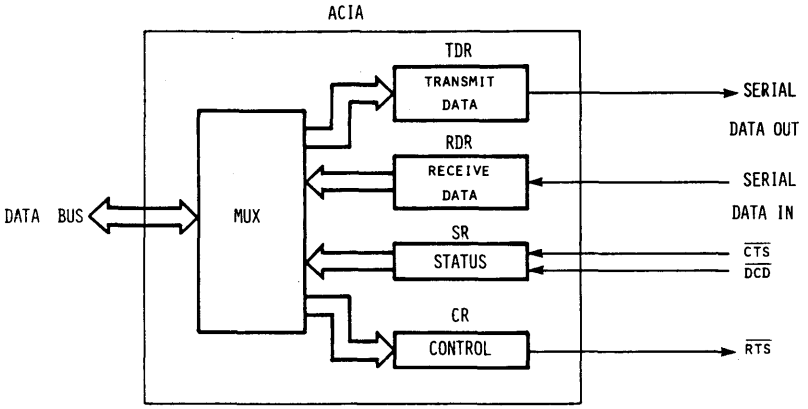


3-19 UART Block Diagram

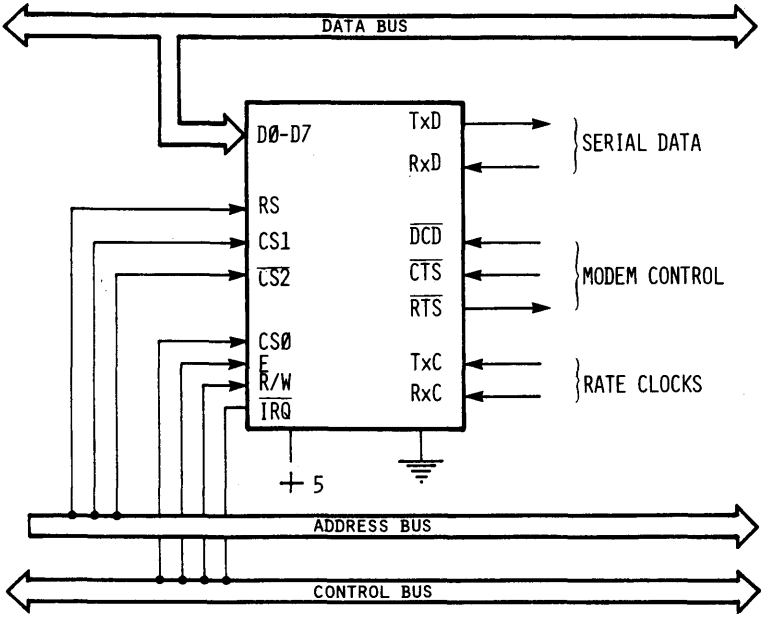
The UART requires both an input port and an output port to interface to a microcomputer system, so subsequent UART's were designed to be directly bus-compatible with microprocessor buses. Two of these are: the Motorola MC6850 ACIA (asynchronous communications interface adaptor), and the Intel 8251 USART (universal synchronous and asynchronous receiver-transmitter).

Example 1: the Motorola 6850 ACIA

The internal block diagram of the ACIA appears on Fig. 3-20. Besides the input and output serial/parallel registers, the control circuitry implements the control functions of the EIA RS232C standard. (See Chapter 6 for details on RS232C).



3-20 6850 ACIA



3-21 6850 ACIA: Functions

Fig. 3-21 breaks down the inputs and outputs into their functions: the serial data, the modem control, the clocks, and the buses. The serial data in and out are TTL-compatible signals and must be buffered to provide the necessary levels to drive serial devices. (See Chapter 4 for a full explanation of how to connect a teletype to an ACIA). The modem-control controls the interface required in an RS232C modem link.

The clocks control the bit rate of the serial data and may be different for transmit and receive sections. The bus signals are the signals used in a 6800 system. The truth table showing the addressing of the internal registers, appears on Table 3-22.

R S	R/W	REGISTER
Ø	Ø	CONTROL
Ø	1	STATUS
1	1	RECEIVE DATA
1	1	TRANSMIT DATA

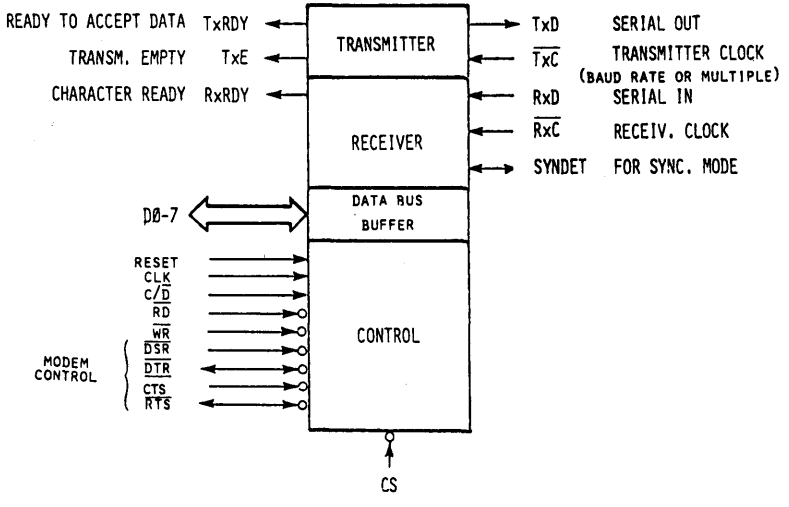
3-22 6850 Internal Register Addressing

Example 2: The Intel 8251 USART

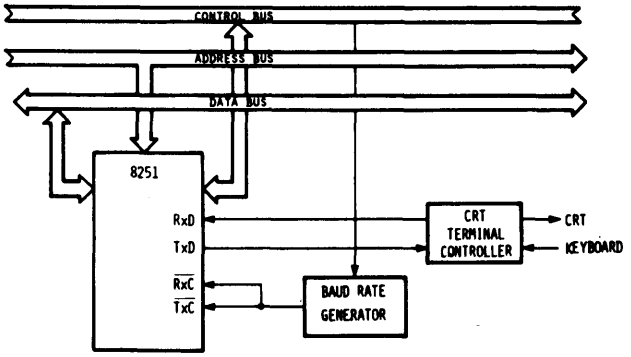
The block-diagram and control signals for the 8251 USART are shown on Fig. 3-23. This device differs from the ACIA: it also provides *synchronous* data transmission and reception, in addition to asynchronous transmission. (Motorola supplies a separate USRT, the "SSDA" for synchronous communication).

The 8251-to-8080 system interface appears on Fig. 3-24. Some of the internal circuitry of the 8251 is dynamic, hence the need for the Φ2 clock signal. The rest of the signals are straightforward.

The USART has five internal registers: receive data, transmit data, mode, status, and control. Upon reset, the first byte sent to the 8251 as control will set the *mode*. The next byte sent as control will be latched-in as *control*. The *mode* determines whether the 8251 is to be used in synchronous, or asynchronous, mode. The *control* indicates the word length and other transmit parameters. Table 3-25 is a truth table of the 8251 bus control signals.



3-23 8251 USART



3-24 8251 to 8080 Interface

Serial Interface Summary:

The two methods presented, hardware and software, illustrate the traditional trade-offs decisions to be made even in the simplest interface design. Most small systems use a software serial interface whereas larger systems tend to use the UART's. Still more sophisticated circuits are being introduced to perform new types of synchronous serial communications. These LSI components implement the other serial standards described in Chapter 6.

$\overline{C/D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	OPERATION
0	0	1	0	8251 TO DATA BUS (READ)
0	1	0	0	DATA BUS TO 8251 (WRITE)
1	0	1	0	STATUS TO DATA BUS
1	1	0	0	DATA BUS TO CONTROL
-	-	-	1	DATA BUS TO 3-STATE

3-25 8251 Addressing Truth Table

THE THREE INPUT-OUTPUT CONTROL METHODS

We have introduced now the components and techniques required for basic I/O interfacing: we can create parallel and serial ports.

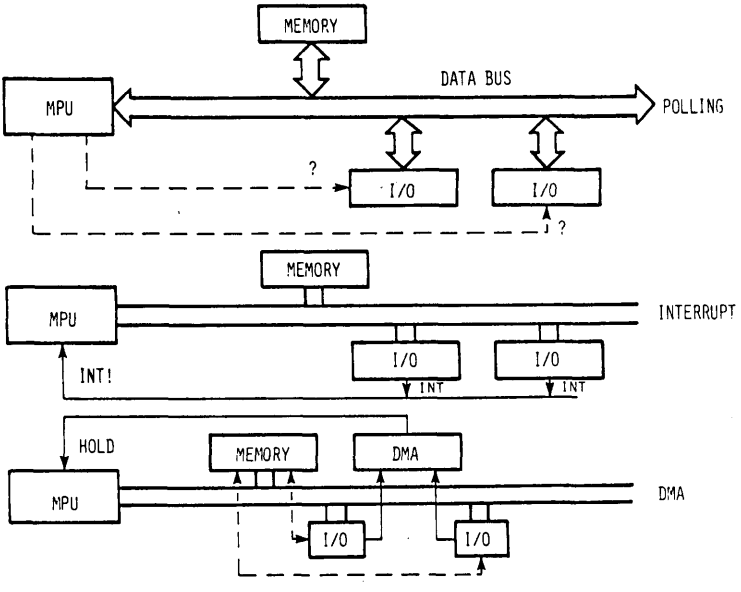
The next problem is to manage data transfers, i.e., to implement a *scheduling-strategy*. Three basic methods are used, and will be briefly described. Additional chips will be introduced to facilitate each of these strategies.

These three methods are illustrated on Fig. 3-26. They are called: polling, interrupt-controlled, and DMA. (Combinations may also be used).

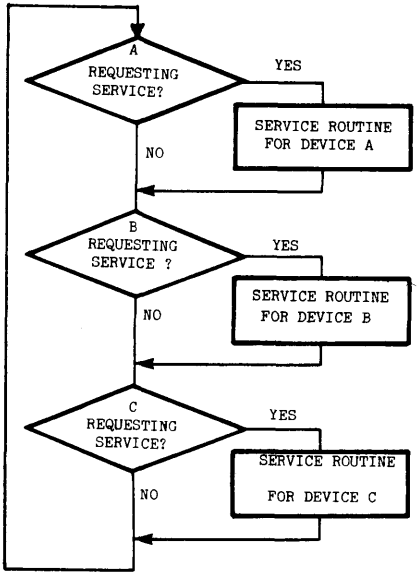
Programmed I/O or Polling:

In programmed input-output, all transfers to and from devices are performed by the program. The processor sends and requests data; all input and output operations are under the control of the program being executed. The transfers must be coordinated by a "handshaking" process. The basic method for determining if an I/O operation is needed or possible is through the use of *flags*. A flag is a bit which, when set, indicates that a condition has occurred that needs attention. For example, a flag indicates "device-ready" = buffer full for an input-device, or buffer empty for an output device.

The flag is continually checked: it is "*polling.*" The characteristic of this approach is to use a minimal amount of hardware at the expense of software overhead.



3-26 Three Methods of I/O Control



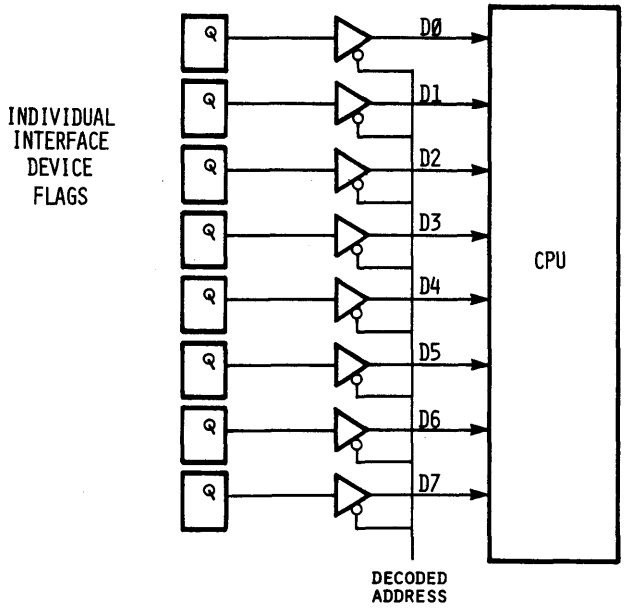
3-27 Polling Loop Flowchart

A flowchart for a *polling loop* appears on Fig. 3-27.

The program continually loops through a series of tests to determine if input or output can/should be performed. When a device needing service is found, the proper service-routine is activated and polling resumes after its completion.

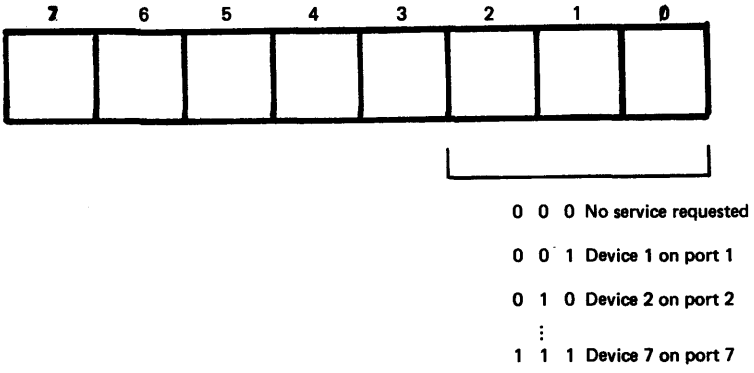
Two basic methods of sensing device-ready flags are used: the use of a simple input-status port, and the use of a priority-encoder input-status port.

The simplest technique is to drive the data-bus with the device-ready flags of eight devices when executing a read-status input-port instruction. Fig. 3-28 illustrates such a system. The input-status port may be any convenient decoded address. Usually, the first or last I/O port addresses are used for this port. When the port is read in, the program will check each bit, determine priority, and branch to the proper service routine.

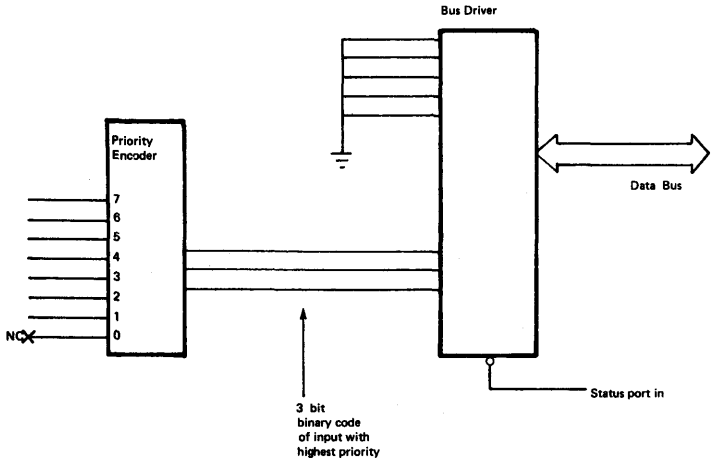


3-28 Device Ready Flag Status Port

The second method is to perform the priority encoding with a look-up ROM or a priority-encoder chip. This way, the status port holds the actual address of the highest-priority device requesting service. Figs. 3-29 and 3-30 show the byte format, and the hardware required.



3-29 Byte Format



3-30 Polling Priority Encoder Hardware

By changing the upper five bits to any other code, other port addresses may be generated. This will save looking up or generating the port address from the device-ready status-port *since that port holds the address of the ready device.*

Polling is the most common and simplest method of I/O control. It requires no special hardware and all input-output transfers are controlled by the program. Transfers are said to be synchronous with program execution.

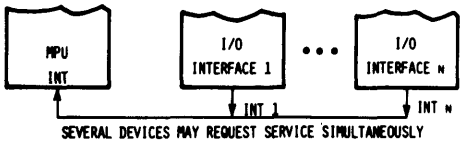
Interrupts

The polling technique has two limitations:

1. It is wasteful of the processor's time as it checks needlessly the status of all peripherals all the time.

2. It is intrinsically slow since it checks the status of all I/O devices before coming back to any specific one. This may be objectionable in a real-time system, where a peripheral expects service within a specified time. In particular, when fast peripherals are connected to a system, polling may simply not be fast enough to satisfy the minimum service requirements. Fast devices such as the floppy disk or a CRT requires a near-instantaneous response-time in order to transfer data without loss.

Polling is a synchronous mechanism, where devices are serviced in sequence. Interrupts are an asynchronous mechanism. The principle of interrupts is illustrated on Fig. 3-31. Each I/O device, or its controller, is connected to an interrupt line. This line will gate an interrupt request to the microprocessor. Whenever one of the I/O devices needs service, it will generate an interrupt pulse or level on this line to request the microprocessor's attention.

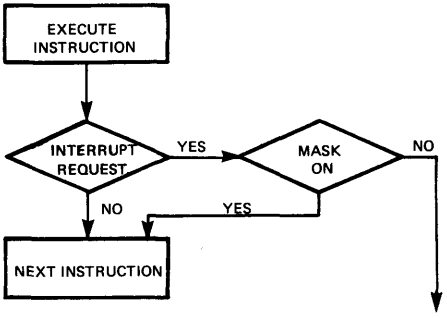


3-31 Interrupt Sequence

A microprocessor will check for interrupts at the end of every instruction. If an interrupt is present, it will service the interrupt. If no interrupt is present, it will fetch the next instruction. This is illustrated in Fig. 3-32.

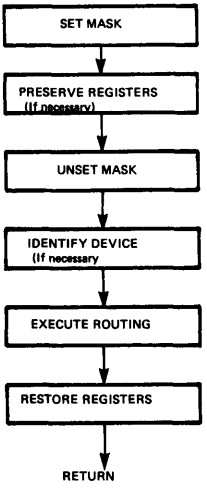
During the execution of some critical processes, it must be guaranteed that the program in execution will not be disturbed by external interrupts. One such example is the execution of a power-fail routine. Power failure can be easily detected. If the system is equipped with a battery back-up for the memory, the processor may preserve the contents of its registers in memory, and shut down the entire system in an orderly fashion. Several milliseconds of processing time are normally left, by the time the power failure is detected. A power-failure routine is then activated which should execute regardless of other less important requests which might occur. Other requests should be "masked-out". (Power-failure is considered a "non-maskable interrupt").

INTERRUPT LOGIC



3-32

This is the purpose of the mask-bit (or mask-register when several interrupt levels are available) in the microprocessor. Whenever the mask-bit is on, interrupts will be ignored (see the chart in Fig. 3-33). The "mask" facility is also often called the "enable." An interrupt will be enabled whenever it is not masked.

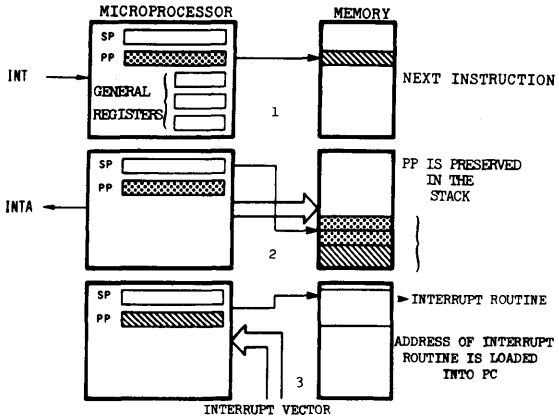


INTERRUPT HANDLER

3-33 Interrupt Control

Servicing the Interrupt

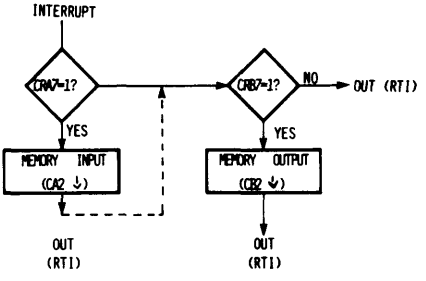
Once the interrupt request has been received, and accepted, by the microprocessor, the device must be serviced. In order to service the device, the microprocessor will execute a specialized service-routine. Two problems occur:



First the status of the program in execution on the microprocessor at the time of the interrupt must be preserved. This implies saving away the contents of all the registers of the microprocessor. These registers will be preserved in the *stack*. At the very minimum, the program counter (PC) must be pushed in the stack, in order to install a new branching address in the PC, for execution of the interrupt-handler. Preserving the rest of the registers can be done in hardware, by the microprocessor, or else may be the responsibility of the interrupt-handling routine. Once the PC (plus possibly the other registers) has been preserved in the stack, the microprocessor will branch to the interrupt handler's address. This is where the second problem arises:

A number of input-output devices are connected to the same interrupt line. Where should the microprocessor branch in order to service this device? The problem is to identify the I/O device which triggered the interrupt. This identification of the device may be done in hardware, in software, or by a combination of both methods. Branching to the I/O device address is called *vectoring the interrupt*. The simpler system, from a hardware standpoint, will not provide vectored interrupts. A *software* routine will determine the identity of the device which requested service. *Polling* will be used. The technique is illustrated in Fig. 3-34. The interrupt identification routine will poll every device connected to the system. It will check their

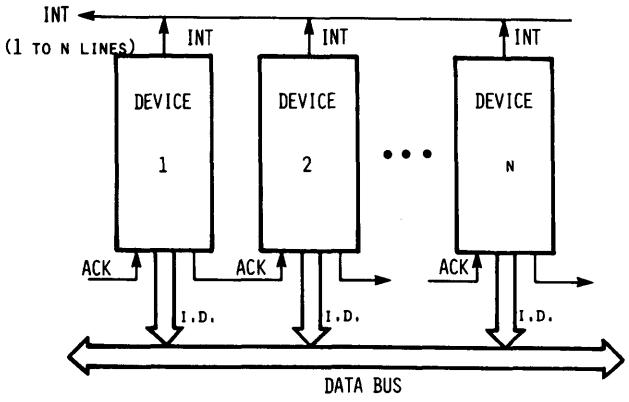
status register, usually testing bit 7. The presence of a 1 in a given bit position will signal that the device did request the interrupt. Having identified the device which has triggered the interrupt, it will then branch to the appropriate interrupt-handling-routine address. The order in which the polling is conducted will determine which device is serviced first. This implements a *software-priority* scheme, in the case where multiple devices might have triggered an interrupt at the same time.



3-34 Polling the Interrupts

A second method, software-driven, but with the help of some additional hardware, is significantly faster. It uses a *daisy-chain* to identify the device which triggered the interrupt. This is illustrated on Fig. 3-35. After preserving the registers, the microprocessor generates an interrupt-acknowledge. This acknowledge is gated to device 1. If device 1 did generate the interrupt, it will place its identification number on the data bus, where it will be read by the microprocessor. If it did not generate the interrupt, it will propagate the acknowledge signal to device 2. Device 2 will follow the same procedure, and so on. Because of the physical arrangement of devices, this interconnect mechanism is called a daisy chain. This mechanism can be implemented by most PIO's.

The fastest method is the *vectored-interrupt*. It becomes the responsibility of the I/O device controller to supply both an interrupt and the *identity* of the device causing the interrupt, or better yet the branching-address for the interrupt-handling routine. If the controller just supplies the identity of the device, it is a simple software task to look-up a table containing a branching address for each device. This is simpler, from a hardware standpoint, but does not achieve the highest possible performance. The highest possible performance is achieved when the microprocessor receives an interrupt and the direct 16-bit branching address. It can then directly

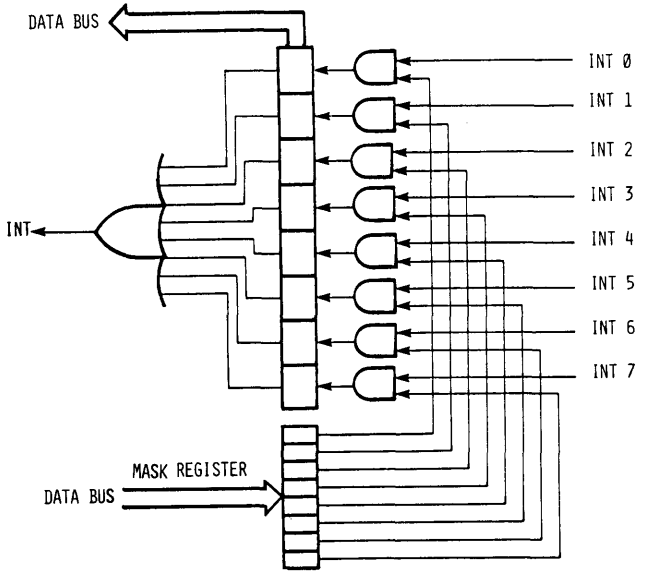


3-35 Daisy Chain Scheme

branch to the required location in the memory, and start servicing the device. The new PIC's (Priority-Interrupt-Controller) chips have made this a practical reality now.

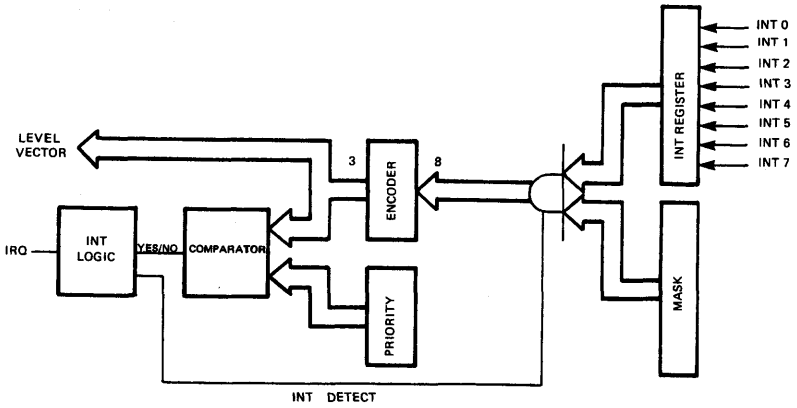
Priorities

One more problem arises: several interrupts may be triggered simultaneously. The microprocessor must then decide in which order they should be serviced. A priority is attached to each device. The microprocessor will service each device in the order of their priorities. In the computer world, priority level 0 is, by convention, the highest one, priority 1 is next, and so on. Typically level 0 will be for a power-failure (PFR or Power-Failure-Restart), level 1 will be for a CRT. Level 2 may be left vacant for the possible addition of a second CRT. Level 3 could be a disk. Level 5 will be a printer. Level 6 will be a teletype. Level 7 will be external switches. Level 4 is unused in this example. Priorities may be enforced in hardware or in software. The software enforcement of priorities has been described above. The routine looking at the devices will simply look at the device with the highest-priority first. Enforcing priorities in hardware is also possible. It is indeed accomplished in the recent PIC's. In addition, these priority-interrupt-controllers provide a full 8-bit mask which allows the programmer to mask selectively any interrupt level. The basic structure of the PIC logic appears on Fig. 3-36. It does not show the address vectoring, but simply the generation of the level-vector. Such a device typically accepts 8 interrupt levels. They appear on the right of the illustration and will set a bit in the

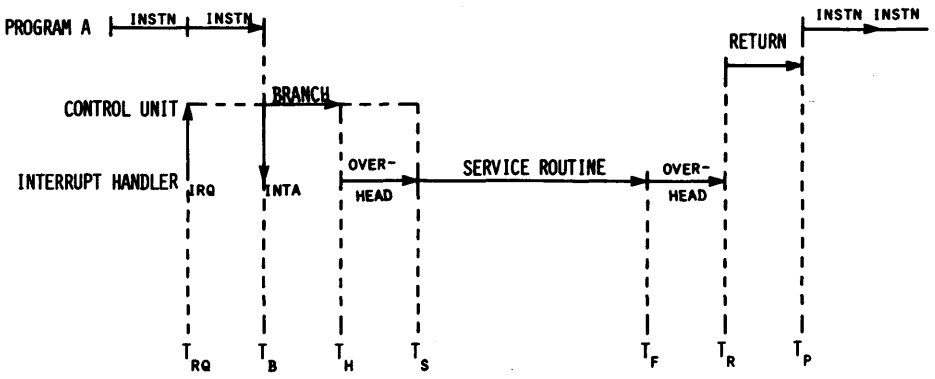


3-36 PIC Logic

interrupt register. The mask-register is used by the programmer to mask-out interrupt levels selectively. Typically, unused interrupt levels will be masked. However it is also possible to mask levels at specific times in the program. A simple AND-gate will allow the propagation of unmasked interrupts. The level of the interrupt of highest priority will be converted to a three-bit code by a an 8 to 3 encoder. One more facility is provided: the level of the interrupt is compared to the contents of the three-bit priority register. The priority-register is set by the user. It will prevent any interruption by an interrupt of level higher than n, where n is the priority. It is a global masking process for any interrupt of level higher than n. A comparator in the PIC determines that the level of the interrupt is acceptable, and will then generate a final interrupt request. The microprocessor will have available the three-bit interrupt vector. A more sophisticated PIC will do more. Recent PIC's will supply directly a 16-bit branching-address. This is simply accomplished by including a RAM of 8 × 16-bit registers within the PIC. The three-bit level vector is then used to select the contents of one of these eight registers. The contents of these 16 8-bit registers are then pulsed on the microprocessor data bus, or sometimes on its address bus. This causes an automatic branch to the specified address. Naturally, these registers are loaded by the programmer.



Interrupt Controller with Priority

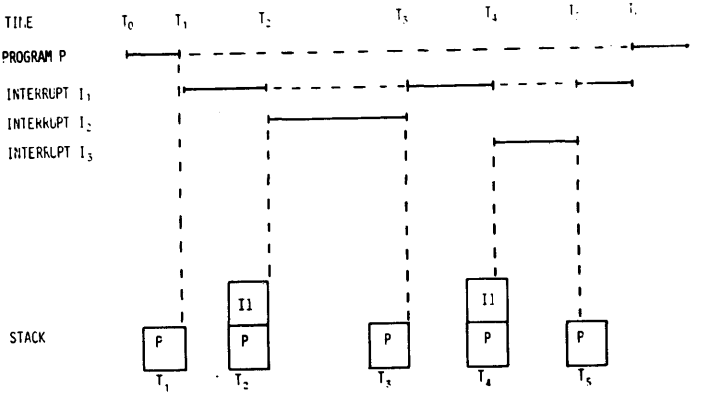


3-37 Interrupt Sequence

Fig. 3-37 illustrates the sequencing of events during an interrupt. Going from left to right on the illustration, program A is in execution until an interrupt request in generated at time TRQ. This interrupt will be taken in account at the end of the instruction, at Time TB. The control unit of the microprocessor will then implement the branch to the necessary address. Once this branch is accomplished, the interrupt handler (the third line of Fig. 3-37) starts execution. The interrupt-handler may have to spend some overhead-time in preserving the registers, which might not have been preserved automatically by the control unit of the microprocessor. The service

routine for the device then executes. At the end of execution, registers must be restored (time TF to TR). A return instruction is then executed, and the control unit restores the previous contents of the program counter (fetched from the stack), so that execution of the previous program A may resume. Program A resumes at time TP.

The time TRQ to Ts is the interrupt-response-time, i.e. the total time that has elapsed between the interrupt request, and the effective time at which the service routine has started doing its useful work. Some manufacturers consider that the response-time is only TRQ to TH. The total length of time lost to the program is TB to TP. The total overhead involved in the interrupt is really TB to Ts + TF to TR. These numbers vary significantly from one microprocessor to another.

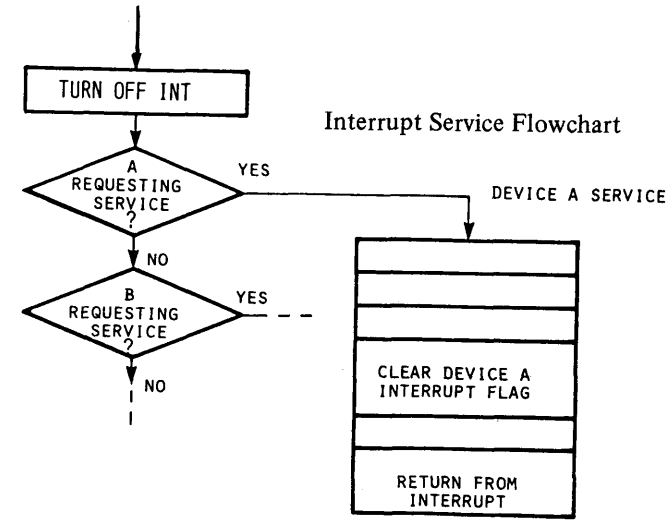


3-38 Stack During Interrupts

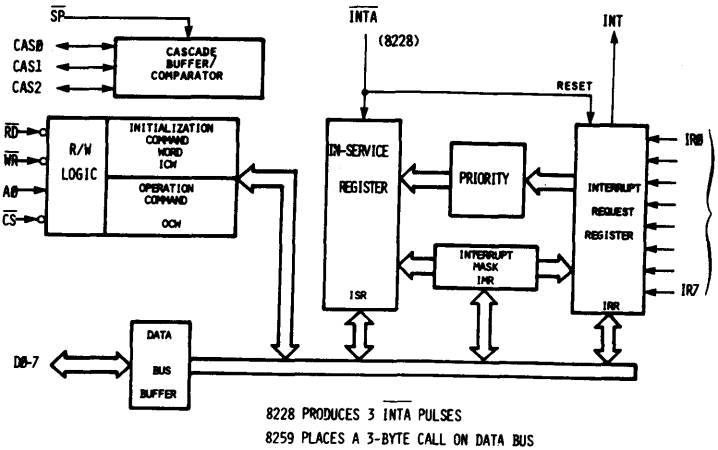
Multiple interrupts and the stack.

Fig. 3-38 illustrates the role of the stack during multiple interrupts. At Time T0 program P is in execution. At time T1 interrupt I1 is accepted. The registers used by program P are then pushed in the stack (see the bottom of the illustration, on the left). Interrupt I1 executes until time T2. At time T2, interrupt I2 occurs, and it is assumed here that I2 is of higher parity. Interrupt I1 is suspended just like program P before. The registers used for interrupt I1 are pushed in the stack. This is illustrated on Fig. 3-38 at the bottom of the illustration, by time T2. Interrupt I2 then executes; this is the third line of Fig. 3-38. Interrupt I2 executes to completion, at time T3. At that time, the contents of the stack will be popped back in the microprocessor and only P is left in the stack. (see Fig. 3-38: the stack contains only P at time T3). Interrupt I1 resumes execution, and, at time T4, it is interrupted again by another interrupt, I3, of higher priority. Again two levels are in the

stack: I1 and P at time T4 (see Fig. 3-38). Interrupt I3 executes to completion, at time T5. At that time I1 is popped from the stack (see Fig. 3-38) and resumes execution. This time it runs to completion until time T6, at which time program P is popped from the stack and resumes execution. It should be noted that the number of levels contained in the stack is equal to the number of suspended programs, i.e. to the number of dashed horizontal lines at any time. This example illustrates the use of the stack during multiple interrupts. Clearly, if a large number of interrupts may occur simultaneously, the programmer should allocate a large enough stack to contain the successive levels.



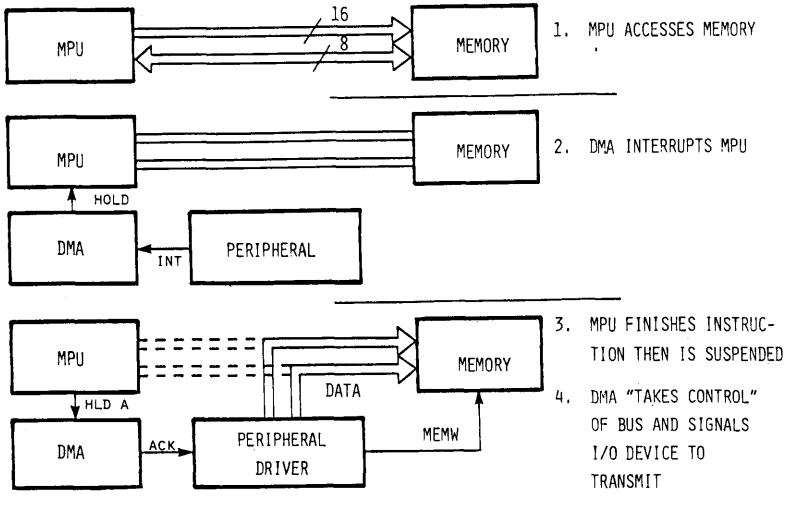
Interrupt Service Flowchart



8259 Interrupt Controller

Direct Memory Access

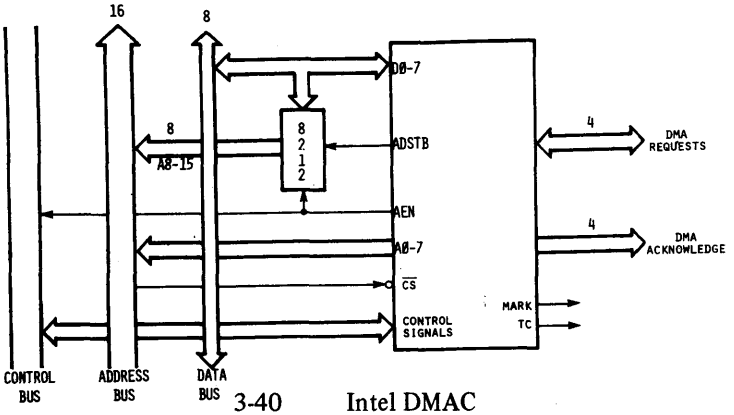
Interrupts guarantee the fastest possible response to an input-output device. However service to the device is accomplished by software. This may still not be fast enough for processes involving fast memory transfers such as disks and CRT displays. Again, the solution is to replace software by hardware. The software routine performing the transfer between the memory and the device is replaced by a specialized hardware processor, the DMAC, or Direct-Memory-Access Controller. A DMAC is a specialized processor designed to perform high speed data transfers between memory and the device. In order to perform these transfers, the DMAC will require the use of both the data-bus and the address-bus. DMAC philosophies differ in the way they obtain access to these buses. For example a DMAC may suspend a processor, or it may stop it, or it may steal memory cycles from the processor, or it may stretch clock pulses. Some sophisticated DMA's such as dynamic-memory-refresh DMA's can also use some portions of the instruction-cycle, when they "know" that the processor will not require the use of the data-bus and the address-bus. A complete discussion of DMA philosophies is beyond the scope of this book. The simplest approach, and the one usually implemented for most microprocessors, is to suspend the operation of the processor. This is the reason for the tri-state buses used for the data and the address-bus. The organization of a DMA system is illustrated on Fig. 3-39. Each device will send its interrupt to the



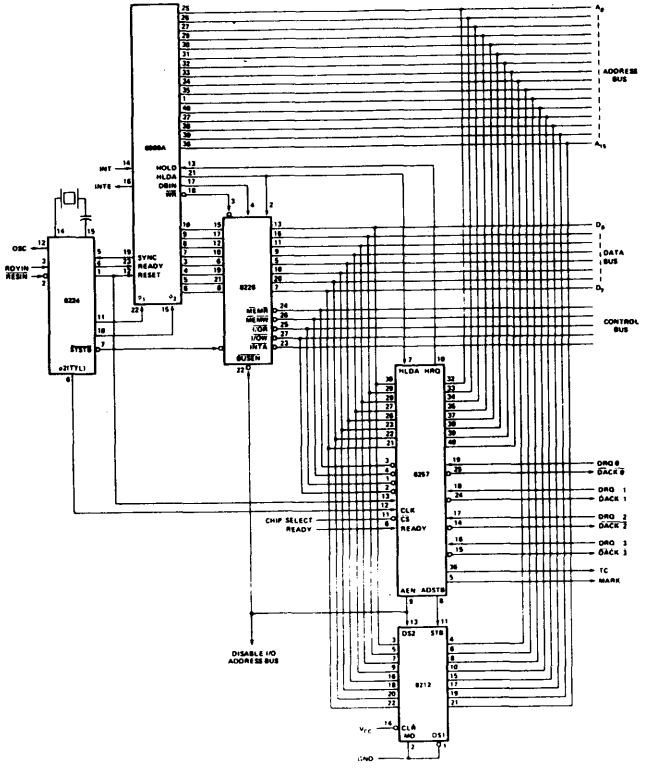
3-39 DMA Controller Operation

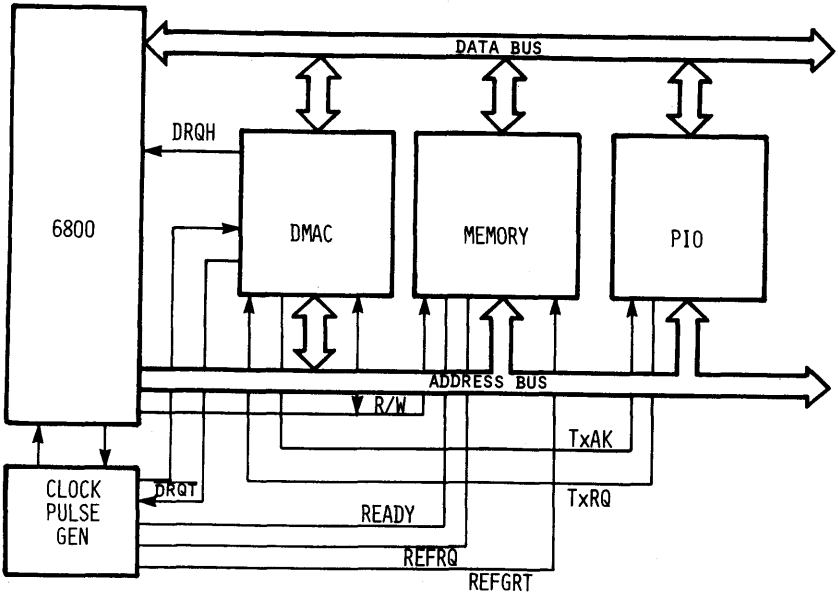
DMAC, rather than to the microprocessor. When the DMAC receives an interrupt from a device, it generates a special signal for the microprocessor, the HOLD signal. The HOLD signal will suspend the microprocessor, and place it in a dormant state. The microprocessor completes its instruction, then releases the data-bus and the address-bus in the high-impedance state. It is said to "float" its buses. It then goes to sleep, and responds with the "HOLD-acknowledge" signal. Upon receipt of the HOLD-acknowledge, the DMA knows that the buses are released. It will then automatically place an address on the address bus, which specifies the memory address at which the data transfer is to take place. A DMAC connected to 8 I/O devices will contain 8 16-bit address-registers for this purpose. Naturally, the contents of these registers have been specified by the programmer for each device. The DMAC specifies the address at which the transfer is to take place, then generates a "read" or a "write" signal, and lets the I/O device generate the data, or receive the data, on the data-bus. In addition, a DMAC contains an automatic sequencing mechanism for block-transfers. This is particularly valuable for transmitting blocks of data (in the case of a disk) or sequences of data (in the case of lines in a CRT). The DMAC is equipped with a counter-register for each device. Typically an 8-bit counter is used which allows automatic transfers of 1 to 256 words. After each word-transfer, the counter is declamented. The data-transfer stops whenever the counter goes down to 0, or whenever the DMA request from the device disappears.

The advantage of a DMA is to guarantee the highest possible transfer-speed for the device. Its disadvantage, naturally, is to slow down the operation of the processor. The DMA is a very complex device whose complexity is analogous to the one of a microprocessor. It is also expensive, since DMA's do not sell in the same quantities as microprocessors. In many instances, it may be cheaper to dedicate a microprocessor plus memory to doing dedicated block-transfers, than to use a DMA-chip. As an example, the structure of an Intel DMAC appears in Fig. 3-40, and the structure of a Motorola 6800 DMAC appears on Fig. 3-41. The DMA controller shown on Fig. 3-41 is a cycle-stealing DMA controller. The address-bus and the R/W float up to 500 ms. However the maximum duration of the suspension may not exceed 5 microseconds, as the dynamic registers of the 6800 would lose their content after this time. The new Motorola 6844 DMAC may operate in three modes: halt-burst, halt-steal (1-byte transfer), and TSC steal. In "halt-burst", a transfer request on T × RQ halts the 6800, and a byte-count of 0 restarts-it. This is a block-transfer. In halt-steal, only one byte is transferred. It has four DMA channels with 16-bit address, and

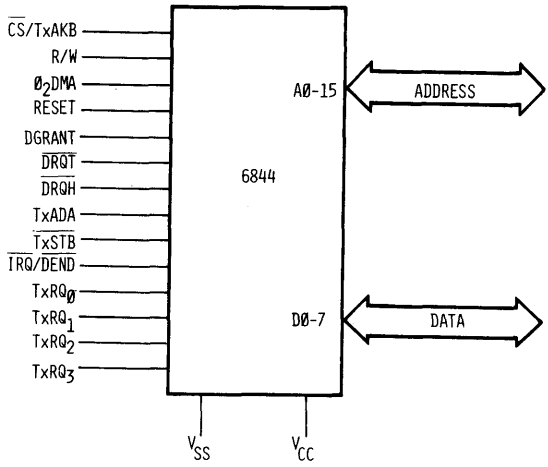


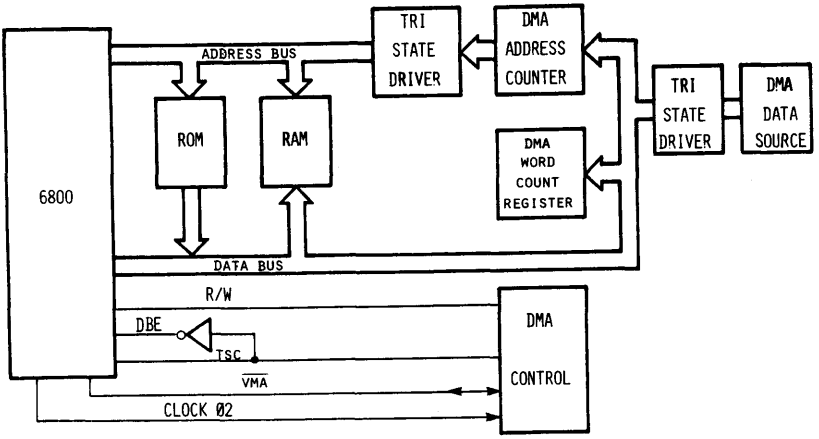
3-40 Intel DMAC



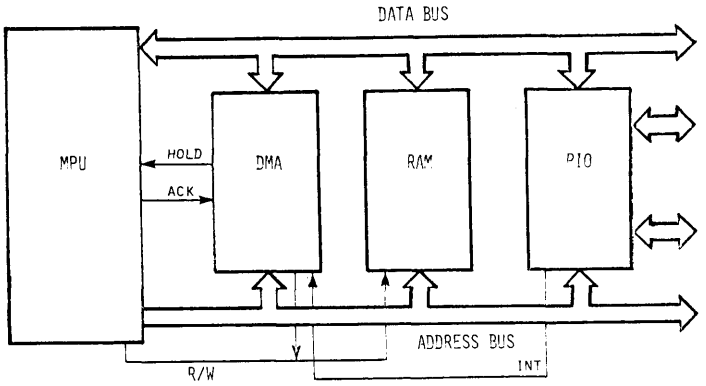


3-41 Motorola DMAC

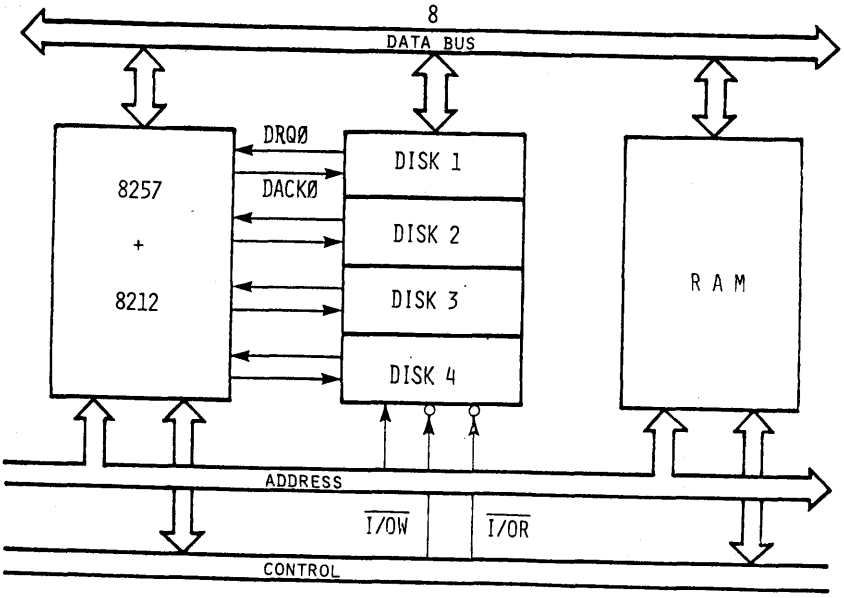




16-bit counter. The maximum transfer rate is 1 megabyte per second. This is illustrated in Figs. 3-42 and 3-43. The Intel 8257 provides four channels and operates by simply suspending the 8080 (for any length of time). It requires an external 8212 latch for bits 8-15 of the address-bus. It is illustrated on Figs. 3-40 and 3-43. Finally, the interconnect of the Am 9517 of AMD to an 8080 system is shown on 3-44.



3-42 DMA Block Diagram

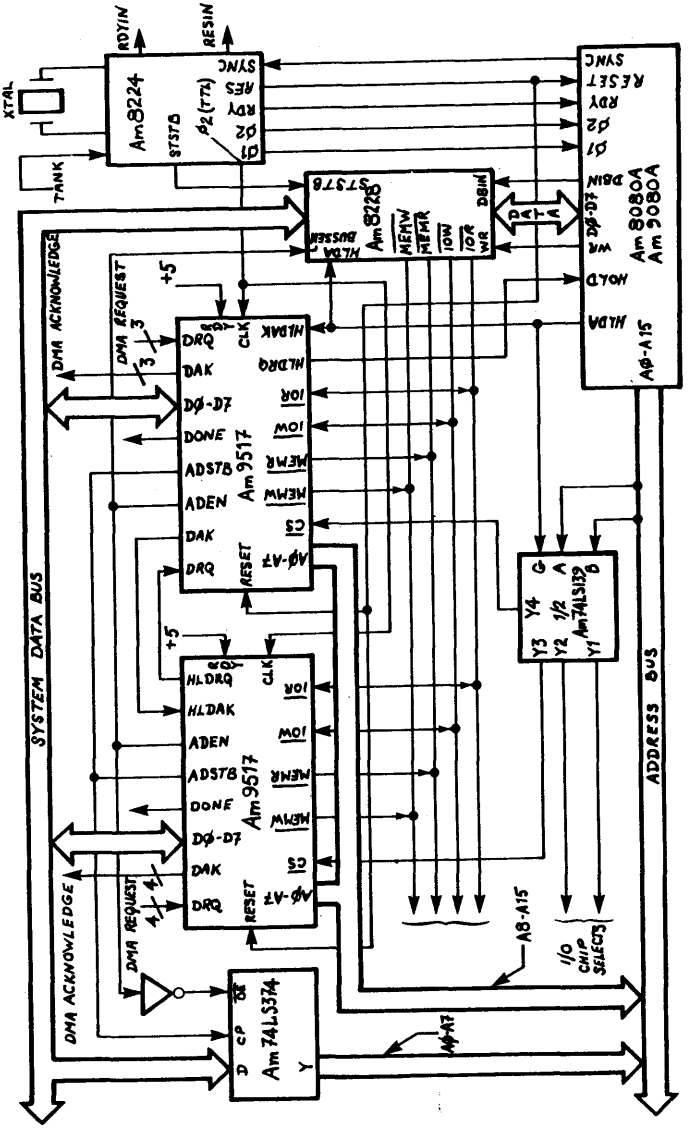


3-43 4 Channels of 8257

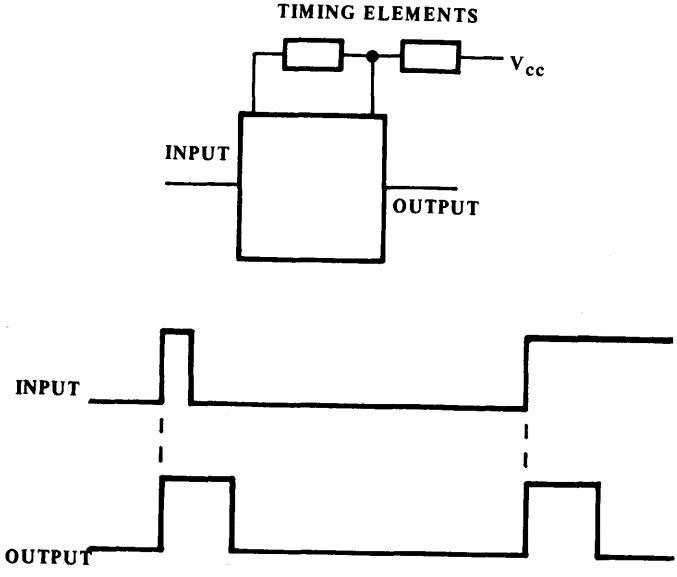
SUMMARY

The basic input-output techniques and components have been presented in this chapter. In an actual system, the designer will select the combination of hardware and software algorithms required to meet his performance and cost constraints. More chips will be introduced in the future, which offer still greater efficiency for high-speed input-output management.

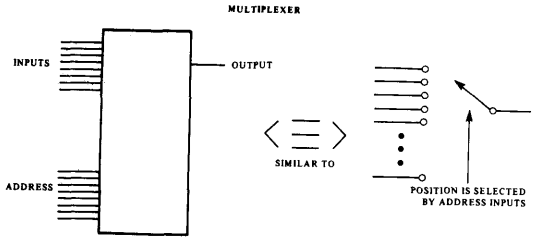
The next, and most important, problem to be solved, is to interface the peripherals. This will be done in Chapter 4.



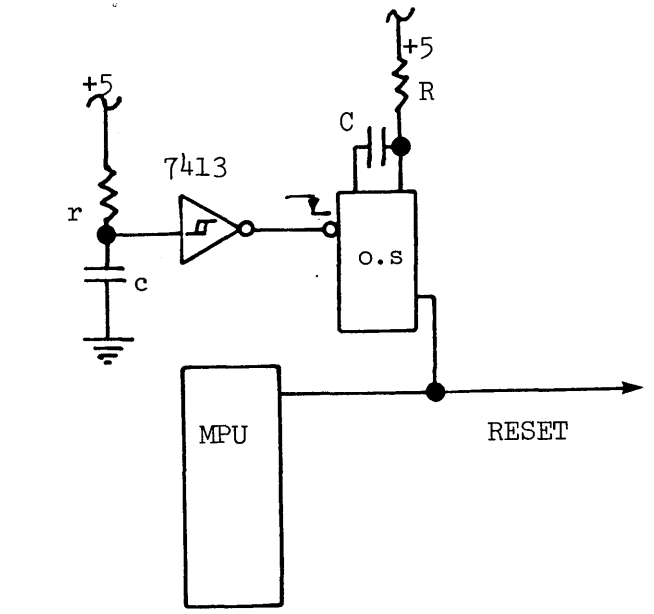
3-44 AM9517 Application Example



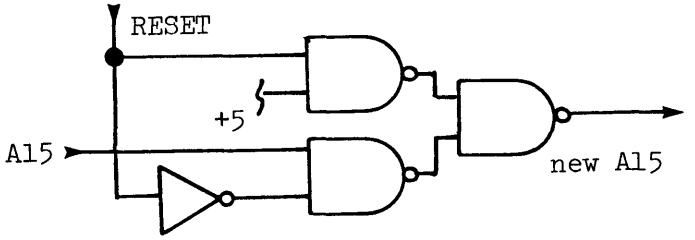
3-46 One Shot Stretches Pulses



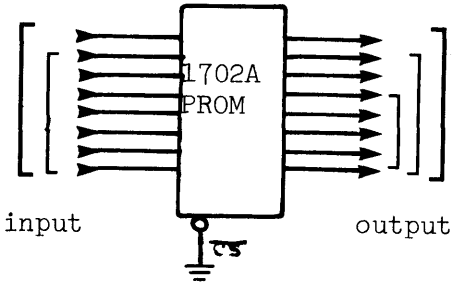
3-47 Multiplexer Operation



3-48 MPU Reset Circuit

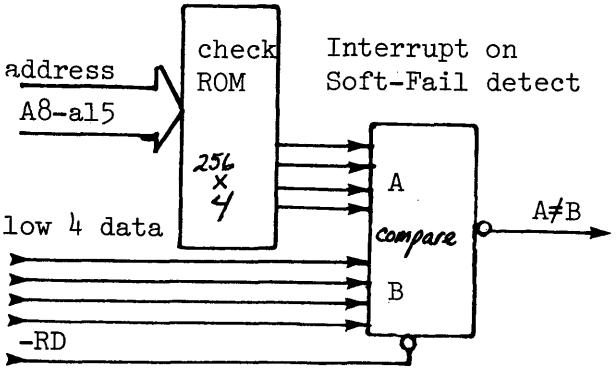


3-49 Address Vector on Reset to a Different Reset Vector



Code Converter:
Load Prom with-
Baudot to ASCII
ASCII to Baudot
EBCDIC to ASCII
ASCII to EBCDIC
or other converson
tables.

3-50 Code Conversion Using PROM



3-51 Software Failure Detecting Load ROM with Table Derived from a Logic Analyzer

CHAPTER 4
PERIPHERAL INTERFACING

INTRODUCTION

Now that the CPU, memory and input-output are connected and working, how do we connect to the teletype in the corner? What about the paper-tape punch, keyboard and telephone line? These are all *peripherals* that allow the user, or another computer, to communicate with the system. In this chapter, a number of common peripherals will be interfaced:

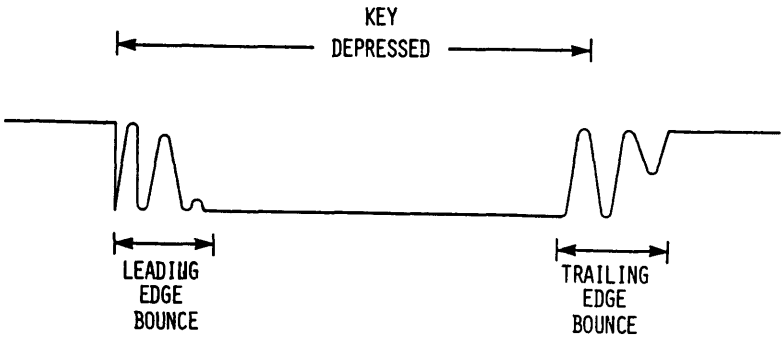
— Keyboard (including ASCII keyboard)
— LED Display
— Teletype (TTY)
— Paper-tape reader (PTR)
— Magnetic Stripe Credit Card Reader
— Cassette Recorder
— Floppy-Disk
— CRT Display

KEYBOARDS

A keyboard consists *of pressure- or touch-activated switches arranged in a matrix fashion.* To detect which key has been pressed usually requires a combination of hardware and/or software means. Two basic types of keyboards are available: *encoded* and *non-encoded*. Encoded keyboards include the necessary hardware to detect which key was pressed and hold that data until a new key stroke. Non-encoded keyboards have no hardware and must be encoded by a software routine or by special hardware.

Bounce

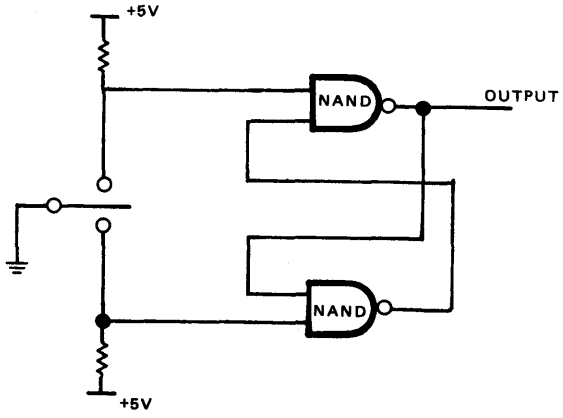
One of the most common problems with a single switch is *bounce*. Key-bounce refers to the fact that when the contacts of a mechanical switch close, they bounce for a short time before staying together. This is also true when the switch opens. Fig. 4-1 is a time-versus-resistance plot of a typical switch contact.



- BOUNCE IS 10-20 MSEC
- HARDWARE SOLUTION: R-C FILTER
- SOFTWARE SOLUTION: VERIFY KEY STATUS FOR 20 MS

4-1 Key Bounce

The solution is to wait for the status of a key to remain stable for perhaps 20 milliseconds. This may be done by *hardware-filtering* or by a *software-delay* routine. The hardware circuit appears in Fig. 4-2 and requires the same circuitry for each key. This circuit is useful for the few front-panel switches in a system. In the case of a larger number of keys, software is often used.



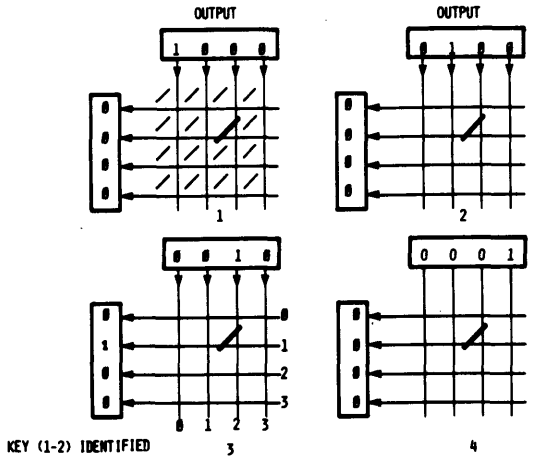
4-2 Debounce Circuit

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

A Hex Keyboard

Non-Encoded Keyboard

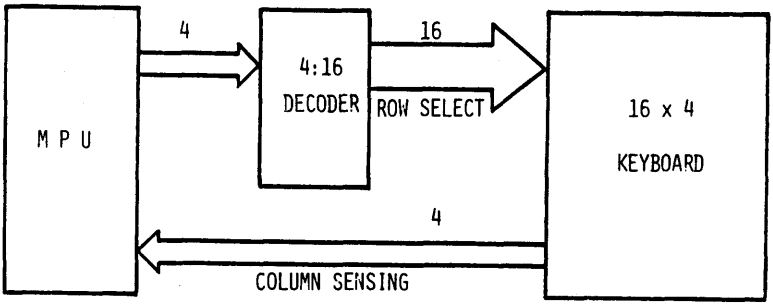
Usually, the keyboard is arranged in a row-and-column fashion, with an n by m key organization. We can scan one set of lines with a "walking one" pattern and sense the other lines for a coincidence. (See Fig. 4-3). This key-identification technique is known as "row-scanning." Once a coincidence is found, it is checked for 20 milliseconds or so, to see if it is stable and then the corresponding data are generated.



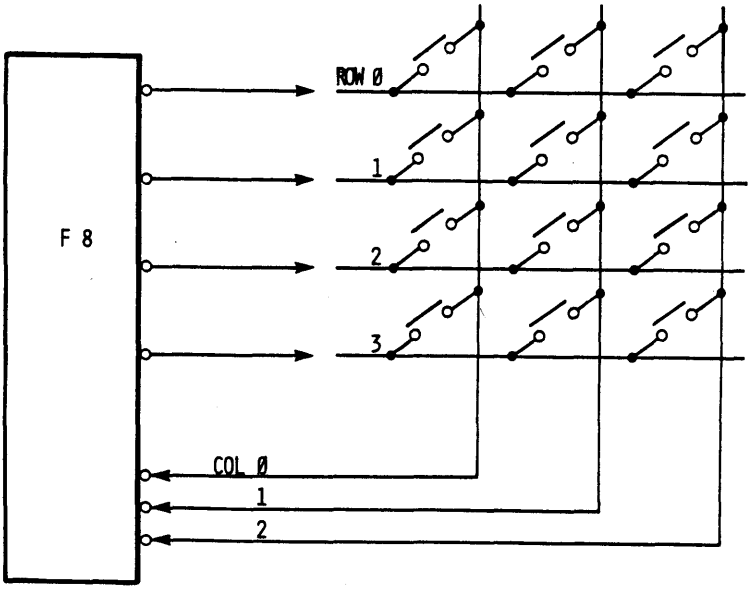
4-3 Walking Ones Keyboard Decode

Larger keyboards require more select or sense lines. Fig. 4-4 shows how a four to sixteen line decoder allows for a 64-key matrix with four bits of output and four bits of input from the microprocessor I/O ports. Fig. 4-5

shows a simple twelve-key matrix using four output bits and three input bits on an F8 microprocessor system.



4-4 4 to 16 Line Decoder with Keyboard



4-5 Twelve-Key Matrix

Rollover

Rollover is the problem caused when more than one key is held down at the same time. It is essential to detect this fact and to prevent wrong codes

from being generated. The three main techniques used to resolve this problem are the *two-key rollover*, the *n-key rollover*, and the *n-key lock-out*.

Two-key rollover provides protection for the case where two keys are pressed at the same time. Two philosophies are used. The simplest two-key rollover simply ignores the reading from the keyboard until only one key closure is detected. The last key to remain pressed is the correct one. This philosophy is normally used when software routines are used to provide keyboard scanning and decoding. The second philosophy is often used by hardware devices. The second key closure is prevented from generating a strobe until the first one is released. This is accomplished by an internal delay mechanism which is latched as long as the first key is pressed. Clearly for better protection, rollover should be provided for more than two keys.

N-key rollover will either ignore all keys pressed until only one remains down, or else store the information in an internal buffer. A significant cost of n-key rollover protection is that most systems need a diode in series with every key in order to eliminate the problem created when three adjacent keys at a right angle are pressed ("ghost key"). This increases the cost very significantly and is seldom used on lost-cost systems.

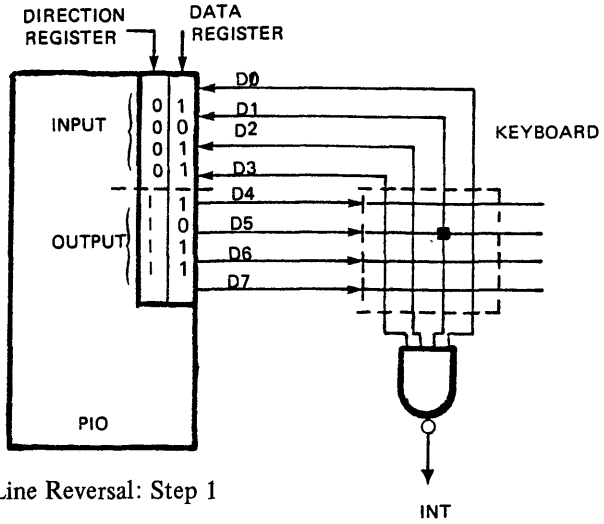
N-key lock-out takes into account only one key pressed. Any additional keys which might have been pressed and released do not generate any codes. By convention it may be the first key pressed which will generate the code, or else the last key left pushed. The system is simplest to implement and most often used. However it may be objectionable to the user as it slows down the typing: each key must be fully released before the next one is pressed down.

Line-reversal Technique

The basic technique used in identifying the key which has been pressed on a keyboard is row-scanning, as described above. However, because of the availability of the universal parallel interface chip, the PIO, another method can now be used. This is the line-reversal technique. This method will use a complete port on a PIO, but will be more efficient soft-ware-wise (faster). This method is illustrated below. In the example, a 16-key keyboard is used. One port of the PIO is dedicated to the keyboard interface. The identification of the key is performed in essentially four instructions only. In practice, some more instructions may be needed, because of the specific structure of the PIO used.

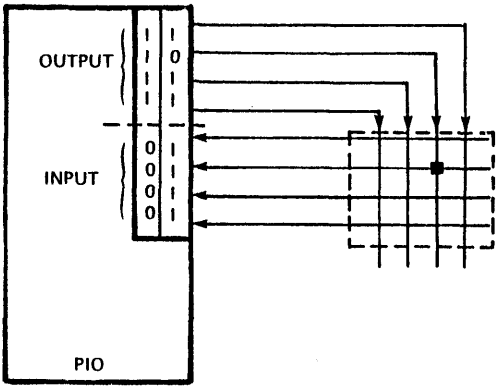
Step one: Output

Initially, the 8 lines of the PIO are configured as 4 lines in, and 4 lines



Line Reversal: Step 1

out. This will be accomplished by loading the proper data pattern in the direction-register, which controls the direction of the lines. In the example, the direction register is loaded with the value "00001111". This results in configuring the data lines D0 through D3 as inputs, and the data lines D4 through D7 as outputs. D0 through D3 are the row outputs of the keyboard. D4 through D7 are the column inputs to the keyboard. It is assumed that the initial value of the data-register is all zeroes. In other words, four zeroes are output on lines D4 through D7, the row inputs to the keyboard. Whenever a key is pressed on the keyboard, the normal output on the column, which is a "one", is grounded by the key closure. As a result, a "zero" value will appear on the column output on which a key has been pressed. In the example presented on the illustration, a "zero" appears on line D1 (the third column from the left of the keyboard). The other three column outputs, i.e. lines D0, D2, D3 have not been grounded by any key closure, and supply a "one" output. Detecting the key closure itself can be accomplished in two ways. A NAND gate, appearing under the keyboard in the illustration, may be used to generate an interrupt to the microprocessor. As an alternative, as usual, a polling program may read the contents of the data register and detect the fact that a zero is present on any one of lines D0 through D3. The problem to be solved here is to identify which key was pressed. The information available so far in the data register, i.e. "10110000" is not sufficient. The column is identified, but not the row. This problem was solved in the row-scanning technique by supplying a "one" on each row in turn. Here a more "elegant" method will be used, which will supply the same information in fewer steps.



Line Reversal: Step 2

Step two: Line Reversal

At this point, the direction of the eight lines is simply reversed. Inputs become outputs, and outputs become inputs. This is illustrated on the right of the drawing. To perform this line-reversal, a single instruction is necessary: "complement the contents of the direction-register". Naturally, this assumes that such an instruction is available. On some microprocessors, two, or even three instructions might be required to perform this on an external location.

The contents of the direction-register are now "11110000". As a result, the contents of bits D0 through D3, which were previously inputs, are now outputs. The value "1011" is therefore output on the columns of the keyboard. As a result, lines D4 through D7 are conditioned by the rows of the keyboard. In this example, the resulting value for D4 through D7 is "1011". Wherever a key was pressed, a "zero" is generated on input. Finally, it is sufficient to read the contents of the data register to know which key was pressed. The contents of the data register in our example is "10111011". It indicates that the key at the intersection of the third column and the third row was pressed. It is then a simple matter of using a branch table, or any other conversion technique, to obtain the code corresponding to the key. In addition, if more than one "zero" is present either in the first "nibble" (group of four bits) or in the second one, it detects a *multiple-key closure,* i.e. a *roll-over* problem. This is usually handled by the jump table. Such a code, having illegal zeroes, will result in a branch to a table entry which is invalid. This can be detected, or else this may cause

the whole process to be restarted again, therefore ignoring the input until only a single key is pressed.

The advantage of this technique is to require a very simple software program, and to eliminate the circuitry needed to scan rows. The disadvantage is to dedicate one port of a PIO to keyboard management. However in view of the very low cost of PIOs, this can be indeed a very inexpensive alternative.

Encoded Keyboard

Not everyone enjoys writing the software required for keyboard encoding. Various types of LSI interface-circuits are used to encode keyboards. Usually, the circuit will scan the matrix, discover a coincidence, provide for some amount of debounce and rollover, and latch the data for use in the system. Some units also provide an internal ROM look-up table to generate the proper code for the key pressed, such as ASCII or EBCDIC.

With this one chip and the microcomputer system, a complete *entry and display* interface is accomplished. Note in Fig. 4-9 that the 8279 forms the complete entry and display section interface for a point of sale terminal using the 8048 single-chip microcomputer system.

Keyboard Encoders

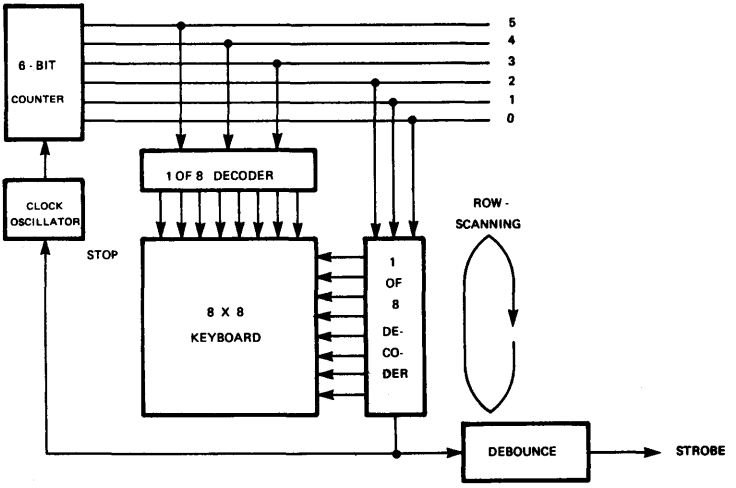
The basic role of the keyboard encoder is to identify the key which has been pressed and to supply the 8 bit key code corresponding to it. In addition, a good keyboard chip should also solve the problems we have described above. It should *debounce* and provide *rollover*-protection. Three essential types of encoders are available: *static* encoders, *scanning* encoders, and the *converting* encoder.

A *static-encoder* simply generates the code corresponding to the key. In order to simplify the key-protection problem, the linear keyboard can be considered. A linear keyboard is, for example, a 64-key keyboard which provides a wire for every key pressed. Detection is then easy. The pulse appears on the wire corresponding to the key pressed. This pulse is then simply transformed into the suitable 8-bit code. However this means 64 separate incoming lines to produce one of 64 8-bit codes. In order to reduce the cost of the wiring, and the necessity for encoders, most keyboards are arranged in matrix fashion, for example 8 by 8. In an 8 by 8 keyboard, only 16 wires are used. The price paid is that the process necessary to identify the key becomes more complex. This requires then a *scanning encoder*, or the use of a *scanning routine*. Expensive ASCII keyboards (full keyboards) can afford the luxury of a linear arrangement in view of the cost of every

key. No scanner is then necessary to identify the key. However most keyboards have the matrix arrangement.

Scanning-Chip

A *scanning-chip* solves the problem of key identification, when using a matrix-keyboard array. Each row of keys is scanned in turn by using a counter. As long as no key is pressed, the scanning goes around in a circular manner. As soon as the key is pressed, a key closure strobe is generated, and scanning stops. The counter can be read: it identifies the row and column on which the key has been pressed. Such a straightforward scanning mechanism may not provide the desired two-key rollover protection. Scanning in this system stops with the first key down which is encountered. When two keys are pressed in close sequence, one which is identified might well be the second one which was pressed. A better scanning mechanism will scan the entire keyboard for key closure and will generate a valid code only if only one key is pressed. Whenever more than one key is pressed, it will simply keep scanning until only one is held down. This has the added advantage of providing intrinsic *automatic debouncing* for the key.



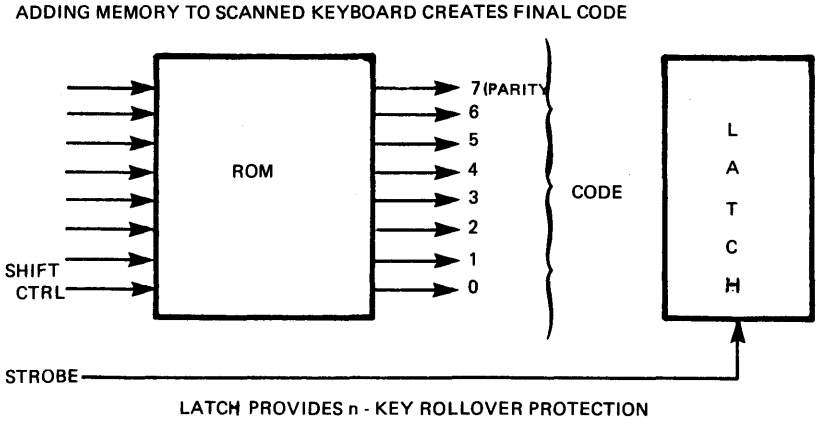
Scanning Keyboard

The above discussion was, in fact, simplified. In order to provide the reading of the key down, it is necessary to supply the voltage on the columns. If all columns were activated at the same time, it would be impossible

to determine which column was pressed. In reality, a one is supplied on a column, then on the next one, then on the next one. Whenever a key closure is detected, the column is known, and the rows are scanned for another closure.

The operation of the scanner is usually the following: a single 6-bit counter is used. The top three bits of the counter are decoded by a 1 to 8 decoder and are used to activate sequentially each one of the 8 columns in turn. The lower three bits of the counter, which change faster than the other ones, are also decoded by a 1 to 8 decoder which is used to scan the rows. This guarantees that every time a one is generated on one of the columns, the 8 rows are scanned in turn. Then the next column is activated. Whenever a key-closure occurs, the detection will occur whenever the row is selected, and this will stop the six-bit counter. The contents of the counter can then be read. They identify the column and the row which correspond to the key closure.

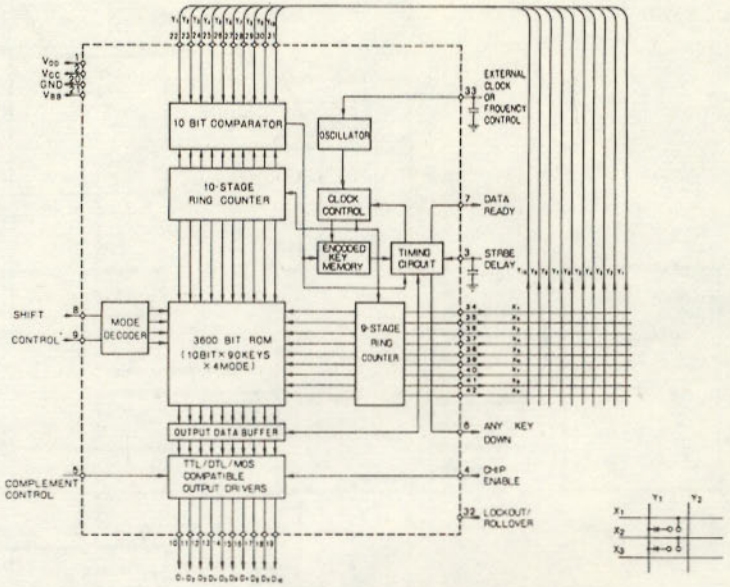
Good keyboard-encoders are equipped with a read-only memory which automatically supplies the output code corresponding to the key pressed. They should also have separate shift and control inputs. In particular this eliminates false output codes whenever wrong keys are pressed.



ROM and Latch

As an example, the NECUPD 364D-02 keyboard incoder appears on illustration 4-6.

It provides n-key lock-out, n-key rollover + debounce, frequency control oscillator, and 4 mode selections: shift, control, and shift plus control.



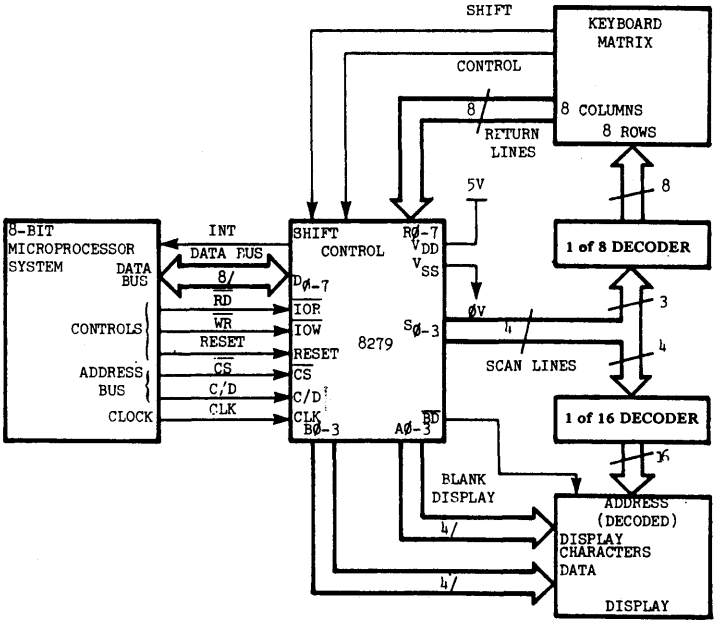
4-6 NEC Keyboard Encoder

It is equipped internally with a 3600 bit ROM. It provides a 10-bit output for 90 keys in 4 modes. The 90 keys of the keyboard must be organized as a 9 by 10 matrix. It is equipped internally with a 10-stage ring counter for the columns, and a 9-stage ring counter for the rows. In addition, its memory output is equipped with an output data buffer. This guarantees that there will be no random code outputs, while scanning occurs with no keys pressed. Other similar encoders are available from a variety of manufacturers, such as General Instruments and others. The on-chip ROM can be mask programmed to provide any desirable coding scheme—such as ASCII or EBCDIC.

This device may be used in a microprocessor system as an input port during the bus. The data ready line can be used to flag the processor when a keystroke is ready to be read.

4-7 ASCII Keyboard

Intel 8279:



4-8 8279 Keyboard/Display Interface

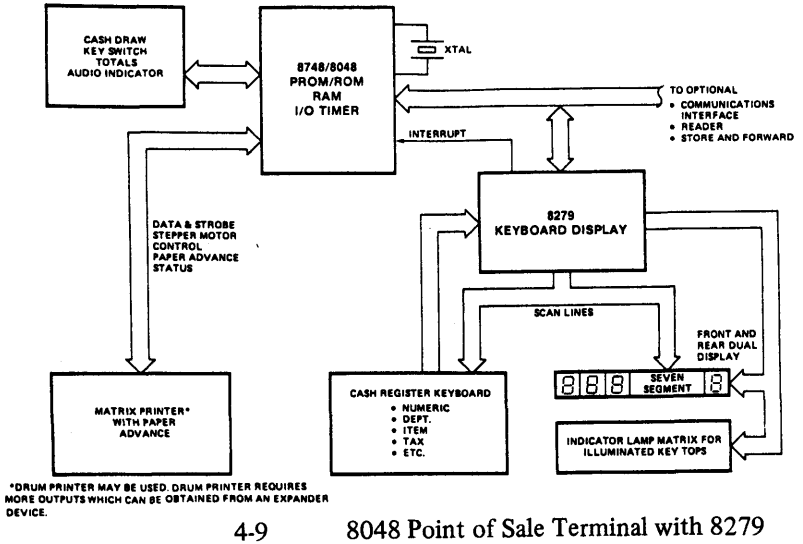
Pictured in Fig. 4-8, this LSI circuit provides for an 8 × 8 keyboard matrix with shift and control keys. In this way, up to 256 codes can be generated. For example, pressing control *and* shift and the letter "p" would be one of the codes.

In addition to encoding the keyboard, the device will also scan a display and light the display to display data stored in a RAM bank in the 8279 Similar devices are available from Rockwell and GI.

ASCII Keyboard

Keyboards may be purchased with the standard teletype or typewriter layouts that generate the seven bit ASCII code. These keyboards contain the keys, plus the LSI keyboard-controller chip. The output is usually seven parallel bits with a strobe pulse. To interface this to a standard serial input, a UART and clock may be added. The complete design appears in Fig. 4-11.

The UART takes the seven bits of data and transmits them in a serial 10 or 11 bit format when the stroke pulse occurs. The keyboard is locked-out

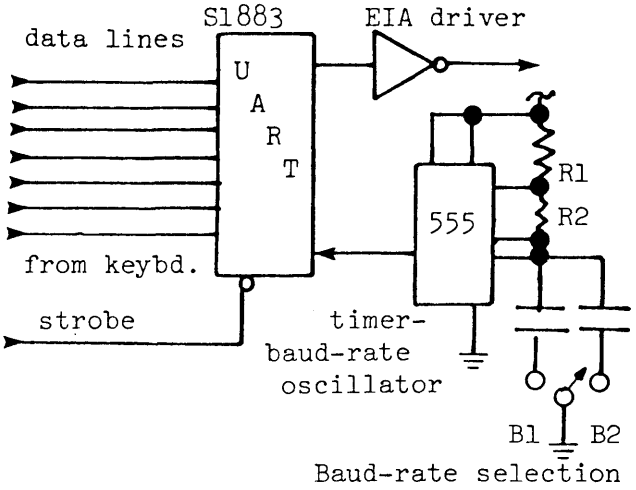


4-9 8048 Point of Sale Terminal with 8279

BIT NUMBERS								0	0	0	0	1	1	1	1
								0	0	1	1	0	0	1	1
								0	1	0	1	0	1	0	1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	HEX 1 / HEX 0	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	"	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	⁄ 6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	11	VT	ESC	+	;	K	[	k	{
			1	1	0	0	12	FF	FS	,	<	L	\	l	¦
			1	1	0	1	13	CR	GS	-	=	M	]	m	}
			1	1	1	0	14	SO	RS	.	>	N	^	n	~
			1	1	1	1	15	SI	US	/	?	O	□	o	DEL

4-10 ASCII Table

while transmitting. The serial clock runs at 16 times the bit rate. For 110 baud, the oscillator is tuned to 1760 hertz. For 300 baud, it is tuned to 4800 hertz.



4-11 ASCII Keyboard Serial Interface

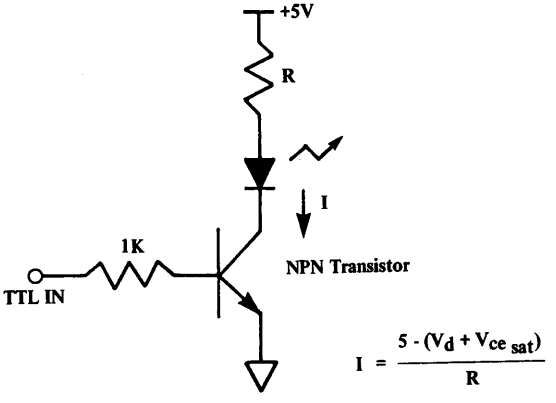
LED DISPLAYS

Light-emitting-diodes (LED's) are commonly used to indicate status or other information to the user. LED displays may take a number of forms. Three of these are: single LED, seven-segment LED, and dot-matrix LED displays.

The *single LED* is a diode with a voltage drop of 1.2 to 2.4 volts, depending on the type. It is a device that emits a narrow wavelength band of visible or infrared light. The most-used LED's are red LED's. Others used, although more expensive and sometimes not as efficient, are green, orange, yellow, and infrared LED's.

Fig. 4-12 shows an LED interface to an output port bit.

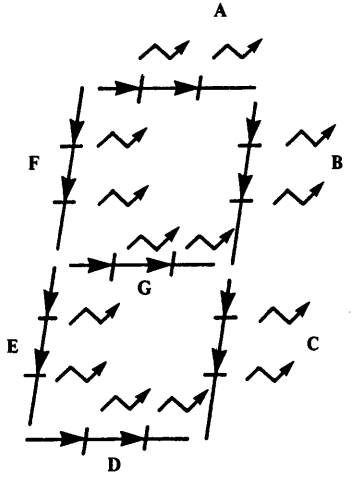
The current, I, that passes through the LED will determine its intensity. The formula given can be simplified to: $I = 3.5/R$ for a five volt supply. Typical currents are from two to twenty milliamps. When the input is less than 0.6 volts, the transistor is off and no current flows. When the input is greater than 0.6 volts, the transistor turns on and allows current to flow, lighting the LED.



4-12 Single LED Interface

Seven-Segment LED

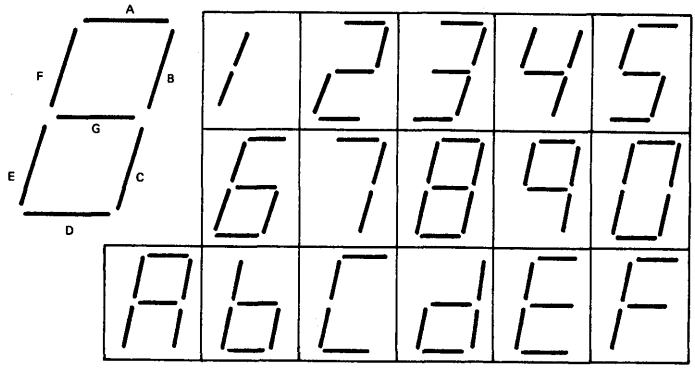
A seven-segment LED display consists of a group of seven elementary LED's arranged as in Fig. 4-13.



4-13 Seven Segment LED

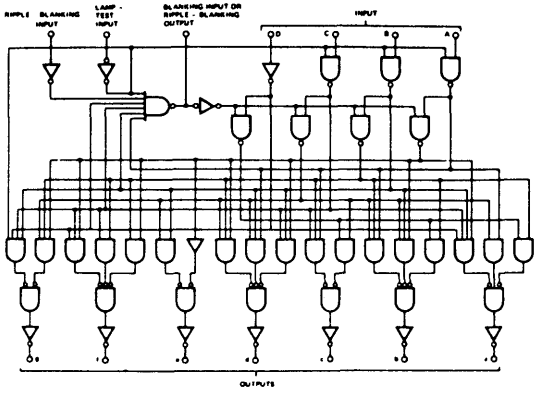
With these segments, we can display the numerals 0 through 9 and some letters of the alphabet. In this way, we have a *readout* of the seven drive signals.

A common interface device is a *BCD-to-seven-segment decoder/driver.* It will convert 4-bit BCD directly into the proper numerals and also drive



7 Segment Characters

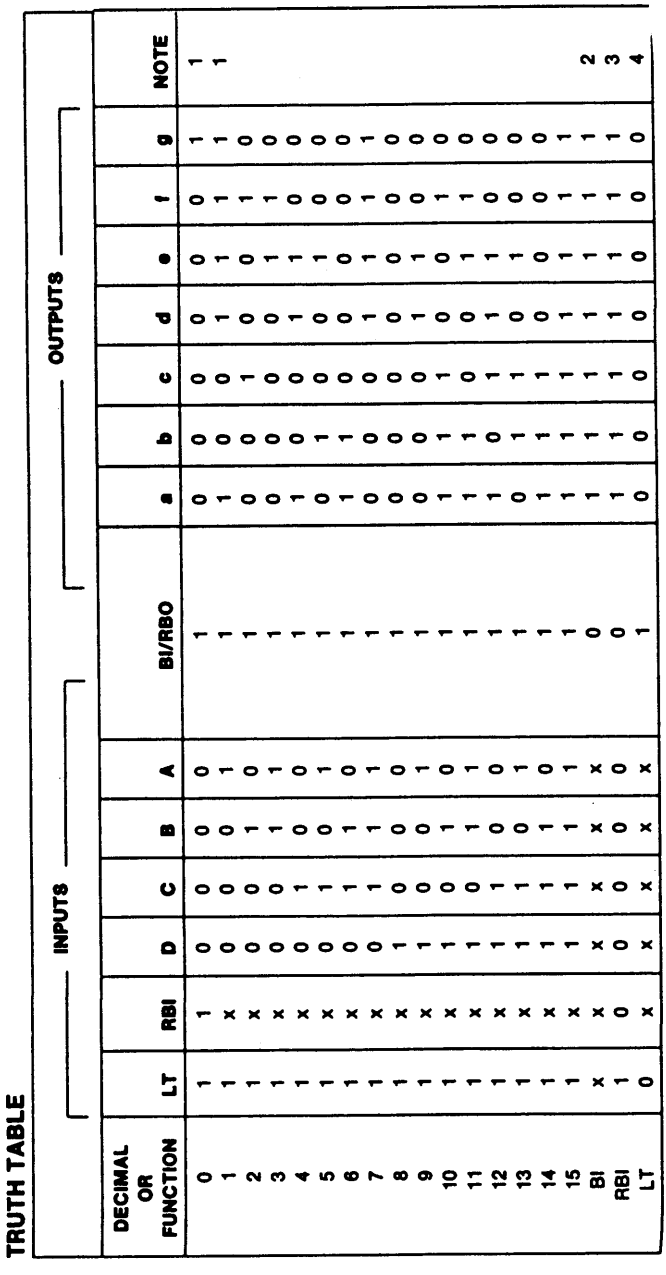
the LED's directly with internal driver transistors. An example is the 7447 pictured in Fig. 4-14. An output port may drive the 7447 with four bits of BCD data to light the proper segments.



4-14 7447 Seven Segment Decoder/Driver

In order to save the cost of having one decoder for each LED digit display, the displays may be multiplexed. Each digit is on for a short time before a new digit is selected and turned on. In this way, one decoder can serve a number of displays. There are many ways to multiplex. Two are presented here:

Fig. 4-15 shows the first scheme, which scans both digit and data. Note how *external drivers* are used. This is because, when multiplexing, each

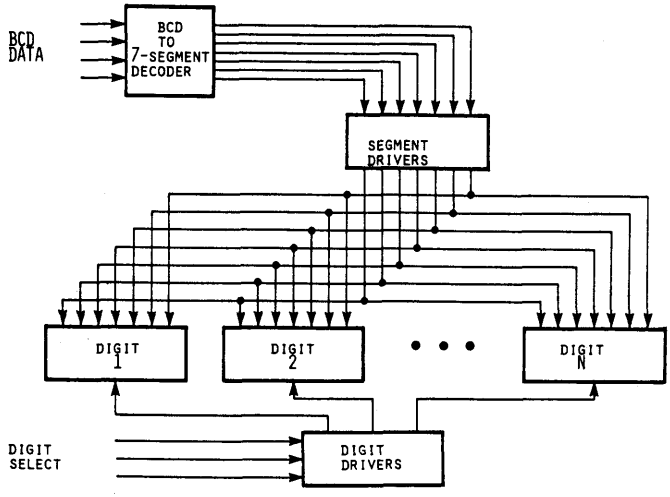


TRUTH TABLE

DECIMAL OR FUNCTION	INPUTS LT	RBI	D	C	B	A	BI/RBO	OUTPUTS a	b	c	d	e	f	g	NOTE
0	1	1	0	0	0	0	1	0	0	0	0	0	0	1	1
1	1	X	0	0	0	1	1	1	0	0	1	1	1	1	1
2	1	X	0	0	1	0	1	0	0	1	0	0	1	0	
3	1	X	0	0	1	1	1	0	0	0	0	1	1	0	
4	1	X	0	1	0	0	1	1	0	0	1	1	0	0	
5	1	X	0	1	0	1	1	0	1	0	0	1	0	0	
6	1	X	0	1	1	0	1	0	1	0	0	0	0	0	
7	1	X	0	1	1	1	1	0	0	0	1	1	1	1	
8	1	X	1	0	0	0	1	0	0	0	0	0	0	0	
9	1	X	1	0	0	1	1	0	0	0	1	1	0	0	
10	1	X	1	0	1	0	1	1	1	1	0	0	1	0	
11	1	X	1	0	1	1	1	1	1	0	0	1	1	0	
12	1	X	1	1	0	0	1	1	0	1	1	1	0	0	
13	1	X	1	1	0	1	1	0	1	1	0	1	0	0	
14	1	X	1	1	1	0	1	1	1	1	1	0	0	0	
15	1	X	1	1	1	1	1	1	1	1	1	1	1	1	
BI	X	X	X	X	X	X	0	1	1	1	1	1	1	1	2
RBI	1	0	0	0	0	0	0	1	1	1	1	1	1	1	3
LT	0	X	X	X	X	X	1	0	0	0	0	0	0	0	4

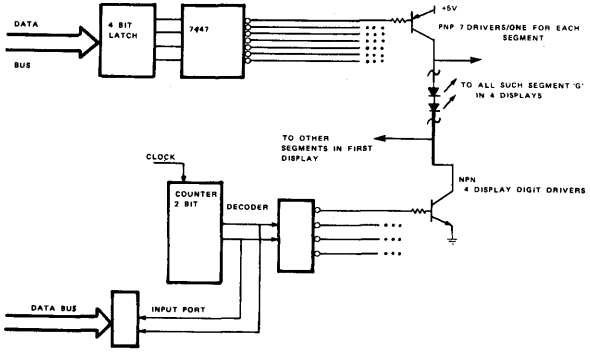
7447 Decoder Truth Table

display must be N times as bright as when it operates alone since it is on 1/N times as long. Thus, currents needed are N times as large. Most integrated circuits cannot provide this current, so external discrete transistors must be used.



4-15 Multiplexing LED's

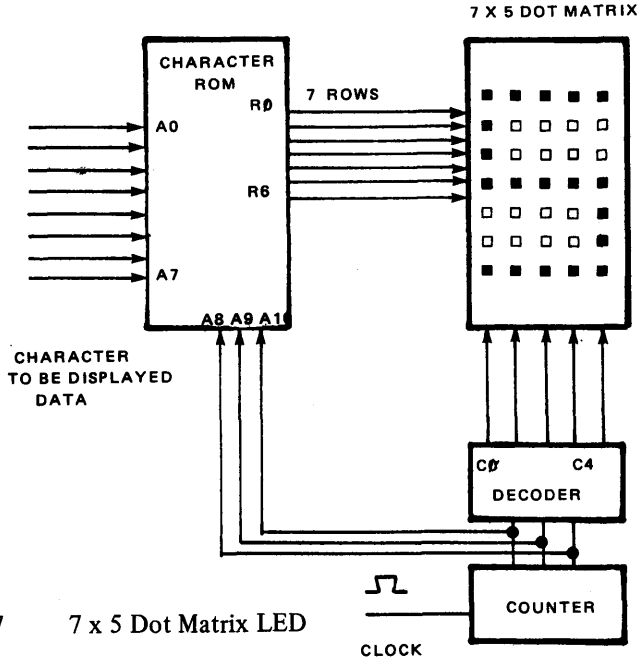
The second scheme, in Fig. 4-16, uses a *counter* to advance the digit count. The count is input to the processor and used to address the proper data for the digit. The data are placed on an output port which drives the 7447 decoder. Note again that current-buffering is needed to increase the brightness.



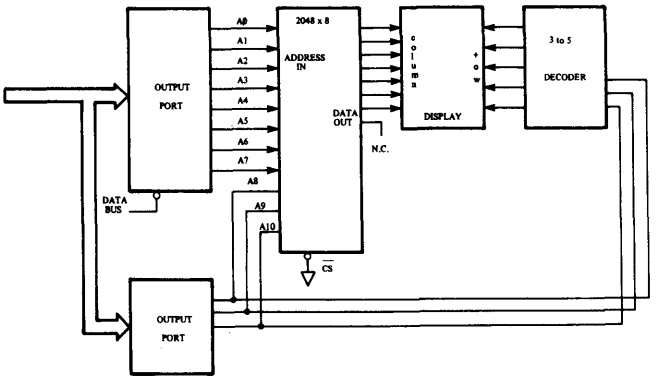
4-16 Multiplex Drivers in Scanning Scheme

Matrix LED

The LED matrix consists of five rows of seven columns of LED's. These 35 LED's can display upper and lower case letters and numbers. A typical arrangement appears on Fig. 4-17.



4-17 7 x 5 Dot Matrix LED



4-18 Counter Multiplexed 7 x 5 Matrix LED

The first output port selects the column data and the second output port selects the row, through the decoder. With this technique, the program will step through the five rows, displaying whatever character has been programmed into the 2048-by-8 read-only-memory.

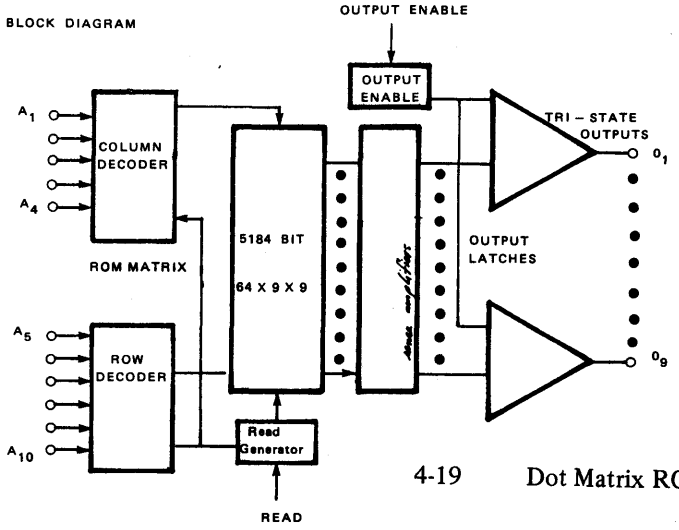
Another technique is to have external hardware step through the rows and display the proper data. Such a method is illustrated in Fig. 4-18.

The counter will start at 0 and count to 4. The character-ROM is being addressed to the character "S". Column 0 addresses the row data to be displayed. They are from R6 to R0: 1001111_2. The clock advances the counter to column 1. The row data are now 1001001_2. This continues through row Column 4 and then repeats. All the letters of the alphabet may be generated this way. A typical character-ROM is shown in Fig. 4-19.

Note that this part is for improved-resolution 7 × 9-displays. Also, this character-ROM may be used with ASCII, Bandot or EBCDIC code, depending on the table ordered.

Summary of Displays

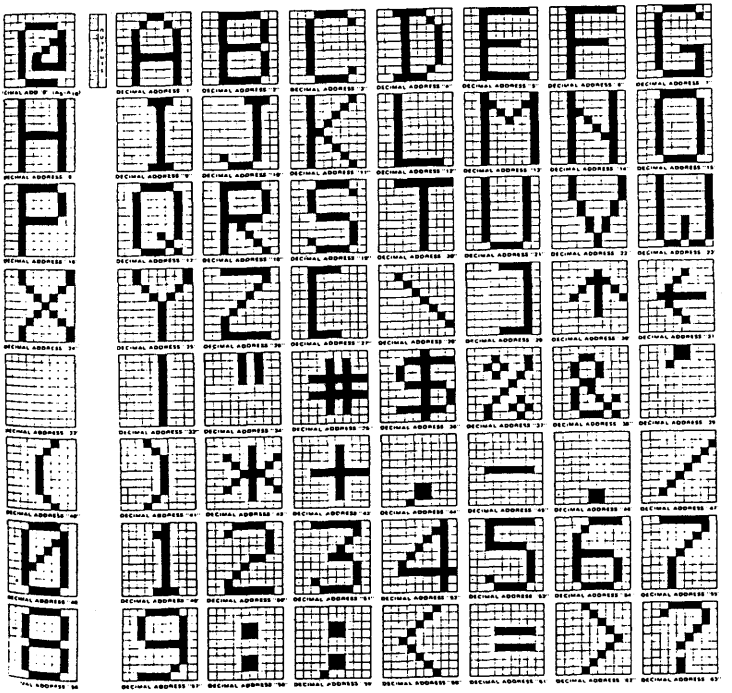
There are many other displays. However, LED type displays are reliable, easy to interface, and illustrate the techniques used in most all other display interfacing techniques. CRT-interfacing will be covered also in this chapter, and the dot-matrix methods will be presented in that section.



4-19 Dot Matrix ROM

CHARACTER FONT

ASCII SET, VERTICAL SCAN 7X9 WITH CODE CONVERSION

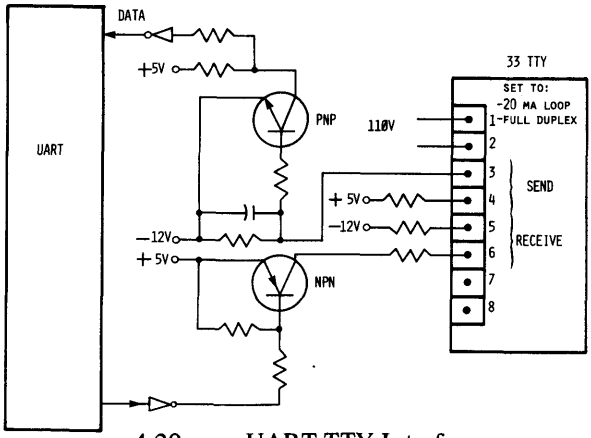


Dot Matrix ROM

TELETYPE

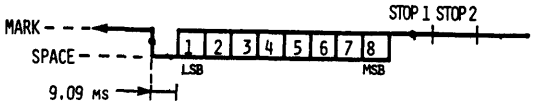
A teletype is a serial mechanical input-output device which usually operates at 110, 150 or 300 baud depending on the model and manufacturer. Three methods of interfacing will be presented here: one for a UART, using a model 33 Teletype®, one for a Motorola ACIA, using a model 33 Teletype® with opto-isolation, and one RS232EIA interface. A model 33 Teletype operates at 10 characters per second. Each character is encoded by eleven bits: one start bit, 8 data bits, and 2 stop bits. The resulting transfer rate is, therefore, 110 baud. The only significant interfacing problem is to

assemble the 8-bit parallel data-byte from these 11 bits. Transmission is asynchronous. The universal interface for a TTY is the UART, which was described in the previous section. It performs automatically all the required functions, and may operate in both directions.

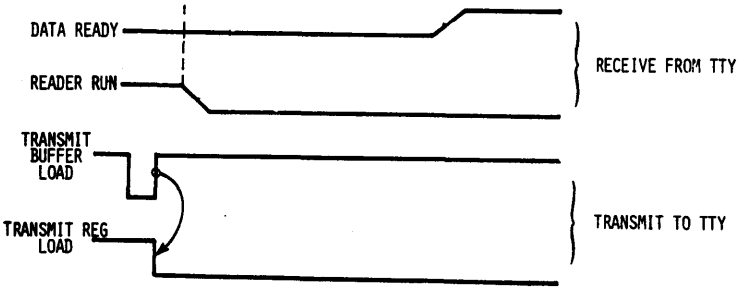


4-20 UART TTY Interface

In Fig. 4-20, the UART is used for serial to parallel and parallel to serial conversion ot the data. Fig. 4-21 illustrates the serial format, and 4-22 illustrates the timing sequence. The schematic of the interface shows how



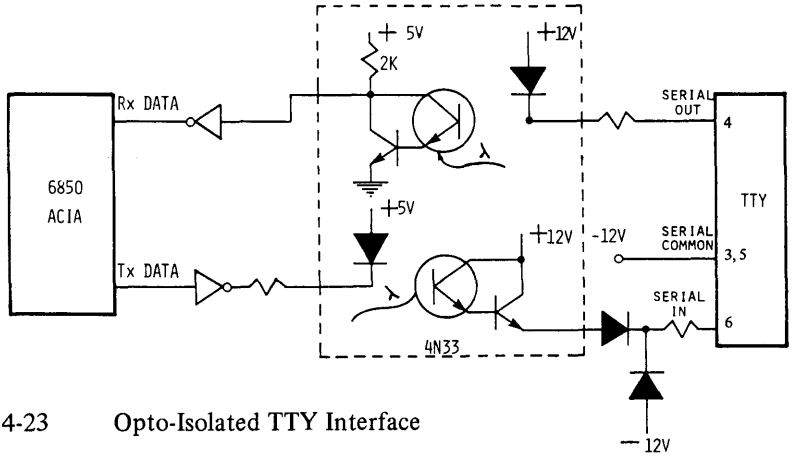
4-21 Serial Data Format



4-22 Timing of UART

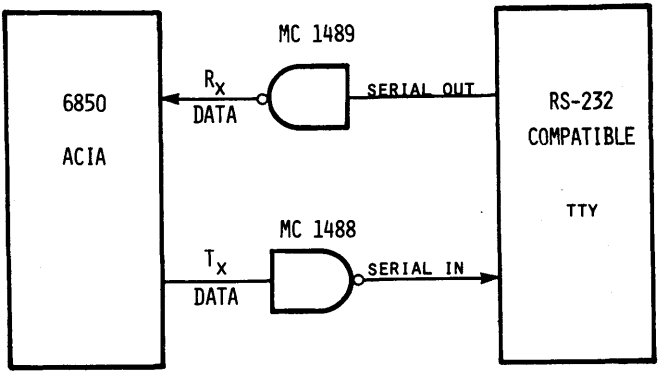
the TTL signals are converted into 20 milliamp current loop signals required by the TTY.

In Fig. 4-23, *opto-isolators* are used to isolate the teletype electrically from the microcomputer system. *This requires that the + and − 12 volts levels also be isolated from the microcomputer.* The ACIA performs the conversion and interfaces directly with the 6800 bus.



4-23 Opto-Isolated TTY Interface

Some teletypes are equipped with EIA-RS232C in a serial configuration. In RS232C teletypes, + and − 12 volt pulses rather than the presence or absence of 20 milliamp currents are used. Fig. 4-24 illustrates a common set of devices for EIA to TTL and TTL to EIA level conversion. These are the MC1489 and MC1488 integrated circuits. There are four translators in each package so a number of lines may be interfaced.



4-24 MC1448 and MC1489

```
NEXT 1   LDA A STACON     LOAD STATUS
         ASR A            SHIFT RDRF BIT TO C-BIT POSITION
         BCS FRAM         CHECK RDRF BIT
         ASR A
         ASR A            SHIFT DCD BIT TO C-BIT POSITION
         BCC NEXT 1       CHECK DCD BIT
         BR ERROR 2       CARRIER LOSS - BRANCH TO ERROR ROUTINE
FRAM     ASR A
         ASR A            SHIFT FE BIT TO C-BIT POSITION
         BCC OVRN         CHECK FE BIT
         BR ERROR 3       FRAMING ERROR - BRANCH TO ERROR ROUTINE
OVRN     ASR A            SHIFT OVRN BIT TO C-BIT POSITION
         BCC PAR          CHECK OVRN BIT
         BR ERROR 4       OVERRUN ERROR - BRANCH TO ERROR ROUTINE
PAR      ASR A            SHIFT PE BIT TO C-BIT POSITION
         BCC R DATA       CHECK PE BIT
         BR ERROR 5       PARITY ERROR - BRANCH TO ERROR ROUTINE
R DATA   LDA B TXRX       LOAD B REGISTER WITH DATA
         RTS              RETURN FROM SUBROUTINE
```

6800 Transmit Subroutine (1)

```
NEXT     LDA A STACON     LOAD STATUS
         ASR A
         ASR A            SHIFT TDRE BIT TO C-BIT POSITION
         BCC TX DATA      CHECK TDRE BIT
         ASR A
         ASR A            SHIFT CTS BIT TO C-BIT POSITION
         BCC NEXT         CHECK CTS
         BR ERROR 1       CARRIER LOSS - BRANCH TO ERROR ROUTINE
TX DATA  STA B TXRX       STORE CHARACTER IN ACIA
         RTS              RETURN FROM SUBROUTINE
```

6800 Transmit Subroutine (2)

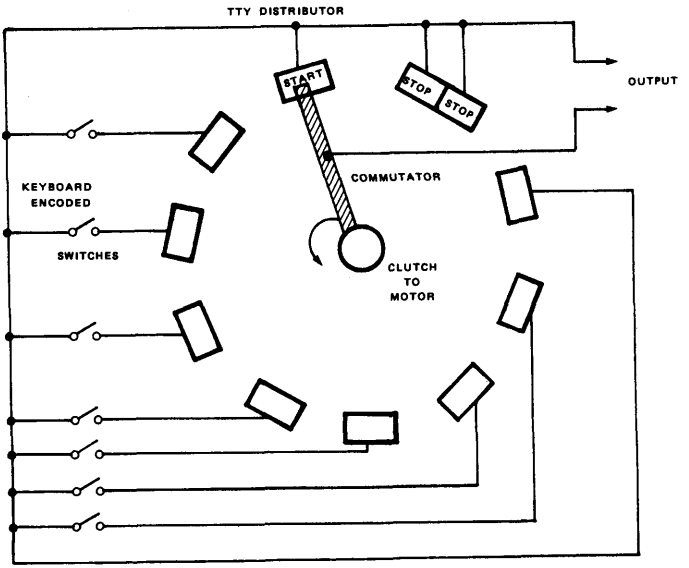
Mechanically, the teletype appears complex, but is really quite simple. To help understand the serial data format, an explanation of what happens internally will be presented.

When the start bit comes in, two things happen: the clutch engages all

mechanical linkages so that a print cycle will occur, and prepares the decoding selector-magnet for the decoding process. The next eight bits come in, 9.09 milliseconds apart. They each trip the selector-magnet, which stops eight notched wheels from spinning—one after the other. In turn, the print bars which select the character on the print-head are raised, or lowered, due to the combination of notches on the wheels. The print-head selects the proper character and the print-hammer strikes the head onto the ribbon and paper. The stop-bits are required to allow enough time to finish the present character before another comes along.

If the punch was on, the selection of the print-bars would also send punches through the paper-tape, while printing the character.

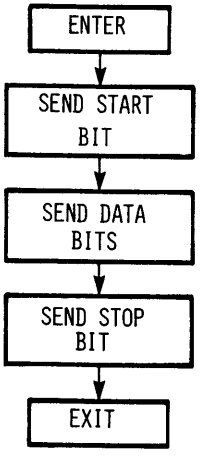
When a key is pressed, the proper bit pattern is placed on eight contacts on the *distributor*. The distributor is like the spark-distributor in an automobile. Fig. 4-25 illustrates the simplicity of this scheme. The motor is engaged to turn the commutator once around, which opens and closes the loop generating the 11-bit pattern for that key.

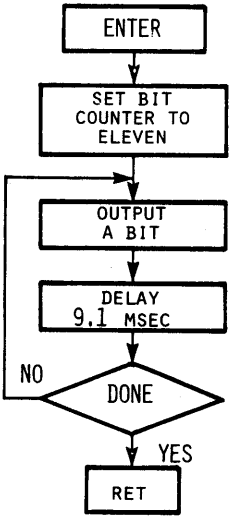


4-25 Distributor in Teletype

Note that the synchronous motor is the timing source for the machine, and an accurate line frequency is necessary, or else the machine will lose sync due to old age, no oil, or other mechanical problems.

A Teletype Output Subroutine





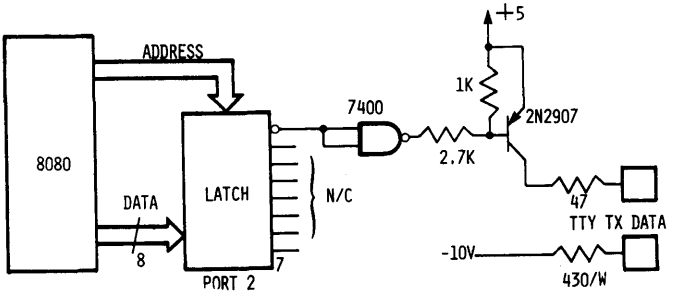
4-26 ACIA Transmit Software

It is assumed here that the teletype is connected to bit 0 of port 2. This simple program will shift-out the 11 bits necessary to represent the character in teletype format. The flow-chart appears on Fig. 4-26, the actual connection appears on 4-27. The program appears below. Register B is used as

```
TELETYPE OUTPUT SUBROUTINE          (ASSUME TTY CONNECTED TO PORT 2 BIT
                                                                    0)
        ;
        ;  THIS SUBROUTINE ENTERED WITH CHARACTER TO BE OUTPUT IN THE
        ;                                                    C REGISTER
        TYOUT:   MVI    B,11     ;  SET COUNTER FOR 11 BITS
                 MOV    A,C      ;  CHARACTER TO ACCUMULATOR
                 ORA    A        ;  CLEAR CARRY-FOR START BIT
                 RAL             ;  MOVE CARRY TO A(0)
        MORE:    OUT    2        ;  SEND TO TTY
                 CALL   DELAY    ;  KILL TIME
                 RAR             ;  POSITION NEXT BIT
                 STC             ;  SET CARRY-FOR STOP BITS
                 DCR    B        ;  DECREMENT BIT COUNTER
                 JNZ    MORE     ;  DONE?
                 RET             ;  YES
        ;
        ;  9 MSEC DELAY (ASSUME NO WAIT STATES)
        ;
        DELAY:   MVI    D,6
        DL0:     MVI    E,2000
        DL1:     DCR    E        ;  1.5 MSEC
                 JNZ    DL1      ;  INNER LOOP
                 DCR    D
                 JNZ    DL0
                 RET
```

8080 TTY Output Program

a counter. It is initially set to 11. The contents of register B will be decremented every time that the bit is shifted out, i.e. transmitted to port 2. It is important to remember that only bit 0 of the accumulator matters in this example. All other bits will be ignored. This is the right-most bit, or least significant bit (LSB) of the accumulator. Initially the accumulator contains the 8 bits to be transmitted. In addition, both the start-bit and the stop-bit must be transmitted. This will be accomplished by using a feature of the rotation instruction of the 8080. The carry bit, which is in fact the ninth bit of the accumulator, in shift operations, will be set to 0. It will then be rotated into the accumulator in bit position 0. This will be the start bit. The crux of the operation is to use a *rotate* instruction. If the contents of the accumulator were simply *shifted* left, the left-most bit would be lost. In this case the left-most bit is preserved in the carry, while a 0 gets written in bit position 0. It will be noted, in the program, that the next operation in the accumulator will be a right rotation. It will re-install the former bit 7, which had been preserved in the carry bit, in its correct position. Finally once this has been done, successive rotations will rotate into the left of the accumulator successive ones created in the carry bit. This will guarantee that the stop bits get transmitted at the end. The sequence of the program is straight forward:



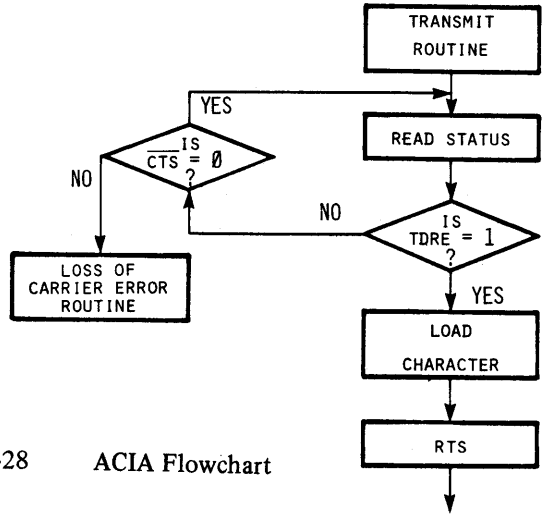
4-27 Hardware TTY Connection

The counter register B is set at value 11, the character which was preserved in register C is loaded into the accumulator A. The accumulator is ored with itself (third instruction). This does not change its contents, but guarantees that the carry is set to 0. This will be the start bit. A right rotate is performed: RAR. This moves the carry into bit position 0 of the accumulator. The output then occurs: OUT 2. The bit is sent to the teletype. Everytime that the bit is sent to the teletype, a delay-loop must be executed to guarantee a 9 ms delay. The delay-routine is implemented as subroutine,

and appears at the bottom of the program. Next, an RAR is executed to shift into bit position 0 the correct next bit. The carry is set in anticipation of ulterior rotations to guarantee that eventually the start bits will be correctly transmitted. The bit-counter (register B) is then decremented and tested. If the counter reaches the value 0, the program ends. If not, the program loops by going back to address MORE, where the next output occurs.

Software Example for ACIA:

This subroutine sends a character to the teletype. If it is not ready to transmit, the subroutine *waits* until ready. It also checks the clear-to-send input (CTS) on the ACIA. This will be used with an EIA-RS232C interface system.



4-28 ACIA Flowchart

The first instruction loads the status of the ACIA into accumulator A. The ready-to-transmit flag is in bit position 1, so it must be shifted twice to the right, into carry, to be tested. If we are ready to transmit, the program goes directly to DATA where the contents of accumulator B are sent to the ACIA.

If the ACIA was not ready to send, the CTS bit would be checked; if it was clear-to-send, a carrier-loss would be indicated and the program would branch to an error routine. If the ACIA is clear-to-send, the transmit-ready flag would be checked until ready. This is a polling technique. Interrupts could also be used.

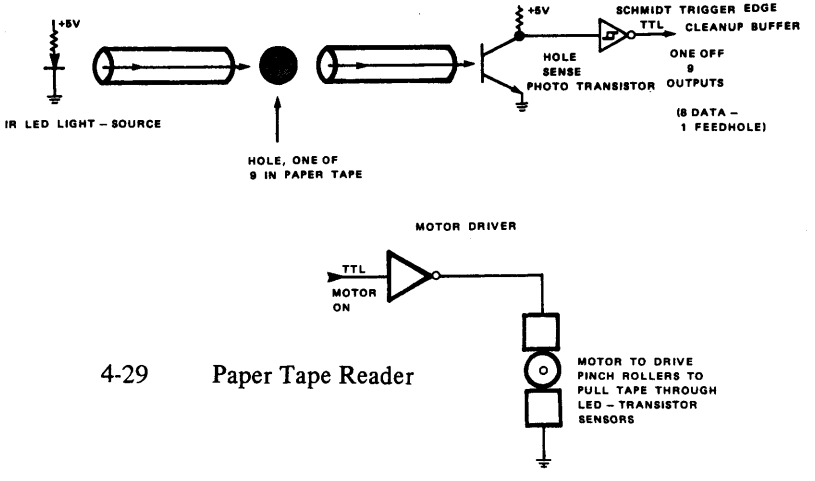
```
NEXT     LDA A STACON        LOAD STATUS
         ASR A
         ASR A               SHIFT TDRE BIT TO C-BIT POSITION
         BCC TX DATA         CHECK TDRE BIT
         ASR A
         ASR A               SHIFT CTS BIT TO C-BIT POSITION
         BCC NEXT            CHECK CTS
         BR ERROR 1          CARRIER LOSS - BRANCH TO ERROR ROUTINE

TX DATA  STA B TXRX          STORE CHARACTER IN ACIA
         RTS                 RETURN FROM SUBROUTINE
```

ACIA Software

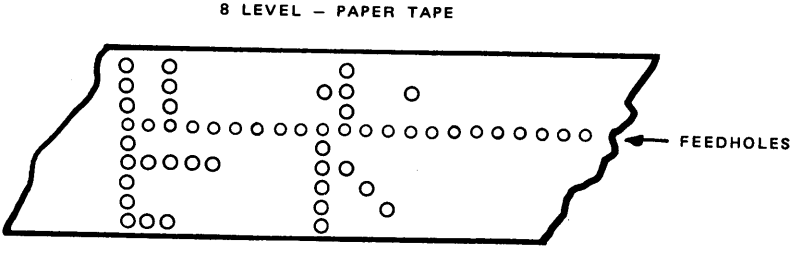
PAPER-TAPE-READER

The teletypewriters usually are slow for reading punched tapes. One helpful peripheral would be a high-speed paper-tape reader. Such a device would optically detect the code pattern on the paper-tape and advance to the next frame quickly. A typical reader has the schematic shown on Fig. 4-29.



4-29 Paper Tape Reader

Our microcomputer must turn-on the motor, sense a feedhole (which are smaller than the data, indicating the center of a bit frame), sense the frame-pattern and store the data before the next feed-hole passes by. When an end-of-tape character is sensed, the reader-motor should turn-off.



4-30 8 Level Bit Frame

A bit-frame for our 8 level tape appears in Fig. 4-30. A typical problem is caused by the ragged edges of the holes, or by dirt on the tape. The hole data appears on Fig. 4-31. Due to this, the feedhole sensing might need some extra delay so that the middle of the feedhole will be the time at which the other holes are sampled. One must know the motor speed to do this.

Some systems can go forward and backwards so that blocks of data with errors may be re-read.

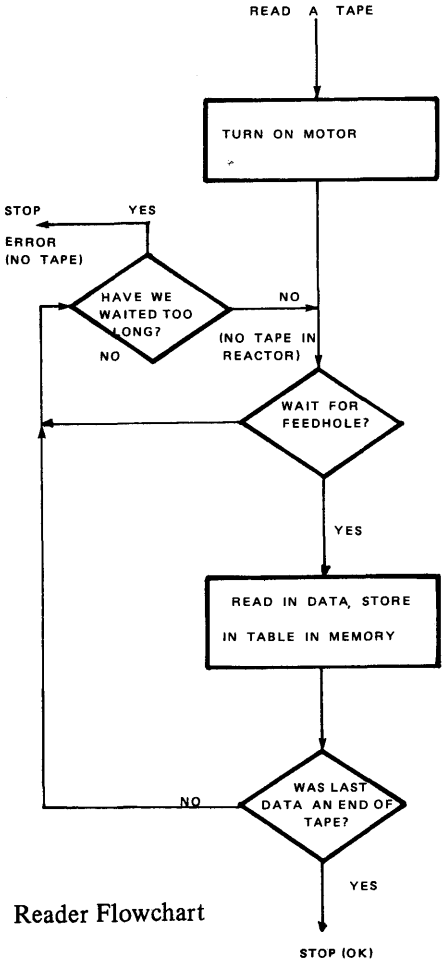
The flowchart for this reader appears on Fig. 4-32.

4-31 Hole Data

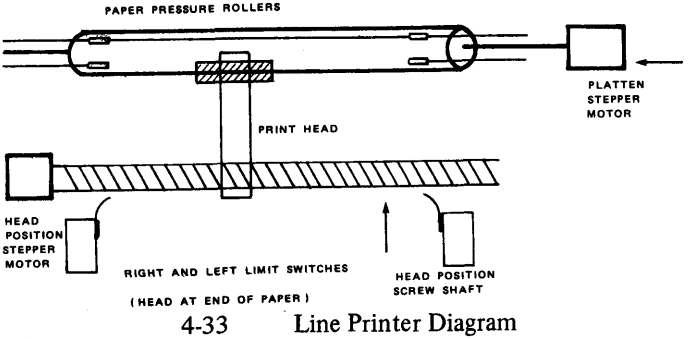
LINE PRINTER

Usually, a teleprinter is too slow for printing long files. In this case, a *line-printer* must be used. There are many types of line-printers. The one described here is a *dot-matrix* type which uses seven print wires to make seven dots on the page. By stepping the print wires across the page and advancing the paper, characters are formed. The basic diagram of our printer appears on Fig. 4-33.

To control this device, we need to step the platen, step the head position, and specify the right combination of seven dots. In addition, we should monitor the head position so that it does not print off the paper.



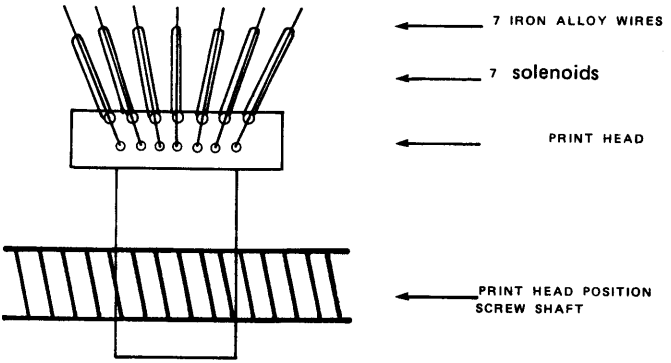
4-32 Reader Flowchart



4-33 Line Printer Diagram

To move the head and platen, *stepping motors* will be used. These motors can move by a small amount, each time they are pulsed. Some are accurate to over 1000 steps per rotation. The ones used here will have 32 steps for the head-motor, and 32 steps for the platen-motor.

Each character printed will be on a 7 × 9 dot-matrix. The head will step once to put it at the next character position. The print wires will print the top dots of the character. Then the platen will step once. The head wires will print again. The process will repeat until the character is finished. When a character has been printed, the head will position itself for the next character.



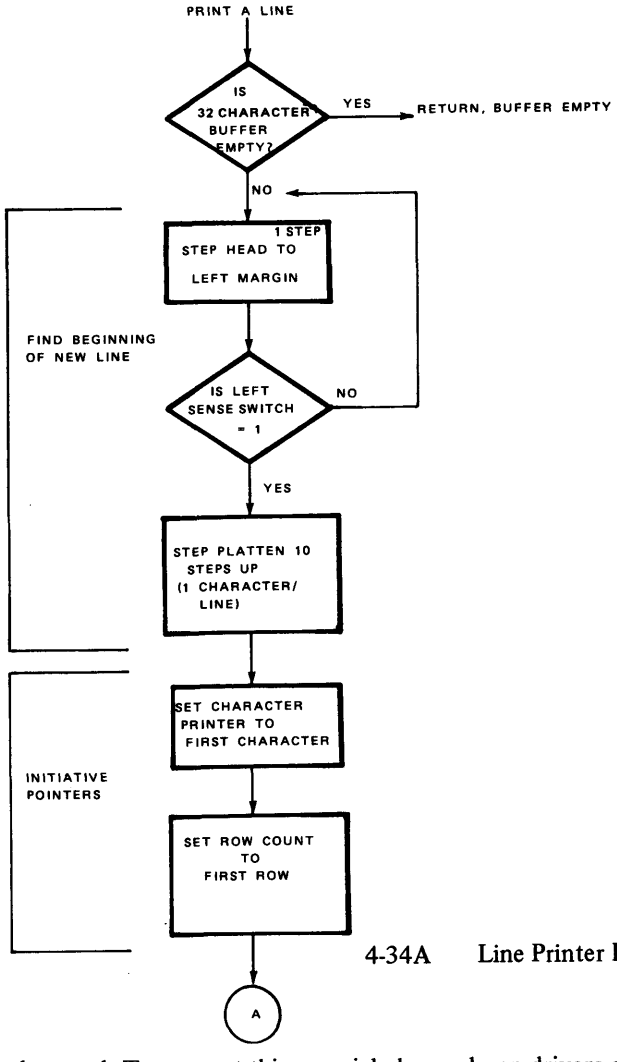
Print Head Detail

But wait—the platen is no longer at the top of the character. To prevent unnecessary stepping, the whole row of the tops of characters will be printed before advancing. The same will be true for each row of dots. Because of this, we will need to *buffer* a complete line before starting to print.

Now, complete 7 by 9 characters may be typed 32 per line. In addition, *pseudo-graphics* may be added by using only the top row as dots and using the full range of platen steps. Fig. 4-34 illustrates the flowchart for the printer interface.

The program will advance to the new line starting position after checking if enough data is present to print. The program will then print dot row by dot row each character on the line.

One important hardware consideration for this printer is that the print wire solenoid will be damaged if left on too long. If our control program were to "crash", or any hardware were to malfunction, the print solenoids

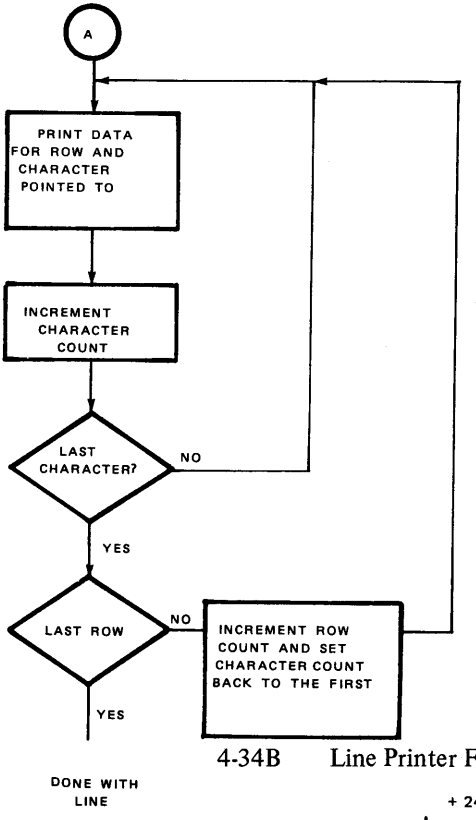


4-34A Line Printer Flowchart

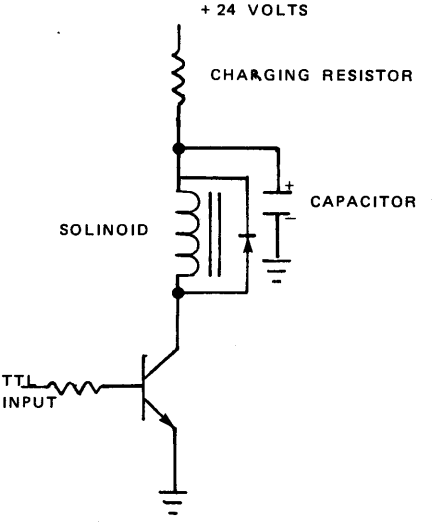
may be damaged. To prevent this, special charge dump drivers are used to trigger the solenoids. This is illustrated for one solenoid on Fig. 4-35.

When the transistor conducts, the energy stored in the capacitor will be used to fire the solenoid and drive the print wire against the ribbon. If the input remains on for too long, the current is limited by the charging resistor.

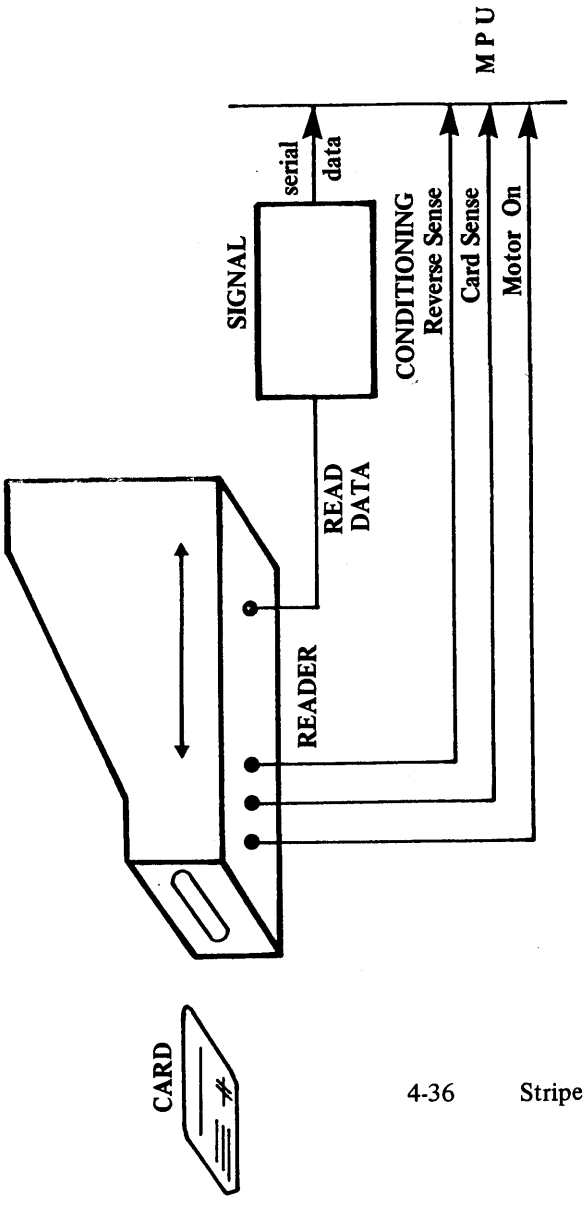
When the transistor stops conducting, the capacitor will charge through the charging resistor so that the circuit will be ready for the next pulse to fire.



4-34B Line Printer Flowchart, Continued



4-35
Change Dump Solenoid Driver



4-36 Stripe Reader

The diode across the solenoid protects the transistor and capacitor from the inductive voltage spike caused by the collapse of the magnetic field in the solenoid, when it is shut-off.

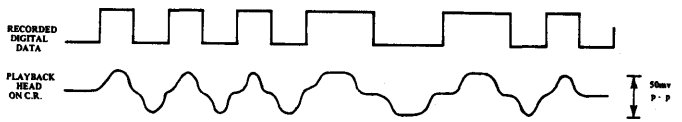
This type of printer is most common in the new point-of-sale terminals. It is inexpensive, has few moving parts and the interface can be done mostly with software routines.

MAGNETIC-STRIPE-CREDIT-CARD CARD-READER

One of the latest developments in the technology has been the use of encoded stripes on the backs of charge or bank-cards to carry information about the bearer's account. Described here is an interface for just such a stripe reader. Fig. 4-36 shows the block diagram of the interface.

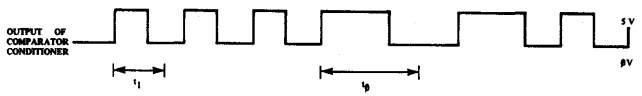
The program will control the decoding of the information on the stripe and the movement of the card in the reader. In normal operation, the card will be sensed at the pressure roller, the drive will be turned on, and the card will be read. If the data is bad or represents a forgery, the card will be "eaten" by the reader. If valid, the card will be returned.

We will assume that the card has been recorded in F2F coding, ("frequency-double frequency"), where each "1" bit is two transitions, and each "0" bit is one transition, per bit cell. Thus, the data off the head may appear as in Fig. 4-37, second trace down.



4-37 Recorded Data

In order to use this signal, it must be conditioned. An analog pulse amplifier-detector will produce an output like the one in Fig. 4-38. The software, through timing loops, may then decode the waveform, back into serial bits, and then into characters. In order to insure proper data, and security, data should be written *three times* in a scrambled form, with various parity checks and heading, and trailing blocks of ones or zeroes.



4-38 Final Data, Read Back and Filtered

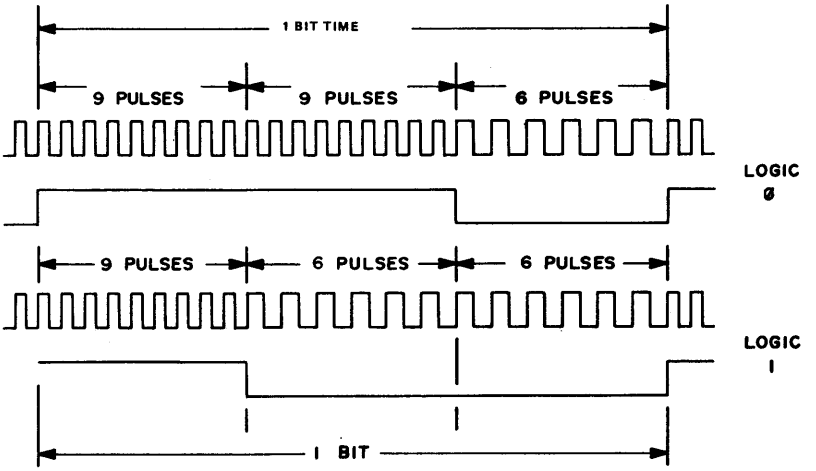
If it is necessary to write on the card, it can be read while going in and written while being returned. One must have a special software routine to reverse the sense of the data so that it can be read again upon reentry.

The control necessary will be three inputs: card-in sense, serial-data read, end-of-card sense (reverse motor to return); and two outputs: motor-on (automatically will reverse, unless turned-off), and serial-data-to-be-written. Thus, one half of a 6820 PIA or 8255 PPI will be sufficient input-out hardware!

THE KIM CASSETTE INTERFACE

In order to save programs and reload them when needed, some form of long-term storage is necessary. The inexpensive portable cassette tape-recorder can be used without modification to store and load digital information. The interface required is simple to build, and easy to program. Described here is the KIM-I® interface to a cassette-recorder.

The format for transmission will need to convert the binary information in memory into a serial stream of bits that can be recorded on the tape. The logic conditions will be represented by the combination of two tones: 3700 hertz and 2400 hertz. The signals for a "1" or a "0" are illustrated on Fig. 4-39.

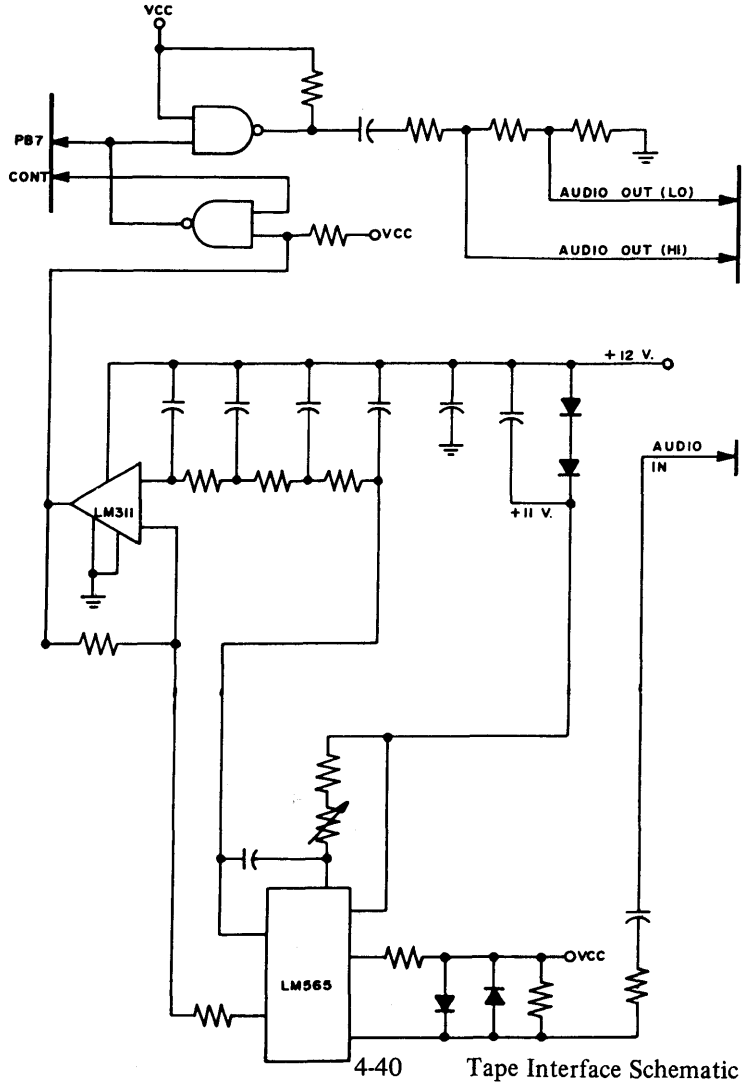


4-39 Bit Format for KIM-I Cassette
Two Tone Combination Recording

®MOS Technology Registered Trademark.

The program will generate these tones by counting loops that will generate either tone. This will use one output bit from the programmable interface and ROM-chip on the board. This output bit will be buffered and filtered to conform to the input specifications of most tape recorders.

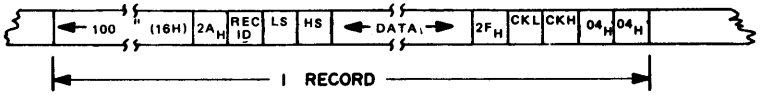
When a tone is sensed, the phase lock loop circuit on the board will differentiate between a 3700 hertz or 2400 hertz tone. By timing the duration of the tones, the data bits may be decoded. Fig. 4-40 is the complete tape recorder interface schematic.



4-40 Tape Interface Schematic

Note that different means of modulation exist that will result in higher densities. Because all timing for transmit and receive is done in software, different timing schemes may be implemented. However, the method described here is the most reliable, as the tape-recorders are not well suited to any higher density recording, due to wow and flutter problems. A high quality tape-deck may be used at higher densities if necessary.

The software breaks each byte of data into two 4-bit nibbles. Each nibble is then converted to a seven-bit ASCII character, plus parity. Two such ASCII characters now represent the original data byte. In order that the recorded block of data be identified, a header and trailer are added. The format appears on Fig. 4-41.



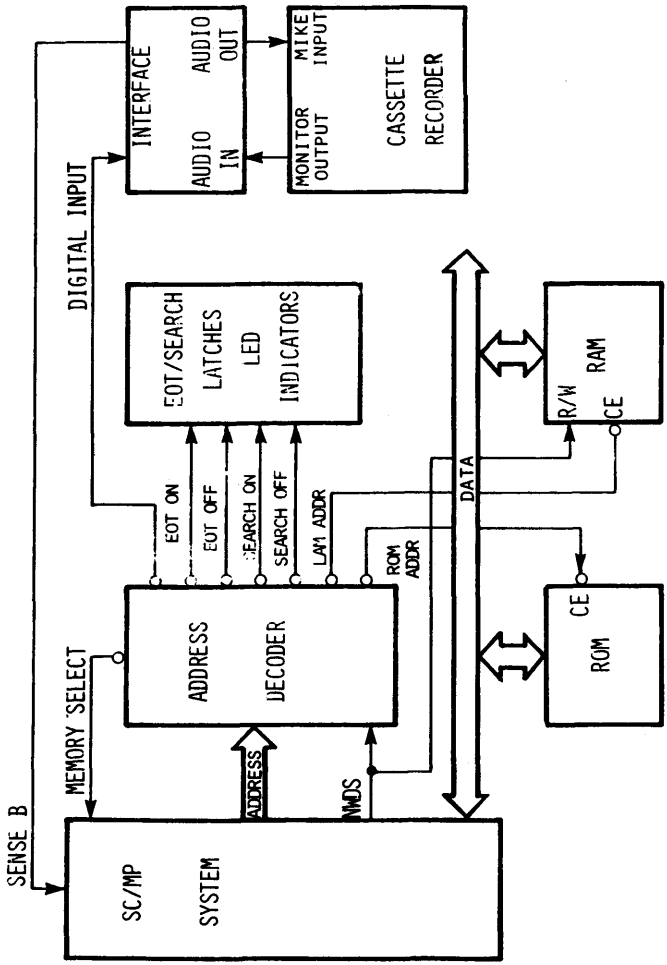
4-41 Tape Data Format

The long block of one hundred 16 hexadecimal bytes allows the software to synchronize to the data rate and find the first bit of each byte without any other timing information. Following the sync characters, come the start-of-record character, and record-number bytes. After that, the starting address of the data-block, and the block itself are written. At the end, a "2F" hexadecimal is written, as well as two check-characters. After that, two "04" hexadecimal are written, to indicate the end of the block.

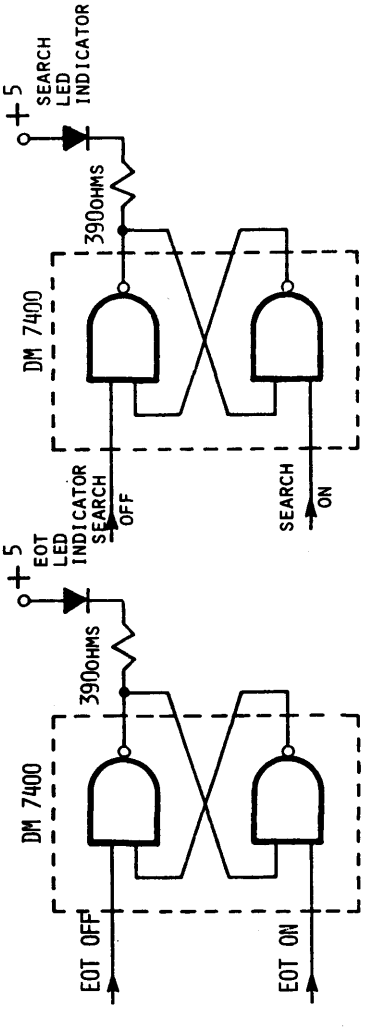
This format is typical of many block-synchronous transmission schemes used. Other examples are the floppy-disk, magnetic-stripe card-reader, and inter-machine communications links (see Chapter 6 for the latter).

SC/MP Cassette Interface

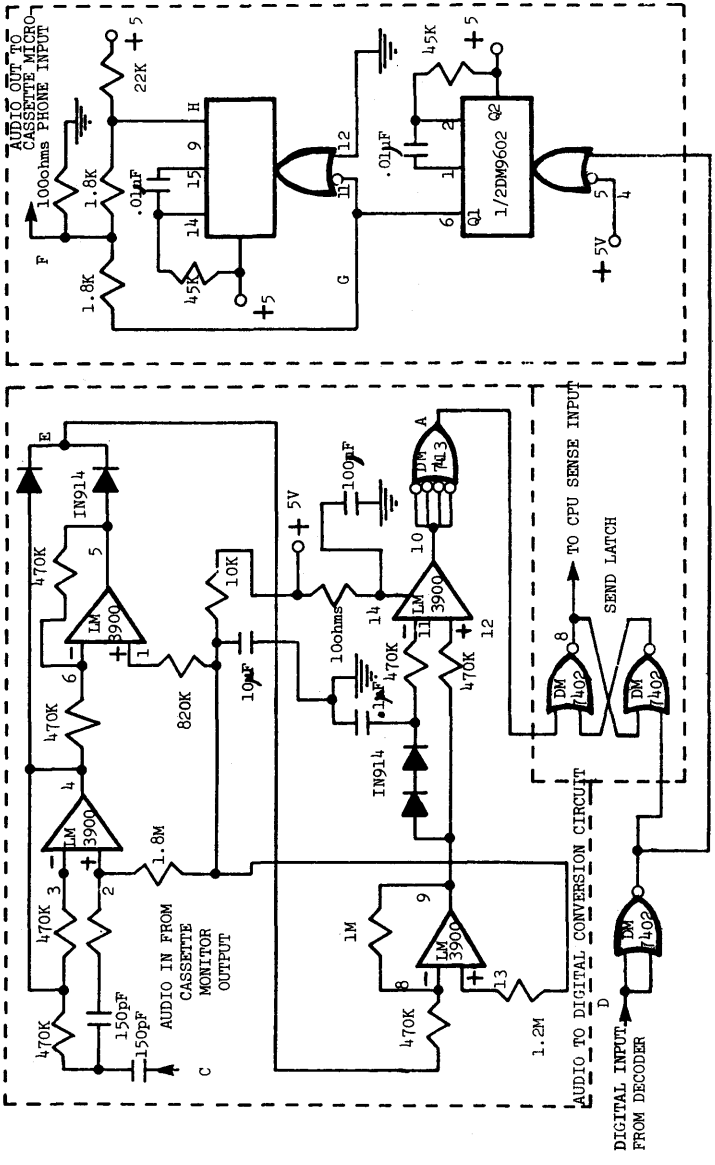
Peripheral decoding is illustrated on Fig. 4-41.1, and the interface circuit on 4-41.2. Memory address decoding appears on Fig. 4-41.3. Finally, the message format is illustrated on 4-41.4. The message format starts with a leader containing 128 bytes of 0's. It is followed by a single byte identification word containing hexadecimal "85". It is then followed by the user-defined addesses entered in 16-bit locations. They are the starting addess of the program being stored, the entry-point address, and the length. The program itself follows, and is terminated by a one byte check-sum. The actual routines used for this interface are presented in Figs. 4.41.5.



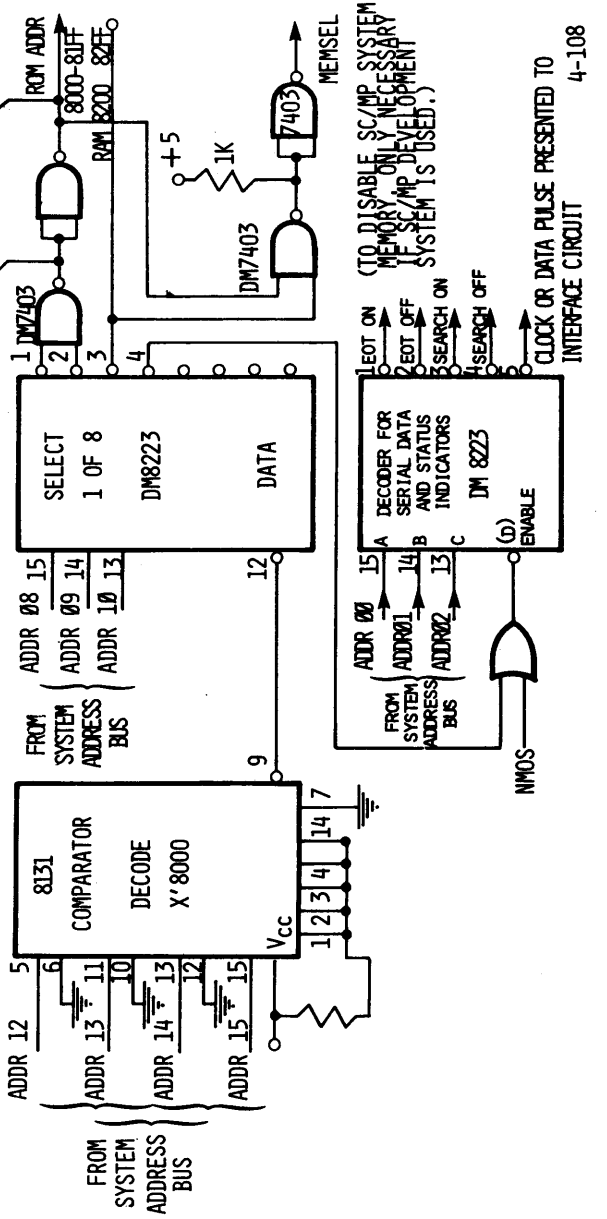
Overall SC/MP Interface



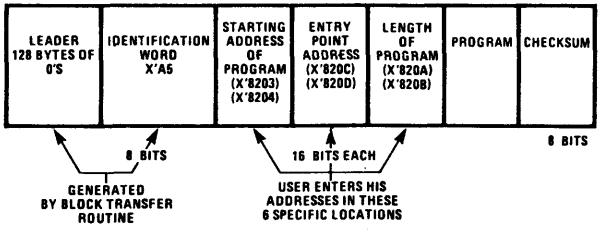
SC/MP Peripheral Decoding



SC/MP Interface Circuit



Memory Address Decoding



Message Format

```
                        TITLE   TAPE10,     SC/MP ROUTINES'

0000                    = X'0000

8200    RAM     =       X'8200          ;  RAM ADDRESS FOR POINTER
8300    PERIPH  =       X'8300          ;  PERIPHERAL ADDRESS FOR POINT

0003    P3      =       3               ;  POINTER #3
0002    P2      =       2               ;  POINTER #2
0001    P1      =       1               ;  POINTER #]

        ;  TEMPORARY DATA IN RAM

0000    CNTU    =       0               ;  INSIDE COUNTER FOR LEADER
0001    CNTL    =       1               ;  OUTSIDE COUNTER FOR LEADER
0002    CKSUM   =       2               ;  CHECK SUM COUNTER
0003    STARTU  =       3               ;  STARTING ADDRESS (UPPER)
0004    STARTL  =       4               ;  STARTING ADDRESS (LOWER)
0005    BITCNT  =       5               ;  BIT COUNTER
0006    TEMP1   =       6               ;  TEMPORARY STORAGE LOCATIONS
0007    TEMP2   =       7               ;  TEMPORARY STORAGE LOCATIONS
0008    TEMP3   =       8               ;  TEMPORARY STORAGE LOCATIONS
0009    TEMP4   =       9               ;  TEMPORARY STORAGE LOCATIONS
000A    WDCNTU  =       10              ;  WORD COUNT (UPPER)
000B    WDCNTL  =       11              ;  WORD COUNT (LOWER)
000C    JUMPU   =       12              ;  TRANSFER ADDRESS (UPPER)
000D    JUMPL   =       13              ;  TRANSFER ADDRESS (LOWER)

        ;  PERIPHERAL ORDER CODES

0000    EOTON   =       0               ;  END OF TAPE LED ON POINTER
0001    EOTOFF  =       1               ;  END OF TALE LED OFF POINTER
0002    SRCHON  =       2               ;  SEARCH LED ON POINTER
0003    SCHROF  =       3               ;  SEARCH LED OFF POINTER
```

```
35          ØØØ4  FLAG    =         4              ;  READ/WRITE FLAG
36
37
                              PAGE 'BOOTSTRAP LOADER'
38
39
40                ;  BOOTSTRAP LOADER ROUTINE.  RECEIVES PROGRAM FROM TAPE.
41                ;  ALL NECESSARY INFORMATION FOR LOADING IS ON TAPE.
42
43                ;  THIS PROGRAM MAY BE REASSEMBLED TO ADDRESS X'ØØØØ TO
44                ;  FUNCTION AS A POWER-ON LOADER.
45
46                ;  INPUT OPERATION OF LED INDICATORS:
47
48                ;       SEARCH LED ON WHEN PROGRAM STARTS
49                ;       SEARCH LED OFF WHEN IDENTIFIER CHARACTER RECEIVED
50                ;       END OF TAPE (EOT) LED ON WHEN RECEPTION COMPLETE
51                ;       SEARCH LED ON IF CHECKSUMS COMFARE
52
53                ;       CONTROL IS THEN TRANSFERRED TO USER PROGRAM
54
55
56  ØØØØ  Ø8    BOOT:  NDP                          ;  FOR RELOCATION TO X'ØØØØ
57  ØØØ1  C4ØØ         LDI       L(RAM)            ;  INITIALIZE RAM POINTER
58  ØØØ3  32            XPAL      P2                ;  IN P2
59  ØØØ4  C482          LDI       H(RAM)
60  ØØØ6  36            XPAH      P2
61  ØØØ7  C4ØØ         LDI       Ø                 ;  CLEAR ACCUMULATOR
62  ØØØ9  CRØ2         ST        CKSUM(P2)         ;  INITIALIZE CHECKSUM COUNTER
63  ØØØB  C4ØØ         LDI       L(PERIPH)         ;  PUT PERIPHERAL POINTER
64  8ØØD  33            XPAL      P3                ;  IN P3
65  8ØØE  C483          LDI       H(PERIPH)
66  8Ø1Ø  37            XPAH      P3
67  8Ø11  CBØ2          ST        SRCHON(P3)        ;  TURN ON SEARCH LED
```

```
 68  8Ø13  CBØ1          ST      EOTOFF(P3)      ;  TURN OFF END OF TAPE LED
 69  8Ø15  C4ØØ          LDI     L(PERIPH)       ;  CLEAR ACCUMULATOR
 70  8Ø17  Ø1            XAE                     ;  CLEAR E REGISTER
 71  8Ø18  C48F          LDI     L(GETBIT)-1     ;  PLACE ADDRESS OF GET BIT
 72  8Ø1A  31            XPAL    P1              ;  IN P1
 73  8Ø1B  C4ØØ          LDI     H(GETBIT
 74  8Ø1D  35            XPAH    P1
 75  8Ø1E  3D    LOCID:  XPPC    P1              ;  GO TO GETBIT FOR INPUT
 76  8Ø1F  4Ø            LDE
 77  8Ø2Ø  E4A5          XRI     X'A5            ;  CHECK FOR PROPER ID CHARACTER
 78  8Ø22  98Ø2          JZ      SETPNT          ;  IF ID RECEIVED, TAKE REST OF
 79  8Ø24  9ØF8          JMP     LOCID           ;  PROGRAM, ELSE GET NEXT BIT.
 80  ØØ26  CBØ3  SETPNT: ST      SRCHOF(P3)      ;  TURN OFF SEARCH LED
 81  ØØ28  C46D          LDI     L(RECV)-1       ;  PLACE ADDRESS OF BYTE RECEIV
 82  8Ø2A  31            XPAL    P1              ;  IN P1
 83  8Ø2B  C48Ø          LDI     H(RECV)
 84  8Ø2D  35            XPAH    P1
 85  8Ø2E  3D            XPAC    P1              ;  GET STARTING ADDRESS (LOWER)
 86  8Ø2F  33            XPAL    P3              ;  AND PLACE IN P3
 87  8Ø3Ø  3D            XPAC    P1              ;  GET STARTING ADDRESS (UPPER)
 88  8Ø31  37            XPAH    P3
 89  8Ø32  3C            XPPC    P1              ;  GET TRANSFER ADDRESS AND
 90  8Ø33  CAØD          ST      JUMPL(P2)       ;  SAVE IN RAM.
 91  8Ø35  3D            XPPC    P1
 92  8Ø36  CAØC          ST      JUMPU(P2)
 93  8Ø38  3D            XPPC    P1              ;  GET WORD COUNT (LOWER)
 94  8Ø39  CAØB          ST      WDCNTL(P2)
 95  8Ø3B  3D            XPPC    P1              ;  GET WORD COUNT (UPPER)
 96  8Ø3C  CAØA          ST      WDCNTU(P2)
 97
 98  8Ø3E  3D    BOOTIN: XPPC    P1              ;  GO TO RECEIVE
 99  8Ø3F  CFØ1          ST      @1(P3)          ;  STORE AND INCREMENT POINTER
100  8Ø41  F2Ø2          ADD     CKSUM(P2)       ;  ADD CHARACTER TO CHECKSUM
101  8Ø43  CAØ2          ST      CKSUM(P2
```

```
102  8045  AA0B          ILD     WDCNTL(P2)     ;  INCREMENT LOWER WORD COUNTER
103  8047  9CF5          JNZ     BOOTIN         ;  CHECK FOR ZERO
104  8049  AA0A          ILD     WDCNTU(P2)     ;  INCREMENT UPPER WORD COUNTER
105  804B  9CF1          JNZ     BOOTIN         ;  CHECK FOR END OF TRANSMISSION
106  804D  3D            XPPC    P1             ;  GET CHECKSUM FROM TAPE
107  804E  E202          XOR     CKSUM(P2)      ;  COMPARE TO CALCULATED VALUE
108  8050  9809          JZ      EXECPR         ;  EXECUTE LOADED PROGRAM
109  8052  C400          LDI     L(PERIPH)      ;
110  8054  33            XPAL    P3
111  8055  C483          LDI     H(PERIPH)
112  8057  37            XPAH    P3
113  8058  CB00          ST      EOTON(P3       ;  TURN ON EOT LED TO INDICATE
114  805A  00            HALT                   ;  CHECKSUM ERROR AND HALT.
115
116  805B  C400  EXECPR: LDI     L(PERIPH)
117  805D  33            XPAL    P3
118  805E  C483          LDI     H(PERIPH)
119  8060  37            XPAH    P3
120  8061  CB00          ST      EOTON(P3)      ;  TURN ON END OF TAPE LED
121  8063  CB02          ST      SRCHON(P3)     ;  TURN ON SERACH LED
122  8065  C20D          LD      JUMP(P2)       ;  LOAD TRANSFER ADDRESS
123  8067  33            XPAL    P3
124  8068  C20C          LD      JUMPU(P2)
125  806A  37            XPAH    P3
126  806B  C7FF          LD      @-1(P3)        ;  DECREMENT POINTER FOR FETCH
127  806D  3F            XPPC    P3             ;  EXECUTE
128
129
130                ;  RECEIVE ROUTINE.  RECEIVES ONE 8-BIT CHARACTER INTO
131                ;  ACCUMULATOR.
132
133
134  806E  C48F  RECV:   LDI     L(GETBITO-1    ;  PLACE ADDRESS OF GETBIT
135  8070  31            XPAL    P1             ;  IN P1.
```

```
136  8Ø71  CAØ7           ST      TEMP2(P2)      ;  SAVE CURRENT CONTENTS OF P1
137  8Ø73  C48Ø           LDI     H(GETBIT)
138  8Ø75  35             XPAH    P1
139  8Ø76  CAØ6           ST      TEMP1(P2)
140  8Ø78  C4Ø8           LDI     8              ;  SET BIT COUNT
141  8Ø7A  CAØ5           ST      BITCNT(P2)
142  8Ø7C  C4ØØ           LDI     Ø              ;  CLEAR ACCUMULATOR
143  8Ø7E  Ø1             XAE                    ;  CLEAR E REGISTER
144  8Ø7F  3D    LOOP:    XPPC    P1             ;  GO TO GETBIT
145  8Ø8Ø  BAØ5           DLD     BITCNT(P2)     ;  DECREMENT BIT COUNT
146  8Ø82  98Ø2           JZ      RETRN2         ;  CHECK FOR ZERO
147  8Ø84  9ØF9           JMP     LOOP
148  8Ø86  C2Ø7  RETRN2:  LD      TEMP2(P2)      ;  RESTORE P1 TO ORIGINAL
149  8Ø88  31             XPAL    P1             ;  CONTENTS.
150  8Ø89  C2Ø6           LD      TEMP1(P2)
151  8Ø8B  35             XPAH    P1
152  8Ø8C  4Ø             LDE                    ;  PLACE CHARACTER IN ACCUMULATOR
153  8Ø8D  3D             XPPC    P1             ;  RETURN
154  8Ø8E  9ØDE           JMP     RECV
155
156
157                ;  GET BIT ROUTINE.  RECEIVES 1 BIT INTO E REGISTER
158
159
160  8Ø9Ø  C4ØØ  GETBIT:  LDI     L(PERIPH)      ;  PLACE PERIPHERAL ADDRESS IN P3
161  8Ø92  32             XPAL    P3
162  8Ø93  CAØ9           ST      TEMP4(P2)      ;  SAVE ORIGINAL CONTENTS OF P3
163  8Ø95  C483           LDI     H(PERIPH)
164  8Ø97  37             XPAH    P3
165  8Ø98  CAØ8           ST      TEMP3(P2)
166  8Ø9A  19             SIO                    ;  SHIFT E REGISTER
167  8Ø9B  Ø6    CKSA:    CSA                    ;  COPY STATUS TO ACCUMULATOR
168  8Ø9C  D42Ø           ANI     X'2Ø           ;  MASK
169  8Ø9E  98Ø2           JZ      CLOCK          ;  IF ZERO, BIT RECEIVED
```

```
170  80A0  90F9          JMP     CKSA            ;  CHECK AGAIN.
171  80A2  C400  CLOCK:  LDI     0               ;  CLEAR ACCUMULATOR FOR DELAY
172  80A4  8F01          DLY     1               ;  DELAY 1 MS (1/4 BIT TIME)
173  80A6  CB04          ST      FLAG(P3)        ;  RESET LATCH
174  80A8  C400          LDI     0               ;  INIT ACCUMULATOR FOR DELAY
175  80AA  8F02          DLY     2               ;  DELAY  PAST MIDDLE OF WINDOW
176  80AC  06            CSA                     ;  COPY STATUS TO ACCUMULATOR
177  80AD  D420          ANI     X'20            ;  MASK
178  80AF  9802          JZ      ONE             ;  IF ZERO, THEN BIT IS A "1"
179  80B1  9004          JMP     RESET
180  80B3  40    ONE:    LDE
181  80B4  DC80          ORI     X'80            ;  ADD "1" BIT TO CHARACTER
182  80B6  01            XAE                     ;  SAVE IN E REGISTER
183  80B7  CB04  RESET:  ST      FLAG(P3)        ;  RESET LATCH
184  80B9  06            CSA                     ;  COPY STATUS TO ACCUMULATOR
185  80BA  D420          ANI     X'20            ;  MASK
186  80BC  98F9          JZ      RESET           ;  CHECK IF LATCH IS RESET
187  80BE  C209  RETRN3: LD      TEMP4(P2)       ;  RESTORE P2
188  80C0  33            XPAL    P3
189  80C1  C208          LD      TEMP3(P2)
190  80C3  37            XPAH    P3
191  80C4  3D            XPPC    P1              ;  RETURN
192  80C5  90C9          JMP     GETBIT
193
194                      PAGE  'DATA WRITE ROUTINES'
195
196
197             ;  SEND 4 SECONDS OF "0" (ABOUT 1000) TO ALLOW FOR
198             ;  TAPE TO SETTLE ON PLAY BACK AND ACT AS LEADER.
199
200             ;  OUTPUT OPERATION OF LED INDICATORS:
201
202             ;          SEARCH LED ON WHEN LEADER COMPLETE
```

```
203                ;         SEARCH LED OFF WHEN TRANSMISSION COMPLETE
204                ;         END OF TAPE LED ON WHEN TRANSMISSION COMPLETE
205
206
207  8ØC7  C4ØØ  INIT:   LDI   L(RAM)        ;  PLACE RAM POINTER IN P2
208  8ØC9  32            XPAL  P2            ;       (LOWER)
209  8ØCA  C482          LDI   H(RAM)
210  8ØCC  36            XPAH  P2            ;       (UPPER)
211  8ØCD  C4ØØ          LDI   L(PERIPH)     ;  PLACE PERIPHERAL ADDRESS
212  8ØCF  33            XPAL  P3            ;  IN P3.
213  8ØDØ  C483          LDI   H(PERIPH)
214  8ØD2  37            XPAH  P3
215  8ØD3  C4ØØ  COMP:   LDI   Ø             ;  CLEAR ACCUMULATOR
216  8ØD5  Ø2            CCL                 ;  CLEAR CARRY/LINK FLAG
217  8ØD6  FAØB          CAD   WDCNTL(P2)    ;  FORM 1'S COMP OF LOWER COUNT
218  8ØD8  CAØB          ST    WDCNTL(P2)
219  8ØDA  C4ØØ          LDI   Ø             ;  TURN OFF SEARCH LED
220  8ØDC  FAØA          CAD   WDCNTU(P2)    ;  TURN OFF END OF TAPE LED
221  8ØDE  CAØA          ST    WDCNTU(P2)
222  8ØEØ  CBØ3          ST    SRCHOF(P3)    ;  TURN OFF SEARCH LED
223  8ØE2  CBØ1          ST    EOTOFF(P3)    ;  TURN OFF END OF TAPE LED
224  8ØE4  C4Ø8  SNDLDR: LDI   8             ;  SET OUTER COUNTER
225  8ØE6  CAØ1          ST    CNTL(P2)
226  8ØE8  C48Ø  CNT1:   LDI   X'8Ø          ;  SET INNER COUNTER
227  8ØEA  CAØØ          ST    CNTU(P2)
228  8ØEC  CBØ4  CNT2:   ST    FLAG(P3)      ;  PULSE WRITE FLAG
229  8ØEE  C4ØØ          LDI   Ø             ;  CLEAR ACCUMULATOR
230  8ØFØ  8FØ4          DLD   4             ;  DELAY 1 BIT TIME
231  8ØF2  BAØØ          JNZ   CNT2          ;  DECREMENT INNER COUNTER
232  8ØF4  9CF6          DLD   CNTL(P2)      ;  CHECK FOR ZERO
233  8ØF6  BAØ1          JP    CNT1          ;  CHECK FOR LESS THAN ZERO
234  8ØF8  94EE          ST    SRCHON(P3)    ;  TURN ON SEARCH LED
235  8ØFA  CBØ2
236
```

```
237
238                ;  BLOCK TRANSFER ROUTINE.  SENDS BLOCK OF DATA TO CASSETTE
239
240                ;  THE FOLLOWING ADDRESSES MUST BE LOADED BY USER BEFORE
241                ;  EXECUTING THE WRITE PROGRAM:
242
243                ;  X'82Ø3 -- UPPER 8 BITS OF PROGRAM ADDRESS
244                ;  X'82Ø4 -- LOWER 8 BITS OF PROGRAM ADDRESS
245                ;  X'82ØA -- UPPER 8 BITS OF PROGRAM LENGTH
246                ;  X'82ØB -- LOWER 8 BITS OF PROGRAM LENGTH
247                ;  X'82ØC -- UPPER 8 BITS OF TRANSFRE ADDRESS (ENTRY POINT)
248                ;  X'82ØD -- LOWER 8 BITS OF TRANSFER ADDRESS
249
250
251  8ØFC  C4ØØ   BLOCK:   LDI     Ø              ;  CLEAR ACCUMULATOR
252  8ØFE  CAØ2            ST      CKSUM(P2)      ;  INITIALIZE CHECKSUM COUNTER
253  81Ø2  C481            LDI     H(WRITE)       ;  PLACE ADDRESS OF WRITE IN P1
254  81Ø3  35              XPAH    P1
255  81Ø5  C444            LDI     L(WRITE)-1
256  81Ø6  31              XPAL    P1
257  81Ø8  C4A5            LDI     X'A5           ;  LOAD ACCUMULATOR WITH ID
258  81Ø9  3D              XPPC    P1             ;  WRITE ID ON TAPE
259  81ØB  C2Ø4            LD      STARTL(P2)     ;  GET STARTING ADDRESS
260  81ØC  3D              XPPC    P1             ;  WRITE ONTO TAPE
261  81ØE  C2Ø3            LD      STARTU(P2)
262  81ØF  3D              XPPC    P1
263  8111  C2ØD            LD      JUMPL(P2)      ;  GET TRANSFER ADDRESS
264  8112  3D              XPPC    P1
265  8114  C2ØC            LD      JUMPU(P2)
266  8115  3D              XPPC    P1
267  8117  C2ØB            LD      WDCNTL(P2)     ;  GET LENGTH
268  8118  3D              XPPC    P1
269  811A  C2ØA            LD      WDCNTU(P2)
270  811B  3D              XPPC    P1
```

```
271  811B  C2Ø4  GETBYT:  LD      STARTL(P2)      ;  PLACE CURRENT ADDRESS IN P1
272  811D  31             XPAL    P1
273  811E  C2Ø3           LD      STARTU(P2)
274  812Ø  35             XPAH    P1
275  8121  C5Ø1           LD      @1(P1)          ;  GET CHARACTER THROUGH
276  8123  Ø1             XAE                     ;  POINTER AND SAVE IN E REGISTER.
217  8124  C444           LDI     L(WRITE)-1      ;  GET ADDRESS OF WRITE AND
278  8126  31             XPAL    P1              ;  SAVE CURRENT CONTENTS OF P1.
279  8127  CAØ4           ST      STARTL(P2)
280  8129  C481           LDI     H(WRITE)
281  812B  35             XPAH    P1
282  812C  CAØ3           ST      STARTU(P2)
283  812E  4Ø             LDE
284  812F  F2Ø2           ADD     CKSUM(P2)       ;  UPDATE CHECKSUM
285  8131  CAØ2           ST      CKSUM(P2)
286  8133  4Ø             LDE                     ;  PLACE CHARACTER IN ACC.
287  8134  3D             XPPC    P1              ;  SEND CHARACTER
288  8135  AAØB           ILD     WDCNTL(P2)      ;  INCREMENT WORD COUNTER
289  8137  9CE2           JNZ     GETBYT          ;  CHECK FOR ZERO
290  8139  AAØA           ILD     WDCNTU(P2)
291  813B  9CDE           JNZ     GETBYT
292  813D  C2Ø2           LD      CKSUM(P2)       ;  SEND CHECKSUM TO TAPE
293  813F  3D             XPPC    P1
294  814Ø  CBØ3           ST      SRCHOF(P3)      ;  TURN OFF SEARCH LED
295  8142  CBØØ           ST      EOTON(P3)       ;  TURN ON END OF TAPE LED
296  8144  ØØ             HALT                    ;  HALT WHEN FINISHED
297
298
299                ;  DATA WRITE ROUTINE.  WRITES 1 8-BIT CHARACTER ON TAPE
300
301
302  8145  Ø1     WRITE:                          ;  SAVE CHARACTER IN E REGISTER
303  8146  C4Ø8           LDI     8               ;  SET BIT COUNT
304  8148  CAØ5           ST      BITCNT(P2)
```

```
305  814A  4Ø     MASK:   LDE
306  814B  D4Ø1           ANI     1           ;  MASK
307  814D  9CØ8           JNZ     SEND1       ;  CHECK IF BIT "Ø" OR "1"
308  814F  C4ØØ           LDI     Ø           ;  CLEAR ACCUMULATOR FOR DELAY
309  8151  CBØ4   SENDØ:  ST      FLAG(P3)    ;  PULSE WRITE FLAG
310  8153  8FØ4           DLY     4           ;  DELAY 1 BIT TIME (  4  MS  )
311  8155  9ØØC           JMP     SHIFT
312  8157  C4ØØ   SEND1:  LDI     Ø
313  8159  CBØ4           ST      FLAG(P3)    ;  PULSE WRITE FLAG
314  815B  8FØ2           DLY     2           ;  DELAY TO MIDDLE OF WINDOW
315  815D  CBØ4           ST      FLAG(P3)    ;  PULSE WRITE FLAG
316  815F  C4ØØ           LDI     Ø           ;  CLEAR ACC. FOR DELAY
317  8161  8FØ2           DLY     2           ;  DELAY TO END OF WINDOW
318
319  8163  19     SHIFT:  SIO                 ;  SHIFT E REGISTER
320  8164  BAØ5           DLD     BITCNT(P2)  ;  DECREMENT BIT COUNTER
321  8166  98Ø2           JZ      RETRN1      ;  CHECK FOR ZERO
322  8168  9ØEØ           JMP     MASK        ;  SEND NEXT BIT
323  816A  3D     RETRN1: XPPC    P1          ;  RETURN
324  816B  9ØD8           JMP     WRITE
325
326        8ØØØ           END     BOOT

BITCNT   ØØØ5          BLOCK   8ØFC *      BOOT    8ØØØ
BOOTIN   8Ø3E          CKSA    8Ø9B        CKSUM   ØØØ2
CLOCK    8ØA2          CNT1    8ØE8        CNT2    ØØEC
CNTL     ØØØ1          CNTU    ØØØØ        COMP    8ØD3 *
EOTOFF   ØØØ1          EOTON   ØØØØ        EXECPR  8Ø58
FLAG     ØØØ4          GETBIT  8Ø9Ø        GETBYT  811B
INIT     8ØC7 *        JUMPL   ØØØD        JUMPU   ØØØC
LOCID    8Ø1E          LOOP    8Ø7F        MASK    814A
ONE      8ØB3          P1      ØØØ1        P2      ØØØ2
P3       ØØØ3          PERIPH  83ØØ        RAM     .82ØØ
```

```
RECV     806E          RESET    80B7          RETRN1   816A
RETRN2   8086          RETRN3   80BE  *       SEND0    8151  *
SEND1    8157          SETPNT   8026          SHIFT    8163
SNDLDR   80E4  *       SRCHOF   0003          SRCHON   0002
STARTL   0004          STARTU   0003          TEMP1    0006
TEMP2    0007          TEMP3    0008          TEMP4    0009
WDCNTL   000B          WDCNTU   000A          WRITE    8145

NO ERROR LINES
SOURCE CHECKSUM=C694
FIRST INPUT SECTOR HEX - 03CC
FINAL INPUT SECTOR HEX - 03EE
```

SC/MP CASSETTE PROGRAM

ONE CHIP DIGITAL CASSETTE CONTROLLER

The NEC UPD371D provides in a single chip most of the functions required for interfacing a *digital* cassette-transport. It uses the ISO format and performs:

— Parallel-to-serial and serial-to-parallel data-conversion (functions normally accomplished by a UART)
— Error-detection, including CRC (CRC will be explained in the disk section)
— Data-encoding two-phase encoding format
— It can control up to two cassette-transports with read/write or rewind on one unit, or simultaneous rewind. It interfaces directly to the 8080A. The structure of the system is illustrated on 4-41.6, and its interface to the 8080 appears on 4-41.7.

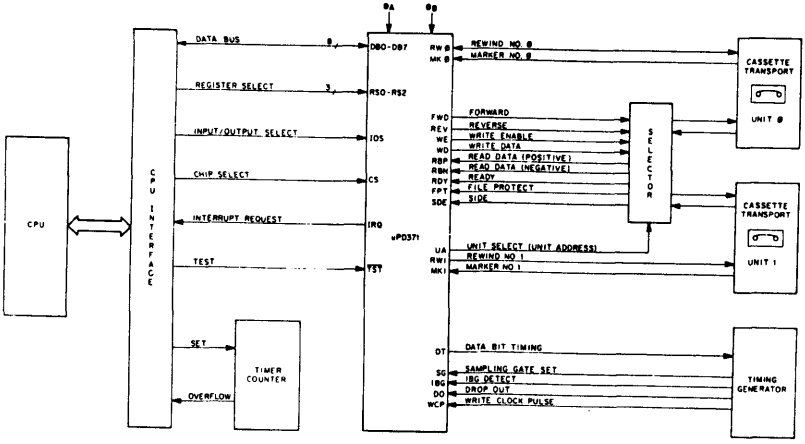


Fig. 4-41.6 NEC UPD37ID Cassette Interface

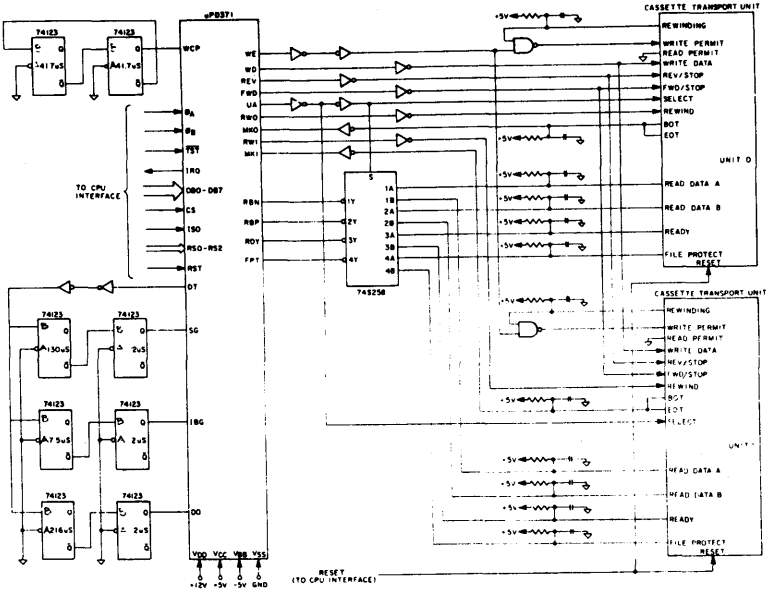


Fig. 4-41.7 8080 Interfaced with UPD37ID

CRT DISPLAY INTERFACE

A number of CRT's displays have been created, to be used specifically as computer terminals. In the microprocessor world, the cost of peripherals is of critical importance. Therefore, the most-often-used CRT-device, in the case of microprocessor systems, is the home television-set. Higher quality CRT-displays are used in the case of development systems, in order to permit the user to display more characters, more lines, or more dots per character. In addition, full graphics capability exists on specialized and expensive displays. We will concentrate here on the direct interface to a television-type display.

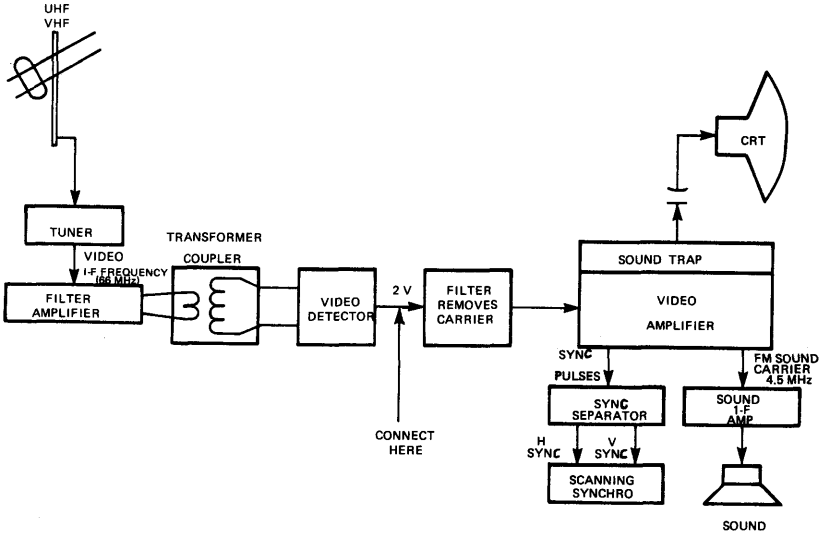


Fig. 4-42 A Television Block Diagram

The organization of a typical television appears on Fig. 4-42. The signal is fed from the antennae into the tuner, which outputs a video i-f frequency, at 4.5 MHz. The signal is fed into a filter-amplifier which is transformer-coupled to the video-detector. The output of the detector is the video-signal proper, with a 2-volt swing. It is fed to a filter, in order to remove the carrier frequency, and then to the video-amplifier. The signal is then split three ways. The video signal is directed to the CRT through a sound trap which eliminates the sound-carrier frequency. The FM sound-carrier is fed to a sound i-f amplifier (4.5 MHz) and the output is fed into the loud-speaker.

Finally, the sync pulses are separated from the video signal, and identified as H (horizontal) sync, and V (vertical) sync. The H-sync and the V-sync are used to synchronize the display on the screen.

The microprocessor system can interface to the television at two points: it can be coupled directly to the television-set antennae—this is the RF modulation method, or else the video signal can be fed directly at the output of the video-detector. This is the direct video input method. The advantage of the RF modulation method is that it does not require any connection inside the set. The output wires of the microprocessor system are simply connected to the antennae screws.

Besides requiring compliance with FCC regulations, the RF modulation method has a bandwidth limitation problem. Using standard television sets, the limit would be from 3 to 3.5 MHz. This limit could be, in fact, significantly lower with lesser quality sets. The bandwidth of the set will severely limit the definition on the screen as well as the total number of characters which can be displayed.

The disadvantage of the direct video input is naturally that it requires a connection within the television-set itself. A few sets are equipped with an external connector for a direct video entry. This is often the case on color television-sets in Europe, but not yet the case in the US.

In order to interface to the television-set, we will review here briefly the principles of television operation, and then present the techniques used to display characters on a screen.

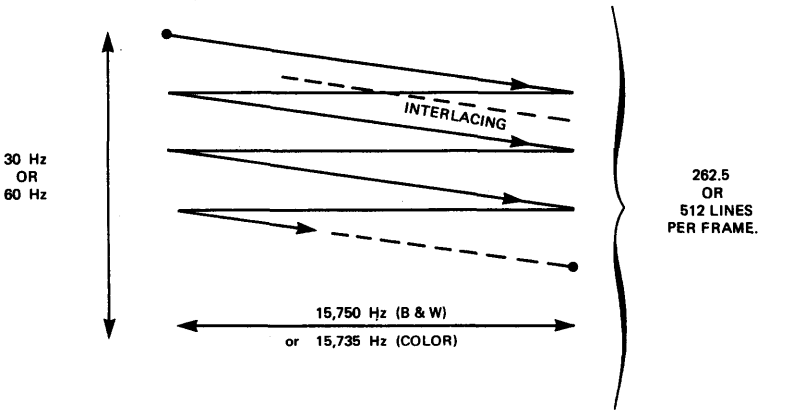


Fig. 4-43 TV Timing

A raster-scan television uses a beam of electrons which is deflected horizontally across the screen, with a varying intensity. When it reaches one end of the screen, the beam is blanked off and it flies back to the other side of the screen, while going down one line. This is called the "horizontal-fly-back" phase. It is illustrated on Fig. 4-43. Two types of scan are used, called respectively the direct-scan and the interlaced-scan. In the interlaced scheme, the screen is scanned *twice*. The second scanning, or field, is made on lines between the previous ones. 262.5 lines are available in each field. An interlaced scheme therefore provides 525 lines per frame. In the case of a TV display connected to a microprocessor, the usual method is not to use interlace, and to use a straight single-scan of the screen on 262 lines. The frame rate is then 60 Hz. Interlaced could be used to provide titles or to superimpose messages or titles on a TV broadcast. Two synchronization signals are used to synchronize the motion of the dot across the screen: The line-sync supplies the flyback signal, and the vertical-sync provides the vertical flyback signal to the beginning of the first line. Some limitations are imposed, which are illustrated on Fig. 4-44. The horizontal scan is usually longer than the screen-size. The amount by which the dot deflects past the end of the screen is called the *screen-overscan*. In addition, the message displayed on the screen is shorter than the screen itself. This is shown as the *display-time* on the illustration. Whenever the dot reaches the end of the display-time, it goes black. The time from the end of the display-time to the line-sync is called the *blank-time*. (See Fig. 4-44.)

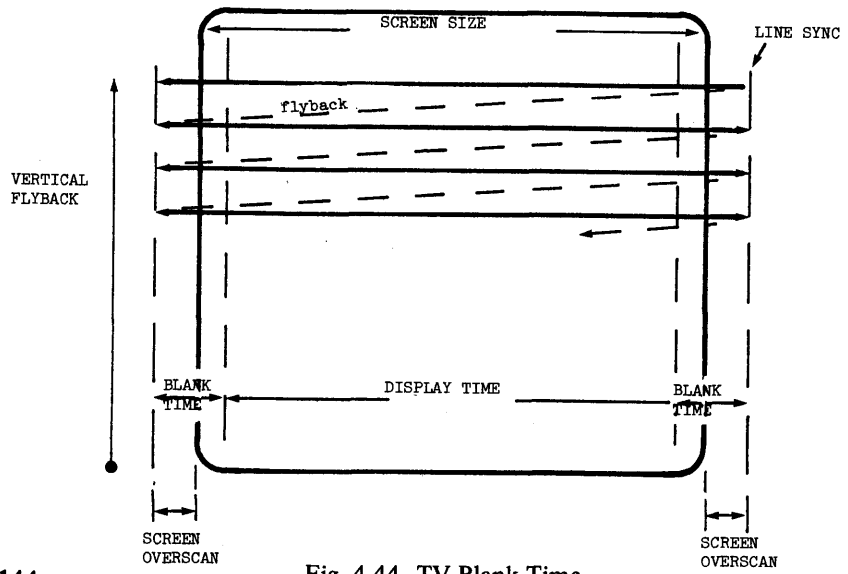


Fig. 4-44 TV Blank Time

Generating Characters

Characters are represented on the screen by a pattern of dots called a *dot-matrix*. Two standard formats are used to represent characters. The most frequently used is the *5 × 7 dot-matrix*. A lesser-used system is the 7 × 9 dot-matrix. The advantage of a 7 × 9 dot-matrix is a better definition of characters, and a more pleasing representation of lower-case letters. However, a 7 × 9 dot-matrix requires the use of a high bandwidth, and, for this reason, is much less used. A 5 × 7 dot-matrix represents each character with 35 dots. It uses 7 rows of 5 dots, and each character is represented by a sequence of dots and un-dots (blank dots or rather "black" dots). The representation of characters is illustrated on Fig. 4-45. Each scan of a TV line will present on the screen the five dots belonging to all the characters of the line. Then, it will present the next row of dots for these characters, and so on. At a minimum, a 5 × 7 dot-matrix will require eight lines on the screen, since one blank line must be used between the characters. In practice, for good visual presentation, ten lines are used, and sometimes twelve, to present a line of characters.

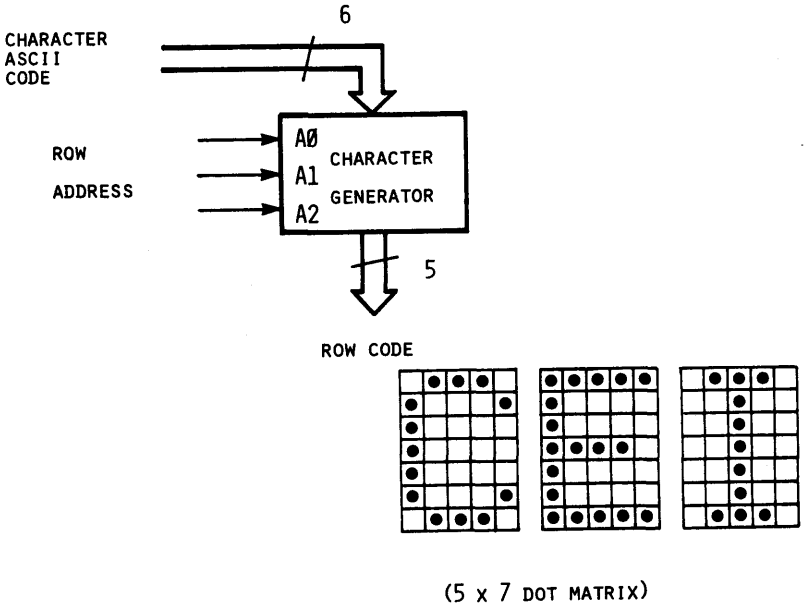
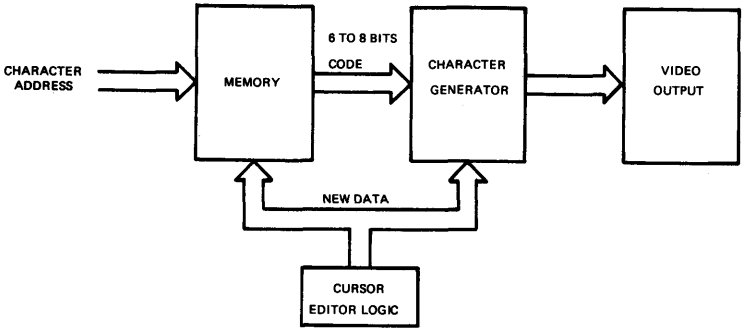


Fig. 4-45 Dot - Matrix Characters

Each character is represented within the microprocessor system by its code, normally ASCII. The table of ASCII codes appears on Fig. 4-46. This seven bit ASCII code must be converted into the dot-matrix representation. This can be accomplished simply by a ROM look-up mechanism. Or a specialized chip may be used, a *dot-matrix character-generator*. When using the generator, the first line of dots for each successive character will be output, then the next one, then the next one, up to the seventh one. A simple counter is used to keep track of the row of dots being currently output. It will be shown in the next sections how the dots are converted into video signals that will be fed to the television-set.



Raster Generation

In addition, the whole picture, or frame, needs to be refreshed at a 60Hz frequency, i.e., 60 times per second to avoid flicker. This implies the necessity of a *refresh-memory*. The timing for refreshing the screen is usually so fast, that a standard microprocessor cannot be used. External circuits such as a DMA, or other special circuits, must be used. The advantage of using a DMA is that the main memory of the microprocessor system can be shared with the screen refresh. However, it slows down the microprocessor's operation. In many cases, *dedicated memory* is used to refresh the screen. In this case, there is no slowdown of the microprocessor's operation.

Character-generators are available from most semiconductor manufacturers, such as Fairchild, General Instruments, Monolithic Memories, MOS Technology, American Microsystems, Electronic Arrays, Signetics, and Texas Instruments.

The number of characters that can be displayed on the screen is limited by the bandwidth of the set being used. Assuming the use of a standard television without modifications, a 5 × 7 dot-matrix will usually be selected, and the popular combination is to use 10 lines of 32 characters, or up to 16 lines of 32 characters, for a total of 512 characters. A complete scan line will require approximately 63.5 microseconds. The usable portion of the scan line will be perhaps 43 microseconds. Displaying 32 characters in 43 microseconds will leave us approximately 1.3 microseconds per character. This leaves plenty of time for using a relatively slow memory. If we were using 80 characters per line, an access time of less than 0.5 microsecond would be required for the memory.

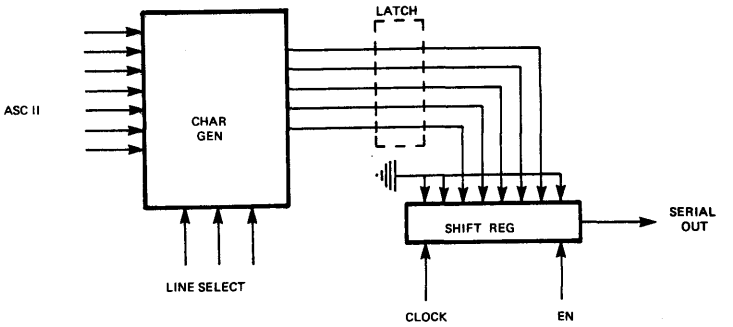


Fig. 4-46 Shift Register Serializes Characters

Converting to Serial Video

The dots coming out of the character generator must how be shifted out into serial form, to be presented as a video signal to the television. This is illustrated on Fig. 4-46. The character-generator provides a row output for each character of the line. The 7-bit ASCII is presented on the left of the character-generator on the illustration, and the three line-select lines, appearing at the bottom of the character-generator, specify which one of the 7 rows of the dot-matrix is being output on the right. The five dots corresponding to the row contents are then gated into the shift-register, and are being clocked out in serial form to the video output.

Four kinds of data must be encoded into a composite video signal:

1. the dots representing the character
2. the eventual blinking signal (usually for the cursor)
3. the cursor
4. finally the H and V Sync signals.

A simple analog switch will normally be used to form this composite video signal and the mechanism is illustrated on Fig. 4-47.

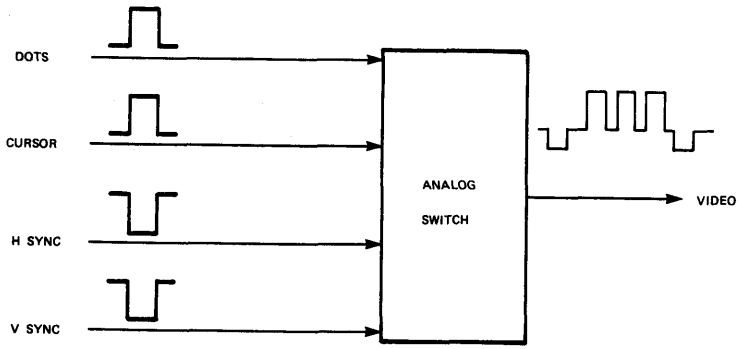


Fig. 4-47 Mixing to Produce Video with Sync

Typical video interface levels are 0 to 2.0 volts, .5 to .75 for the black level, and .15 to 2 volts for the white level. This is illustrated on Fig. 4-48. The sync signal is referred to as the sync tip. Its duration is 4.7 us. It is followed by the black and white dot signals encoded as a voltage swing between .5 and 2 volts. The timing appears on Fig. 4-49. On a standard television, white is 100% level, black is 25 to 30%, and sync is 0%. Typical voltage swing is 2 volts. Standard television line time is 45 us.

Finally, the composite video output can be connected to the television set either directly, at the level of video entry which has been presented, or through an RF modulator, for connection to the television antenna. This is illustrated on Fig. 4-50.

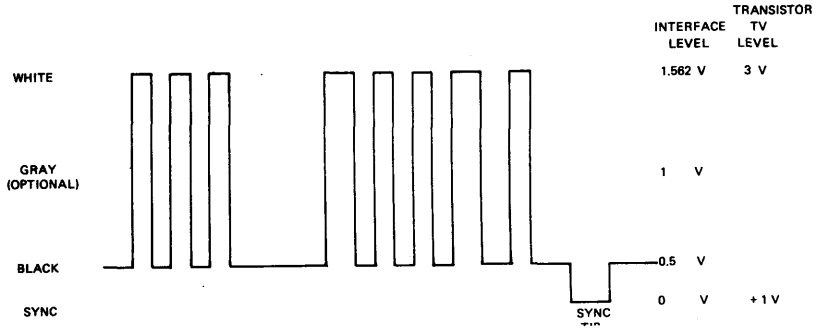


Fig. 4-48 Composite Video and Sync

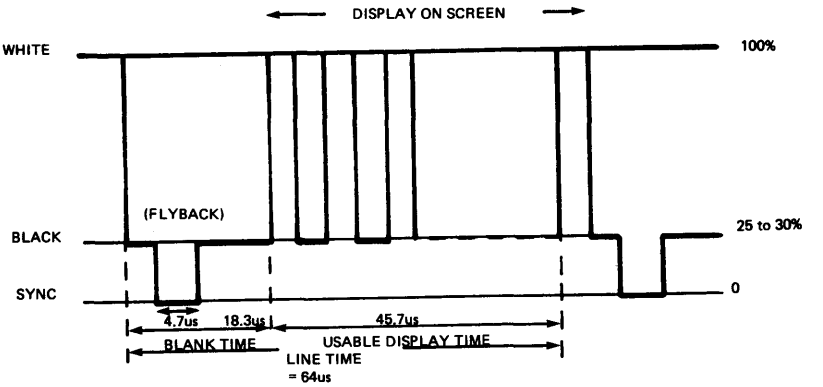


Fig. 4-49 TV Timing

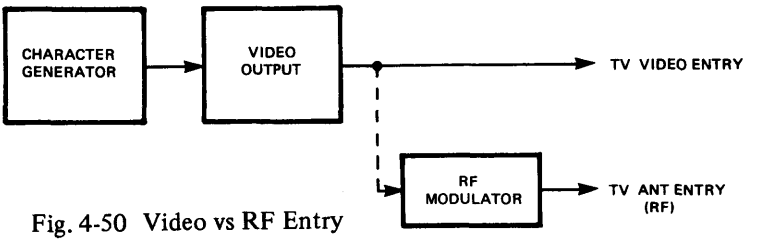
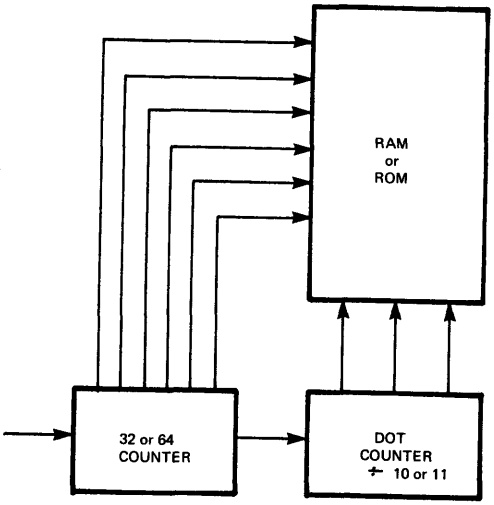


Fig. 4-50 Video vs RF Entry



Character Memory Timing Generation

Refresh Memory

For simplicity in the design, the refresh is usually performed from a dedicated memory. However, a microprocessor system equipped with a DMA can be directly used to refresh a screen. In this case, dual line buffers are used during the DMA transfers between the microprocessor's memory and the television display. This is illustrated on Fig. 4-51. The DMA will first fill line buffer 1. During this time, line buffer 2, which was presumed to be full, will empty itself into the output paths, on the right of the illustration. Typically line buffer 2 will empty itself during time 2T or more, where T is the time necessary for the DMA to fill one of the buffers. Whenever line buffer 2 will have finished emptying itself, line buffer 1, which was long since full, will be switched on, and will start emptying itself through the multiplexer. As soon as line buffer 1 is switched on, the DMA will quickly refill line buffer 2. This dual buffering scheme guarantees continuous system operation. The only timing requirement is that the DMA be capable of filling one of the line buffers in less time than it takes the other to empty itself. Clearly the DMA should do better than this. The DMA should be capable of loading one of the line buffers much faster than the other empties itself. Otherwise, the memory and the DMA would practically be used exclusively for memory refresh, and no program could execute on the microprocessor itself.

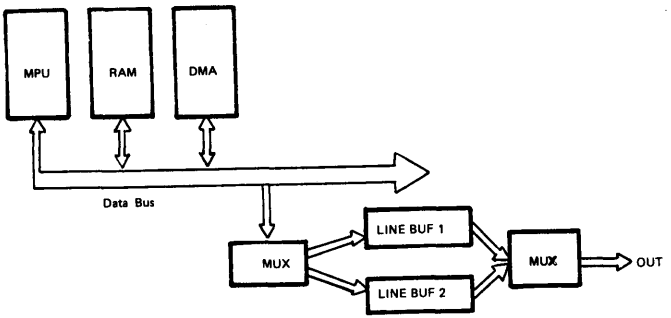


Fig. 4-51MPURequires Line Buffers

One-Chip CRT Controllers

The new one-chip CRT-controllers (CRTC's) simplify the interfacing of a microprocessor system to a CRT. However, despite their name, they do not implement in a single chip all the functions required to interface to a CRT. They are intended for raster-scan CRT, and usually require a RAM page-buffer. This RAM page-buffer may have a size of 2K words or more (requiring then 11 address outputs, at least). A 2K RAM is sufficient for 25 lines of 80 characters.

The CRTC provides the logic for cursor control, sync pulse generation, and dot-row selection in an external character-generator. All present CRT's require an external refresh, a ROM character-generator, and the downstream logic which has been described, including essentially the shift-register and video output. The use of such a typical CRTC is illustrated on Fig. 4-52.

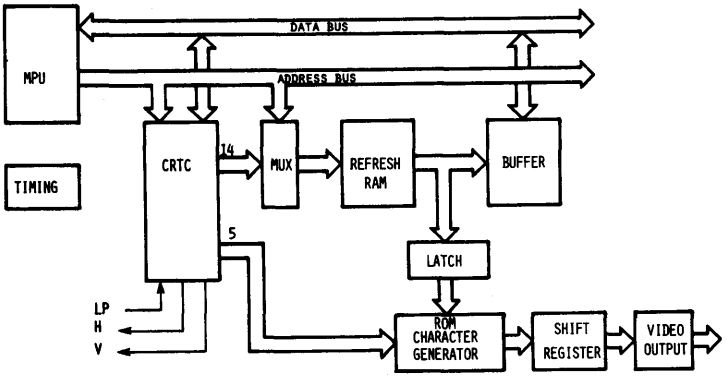


Fig. 4-52 CRT Controller Block Diagram

The Motorola 6845 CRTC

The chip pinout appears on Fig. 4-53. It generates the row-count for the character-generator, the V and H sync, the blanking signal, and a 14-bit refresh address for the RAM buffer. In addition, it provides scrolling and paging. *Scrolling* refers to the vertical shifting of lines across the screen. *Paging* refers to the automatic display of the next screen-full of characters. It is equipped with a cursor register, a light-pen register, and does not need a line buffer.

Programmable features are:

— Dot/rasters per character
— Characters per line
— Lines per sync
— Horizontal/vertical sync position
— Cursor appearance.

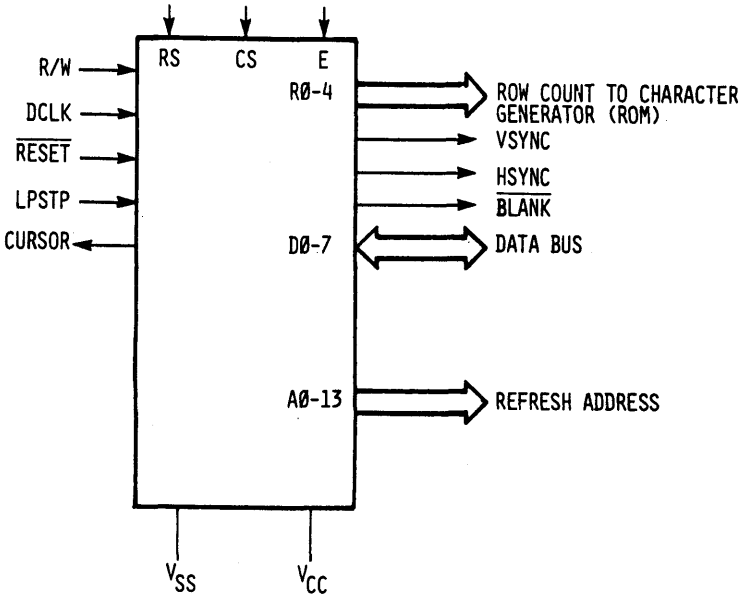


Fig. 4-53 CRT Chip Pinout

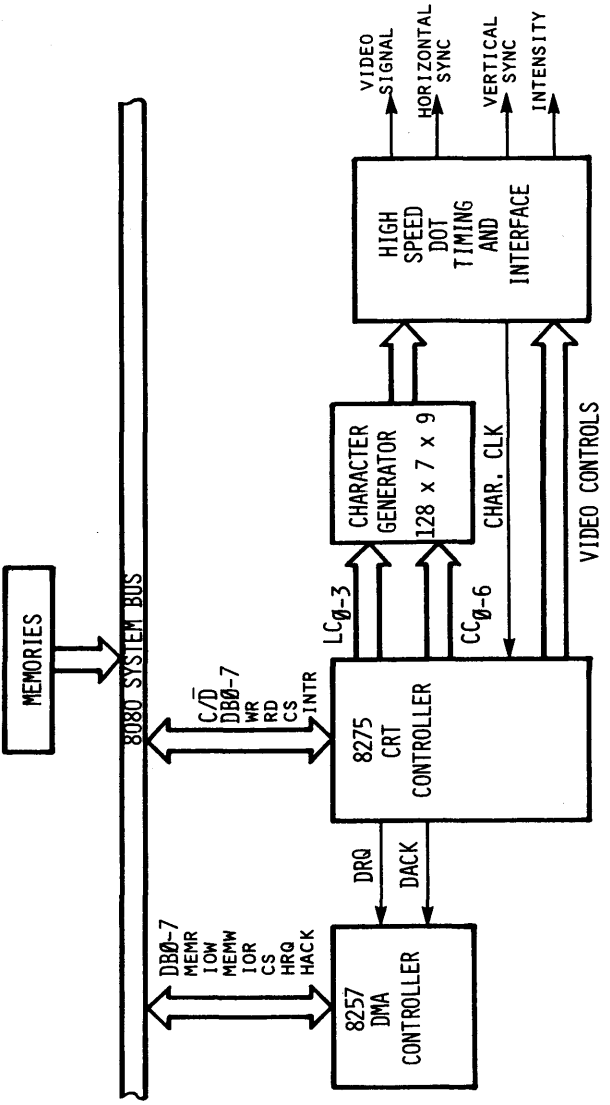


Fig. 4-54 Using DMA to Refresh CRTC Memory

The Intel 8275 CRTC

Similarly, the Intel 8275 CRTC will interface to a 5 × 7 or 7 × 9 character-generator, and generate all the usual video controls. The basic interconnect of the chip in a system appears in Fig. 4-54.

As usual, the CRTC provides 11 address lines to address the buffer. It includes logic for cursor control, (CM0/CM2 inputs on Fig. 4-54) and sync pulse generation (COMP SYNC, VRT SYNC).

It is programmable:

— Display format (FS0-FS2 control inputs)
— Matrix size (5 × 7 or 7 × 9 dot-matrix)
— Scroll-mode (this is controlled by the scroll input)
— Auto-feeding of new line
— Refresh rate (50 Hz/60 Hz – RR input)

Other output signals are:

— DLC0-3 is the dot line counter: it provides the line-address in a character.
— LDV is "loaded video". It is the output dot into the external shift-register.
— Blank is the blanking signal
— Blink is for flashing the cursor or any other symbol on the screen.

As an example, the 8 code-combinations allowed by CM0-CM1-CM2 for cursor motion-control appear on Fig. 4-55.

FLOPPY DISK

A floppy-disk and its controller appear on illustration 4-57. A *floppy-disk* is simply a disk coded with a magnetic material, and divided into *sectors* and *tracks*, on which data is recorded. It provides a very low-cost storage medium with high-speed access and a large capacity. Two types of floppy disk exist today: the regular floppy-disk and the mini-floppy.

A regular floppy-disk such as the SHUGART SA800 provides the following facilities: (It can be either single-density or double-density. We assume single density here.)

— Total capacity per disk: 3.2 Megabits.
— Capacity per track: 41.7 kilobits (unformatted).

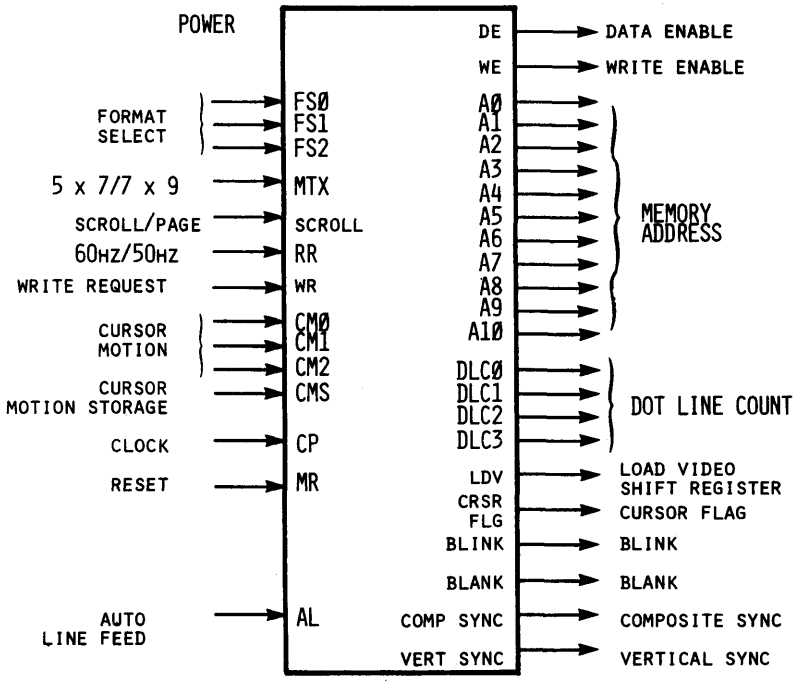


Fig. 4-55 CRTC Pinout

CM2	CM1	CM0	FUNCTION
L	L	L	UP
L	L	H	RETURN
L	H	L	LEFT
L	H	H	HOME
H	L	L	DOWN
H	L	H	NEW LINE
H	H	L	RIGHT
H	H	H	OUTPUT CURSOR ADDRESS (ADDRESS IS VALID WHEN DE OUTPUT IS LOW)

Fig. 4-56 Cursor Functions

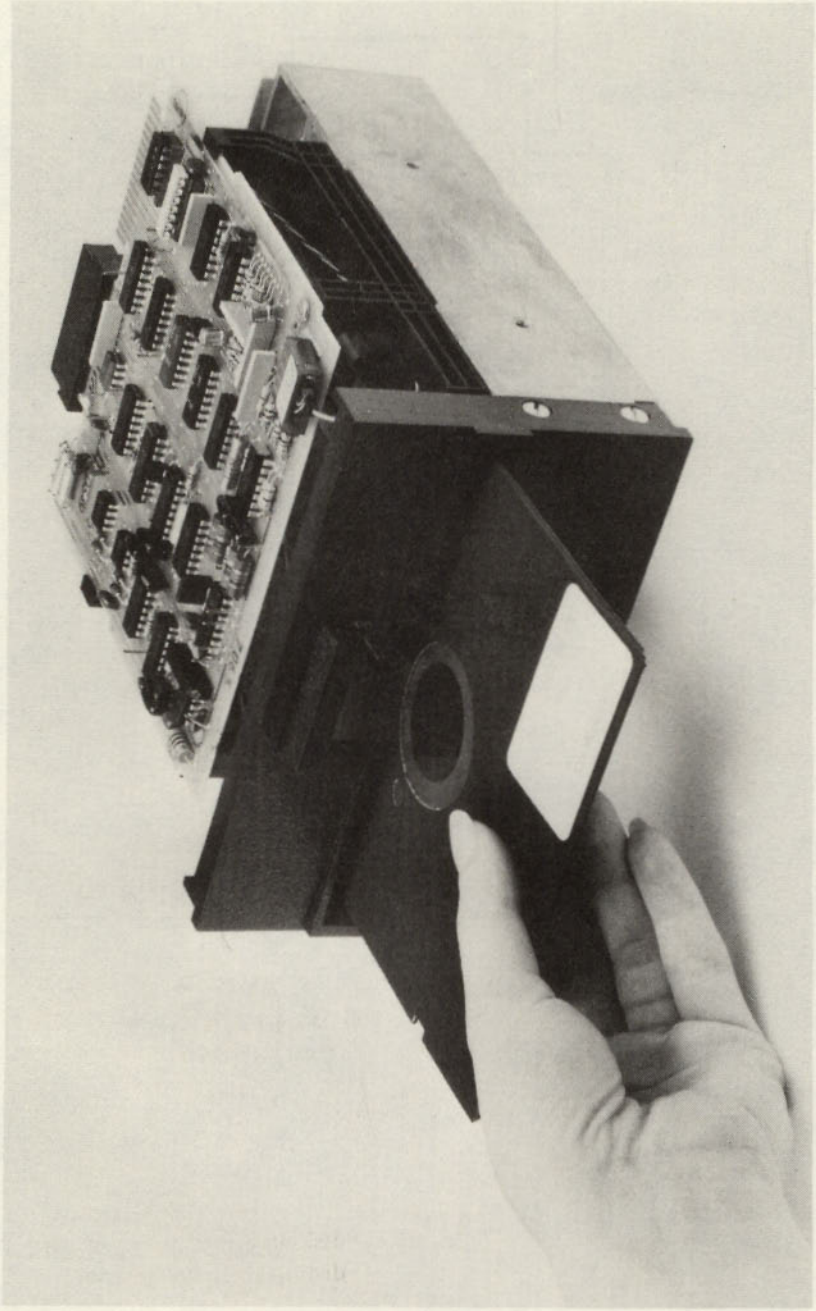


Fig. 4-57 Shugart Mini-Floppy

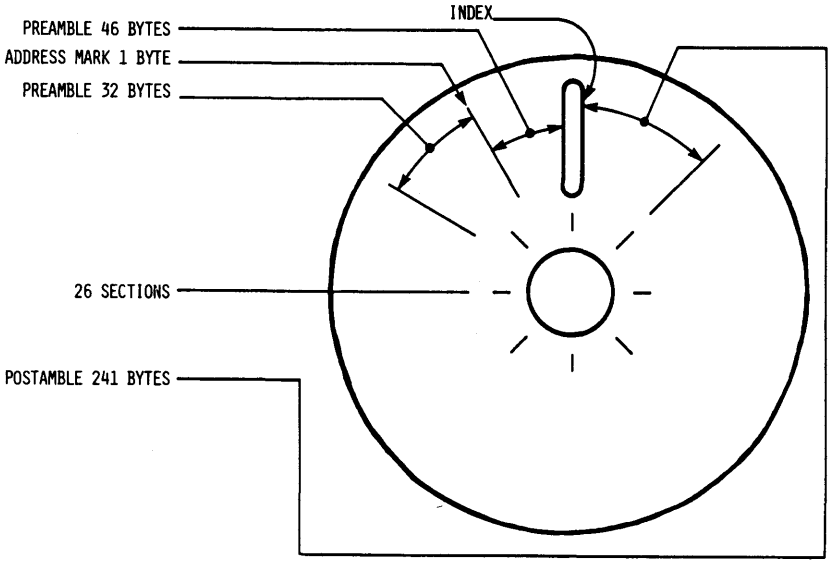


Fig. 4-58 Floppy-Disk Format

In IBM format, capacities become:

— per disk: 2.0 megabits
— per track: 26.6 kilobits.
— Transfer rate is 250 kilobits/sec.

The access times are:

— track to track: 8 ms
— Average access time: 250 ms
— Settling time: 8 ms
— The head load time is: 35 ms.
— The rotational speed of the disk is 360 rpm and the recording density (inside track) is 3200 bpi for single density, and 6400 bpi for double density.
— The track density is 48 tpi and the number of tracks is 77.

For a mini-floppy, the characteristics are:

— capacity:
  — Unformatted: 109.4 kilobytes per disk and 3125 bytes per track.
  — Formatted: Two cases must be distinguished: soft-format and hard-format.

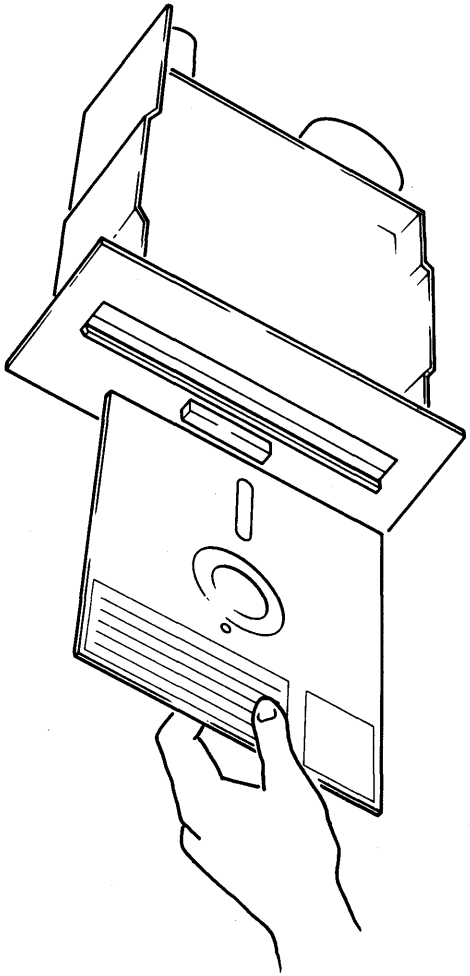


Fig. 4-59 Floppy-Diskette with Drive

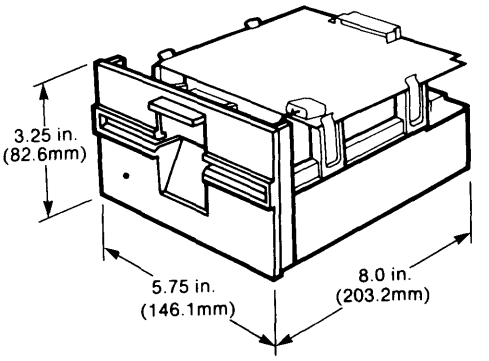


Fig.4-60 Size of Mini-Floppy

In a *hard-format*, actual holes are punched on the disk, to mark the beginning of the new sector. In a *soft-format*, only one hole is punched to indicate the beginning of every track, but the length of sectors on the track is left up to the designer, or the programmer.

	Soft	**Hard**
Per disk:	80.6 Kbytes	72.03 Kbytes
Per track:	2304 bytes	2058 Kbytes
Per sector:	128 bytes	128 bytes
Sectors/track:	18	16

The transfer-rate is 125.0 kilobits per second.
The access-time is:
Track-to-track: 40 ms
Average: 463 ms
Settling time: 10 ms
The head loading time is: 75 ms
Rotational speed is: 300 rpm
Density is 2581 bpi (for the inside track)
The total number of tracks is: 35
Track density is: 48 bpi

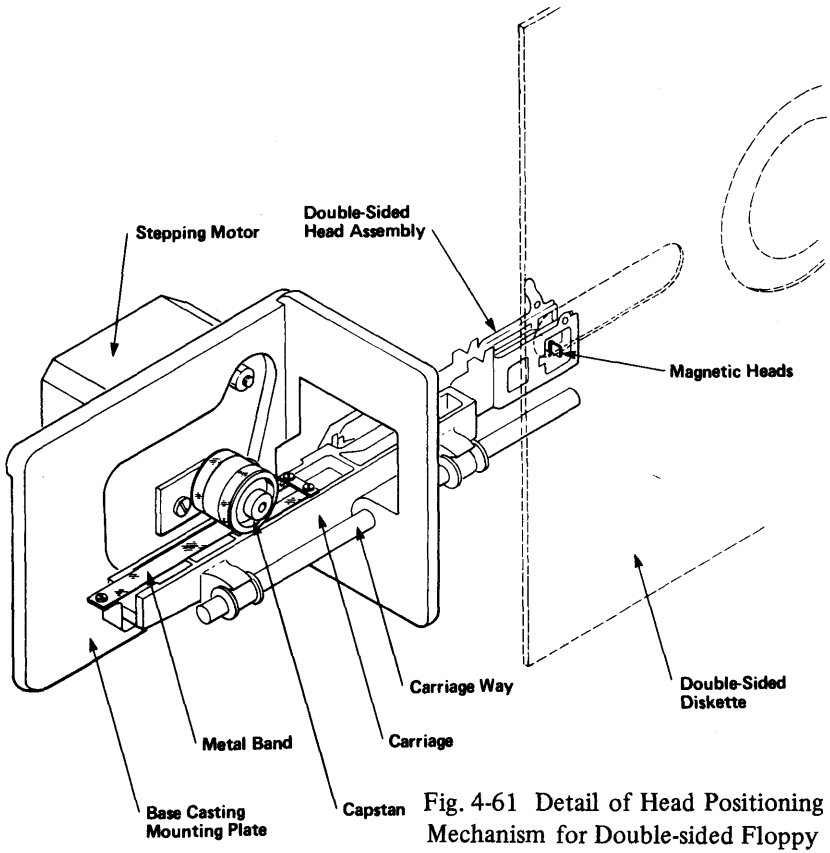


Fig. 4-61 Detail of Head Positioning Mechanism for Double-sided Floppy

These numbers are based on the SHUGART SA400 disk drive. The disk itself is 130.2 millimeters and recording method is FM.

— Reliability data are:
  Life is rated at 10.3^{6} passes per track. MTTR is 30 minutes. MTBF is 8000 POM.
— Errors rating are:
  — Soft: 10^{-8}
  — Hard: 10^{-11}
  — Seek: 10^{-6}

Consumption is: 15 watts is continuous duty, and 7.5 watts in standby power. Required power supply is 12 and 5 volts DC.

A disk can be simply protected against accidental erasure by using a write protect tab on the disk cardboard envelope. This is illustrated on Fig. 4-62.

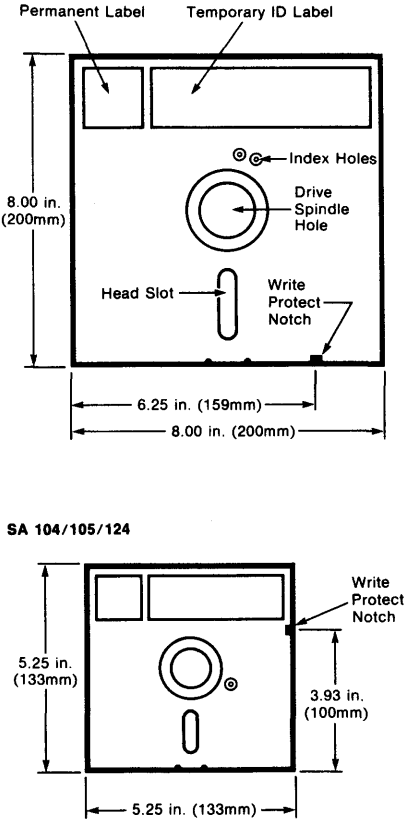


Fig. 4-62 Comparison: "Floppy" vs "Flippy"

The Disk Drive

The disk drive itself includes the following facilities:

1. Read/write control, plus control electronics (2 PC boards)
2. The drive mechanism
3. The read/write head positioning mechanism
4. The read write head

The read-write facilities, mentioned in 1 above, include:

— index and sector detection
— R/W head position actuator drivers
— R/W load actuator drivers
— Write drivers
— Read amplifier, plus transition detectors
— Write-protect detector
— Drive-select circuits
— Drive-motor control circuits

Accessing a Track

The head moves over the disk surface from track to track. It is moved along a radius of the disk by a stepping motor. In order to access a track, the following sequence will occur:

1. the drive-select must be activated. Usually a disk controller may control more than 1 unit, and will enable the drive-select of the mechanism which is selected for access.
2. the direction-select will be set, resulting in a latching of the direction of the movement of the head. The head will move either towards the center of the disk, or towards its periphery.
3. the write-gate goes inactive. During head movement, no writing should occur.
4. the step-line will be pulsed until the desired track is reached. Each pulse will result in a step of the head over to the next track, in the direction which has been latched.

Reading and Writing

Reading is simply accomplished by:

— activate the drive-select
— write-gate inactive.

Writing is accomplished by:

— activate the drive-select
— activate the write-gate
— pulse data in on the write data-line.

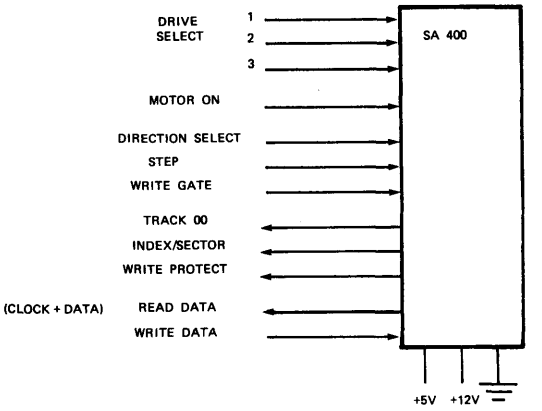


Fig. 4-63 SA 400 Floppy-Disk Drive

Signals of the Disk Drive

The signals required by, or generated by, the SA 400 mini-floppy disk-drive appear on illustration 4-50. Six essential signals are used to communicate with the disk drive:

MOTOR ON

The signal will turn the motor on, or off. When turning the motor on, 1 second should be allowed after activation. Conversely, the disk-drive should be deactivated after 2 seconds (or 10 revolutions), whenever no further commands are issued. This will extend the life of the drive.

DIRECTION SELECT

This input selects the direction in which the read/write head will be moved. The actual motion will be accomplished by pulsing the step line.

STEP

This moves the head by 1 track position towards the center or away from it. The movement occurs on the trailing edge of the pulse.

WRITE GATE

Write is enabled when this line is active. Read is specified when the line is inactive.

TRACK 00

The signal indicates that the head has reached the outside of the disk, i.e., its outermost track or track 0. The head will move no further even if additional step commands are issued.

INDEX/SECTOR

A signal is issued whenever a hole is sensed in the disk. Two types of holes may be used, index-holes, and sector-holes. Every disk will provide an *index-hole* marking the beginning of the first sector on the disk.

A hard-formatted disk, which will be described below, has an addtional number of holes marking the beginning of every sector. When soft-sector is used, one pulse is issued per revolution at the beginning of a track. This is every 200 ms. When using a hard-sectored disk, 11 or 17 pulses are issued per revolution.

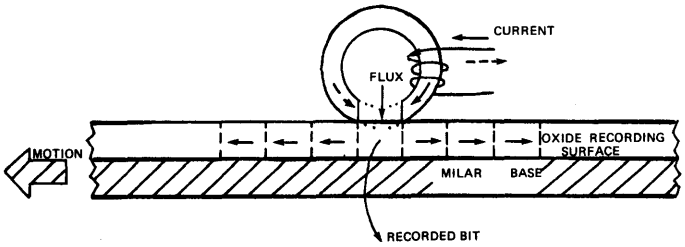


Fig. 4-64 Recording A Bit On A Disk

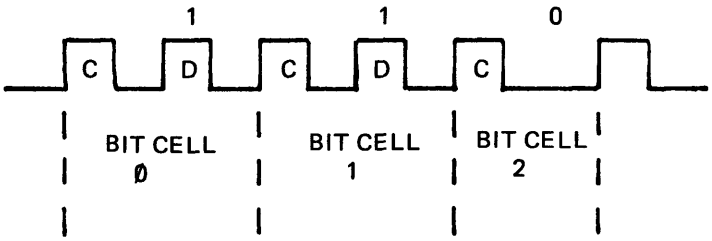


Fig. 4-65 Representing Check and Data

Disk Formatting

Both clock and data information are encoded into the same signal. Clock pulses are issued for every bit. A "0" data is indicated by no further pulse during the bit cell time. This is illustrated on Fig. 4-65. A "1" is indicated by a data pulse occurring in the middle of the bit cell interval.

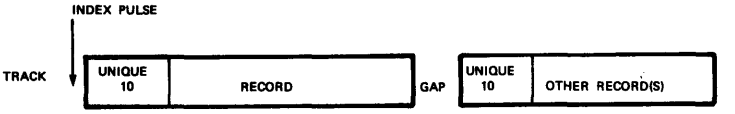


Fig. 4-66 Record Identifier

Soft-sectoring refers to the fact that the division of the disk or track into sectors is performed by *software*. This is opposed to hard-sectoring, where the beginning of each sector is physically delineated by a hole punched in the disk. In soft-sectoring, each track is started by a physical *index-pulse*, corresponding to the detection of the index-hole on the disk. Every record is preceded by a unique *identifier*. See Fig. 4-46. Successive records are separated by *gaps*. Gaps are necessary in order to upgrade information without erasing the following or the preceding record. Because of minor speed variations in the disk drive motor, whenever a record will have all or part of its contents rewritten, the end of the record might extend beyond the previous record end.

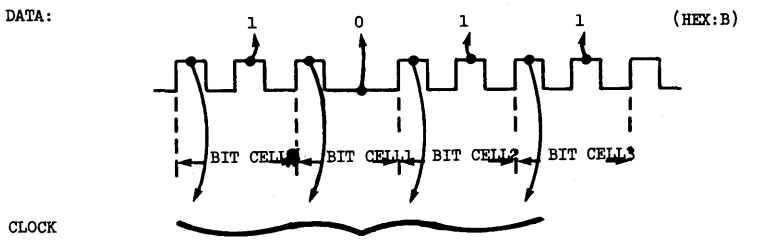


Fig. 4-67 Identifier Format

	DATA	CLOCK
INDEX ADDRESS MARK	FC	D7
ID ADDRESS MARK	FE	C7
DATA ADDRESS MARK	FB	C7
DELETED DATA AM	F8	C7

ADDRESS MARKS

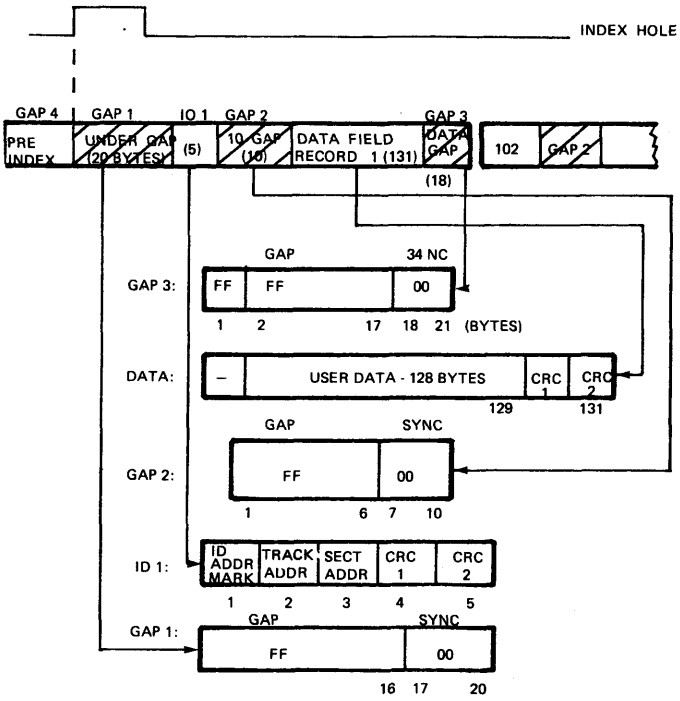


Fig. 4-68 IBM Floppy-Disk Format

For this reason, a blank gap must be provided between the end of one record, and the beginning of the next one. In fact, a gap must be provided between any two zones which might be updated separately. Most often, the IBM disk-track format is used, sometimes with minor variations. This format is illustrated on Fig. 4-68. Four kinds of gaps are used:

Gap 4 is used only once on the track. It is the free-index gap. It appears at the end of the track just before the index-hole position.

Gap 1 is called the index-gap, and is used at the beginning of every track. It contains 20 bytes: the first 16 bytes contain the hexadecimal pattern "FF" followed by 4 bytes containing "00". These four bytes of 0's are the classical way to provide the synchronization for the data-separator. The length of gap 1 may never vary in length. The index-gap is followed by the identification of the first record.

ID 1 is the identification-field of the first record. It uses 5 bytes: the ID address-mark, the track-address, the sector-address, and two CRC checksum bytes to verify the integrity of the field. The track-address and the sector-address provide a verification that the right track and sector have indeed been accessed.

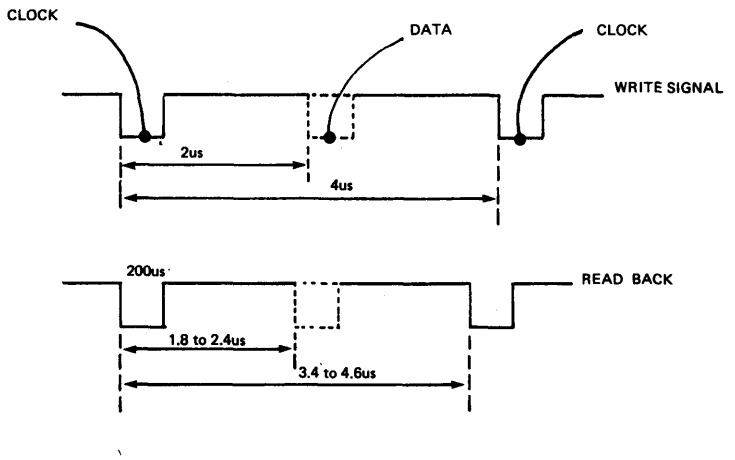


Fig. 4-69 Timing

Gap 2 is called the ID-gap and separates each successive identification field from its data field. It uses 10 bytes. The first 6 bytes contain the hexadecimal pattern "F". It is followed by the four usual synchronization bytes containing "00". The length of gap 2 may vary in length after file updating.

The first *record*, or data-field follows. It uses 131 bytes (see Fig. 4-47). The first bytes contains data or deleted address-mark. It is followed by the actual 128 bytes of user data. It is terminated by the two usual CRC check-sum bytes.

Finally, *Gap 3* terminates the first record. It is called the data-gap and uses 18 bytes. The first 17 bytes are set to the pattern "FF", and the four last bytes contain "00", for the sync. Every successive record on the disk, or sector, will start with ID, gap 2, and so on.

Hard-sectoring

When using hard-sectoring, a special diskette and drive are used. A hole is punched at the beginning of every sector on the disk. Each sector is then started by a physical sector pulse. In the case of the mini-floppy disk, two configurations are used: 16 sectors of 128 bytes or 10 sectors of 256 bytes per track. The track is started by the index pulse. This is illustrated on Fig. 4-70.

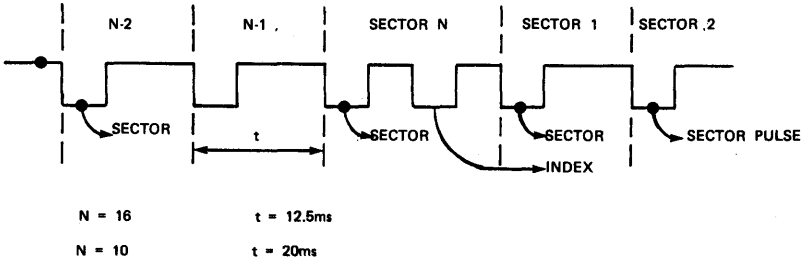


Fig. 4-70 Hard-Sectored Disk Timing

Error Detection and Correction

Three types of errors are distinguished:

Write Error

This corresponds to the case where the data being written on the disk is not written correctly. The way to verify whether data has been correctly written is to use a "write-check" procedure, where the data is read again during the next revolution of the disk. Normally, the user will simply write again data which has not been correctly written on the disk, and attempt to do so repeatedly (up to 10 times). If this effort fails continuously, the sector or the track must be considered as damaged and not usable.

Read Errror

Two types of read errors must be distinguished:

1. Soft: this corresponds to the case where the error has been transient and is corrected by simple re-reading (up to 10 times) or by moving the head back and forth once.

Typically the head is moved one more step in its previous direction, then moved back. Usually this corrects most reading errors. If this procedure fails, we have a hard error:

2. Hard: Whenever usual correction procedures fail to read data from the disk, it must be deemed unrecoverable. This is a fatal error. Data is lost.

SEEK ERROR

This corresponds to the case where the head does not reach the correct track. This can be verified by reading the ID field at the beginning of the track. It contains the track address. Whenever an error is detected, the track-counter of the disk drive must be recalibrated. The head is moved back to track 00 and a new Seek order is issued.

DETECTING ERRORS

Universally, the error-detection for any data written on a disk is accomplished by using a *check-sum* method. Cyclic-redundancy-check (CRC) is used for this purpose. Each field is terminated with two CRC bytes. The data bits are divided by a generator polynominal G(X) such as $G(X) = X^{16} + X^{12} + X^3 + 1$. The remainder of this division is called the CRC. It is written in the two bytes that follow the data. When reading back data from the diskette, everything is read, including data in the CRC bytes.

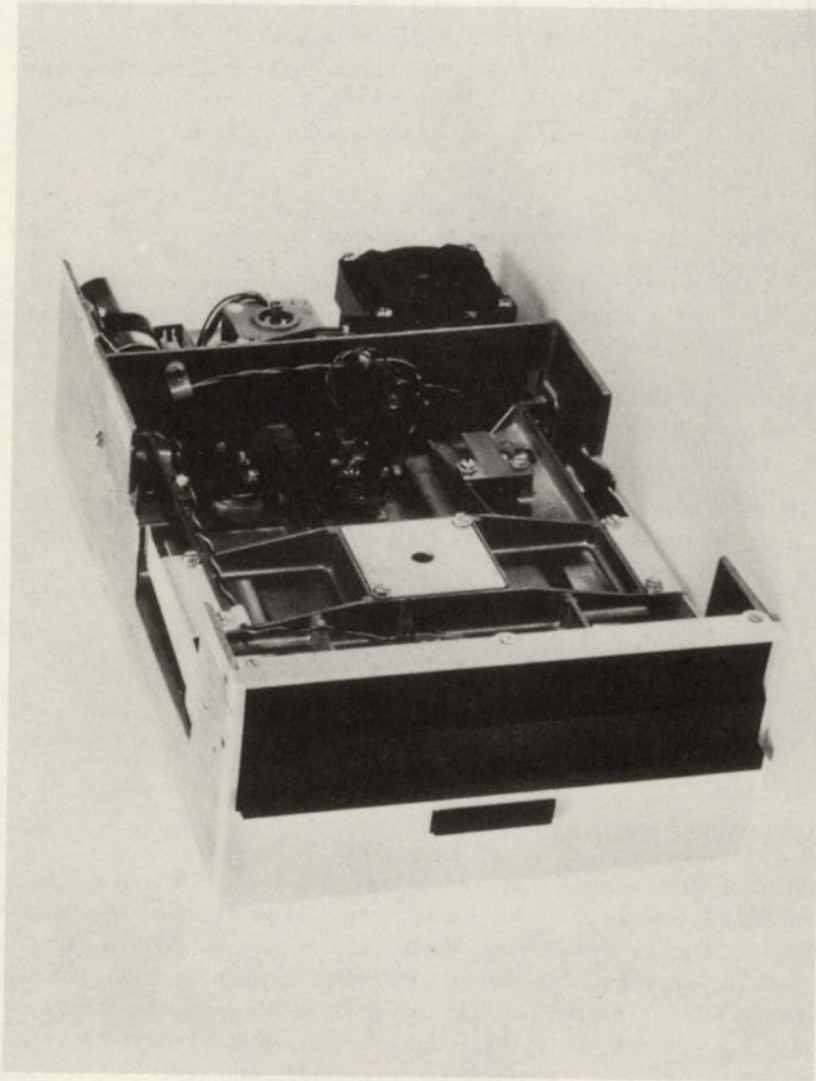


Fig. 4-71 Picture of Double-Sided Floppy

If the remainder of the division by the G(X) polynomial is not 0, an error has been detected.

Single-chip CRC's exist such as the Fairchild 9401, the Motorola 8501, and others, that will detect such failures in a single chip. One-chip floppy-disk-controllers (FDC) also accomplish the CRC generation and checking, within the single chip.

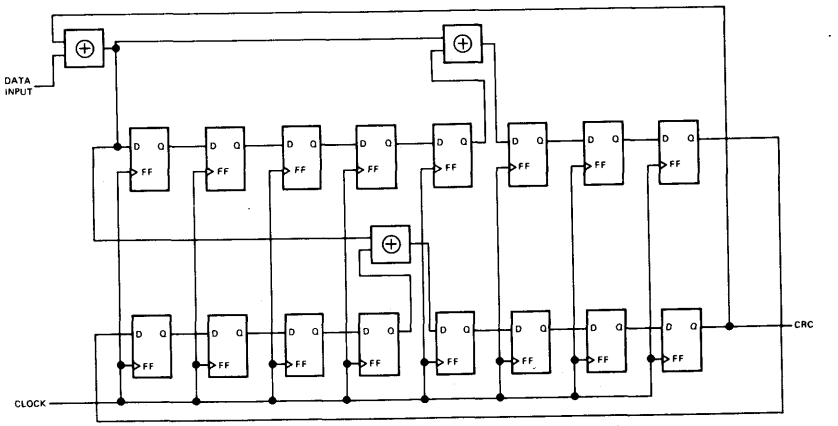


Fig. 4-72 CRC Check Hardware Detail

Cyclic Redundancy Check

CRC is the favorite method for varifying the integrity of memory areas with a minimal waste of bits. Parity will detect a single-bit error within a word. Whenever parity is not available, or would be too costly to provide, CRC is used to detect errors in a block of words. In particular CRC is almost always used in the case of floppy-disks, and tape-cassettes. In addition, it is often used to verify the integrity of a ROM. The principle of a CRC technique is the following: the eight bits of the word are treated as coefficients of a polynomial of degree 7.

The bit pattern $B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$ is interpreted as $B_7\ X^7 + B_6\ X^6 + B_5\ X^5 + B_4\ X^4 + B_3\ X^3 + B_2\ X^2 + B_1\ X^1 + B_0\ X^0$.

X is called here a dummy variable.

For example, the binary word: "10000011" will represent:

$B(X) = 1\ X^7 + 0\ X^6 + 0\ X^5 + 0\ X^4 + 0X^3 + 0\ X^2 + 1\ X^1 + 1\ X^0$

$= X^7 + X^2 + 1.$

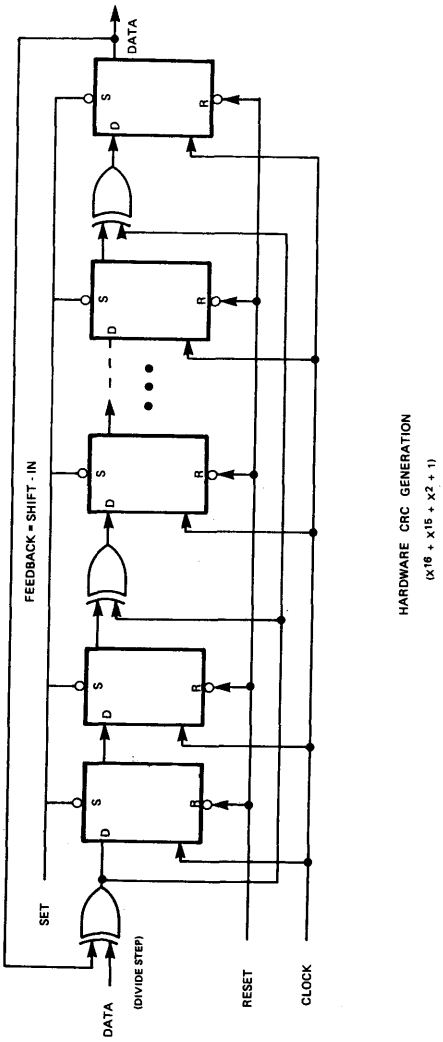


Fig. 4-73 CRC Generation Hardware Detail

A generator polynomial G(X) will be used. The polynomial B(X) corresponding to the binary word is divided by this generator G(X). The result is the quotient Q(X) and a reminder R(X).

$B(X) = G(X)\,Q(R) + R(X)$

The value of CRC-redundancy-checking is to append to a bit string an extra byte (or bytes), equal to R(X), so that the total string will be exactly divisible by the generator polynomial. The above equation can be rewritten: $B(X) - R(X) = Q(X) \cdot G(X)$. The string formed by B and the remainder R is exactly divisble by G(X). The extra bits appended to the string B are called the CRC bits (or bytes). When receiving for the first time a string B, the CRC generator will compute the remainder R which will be appended to the string. When the string will be retrieved another time, the complete sequence of bits, including the CRC bits will be read. They should then be exactly divisible by the generator polynomial G(X). If they are not, an error has been detected. If they are divisible, no error has occurred, or else a non-detectable error has occurred.

As usual, the CRC algorithm can be implemented either in hardware, or in software. One-chip CRC generators are available. An example of a programmed CRC, using the Signetics 2650 appears on Fig. 4-52. The program which implements an emulation of the hardware appearing in Fig. 4-53. The hardware division is accomplished there by the shift-register with feedback. The CRC generator corresponding to the illustration is $G(X) = X^{16} + X^{15} + X^2 + 1$. The exclusive-OR feedback accomplishes the division during the successive shifts through the flip-flops of the register.

Summary of Disk Operation

The complete principles of floppy-disk operation have now been presented. The signals necessary to drive the disk, its operation, the formatting of data, as well as the error-checking mechanisms that must be implemented. We will now describe the implementation of a disk-drive controller to be interfaced to a microprocessor system.

Example: The SHUGART SA 4400 Mini-floppy Controller

This controller-board is implemented with the SMS/Signetics 300 bipolar controller chip. It is designed to control 1, 2, or 3 SA 400 mini-floppies. It will be briefly described here, in order to show the capabilities of a full mini-floppy controller. Then other compact designs will be presented, using the new FDC chips.

This controller is compatible with the IBM 3740 format, but uses a modified gap structure (the pre-index gap, gap 4, is shorter). It provides a 128-byte buffer for the data. Eight control functionssare supplied:

```
* CYCLIC REDUNDANCY CHECK SUBROUTINE (SIGNETICS 2650)
*
* THIS ROUTINE GENERATES A 16-BIT CHECK CHARACTER FOR
*   THE DATA CHARACTER IN RØ; VARIOUS POLYNOMIALS
*   CAN BE ACCOMODATED BY CHANGING THE CONSTANTS
*   SPECIFIED AT PROGRAM LOCATIONS CKØ AND CK1 AS PRR
*   THE TABLE BELOW
*
* DEFINITION OF SYMBOLS
*
RØ      EQU     Ø       PROCESSOR REGISTERS
R1      EQU     1
R2      EQU     2
WC      EQU     H'Ø8'   PSL:  1=WITH,Ø=WITHOUT CARRY
C       EQU     H'Ø1'         CARRY/BORROW
UN      EQU     3       BRANCH CONDITION  UNCONDITIONAL
EQ      EQU     Ø                         EQUAL
*
* TABLE OF POLYNOMIALS
*
CRCFØ   EQU     H'4Ø'   CRC16 FORWARD
CRCF1   EQU     H'Ø2'
CRCRØ   EQU     H'2Ø'   CRC16 REVERSE
CRCR1   EQU     H'Ø1'
CCIFØ   EQU     H'Ø8'   CCITT FORWARD
CCIF1   EQU     H'1Ø'
CCIRØ   EQU     H'Ø4'   CCITT REVERSE
CCIR1   EQU     H'Ø8'
*
* BEGINNING OF SUBROUTINE
*
        ORG Ø
*                       INITIALIZATION
CRCGEN PPSL     WC      OPERATIONS WITH CARRY
        LODI,R2 8       INITIALIZE BIT COUNTER
        LODA,R1 CRC+1   GET OLD REMAINDER LSB
        EORA,RØ CRC     EX-OR OLD REMAINDER MSB WITH DATA
*
TEST    CPSL    C       CLEAR CARRY
        TMI,RØ  H'8Ø'   TEST MS-BIT OF RØ
        BCFR,EQ SHIFT   BRANCH IF NOT A '1'
        PPSL    C       PRESET CARRY
CKØ     EORI,RØ CRCFØ   APPLY 'FEEDBACK'
CK1     EORI,R1 CRCF1
*
SHIFT   RRL,R1          SHIFT THE DOUBLE CHARACTER
        RRL,RØ
        BDRR,R2 TEST    CHECK IF DONE
        STRA,RØ CRC     SAVE THE NEW REMAINDER
        STRA,R1 CRC+1
        RETC,UN
*
* RAM AREA
*
        ORG     H'5ØØ'
CRC     RES     2       REMAINDER MSB IN CRC
        END     CRCGEN
```

Fig. 4-74 2650 Check Program

— INIT: it resets the controller in the disk
— SEEK: steps ahead to the specified track
— READ: reads a sector (128 bytes)
— READ ID: reads the nest sector-identification
— WRITE: writes a sector of data (128 bytes) with data AM

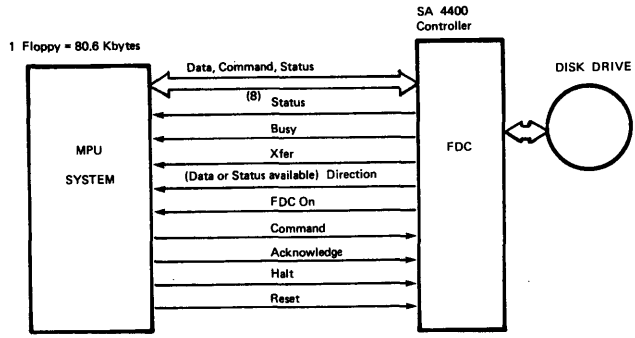


Fig. 4-75 Interface Signals

The previous three commands will read or write data between the host-processor and the disk-buffer, or between the buffer and the disk.

— WRITE — DDL: accomplishes the same as the WRITE command but with deleted data AM (address mark)
— FORMAT: writes address-marks, gaps, data on the entire track in 3740 format
— STATUS: gets status for the drive

The signals used by the 4400 interface to communicate with the host microprocessor system appear on illustration 4-75. The basic sequence of events implemented by the controller is simply:

1. Seek track.
2. Find sector.
3. Shift and transfer the desired number of sectors.
4. Check the CRC.

Few commands are necessary for the controller's operation and most controllers provide six to ten commands only.

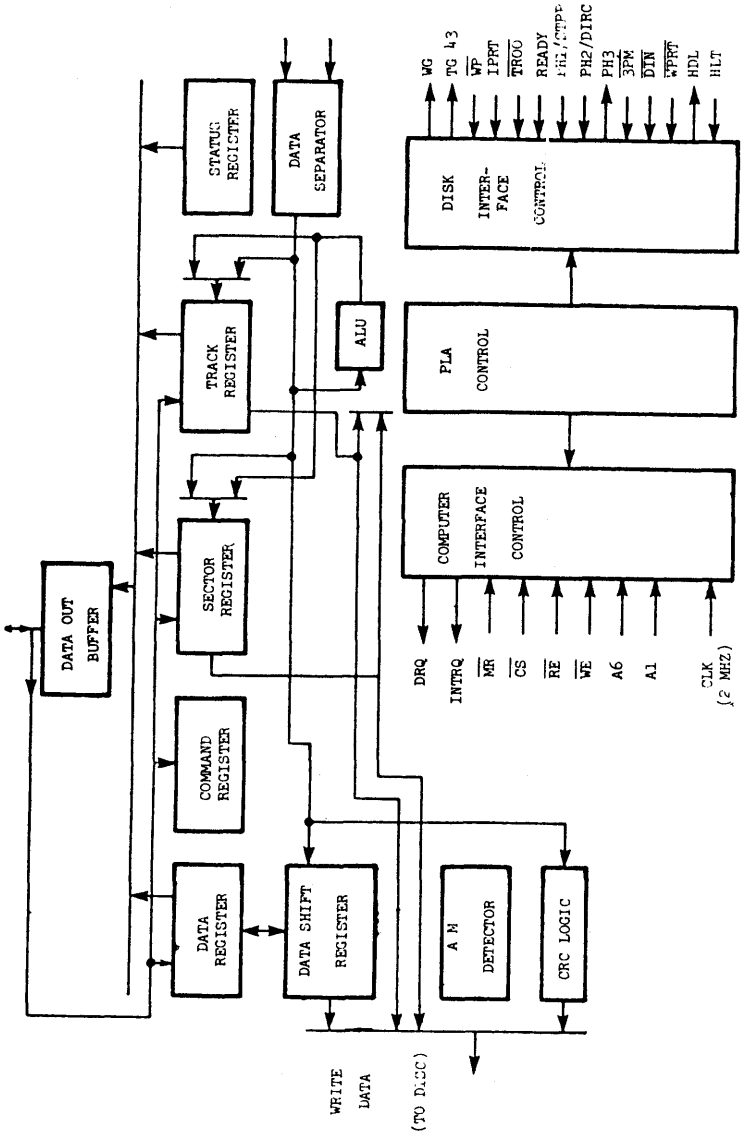


Fig. 4-76. FDC Chip: Western Digital

Western Digital FD1771D FDC

This one-chip floppy-disk controller — formatter will interface to most drive manufacturers and is naturally IBM 3740 compatible. It provides:

- — automatic track-seek with verification. This feature must be provided on all FDC's.
- — soft-sector format compatibility. This feature should be standard on an FDC.
- — read or write with:
  - — single or multiple records
  - — automatic sector search
  - — entire track read or write

Again, these features should be standard in an FDC.

- — programmable controls:
  - — track to track stepping time
  - — head-settling time
  - — head engage time
  - — three-phase or step-plus-direction motor-control
  - — DMA or program transfers

The alert reader will notice that all of the above features are essentially standard for all FDC's. The differences are usually the level of the number of disk-drives that one chip will control simultaneously.

The internal architecture of the FD1771B appears on illustration 4-76. It will be described in detail now. It contains five essential functional circuits, six registers, and two interfaces: a processor-interface and a floppy-disk interface. Each will now be examined.

The Four Functional Circuits

The four essential circuits, which appear on the illustration, are:

- — the CRC logic which generates the check-character.
- — the ALU (Arithmetic-Logical-Unit), which was used for the obvious arithmetic functions, in particular, to compare characters for incrementing or decrementing contents.
- — the disk-interface control.
- — the computer-interface control.

Both interfaces will be described below.

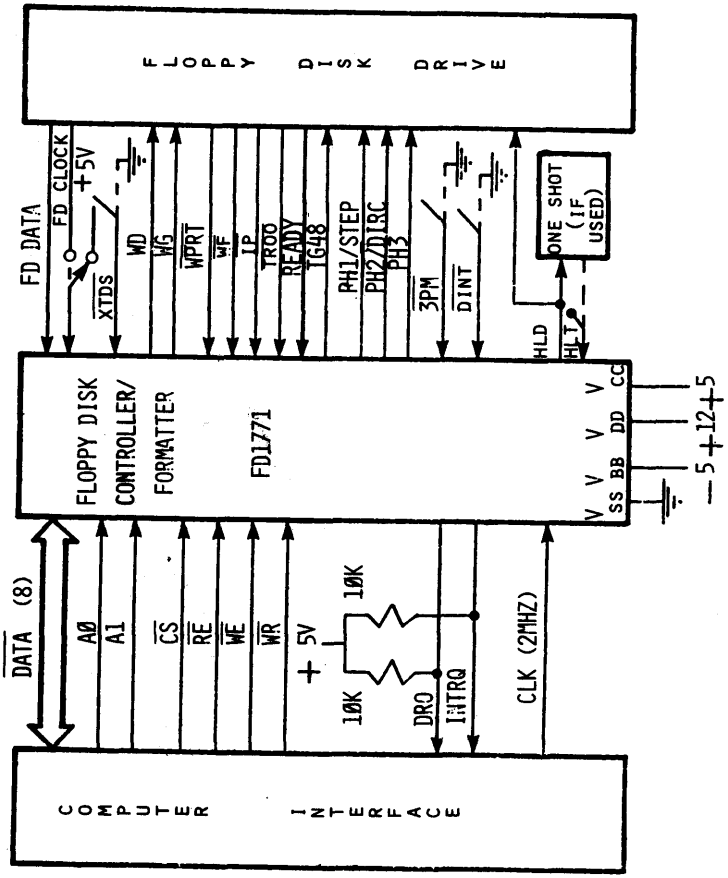


Fig. 4-77 Floppy-Disk Interface Using FD1771.

The Six Internal Registers

From left to right in illustration 4-76, one can distinguish:

1. the *data-shift register:* assembles 8 bits from the floppy-disk data, or serializes 8 bits received from the microprocessor data-bus into the floppy-disk data-line.
2. the *data-register* is a simple holding register for a byte during read and write operations. Communicates with the data-out buffer, and may receive data directly from the microprocessor data bus.
3. The *command-register* is used to hold the 8 bit command being executed. This register is loaded by the programmer and specifies the mode of operation of the disk.
4. The *sector-register* holds the address of the desired sector position.
5. The *track-register* holds the track number of the current head position. It is incremented towards the inside (up to track 76 on the regular-size disk), and decremented otherwise.
6. The *status-register* simply holds the status information of the controller.

Processor Interface

The processor-interface and the floppy-disk interface are illustrated on 4-77. The FDC communicates with the processor via 8 bi-directional data lines labelled DAL (Data-Access-Lines). An input is specified when $\overline{CS}$ and $\overline{WE}$ (write-enable) are active. An act is specified when $\overline{CS}$ and $\overline{RE}$ (read-enable) are active. The internal destination is specified by A1-A0 according to the table below:

The data-request-output (DRO) is used for the DMA. The interrupt-request (INTRT) is activated by various conditions.

(COMMAND WORD)			RATE
BIT 1	BIT Ø	PERIOD (MS)	(STEPS/S)
Ø	Ø	6	166
Ø	1	6	166
1	Ø	8	125
1	1	10	100

Fig. 4-78 Command Word Bits

A 1	A Ø	$\overline{RE}$	$\overline{WE}$
Ø	Ø	STATUS REG.	COMMAND REG.
Ø	1	TRACK REG.	TRACK REG.
1	Ø	SECTOR REG.	SECTOR REG.
1	1	DATA REG.	DATA REG.

Fig. 4-79 Register Addressing

Floppy Disk Interface

The signals appear on the right of illustration 4-77. They provide head-positioning controls, write controls, and data-transfers. The clock is a 2MHz square-wave clock, internally divided by 4, yielding 500 KHz. It provides three programmable stepping-rates, controlled by bit 0 and bit 1 of the command word according to the table below:

The head-settling time is additional and involves 10 milliseconds.

Disk Operation

A *read-operation* on the disk is performed in five steps:

1. Load the track-register
2. Give the Seek-command
3. Wait for verification
4. Transfer data to the microprocessor under interrupt control.
5. Check for interrupt after the correct number of transfers.

Conversely, a *write-operation* is performed in seven steps:

1. Load the track-register
2. Give the seek-command
3. Wait for verification
4. Give the write-command
5. Load the first data after the data-request is received.
6. Load the remaining data
7. Check $\overline{BUSY}$ and CRC-error flag

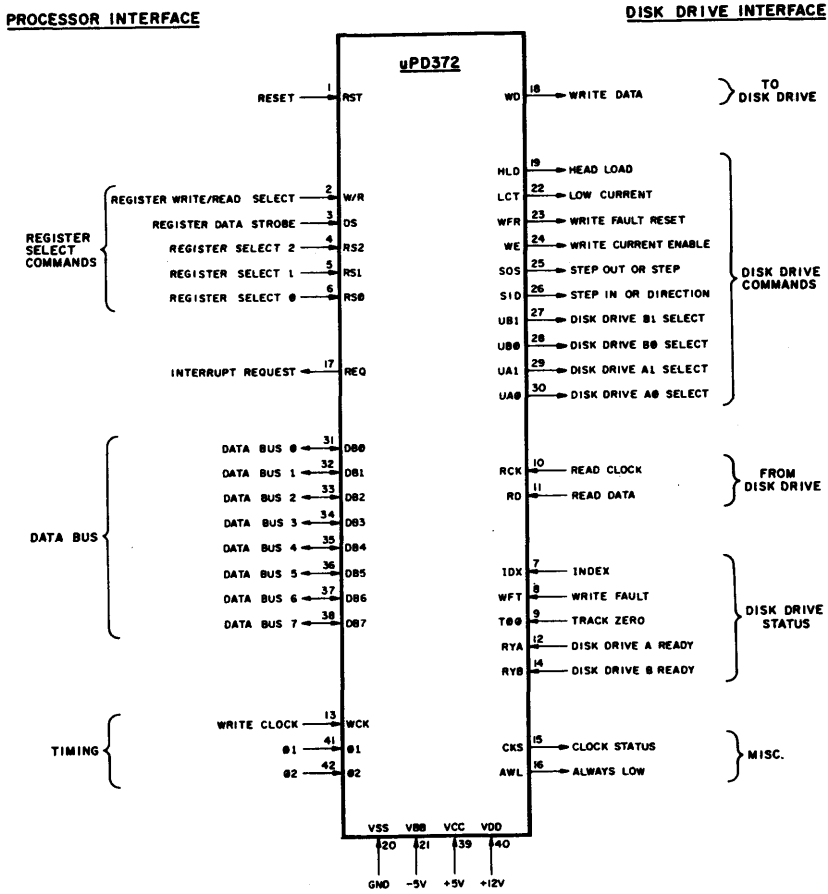


Fig. 4-80 NEC UPD372 FDC

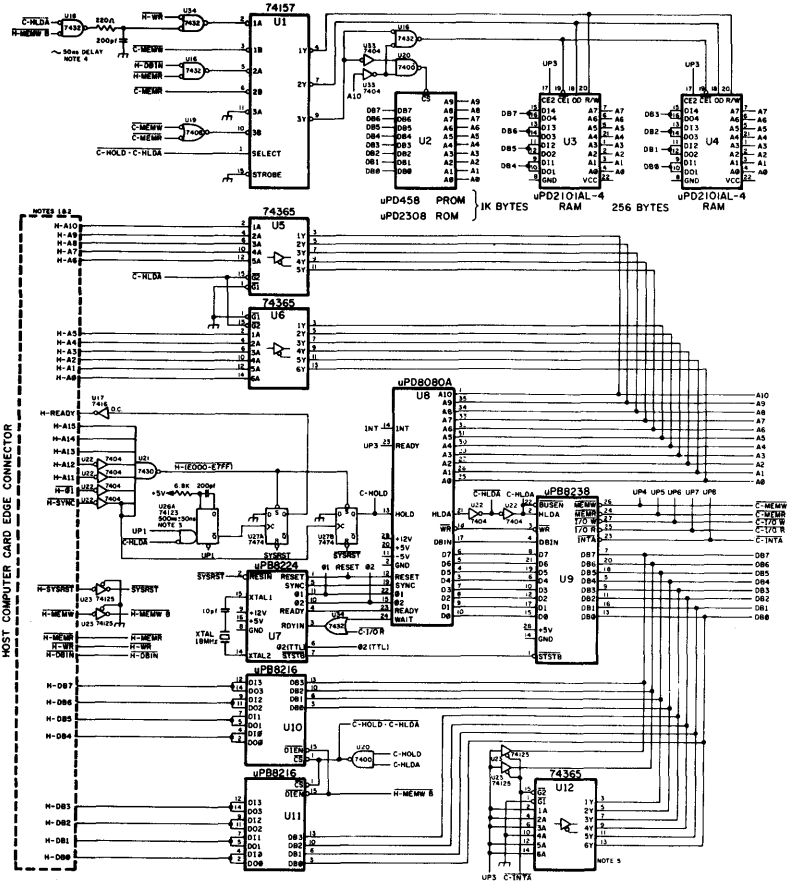
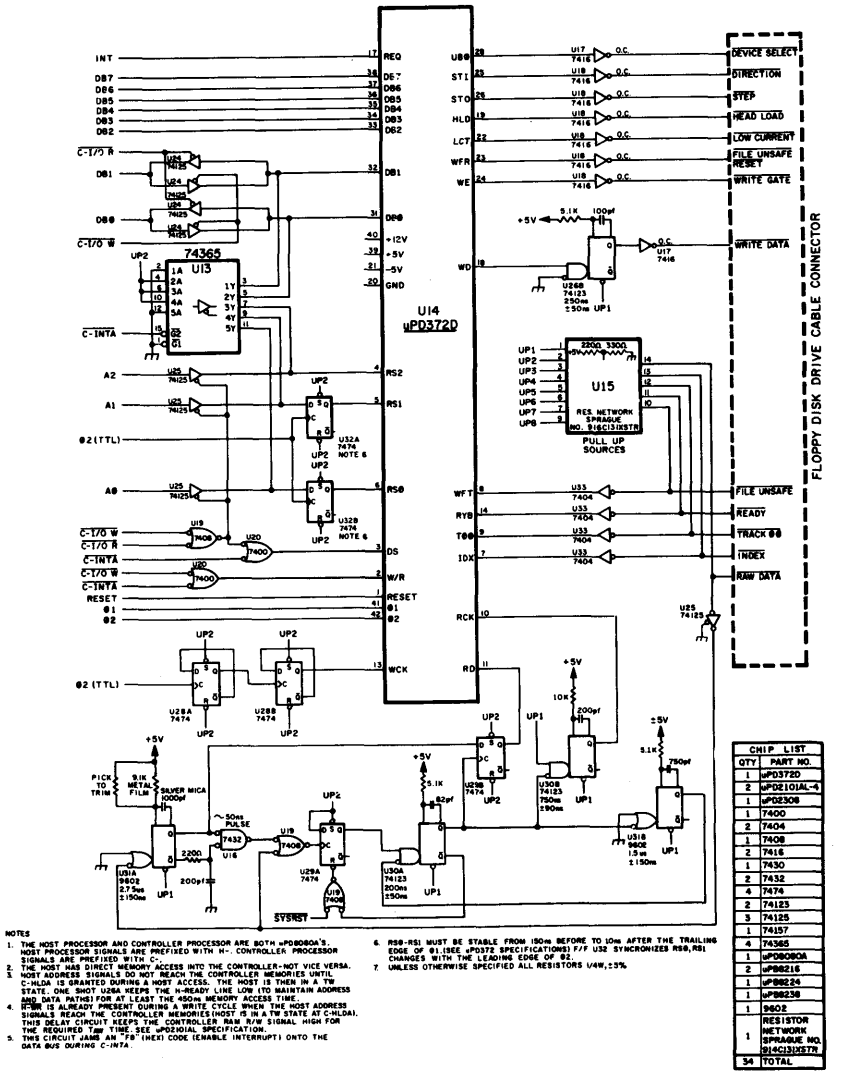


Fig. 4-81 NEC 8080 Disk Controller



NOTES

1. THE HOST PROCESSOR AND CONTROLLER PROCESSOR ARE BOTH μPD8080A'S. HOST PROCESSOR SIGNALS ARE PREFIXED WITH H-. CONTROLLER PROCESSOR SIGNALS ARE PREFIXED WITH C-.
2. THE HOST HAS DIRECT MEMORY ACCESS INTO THE CONTROLLER-NOT VICE VERSA.
3. HOST ADDRESS SIGNALS DO NOT REACH THE CONTROLLER MEMORIES UNTIL C-HLDA IS GRANTED DURING A HOST ACCESS. THE HOST IS THEN IN A 'FB' STATE. ONE SHOT U28A KEEPS THE H-READY LINE LOW (TO MAINTAIN ADDRESS AND DATA PATHS) FOR AT LEAST THE 450ns MEMORY ACCESS TIME.
4. IF RAW IS ALREADY PRESENT DURING A WRITE CYCLE WHEN THE HOST ADDRESS SIGNALS REACH THE CONTROLLER MEMORIES (HOST IS IN A FW STATE AT C-HLDA), THIS DELAY CIRCUIT KEEPS THE CONTROLLER RAW R/W SIGNAL HIGH FOR THE REQUIRED TIME. SEE μPD201A SPECIFICATION.
5. THIS CIRCUIT JAMS AN 'FB' (HEX) CODE (ENABLE INTERRUPT) ONTO THE DATA BUS DURING C-INTA.
6. RS0-RS1 MUST BE STABLE FROM 150ns BEFORE TO 10ns AFTER THE TRAILING EDGE OF Ø1 (SEE μPD372 SPECIFICATIONS) F/F U32 SYNCRONIZES RS0,RS1 CHANGES WITH THE LEADING EDGE OF Ø2.
7. UNLESS OTHERWISE SPECIFIED ALL RESISTORS 1/4W, ±5%

Fig. 4-82 NEC 8080 Disk Controller

Summary

The FD1771D illustrates how it is possible to integrate most of the functions required for the control of a regular floppy-disk into a single chip. It provides essentially all the facilities needed to control and format the disk.

OTHER FDC's

The NEC FDC is called the UPD372D. It is compatible with the IBM 3740 as well as the SHUGART mini-floppy. It provides the usual facilities, such as CRC-generation, programmable step-pulse, track-stepping rate, sector-size, data-transfer rate. In addition, it controls up to 4 disk drives, but with read/write limited to one drive, with simultaneous track-seek on the others.

Other disk drives are:

CAL COMP 140, CDC BR 803, GSI 050, and 110, INNOVEX 210, ORBIS 74, PERSCI 75, PERTEC FD400, POTTER DD4740, SYCOR 145.

The UPD 372D chip appears on Fig. 4-47. A complete interface using the 372D appears on Figs. 4-62 and 4-63.

The Motorola 6843FDC

This FDC is designed for direct interface to the 6800. It provides 10 macrocommands:

1. Seek track 0 (STZ)
2. Seek (SEK)
3. Single-sector-write (SSW)
4. SSW with delected address-mark (SWD)
5. Single-sector read (SSR)
6. Read CRC (RCR)
7. Multiple-sector-write (MSW)
8. Multiple-sector-read (MSR)
9. Free-format-write (FFW)
10. Free-format-read (FFR)

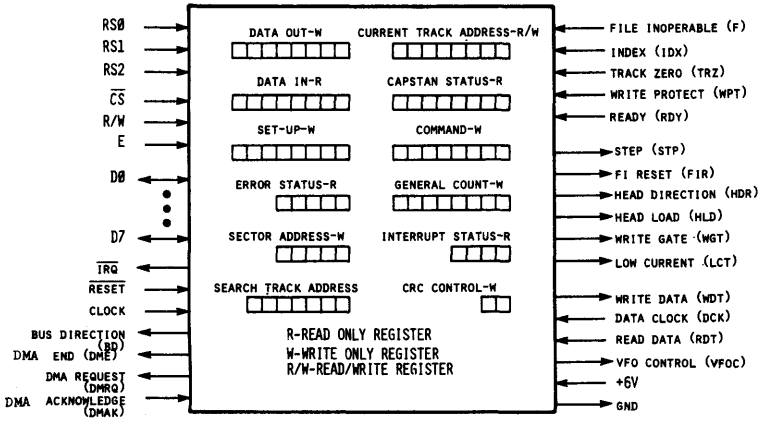


Fig. 4-83 Register Format

It is naturally equipped with two programmable delays for seek-time and for settling-time. The chip signals are illustrated in 4-65. This FDC requires three DMA channels. It uses an average of three percent MPU time. Assuming 256 KPS transfer rate, the maximum MPU load is 12.5%.

Rockwell 10936 FDC.

The basic interconnect of this FDC in a Rockwell system appears on Fig. 4-84. It uses three DMA channels (see Fig. 4-85), where channel 7 refreshes channel 1. The FDC I/O instructions appear in Fig. 4-86. Typical floppy disk routines for the Rockwell PPS-8 appear on Fig. 4-87.

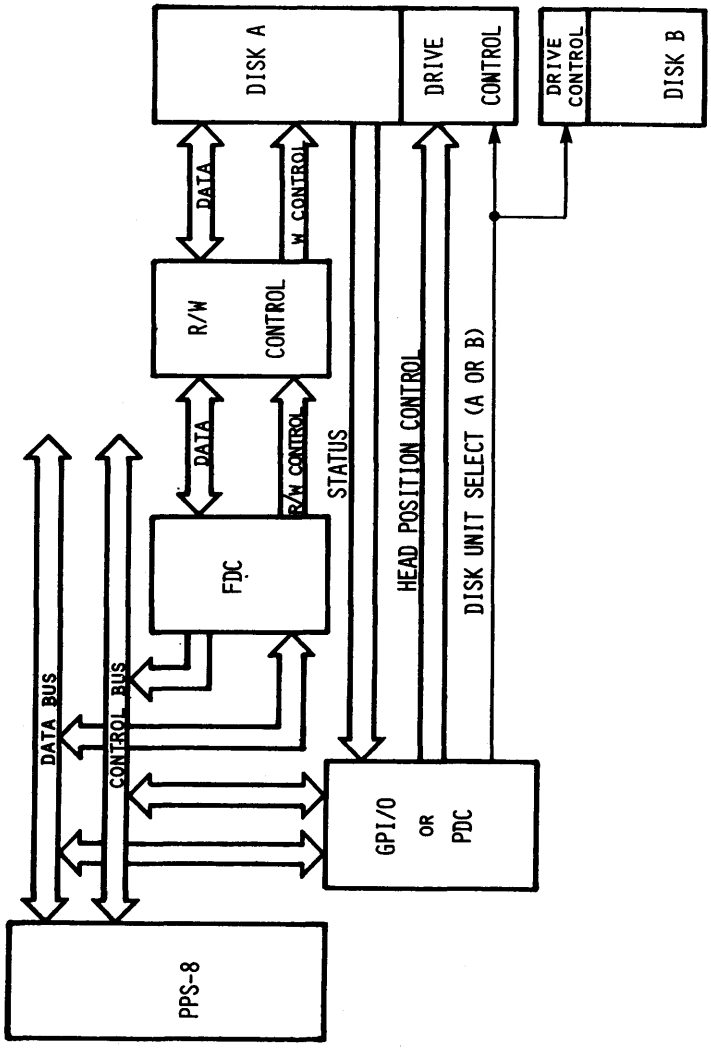


Fig. 4-84 PPS-8 FDC

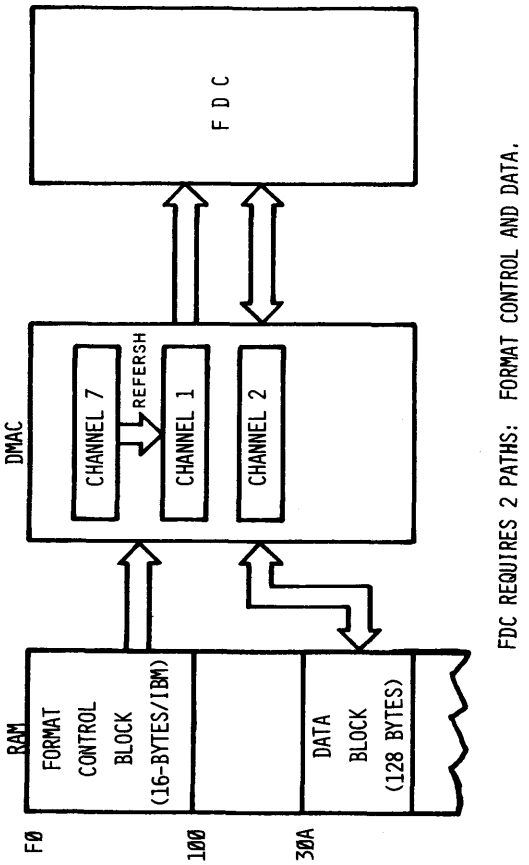


Fig. 4-85 PPS-8 FDC DMAC Block Diagram

0 1 S S 0 0 0 0	NOOP
0 1 S S 0 0 0 1	START
0 1 S S 0 0 1 0	LOAD
0 1 S S 0 0 1 1	CLEAR
0 1 S S 1 0 1 0	READ DATA
0 1 S S 1 1 0 0	READ STATUS
0 1 S S 1 1 0 1	READ STATUS
0 1 S S 0 1 0 0	NOOP
0 1 S S 1 1 1 0	NOOP
0 1 S S 1 0 0 -	UNDEFINED READ
0 0 0 0 1 0 0 0	READ INTERRUPT STATUS

Fig. 4-86 Commands

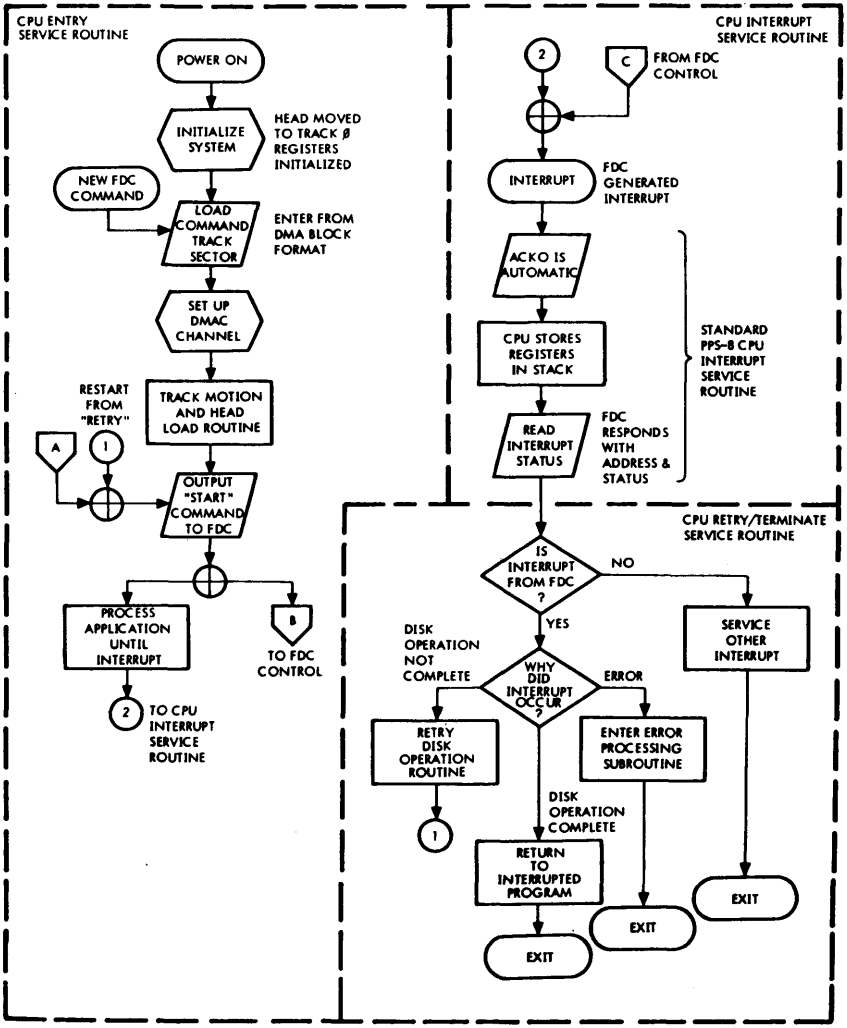
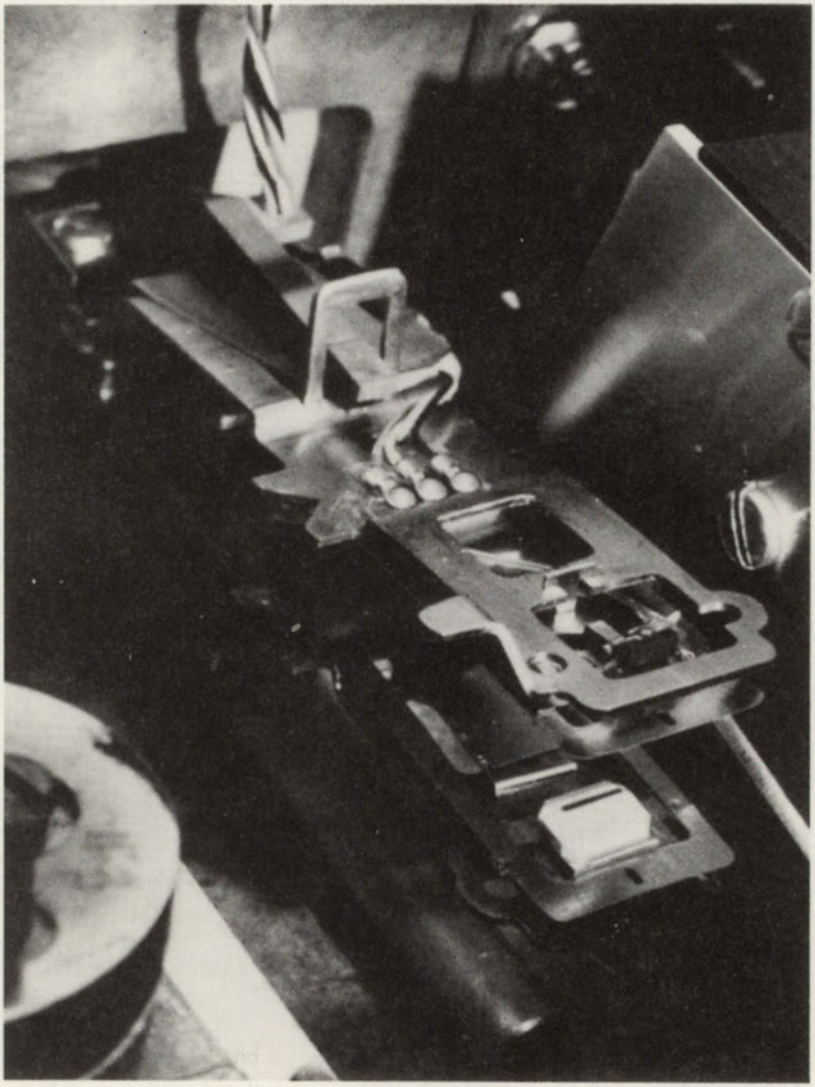


Fig. 4-87 Software Flowchart



CHAPTER 5
ANALOG TO DIGITAL AND DIGITAL TO ANALOG CONVERSION

INTRODUCTION

In any system, two basic kinds of signals must be measured, or generated. They are *analog* and *digital* signals. *Analog signals* assume a *continuous* range of values, whereas *digital signals* assume only a *finite number* of values. As an example, a binary signal is a digital signal which assumes one of two values, either "on" or "off" ("1" or "0"). A typical example of an analog signal is the value of the temperature in an oven. The temperature, being an analog variable, can assume an infinite number of intermediate values.

In view of the finite precision and limited storage of a computer, a digital representation will be used. The precision of the measurement is said to be limited to n significant digits. In addition, *sampling* will be used to reduce the overall storage required. The concept of sampling will be presented below.

This chapter will explain how to perform analog-to-digital conversion (A/D) and digital-to-analog conversion (D/A). In addition, the specific components required to build a complete data collection system will be introduced. We will consider successively:

— a real D/A converter (or DAC)
— a real A/D converter (or ADC)
— the sampling process
— analog multiplexing.

Finally, all these techniques will be used to design a complete data collection system.

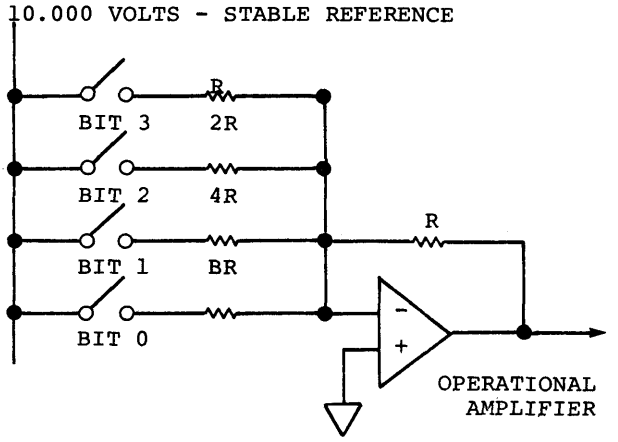
A CONCEPTUAL D/A

Let us consider the problem of converting a binary number into an analog voltage. This is the typical problem of digital-to-analog conversion.

A simple solution is the following: a voltage is generated for each bit-position of the binary number. The value of the voltage is proportional to the binary weight of the bit.

For example, bit 0 will generate a voltage $V(2^0)$; bit 1 will generate a voltage $2V(2^1)$; bit 2 will generate a voltage $4V(2^2)$; and, bit n will generate a voltage $2n \times V$. The resulting voltages are simply added. The result is proportional to the original binary number.

A simple 4-bit D/A appears on Fig. 5-1. This D/A consists of: four switches, four proportional summing resistors, an operational amplifier, and a proportional feedback-resistor. The values of the resistors are in the proportion 1, 2, 4, 8. This results in gains of: $-\frac{1}{8}$, $-\frac{1}{4}$, $-\frac{1}{2}$, and -1. Let us examine the function of this circuit.



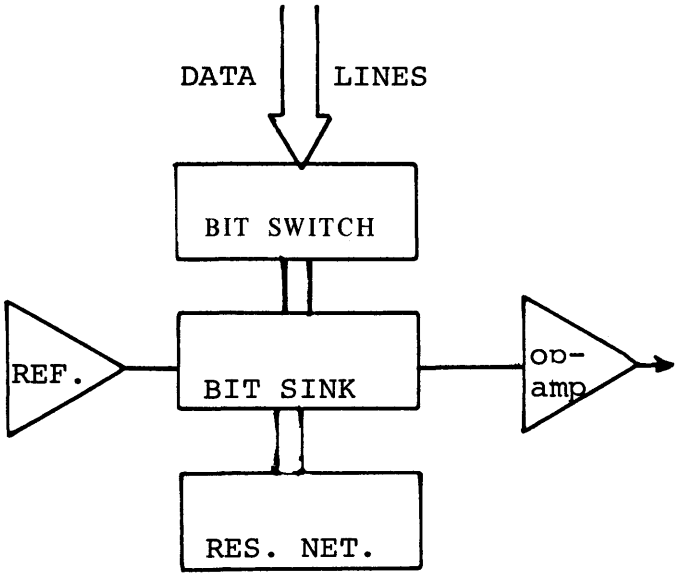
5-1 A Simple 4-Bit D/A

Let us begin with all the switches in the *open* position. Since there is no input to the operational amplifier, the output will be "0". Closing the bit switch numbered "0" will apply the $-10V$ reference to the input of the operational amplifier, through the resistor marked 8R. This will result in an output voltage of 1.25V (due to the gain of $-\frac{1}{8}$ at this point). Closing the switch marked "bit 1" will then add 2.5V to the previous value (1.25V) (due to the gain of $-\frac{1}{4}$ at this point). The resulting output is 3.75V. If all switches are *closed,* the resulting output voltage is 10.0 + 5.0 + 2.5 + 1.25 or 18.75 volts. Here, we have converted a 4-bit binary number, represented by the four switches, into a voltage. It is the analog representation of one of the 16 possible digital values.

We will now examine the structure of a practical D/A converter.

A Practical D/A

The practical design in Fig. 5-2 illustrates the typical design for a monolithic D/A converter. This device has four bits of resolution. Practically, currents are summed instead of voltages, due to the fact that currents are easier to switch on and off accurately. To provide a voltage output, the last stage of the converter is a *current-to-voltage converter.* This is easily done by an operational amplifier. Typical converters have eight bits of resolution.

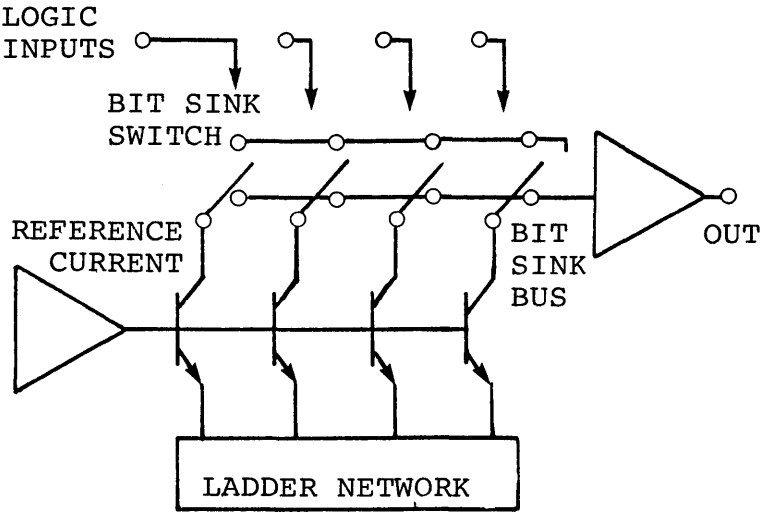


5-2 A Practical Converter Schematic

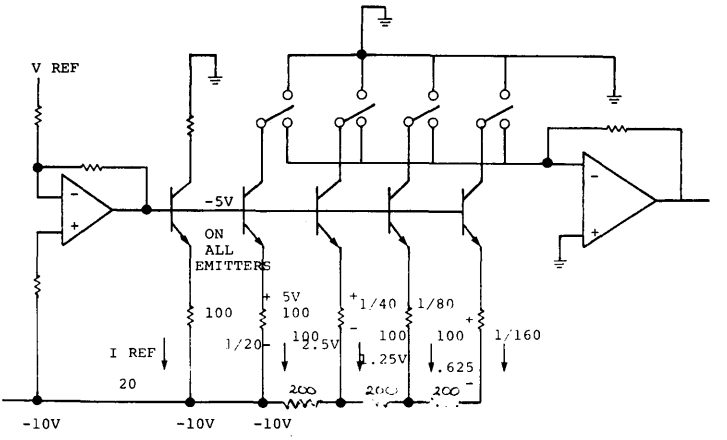
Fig. 5-3 illustrates the functional elements of our converter. They are: the reference-current source, the bit-sink transistors, the ladder-resistor-network, the bit-sink switches, and the voltage-current converter.

The bit-sink current reference establishes a stable reference current. The bit-sink current sources will be proportional to this reference current. The current in each bit-sink transistor is established by its position on the R-2R ladder-resistor-network. The R-2R resistor-network produces a 2^{-n} series of currents flowing through each bit-sink collector. The switches will route the current either to the bit-sink bus, which connects to the current-to-

voltage converter—or to ground. In our example, these currents are $^1/_{20}$ ampere, $^1/_{40}$ ampere, $^1/_{80}$ ampere, and $^1/_{160}$ ampere. These elements combine to perform the conversion from a 4-bit binary number into an analog voltage.

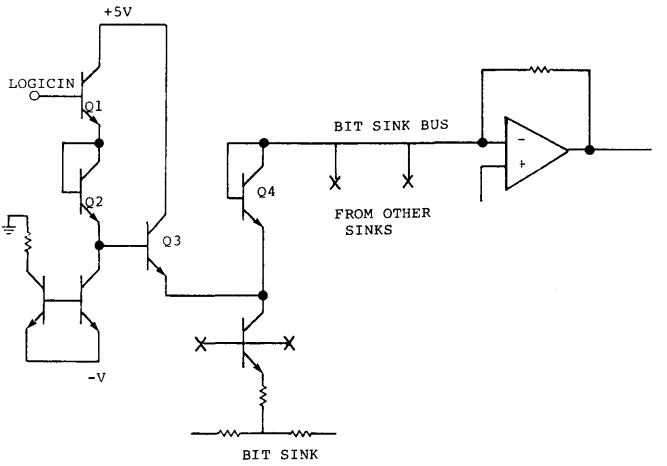


5-3 Monolithic Converter Functional Elements



5-4 Completed Monolithic Converter

An actual monolithic converter uses transistors as switches to route the current between the bit-sink bus to the amplifier and ground. Fig. 5-4 shows the logic-signal-to bit-current-sink switches interface circuitry. When the input is a logic "0", which corresponds to 0V, the bit-sink will draw current through Q4 to the bit-sink bus. When the input is a logic "1", which corresponds to an input voltage greater than 2V, the bit-sink will draw current through Q3, instead of Q4, disconnecting the bit-sink bus from this sink bit. The four binary signals will switch the four bit-sinks on and off the bit-sink bus. The resulting current is converted to the output voltage.



Detail - The Bit Switches

By extending the R-2R ladder network and adding more bit-sink transistors, we can increase the resolution of our converter to more than 10 bits. Any more than 14 bits results in stability problems that this simple circuit cannot overcome. In fact, 16-bit converters are usually certified to be calibrated against a national standard. (One must remember that 16 bits results in an accuracy of 1 part in 65,000!).

Real Products

Table 5-5 represents a sampling of some real products that perform D/A conversion. Cost increases with speed.

TABLE 5-5 D/A Converters

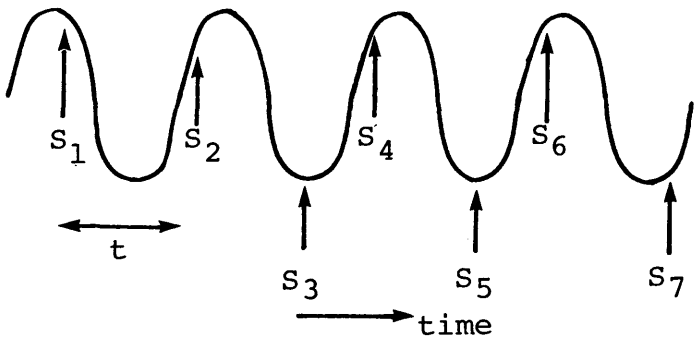
Manufacturer	Type #	Resolution	Speed
Motorola	MC1408	8	300ns
PMI	DAC-08	8	100ns
PMI	DAC-03	10	250ns
Analog Devices	AD7520	10	500ns
Datel	DAC-4Z12D	12	1us
Burr-Brown	DAC70/CSB	16	75us

THE A/D

Now that we have converted the binary representation of a number into an analog signal, we must solve the reverse problem. We must measure an analog signal and convert it into a binary number. There are three methods of conversion: successive approximation, integration, and direct comparison. Before discussing these, we must first examine the concept of sampling.

Sampling

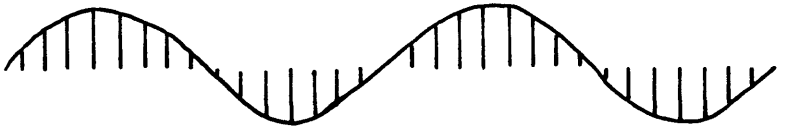
The binary number representing our analog signal represents a value at one point in time. This is known as a *sample*. In the following waveform, in Fig. 5-6, we have sampled where indicated. The sample values will not give us any information as to the true shape of the analog signal. We must collect samples which will accurately represent the signal. The frequency at which we sample is known as the *sampling rate*. In order to represent accurately we must sample more frequently. How often must we sample?



5-6 Infrequent Sampling

The Sampling Theorem

The answer lies in the sampling theorem: *We must sample at least twice as fast as the fastest occurring signal in our system.* As a rough rule, in order to represent our signal, we must sample at least 10 times as fast as our average frequency. Fig. 5-7 illustrates the results of more frequent sampling.



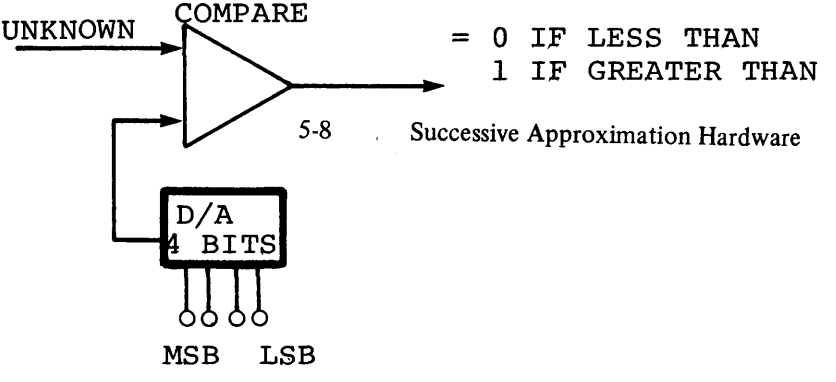
5-7 Frequent Sampling

Sample and Hold

The analog input to a converter must be stable for the duration of the time it takes to complete the conversion. This may be accomplished by using a *sample and hold* circuit. This device will sample the analog input and hold it constant until the next sample of the input. The device holds the sample in a high-quality capacitor, buffered by an operational amplifier. Sample-and-holds are available in both monolithic and hybrid forms.

SUCCESSIVE APPROXIMATION A/D CONVERSION

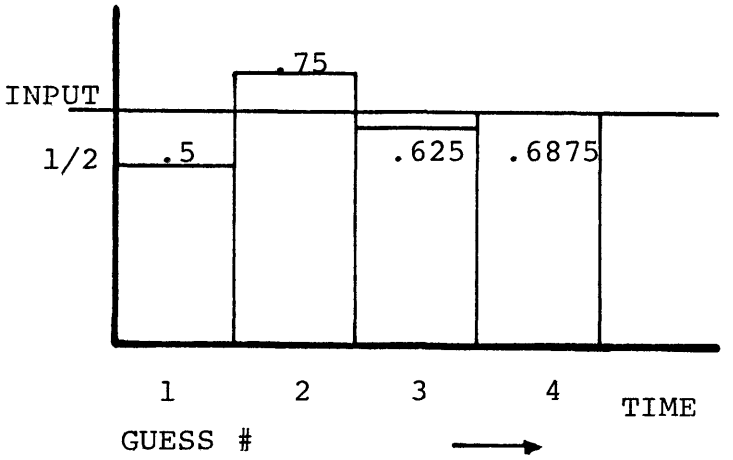
By using a D/A we can perform A/D conversion. Let us compare our unknown analog input signal to a "guess". Increasing or decreasing our guess, based upon the knowledge of our guess being too large or too small, allows us to converge towards the correct value of the analog input signal. In Fig. 5-8, we have connected the output of our 4-bit D/A converter to the



5-8 Successive Approximation Hardware

input of a *comparator*. The other input is connected to our unknown analog signal. The output of the comparator will be "0" if the unknown is *less* than the D/A output, or "1" if the unknown is *greater than* the D/A output.

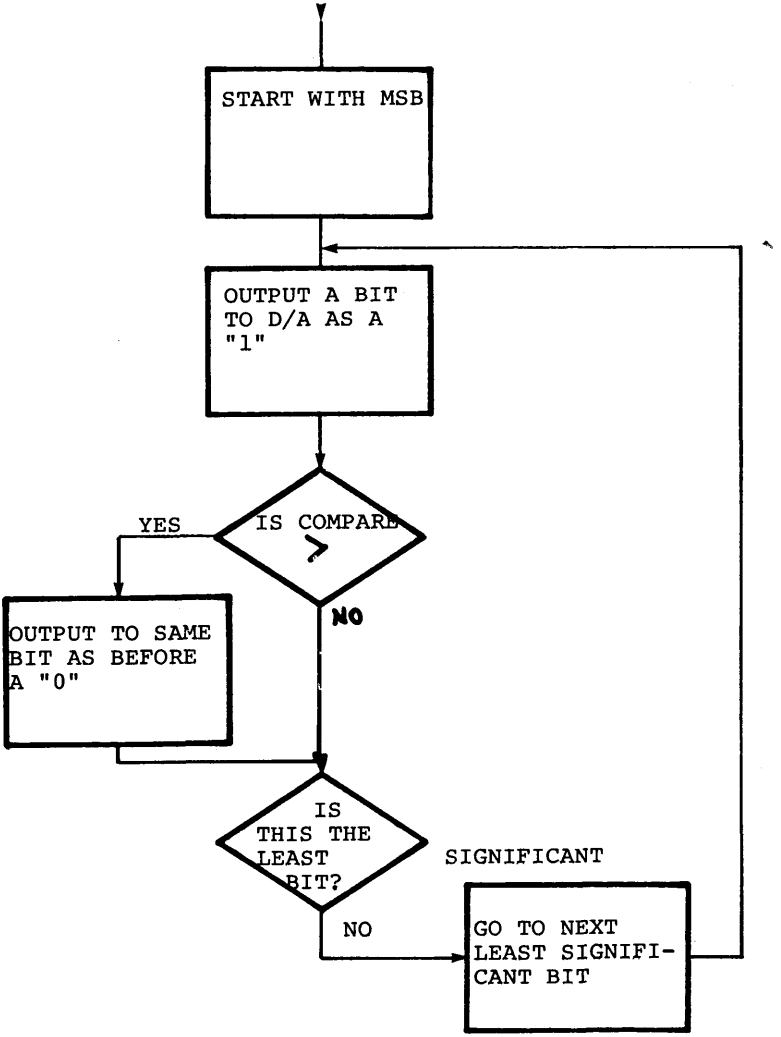
The algorithm for "guessing" is to turn-on each successive bit in our binary number starting with the most significant bit (MSB). As we turn on each bit, we test to see if we are greater than, or less than, at the comparator output. If less-than, we leave the bit on; if greater than, we turn the bit off. In either case, we go to the next least-significant-bit until we reach the last bit. Fig. 5-9 illustrates such a guessing procedure for our 4-bit converter. Fig. 5-10 illustrates the flowchart for the same guessing procedure. This procedure is known as *successive-approximation analog-to-digital conversion.*



5-9 Guess Versus Time: Successive Approximation

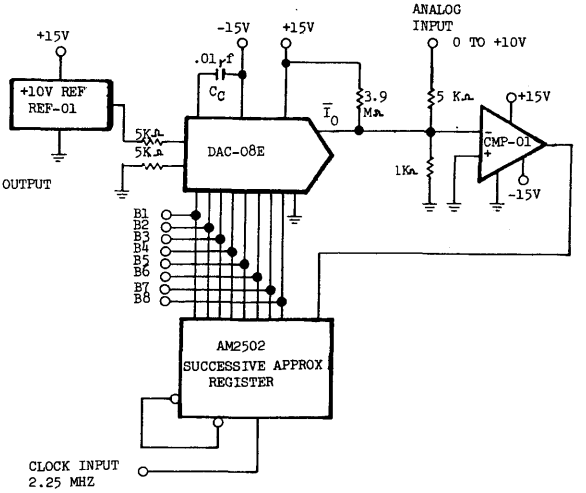
Using this method, we must make as many guesses as there are bits in the binary number being converted to. This is the most common method for A/D conversion.

There are two ways to perform successive approximation: using hardware, and using software mixed with hardware. Fig. 5-11 illustrates a practical A/D design using a *successive-approximation register* or SAR. This successive-approximation-register performs the bit-shift and test-function in hardware. The other alternative is to perform the function of the successive-approximation-register by a *software* algorithm. A complete

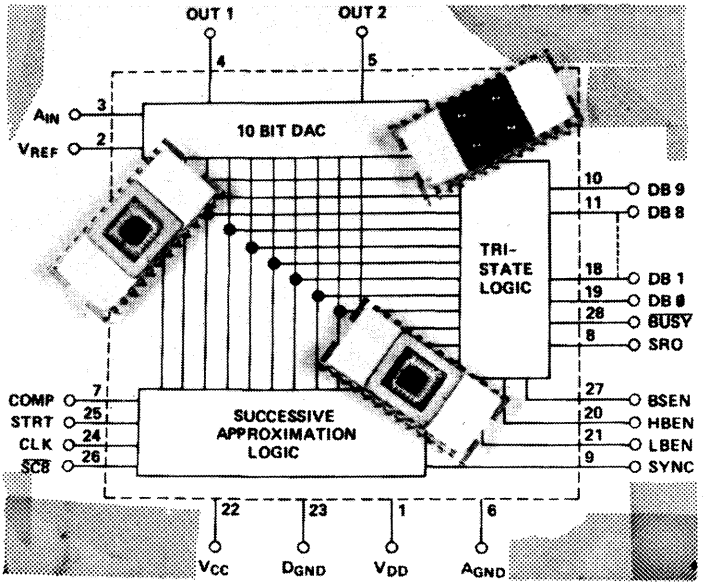


5-10 Successive Approximation Flowchart

monolithic A/D converter without reference, but with the successive-approximation-register, is illustrated in Fig. 5-12. Besides successive-approximation, two other essential techniques are used for conversion: integration, and direct comparison.



5-11 A/D Using a SAR

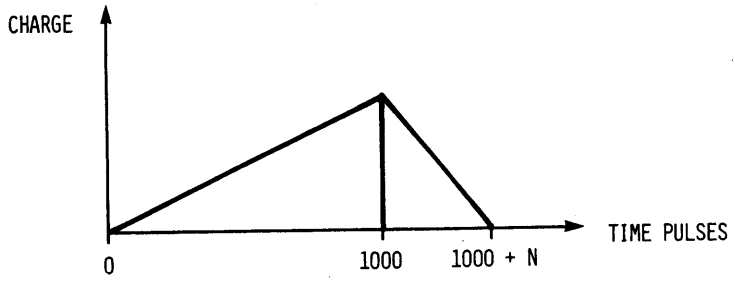


5-12 Monolithic A/D

INTEGRATION

The second way of performing the analog-to-digital conversion is *analog integration*. Measuring the time that it takes for a capacitor to charge to the unknown voltage, and to discharge under a known reference voltage, form the basis of this method. The time-ratio between our known and unknown voltages is equal to the ratio of the values of the two time measurements.

In practice, we integrate a positive unknown voltage and a negative known reference. The positive voltage will result in the charge increasing on the capacitor. After a known period of time, the reference voltage will be applied to the integrator and we will measure the time required for the charge to reach "0". Fig. 5-13 illustrates the timing diagram of such a technique.

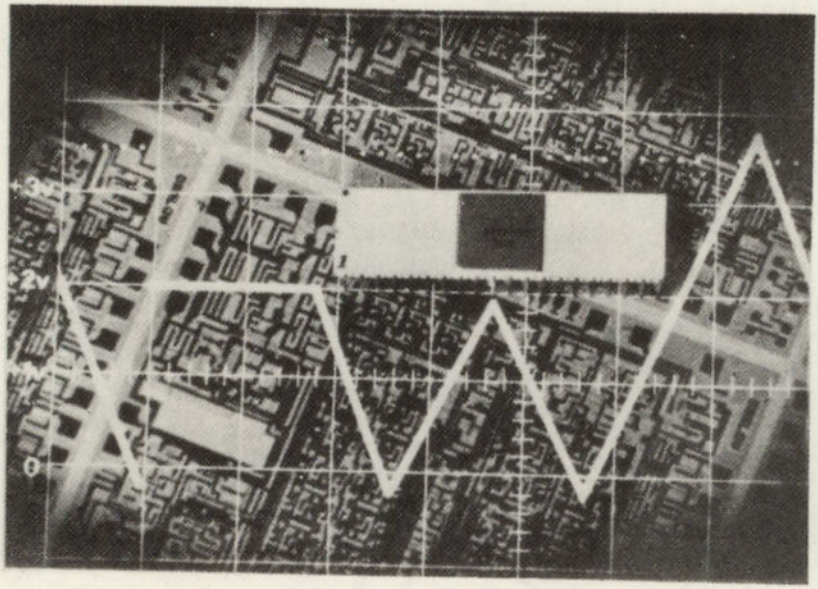


- INTEGRATING CAPACITOR CHARGES AT CONSTANT RATE PROPORTIONAL TO INPUT VOLTAGE
- DISCHARGES BY CALIBRATED REFERENCE CURRENT

5-13 Integration Timing

Fig. 5-14 illustrates a monolithic conversion device using a modification of this *dual-slope integration technique* known as *quad slope*. In addition to integrating the known and unknown voltages, it also integrates inaccuracies caused by offset and ground errors that may be present.

The dual-slope integration technique results in high accuracy measurements. This accuracy comes at the expense of the time that it takes to convert the analog signal. Thus, dual-slope conversion is *slow*, compared to a successive-approximation conversion technique discussed earlier, but *more accurate*.



5-14 Quad Slope Monolithic Converter

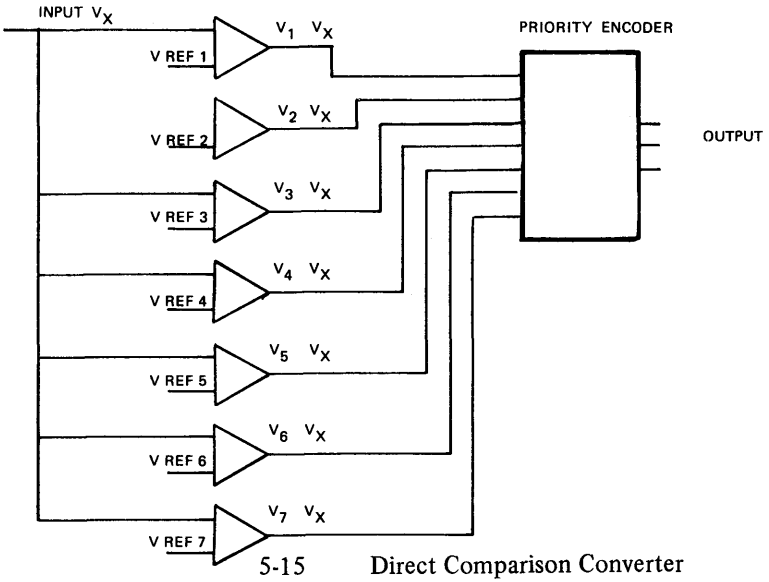
DIRECT COMPARISON

Direct comparison is used only where *extreme speed* is required. Usually less than five bits of resolution are needed in such applications. The circuitry contains 2^{n-1} comparators (where n is the number of bits in the binary output word desired). Let us examine how this works.

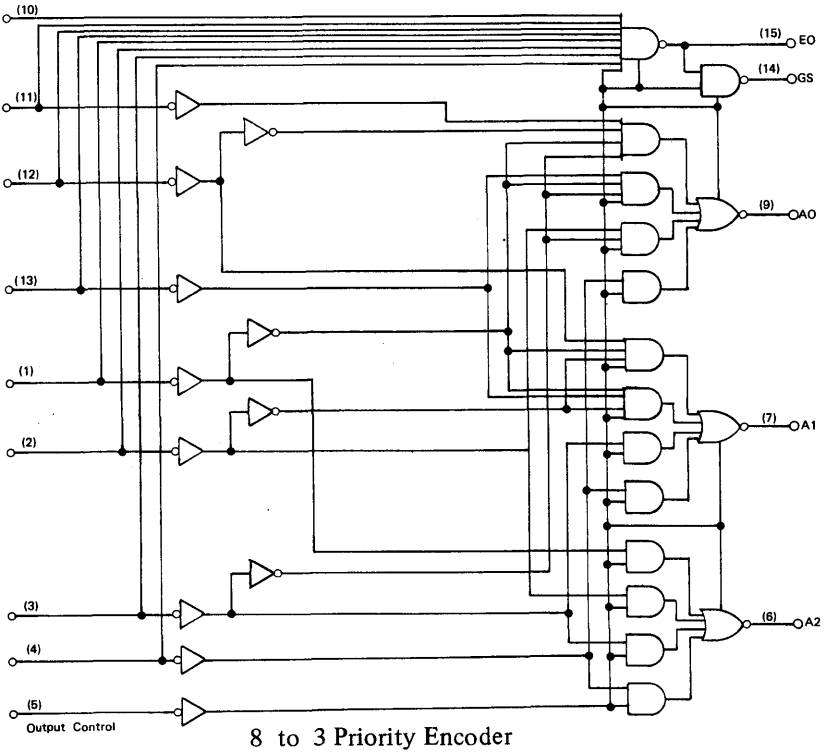
We have a 3-bit direct comparison converter. Our input can be measured in terms of eight levels. Fig. 5-15 illustrates the structure of our converter. The seven comparators will establish if our input voltage is greater than, or less than, each of the possible eight reference values. For example, if all comparators below the fifth one are on, and all above it are off, then the priority encoder will encode the eight inputs into a 3-bit binary number, 100_2. Other inputs will be encoded into other 3-bit representations.

Such systems provide a resolution of five bits in less than 100 ns per conversion. The need for many comparators and reference voltages, and a complex priority scheme, results in this method being the most expensive for anything beyond 3-bits of resolution.

However, AMD has announced a 4-bit monolithic device for less than $50 that will perform this direct comparison in less than 50 ns.



5-15 Direct Comparison Converter



8 to 3 Priority Encoder

TYPICAL DEVICES

Table 5-16 lists examples of conversion devices and the technologies used to implement them. In general, the faster the conversion, the more expensive; the more accurate, the more expensive; and, the more external components required, the more expensive.

TABLE 5-16 A/D CONVERTERS

Manufacturer	Type #	Resolution	Speed	Type of Conversion	Cost
National	MM5357	8	40us	SA	$10
PMI	AD-02	8	8us	SA	
Analog Devices	AD7570	10	18us	SA	$70
Datel	ADC-EK12B	12	24ms	Integrating	
Analog Devices	AD7550	13	40ms	Integrating	$25

A/D SUMMARY

The three techniques of A/D conversion, successive approximation, integration, and direct comparison, are all available as monolithic LSI modules. The trade-offs among the three techniques are simple. The direct converter is fast, but with little accuracy. The successive-approximation converter is of medium speed, and average resolution. Dual-slope integration conversion has the highest accuracy, but requires the most time to perform this conversion. The time the analog signal must be held stable, and the time it takes to convert, determine the maximum sampling frequency and the need for a sample-and-hold circuit.

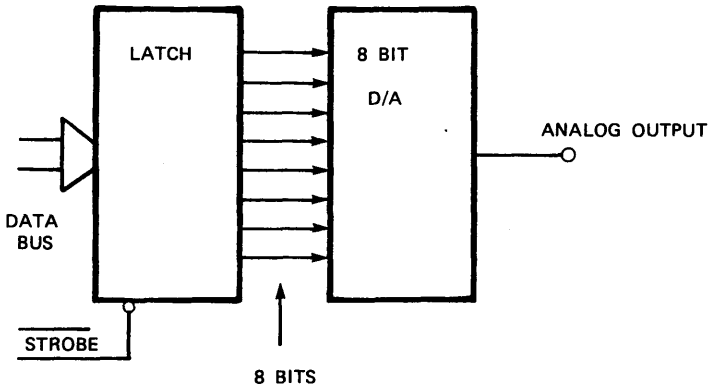
CONSTRUCTING ANALOG-DIGITAL DATA COLLECTION SYSTEMS

To interface these analog conversion products to our digital system to collect, analyze, and control analog signals, we must find out how to use these products optimally. Usually more than one signal needs to be monitored. This indicates that more than one A/D and D/A are required. In some systems, many hundreds of analog signals need to be measured. The techniques used to design a cost-effective system are: simple interfaces, multiplexing, and scaling.

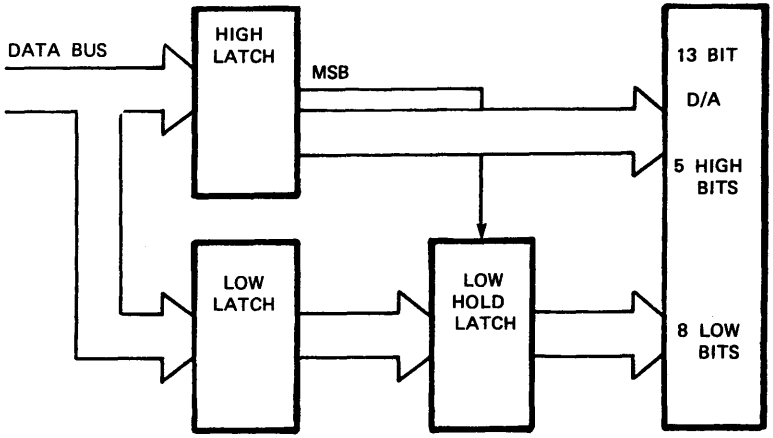
Interfacing the D/A

The D/A converter requires a parallel digital word that will remain stable as long as the analog output is needed. This is easily accomplished for eight

or less bits since most microcomputers have output-latches eight-bit wide. Fig. 5-17 shows such an interface. In the case where the D/A has more than eight bits of resolution, special techniques may be required for interfacing:



5-17 Parallel Output D/A Interface



5-18 Adding the Extra Latch

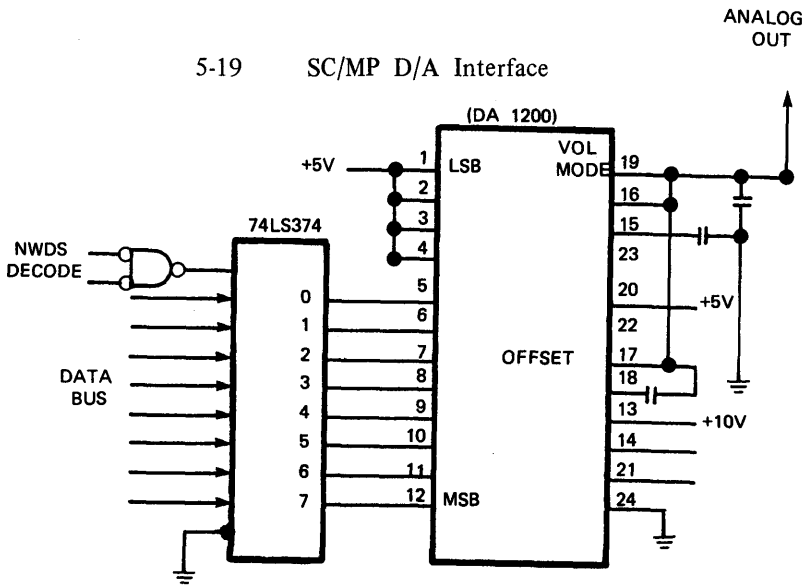
For example, take the case of interfacing a 12-bit D/A converter. If we use two separate 8-bit latches, using 8 bits from the first, and 4 bits from the second, there is a problem. When the first latch is loaded, the D/A converter immediately begins converting to the new value presented. However, some microseconds later we change the second latch, so as to complete the needed bits for the D/A. The effect causes a *glitch* on the D/A output because of the input change. All input bits to a D/A converter must be

changed at the same time in order to prevent output glitches. Fig. 5-18 indicates how an extra latch is added to the low-order latch path, in order to prevent the low bits from changing, until the bits on the high latch change. The low-byte is sent first, and the high-byte is sent second, with the most-significant-bit equal to "1". When the temporary holding low latch is strobed by the "1" from the high latch, the low order bits will pass through to the converter, delayed by the delay of the latch. If this delay is also too severe, a fourth latch may be used in the high bit path to equalize delays.

The new D/A converters *include an on-chip latch* for ease in interfacing.

Example of D/A Interfacing

Fig. 5-19 is the schematic for a D/A interface to the SC/MP microprocessor. The 74LS374 octal latch is used to hold the information while the D/A performs the conversion. Even though this converter has 12 bits of resolution, only 8 are used in this example. The unused inputs are tied to +5V. Unused inputs may be tied to either +5, or 0.0V depending on the binary coding scheme used.



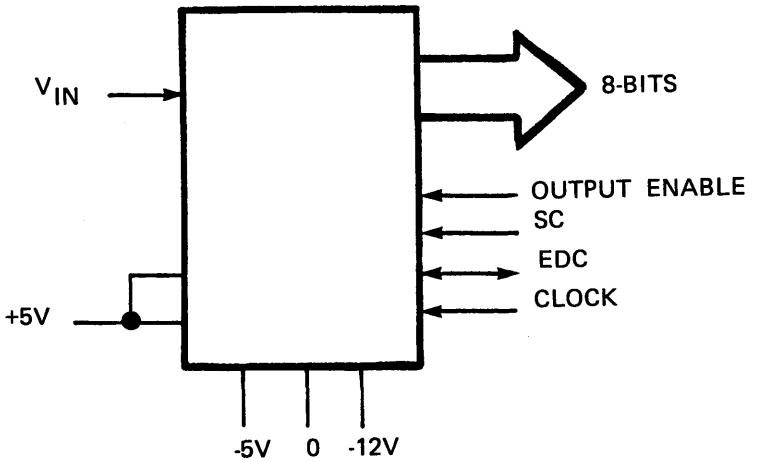
5-19 SC/MP D/A Interface

Interfacing the A/D

As D/A's require an output port, A/D's require an input port. In addition, the A/D requires an output to initiate a new conversion, and an input to indicate when that conversion is complete. The A/D may take as long as 100 ms to complete its conversion. To prevent the processor from executing instructions while waiting, would waste valuable time for processing. The A/D as an input device should operate on a polled or interrupt basis. Five A/D interface examples are presented here:

Examples of A/D Interfacing

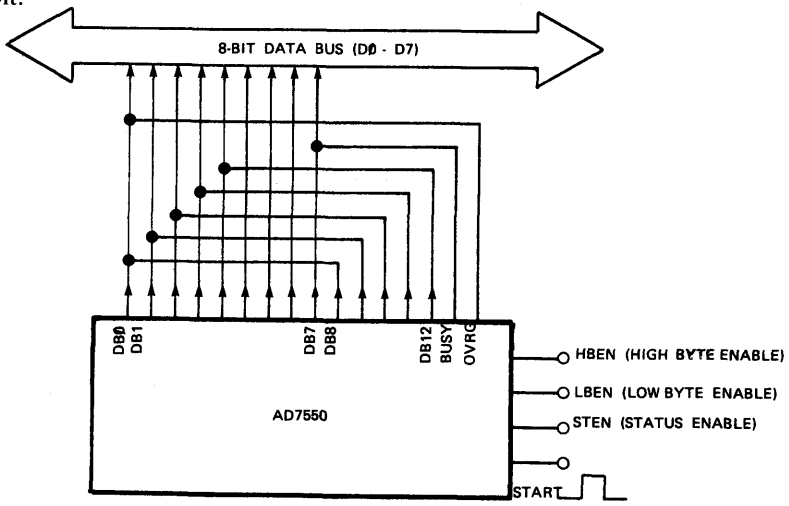
Fig. 5-20 illustrates a National MM5357 A/D converter. The device has eight bits of data output, a start-conversion input, an end-of-conversion output, an output-enable, and a clock signal input. The start-conversion line (SC) may be activated from an output port bit, or it may be tied to the end-of-conversion signal (EOC). If SC is tied to EOC, as soon as a conversion is complete, a new one will immediately begin. The end-of-conversion signal can be connected to an input-port bit, so it may be polled. EOC can also activate an interrupt input, depending on the software considerations.



5-20 The MM5357 Converter

Fig. 5-21 illustrates the use of an A/D converter, where the status of the conversion can be read on the data bus, without the use of bus drivers, or a separate input port. The Analog Devices' AD7550 accomplishes this by

using internal *tri-state output buffers*, that may be independently enabled for the low byte, high byte, and status outputs. The only signal required is the start-conversion signal, which must be generated from an output port bit.

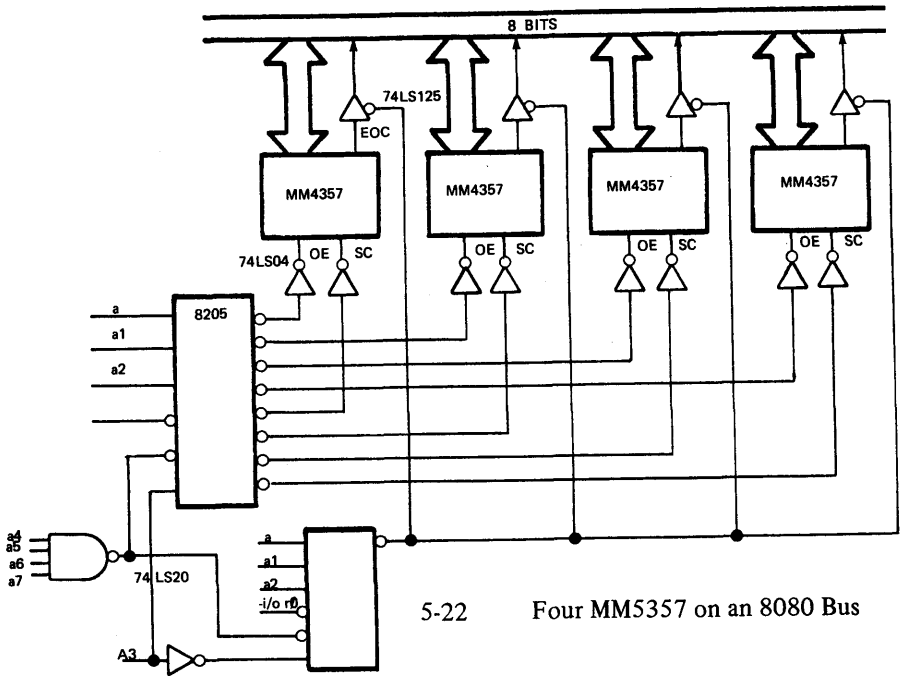


5-21 Analog Devices' AD7550 Interface

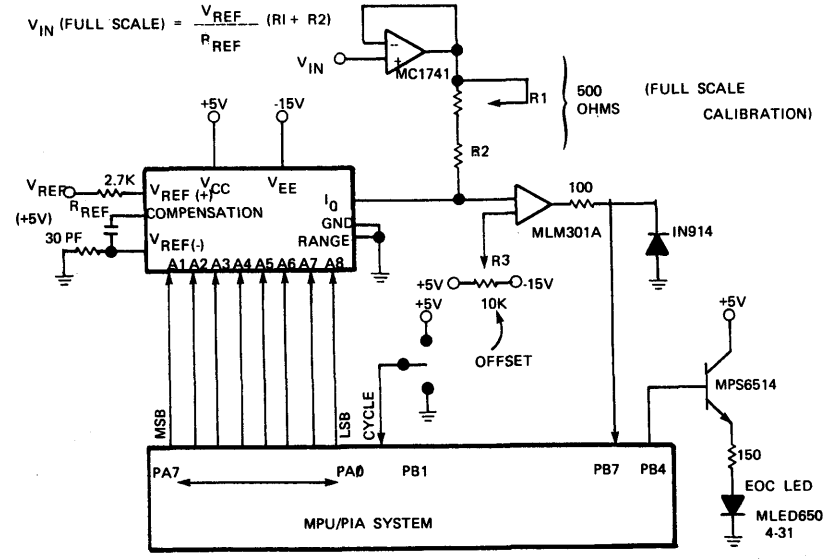
Most systems require more than one analog input. To provide these inputs, we can connect a number of A/D's to the bus and select them with a *decoder*. Each input would have its own A/D. Fig. 5-22 shows the schematic for four converters in an 8080 system. The 8205 decoder selects the data read from ports "F8", "F9", "FA", and "FB" hexadecimal. The ports "FC", "FD", "FE", and "FF", when read, trigger the start conversion lines on the corresponding A/D. Input port "F0" is the end-of-conversion status word, with the lower four bits corresponding to the end-of-conversion outputs of the four A/D's. This port is polled by the program to control the A/D's.

Fig. 5-23 shows a technique for successive-approximation using a D/A comparator. The peripheral input adapter (PIA) controls the D/A converter and has the output of the comparator as its input. By monitoring the output bit of the comparator, we can tell whether the byte output on PA0-PA7 was either too large or too small. The successive-approximation flowchart may be coded in software and used to control the progress of the conversion.

Through these five examples, we have examined the techniques for interfacing the A/D to our microprocessor system. Other A/D converters have similar interfacing requirements. Careful study of the data sheets and a programmable input-output port solve most A/D interfacing problems.



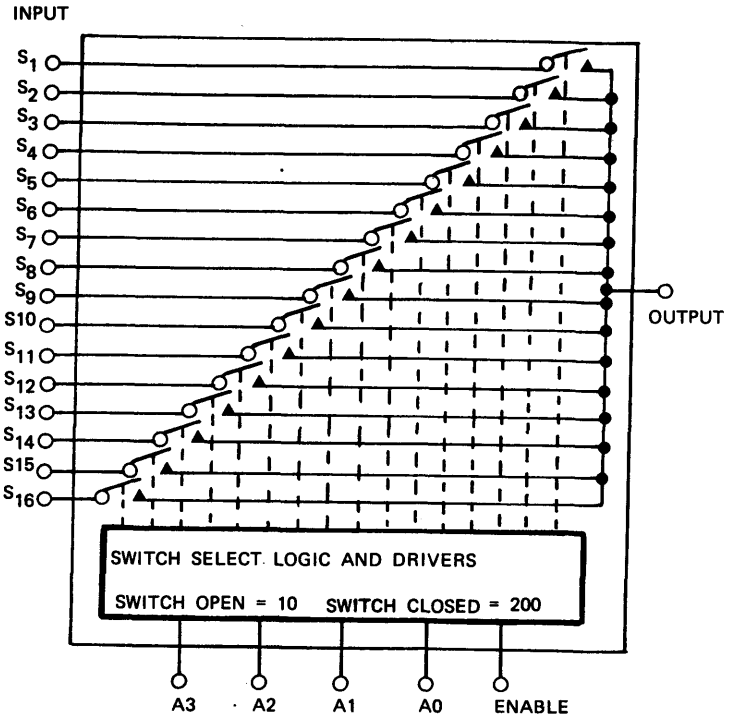
5-22 Four MM5357 on an 8080 Bus



5-23 Using a D/A for Successive Approximation A/D

More Channels

If the need arises for more analog input channels, or it is too expensive to have a single A/D per channel, another alternative exists for design. More channels can be added through the use of *analog multiplexing*. The multiplexer acts as a switch so that the input of the A/D is selected from many sample inputs. It is illustrated in Fig. 5-24. The entire A/D system will contain other components as well. We will now closely examine a hybrid device from Burr-Brown that interfaces directly with the 6800 bus.

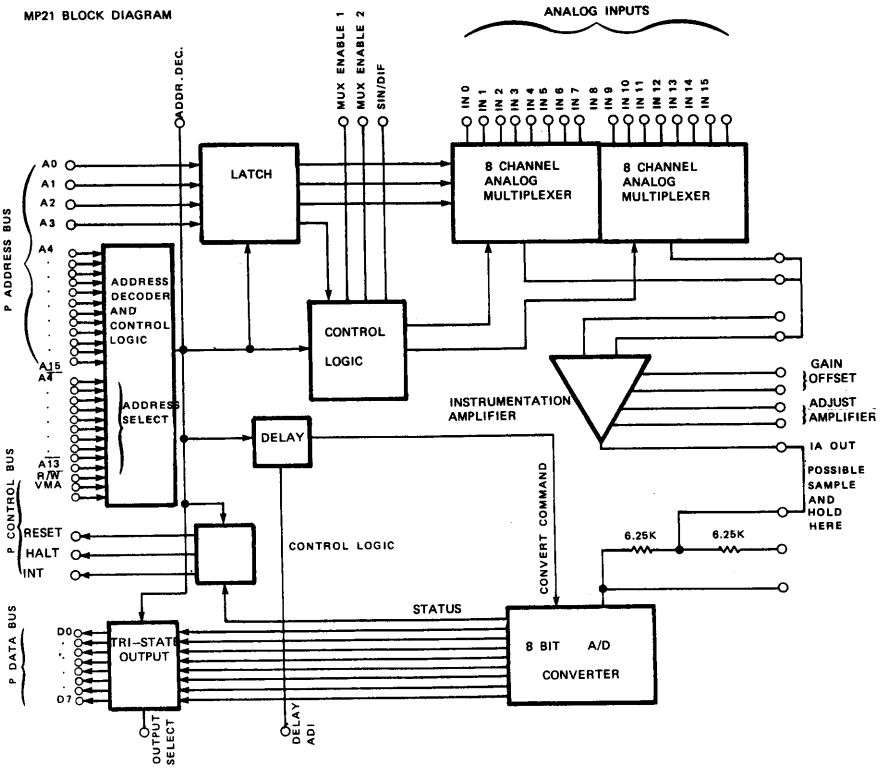


5-24 An Analog Multiplexer

The MP21

The MP21 module contains all necessary components to provide a complete A/D system for a 6800 microprocessor. The block diagram in Fig. 5-25 illustrates the internal functions of the MP21 module. The module has a 16-channel multiplexer, that can be wired to provide eight differential inputs

or 16 single-ended inputs. The channel-number is selected by the lower four address-bits and latched on a read-command by the control-logic.



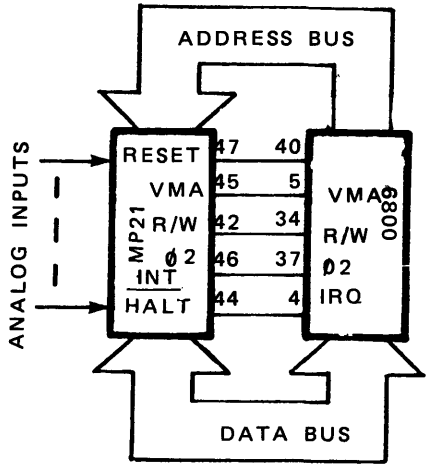
5-25 The Internal MP21 Schematic

The instrumentation amplifier provides the differential-to-single-ended conversion (if required), and can be programmed by external resistors to provide different gains and offsets. If required, a sample-and-hold may be inserted in between the multiplexer and the instrumentation amplifier. Additional multiplexers can be added at this point also to increase the number of input channels.

The heart of the unit is the 8-bit A/D converter which performs the conversion. The end-of-conversion will interrupt the 6800 through the internal interrupt control logic on the hybrid module.

All necessary interfacing has been done for the user so that the module

will be as simple to use as possible. Fig. 5-26 indicates the necessary signals for a typical application utilizing a 650X or 6800 processor.



5-26 6800 and 650X Interface

Techniques for Increasing the Resolution

There are two basic techniques for extending the resolution of our A/D conversion without changing the basic accuracy of our A/D converter. These are *scaling* and *offset*.

Scaling

If the input signal is 1.0 volts and the full scale input of the A/D is +10.0 volts, we should increase the gain of the amplifiers before the A/D converter, so as to take advantage of the full-scale resolution of the A/D. By increasing the gain by a known amount, we can measure smaller signals more accurately. If the input was 20.0 volts, we could decrease the gain of the input amplifier in order to attenuate the input signal. This will allow us to measure larger voltages than could otherwise normally be measured. By these examples, the need for scaling becomes evident. *We scale the input to obtain maximum information upon conversion from our A/D converter.*

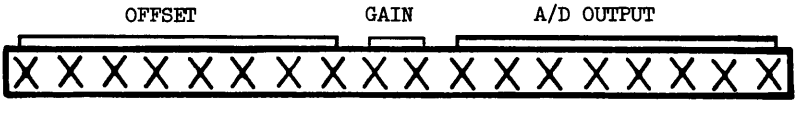
Offset

By connecting the output of a separate D/A to the offset input before our amplifier, we could automatically correct for offset errors, or we could

offset a voltage to increase the accuracy further. If the input is 10.0 volts and we are interested in small changes around this value, we can offset the input by an equal and opposite amount. The output of the offset D/A is then −10.0 volts. Adding the two together, we get some small value which depends upon the difference between the offset D/A and our input voltage. Now, we increase the gain of the input amplifier so that any difference between the input 10.0 volts and the offset 10.0 volts can be measured with the full accuracy of the A/D converter.

Summary of Enhanced Resolution Techniques

By these methods, an 8-bit resolution A/D can be enhanced to provide many more bits of magnitude information in a coded form. The form of our information for this example is listed in Table 5-27.



± 10 VOLTS	x 1/2	(± 10 VOLTS)
	x 1	
	x 10	
	x 100	

5-27 Scaling, Offset A/D Data Format

Of course, the accuracy of the amplifier and offset D/A must be sufficient not to introduce errors of their own in the measurement process.

SUMMARY

Our microprocessor can now be used to gather information, process it, and output that information in a new form in the analog world through the use of these conversion products. The D/A or digital-to-analog-converter providing the microcomputer with the means for generating the analog signals, and the A/D or analog-to-digital-converter providing the means for measuring the analog signals, form the basis of any conversion system. The use of sample-and-hold, multiplexers, and scaling/offset techniques, allow us to quantify any signal, process it, and pass it back in most any form we require.

One last constraint may be to supply interface signals in a standard form. This will be examined in Chapter 6.

ANALOG CIRCUITRY - A/D AND D/A CONVERSION

CHAPTER 6
BUS STANDARDS AND TECHNIQUES

INTRODUCTION

Connecting more than one module requires a communication path. Each module must be able to talk and listen to its neighbors. The components on a module need to communicate with one another. The problem of component interconnection has been addressed in Chapters 2 and 3. The techniques of module-to-module, and system-to-system communication will be covered in this chapter on *busing techniques*.

Two bus types will be distinguished: parallel buses and bit-serial buses. They are:

- parallel
  - microprocessor S100 Bus
  - microprocessor 6800 Bus
  - IEEE-488 general interface bus
  - IEEE-583 CAMAC interface system
- serial
  - EIA-RS232C asynchronous communications
  - EIA-RS422&423 asynchronous and synchronous communications
  - ASCII information standard
  - synchronous communication

Parallel buses are useful for high-speed module-to-module communication in the case of microprocessor buses, and for system-to-system communication in the case of IEEE-488. CAMAC is the only exception—the CAMAC standard covers all communications from the component level on up.

Serial buses require fewer lines, and are used to connect communications terminals to the computer system. Terminals such as, CRT's, (cathode-ray-tube terminals), teleprinters, teletypewriters, and remote data collection devices, all rely on some form of bit-serial communication.

Serial standards cover the bit-rates, electrical characteristics, and data format. There are basically two types of standards: asynchronous and synchronous. The asynchronous standard is used for data rates of less than

20,000 bits-per-second, and the synchronous standard is used for data rates of more than 10,000 bits-per-second. In the overlapping region, both types may be used.

An example of an S100 bus interface to an inexpensive analog-to-digital converter will be presented at the end of this chapter.

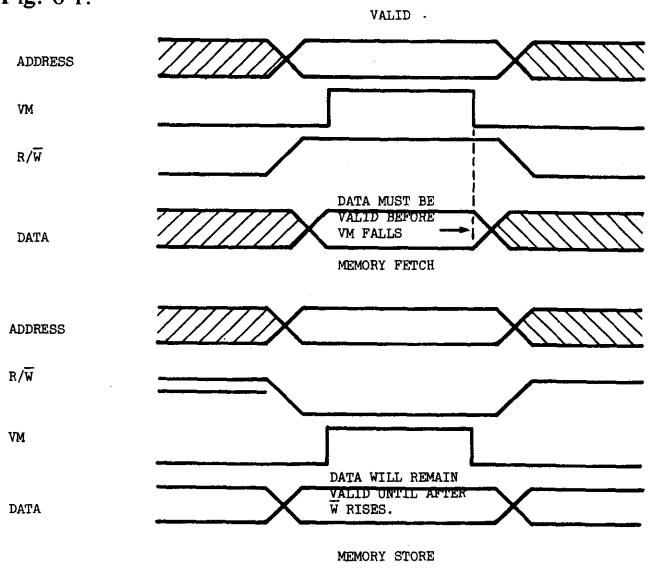
PARALLEL BUSES:

Parallel buses transfer all bits of information across separate wires, at the same time. Lines must be provided for the data-bus, lines for the address-bus, and lines for the control-bus. Each set of lines contains information on the current cycle of operation.

A typical microprocessor system, will need 8 data, 16 address, and 5-12 control lines.

— The 8 data lines are for all transfers in and out of the processor.
— The 16 address lines determine what memory location or I/O port the transfer is for.
— The 5 basic control lines will be a read or write cycle line, a valid address present line, an interrupt line, a DMA request line, and a wait line.

In this basic system, the control bus will have the timing shown in Fig. 6-1.



6.1 Timing on the Control Bus

These 29 signals are all that are needed for most simple parallel buses. Timing will vary, and separate read and write lines may be used, but all buses function in a similar fashion.

Future systems will require at least 16 data lines and perhaps as many as 24 address lines. In addition, many additional control lines are desirable for flexible input-output management.

THE S100 BUS

The "hobby-computer" market was revealed at the Atlantic-City conference in August 1976. The impact of one company, however, was greater than most at the time. This was MITS, the producers of the Altair microcomputers. The bus they used in their 8080-based system had 100 lines. Other manufacturers (in particular IMSAI) realized that making their memories and peripherals compatible would help them sell in this new market. Now there are over 600 different types of boards and systems available for this bus from over 100 manufacturers.

The bus signals and definitions are presented in Tables 6-2 through 6-8.

Some problems of this bus are: clock lines adjacent to control signals, pin layout problems, and power supply distribution.

The Φ1, Φ2, and 2MHz clock signals are near nine other control signals. All of these clock-pulses have sharp rise and fall times and occur continuously. Because of this, these clock-signals are most easily coupled to the other lines, unless unusal shielding measures are taken. Because of the 2MHz clock present, the bus must be designed for 4MHz noise immunity when no other signal occurs at that rate.

What if a board is unplugged with the power on? The possibility of the −18 volts touching the +8 volts, due to misalignment, is great. If this happens . . . well, let us hope it doesn't. At best, only the regulators may blow out; at worst, every chip tied to +5 volts may be damaged.

Ideally, boards should be protected against being unplugged or reversed with power on. A symmetric arrangement of power pins that will shut down all power if boards are inserted improperly is one good idea, and the careful distribution of voltages in between grounded pins is another good idea. Variations in supply voltage from module to module reduces noise immunity and may cause difficulties. The solution is to use high quality regulators costing more or matching those used (an impossible job). There is no best way to solve this problem—and a central power distribution scheme has its own problems.

The interrupt lines are reserved for interrupt requests to an interrupt-controller board on the bus. No standard way of using these is established

PIN NUMBER	SYMBOL	NAME	FUNCTION
1	+8V	+8 Volts	Unregulated voltage on bus, supplied to PC boards and regulated to 5V.
2	+18V	+18 Volts	Positive pre-regulated voltage.
3	XRDY	EXTERNAL READY	External ready input to CPU Board's ready circuitry.
4	VI0	Vectored Interrupt Line 0	
5	VI1	Vectored Interrupt Line 1	
6	VI2	Vectored Interrupt Line 2	
7	VI3	Vectored Interrupt Line 3	
8	VI4	Vectored Interrupt Line 4	
9	VI5	Vectored Interrupt Line 5	
10	VI6	Vectored Interrupt Line 6	
11	VI7	Vectored Interrupt Line 7	
12	*XRDY2	EXTERNAL READY 2	A second external ready line similar to XRDY.
	*New bus signal for 8800b.		
13 to 17	TO BE DEFINED		
18	STAT DSB	STATUS DISABLE	Allows the buffers for the 8 status lines to be tri-stated.

PIN NUMBER	SYMBOL	NAME	FUNCTION
19	C/C DSB	COMMAND/CONTROL DISABLE	Allows the buffers for the 6 output command/ control lines to be tri-stated.
20	UNPROT	UNPROTECT	Input to the memory protect flip-flop on a given memory board.
21	SS	SINGLE STEP	Indicates that the machine is in the process of performing a single step (i.e., that SS flip-flop on D/C is set).
22	ADD DSB	ADDRESS DISABLE	Allows the buffers for the 16 address lines to be tri-stated.
23	DO DSB	DATA OUT DISABLE	Allows the buffers for the 8 data output lines to be tri-stated.
24	02	PHASE 2 CLOCK	
25	01	PHASE 1 CLOCK	
26	PHLDA	HOLD ACKNOWLEDGE	Processor command/ control output signal that appears in response to the HOLD signal; indicates that the data and address bus will go to the high impedance state and processor will enter HOLD state after completion of the current machine cycle.
27	PWAIT	WAIT	Processor command/ control signal that appears in response to the READY signal going low; indicates processor will enter a series of .5 microsecond WAIT states until READY again goes high.

PIN NUMBER	SYMBOL	NAME	FUNCTION
28	PINTE	INTERRUPT ENABLE	Processor command/ control output signal; indicates interrupts are enabled, as determined by the contents of the CPU internal interrupt flip-flop. When the flip-flop is set (Enable Interrupt instruction), interrupts are accepted by the CPU; when it is reset (Disable Interrupt instruction), interrupts are inhibited.
29	A5	Address Line 5	
30	A4	Address Line 4	
31	A3	Address Line 3	
32	A15	Address Line 15	(MSB)
33	A12	Address Line 12	
34	A9	Address Line 9	
35	DO1	Data Out Line 1	
36	DO0	Data Out Line 0	(LSB)
37	A10	Address Line 10	
38	DO4	Data Out Line 4	
39	DO5	Data Out Line 5	
40	DO6	Data Out Line 6	
41	DI2	Data In LIne 2	
42	DI3	Data In Line 3	
43	D17	Data In Line 7	(MSB)
44	SM1	MACHINE CYCLE 1	Status output signal that indicates that the processor is in the fetch cycle for the first byte of an instruction.

PIN NUMBER	SYMBOL	NAME	FUNCTION
45	SOUT	OUTPUT	Status output signal that indicates the address bus contains the address of an output device and the data bus will contain the output data when PWR is active.
46	SINP	INPUT	Status output signal that indicates the address bus contains the address of an input device and the input data should be placed on the data bus when PDBIN is active.
47	SMEMR	MEMORY READ	Status output signal that indicates the data bus will be used to read memory data.
48	SHLTA	HALT	Status output signal that acknowledges a HALT instruction.
49	CLOCK	CLOCK	Inverted output of the 02 CLOCK'
50	GND	GROUND	
51	+8V	+8 Volts	Unregulated input to 5 volt regulators.
52	-18V	-18 Volts	Negative pre-regulated voltage.
53	SSWI	SENSE SWITCH INPUT	Indicates that an input data transfer from the sense switches is to take place. This signal is used by the Display/ Control logic to: a) Enable sense switch drivers; b) Enable the Display/ Control Board driver's Data Input (FDI0-FDI7); c) Disable the CPU Board Data Input Drivers (DI0-DI7).

PIN NUMBER	SYMBOL	NAME	FUNCTION
54	EXT CLR	EXTERNAL CLEAR	Clear signal for I/O devices (front-panel switch closure to ground).
55	*RTC	REAL-TIME CLOCK	60HZ signal is used as timing reference by the Real-Time Clock/ Vectored Interrupt Board.
56	*STSTB	STATUS STROBE	Output strobe signal supplied by the 8224 clock generator. Primary purpose is to strobe the 8212 status latch so that status is set up as soon in the machine cycle as possible. This signal is also used by Display/Control logic.
57	*DIGI	DATA INPUT GATE 1	Output signal from the Display/Control logic that determines which set of Data Input Drivers have control of the CPU board's bidirectional data bus. If DIGI is HIGH, the CPU drivers have control; if it is LOW, the Display/ Control logic drivers have control.
58	*FRDY	FRONT PANEL READY	Output signal from D/C logic that allows the front panel to control the READY lines to the CPU.
59 to 67	TO BE DEFINED		

*New bus signal for 8800b.

PIN NUMBER	SYMBOL	NAME	FUNCTION
68	MWRITE	MEMORY WRITE	Indicates that the data present on the Data Out Bus is to be written into the memory location currently on the address bus.
69	PS	PROTECT STATUS	Indicates the status of the memory protect flip-flop on the memory board currently addressed.
70	PROT	PROTECT	Input to the memory protect flip-flop on the board currently addressed.
71	RUN	RUN	Indicates that the 64 /RUN flip-flop is Reset; i.e., machine is in RUN mode.
72	PRDY	PROCESSOR READY	Memory and I/O input to the CPU Board wait circuitry.
73	PINT	INTERRUPT REQUEST	The processor recognizes an interrupt request on this line at the end of the current instruction or while halted. If the processor is in the HOLD state or the Interrupt Enable flip-flop is reset, it will not honor the request.

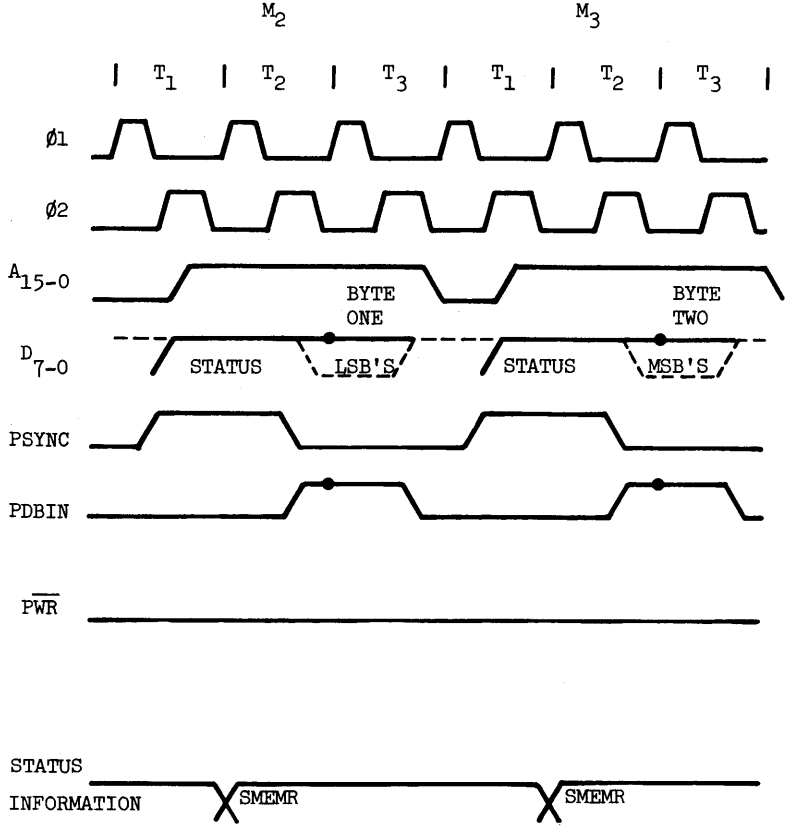
PIN NUMBER	SYMBOL	NAME	FUNCTION
74	PHOLD	HOLD	Processor command/ control input signal that requests the processor enter the HOLD state; allows an external device to gain control of address and data buses as soon as the processor has completed its uses of these buses for the current machine cycle.
75	PRESET	RESET	Processor command/ control input; while activated, the content of the program counter is cleared and the instruction register is set to 0.
76	PSYNC	SYNC	Processor command/ control output; provides a signal to indicate the beginning of each machine cycle.
77	PWR	WRITE	Processor command/ control output; used for memory write or I/O output control. Data on the data bus is stable while the PWR is active.
78	PDBIN	DATA BUS IN	Processor command/ control output; indicates to external circuits that the data bus is in the input mode.
79	A0	Address Line 0	(LSB)
80	A1	Address Line 1	
81	A2	Address Line 2	

PIN NUMBER	SYMBOL	NAME	FUNCTION
82	A6	Address Line 6	
83	A7	Address Line 7	
84	A8	Address Line 8	
85	A13	Address Line 13	
86	A14	Address Line 14	
87	A11	Address Line 11	
88	DO2	Data Out Line 2	
89	DO3	Data Out Line 3	
90	DO7	Data Out Line 7	
91	DI4	Data In Line 4	
92	DI5	Data In Line 5	
93	DI6	Data In Line 6	
94	DI1	Data In Line 1	
95	DI0	Data In Line 0	(LSB)
96	SINTA	INTERRUPT ACKNOWLEDGE	Status output signal; acknowledges signal for INTERRUPT request.
97	SWO	WRITE OUT	Status output signal; indicates that the operation in the current machine cycle will be a WRITE memory or output function.
98	SSTACK	STACK	Status output signal; indicates that the address bus holds the pushdown stack address from the Stack Pointer.
99	POC	POWER-ON CLEAR	
100	GND	GROUND	

as Z-80's, 6502's (and even 6800's) can also be used (and are used) in S100 systems.

The other host of signals are control signals. The S100 bus has far more than anyone will ever need of these, and suffers from being designed before a system-controller chip was made available for the 8080. Because of this, many of the signals are due to the original Intel problem with pin limitations, as discussed in Chapter 2. Obviously, a new S100 bus would be needed, with these control signals reduced to a managable number. This will probably never happen. *A standard can always be improved: but it won't be—this is why it is a standard!*

The S100 bus is a practical bus, and will perform well in most applications. The problems mentioned here should be avoided, when new bussing schemes are being considered in the next few years for future systems.



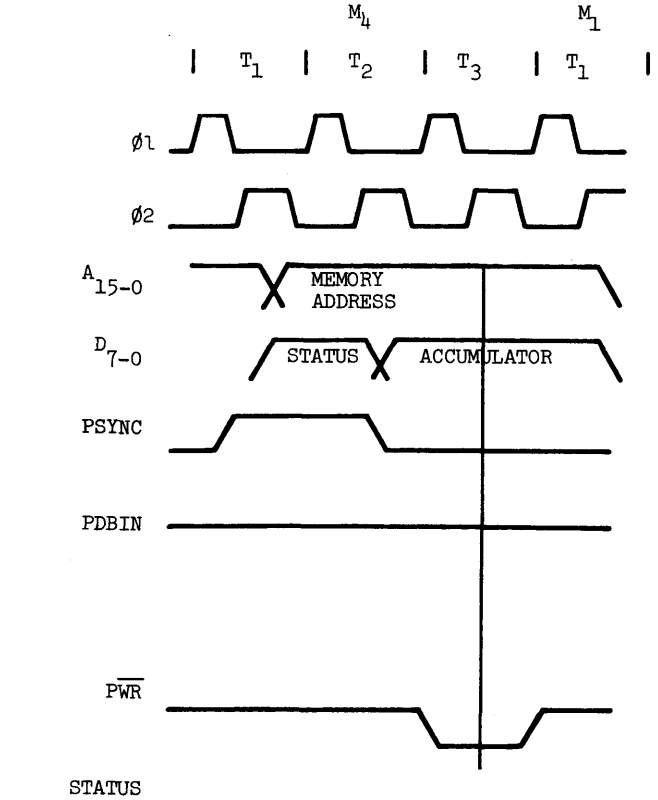
6.9 Memory Read Cycle on S100 Bus

The bus provides: 8 data in, 8 data out, 16 address, 3 power-supply, 8 interrupt and 39 control lines. Other pins are unused or reserved for future use.

The *data-bus* has been changed from the normal bidirectional 8080 bus to two unidirectional data buses. One for data-input to the processor, and one for data-output from the processor. In this system, there is no real advantage to this, as many peripherals actually hardwire the two buses together. There is also no real disadvantage, except the need for eight more pins.

The *address-bus* is the typical buffered 16-address-lines, which are found in every standard microprocessor system.

The *power-supplies* are most interesting. There are two philosophies for



6.10 Memory Write Cycle on S100 Bus

power distribution: regulate at a control location and distribute power, or regulate locally on each module. Altair chose the latter. It is a good choice because power distribution to the modules is simplified, and noise cross-coupling between the modules is reduced. It is a more expensive choice in that the regulators cost must more than a single good regulator would cost, and it is a marginal choice due to the variations in regulated voltages between modules.

The design of an S100-bus-compatible peripheral is discussed in the example at the end of this chapter. Timing diagrams for memory-fetch and store cycles are presented in Fig. 6-9 and 6-10. They illustrate the basic timing of the 8080 system, and the basic signals used for these transfers. Note how the most important signals are $\overline{\text{PWR}}$ and PDBIN. These two signals control the direction of the data on the buses: fetching or storing. In conjunction with the status information, all memory transfers can be identified by these few lines.

A 6800 SYSTEM BUS:

Described here is the Altair-680B 6800-system-bus. This bus was well thought-out, by comparison to the problems of the S100-bus.

The system has eight bidirectional lines for data, sixteen unidirectional lines for address, and nine control lines.

The data and address buses are quite the same as any other system's buses. The control lines contain the minimum number of useful lines needed. They are: clock Φ2, reset, halt, R/$\overline{\text{W}}$, VMA, DBE, R/$\overline{\text{W}}$-P, BA, and TSC. These are summarized in Table 6-11. Not described in the Table are the IRQ and NMI interrupt-request lines. They appear also on the control bus.

This bus provides clean, concise signals for fetching and storing information. It is an example of a well thought-out design. Unfortunately, the Φ1, and Φ, drive and Φ2-drive clock signals are present on this bus for no reason, except presumably to decrease noise-immunity. One well-isolated high-speed clock is all that most buses can have, without resorting to unusual and expensive shielded backplanes.

IEEE-488-1975

This bus is intended for connecting systems, rather than modules. Such devices as computers, voltmeters, power-supplies, frequency-generators, and others can be equipped with a 488 bus. The 488-bus was a result of three years of discussion in the IEC (International Electrotechnical Commis-

SYSTEM CONTROL BUS

The System Control Bus consists of the following signals:

CLOCK: The system clock is a 500 KHz asymmetrical, two phase, non-overlapping clock that runs at the vcc voltage level. Phase one (01) is used for internal chip operations. All data transfers take place during Phase Two (02). Therefore, 02 is used throughout the system to enable memory and interfaces such as the Asynchronous Communication Interface Adapter (ACIA).

RESET: This input is used to initialize the system after a power down condition due to either an initial start-up or power failure. It is also used to reinitialize the MPU at any time after start up. When a positive edge is detected on the RESET input, which is caused by a manual front panel reset, the MPU will begin the restart sequence. Within the restart sequence, the Program Counter is loaded with the contents of the reset vector location (FFFE, FFFF), which contains the starting address of the System Monitor.

HALT: The Halt line is used for external control of program execution. When in the high state (RUN), the MPU will fetch the instruction addressed by the program counter and begin program execution. When the Halt line is low, all of the activity within the MPU will be halted. The Bus Available (BA) signal will then go high and the Read-Write (R/W), Address and Data lines will all be in the high impedance state. With BA high, the front panel addressing and data deposit functions will be enabled.

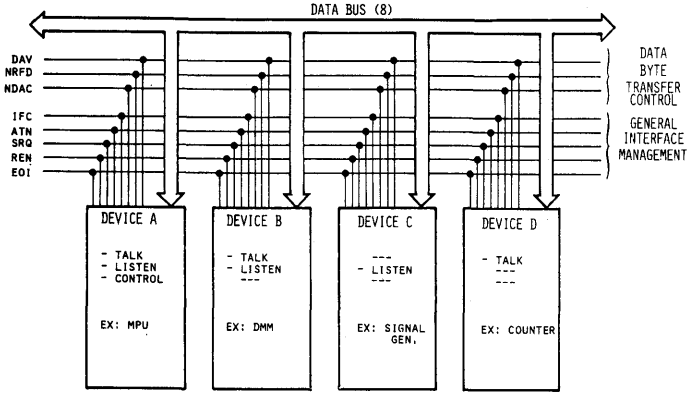
R/W: Read/Write controls and indicates the direction of data transfer. When in the high state (READ), data is read into the MPU from memory and peripherals. When in the low state (WRITE), data is written into memory or peripherals. When the processor is halted, R/W will turn to the off (high impedance) state.

VMA: The VMA output indicates to the memory or the peripherals, such as an ACIA, that a stable, valid memory address is on the bus.

DBE: The DBE input is the three-state control signal for the MPU data bus and will enable the bus drivers of the 6800 when in the high state. Phase 2 is used to directly drive this input. During an MPU read cycle, the data bus drivers are disabled internally, i.e., within the MPU.

R/W-P: Read/Write-Prime is developed by NANDing the Read/Write signal and 02. The Read/Write-Prime signal assures that data will always be read or written while the data bus is enabled and not during period of invalid data.

sion). In 1974, the IEEE approved the IEC draft, resulting in IEEE-488-1975. Hewlett-Packard was one of the prime influences in the development of this bus, and the handshake technique used is patented by Hewlett-Packard. All producers of a 488 compatible interface must purchase the license to use the bus handshake circuitry (The bus is sometimes called HPIB or Hewlett-Packard Interface Bus).



6.12 488 Bus Signals

The basic bus connects to devices that can do one or more of the following:

1. control other units — *controller*
2. take information from the controlling unit — *listener*
3. give information to the controlling unit — *talker*

The bus consists of eight bidirectional data lines, three byte-transfer control lines, and five general control lines.

The eight data lines will carry: device commands (only 7 bits used), address and data (8 bits).

Since this system *has no address or complete-control buses,* the data bus is used to perform all these functions. The rest of the lines control the function of the data-bus and how it is used.

The transfer-control lines are used to implement the "handshaking" required between the device outputting and the device inputting.

The last five lines control the general conditions of the system. These are: Attention, Interface Clear, Service Request, Remote Enable, and End-or-Identify.

Attention, when false, indicates that the data-lines contain data from one to eight bits. When true, the data-bus contains a seven-bit command or seven-bit address.

Interface Clear puts the system in a known state. It is similar to a system-reset.

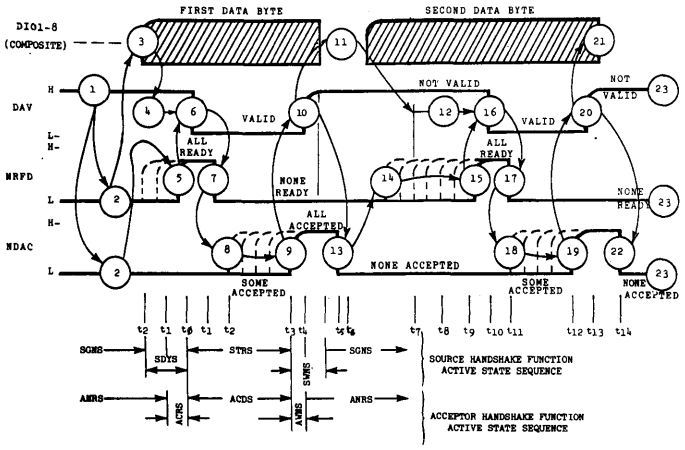
Service Request, when set true, flags the controlling unit to indicate a device needs attention.

Remote Enable sets the mode of each device, in conjunction with other codes, to operate remotely or locally.

End-or-Identify is used to flag the controlling unit, as to the end of a data transfer.

The "handshaking" function is used when devices must wait for information to become available. One line says, "How do you do?" The other replies, "Fine, thank you, I have something for you". In return, the reply is: "Please give it to me, I am ready". The dialogue continues with, "OK, here it is", and ends with, "Thank you, nice meeting you."

In our case, we have three lines: DAV (data valid on data lines), NFRD (not-ready-for-data; true indicates information accepted by listening device), and NDAC (not-data-accepted; true indicates system module ready to accept data). The timing of the handshake appears in Fig. 6-13.

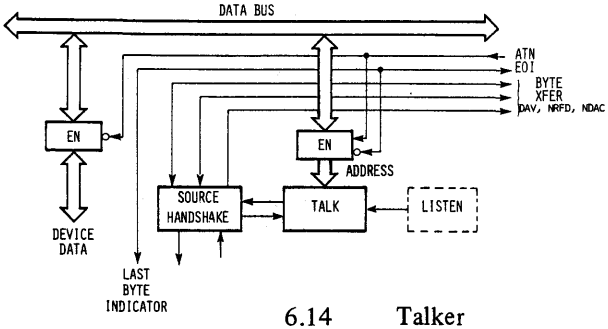


6.13 488 Handshake Timing

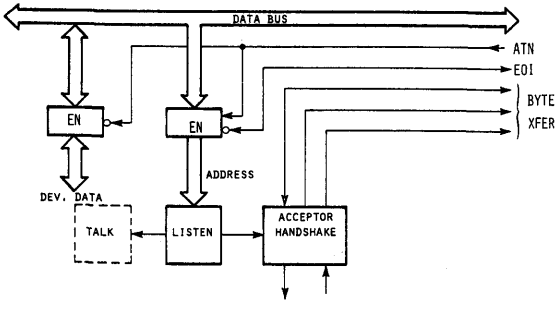
Note how all listening devices must accept the transfer of data before the next transfer is initiated. If it appears complex — it is! Use of this standard

requires complete knowledge of all the states allowed by the protocol.

Some simple examples are presented in Fig. 6-14.



6.14 Talker



6.14½ Listener

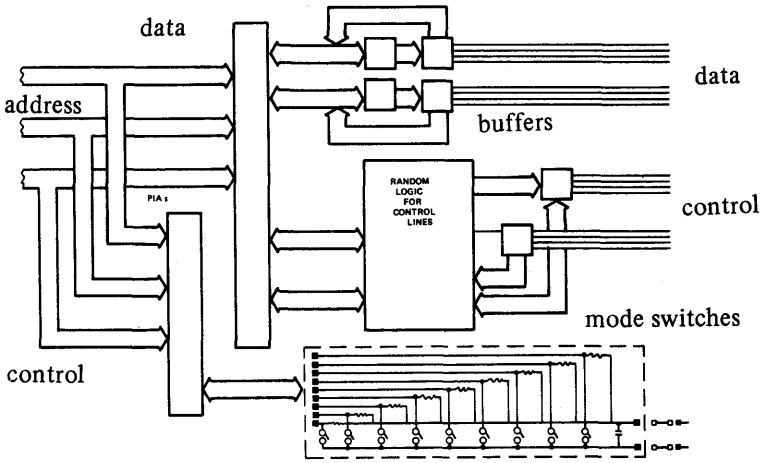
In the "talk" example, the controller sends the address and command-to-talk to the talker, by using ATN and the data-bus. Upon recognizing its address, and the command, the talker will then send information to a listener, via the data-bus, using the handshake signals. When the transfer is finished, the EOI line may be used to indicate the end of the block.

The "listen" example works similarly. The controller sends the address via the data-bus, using the ATN line as before. In this case, the command sent next is for the device to listen to a talker. The transfer of data, byte by byte, is then begun using the data-bus, and handshake signals. The EOI then indicates that the transfer is complete.

In summary, the IEEE-488 bus represents quite an advancement in intelligent data acquisition systems. As more manufacturers produce compatible equipment, the standard will become even more widespread. In fact, the Commodore Business Machines home microcomputer system is equipped

with an IEEE-488 bus interface. This may indicate a new trend in home computing as well as in industry.

The example presented here illustrates how a 6800 system can be interfaced to the 488 bus. The schematic appears in Fig. 6-15, and contains two PIA's, bus transceivers, and some random logic for the control lines. The software uses over 800 bytes of ROM for the program, and 1024 bytes of RAM for buffer space.



6.15 6800 to 488 Bus Interface

CAMAC

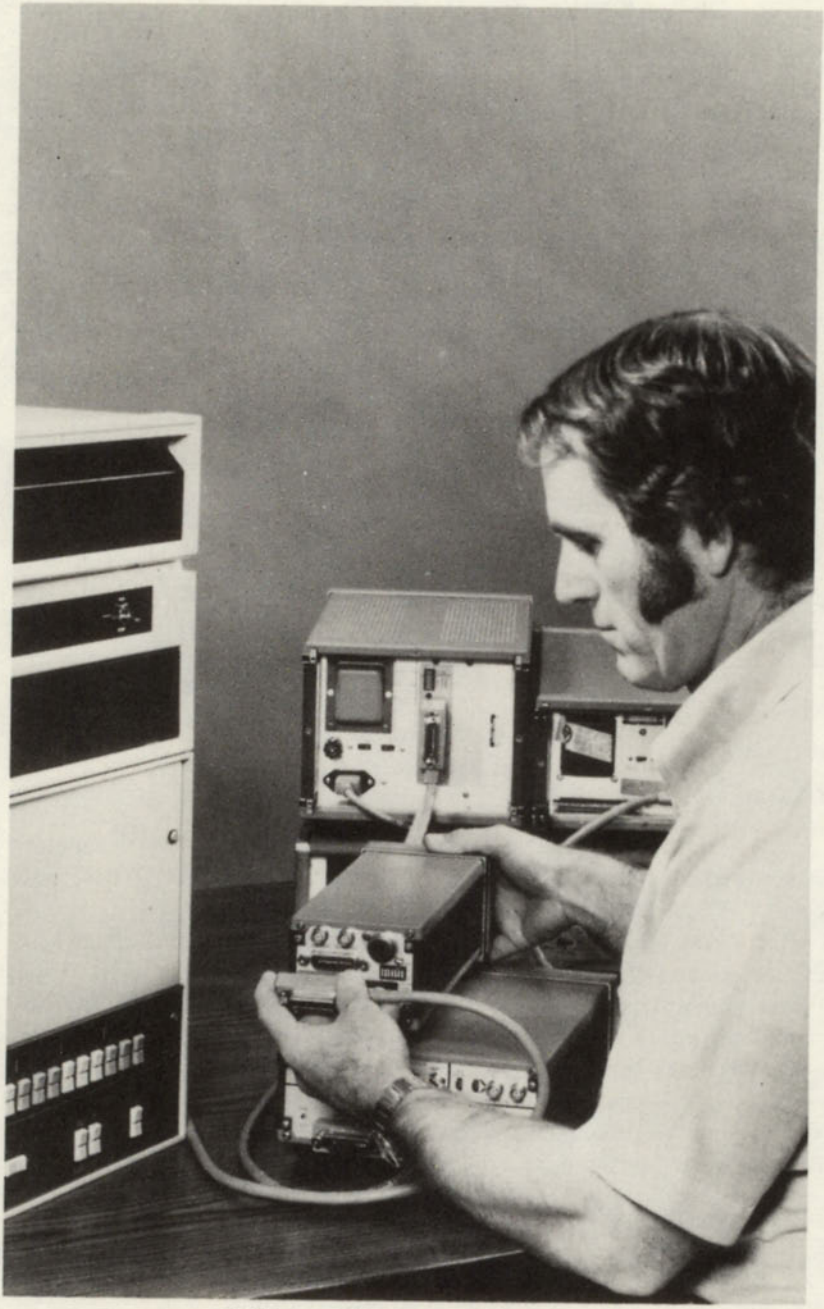
The IEEE-583 standard describes what is known as the "Computer-Automated-Measurement-and-Control-Standard or CAMAC. These also cover CAMAC related standards.

The CAMAC concept covers all areas of instrument interfacing. There is the rack and card-cage standards for physical dimensions, there is the power-supply standards, and there is the "dataway" bus standard. In addition, there are standards for the inter-rack bus: the "parallel highway," and serial inter-rack communications: the "serial highway."

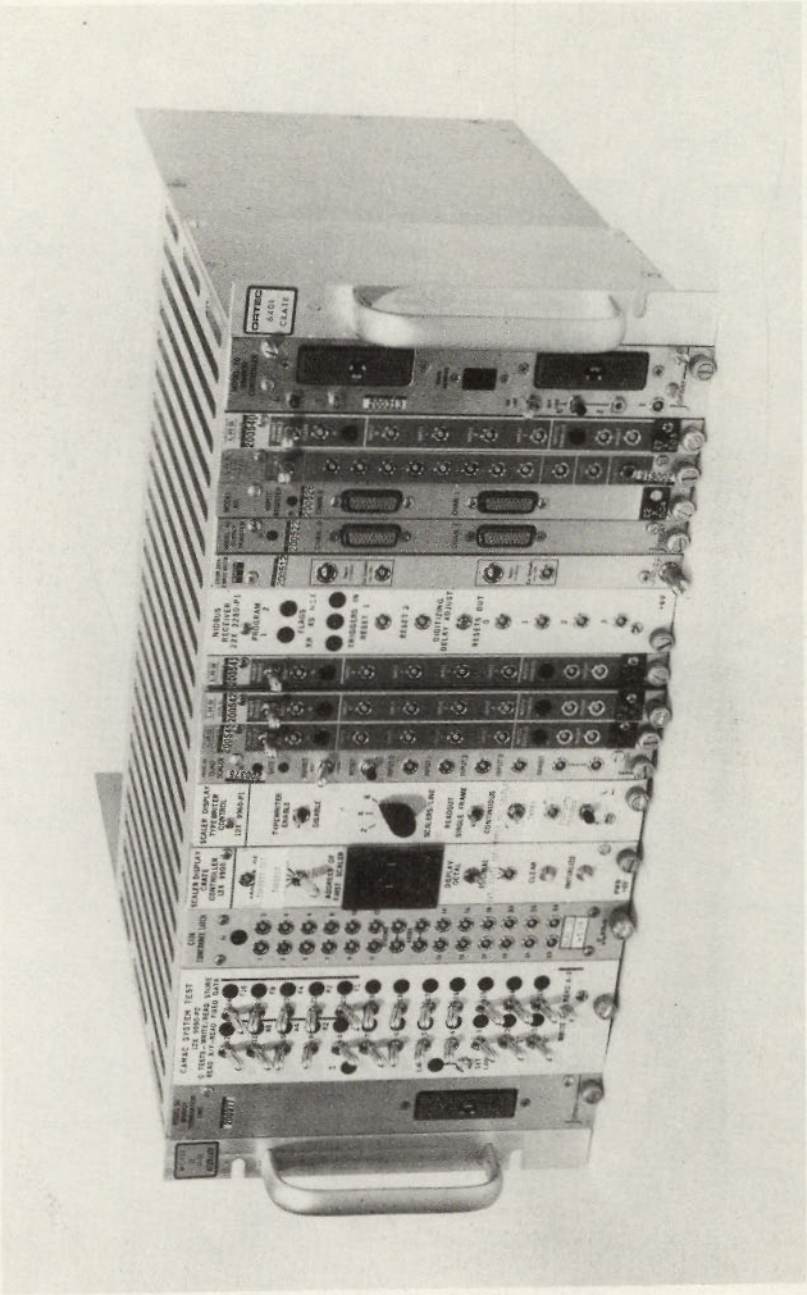
It was developed for the nuclear industry, and all domains of the CAMAC standard contain rigorous specifications. CAMAC systems are required to be built to quite exacting standards.

Physical Dimensions:

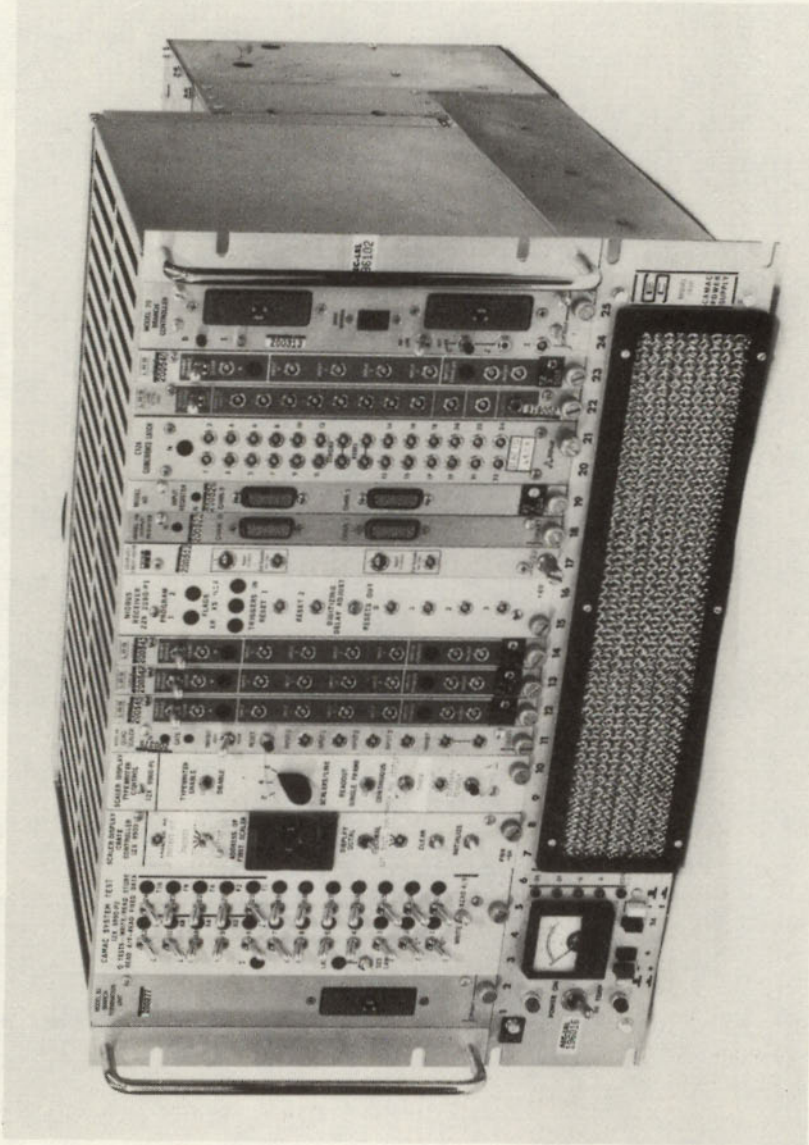
Fig. 6-17 illustrates a CAMAC "crate". The crate is the basic system sub-unit. It contains a controller, and up to 24 peripheral interfaces. The



6.16 HP 1600S Analyzer



6.17 CAMAC CRATE with Modules



6.18 CRATE and Power Supply

size of each card, and the connector types, are all specified.

Power-Supply

The power-supply is a four-voltage type, supplying regulated ±6 and ±24 volts. Stability, regulation, and transient suppression are all covered in the standard. Remember that the power supply, while often ignored, is the basic most important unit in any system. Any flaws in the power supply will show up everywhere else in the system. Thus, CAMAC does something no other standard does: it guarantees the user that the power supply will be the least of all problems in the system. Fig. 6-18 illustrates the crate and power-supply (Pictures are courtesy of Lawrence Berkeley Laboratory).

Dataway

The CAMAC Dataway bus consists of the following lines: three control, five command, five address, twenty-four read, twenty-four write, two timing, and four status. The lines are described in Table 6-19.

The three controls are: initialize, inhibit and clear. These signals are used to put all devices on the dataway into a known state.

The five command lines determine the function to be performed. The 32 possible functions are all defined in the standard. Some functions are for read, write, and status transfers. Others are either reserved for future use, or not defined.

The 24 read and write lines form the data bus. If extra address information is required, the data buses may be used to load further address information. 24 bits allows for simultaneous transfer of three 8-bit bytes for efficient operation. Since some CAMAC systems contain microprocessors, these 24 lines could carry the address and data from the microprocessor. Since data transfers may occur as fast as 10^6 period second, this bus has a greater bandwidth than the other buses so far described.

CAMAC can transfer: 24 bits × 10^6 transfers/second, or 24 million bits/second. This is important in nuclear applications, where large amounts of data must be transferred quickly during each experiment.

The two timing signals provide the information necessary to indicate when data are valid.

The status lines are used to monitor the requests for service to the controller from the peripheral dataway interfaces. There can be 24 separate requests in a single crate.

In summary, the CAMAC standard truly implements a *concept*. It covers all aspects of the communication problem. It includes standards for data-formatting and crate-to-crate communications, as well as software conventions.

A list of Dataway signals available at each of the normal stations 1 through 24 of a 25-station CAMAC crate

Title	Designation	Use in Module
Common Control Signals		Sets registers or control functions in a module to an initial state, particularly when power turned on.
Initialize	Z	
Inhibit	I	Disables features for duration of signal.
Clear	C	Clears registers, or resets flip-flops.
Commands, addressed Function codes	F1,2,4,8,16	Carried on Dataway in binary code. Defines the function to be performed in a module during command operations.
Addressing Station number	N	Selects the module. There is an individual line from crate controller to each station.
Subaddress	A1,2,4,8	Also binary coded. Selects a location, within the module, to which the command is directed. There are 16 possible subaddresses.
Data Read bus	R1-R24	Transmits digital information from module to Crate Controller. Format is bit-parallel words, 24 bits maximum.
Write bus	W1-W24	Transmits digital information from Crate Controller to module. Format is same as for Read bus.
Timing Strobe 1 and Strobe 2	S1,S2	These strobes are generated by CC during every Dataway operation. Used by modules for timing acceptance of data or execution of features of an operation.
Status Look-at-Me	L	A signal from module to Crate Controller indicating request for service or attention. There is an individual line from each module to control station.
Q-Response	Q	A one-bit reply by module to certain commands issued by Crate Controller.
Command Accepted	X	Indicates the ability of a module to execute the current command operation.
Busy	B	Indicates a Dataway operation is in progress.

6.19 Dataway Signals

SERIAL STANDARDS:

Serial transmission requires only one or two wires to carry all necessary signals between modules or systems. In order to transmit address, data, and control, they must be sent bit by bit.

Described here are the RS232C, RS422 and 423, asynchronous and synchronous communication standards. In addition, data standards such as ASCII and SDLC will be covered.

EIA-RS232C

The Electronics Industry Association (EIA) standard RS232C covers the electrical specifications for bit-serial transmission, as well as the physical specifications. It defines the handshaking signals used to control standard telephone connection equipment, and standard modulator-demodulators (modems).

Electrically, the standard uses nominal plus and minus 12 volt pulses to effect information transfer. The RS232C standard specifies a 25-pin connector with the signals shown in Table 6-20. All 25 lines are specified, but only the first fifteen in the Table will be described.

-	GROUND	
-	XMIT DATA	(TO COM EQUIPMENT)
-	REC DATA	(FROM COM)
-	REQUEST TO SENT	(TO COM)
-	CLEAR TO SEND	(FROM COM)
-	DATA SET READY	(FROM COM)
-	DATA TERMINAL READY	(TO COM)
-	RING INDICATOR	(FROM COM)
-	RECEIVED LINE SIGNAL DETECTOR	(FROM COM)
-	SIGNAL QUALITY DETECTOR	(FROM COM)
-	DATA RATE SELECTOR	(TO COM)
-	DATA RATE SELECTOR	(FROM COM)
-	TRANSMITTER TIMING	(TO COM)
-	TRANSMITTER TIMING	(FROM COM)
-	RECEIVER TIMING	(FROM COM)
+	SECONDARY DATA AND REQUESTS	

6.20 EIA RS232C Signals

The secondary lines provide the data and control paths for a second serial channel running at a much lower speed than the primary channel. The second channel is then identical to the first, except for speed. The second channel is hardly ever used, but when it is, it contains control information for the modems connected at each end of the communications line.

The main signal lines are transmit data and receive data. These lines are

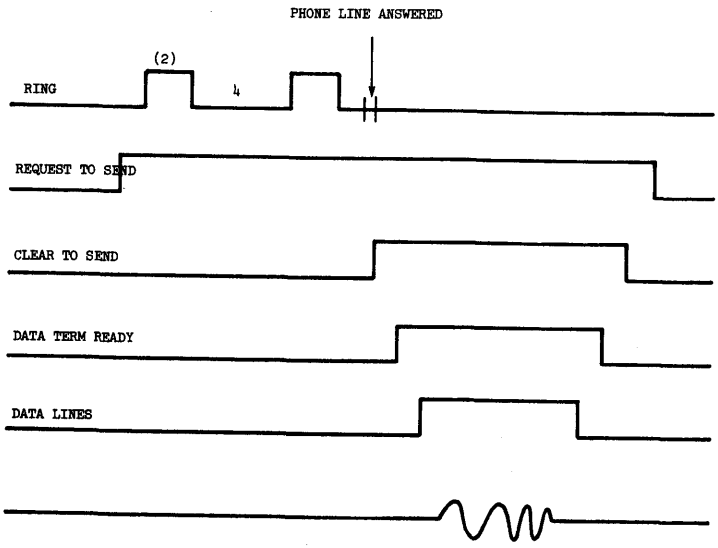
used to send serial information between the two systems. The bit rate may be any one of the following standard rates:

19,200	1,200	110
9,600	600	75
4,800	300	50
2,400	150	

Other rates are also occasionally used. The teletypewriter terminals run at 110, 150, or 300 bits/second. CRT terminals typically use any of the speeds above 1,200.

Quite often, serial data are transmitted over telephone voice-grade lines. The data must first be modulated, so that they may be transmitted. For bit rates of less than 300, the method of modulation is known as FSK: frequency-shift-keying. The "marking" or logic "1" condition is represented by a tone of given frequency, and the "spacing" or logic "0" condition is represented by a second, different, frequency. Bit rates above 300 must use phase-modulation techniques, due to the lack of available bandwidth. Quite often, voice-grade lines are too noisy for high-rate communications. More expensive data-grade lines must be used.

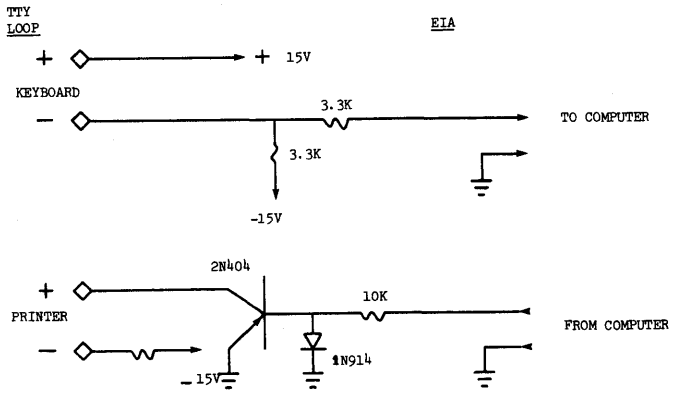
The other signals are used to indicate the status of the modulator-demodulator (modem) communications link. Signals such as: "request-to-send", "clear-to-send", "data-set-ready", "data-terminal-ready", are used to control the modem link.



6.21 EIA RS232C Modem Handshake

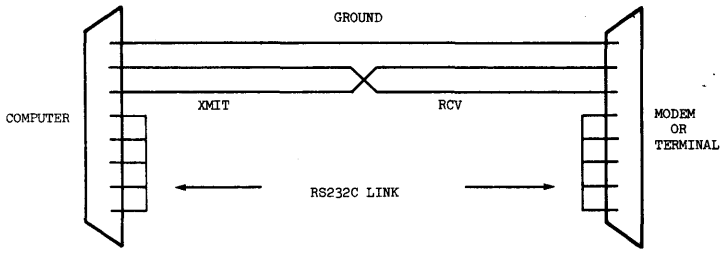
The timing in Fig. 6-21 is meant to show a typical communications transaction. Note how the signals between the modem (communications equipment) and the computer (or terminal) implement a similar kind of handshake to that used in most buses—especially the IEEE-488. The difference, in this case, is that the handshake is used only at the beginning, and end, of a block of serial data.

RS232C is popular, as almost all dial-up time-share systems use this standard in their communication subsystems. A similar standard is *current-loop*. This is used in the mechanical teletypewriters. A good thing to do is to convert all loop devices to EIA-RS232C via a loop-to-EIA converter. In this way, all communications become standardized. A loop-to-EIA converter for a teletype is shown in Fig. 6-22. Also useful is what is known as *auto-loop back*, shown in Fig. 6-23. This is where the computer, terminal, or modem, does not have the full standard implemented. The jumpers specified will usually allow the devices to believe that all conditions are "OK" for data to pass.



6.22

PINS 4, 5, 8, 20 TOGETHER
ON EACH PLUG



6.23 Auto Loop Back Connection

RS422 and 423:

RS232C transmits signals as single-ended voltages. The "mark" or "space" condition is represented by the voltage between two wires. Thus, the transmit path has two wires, and the receive path has two wires. The advantage is that the path may be physically longer between devices, due to the noise immunity of a differential channel. In the same way, the data rate may be higher, due to reduced noise effects.

CHARACTERISTIC	RS232	RS422	RS423
MAXIMUM LINE LENGTH	100 ft.	5000 ft.	5000 ft.
MAXIMUM BITS/SEC.	2×10^4	10^6	10^5
DATA "1" = MARKING DATA "0" = SPACING	-1.5V -36V +1.5V +36V	VA VB VA VB	VA = - VB = +
SHORT CIRCUIT	100	100	100
POWER-OFF LEAKAGE, MAXIMUM VOLT APPLIED TO UNPOWERED	300	100 A	100 A
RECEIVER INPUT, MINIMUM	1.5V (single-ended)	100 mV (differential)	100 mV (differential)

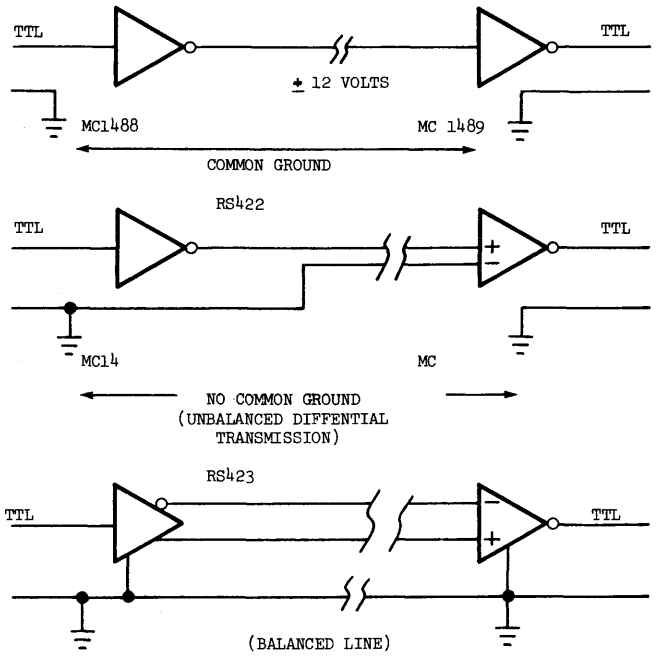
6.24 Comparison of RS232C, RS422, and RS423

Fig. 6-24 illustrates the difference between the three standards. Fig. 6-25 shows the types of drivers and receivers used for the lines. RS422 and 423 are not used often due to the already widespread use of RS232C and the infrequent need for such high data rates and line lengths.

Of course, the data sent back and forth may be formatted in many ways. The topics of asynchronous, and synchronous data transmission and standards for information exchange will be covered next.

ASYNCHRONOUS COMMUNICATION

When data are sent in bursts of equal duration, without clock information, they are being sent *asynchronously,* without a clock. When data are

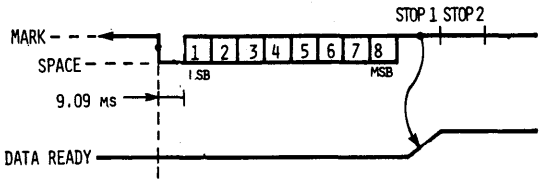


6.25 Drivers for RS422 and RS423

sent with synchronizing character codes imbedded within the blocks, they are being sent *synchronously:* with a clock.

The most common asynchronous data structure, shown in Fig. 6-26, is used by most CRT's and teletypewriters. It consists of the 10 (or 11) bits described in Chapter 4. The start bit, eight data, and one or two stop bits, comprise a *character.* The most popular standards for character codes are ASCII and EBCDIC.

ASCII stands for "American Standard Code for Information Exchange". It uses seven bits to encode 128 possible characters. An eighth bit



6.26 Serial Data Format

may be used for parity. Note that many codes are used for controlling the functions of a data link. Codes such as: "Begin Text", "End of Text", etc. are used to format and transfer blocks of characters.

EBCDIC is similar except that the 128 codes are encoded differently. Simple code-conversion ROM's can convert ASCII to EBCDIC, and EBCDIC to ASCII. Such a ROM has 8 inputs: seven address-lines for the data input, and one address-line to specify the conversion (either ASCII to EBCDIC or EBCDIC to ASCII). It has seven outputs for the converted character. The size of this ROM would be 256 bytes by 7 bits/byte. This is a small ROM by today's standards and it is relatively inexpensive to program or purchase.

Who uses EBCDIC? IBM. Who uses ASCII? Practically everyone else. Other codes exist, such as the five-bit Baudot code (obsolete today) which can also be converted also by a look-up ROM.

Naturally, a program may be also be used to convert from one code to another.

BIT NUMBERS								0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_7	b_6	b_5	b_4	b_3	b_2	b_1	HEX 1 / HEX ø	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	"	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	' 6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	11	VT	ESC	+	;	K	[	k	{
			1	1	0	0	12	FF	FS	,	<	L	\	l	;
			1	1	0	1	13	CR	GS	-	=	M	]	m	}
			1	1	1	0	14	SO	RS	.	>	N	^	n	~
			1	1	1	1	15	SI	US	/	?	O	□	o	DEL

6.27 ASCII Code Table

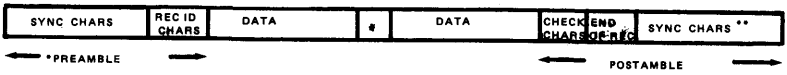
Decimal	EBCDIC 8 bit	Hexadecimal	EBCDIC 3 bit	Hexadecimal	EBCDIC 8 bit	Hexadecimal	EBCDIC 8 bit	Hexadecimal	EBCDIC 8 bit
0		20		5A	!	90	- /	C6	F
1		21		5B	$	91		C7	G
2		22		5C	*	92		C8	H
3		23		5D	)	93		C9	I
4		24		5E	;	94		CA	
5		25		5F	¬	95		CB	
6		26		60	-	96		CC	
7		27		61	/	97		CD	
8		28		62		98		CE	
9		29		63		99		CF	
A		2A		64		9A		D0	}
B		2B		65		9B		D1	J
C		2C		66		9C		D2	K
D		2D		67		9D		D3	L
E		2E		68		9E		D4	M
F		2F		69		9F		D5	N
10		30		6A	¦	A0		D6	O
11		31		6B	,	A1	~	D7	P
12		32		6C	%	A2	s	D8	Q
13		33		6D	_	A3	t	D9	R
14		34		6E	>	A4	u	DA	
15		35		6F	?	A5	v	DB	
16		36		70		A6	w	DC	
17		37		71		A7	x	DD	
18		38		72		A8	y	DE	
19		39		73		A9	z	DF	
1A		3A		74		AA		E0	\
1B		3B		75		AB		E1	
1C		3C		76		AC		E2	S
1D		3D		77		AD		E3	T
1E		3E		78		AE		E4	U
1F		3F	blank	79	`	AF		E5	V
20		40		7A	:	B0		E6	W
21		41		7B	#	B1		E7	X
22		42		7C	@	B2		E8	Y
23		43		7D	'	B3		E9	Z
24		44		7E	=	B4		EA	
25		45		7F	"	B5		EB	
26		46		80		B6		EC	
27		47		81	a	B7		ED	
28		48		82	b	B8		EE	
29		49		83	c	B9		EF	
2A		4A		84	d	BA		F0	0
2B		4B		85	e	BB		F1	1
2C		4C		86	f	BC		F2	2
				87	g	BD		F3	3
				88	h	BE		F4	4
				89	i	BF		F5	5
				8A		C0	{	F6	6
				8B		C1	A	F7	7
				8C		C2	B	F8	8
				8D		C3	C	F9	9
				8E		C4	D	FA	
				8F		C5	E	FB	
				8E		C6	F	FC	
				8F		C7	G	FD	
				90		C8	H	FE	
				91		C9	I	FF	
				92		CA			
				93		CB			
				94		CC			
				95		CD			
				96		CE			
				97		CF			
				98		D0	}		
				99	j	D1	J		
				9A	k	D2	K		
				9B	l	D3	L		
				9C	m	D4	M		
				9D	n	D5	N		
				9E	o	D6	O		
				9F	p	D7	P		
				A0	q	D8	Q		
				A1	r	D9	R		
				A2		DA			
				A3	~	DB			
				A4	s	DC			
				A5	t	DD			
				A6	u	DE			
				A7	v	DF			
				A8	w	E0	\		
				A9	x	E1			
				AA	y	E2	S		
				AB	z	E3	T		
				AC		E4	U		
				AD		E5	V		
				AE		E6	W		
				AF		E7	X		
				B0		E8	Y		
				B1		E9	Z		
				B2		EA			
				B3		EB			
				B4		EC			
				B5		ED			
						EE			
						EF			
						F0	0		
						F1	1		
						F2	2		
						F3	3		
						F4	4		
						F5	5		
						F6	6		
						F7	7		
						F8	8		
						F9	9		
						FA			
						FB			
						FC			
						FD			
						FE			
						FF			

6.28 EBCDIC CODE TABLE

SYNCHRONOUS COMMUNICATION

An asynchronous transmission format contains at least two extra bits per character: start and stop. When data are sent as a continuous stream of bits, with no start or stop, the receiver might loose its timing, and scramble the incoming data. To prevent this, *synchronizing characters* are sent every hundred or so bytes. There exists the necessary logic, at the receiving end, to resynchronize the decoding circuitry, often enought to remain locked in. Using this method, known as *synchronous communication,* there will only be eight extra bits, for every 800 bits. This is 1% extra data versus 20% extra data in the asynchronous case.

Various *protocols* are used for synchronous data links. A simple one that was described earlier is the data-format for the floppy-disk, or the KIM-1 tape-recording standard. The format for the disk is similar to most synchronous formats. In general, the transmission begins with a few synchronizing characters, continues with long data blocks interspersed with synchronizing characters, and ends with a parity or check-sum character, plus end-of-record character. Upon receiving a block, if the check-character does not agree with the data, the receiving end will ask the transmitter to retransmit or try again.



Synchronous Data Format

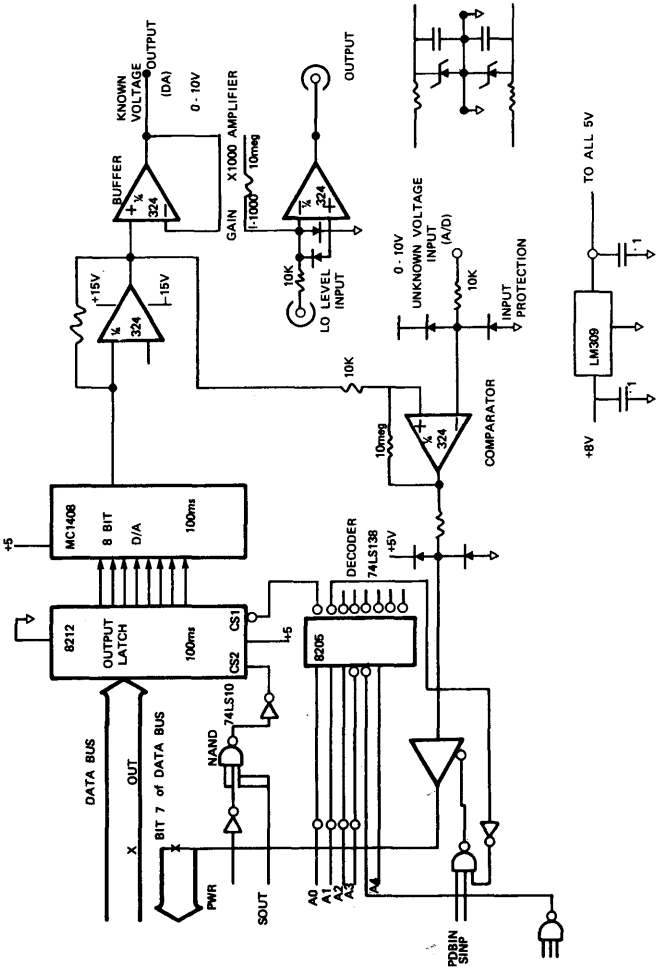
*SYNC AS OFTEN AS NEEDED MAY NOT BE NECESSARY

**MAY NOT BE NECESSARY

IBM Bi-sync, SDLC, and other protocols differ in complexity and communications control abilities. Basically, they are all synchronous formats. An important point of synchronous communications is that the error checking schemes are much more complex than in asynchronous communications. Since synchronous communication saves as to the number of bits transmitted, extra bits are sometimes added to each block so that, not only errors can be detected, but they can be corrected. This means retransmission may not be necessary.

A CASE-STUDY: INEXPENSIVE ANALOG BOARD FOR S100 BUS:

This circuit in Fig. 6-29 shows a digital-to-analog converter with analog-to-digital conversion capability. The circuit has 6 integrated circuits: one



6.29 S100 A/D, D/A Board

triple three-input nand-gate, one 74LS138 decoder, one 74LS125 tri-state bus-driver, one 8212 octal-latch, one MC1408 D/A converter, and one LM324 quad operational amplifier. With these components, an S100 bus analog measurement assembly has been designed.

Features of this module are:

— S100 bus compatible, only 1 LSTTL load per bus-line
— 8-bit resolution for both D/A and A/D
— D/A conversion in 20us
— A/D conversion in 1ms
— 0-10 volt input and output with extra 1 to 1000 gain stage for low-level inputs.

The circuit will be described part by part, to explain the function of each component.

The Hardware:

The output data-bus which performs all data transfers to memory or output ports is connected to an 8212 latch. Each bit is loaded by an input of the latch. Each input represents ⅔ of a low-power-Schottky input load.

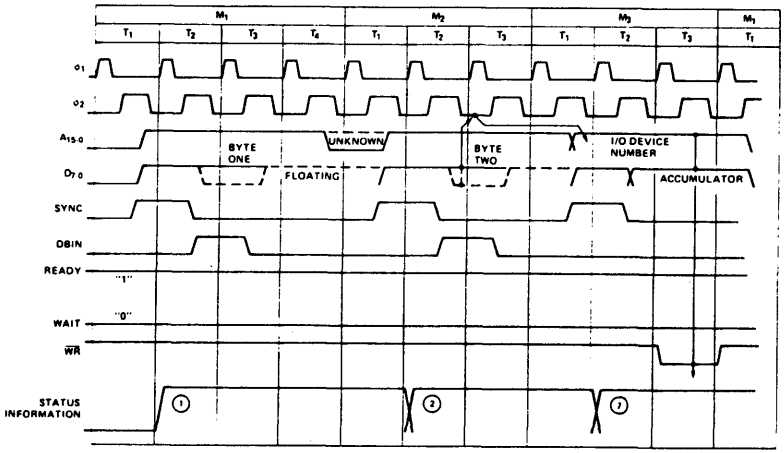
The 74LS138 decoder, along with the 74LS10 and 74LS04 decodes the output to port "F8" (hexadecimal). The address is partially decoded by ⅓ of the 74LS10 so that bits A7, A6, A5 must all be "1's" to enable the 74LS138 decoder. Then the bits A0, A1, A2, A3, and A4 are decoded by the 74LS138. The first output represents "F0" on the low eight address-bits. This enables one of the chip-selects on the 8212 latch.

The other chip-select is driven by the condition $P\overline{WR}$ false and SOUT true. This is done by inverting $P\overline{WR}$ and "NANDing" it with SOUT. The output of the NAND is passed through an inverter to the second chip-select of the 8212.

This way, the output data-bus is latched into the 8212 latch when the address is "F0", and the control signals indicate an output instruction is being executed. The timing is shown in Fig. 6-30.

The latched data is sent to a MC1408 digital-to-analog converter. At the output of the converter, a current proportional to the binary input is present. In order to convert it to a voltage, a current-to-voltage converter circuit is used. It is implemented with ¼ of the LM324 quad op-amp.

The output is now a voltage between 0 and 10 volts for inputs between "00" and "FF" (hexadecimal). The next op-amp, in the LM324, is used to buffer the output so that an output may be driven, without affecting the comparator section.



6.30 S100 Output Write Cycle Timing

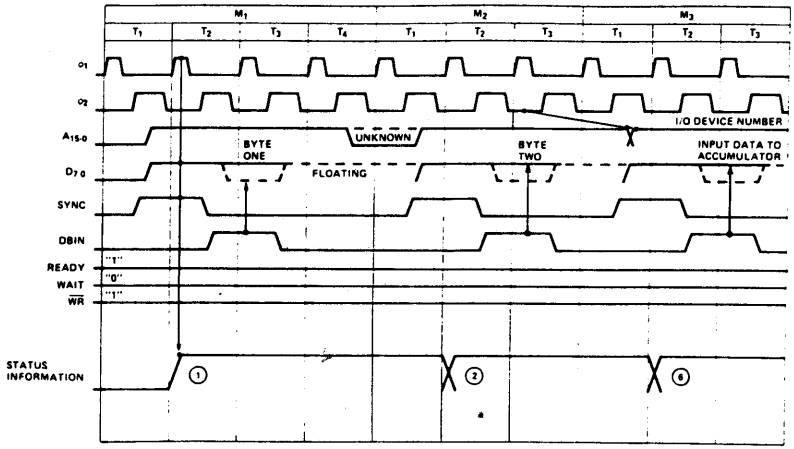
The third op-amp is used as a comparator for the analog-to-digital conversion. The op-amp compares the unknown input with the output of the D/A. If the unknown signal is too small, a variable-gain amplifier, implemented with the fourth op-amp is used to boost the signal. Note the protection diodes, that are used, so that no damage will be caused to the inputs, as long as voltage transients there are kept below 100 volts.

The output of the comparator is clamped to TTL levels by the resistor-diode combination, so the 74LS125 tri-state driver can be driven. The driver is enabled by an input command, and the address "F9" (hexadecimal). The decoding is done similarly to the output port, except that the second output of the 74LS138 is used to decode the address "F1." In addition, the control lines PDBIN and SINP are "ANDed" with the address, to enable the driver to bit 7 of the data-bus.

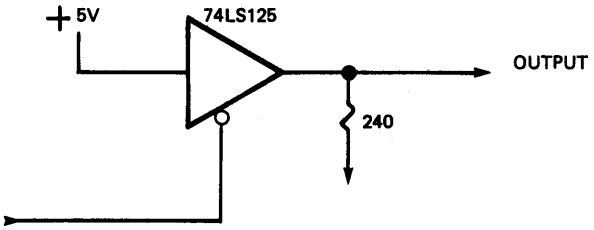
By driving bit 7, we can input from port "F1", rotate bit 7 into the carry bit, and test the carry, to see if we are above, or below, the unknown input voltage. Outputting a new value to port "F0", and checking "F7" again, will form the basis of our analog-to-digital converter. Timing for an input operation appears in Fig. 6-31.

Power is supplied by the +5 volt voltage-regulator for all Vcc pins, and the Zener-diode regulators for the + and − 15 volts voltages, required for the op-amp package.

Note that three of the bus drivers were used as inverters. Fig. 6-32 shows how this is done.



6.31 S100 Input Read Cycle Timing



6.32 Use of 74LSI25 As An Inverter

When the input is low, the driver is enabled, and the output will be pulled up to a logic "1". When the input is high, the driver is disabled and the 240-ohm resistor pulls the output to a logic "0". We could have used a hex inverter for these functions, but it would have increased the parts count.

The Software

For digital-to-analog conversion, the binary value to be converted is output to port "F0". Each step represents 10.0 volts/256 = 39.0625 millivolts. This means that if you want 2.5 volts out, the binary number is:

$$\frac{\text{vout}}{39.0625 \times 10^{-3}} = \text{Num}_{10} \xrightarrow[\text{to binary}]{\text{convert}} \text{Bin}_2$$

or

$$\frac{2.5}{39.0625 \times 10^3} = 64_{10} \longrightarrow 0100\ 0000_2$$

or 40 hexadecimal. 80 hex will be 5 volts, because the converter is linear. In software we need:

```
MOV A, M  : get value from memory to output
OUT F0H   : output
```

QUESTION: What is the highest frequency we could generate with this converter?

ANSWER: Since the sampling theorem states we need to sample, or, alternatively, to output a value, at least at twice the rate of the highest frequency—we would have:

$$\frac{1}{\text{conversion}} \cdot \tfrac{1}{2} = \text{f max}$$

or

$$\frac{1}{20 \times 10^{-6}} \cdot \tfrac{1}{2} = 250\ \text{Khz}$$

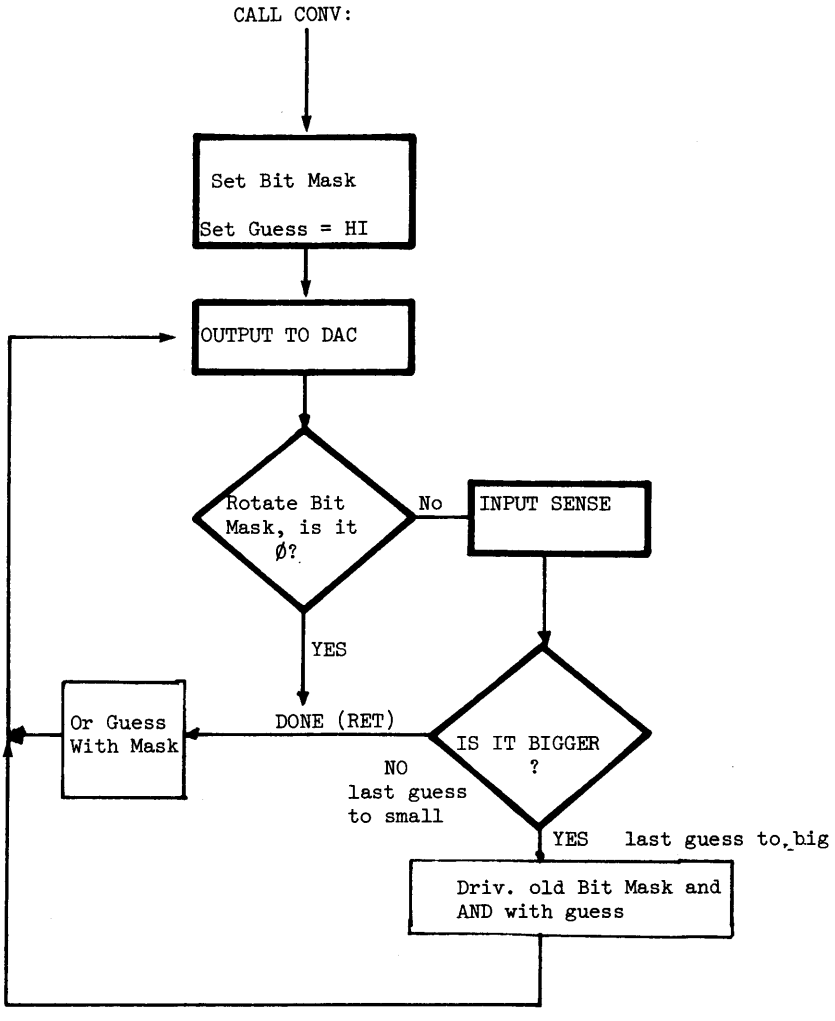
In practice, our program will not be able to fetch information fast enough to use this bandwidth; but, we will be able to generate music or voice range sounds.

Analog-to-Digital Conversion

To perform the A/D conversion, we need to implement the successive-approximation algorithm in software. Another technique which can be used is the counting conversion technique. Both will be discussed.

Successive-approximation was presented in Chapter 5. In order to code this into an 8080 assembly-language subroutine, we need to examine the flowchart of Fig. 6-33.

A program that will perform this conversion appears on Fig. 6-34. Note how this program uses the "NOP" and "CMP E,M" instructions to balance the timing of the "JC" instruction. This is done so that the conversion will take the same amount of time to execute through either path of the flowchart.



6.33 Successive Approximation Flowchart

The conversion time is 373.5 uS according to the instruction execution times, without a wait state. We can only sample every 380 uS approximately.

QUESTION: What is the highest frequency we can sample?

```
            MVI     D, 80H :  temp mask in D
            MVI     B, 80H :  mask in B
            MVI     C, 80H :  guess in C
GUESSOUT:   MOV     A,C

            OUT     DAC    :  OUTPUT GUESS

            MOV     A,B
            RRC
            RC             :  done if carry bit
                              set
            MOV     B,A

            IN      SENSE
            RLC
            JC      bigger
            MOV     A,D
            RRC
            MOV     D,A
            MOV     A,C
            ORA     B
            MOV     C,A
            CMP     E,M
            JMP     GUESSOUT

BIGGER:     MOV     A,D
            CMP
            AND     C
            MOV     A,D
            RRC
            MOV     D,A
            NOP
            JMP     GUESSOUT
```

6.34 Program For A/D Conversion

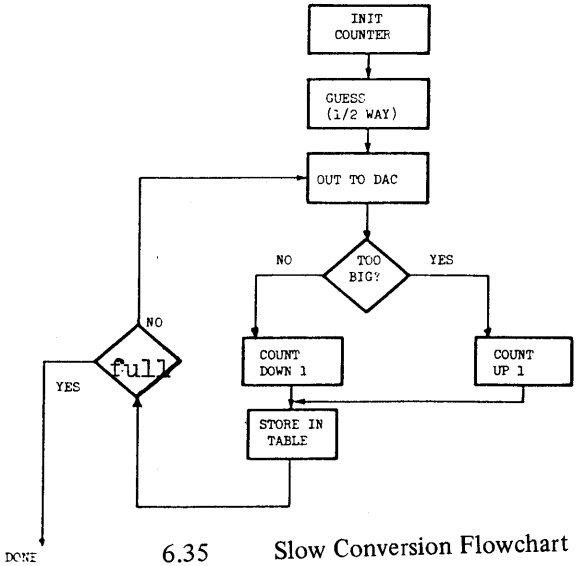
ANSWER: Again, according to the sampling theorem it is:

$$\frac{1}{\text{conversion time}} \cdot \tfrac{1}{2} = \text{f max}$$

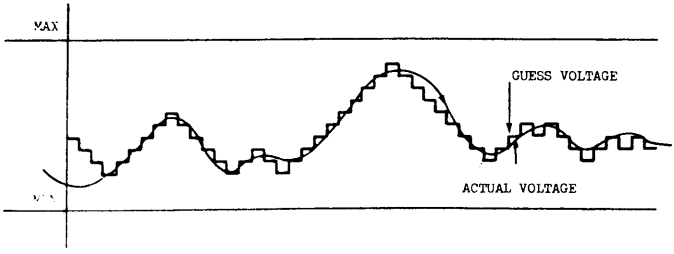
$$\frac{1}{380 \times 10^{-6}} \cdot \tfrac{1}{2} = 1316 \text{ HZ}$$

This means our converter can just barely go fast enough to digitize speech.

If we know our input is a slowly-varying waveform, we can convert in a simple fashion. The flowchart appears on Fig. 6-35.



6.35 Slow Conversion Flowchart



The routine, as coded in Fig. 6-36 will place a new guess in memory every 45 uS. As soon as 256 samples have been taken, the program will exit. Note how instructions to store guesses, and "check for the end-of-table",

```
        LXI H, TABLE START
        MVZ A, 80H
LOOP:   OUT DAC     :  OUT GUESS
        MOV M,A     :  STORE GUESS
        INX  H      :  ADVANCE TO NEXT ENTRY
                           POINT
        MOV B,A
        MOV A,L
        CPI
        RZ
        MOV A,B
        IN SENSE
        RRC
        JC   BIGGER
        INR A
        CMP E,M
        JMP  LOOP

BIGGER  DCR A
        NOP
        JMP  LOOP
```

6.36 Software For Slow Guess Table Converter

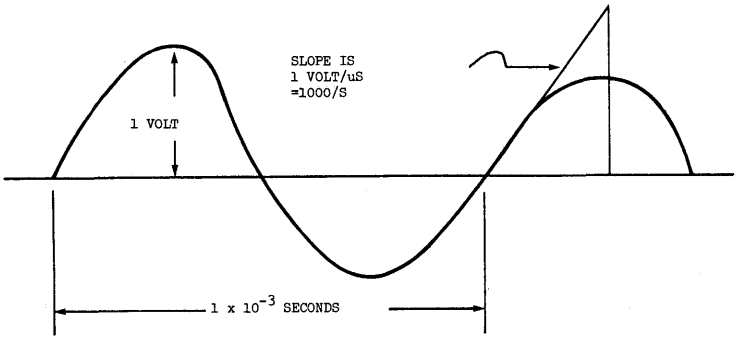
are placed before the "IN" instruction, to allow for the settling time of the comparator.

This scheme does not really convert to a number, for each sample: it merely *tries to track the slope of the input-signal.* This means that, as long as the input changes no more than:

$$\frac{39.0625 \times 10^{-6} \text{ volts}}{45.0 \times 10^{-6} \text{ seconds}} = .86 \text{ volts/second},$$

the numbers in the table will be accurate. How fast does a 1KHz sine wave change at its steepest slope? In Fig. 6-37 we find that it is 1000 volts/second:

So, we are limited by this method to sampling low frequencies, much below 1 Hz.



6.37 1000 volts/second 1KHz sine wave

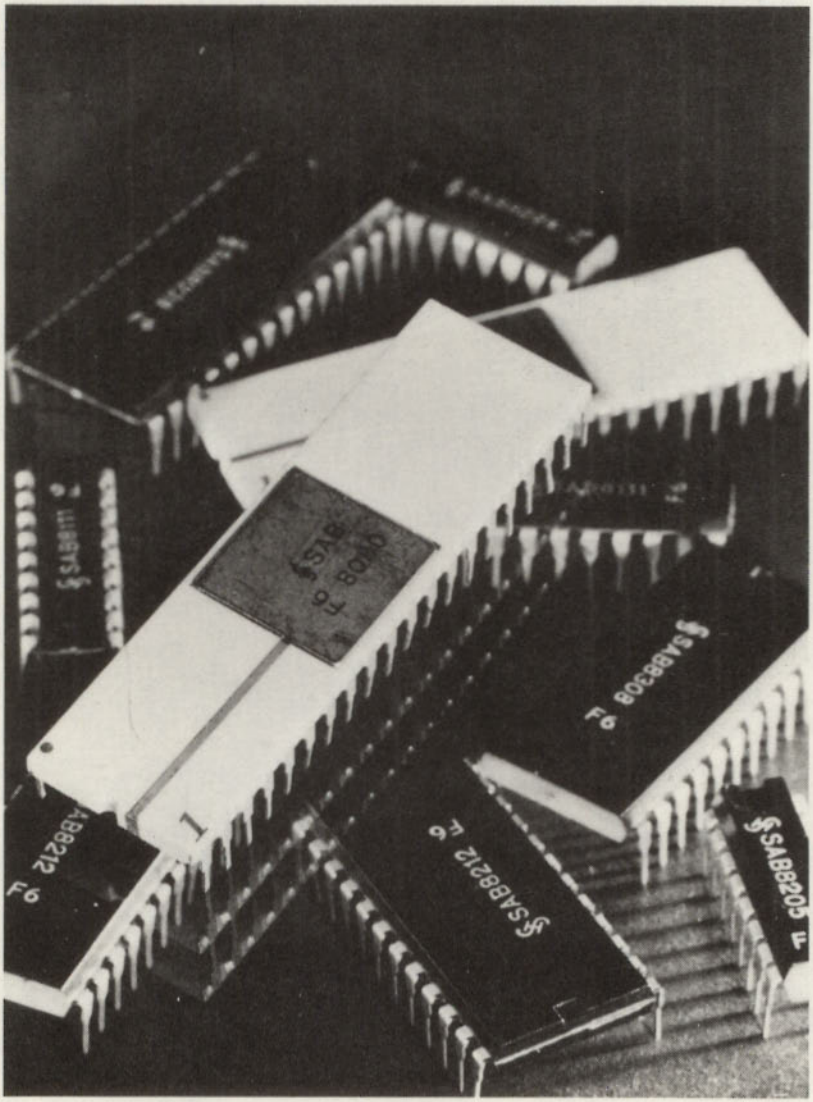
SUMMARY

We have designed an analog data-collection and control board. It was designed to be connected to the S100 bus. Software was written to use the features of this D/A and A/D-converter.

The buses and standards described are intended to make the job of interfacing easier. To plug the device into a system with no extra work is every interface designer's dream. We have seen how the many users of the S100, CAMAC, IEEE-488 and EIA-RS232C standards create a large need for standard-compatible devices, modules, and systems. If at all possible—*stay within a standard.* The design will be easier and your time may be spent on the harder problems.

Parallel and serial bus standards, methods of communication between

modules, and an actual bus-interface example were presented. The S100 bus is the most popular parallel bus used now, with over 600 different types of compatible boards being produced. The serial RS232C standard is the most popular standard for data communications, and versions of data formatting are used, with modems, to store and retrieve data from cassettes and cartridges, as described in Chapter 4.

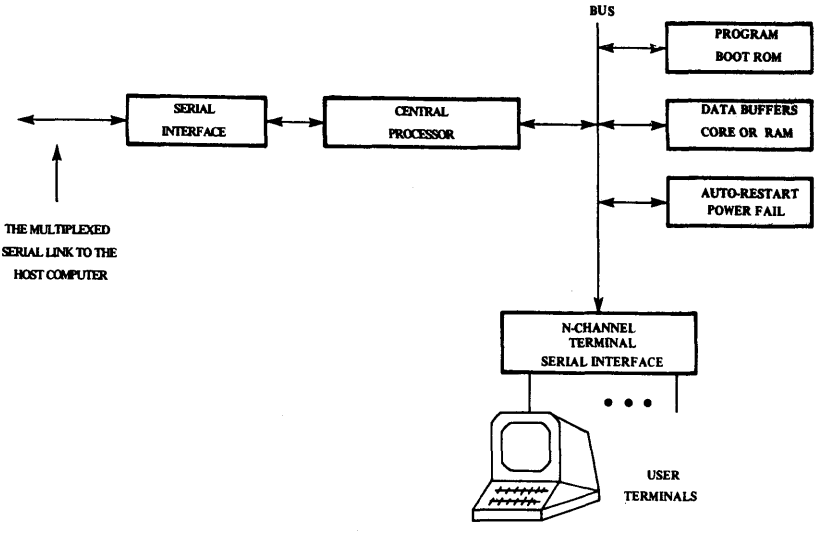

§SAB8212 P
§SAB8212 P
§SAB8308 P
§SAB8205 P

CHAPTER 7
THE MULTIPLEXER — A CASE STUDY

INTRODUCTION

This system is intended to concentrate 32 EIA RS232C compatible terminals onto a single two-way high-speed transmission line. Each terminal has buffered output and character-by-character input. Thus, the host computer can spend less time executing the multiplexing task.

Designed for a PDP 11/70 , the system is also applicable, with only code changes in the host machine, to almost any host computer. The cost of providing this function is $50 per channel, as compared to usually around $250 per channel. The system is also cost-effective in clusters of less than 32 terminals.



7.0 System Overview

The system uses the 8080 microprocessor, 8251 USRT, 8259 interrupt controller, and other components in the 8080 family. The system has no modem-control features, as it was intended to be at the sight of the terminals, saving even more money in man-hours of time, and cost of wire for connection. This does not even include the cost-benefit of fewer telephone lines and modems.

THE SPECIFICATIONS

The ability to connect a large umber of terminals to a time-sharing facility always presents the engineer with a number of problems. Most have to do with the interconnection headaches of modems, telephone wiring, patchboards for testing, and internal machine interfacing.

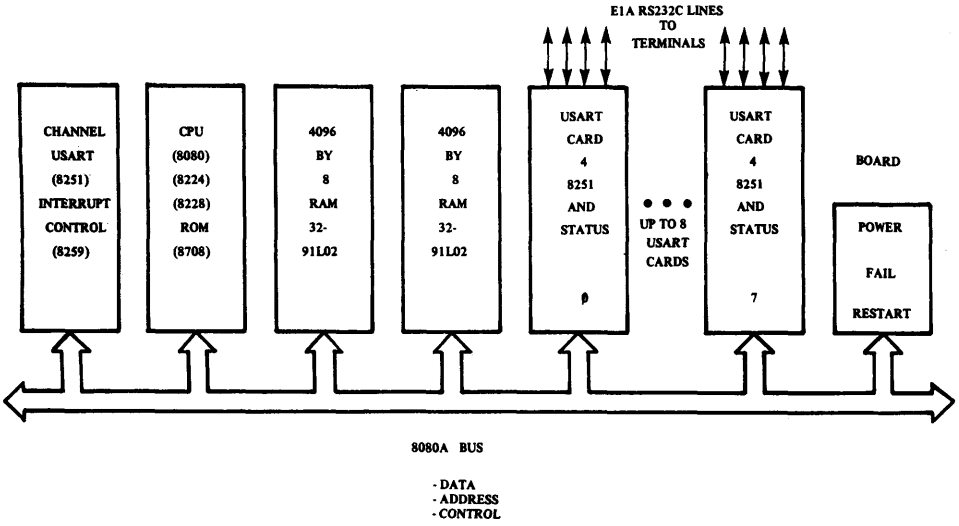
Remotely-located concentrators would eliminate many problems. The new problem: cost. The design goal here is to service 32 terminals at an input rate never exceeding 30 characters-per-second, and an output rate as fast as possible. Given that the 8080A could execute roughly 300 instructions in the time between characters at 9600 baud, if it were to service 32 terminals on input, it would have to have less than 300 instructions in the polling loop for the terminals. Any time left over would be used for output. The code would have to be thought out byte-by-byte, with all coding being carefully optimized. A prototype was built, under the assumption that it could at least service 16 terminals in a degraded mode.

The typical statistics of our input was a maximum of 150 baud for any second, and a rate of 50 baud, for all 32 terminals combined. Thus, when completed, the multiplexer could handle a maximum of 150 baud on all 32 at once, or a maximum of 300 baud on one. The output was a minimum of 300 baud for all 32 at once, and a typical 6000 baud, when there was a specific demand from a single user.

ARCHITECTURE

The architectual block diagram is presented in Fig. 7-1. Each terminal has its own USART, because each needs a dedicated serial interface. The USART's are grouped into fours, and then placed onto cards, which are on the 8080A system bus. There are 8,192 bytes of RAM for data storage, and 1,024 bytes of EAROM for program, in the system. Lastly, there is an interrupt-controller and high-speed-channel card, which is on the bus.

Each terminal, through its USART, has a 128-character buffer associated with it, for buffering output to the terminal. This takes 4,096 bytes of the available RAM. The terminals-to-host queue is 256 characters long. These lengths were chosen to optimize the communication-channel transfers. The method will not be discussed here.



7.1 Multiplexer Block Diagram

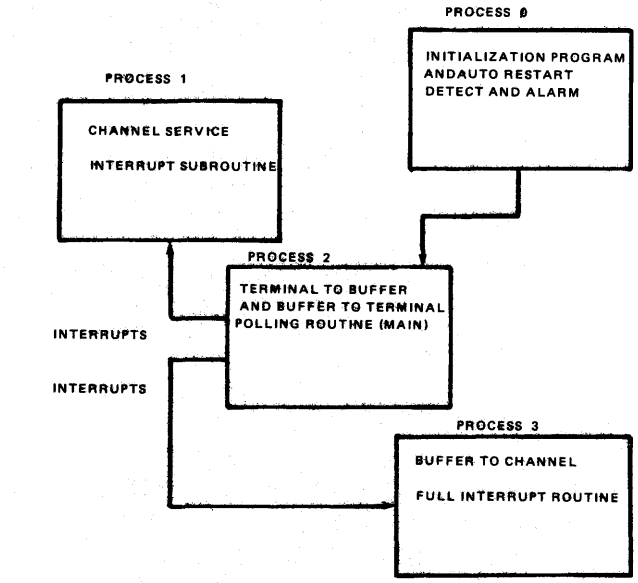
There are three processes, running one at a time: input-output service polling routine, host-to-terminal buffer interrupt process, and terminal-holding-queue to host interrupt process. They will be described in the following section.

SOFTWARE

A flowchart of the software appears in Figs. 7-2, 7-3, 7-4 and 7-5. The software can be divided into four parts: the initialization routine, the polling routine, the interrupt routine to fill terminal buffers from the host, and the interrupt routine to empty the terminal-to-host waiting-queue.

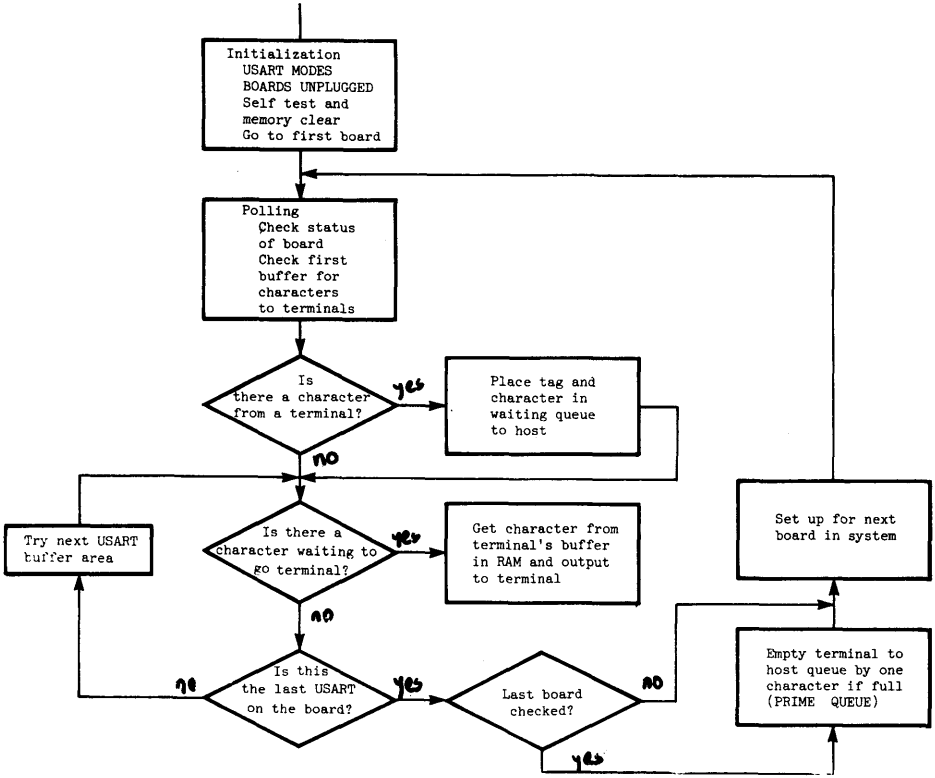
The initialization runs only when reset, then the latter processes may run, one at a time. They communicate only through the output data-buffers and share no other common memory space, other than pointer tables.

The initialization routine clears all memory, sets up tables, finds which boards are plugged in, resets all USART's, and will print out errors, if a debug board is installed. This is roughly all the system housekeeping. It sets the stack-pointer, resets and sets the mode, speed, and number of bits-per-word on the USART's. This section of the program is 60% of the code used for the whole application.

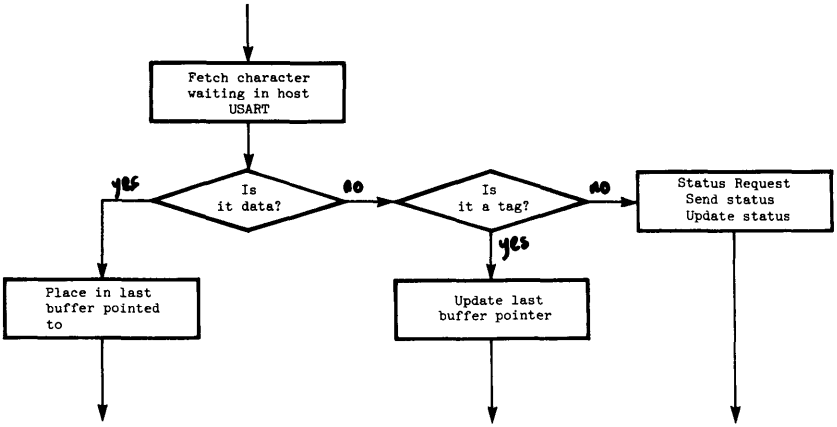


TOTAL 8080A BYTES FOR PROGRAM: 526 BYTES! LESS THAN ¼ OF THE 2708 USED

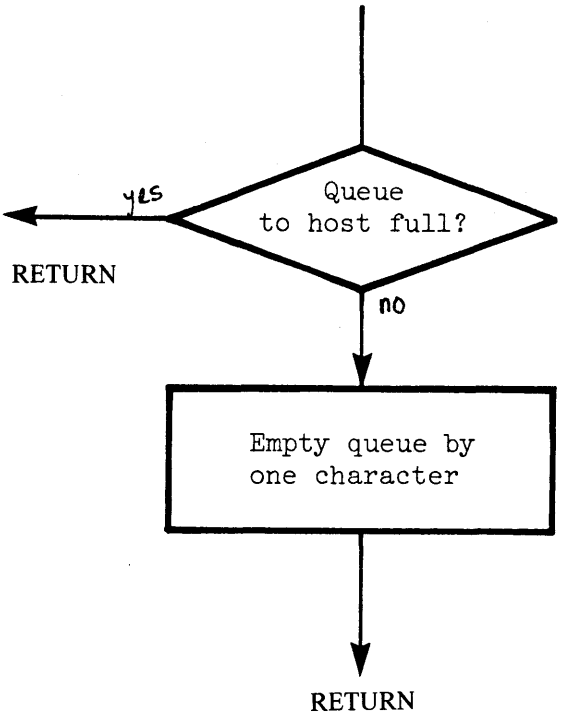
7.2 Multiplexer Software: Overall Program Flow



7.3 Multiplexer Software: Polling Loop



7.4 Multiplexer Software: Host to Mux Interrupt



7.5 Multiplexer Software: Mux to Host Queue Interrupt

The polling routine goes through the list set up by the initialization program, testing to see if there has been a character typed by a terminal, or if there is data in a buffer, to be output to a terminal. Thus, each of the 32 terminals is serviced once during each pass. If the channel-to-host is busy (it takes 1 millisecond to transmit a character at 9600 baud), the characters are put into a waiting queue, that will be serviced when the "channel-not-busy" interrupt comes in. If the channel is not-busy, the waiting queue is emptied by one character, and the character currently waiting is placed at the end of the line, in the queue. In this way, the queue-service routine is primed and will continue to interrupt, when not busy, to empty all the characters waiting for the channel. The format used for data transmission is the following: the tag for that terminal is sent first, and then the character is sent to the host, via the queue routine. Each board has its own priority table, so that only one input is processed, per pass, per board. After a character is transmitted, or, if a board has no characters, the buffer area for each terminal is then checked to find if there is an output character pending. These are placed in the buffer by the host-interrupt routine. If so, the buffer gives its character to the USART, to be transmitted, and all the pointers are updated. When there are no incoming characters, and no buffer is full, the system still polls each board for input, and each USART buffer for output.

The channel-queue-interrupt routine looks at the queue, and transmits a character, if there is one waiting; otherwise it returns. This routine will not be called again by interrupt, until the polling routine primes it by sending a character.

The host-interrupt routine waits for information to come from the 11/70, or host-machine, before it executes. When a character is received, and ready, an interrupt is generated, that then starts this interrupt process. This process checks the incoming character and, if it is data, places it in the appropriate output buffer area. After this, polling resumes. Other characters from the host perform status requests, data-tag-switch, and soft-restart commands.

The host-interrupt routine may interrupt at any time during polling. It first saves the status-vector of the machine, then picks up the character that caused the interrupt. If the most-significant-bit (MSB) is a "1", the character is a tag, or a command. If it is a tag, it is stored, so that the following data characters are loaded into the buffer pointed to by the last tag.

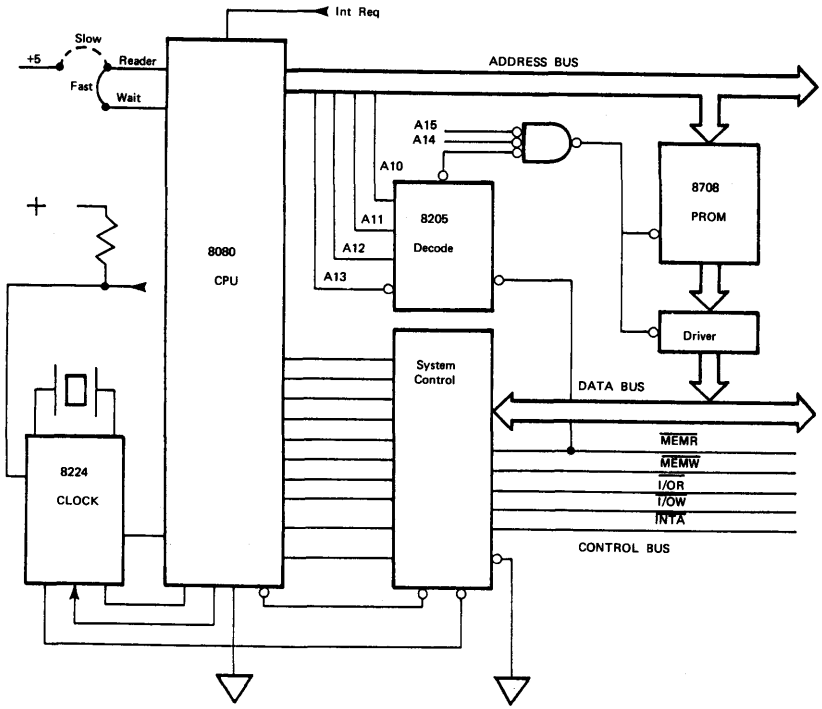
The most-significant-bit could also mean that it is a command. The commands allowed are: "status-request", "status-change", and "soft-restart". "Status-request" will send back a status-tag followed by the status of that USART. "Status-change" will take the next character, and

transfer it to the USART control register. This can be used to turn ports on or off, and change baud-rate by a factor of four. "Soft-restart" will re-initialize the entire system. Caution is advised in the use of these controls: do not expect the data buffers to be unaffected by their use! This is because these commands require more time than is allowed to poll all the terminals. Thus, interrupts are locked-out and characters may be lost. These commands are usually used to re-initialize the system from the host, after the host crashes.

The most-significant-bit being "0" means that the character presents data. This character is then loaded into the last place in the buffer pointed to by the last tag. All following characters will load into the same buffer, until a new tag is sent.

The CPU and PROM Module

In Fig. 7-6, we see the 8080 CPU-board schematic. This board contains all of the necessary CPU interface circuitry along with one 2708 programmable ROM, and the necessary bus buffers.



7.6 CPU Board Schematic

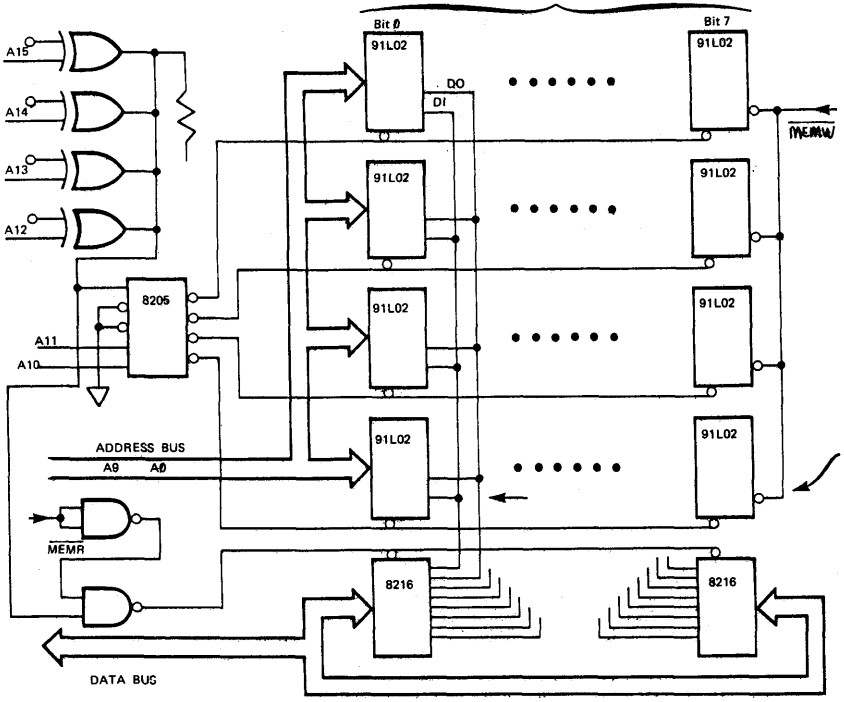
The 8080 needs a clock, and a system-controller. These functions are provided by the 8224 and the 8228 chips, respectively. The 8224 provides the necessary timing from the 18 megahertz crystal to drive the two-phase clock of the 8080. It also provides the reset signal synchronization necessary.

The 8228 system-controller provides the system with the control-bus and also buffers the data-bus, so that all of the modules in the system can be driven with no load limitations.

Also on this board are 1,024 bytes of EPROM provided by the 2708. Notice that the selection of this device is fully decoded. The EPROM will only respond to addresses from "0000" hexadecimal to "03FF" hexadecimal. This is where the multiplexer program resides.

The selection is done as follows: all address bits A10 through A15 must be low, to enable the EPROM, as well as the $\overline{\text{MEMR}}$ signal. The first four of those signals, along with this $\overline{\text{MEMR}}$, go to a 1-of-8 decoder, an 8205. If all of these are zero, then the first output is selected. Then this output is

7.7 RAM Board



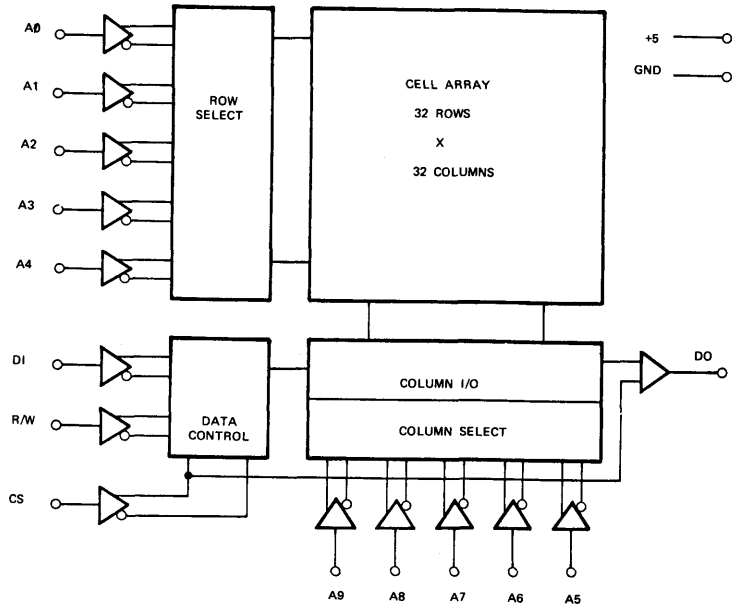
checked with the last two address lines. If all are zero, then the $\overline{CS}$ is held low, selecting the EPROM. The EPROM bus driver, an 8212, is also enabled at this time to drive the appropriate cells' data onto the data-bus, to be read by the processor.

RAM Modules

There are two memory-cards in this system. They are both identical, except one is for addresses "1000" hexadecimal through "1FFF" hexadecimal, and the other is for addresses "2000" hexadecimal through "2FFF" hexadecimal. These two cards provide 8,192 bytes of RAM storage.

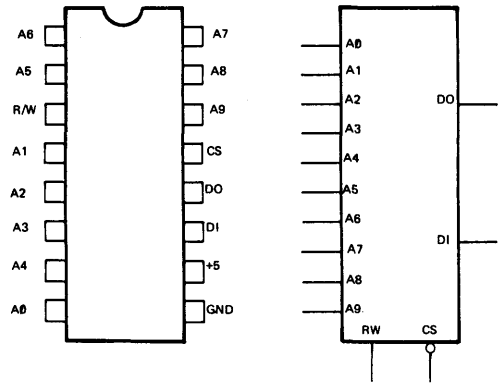
Each card contains 32 static 1,024 × 1-bit RAM chips, bus-drivers and receivers, and address-selection logic.

A single RAM chip can store 1,024 bits of information. In order to store 4,096 × 8 bits, we need to organize these chips into a *memory array*. Note that we need one chip for each bit, and that we need four sets for 4,096 bytes.

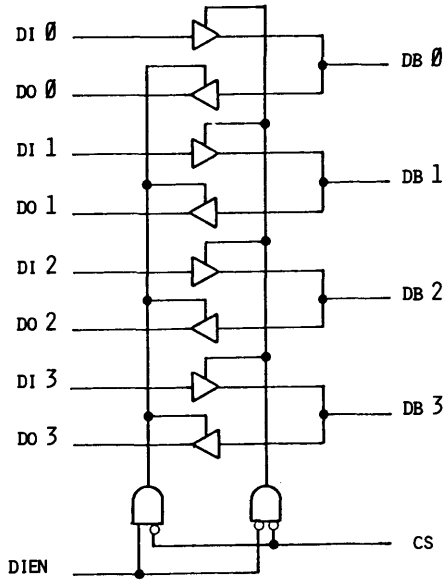


7.8 Detail of 91L02C

Since, for any group of 1,024 bytes, eight 91L02's will need to be enabled, the chip-selects for each of the groups of eight are tied together. From there, these four group-selects go to a 1-of-8 decoder.



Pinout of 91L02C



7.9 8216 Bidirectional Bus Drivers

The data bits are bussed from each group in the direction perpendicular to the chip select. All bit 0's should be tied together, as well as bit 1's, bit 2's, bit 3's, etc. Since 91L02's cannot drive the bus directly, all input data lines come from an 8216 bidirectional bus-driver-and-receiver. In a similar fashion, all data outputs from the 91L02's go to the 8216 bidirectional bus-drivers. An illustration of the 8216 appears on Fig. 7-9.

Two of these devices will provide a standard method of listening to, and driving, the data-bus. The $\overline{\text{DIEN}}$ signal controls whether the bus is driven by the 8216, or whether the bus is listened to. The $\overline{\text{CS}}$ enables the outputs to drive both the bus, and the D0 outputs. If $\overline{\text{CS}}$ is high, all of the DB and D0 pins are in the high-impedance state.

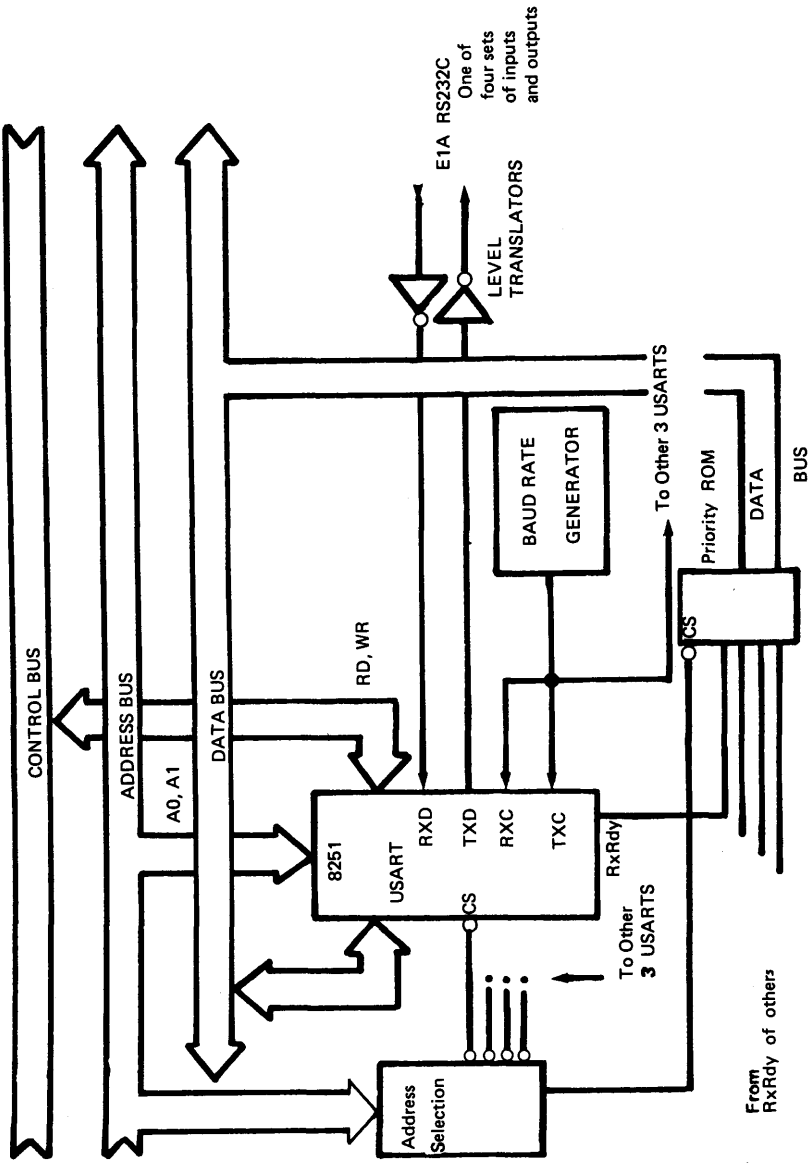
The direction of data-flow is determined by the $\overline{\text{MEMR}}$ signal. When it is low, the RAM will put data out onto the DI lines of the 8216's. The bus-drivers will be enabled, to drive the 8080 data-bus with this data. At all other times, the memory-array listens to the bus. The only time it will write data into the memory is when the $\overline{\text{MEMW}}$ signal goes low, and the chips are selected.

The address-selection is performed in a way so that the address of the board may be selected by jumper wires. The low ten address bits go directly to the 91L02's. The next two bits go to a 1-of-8 decoder (8205) to select one of the four sets of eight memory chips. The enable line of the 8205 comes from a wire-ANDed combination of exclusive-or (XOR) gates.

Only when all of the outputs from these four gates are high, will the memory board be enabled. Each XOR gate compares an address-bit with a jumper wired to "1" or "0". If both are identical, the output will be "0". If they are different, the output will be "1". To set these jumpers for the right address, we set the jumper to the opposite of what the high four address-bits should be. If we want "0010", for A15-A12, the jumpers should be tied to "1", "1", "0", "1", respectively. In this way, the board will respond only when an address lies in the area of $0100XXXX\ XXXXXXX_2$. This is pages "20" through "2F" hexadecimal, or "2000" through "2FFF" hexadecimal. *Exercise for the alert reader: What should the jumpers be for "1000" through "1FFF"?*

The USART Board

In Fig. 7-10, the basic card for all the terminals' interface is shown. This card contains four 8251 USART's, a baud-rate clock-generator, and a priority-encoded status-generation PROM.



7.10 USART Board

The 8251 is the basic serial interface element. Grouped four to a card, they are connected together on their data-buses to form an on-card data-bus. Similar to the memory card, this on-card bus is buffered by 8216's onto the system-bus. This is because the 8251 cannot drive more than eight other LSTTL loads. The 8251 is selected by implementing an address-decoding technique, using an 8205. Note how these devices are memory-mapped input-output. That is, since the same signals that control memory ($\overline{\text{MEMW}}$, $\overline{\text{MEMR}}$) control the USART's, they appear as memory locations. According to our memory map, when bit A15 is high, we are addressing input-output. This corresponds to locations from "8000" to "8FFF" hexadecimal. Note that since the lower eight address-lines are not decoded, these are "don't cares" in our memory-mapped I/O map.

The first card starts at "80XX" (where "XX" means that these bits do not matter) and, since each USART has two registers (input-output and control), the address ends at "87XX" hexadecimal. The next card goes from "88XX" to "8FXX", and so on, with the last card addressed by "B8XX" to "BFXX". The even page-addresses are the status-registers, and the odd ones are the data-in and data-out registers.

Note also that there is a special PROM on the card, which is decoded by a separate decoder. Its address is "70XX" for the first card and "77XX" for the last card. The function of this PROM is to place on the data-bus the actual address of the USART which has received a character from its terminal. How is this done? Each of the "RxRdy" lines on the USART's indicate whether a character has been received. These four lines, one from each USART, are tied to the *address-lines* of the PROM.

One of 16 possible bytes may be selected by the decoding. The fifth address-bit is jumpered to a one or a zero. In this way, the same PROM can be used for board 0 or board 1, by placing in the other 16 locations the addresses for board 1, and setting the jumper on the fifth address bit to a 1. (Jumper to zero for even, one for odd). What are these 16 locations? They are simply a table of the addresses "81", "83", "85" and "87" hexadecimal for board zero, and "89", "8B", "8D", "8F" for board one. Similar PROM's are made for the other six boards.

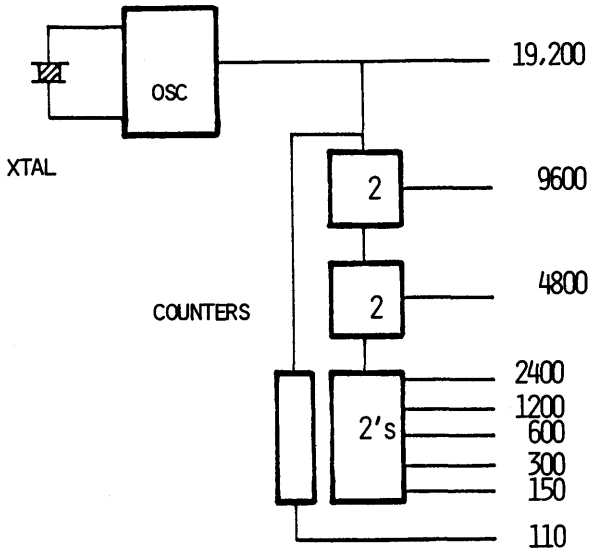
The values are placed in such a way that the first location in the PROM is a byte of zeroes. That way, when no USART has a character, and all RxRdy lines are low, the byte of status is all zeroes, indicating that there is "nothing" to do for this board. If it is not zero, then a character is waiting. To make sure that it is easy to tell which USART is waiting, the next location contains the value "81": if the first USART is waiting, and all the others are not, the program will receive an "81" from the status PROM. The program can then use this value to directly address the actual character

waiting. What is more, the value "81" can be masked, to form the tag for the data fetched.

The next two locations contain "83", the next four "85", and the next eight, "87". In this way, a priority table is formed so that, as each USART is serviced, the next one waiting will be serviced in turn.

This method of addressing the status-PROM allows the program to use only a few instructions to identify which USART, out of 32 possible ones, is ready with a character, fetch the character, and generate the proper tag from that status information.

There are two interface chips to take the TTL serial inputs and outputs from the USART's and convert them to EIA RS232C +12 and −12 volt serial pulses. These are simple level-translator integrated circuits.

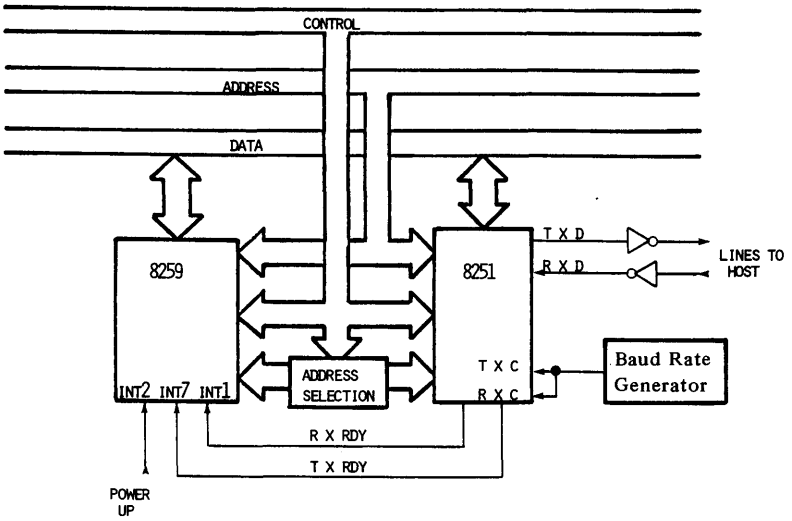


7.11 Baud Rate Generator

The last section consists of an astable multivibrator, synchronized by a crystal to provide the timing for the serial bit clocks. Two simple dividers are on each board to provide the USART's with all of the common serial rates. This is shown in Fig. 7-11.

The Host Interface Board

This module contains: the host USART, the interrupt-controller, and a baud-rate generator for the host-to-multiplexer communication rates. It appears in Fig. 7-12.



7.12 Host Interface Board

The units on this board are addressed as input-output ports, instead of memory locations. The USART is addressed as ports "F9" and "FA" hexadecimal, for control and data, respectively. There is a duplicate of the baud-rate circuit here to generate the "TxC" and "RxC" signals for the host-to-multiplexer USART, as these rates may differ from any of the others in a typical system.

The interrupt-controller takes the "RxRdy" and "TxRdy" signals from the USART, and generates two interrupt-vectors, number 1 and number 7. Number 1 is to signal that a character has been received from the host, and should be processed, and number 7 indicates that the USART can be re-loaded to transmit another character to the host.

The 8259 interrupt-controller is set up by the initialization routine, to call the service routines at the proper locations, and service the interrupts on a rotating basis. After an interrupt has been serviced, the software will reset the corresponding bit-flag in the 8259, and proceed with polling, until a new interrupt arrives.

Fig. 7-13 illustrates the initialization procedure of the PIC and Fig. 7-14 presents the interrupt-handling code at the beginning of memory.

PORTs F7 and F8 are PIC

CONTROL	ADDRESS	DATA	OPERATION	
WRITE I/O	F8	32	sets low address for call	for INT 1
WRITE I/O	F7	ØØ	sets high address for call	
WRITE I/O	F8	F2	sets low address for call	for INT 7
WRITE I/O	F7	ØØ	sets high address for call	
WRITE I/O	F7	70	enables only INT 1 and INT 7	
WRITE I/O	F8	AØ	sets rotating priority reset mode	

7.13 PIC Software Load Format

```
0000                    ORG OH              ;INITIALIZATION STARTS
0000 00         RST0:   NOP
0001 31FF2F             LXI SP,2FFFH        ;SET THE STACK POINTER
0004 F3                 DI                  ;DISABLE THE INTRRUPTS
0005 C3D700             JMP INIT            ;SYSTEM RESTART UPON RESET
0008 C5         RST1:   PUSH B              ;HOST TO MUX RST VECTOR
0009 D5                 PUSH D              ;PUSH STATUS VECTOR
000A E5                 PUSH H
000B F5                 PUSH PSW
000C CD4900             CALL INT70          ;INT70 GETS THE CHARACTER FROM
000F 3E08               MVI A,0008H         ;HOST--DECODES IT AND RETURNS.
0011 D3F8               OUT 00F8H           ;INTERRUPT CONTROLLER RESET INT 1
0013 F1                 POP PSW
0014 E1                 POP H
0015 D1                 POP D               ;FLAG
0016 C1                 POP B               ;POP STATUS VECTOR
0017 EF                 RST 5               ;PRIME QUEUE
0018 FB                 EI

0019 C9                 RET
0020                    ORG 0020H
0020 CDC700     RST4:   CALL SND50          ;SOFTWARE RESET
0023 C7                 RST 0

0028                    ORG 0028H
0028 F5         RST5:   PUSH PSW            ;SAVE A AND FLAGS
0029 DBFA               IN 00FAH            ;READ THE USRT STATUS
002B E601               ANI 0001H           ;CHK FOR TXRDY
002D CA3100             JZ POPAF            ;IF USRT IS BUSY RETURN
0030 FF                 RST 7               ;ELSE CALL RST7 FOR FIFO SERVICE
                                            ;TO CHK IF ANYTHING IS IN THE
                                            ;FIFO TO SEND TO 11/70
0031 F1         POPAF:  POP PSW
0032 C9                 RET
0038                    ORG 0038H
0038 C5         RST7:   PUSH B              ;MUX TO HOST RST VECTOR
0039 D5                 PUSH D
003A E5                 PUSH H              ;CHANNEL NOT BUST
003B F5                 PUSH PSW
003C CD1802             CALL OINT           ;OINT IS OUTPUT A CHARACTER
003F 3E08               MVI A,0008H
0041 D3F8               OUT 00F8H           ;FROM QUEUE
0043 F1                 POP PSW
0044 E1                 POP H
0045 D1                 POP D
```

7.14 Example of Interrupt Control

RSTØ; Hardware Initialize.

RST1; Character from Host has arrived.

RST4; Soft-reset on program fail ROM detect.

RST5; Channel to Host is not-busy check. Mux to Host buffer queue should be emptied.

RST7; Channel to Host is not-busy. Check buffer queue for characters, if any transmit, if not, return.

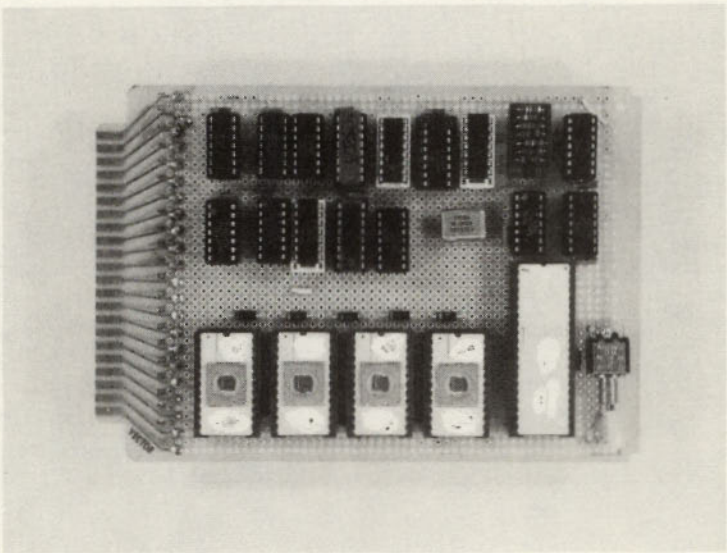
The channel-to-host was set to 9600 baud in both directions. The characters from each terminal must be echoed, as this is a full duplex system. For every character generated, the host must process and return the echo. There are 24 Lear-Siegler ADM-3s terminals, set to 9600-baud input and output. There are also four 300-baud terminals and four 300-baud dial-up lines on the multiplexer.

Typical averaged input rate is ten characters-per-second. Average output rate is 200 characters-per-second. Buffers in the host, for characters waiting for output channel are 95% of the time empty, indicating the host can get rid of data as fast as the channel can handle it, rather than as fast as the terminals can print. Maximum rates measured are 15 characters-per-second on input, and 620 characters-per-second on output. The maximum and typical figures were obtained over a 17-hour period, when 90% of the terminals on the multiplexer were in use.

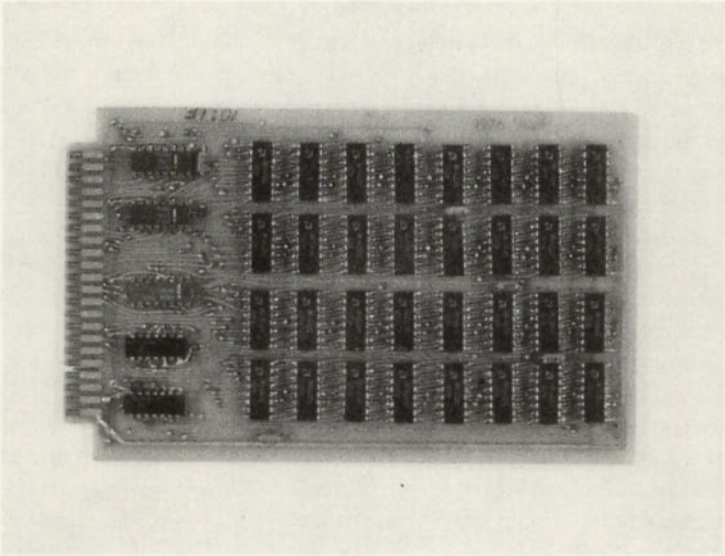
Error rates were entirely due to the channel, or at least indistinguishable from other errors, such as operator errors, and host errors.

Photographs of the printed-circuit boards appear in Figs. 7-16, 7-17, 7-18, 7-19.

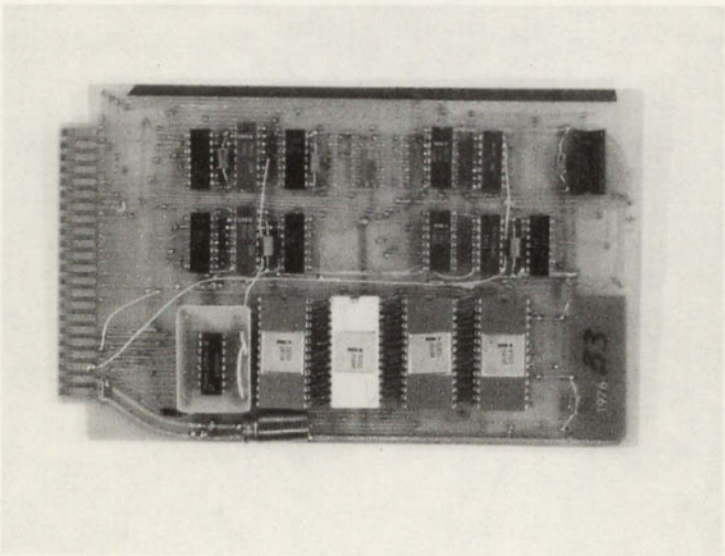
PICTURES OF MULTIPLEXER PROTOTYPE P.C. BOARDS



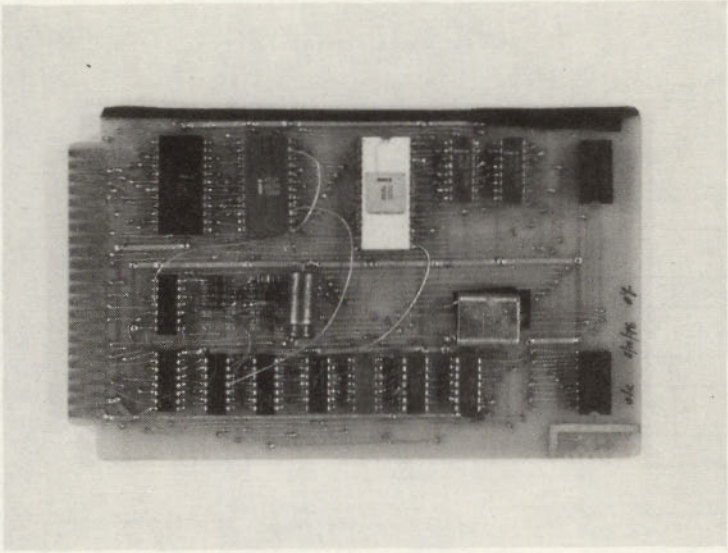
7.16 CPU



7.17 RAM



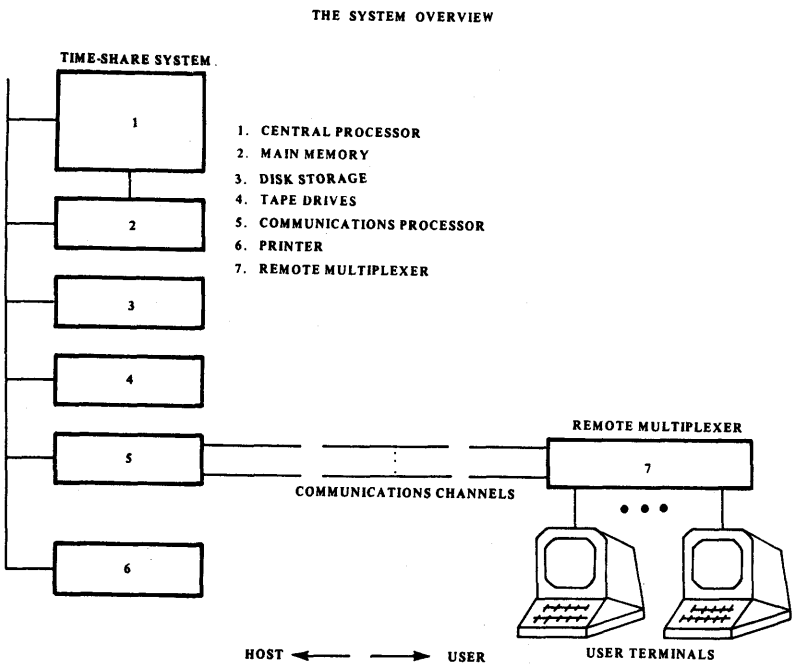
7.18 TERMINALS' USARTS



7.19 HOST AND INTERRUPT CONTROL

CONCLUSION

In this chapter, a complete interface was described. A step-by-step discussion of how each component was integrated into a module, how the modules created a subsystem and then the overall system, should enable the reader to follow through most any other microprocessor interface application. This particular application utilizes most of the techniques discussed in previous chapters: interrupts, memory and I/O management, integrating special techniques for software reduction in hardware, and external device interface were used here.



7.20 OVERALL SYSTEM

CHAPTER 8
TESTING

INTRODUCTION

What do you do when it doesn't work? What went wrong and why? The debugging process, also known as testing or trouble-shooting, is an integral part of any system design. Murphy's Law usually holds: if anything can go wrong, it will.

When faced with a misbehaving system, there are a number of techniques available to the designer to identify and correct problems. In this chapter, the causes of common problems, and their solutions, will be presented. Problems such as: component failure, software failure, noise-induced failure, will be analyzed, and methods for identifying them will be presented.

The tools necessary to identify and locate these problems will also be described: voltmeter, logic probe, signature analyzer, oscilloscope, digital analyzer, in-circuit emulator, emulator, and simulator.

Finally, a case history of the "One Bit in 16,384" will be presented. The example illustrates the debugging phase in the actual design of the multiplexer presented in Chapter 7.

WHAT GOES WRONG?

Four essential problems may arise in a system: wiring fault—a short or open circuit; component failure—including wrong value components; software bugs; and noise or interference—either internal or external.

Wiring faults are detected by a resistance-check from point to point in the system. Check each wire: *make sure it goes to the right pin and no other* on the integrated circuit. Make sure you look-up circuit pin-outs twice. *Do not be confident that the schematic is without fault until the system works.*

Wiring faults are the most common and troublesome problems. They are easily solved—although they take time. Most circuit boards are "buzz-tested" with a simple continuity-checker that emits a tone for a short, and no noise for an open. Such a tester leaves both hands and eyes free to keep track of the wiring.

Component Failure

Components such as resistors, capacitors, inductors, transformers,

transistors, diodes, integrated circuits, and connectors may all experience failures. Resistors crack open, capacitors leak out their electrolyte. In short, *no component is perfect. Everything fails sooner or later.* Each component is given a figure of merit, known as its *mean-time-between-failures* or *MTBF*. This is a statistical prediction, in hours, of *how long the part will last in a given environment.* A table of percent/1000 hours failure-rate is shown in Table 8-1 for applications in military avionics.

TABLE 8-1

Component	(%/1,000 hr) Failure Rate
1. Capacitor	0.02
2. Connector contact	0.005
3. Diode	0.013
4. Integrated circuits, SSI, MSI, and LSI	0.015
5. Quartz crystal	0.05
6. Resistor	0.002
7. Soldered joint	0.0002
8. Transformer	0.5
9. Transistor	0.04
10. Variable resistor	0.01
11. Wire-wrapped joint	0.00002

Some parts last longer, on the average, than others. Of course, *this table assumes that all parts are being used properly.* These figures are based on accelerated-life-tests on a large sample for each part.

Failure-rate is defined as 1/MTBF. Knowing the failure-rate of each component in a system will yield the failure-rate for the entire system. The rule is to add the failure-rates of all of the components in the system. This gives the system failure-rate—the inverse of which is the system mean-time-between-failures.

For example, suppose we have three LSI chips, one crystal, ten resistors, ten capacitors, a printed circuit board with connectors, a transformer, four diodes, and an IC voltage-regulator. This system is to be used in the same environment as the components that were tested. What is the system failure-rate? Using Table 8-1, we find:

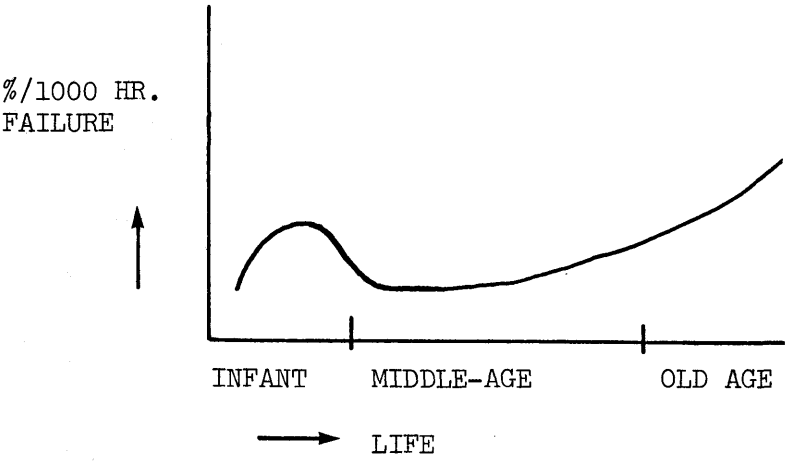
four IC's	.06	
crystal	.05	
ten resistors	.02	
ten capacitors	.50	
P.C. board	~.60	(10 connectors, 500 soldered points)
transformer	.50	
diodes	.052	
TOTAL	1.82%/1,000 hours	

This yields a MTBF for the system of:

$$1/1.82\%/1{,}000 \text{ hours or } \approx 60{,}000 \text{ hours}$$

Suppose we made 1000 of these systems, and used them in the specified environment? After 1000 hours, it would be most probable that 18 would have failed. After 10,000 hours, 180 would have failed.

How often do parts fail? This simple question, which we have answered on an average basis, tells us nothing about the *distribution of failures*. It gives the *mean*. Most components exhibit the following lifetime characteristics shown in Fig. 8-2.



8.2 %/1000 hr Failure versus Age

Most failures occur when new, or when old, and fewer failures occur in the "middle-age" of the components.

"New" and "old" differs for each component. In-depth analysis of the entire system involves simple, but time-consuming calculations, concerning each component's lifetime failure history.

A "burn-in" test tries to weed out the *"infant-mortality"* part of the curve before parts are shipped to the buyer.

The Table is accurate only for the environment specified. Commercial, industrial, and military applications, all lead to different ways of measuring the MTBF. A unit designed for a child's toy may last five years, if used as a toy; if shot into space, it would not last five minutes. The application's environment determines the basic reliability statistics to be used.

We have only addressed so far the topic of *reliability*. A separate problem is *quality*. Contrary to intuition, high quality doesn't always mean high reliability. Quality refers to how well a component will last, doing its job. The part may be noisy, dissipate lots of heat—but it may also work longer than a part that is quiet and dissipates less heat. Only thorough statistical analysis can determine reliability. Quality is easily measured on a part-by-part basis.

Software

Software can be at fault. For example, suppose there is a special routine in the program to handle a power failure. The problem is that, when coding, a mistake was made in the part of the program which restores the machine when power returns. *If you never tested this routine, it may not be used until the power fails. Only then, will you know that your machine does not meet specifications.*

A second example is when an arithmetic calculation causes an overflow-and-halt condition only when some measured input value is "0". The system may work well for months, and then stop mysteriously, every two days after that. Software problems, or *bugs*, are often the hardest to identify.

Tools for finding who is at fault, engineer vs. programmer, will be after the noise discussion. However, software problems are the most common in a microcomputer system. No program is ever perfect. A program is limited in precision, speed, and flexibility. The smart programmer is a complete pessimist about his software until it has been running for a number of years.

Noise

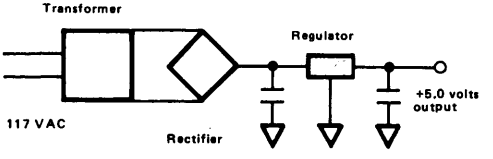
Noise is everywhere. Whenever there is a current in a wire, there is an electromagnetic field. Thus, fields from power transformers, motors, and electrical wiring are everywhere. In addition, with all the radio, television, citizens' band, and amateur radio transmitters—any length of wire becomes an antenna. *Not only can noise come from the outside, but it can be generated inside your system.*

Four examples are:

1. When integrated circuits switch, they generate small current changes in their power requirements because of internal circuit characteristics. If too many circuits switch at once, the power-supply voltage may change enough to affect other parts of the circuit. There are usually bypass-capacitors near each integrated circuit to prevent this type of noise.

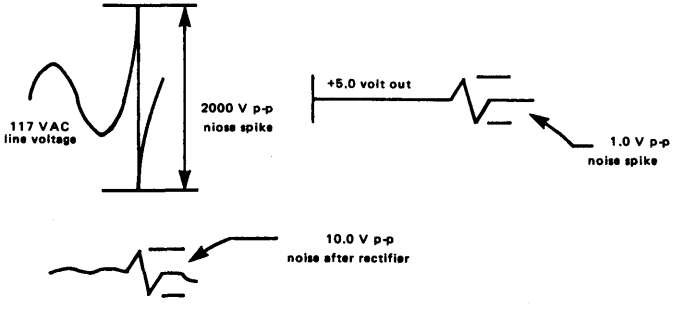
2. If two wires are close together, a pulse traveling along one induces a pulse in the other, because of the transformer action between the two. The induced pulse may reflect, and toggle a flip-flop, or cause the data to be incorrect. To prevent this, twisted and shielded pair transmission lines are used.

3. The power-supply may not be properly designed. There is a small amount of 60-cycle ripple on the 5-volt supply. This may affect the contents of memory, and cause an improper read or write. Proper power-supply design accounts for the droop in voltage under heavy load before the regulation circuits.



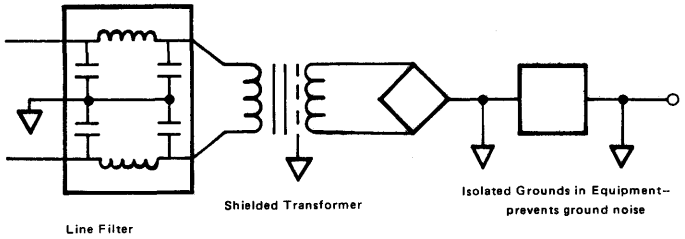
8.3 Noise Spike on Power Line

4. In Fig. 8-3, there is a typical noise spike from turning on a teletype on the power line. Notice what happens to that noise spike in a plain power supply in Fig. 8-4, without noise filters. If that glitch happens at a crucial moment, data are lost and the machine fails.



8.4 Power Supply Without Line Filter

The solution here is to use a line filter, and a shielded transformer, that prevents high-frequency noise pulses from getting through as in Fig. 8-5.



8.5 Power Supply With Line Filter

Summary of Common Failures

Components fail at predictable rates, software may not be reliable and correct, and noise may be happening all around and inside the system. How do we go about finding the fault in a rational fashion?

The next section will deal with the tools used to find the faults and identify them. In this discussion of tools, the tell-tale signs of each problem will be discussed.

The Trouble-Shooting Tools

We will present here the, tools available, and the kinds of problems which can be identified with them. Tools will be examined closely as to their own limitations.

Table 8-6 presents a short summary of problems and tools. The discussion will follow this Table and expand on each problem—what a tool can do to find it, and how long it would take.

Simple Problems

Short and open conductors, wrong voltages—these are the most common problems. Luckily, they are the easiest to detect. Any ohm-meter can check for gross conditions such as open, or short, and a digital voltmeter (DVM) or volt-ohm-milliampmeter (VOM) will suffice, to check voltage and currents. If you know your components and design, it is an easy matter (although time-consuming) to make sure everything goes where it belongs, and draws the right currents from the proper voltages.

THE DEBUG MATRIX: PROBLEMS & TOOLS

You can solve problems like:	You have Equipment: VOM	PROBES	SGN.ANA.	OSC.	D.D.A.	I.C.E.	EMU.
· shorts, opens, wrong voltages	yes	maybe	no	yes	maybe	maybe	no
· bad resistors, caparitors	yes	no	no	yes	no	no	no
· unknown logic signals bad-fault tree already generated	yes	yes	yes	yes	yes	no	no
· unknown logic signals bad-fault tree available	yes time consuming	yes	no	yes time consuming	yes	yes	no
· software problem	no	no	no	maybe	yes	yes	yes

To fix a typical Problem:	You need at least					
Eventually	yes	yes		yes		
In an average time	yes			yes	yes	
Fastest way possible	yes			yes	yes	yes

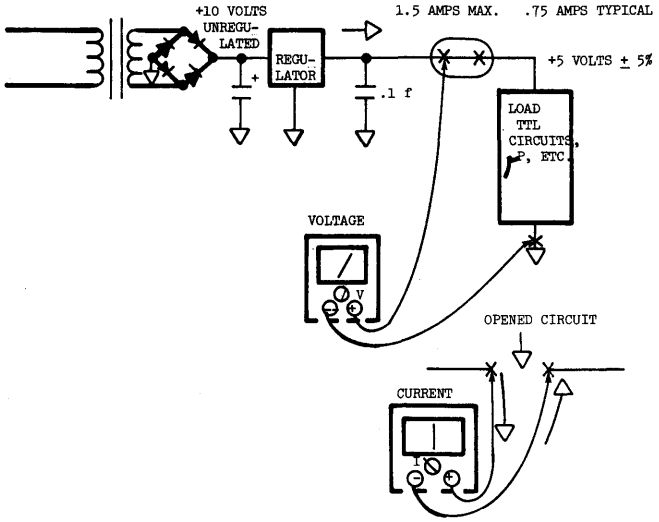
TABLE OF ABBREVIATIONS

VOM	VOLT-OHM-MILLIAMPMETER
PROBES	LOGIC PROBES
SGN.ANA.	SIGNATURE ANALYSER
OSC.	OSCILLOSCOPE
D.D.A.	DIGITAL DOMAIN ANALYSER
I.C.E.	IN CIRCUIT EMULATOR
EMU.	SOFTWARE EMULATOR OR SIMULATOR

8.6 Problems and Tools

The VOM

To measure a voltage, the meter is placed in parallel with the circuit element. Fig. 8-7 shows the measurement of the power-supply voltage at the output of a regulator. The VOM will easily measure all such voltages, but be warned that it will not detect excessive ripple or noise on the power supplies.



8.7 Measuring Voltage and Current With a VOM

To measure a current, the meter must be placed in series with the component. This means the circuit must be broken. If possible, connections should be made, so that in-circuit current measurements need not cut wires or traces. Any dynamic behavior of the circuit may not be measurable, yet could cause problems.

In the power-supply example, the meter measured the voltage across the load, and then by disconnecting the load, and reconnecting it through the meter, the current was measured. Be sure to check that these measurements are within the required tolerances. Improper values may indicate later trouble.

Bad Components

Resistors, capacitors, diodes, and transistors can all be checked against known good devices. They can be measured with the DVM or VOM to determine whether they are basically functional. Other special test equipment is needed for diodes and transistors to establish device characteristics.

Integrated circuits are difficult to test without expensive equipment. When debugging, several of each device used should be kept in stock, in order to replace a device with an inherent malfunction. Once the entire circuit is working, all devices in stock should be tested in the prototype system to make sure that no marginal problems occur in production, due to component tolerance changes.

Simple problems usually prevent the system from working at all. Intermittent failures are most often due to connector or bad solder joint problems. These should be checked first, before assuming something else is a fault. All intermittent problems will require an oscilloscope (preferably with storage) or a logic-analyzer for quick, effective debugging.

All static problems can be solved. This is the first step: be completely confident about this stage before continuing.

Design Problems

You thought you knew what you wanted—but you didn't. Yes, we all make mistakes so we might as well admit it. Design errors are divided into two general categories: *improper specification* and *improper use*. Examples of each follow.

Improper Use:

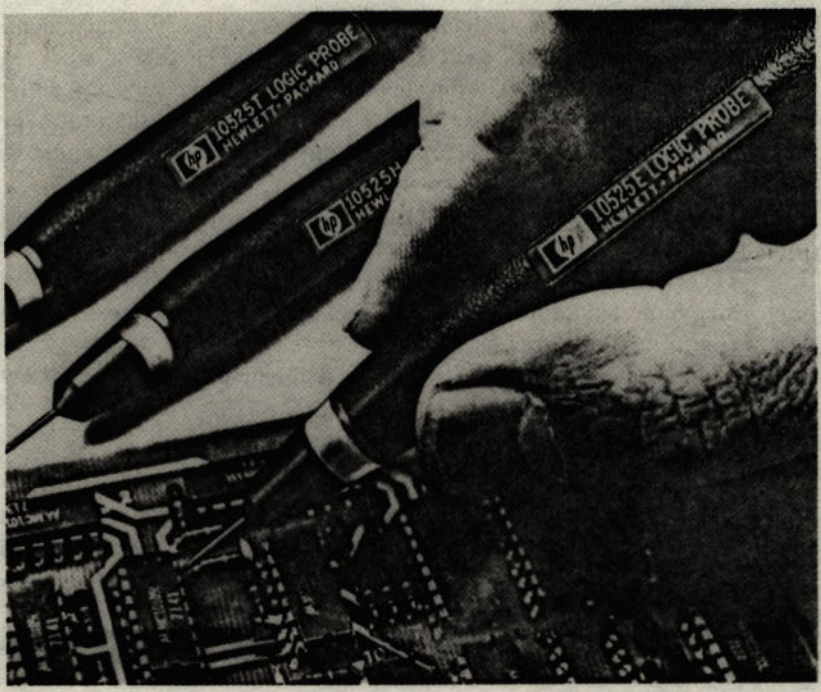
Passing too much current through a resistor will cause it to burn up. Applying too much voltage to a capacitor will cause it to short. Every device has its limits. The "too much" problem is the most common. For example, too many loads on a single output line may cause the system to read or write improper data values intermittently depending on temperature variations.

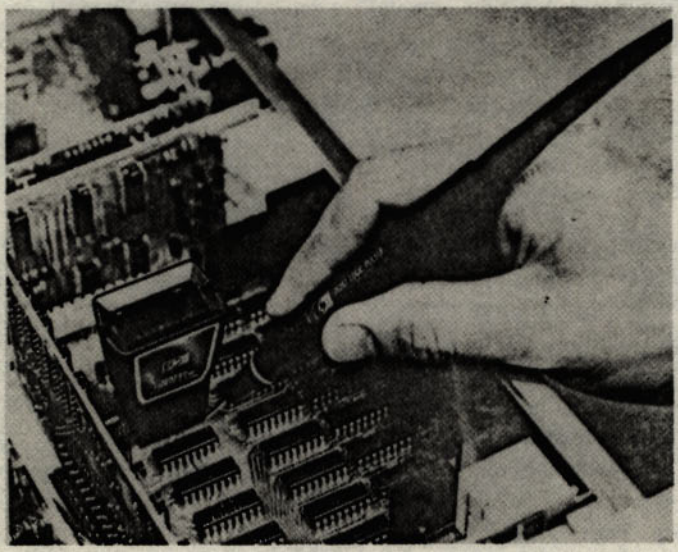
Improper Specification

If we believe a part to be able to drive 30 bus loads when it can only drive 20—this is improper specification. It simply was not noticed in the data-sheet upon specification.

More subtly, the timing of a particular part may be misunderstood. For example, if the address gated to a memory part must be stable 20 nanoseconds before the data and write pulses, this may have been overlooked and the system timing design violates this condition.

Design problems require a full range of equipment for proper trouble-shooting, but a VOM-oscilloscope combination will suffice if time is of little concern. These problems manifest themselves primarily in an intermittent fashion in the case of overloading bus lines, and in burning and smoking parts, in the case of overvoltage/current.





8.8 Logic Probes

The burning parts problems are simple—get a bigger part or improve the design so it will work with the parts you have.

The intermittent problems require that all input-output loading be checked, all device specifications be checked, and the system operated at different temperatures to localize the sensitive component(s).

A can of freeze spray and a heat lamp can locate temperature sensitive problems quickly and easily, by selectively heating and cooling the suspected parts.

Logic Probes

Logic probes can verify logic levels quickly so as to isolate any static conditions efficiently. The probes will indicate whether a signal is a 0, 1, or undetermined by using an LED indicator or light bulb. Watch out for undetermined states: unless it is a tri-state bus floating, and it is supposed to be floating, something may be wrong. Fig. 8-8 illustrates a logic probe in use.

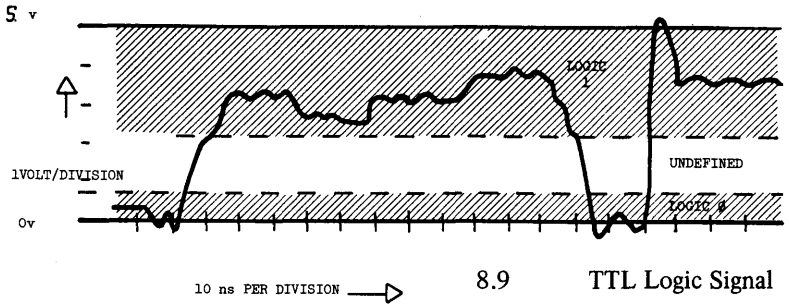
DYNAMIC PROBLEMS

In operation, the system doesn't work. *The VOM, logic probe etc. will not indicate time. Thus, they are of little use in the dynamic case.* We need devices which will indicate that the logic level timing is correct.

The Oscilloscope

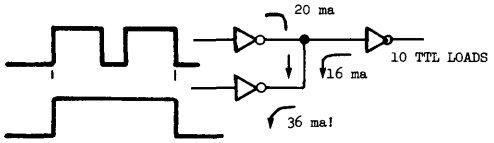
To obtain timing information, the *oscilloscope* is most commonly used. With one or more traces, events may be measured accurately in amplitude and duration, in function of time. In microprocessor systems, events as short as 10 nanoseconds should be observable with an oscilloscope. A 10 nanosecond square wave will appear as a sine wave on a 10 megahertz oscilloscope. Thus, to see these events clearly, a 50 or 100 megahertz scope is desirable. Fig. 8-9 illustrates the trace on a typical oscilloscope of a TTL logic control signal.

The logic zone definitions here are for standard TTL. The logic "0" signal is for any voltage between −0.6 and +0.8 volts. The logic "1" signal is from +2.0 volts to +5.5 volts. Anything in the zone from +0.8 to +2.0 is considered undefined. Transitions from one level to the other should occur in much less than one microsecond to avoid noise problems. The oscilloscope will indicate if a bad logic level is present. For example, if two TTL outputs are connected together, we have violated a design rule. If the condition occurs, where the two outputs wish to go in opposite directions, one of

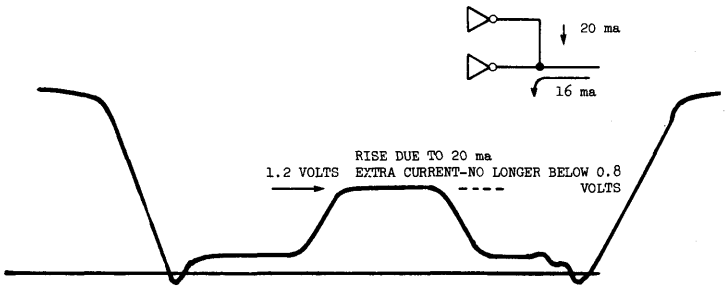


8.9 TTL Logic Signal

the gates may be destroyed. If the condition occurs for only a few microseconds at a time, no harm will be done; however, the fault will cause problems. Fig. 8-10 shows a trace for such a condition. Note how the logic "0" level is not correct.

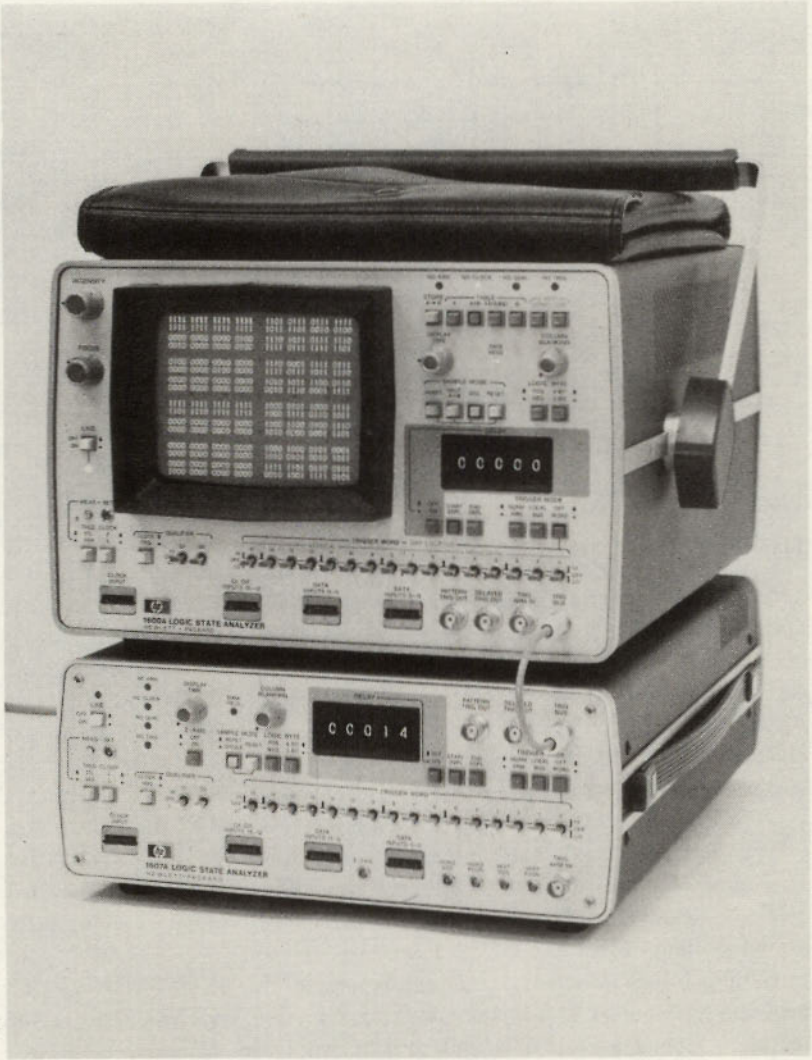


8.10 TTL OUTPUT FAULT



Such a measurement, along with the knowledge of the logic family drive specifications, will indicate to the trouble-shooter where the fault lies.

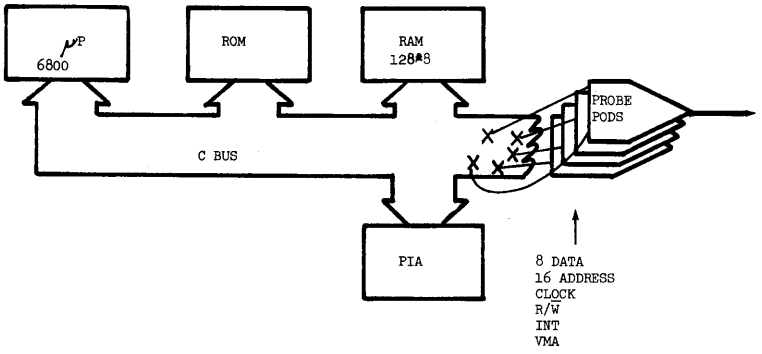
Observing chip-select, control and bus lines with the oscilloscope will clue you to load problems, timing problems, and noise problems. Make sure that the logic levels are well defined. TTL "0" should be from −0.6 to +0.8 volts. TTL "1" should be from 2.0 to 5.5 volts. Anything else means trouble.



1600S Analyzer

STATE MEASUREMENT

All system timing and system logic levels are correct when observing any single bit or line—but we need to observe all the lines at once in time. We could gather 16 oscilloscopes together, and early analyzers were simply multi-channel oscilloscopes, but is is not specially convenient to observe 32 tiny traces on the face of an oscilloscope tube. For this reason, we developed *logic analyzers*, or more accurately, *digital-domain analyzers.*



Connecting the Analyzer

Logic Analyzer

What does a digital-domain analyzer do? It allows to observe up to 32 nodes in the system, *simultaneously*. It will display these bits in binary, octal, hexadecimal, or in the form of conventional oscilloscope traces. It will begin displaying the information when a given combination of bits, or *trigger* occurs. It will store every clock cycle, or more often a new set of signals, and be able to display a few sets of signals *before* and *after* the trigger set. Each set of signals in time is known as a *state*.

Available analyzers fall into two categories: those that emphasize *timing* information, and those that emphasize *state* information.

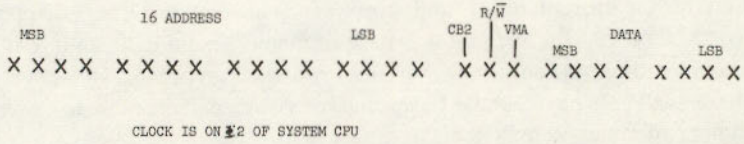
Timing-oriented analyzers are merely multi-channel oscilloscopes. These devices are useful where logic glitches, noise, or logic level problems are suspected.

State-oriented analyzers attempt to present the flow of the system's program by monitoring all important circuit points. State-analyzers are effective in debugging software and complex software-hardware faults.

Example of a State Analyzer

The Hewlett-Packard 1600S Analyzer has 32 channels, two clocks, four trigger qualifiers, and many other features. The instrument will take a "snapshot" of the state of the system on every clock cycle. We will use the HP1600S to observe the interrupt cycle in a 6800-based processing system.

Table 8-11 lists the format of the data displayed on the 1600S. The probes were attached to the lines indicated. The clock was connected to Φ2.

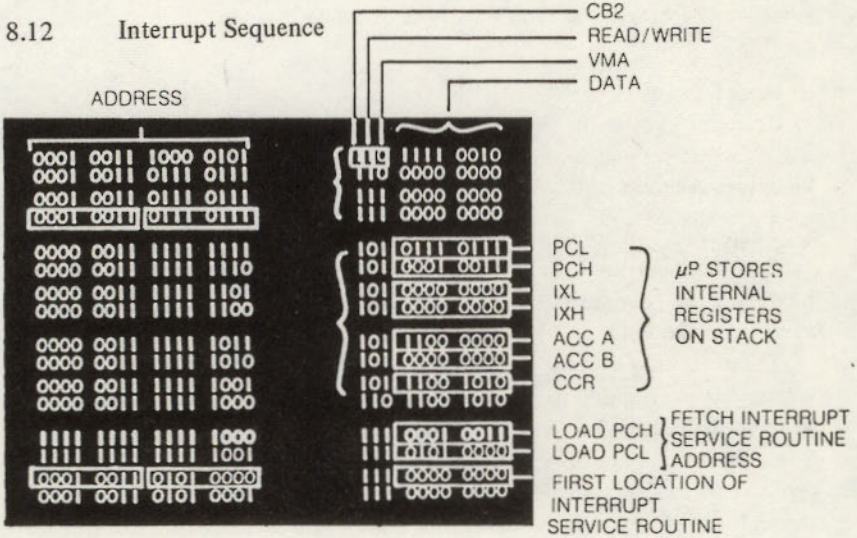


8.11 HP1600S Display Format for 6800 Interrupt

The 1600S was triggered by the interrupt signal. In Fig. 8-12, the state flow is displayed. The data displayed are:

1. The current instruction cycle is finished. Instruction is an "F2" hex at location "1385" hex.
2. The status is now pushed onto the stack, before going to the service routine vector location. Note the stack is at locations "3FF" hex downward. The program-counter, index-register, accumulators, and flags, are stored in successive locations in the stack.

8.12 Interrupt Sequence



3. The microprocessor now fetches the contents of addresses "FFF8" and "FFF9" hex. The contents are transferred to the program-counter.

4. Interrupt-service routine begins at "1351" hex. Execution continues from this point.

With such a device we have a roadmap of where the system was, where it is, and where it is going.

Some analyzers store a proper sequence of states, continuously compare those with the current states, and stop upon a mismatch. Others display a "1", "X", or "0" for each bit in a page of memory, and indicate if that bit has been read or written. Some store more states than others. However, all of these analyzers have similar basic characteristics of being able to observe a number of states in a system, in a time sequence.

The digital domain analyzer allows the designer to monitor software execution so that wrong data, wrong addresses, or wrong instructions, may be found. If a digital-domain analyzer is used to trigger an oscilloscope, noise problems and subtle timing problems may also be identified.

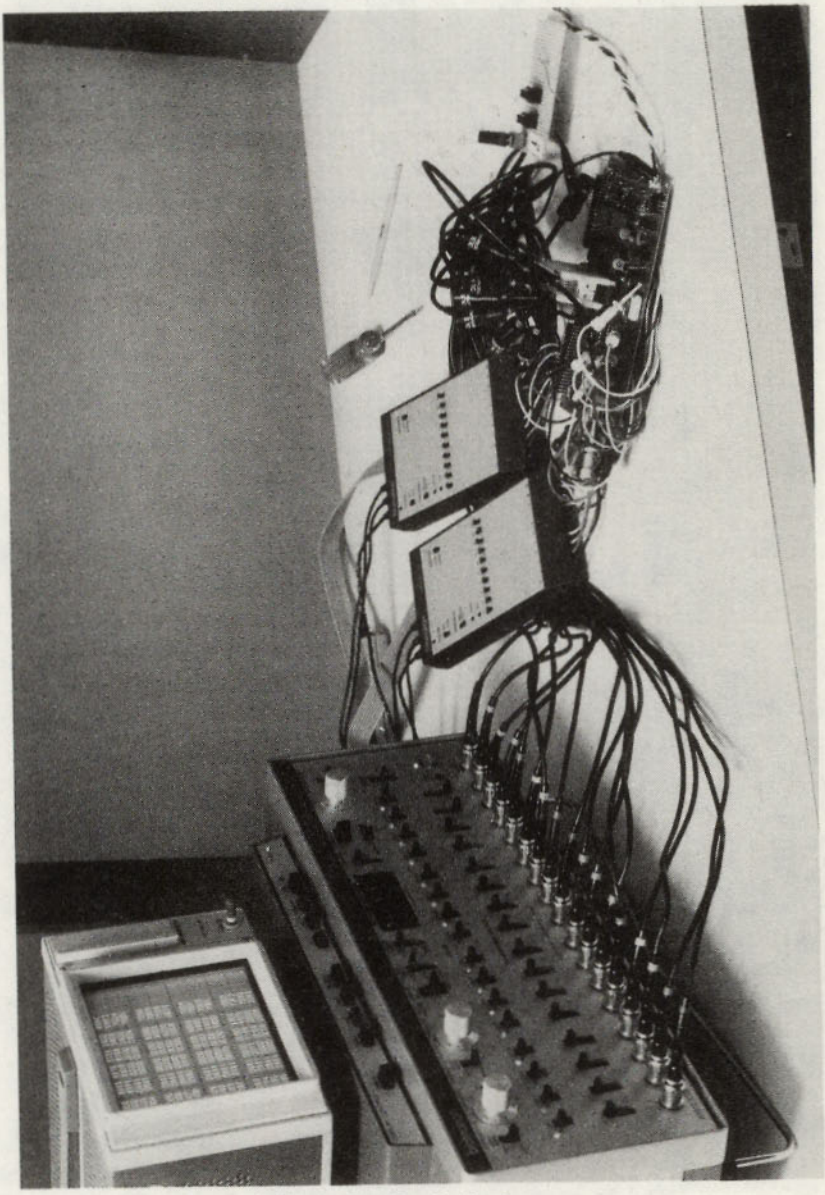
In-Circuit Emulation

In-circuit emulation allows to "get inside" the microprocessor itself, dynamically watch where it is going, what it is reading and writing. It makes it possible to monitor the processor itself. It includes breakpoints and test routines to allow you to "catch" a specific section of code as it goes by, and display the contents of the internal registers. By checking these against what you expected, the fault may be located.

Signature Analysis

There are a whole range of special tools usable only once an initial system has been built and tested. These systems rely on the known behavior of the original system in order to predict what went wrong in the system, in the field.

These techniques rely on a *fault-tree*. That is: everything that could go wrong has been made to go wrong, and in each case, nodes in the circuit were measured to discover just how such a failure would manifest itself. Some fault-trees are short: if the fuse blows, replace it; if the fuse blows again, call the service department. Some trees walk the service person through the entire system, depending on measured values.



Biomation Logic Analyzer

8.13 HP ICE for 8080 System
Mnemonic Analyzer

HP Signature Analyzer

A Signature Analyzer

This device relies on the fact that any repetitive sequence of signal values may be stored in a recirculating shift-register, whose value, clocked into a display each time around, will have a certain value. A device can be designed so that the probability of two bit streams having the same value or "signature" is extremely small.

Thus, each node in a system will have its own signature when it is working correctly. It will also have a special signature for each possible problem. By using a fault-tree method, developed by using the analyzer, all faulty equipment can be debugged quickly down to a faulty component.

It will not find initial software problems, or the cause of intermittent failures in a system.

In Fig. 8-14, we see the trouble-shooting flowchart, using an HP5004A Signature Analyzer. These signatures were generated on a good instrument and the chart developed to speed repair.

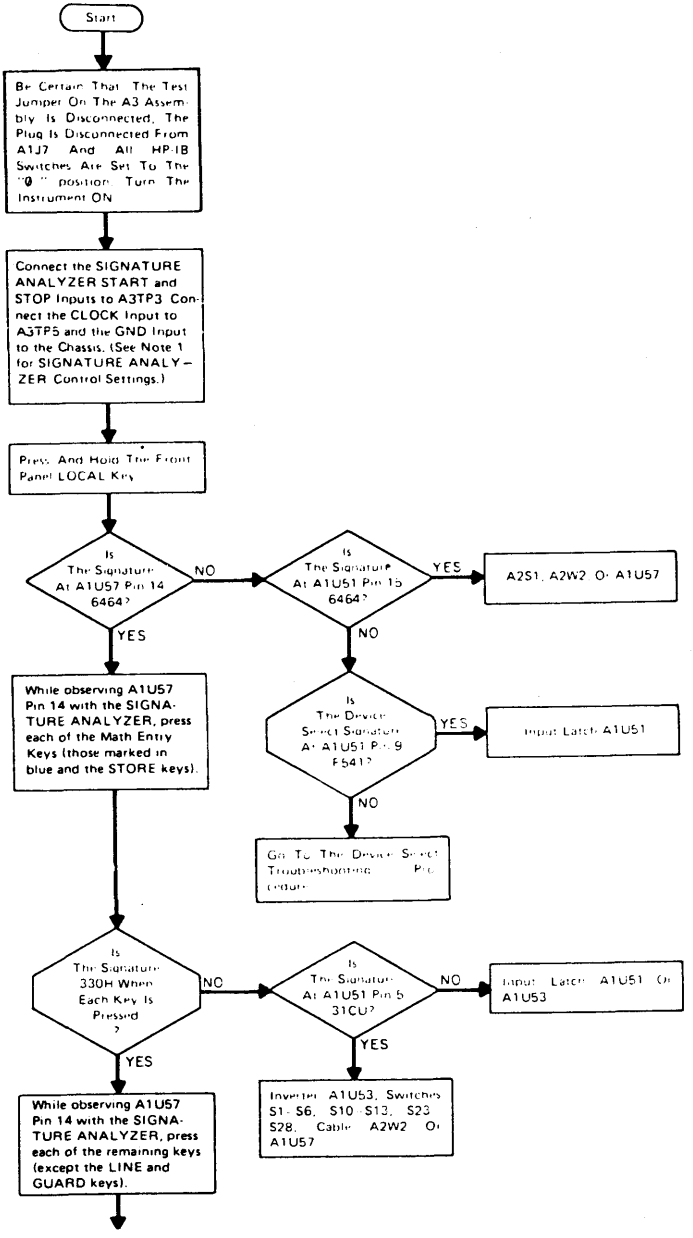
SOFTWARE TESTING TECHNIQUES

The underlying principle of all testing techniques is to compare an existing board, component, or system, to "what it should be". The problem may naturally be to know what it should be, or else to implement a reasonable procedure for performing this comparison in a systematic manner. In addition, two supplementary problems arise: making the measurements themselves, and recording the history of the last n signals. For this purpose, a number of new hardward and software tools have been developed. The test instruments and techniques used in performing such comparisons have been described in the preceding section. As usual in the computer world, either hardware or software methods can be used. The purpose of this section is to explain the software testing techniques.

The four basic methods used in testing microprocessor-related equipment are: comparison testing, self-diagnostic, stored-response, algorithmic pattern generation.

Comparison Testing

In this method, a device, or a board, is compared to a known "good" device, or "good" board. They share the same common input, and outputs are compared. This is a hardware method, and the required tools have been presented. The next three techniques are essentially software techniques.



8.14 Fault Tree

Self-diagnostic

In the self-diagnostic method, the microprocessor system itself determines whether it is operational, and if not, which part of the system is defective. The basic principle of self-diagnostic is to execute a "worst-case" sequence, and to observe the results. In the case of the MPU itself, a worse-case sequence of instructions is usually available from the manufacturer. Typically such a sequence will exercise all the machine's instructions, in a pre-determined order. In addition, it may include some critical sequence of instructions which has been found to fail in some cases. Clearly this information is usually available only from the manufacturer. Most of them are cooperative in supplying such a program. Naturally, the following question arises: *what if the MPU itself is indeed defective?* If it is defective, it is likely that the program will not terminate successfully, and that the system will "crash", with no external warning. When performing such a self-diagnostic, an automatic warning mechanism must be used. For example, the system will print a message on the printer saying "undergoing diagnostic testing at time X". At time X plus one minute, the system should have completed diagnostic testing, and should print the message "diagnostic-testing completed successfully". If such a message is not printed, the system will be assumed to have failed. Optionally, external devices may be set. For example, an external alarm, with its own timing mechanism may be actuated at the beginning of the test. Unless the timer is reset within a specified amount of time, the alarm will go off, signaling an MPU failure automatically.

Such self-diagnostic programs are extensively used on systems enjoying idle time. It is a simple matter to write the basic test program using most of the machine instructions, and residing in some unused portion of the ROM. Whenever the microprocessor is idle, such a program may be run, and thus verify the machine integrity. In addition, if it is run continuously for a period of time, it will help isolate intermittent failures of the system. Naturally, it need not reside in ROM, and may be loaded in RAM from an external device.

Self-diagnostic is also used to test memory or input/output facilities. The topic of memory-testing will be addressed in detail, in a paragraph below on algorithmic pattern-generation. In the case of a ROM memory, the simplest form of self-diagnostic is called *checksum-validation.* In this technique, each block of data such as 16, 32, or 256 words is followed by a one byte or two-byte checksum. Typically, such a checksum is computed by summing the n half-bytes of the block of n words, using hexadecimal arithmetic. This sum is then truncated to the last four binary digits, and the checksum character is the ASCII encoding of the resulting hexadecimal digit. A simple program

executing in a secure area of the ROM (a portion of the ROM which is assumed to be good) can read the contents of the rest of the ROM, recompute the checksum, and then compare it to the value which has been stored. If a mismatch is detected, a ROM failure has been identified.

Testing input-output interfaces and I/O devices is usually complex, in view of the delicate timing relationships involved. However a rough checking is possible as to the correct overall operation of the devices themselves. Provided that feedback information is available from the device, an order will be issued by the program such as: "close relay A." Provided that the feedback path be available, relay closure can be verified within n milliseconds. In this way, the system can exercise all of the external control devices, and verify their proper overall operation. In addition, during systems operation, "reasonableness-tests" are usually run on all input devices (see book C20 for a complete discussion). Such tests will compare the value of input parameters to values in a table, stored in the memory, and determine whether this input data is "reasonable." For example, when measuring the temperature of water, temperatures below 0^{O} C and over 100^{O} C will be deemed "unreasonable." Similarly, for a microprocessor controlling a traffic light at an intersection, detecting vehicle speeds over 200 mph will be deemed unreasonable. Naturally tests can be much finer than the simplified examples, in a specific environment. Such reasonableness-testing will detect intermittent and permanent failures of input devices and will set-off an external alarm.

Stored-response

In the stored-response method, a large-scale computer system is used to emulate, or simulate, the device, or the board, under test. First, a program is used to measure the characteristics of the device, or system, under test, preferably under dynamic circumstances. This data is then recorded, and will be used by the comparison program. The comparison program is then applied to the device. It will generate input signals. The outputs are measured, and compared to the previous response of the system, as stored in the tables. In such a system, two phases are necessary. The first phase is a characterization phase where the computer system is used to record essential responses of the system that will be later used as references. Once these responses are obtained, in phase two, the system will only run in comparison mode by executing a specific test program and measuring the response.

This method is used essentially in production, and for incoming testers. The cost of the system required to provide efficient stored-response testing, plus the programs, can range from $50,000 to $500,000.

Algorithmic pattern generation

Algorithmic pattern-generation is essentially used for testing RAM memory. The principle is to write a pattern in the memory, and then verify that:

1: it was written correctly, and

2: that nothing was written anywhere else because of a RAM malfunction. The two basic pattern generation techniques used in RAM testing are fixed-pattern tests, and galloping-pattern tests.

Fixed-pattern Testing

In a *fixed-pattern* test, identical, alternating, cyclical patterns are successively written, then read, at each memory location. This will detect gross RAM failures. However, this will not detect *pattern-sensitivity* problems. Pattern-sensitivity is a typical source of failure in high density chips. Because of the geometrical layout of the chip, some combination of bits written at some instant of time in memory cells might cause some other bit position elsewhere in the device to turn on or off. This problem can happen in RAM memories or in microprocessors themselves. Whenever this problem occurs in a microprocessor, it is a basic design failure, and there is not much the user can do about it. The best that can be done by the user is to run a worst-case program, supplied by the manufacturer, which has been shown to make similar units fail because of the specific sequence of instructions involved. This problem will not be considered here as it is deemed highly infrequent, once a chip has been in the field for more than a year. In the case of memory, however, especially in the case of high-density memory, pattern-sensitivity is a frequent problem which can be diagnosed relatively easily using an algorithmic pattern-generation test. This will be described in the following section.

Galloping-pattern testing

The galloping pattern test is usually abbreviated "galpat." The principle of this technique is to write successive binary values into memory cells, then compare them to all of the rest of the memory, before moving on to the next memory location. In this way, if writing into memory cell zero affected the contents of memory cell 102, this will be detected by the test. In a typical galpat, the memory will be initialized with a known content, such as all ones, or all zeros. The basic test algorithm is the following:

1. The contents of a location L-1 are tested against the contents of all other memory locations. They should match.

2. The address L-1 is then incremented by one, and step one is carried out until all memory locations are tested.

3. The initial data pattern is then complemented, and one goes back to step one.

Many variations are possible on this basic galput. They have been nicknamed "marching ones and zeros," "walking ones and zeros," and "galloping patterns" (galpat one and galpat two).

Ideally, one should write all possible patterns in each memory location, and after writing a pattern in every word, check every other word of the memory, to verify whether it might have been changed. In addition, after checking each of the other memory words, one should immediately come back to the original memory location under test in order to verify that its pattern has not been changed by the tests performed on another memory location. It could happen that the fact of checking every other memory location would modify the original contents of the memory cell, then modify them again so that eventually they would have the correct initial contents. A possible failure would then not be detected if one did not come back every time to verify the contents of the initial cell. It is easy to see that such exhaustive testing will require a very high number of operations. A simple memory exerciser, checking a 32 K memory will typically run for several minutes. It will, for example, write all zeros, or all ones, or write its own address in each memory location, and then rotate these addresses through the available memory. If the test uses galpat techniques, it could easily run for half an hour, or even for several hours. For this reason, these tests are usually run only during the initial debugging phase of the system, or when a malfunction is suspected. It is not practical to consider their use once the microprocessor system is operational, unless a simplified version is used.

SIMULATION AND EMULATION

Let us first introduce the basic definitions. *Simulation* refers to the functional replacement of a hardware device by a program. It is said that the device is simulated by software. The program will generate the same outputs as the hardware device, in response to the same inputs. Unfortunately, it will perform such a simulation much *slower* than the original device.

Emulation refers essentially to a simulation performed in *real-time.* In fact, many emulators will simulate the operation of a complete system even faster than the model. For example, many bit-slice systems emulate the instruction-set of another computer. They will execute all the instructions of the processor being emulated at the same speed, or sometimes even faster.

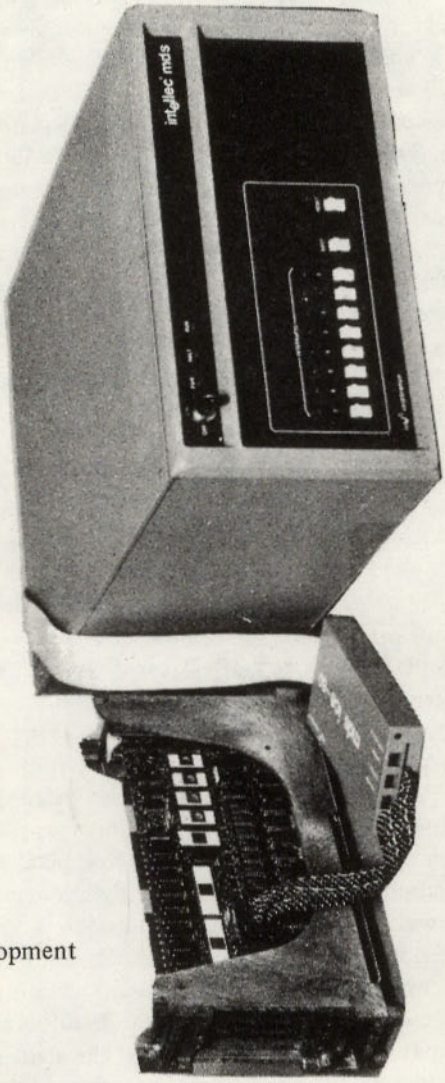
Simulation is used for two essential devices: the microprocessor itself and the ROM memory. ROM simulation, or emulation, is performed by executing programs out of RAM, as if they were in ROM. This is normally done during the development phase of all programs. Clearly an initial program will contain a number of bugs, and should not be directly placed in a final ROM or PROM. In a typical development-system, such a program will be installed in RAM memory and be tested and debugged there. The two main problems are to convert the addresses of the final program into those required by the ROM and to maintain speed compatibility. Typically the RAM-board resides at a specific physical address which will not correspond to the actual address of the ROM chip in the final system. The second problem is a synchronization problem whenever a slow RAM is used initially, and a program is then installed on a faster ROM. Such ROM emulation or ROM simulation facilities are a normal part of any microcomputer development system and will not be addressed in greater detail here.

Simulation and emulating the microprocessor itself is much more complex. Simulating the microprocessor is used in two cases:

1. when the MPU itself is not available.
2. for convenience in debugging.

These two cases will not be detailed. When programs are developed on a large-scale system, *cross programs* are used. A cross-assembler will create for example 8080 code on an IBM 370. It is necessary to test the correct execution of the resulting 8080 code. This will be performed with a simulator. An 8080 simulator will be used, which executes all the 8080 instructions in simulated time. In this way the complete logic of the program will be tested. The essential limitation of such a simulator is the fact that no input-output can be tested, unless the user deposits known data at the right time into selected memory locations. Input-output registers are then simulated by memory locations. Unfortunately the timing of input-output is often random, and almost always complex. For this reason, a simulator is only used to test the overall logic of a program. This is fine for testing numerical algorithms, such as a floating-point package. This is inadequate for debugging a complex input-output interface.

In any system where the user must test real input-output in real time, one of the most significant aids in testing is the emulation of the microprocessor itself. This is called "in-circuit-emulation."



8.15

Software Development

In-Circuit-Emulation

In-circuit-emulation was originally introduced by INTEL on its MDS system, and is now available on every leading microprocessor development

system, as well as on independent systems. The picture of an actual "in-circuit-emulator" ("ICE") appears on the illustration. A special board has been inserted on the INTEL MDS system on the left which provides the in-circuit-emulation facility. On the right appears a system under development. The board with the microprocessor itself has been pulled out of the rack and plugged into an extension board so that its components be easily accessible. The 8080 itself has been removed from its socket, and a special cable called the "umbilical cord" has been plugged into the socket. This is the cable appearing in the illustration. This 40-line cable is terminated by a 40-pin connector identical to an actual 8080. The essential difference is that all the signals carried by this cable are generated by, or under the control of, the in-circuit-emulator, rather than the real 8080. What is the purpose of replacing an actual 8080 by a software emulator? The essential facility provided by the emulator is to completely control and test the system under development (on the right) *from the console.* It is possible *to stop* the operation of the 8080. It is possible to examine the registers or change them. Doing this on an actual 8080 would require opening up the package, removing the lid, and placing microprobes under a microscope, to obtain the contents of the registers, if indeed this were possible. The contents of the registers are not available in an actual 8080. Only the values on the busses are. Using an emulator, it is possible to stop the operation of the 8080 automatically, using *breakpoints* in the program. This facility will be clarified below. It is possible to examine, or change, registers, as well as the contents of the memory. It is possible to sit at the keyboard, and execute *actual input-output instructions,* such as closing a relay, by hitting a key on the keyboard. It is then possible to stop again the processor and examine the busses, the registers, or the memory. In addition, all the operations may be performed in conjunction with the powerful software-aids available in a development-system. Examining, or changing, memory can be performed in symbolic form, rather than in binary or hexadecimal. This is called *symbolic debugging.*

Breakpoints are a facility to specify and address where the program will automatically stop. Addresses are selected, and a list of breakpoints is given to the emulator. When the specified location is reached during execution, the emulated microprocessor will automatically stop, and allow the user to verify contents of registers, busses or memory. In addition, an in-circuit-emulator provides an essential capability called *trace-back.* It provides essentially a snapshot of the history of the signals on the busses during a specified length of time. In the case of the INTEL ICE, it provides a 44 machine-cycle trace-back. Whenever a breakpoint

is encountered, the in-circuit-emulator stops the execution, and provides the user with a symbolic debugging facility. Typically, when an error is detected at the breakpoint, it was not caused by the instruction at the breakpoint, but was the result of a *previous* instruction in the program. The essential problem is to locate the previous instruction which caused the problem. This is a *tracing* problem. With the traceback capability, it is possible to examine the previous signals, and to determine which were the instructions executed before the detection of the error. If this historical record is not long enough, an earlier breakpoint can be set-up and an additional segment of the history of the system will become available. This process can be repeated until the error is finally identified.

An in-circuit-emulator does not require an important configuration for software or hardware to execute. It is an essential debugging capability, as it provides for the first time a tool for checking the operation of the complete system including the actual input-output devices, in real time. In any system involving debugging of actual input-output boards, or interfaces, the availability of an in-circuit-emulator is essential.

DEBUGGING A CONCEPTUAL PROCESSOR

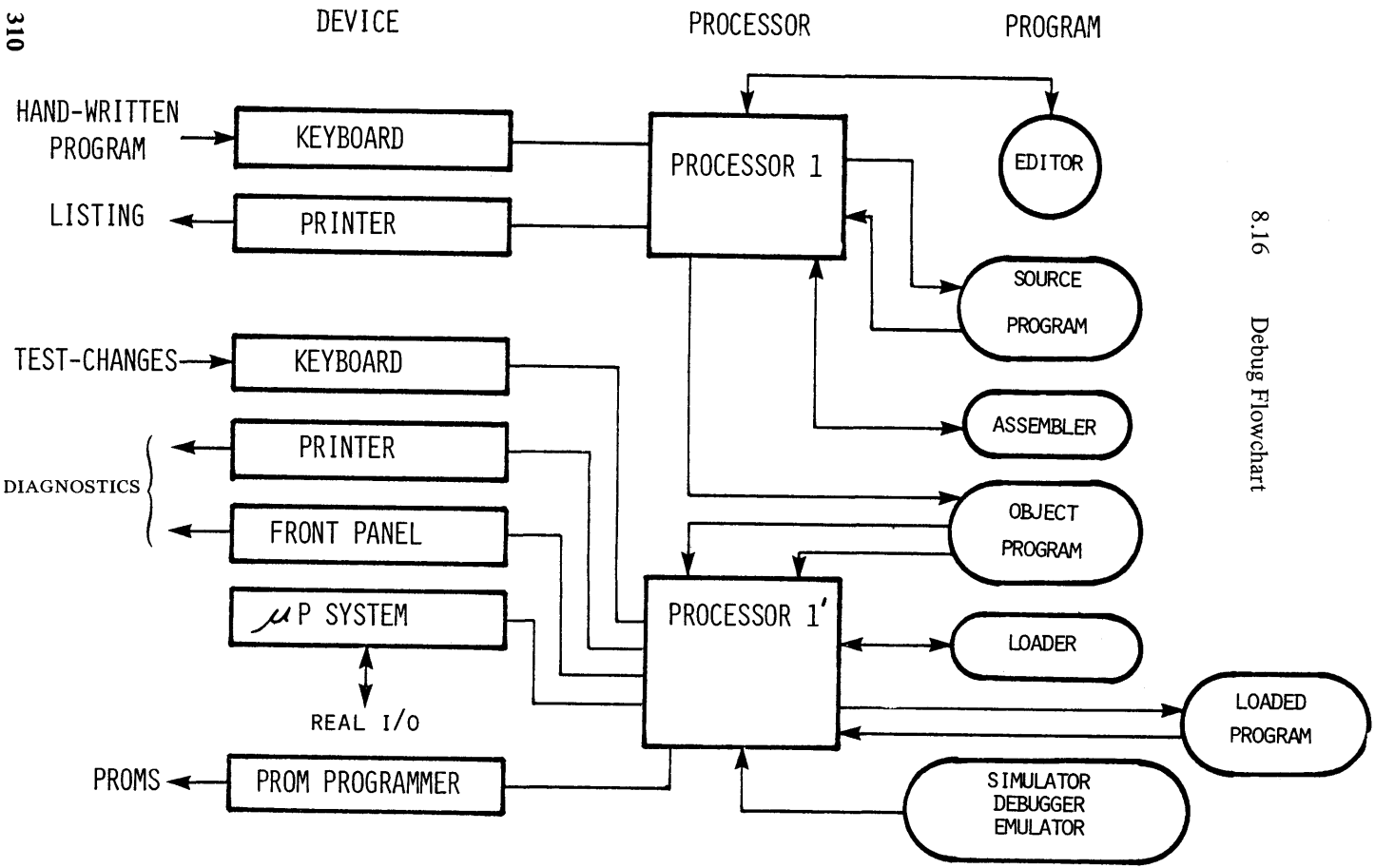
After all logic levels are verified to be reasonable, the system is ready for some *simple* test programs. Do not get too ahead of yourself here! Try simple things such as: address sequentially every possible memory location, jump to "0000" hexadecimal continually, input from a port, and output the data input to an output port. Put these tests in separate PROM's so that they can be executed individually.

The address-test should result in each of the address bit lines toggling at increasing long time with square waves.

The jump-test is so short, that it is usually possible to observe all lines with an oscilloscope to check all dynamic conditions. Also, all of the address bits—from bit A2 to bit A15—should be all zeroes in the suggested test.

The input-output will allow each input bit to be tested. If the bit is held high, the corresponding bit on the output should also go high. If it does not, there is a fault with the input-output scheme in the system, or the microprocessor.

Now it can get interesting. Try larger programs, working your way up to the final applications program. *At this point, all problems should be software ones.* If you are sure it is hardware—why? Go back and write different simple test programs to establish whether you are right or wrong. Remember: if a few instructions work OK, usually they all work OK.



8.16 Debug Flowchart

A helpful point here is that small debugging systems' software ROM's are available for most prototyping situations. They are usually called Hex (or Octal) Debug and Test Programs, or System Monitors. Fig. 8-16 illustrates the debug flow for a typical situation.

Typical Problems Unique to Micros

The following is a list of some interesting problems the authors have found:

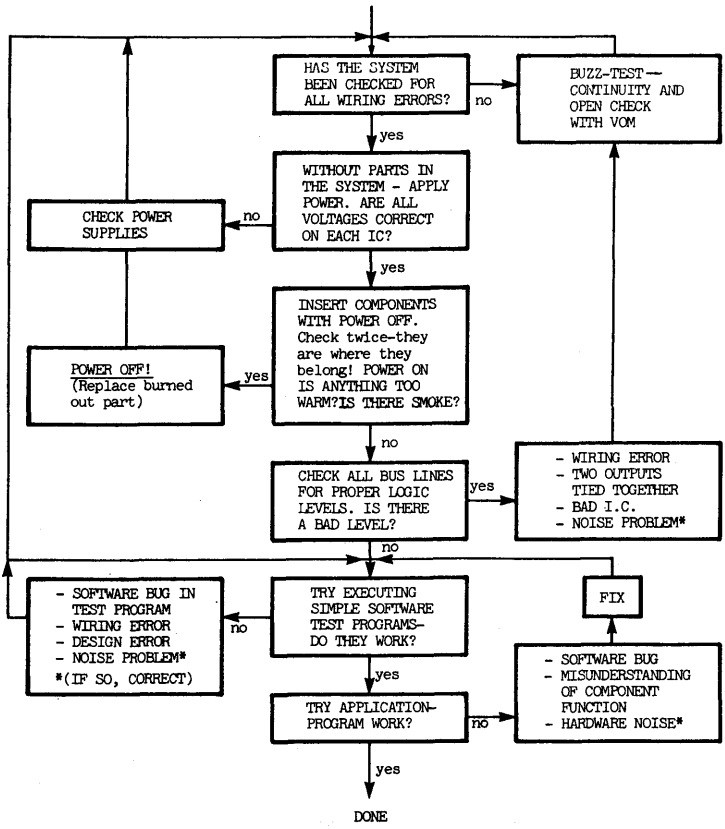
— A bad address bit on the microprocessor causing any program beyond "1FFF" hex to not execute properly.
— Excessively leaky EPROM lost its data before you could plug it back into the system from the PROM-programmer.
— PMOS and NMOS circuits cannot always be connected without buffering. This is true of all logic families. "TTL-compatible" means it will connect to TTL—*not that it connects to something else labelled TTL-compatible!* This may cause serious problems. As an example: a PMOS address-line to an NMOS RAM may cause one bit in the RAM to go bad *at random*! These problems are usually heat and power-supply sensitive. Your system should work over a wide range of temperature and over a specified power-supply range. Check all specifications closely.
— Dynamic RAM's can and do go bad at one single bit location at random. This is the reason for error-detection, parity, and error-correction in large memory systems.
— Know your buses. As a rule, connect no more than one input and one output to any bus line. Overlooking this may cause noise sensitivity problems due to overloading. The most common line that violates this rule is the RESET line.
— *Don't plug it in upside down or skewed down by one pin. Know which way is up, down, right and left.* If in doubt, measure your circuit at the socket and call the manufacturer to find where pin one is.

A trouble-shooting flowchart is present in Fig. 8-17.

THE ONE BIT IN 16,384

The multiplexer design described in Chapter 8 took six man-months to debug completely, with two full-time engineers assigned to this task, with all of the tools mentioned in this chapter available to them. Thus, the real cost of debugging this system was:

— 6 months salary: $10,000
— 6 months equipment: $15,000 (if rented)
$ 8,000 (five-year use).



8.17 Trouble-Shooting Flowchart

This section will focus on actual problems, as they were found.

Week 1:

Wire-wrapped version of design finished. Buzz-testing begun.

Week 2:

Buzz-test finished. Each module has about 20 errors out of 1000 connections. Power applied and one board had a short between power and ground. Power supply blew up. Wire found by applying large current to board with no parts in it, and "burning out" the short. It was a shorted bypass capacitor on a memory board.

Week 3:

Each board being checked for logic signals, etc., separately. Average of one more error per board found in wiring. Printed-circuit boards being made for wire-wrap modules.

Week 4:

Prototype system executes all simple test routines. Bad memory chip found in RAM boards upon a memory test program that wrote alternate ones and zeroes into every cell.

Week 5:

Bus loading problem with system program. EPROM on CPU card, a buffer added to this card. Applications program can do input and output for a while without crashing.

Week 6:

Looks like only software problems now. P.C. layouts are ready for wiring check before boards are made.

Week 7:

P.C. board layout approved, about 5 errors per board found. System has a baffling problem: will run for a few hours then give garbage to host system.

Week 8:

P.C. boards back and debugged. Replaced wire-wrap boards with P.C. boards, one at a time, to check for errors.

Week 9:

Still fixing wiring errors on P.C. boards. System still acting funny. Logic Analyzer is being used extensively to find the problem.

Week 10:

Bad bus driver on host USART card found. Now only crashes every day or so. P.C. boards finished. System will sometimes pick up improper data from terminal. In-circuit emulator being used to check the data pick-up routine on a trace-back basis. Problem only happens every 8 hours or so—thus, truly difficult to catch.

Week 11:

Argument between programmers and designers—unhealthy finger-pointing session. Friday the fault is found. Two problems.

Week 12:

There was a bad bit in the EPROM used for the program, and the carry bit was not cleared upon entering the interrupt routines, where an add with carry instruction was used, instead of an add with no carry instruction. The

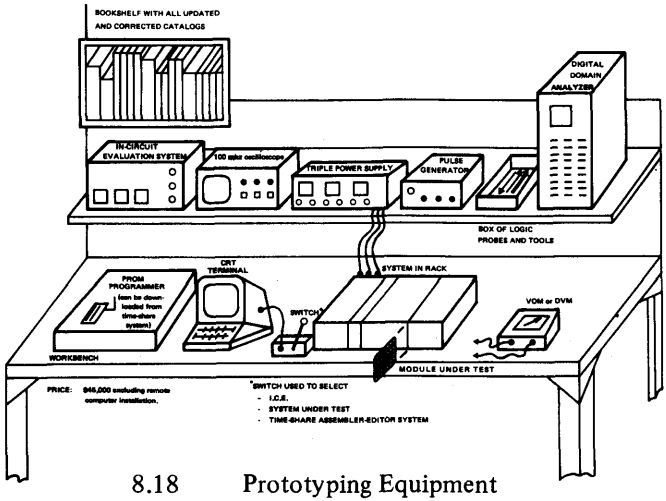
instruction determined the location of the data to be transmitted, hence it would occasionally get the wrong data upon encountering a carry set after an interrupt. The problem of the bad bit came by checking the PROM against the listing four times (it escaped detection that long!). The problem of the wrong instruction was traced back using the Logic Analyzer when it triggered on a read from the wrong place.

Epilog:

Except for statistical failures, three identical systems have been in use since the end of Week 12. There have been fewer failures in the multiplexers, with ten times less downtime, than in the main computers to which they are connected.

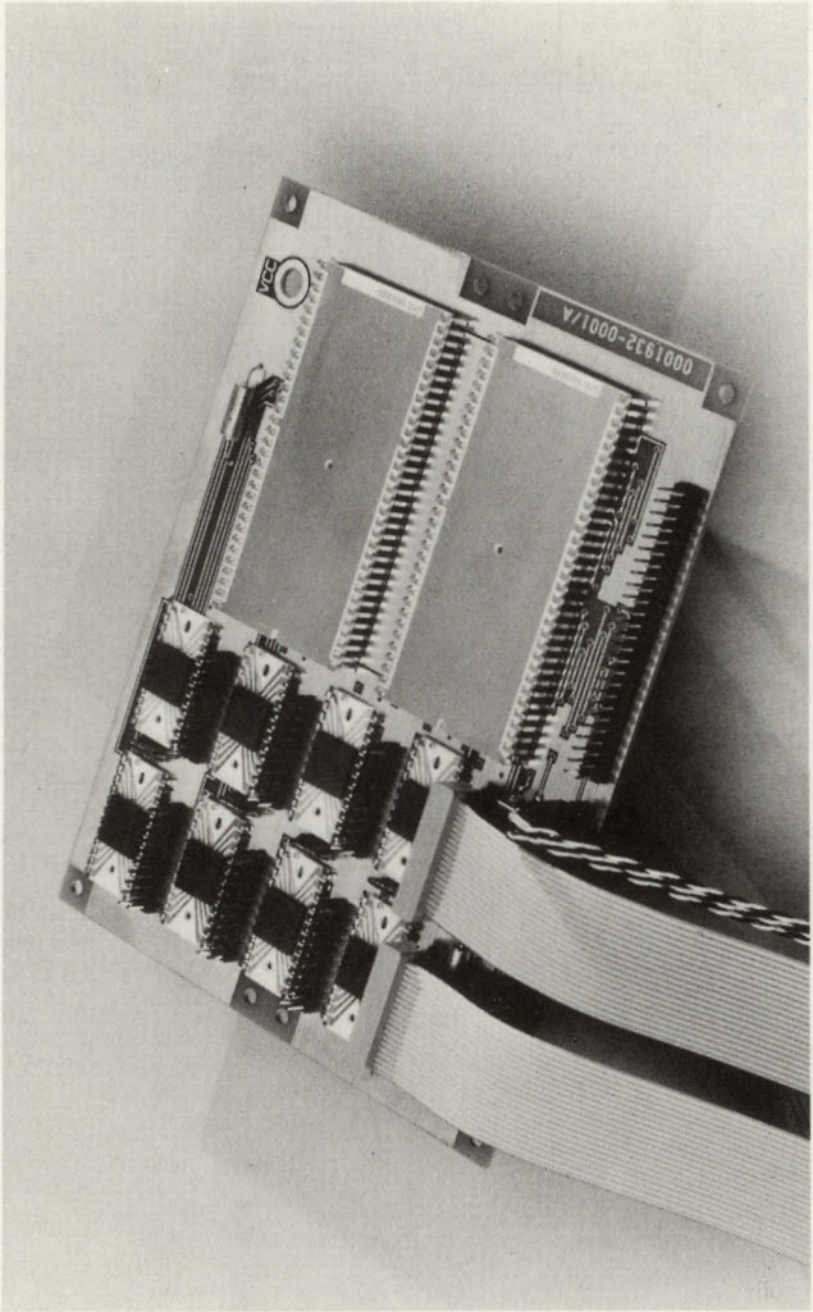
SUMMARY

Components, software, and noise are the only "things to blame" if a problem occurs. The flowcharts presented have described a simple method of approaching typical microprocessor-related problems. The equipment needed for a good microprocessor debugging station was presented, and examples of each have been given. For reference, all of the equipment required in a prototyping situation is illustrated in Fig. 8-18. Note the cost: typically $45,000. Use anything less, and the time required to fix things, or find out what is wrong, will increase.



8.18 Prototyping Equipment

Future hardware debugging tools will be oriented towards the state-type of analyzer discussed. A large number of state, trace, and trigger-capabilities, as well as the ability to format the display of the states in any machine's mnemonics will be features of the new machines. Also their use on minicomputers and large computers will become widespread with some systems including an analyzer in the unit for self-diagnosis.


VCA
0001932-0001/A

CHAPTER 9
EVOLUTION

TECHNOLOGICAL EVOLUTION

Beginning with the fundamentals of system interconnection, we have traveled through the interfacing techniques. Throughout, the direction of the evolution has been towards the use of completely integrated interfaces. The original racks, full of circuitry, previously required, have now been reduced to a small number of LSI chips. The future will bring more intelligent peripheral-chips which will result in increased performance and flexibility.

The central processor at the heart of every system, is now a single LSI chip. The interconnection of the memories and processor will be eliminated in the future by *one-chip microcomputers.* These one-chip devices will contain adequate ROM, RAM, and input-output facilities to perform most interface tasks. Such devices are already being introduced: the Texas Instrument 9940, the Intel 8048, Fairchild Mostek 3870, and others. They are characterized by a 1K to 2K ROM, plus 64 to 128 words of RAM, plus clock and timer on the MPU chip. The 16 pins freed by the unnecessary address bus become available as two 8-bit input-output ports.

The Texas Instrument 9940 is a 16-bit microcomputer with 1K ROM, RAM, and input-output in a single chip. The power of a 16-bit instruction allows the implementation of a complete instruction-set, including hardware multiply and divide. Unfortunately, the small ROM is a major limitation.

The Intel 8048 integrates a 1K by 8 PROM and a 32-byte register file on a single chip, and provides 27 lines of input-output. An EPROM version, the 8748, allows the program to be erased and reprogrammed during development. The versatility gained by using an eraseable ROM, on the same chip as the processor and input-output, makes 8748 easily adaptable to changing interface requirements.

The 8041 is a "slave-version" of the 8048, intended as a "universal peripheral interface." It can be programmed to act as any device controller, and interfaces easily to a standard microprocessor system.

The Mostek Fairchild 3870 integrates a 2K ROM, plus RAM, and is software-compatible with the F8.

PROGRAMMABLE INTERFACES

Because of the low-cost of one-chip processors, device interface chips are becoming "intelligent," i.e. processor-equipped. They receive instructions from the MPU, and implement all required control and sequencing. The decoding and sequencing are usually accomplished by a microprogram internal to the chip.

It is interesting to note that the complexity of a standard MPU is about 6000 transistors. The complexity of an FDC or CRTC is 15000 to 22000 transistors.

One-chip interfaces are special-purpose processors for device control. As integration progresses, the complete controller will eventually be shrunk in a single-chip.

COST

The cost of interfaces will probably remain higher than the cost of a processor, because of higher complexity, and lower volume. However, it has become almost negligible compared to the cost of peripherals.

"PLASTIC SOFTWARE"

As soon as a software algorithm becomes well-defined, it can now be solidified into LSI at low-cost. This is "plastic-software": programs can be purchased as a plastic LSI chip.

In the next step of evolution, it is likely that many of the algorithms or programs which have been presented throughout this book will be implemented as part of complex LSI chips. They will have become plastic software.

Interfacing will then have been essentially reduced to the simple interconnect of the required chips. When this time comes, it is hoped that the techniques presented here will contribute to understanding it.

APPENDIX A

MANUFACTURERS

AMD (Advanced Micro Devices)
901 Thompson Place
Sunnyvale, CA 94608
(408) 732-2400
Telex: 346306

AMI (American Microsystems)
3800 Homestead Road
Santa Clara, CA 95051
(408) 246-0330

DATA GENERAL
Southboro, MASS 01772
(617) 485-9100
Telex: 48460

ELECTRONIC ARRAYS
550 East Middlefield Road
Mountain View, CA 94043
(415) 964-4321

FAIRCHILD SEMICONDUCTOR
1725 Technology Drive
San Jose, CA 95110
(408) 998-0123

GI (General Instruments)
600 West John Street
Hicksville, NY 16002
(516) 733-3107
TWX: (510) 221-1666

HARRIS SEMICONDUCTOR
Box 883
Melbourne, FLA 32901
(305) 724-7430
TWX: (510) 959-6259

INTEL
3065 Bowers Avenue
Santa Clara, CA 95051
(408) 246-7501
Telex: 346372

INTERSIL
10090 North Tantau Avenue
Cuppertino, CA 95014
(408) 996-5000
TWX: (916) 338-0228

MMI (Monolithic Memories)
1165 East Arques Avenue
Sunnyvale, CA 94086
(408) 739-3535

MOS TECHNOLOGY
950 Rittenhouse Road
Norristown, PA 19401
(215) 666-7950
TWX: (510) 660-4033

MOSTEK
1215 West Crosby Road
Carollton, TX 75006
(214) 242-0444
Telex: 30423

MOTOROLA SEMICONDUCTOR
Box 20912
Phoenix, ARIZ 85036
(602) 244-6900
Telex: 67325

NS (National Semiconductor)
2900 Semiconductor Drive
Santa Clara, CA 95051
(408) 732-5000
TWX: (910) 339-9240

RAYTHEON SEMICONDUCTOR
350 Ellis Street
Mountain View, CA 94042
(415) 968-9211
TWX: (910) 379-6481

RCA SOLID STATE
Box 3200
Somerville, NJ 08876
(201) 722-3200
TWX: (718) 480-9333

ROCKWELL INTERNATIONAL
Box 3669
Anaheim, CA 92803
(714) 632-3698

SIGNETICS
811 East Arques Avenue
Sunnyvale, CA 94086
(408) 739-7700

SYNERTEK
3050 Coronado Drive
Santa Clara, CA 95051
(408) 984-8900
TWX: (910) 338-0135

TI (Texas Instruments)
Digital Systems Division
P.O. Box 1444
Houston, TX 77001
(713) 494-5115

WESTERN DIGITAL CORP.
3128 Redhill Avenue
Newport Beach, CA 92663
(714) 557-3550
TWX: (910) 595-1139

ZILOG
170 State Street
Los Altos, CA 94022
(415) 526-2748
TWX: (910) 370-7955

APPENDIX B
S-100
MANUFACTURERS

COMPUTER SYSTEMS

Byte Shop Byt-B	349.00
Computer Power & Light COMPAL-80 (assembled)	2,300.00
Cromemco Z-1 (assembled)	2,495.00
Cromemco Z-2K	595.00
Electronic Control Technology ECT-100-8080	320.00
Electronic Control Technology ECT-100-Z80	420.00
Equinox 100	699.00
Forethought Products KIMSI connector and KIM (6502)	370.00
IMSAI 8080 Computer (chassis, power, & CPU)	699.00
IMSAI PKG-1	4,444.00
IMSAI PKG-2	9,013.00
MITS Altair 8800B	875.00
Morrow's Micro Stuff Signa 100	250.00
PolyMorphic Systems POLY-88 System 0	525.00
PolyMorphic Systems POLY-88 System 2	735.00
PolyMorphic Systems POLY-88 System 6	1,575.00
PolyMorphic Disk System (1 disk)	3,250.00
Processor Technology SOL-PC Single Board	475.00
Processor Technology SOL-10 Terminal Computer	795.00
Processor Technology SOL-20 Terminal Computer	995.00
Processor Technology System I	1,649.00
Processor Technology System II	1,883.00
Processor Technology System III	4,237.00
Quay AI Z-80 CPU, SIO, PIO, ROM, Programmer Board	450.00
Technical Design Labs XITAN Alpha 1	769.00
Technical Design Labs XITAN Alpha 2	1,369.00
Vector Graphics Vector I	699.00
Vector Graphics Vector I without PROM/RAM	519.00
Vector Graphics Vector I without CPU	499.00
Vector Graphics Vector I without CPU, PROM/RAM	349.00
Western Data Systems DATA HANDLER (used MOS 6502)	179.95
Western Data Systems DATA HANDLER (bare bones)	79.95

SECOND OR REPLACEMENT CPU BOARD

Affordable Computer Products AZPU (uses Z-80)	249.00
Alpha Micro Systems AM-100 (16 bit)	1,495.00
CGRS 6502	?
Cromemco ZPU (uses Z-80/4 microprocessor)	295.00
IMSAI MPU-A (requires additional boards)	190.00
MRS AM6800 CK (uses 6800 MPU)	110.00
MRS AM6800 (without the 6800 MPU chip)	78.00
MRS AM6800 PC Board	30.00
R.H.S. Marketing Piggy-Back Z80-80 (assembled)	159.95
SD Sales Z-80 CPU	149.00
Technical Design Labs Z-80 (uses Z-80)	269.00

READ/WRITE MEMORY BOARD

Advanced Microcomputer Products Logos 8K RAM	219.95
Advanced Microcomputer Products 801C 8K RAM	207.95
Advanced Microcomputer Products 32K RAM	1,150.00
Artec 32K Memory Board (8K, 250 nS)	290.00
Artec 32K Memory Board (32K, 250 nS)	1,055.00
Associated Electronics 15K Pseudo-Static	349.95
Base-2 BKS-A	98.00
Base-2 BKS-B (450 nS)	123.00
Base-2 BKS-Z	143.00
BISI CCK Board (64K)	190.00
Crestline Micro Systems (8K, low power, assembled)	179.00
Cromemco 4KZ (4K 4MHz) (Bank selectable)	195.00
Cromemco 16KZ (16K 250 nX access and cycle)	495.00
Cybercom MB6A Blue Board (8K static)	250.00
Cybercom MB7 (16K low power static)	525.00
Data Sync 16K (assembled)	298.00
Duston 8K Memory Board (bare)	29.00
Dutronics 4KLST (4K low power static)	139.00
Dutronics 8KLST (8K low power static)	285.00
E.E. & P.S. 8K (8K static)	295.00
E.E. & P.S. 16K (16K dynamic)	599.00
E.E. & P.S. 32K (32K dynamic)	895.00
Electronic Control Technology 8KM (8K 215 nS)	295.00
Electronic Control Technology 16K RAM (16K static)	555.00
Electronic Control Technology 16K RAM (with only 4K)	169.00
Electronic Control Technology 16K RAM (with only 8K)	295.00
Electronic Control Technology 16K RAM (with only 12K)	425.00
Extensys RM64-32 (32K)	895.00
Extensys RM64-48 (48K)	1,195.00
Extensys RM64-64 (64K)	1,495.00
Franklin Electric 8K Static RAM	225.00
Godbout Econoram (4K static)	99.95

Godbout Econoram II (8K)	163.84
IMSAI RAM 4A-4 (4K without sockets)	139.00
IMSAI RAM 4A-4 (4K with sockets)	159.00
IMSAI 65K (dynamic)	2,599.00
IMSAI 32K (dynamic)	749.00
IMSAI 16K (dynamic)	449.00
Kent-Moore 4K (assembled)	107.00
Microdesign MR8 (EPROM/RAM)	124.95
Micromation JUMP START (4K static)	145.00
Midwest Scientific Instruments PROM/RAM Board	95.00
Mikro-D MD-2046-4 (4K static)	205.00
Mikro-D MD-2046-8 (8K static)	345.00
Mikro-D MD-2046-12 (12K static)	485.00
Mikro-D MD 2046-16 (16K static)	625.00
MiniMicroMart C-80-4K-100 (4K blank board)	39.95
MiniMicroMart C-80-4K-700 (4K blank board plus)	49.95
MiniMicroMart C-80-4K-300S (4K 2102)	79.95
MiniMicroMart C-80-4K-300LP (4K 9IL02A)	99.95
MiniMicroMart C-80-4K-350LP (4K 9IL02C)	129.95
MiniMicroMart C-80-16K-300 (16K E MM4200)	479.95
MITS 88-4MCS (4K static)	167.00
MITS 88-16MCS (16K static)	765.00
MITS 88-S4K (4K dynamic)	155.00
Morrow Intelligent Cassette (512 static)	96.00
Mountain Hardware PROROM (256)	164.00
Omni (16K static)	459.00
Omni with paging option (16K static)	468.00
Prime Rodi x 40K (dynamic)	1,490.00
Prime Rodi x 48K (dynamic)	1,580.00
Prime Rodi x 56K (dynamic)	1,670.00
Prime Rodi x 64K (dynamic)	1,750.00
Processor Technology 4KRA (4K static with sockets)	154.00
Processor Technology 8KRA (8K static with sockets)	295.00
Processor Technology 16KRA (16K static assembled)	529.00
PolyMorphic Systems MEM-8K (8K static)	300.00
R.H.S. Marketing DYNABYTE 16K (dynamic, assembled)	485.00
J-K Electronics DYNA-RAM 16 (16K)	339.00
S. D. Sales Company 4K (4K static)	89.95
Seals Electronics 8KSC-8 (8K static)	269.00
Seals Electronics 8KSC-Z (8K 250 nS)	295.00
Seals Electronics 8KSCL M (less memory chips)	124.00
Seals Electronics 16KSC-16 (16K static)	579.00
Solid State Music M8-4 (4K 9IL02A)	129.95
Solid State Music M8-4 (8K 9IL02A)	209.00
Solid State Music M8-4 (board only)	30.00
Solid State Music M8-4 (board only)	35.00
Solid State Music M8-6 (8K 9IL02APC static)	265.00
Solid State Music M8-7 (16K static)	525.00
Technical Design Labs Z8K (4K 215 nS)	169.00
Technical Design Labs Z8K (8K 215 nS)	295.00

Technical Design Labs Z12K (12K 215 nS)	435.00
Technical Design Labs Z16K (16K 215 nS)	574.00
Technical Design Labs Z Monitor Board with 2K RAM	295.00
Vandenberg 16K RAM (dynamic)	299.00
Vector Graphics 8K RAM	265.00
Vector Graphic Reset and Go PROM/RAM	89.00
Xybek PRAMMER (256 bytes & 1702 PROMs)	189.00

PROM PROGRAMMER BOARD

Cromemco BYTESAVER for 2704 & 2708	145.00
Mountain Hardware PROROM (AMI 6834)	164.00
Quay AI Z-80 with 2708 Programmer	450.00
Szerlip Enterprises The Prom Setter (1702A and 2708)	165.00
Xebek PRAMMER for 1702 (with 1702 & RAM)	209.00

PLUG IN SOFTWARE BOARD

Computer Kits Power-Start	165.00
Cromemco Z80 Monitor Board with PROM Programmer	220.00
Godbout 8080 Software Board	189.95
Microdesign MR8 with MM2K	224.45
Micronics Better Bug Trap (assembled)	180.00
Midwest Scientific Instruments PROM/RAM Monitor	245.00
Mountain Hardware PROROM	164.00
National Multiplex Corp No. 2 SIO with monitor	140.00
Processor Technology ALS-8 (assembled)	425.00
Processor Technology ALS-8 with SIM-I	520.00
Processor Technology ALS-8 with TXT-I	520.00
Technical Design Labs Z System Monitor Board	295.00
Vector Graphics Reset and Go (2 1702A)	129.00
Vector Graphics Reset and Go (3 1702A)	159.00

SERIAL INTERFACE BOARD

Advanced Microcomputer Products (3P + S compatible)	125.00
Cromemco TU-ART (2 parts)	195.00
IMSAI SIO 2-1 (one part, without cables)	125.00
IMSAI SIO 2-2 (two parts, without cables)	156.00
IMSAI SIO (serial, parallel, & tape interface)	195.00
Morrow Intelligent Cassette with one part	108.00
MiniMicroMart C80-SI/O-300 (TTL)	44.95
MITS 88-2SIO (one part)	150.00
MITS 88-2SIO + SP (two parts)	188.00
MITS 88 SIOB	124.00
National Multiplex Corp No. 2 SIO with ROM	140.00

Processor Technology 3P+S (with sockets)	149.00
Solid State Music I/O-2 (two parts)	47.50
Solid State Music I/O-2 (PC board only)	25.00
Technical Design Labs Z Monitor Board (two parts)	295.00
WIZARD PSIOB (3P+S compatible)	125.00

ANALOG INTERFACE BOARD

Cromemco D+7AIO (7analog inputs & 7 outputs)	145.00
Micro Data ADC/DAC	250.00
MITS 88-ADC (assembled only)	524.00
MITS 88-Mux (assembled only)	319.00
MITS AD/DA (assembled)	235.00
PolyMorphic Systems ADA/I (1 analog output)	145.00
PolyMorphic Systems ADA/2 (2 analog outputs)	195.00

MODEM BOARD

International Data Systems 88-MODE M	199.00
Hayes 80-103A (assembled)	279.95
Hayes 80-103A (board only)	49.95

AUDIO CASSETTE INTERFACE BOARD

Affordable Computer Products Triple Standard	135.00
DAJEN Cassette Interface	120.00
DAJEN Universal Cassette Interface (Relay Control)	135.00
IMSAI MIO (tape interface, parallel, & serial)	195.00
MiniTerm Associates MERLIN with cassette interface	298.00
MITS 88-ACR	145.00
National Multiplex Corp No. 2 SIO with ROM	140.00
Morrow Intelligent Cassette Interface	96.00
Morrow Intelligent Cassette Interface (3 drives)	102.00
PerCom Data CI-812	89.95
Processor Technology CUTS	87.00
RO-CHE with Tarbell (two parts)	245.00
RO-CHE with Tarbell (four parts)	245.00
Tarbell	120.00

TAPE DRIVE INTERFACE BOARDS

MECA ALPHA-I System	400.00
Micro Design Model 100 (assembled)	600.00
Micro Design Model 200 (assembled)	875.00
MicroLogic M712 DG PhiDeck	69.95
National M.C. 2 SIO (R) 1 ROM	169.95

National M.C. 2 SIO (R) 2 ROM	189.95
National M.C. 2 SIO (R) with 3M3 (3M drive)	369.90
National M.C. 2 SIO (R) with 3M3 (mini 3M drive)	339.90

FLOPPY DISK INTERFACE BOARD

Alpha Micro Systems AM-200 Controller	695.00
Alpha Micro Systems AM-201 Controller	695.00
CHP Floppy Disk Controller	300.00
Computer Hobbyist Products Controller	300.00
Computer Hobbyist Products (single drive)	850.00
DigiComm 8040 Floppy Disk Controller	265.00
Digital Systems IBM campatible	1,595.00
Digital Systems dual IBM campatible	2,170.00
iCOM Microfloppy Model FD2411 (assembled)	1,095.00
IMSAI F IF	599.00
IMSAI F DC2-1 & F IF	1,694.00
IMSAI F DC2-2 & F IF	2,789.00
INFO 2000 Adapter (without RAM)	120.00
INFO 2000 Adapter (with 4K RAM)	160.00
INFO 2000 Adapter - Per Sci 1070 Controller	860.00
Micromation Universal Disc Controller	229.00
Micromation MACRO DISC System, Model 164K	900.00
Micromation MACRO DISC System, Model 256K	1,100.00
Micropolis 1053 Mod II (630K)	1,795.00
Micropolis 1043 Mod II (315K)	1,095.00
Micropolis 1053 Mod I (286K)	1,545.00
Micropolis 1043 Mod I (143K)	945.00
MITS 88-DCDD (Controller & disk)	1,425.00
MITS 88-DISK	1,215.00
North Star Computers MICRO-DISK	699.00
PerCom Data Co.	695.00
Peripheral Vision interface and floppy	750.00
Peripheral Vision IFF-KC interface	245.00
Pertec RD2411	1,095.00
Processor Applications FDC-1016K Controller	395.00
Processor Technology Helios (dual)	1,895.00
Realistic Controls ZI/25	1,095.00
Synetic Designs interface and floppys	2,690.00
Tarbell Bare Board Interface	40.00
Tarbell Interface	190.00

HARD DISK INTERFACE BOARD

IMSAI DISK-50	12,500.00
IMSAI DISK-80	14,700.00
IMSAI DISK-200	24,500.00
IMSAI Interface (assembled)	3,900.00

PROM BOARD

Crea Comp M 100/16 (16K, 2116)	485.00
Crea Comp M 100/16 (with parity)	560.00
Crea Comp M 100/32 (32K, 2116)	885.00
Crea Comp M 100/32 (with parity)	990.00
Cromemco BYTESAVER (8K)	145.00
Cromemco 16KPR-K (16K, Bank selectable)	145.00
DigiComm Byteuser (uses 2708)	65.00
Digiteck PROM CARD (2K assembled without PROMS)	56.95
Electronic Control Technology 2K ROM/2K RAM	120.00
Godbout Econoram (2K)	135.00
Godbout Econoram (4K)	179.95
Godbout Econoram (8K)	269.00
IBEX 16K PROM Board	85.00
IMSAI PROM 4-4 (4K PROM)	399.00
IMSAI PROM 4-512 (1/2K PROM)	165.00
Microdesign MR8 (for 2708)	99.50
Midwest Scientific Instruments PROM/RAM Board	95.00
MiniMicroMart C80-1702-1 (all except PROMS)	49.95
MiniMicroMart C80-2708-2 (all except PROMS)	49.95
MiniMicroMart C80-256 (boot strap board, fuse link)	34.95
MITS PMC (2K)	85.00
Processor Technology 2KRO	65.00
Seals Electronics 4KROM	119.00
Solid State Music MB-3 2K (8 1702As)	105.00
Solid State Music MB-3 4K (16 1702As)	145.00
Solid State Music MB-3 (without PROMs)	65.00
Solid State Music MB-8 (2708)	85.00
Vector Graphic Reset and Go PROM/RAM	89.00
Xybek PRAMMER for 1702 (with a 1702 & RAM)	189.00

MEMORY CONTROL BOARD

IMSAI IMM ROM Control Kit	299.00
IMSAI IMM EROM Control Kit	499.00

HARDWARE MULTIPLY/DIVIDE BOARD

GNAT 8006 Module (5 u-sec. process time)	225.00
GNAT 8006 Module (2.5 u-sec. process time)	275.00
North Star Computers (floating point)	359.00

CALCULATOR INTERFACE BOARD

COMPU/TIME CT 100	195.00
COMPU/TIME C 101	149.00
MiniMicroMart C80-SCI-300	99.95

SPEECH SYNTHESIZER BOARD

Ai Cybernetic Systems Model 1000	325.00
Computalker Speech Synthesizer CT-I	395.00
Logistics Synthesizer (multipurpose)	525.00

SPEECH RECOGNITION BOARD

Heuristic Speechlab	245.00
Phonics SR/8 (assembled)	550.00

JOYSTICK INTERFACE KITS

Cromemco Joystick Kit & D+7AI/O	210.00
Cromemco Dual Joystick Kits & D+7AI/O	275.00

INTERRUPT BOARD

Cromemco TU-ART	195.00
El Paso Computer Group (board only)	20.00
IMSAI PIC-8 (with internal clock)	125.00
MITS 88-VI/RTC	136.00

REAL-TIME CLOCK

Comptek CL2400	98.00
COMPU/TIME CT 100	195.00
COMPU/TIME T 102	165.00
International Data Systems SMP-88	96.00
Lincoln Semiconductor Clock and Display Driver	95.00

AC POWER CONTROL

Comptek PC3216 Control Logic Interface	189.00
Comptek PC3216 & PC3202 Power Control Unit	228.50
Comptek PC3216 & 16 PC3202 16 Channel System	821.00
Comptek PC3232 Control Logic Interface	299.00
E.E. & P.S. II5V I/O	249.00
Mullen Relay/Opto Isolator Control Board	117.00

BATTERY BACK-UP BOARD

Seals Electronics BBUC (12 amper hours)	55.00
E.E. & P.S.	55.00

MUSIC SYNTHESIZER BOARD

ALF Quad Cromatic Pitch Generator (1 channel)	111.00
ALF Quad Cromatic Pitch Generator (2 channels)	127.00
ALF Quad Cromatic Pitch Generator (3 channels)	143.00
ALF Quad Cromatic Pitch Generator (4 channels)	159.00
Cybercam 581 Synthesizer Kit	250.00
Galazy Systems MG-1	299.00
Logistics Synthesizer (multipurpose)	525.00
SRS Polyphonic Synthesizer SRS-320 (assembled)	175.00
SRS Polyphonic Synthesizer SRS-321 for the SRS-320	175.00

PRINTER INTERFACE BOARD

Peripheral Vision PRT-KC Printer Kit	495.00

FREQUENCY COUNTER BOARD

International Data Systems 88-UFC	149.00

IBM SELECTRIC INTERFACE BOARD

Micromation TYPEAWAY	225.00

PARALLEL INTERFACE BOARD

Advanced Microcomputer Products (3P+S compatible)	125.00
Cromemco D+7AIO (one part with seven analog parts)	145.00
Cromemco TU-ART (2 parts)	195.00
IMSAI PIO 4-I (one port without cables)	93.00
IMSAI PIO 4-I & PIOM (two ports without cables)	115.00
IMSAI PIO 4-I & PIOM (three ports without cables)	137.00
IMSAI PIO 4-4 (four ports without cables)	156.00
IMSAI PIO 6-3 (three ports and bus without cables)	139.00
IMSAI PIO 6-6 (six ports and bus without cables)	169.00
IMSAI MOI (two ports & serial & tape interface)	195.00
MicroLogic M712 (one port)	69.95
MiniMicroMart C80-P I/O (two ports)	49.95
MiniMicroMart C80-P I/O with cables C80-P I/O-540	57.45
MITS 88-4PIO (one port)	105.00
MITS 88-4PIO + PP (two ports)	148.00
MITS 88-4PIO + 2PP (three ports)	191.00
MITS 88-4PIO + 3PP (four ports)	234.00
Morrow Intelligent Cassette with one port	102.00
PolyMorphic VTI/32 (one input port with video)	185.00
PolyMorphic VTI/64 (one input port with video)	210.00

Processor Technology 3P+S (with sockets)	149.00
Solid State Music I/O-1 (one port)	42.00
Solid State Music I/O-1 (PC board only)	25.00
Solid State Music I/O-2 (two ports)	47.50
Solid State Music I/O-2 (PC board only)	25.00
Technical Design Labs Z Monitor Board (one port)	295.00
WIZARD PSIOB (3P+S compatible)	125.00

PROTOTYPE BOARD

Advanced Microcomputer Products Universal Proto	39.95
Artec GP-100	20.00
Cromemco WWB-2K	35.00
Electronic Control Technology PB-1	22.00
E.E. & P.S. Wire Wrap	39.00
E&L Instruments Breadboarding/Interfacing Station	241.50
Electronic Control Technology PB-1	28.00
Galaxy Systems PB-1	30.00
Harnestead Technology HTC-88P (QT sockets)	138.00
Harnestead Technology HTC-88PF (fell pattern)	38.00
IMSAI GP-88	39.80
IMSAI 88C-5 & P106-6 Intelligent Breadboard System	699.00
IMSAI 88C-3 & P106-3 Intelligent Breadboard System	464.00
MiniMicroMart C-80-WW (wire wrap type)	19.95
MiniMicroMart C-80-DIP (for point to point)	18.95
MiniMicroMart C-80-BUS-WW (wire wrap)	21.95
MiniMicroMart C-80-BUS-WW-125 (with components)	27.45
MiniMicroMart C-80-DIP-BUS (for point to point)	20.95
MiniMicroMart C-80-DIP-BUS-125 (with components)	26.45
MITS 88-PPCB	45.00
MITS 88-WWB	20.00
PolyMorphics Poly I/O	55.00
Processor Technology WWB	40.00
Sargent's Dist. Co.	25.00
Seals Electronics WWC	37.50
Tarbell Electronics	28.00
Vector 8800V	19.95
Vector 8800-A	29.95
Vector 8800-B	89.00

EXTENDED BOARD

Advanced Microcomputer Products Extender	34.95
Artec EXT-100	12.00
Cromemco EXC-2	35.00
E.E. & P.S. Extender W/C	34.00
Galaxy Systems EX-1	25.00
IMSAI EXT	39.00
MiniMicroMart C-80-EXC	24.95

Mullen (with logic probe)	35.00
Processor Technology EXB	35.00
Seals Electronics EXT	29.00
Solid State Music (less connectors)	8.00
Solid State Music (w/w connector)	12.50
Suntronics EXT-I	9.95
Vector 3690-12 (assembled)	25.00

ADAPTER BOARD

MiniMicroMart C80-8A (for MOD 8/C-MOD 80 boards)	19.95
Forethought Products KIMSI (for KIM)	125.00

CARD CAGE AND/OR MOTHERBOARD

Advanced Microcomputer Products 8 slot MS w/connectors	79.95
Byte, Inc. Byt-8	229.00
Computer Data Systems Versatile CRT (assembled)	699.95
Electronic Control Technology ECT-100	100.00
Electronic Control Technology MB-20	60.00
Godbout Motherboard (10 slot)	85..00
Godbout Motherboard (18 slot)	118.00
Integrand Research Corp. 808	200.00
Integrand Research Corp. 808A	275.00
MiniMicroMart Expander (4 slots)	10.95
MiniMicroMart Expander (9 slots)	17.95
Morrow MotherBoard	76.00
Objective Design Crate Book (plans only)	19.95
PolyMorphic P+S Chassis	235.00
TEI Model MCS-112	316.00
T&H Engineering Low Cost Buses	149.00
Vector 18 Slot Motherboard	49.00

TERMINATION BOARD

Godbout	25.00

VIDEO INTERFACE BOARD - BLACK & WHITE

Computer Kits INTELLITERM (characters)	395.00
Computer Graphics GDT-I (graphics and light pen)	185.00
Environmental Interface II (monitor)	245.00
Environmental Interface III (oscilloscope)	495.00
Kent-Moore alpha (assembled)	107.00
Kent-Moore graphic (assembled)	137.00

Micro GRAPHICS "THE DEALER" (graphics and characters)	249.00
MiniMicroMart C80-VBA	149.95
MiniTerm Associates MERLIN (without memory)	269.00
MiniTerm Associates MERLIN (with memory)	303.95
MiniTerm Associates MERLIN Super Dense Graphics	308.00
Polymorphics VTI/64 (graphics and characters)	210.00
Processor Technology VDM-I (characters)	199.00
Solid State Music 64 x 16 (graphics and characters)	179.95

VIDEO INTERFACE BOARD - COLOR

Cromemco TV DAZZLER (graphics)	215.00

TV CAMERA INTERFACE BOARD

Cromemco 88-CCC-K	195.00
Cromemco 88-CCC-K with Camera Kit 88-ACC-K	390.00
Environmental Interface I	295.00
Environmental Interface with camera	595.00

Affordable Computer Products
Byte Shop No. 2
3400 El Camino Real
Santa Clara, CA 95051
(408) 249-4221

Advanced Microcomputer Products
P.O. Box 17329
Irvine, CA 92713
(714) 558-8813

Ai Cybernetic Systems
P.O. Box 4691
University Park, NM 88003

ALF Products, Inc.
128 S. Taft
Lakewood, CO 80228

Alpha Micro Systems
17875 N. SkyPark North
Irvine, CA 92714
(714) 957-1404

Altair (see MITS)

Artec Electronics, Inc.
605 Old Country Road
San Carlos, CA 94070
(415) 592-2740

Associated Electronics
12444 Lambert Circle
Garden Grove, CA 92641
(714) 539-0735

Base-2, Inc.
P.O. Box 9941
Marina del Rey, CA 90291

Byte Shop
1450 Koll Circle, No. 105
San Jose, CA 95112

CGRS Microtech, Inc.
Unknown

CHP, Inc.
P.O. Box 18113
San Jose, CA 95158

Comptek
P.O. Box 516
La Canada, CA 91011
(213) 790-7957

Computalker Consultants
P.O. Box 1951
Santa Monica, CA 90406

Computer Data Systems
English Village, Atram 3
Newark, DE 19711

Computer Kits Inc.
1044 University Avenue
Berkeley, CA 94710
(415) 845-5300

Computer Graphics Associates
56 Sicker Road
Latham, NY 12110

Computer Hobbyist Products, Inc.
P.O. Box 18113
San Jose, CA 95158
(408) 629-9108

COMPU/TIME
P.O. Box 417
Huntington Beach, CA 92648
(714) 638-2094

Computer Power & Light
12321 Ventura Blvd.
Studio City, CA 91604
(213) 760-0405

Crea Comp System, Inc.
Suite 305
4175 Veterans Highways
Ronkonkoma, NY 11779
(516) 585-1606

Crestline Micro Systems
P.O. Box 3313
Riverside, CA 92519

Cromemco
2432 Charleston Road
Mountain View, CA 94043
(415) 964-7400

Cybercom
2102A Walsh Avenue
Santa Clara, CA 95050
(408) 246-2707

DAJEN
David C. Jenkins
7214 Springleaf Court
Citrus Heights, CA 95610
(916) 723-1050

Data Sync
201 W. Mill
Santa Maria, CA 93454
(805) 963-8678

DigiComm
6205 Rose Court
Roseville, CA 95678

Digital Systems
1154 Dunsmuir Place
Livermore, CA
(415) 413-4078

Digiteck
P.O. Box 6838
Grosse Point, Michigan 48236

Duston, Forrest
885 Aster Avenue
Palatine, Il 60067

Dutronics
P.O. Box 9160
Stockton, CA 94608

E & L Instruments, Inc.
61 First Street
Derby, Conn. 06418
(203) 735-8774

E.E. & P.S.
Electronic Eng. & Production Service
Route No. 2
Louisville, Tennessee
(615) 984-9640

Electronic Control Technology
P.O. Box 6
Union City, NJ 07083

El Paso Computer Group
9716 Saigon Drive
El Pase, TX 79925

Environmental Interfaces
3207 Meadowbrook Blvd.
Cleveland, Ohio 44118
(216) 371-8482

Equinox Division
Parasitic Engineering
P.O. Box 6314
Albany, CA 94706
(800) 648-5311

Extensys Corp.
592 Weddell Drive, S-3
Sunnyvale, CA 94086
(408) 734-1525

Forethought Products
P.O. Box 386-A
Coburg, Oregon 97401

Franklin Electric Co.
733 Lakefield Road
Westlake Village, CA 91361
(805) 497-7755

Galaxy Systems
P.O. Box 2475
Woodland Hills, CA 91364
(213) 888-7233

GNAT Computers
8869 Balboa, Unit C
San Diego, CA 12123

Godbout Electronics
Box 2355
Oakland Airport, CA 94614

Hayes
P.O. Box 9884
Atlanta, GA 30319
(404) 231-0574

Heuristic, Inc.
900 N. San Antonio Road
Suite C-1
Los Altos, CA 94022

Hornestead Technologies Corp.
891 Briarcliff Road N.E.
Suite B-11
Atlanta, GA 30306

iCOM Division
6741 Variel Avenue
Conoga Park, CA 91303
(213) 348-1391

IBEX
1010 Morse Avenue, No. 5
Sunnyvale, CA 94086
739-3770

I M S Associates, Inc.
14860 Wicks Blvd.
San Leandro, CA 94577
(415) 483-2093

INFO 2000
P.O. Box 316
Culver City, CA 90230

Integrand Research Corp.
8474 Avenue 296
Visalia, CA 93277
(209) 733-9288

International Data Systems
400 North Washington Street,
Suite 200
Falls Church, VA 22046
(703) 536-7373

Kent-Moore Instrument Co.
P.O. Box 507
Industrial Avenue
Pioneer, Ohio 43554
(419) 737-2352

Lewis and Associates
68 Post Street, Suite 506
San Francisco, CA 94104
(415) 391-1498

Lincoln Semiconductor
P.O. Box 68
Milpitas, CA 95035
(408) 734-8020

Logistics
Box 9970
Marina Del Rey, CA 90291

North Star Computers
2465 Fourth Street
Berkeley, CA 94710

MECA
7344 Warnego Trail
Yucca Valley, CA 92284
(714) 365-7686

Micro Data
3199 Trinity Place
San Jose, CA 95124

Microdesign
8187 Havasu Circle
Buena Park, CA 90621
(415) 465-1861

Micro Designs, Inc.
499 Embarcadero
Oakland, CA 94606
(415) 465-1861

MicroGRAPHICS
P.O. Box 2189, Station A
Champaign, IL 61820

MicroLogic
P.O. Box 55484
Indianapolis, IN 46220

Micromation
524 Union Street
San Francisco, CA 94133
(415) 398-0289

Micronics, Inc.
P.O. Box 3514
Greenville, NC 27834

Micropolis Corp.
9017 Reseda Blvd.
Northridge, CA 91324

Midwest Scientific Instruments
220 West Cedar
Olathe, Kansas 66061

MIKRA-D, Inc.
P.O. Box 403
Hollister, Mass. 01746

Mini Micro Mart
1618 James Street
Syrecuse, NY 13203

MiniTerm Associates
Box 268
Bedford, Mass. 01730

MITS (Altair)
2450 Alamo S. E.
Albuquerque, NM 87106

Morrow's Micro-Stuff
Box 6194
Albany, CA 94706

MRS
P.O. Box 1220
Hawthorne, CA 90250

Mullen Computer Boards
Box 6214
Hayward, CA 94545

Mountain Hardware
Box 1133
Ben Lamand, CA 95005

National Multiplex Corp.
3474 Rand Avenue, Box 288
South Plainfield, NJ 07080

Objective Design, Inc.
P.O. Box 7536 Univ. Station
Provo, Utah 84602

PerCom Data Company
4021 Windsor
Garland, TX 75042

Peripheral Vision
P.O. Box 6267
Denver, Colorado 80206

Phonics, Inc.
P.O. Box 62275
Sunnyvale, CA 94086

Polymorphic Systems
737 S. Kellogg
Galeta, CA 94608

Prime Rodix Inc.
P.O. Box 11245
Denver, Colorado 80211

Processor Applications, Ltd.
2801 East Valley Veiw Avenue
West Covina, CA 91792

Processor Technology
6200-L Hollis Street
Emeryville, CA 94608

Quay Corporation
P.O. Box 386
Freehold, NJ 07728

Realistic Controls Corporation
3530 Warrensville Center Road
Cleveland, Ohio 44122

R.H.S. Marketing
2233 El Camino Real
Palo Alto, CA 94306

RO-CHE Systems
7101 Mammoth Avenue
Van Nuys, CA 91405

S. D. Sales
P.O. Box 28810
Dallas, Texas 75228

Sargent's Dist. Co.
4209 Knoxville
Lakewood, CA 90713

Scientific Research Instruments
P.O. Drawer C
Marcy, NJ 13403

Seals Electronics
Box 11651
Knoxville, TN 37919

Smoke Signal Boardcasting
P.O. Box 2017
Hollywood, CA 90028

Solid State Music
MIKOS
419 Portofino Drive
San Carlos, CA 94070

Stillman Research Systems (SRS)
P.O. Box 14036
Phoenix, AZ 85063

Suntronics Company
360 Merrimack Street
Lawrence, MA 01843

Synetic Designs Company
P.O. Box 2627
Pomona, CA 91766

Szerlip Enterprises
1414 W. 259th Street
Harbor City, CA 90710

TEI Inc.
7231 Fondren Road
Houston, Texas 77036

T&H Engineering
P.O. Box 352
Cardiff, CA 92007

Tarbell Electronics
20620 South Leapwood Avenue
Suite P
Carson, CA 90746

Technical Design Labs Inc.
342 Columbus Avenue
Trenton, NJ 08629

Vandenberg Data Products
P.O. Box 2507
Santa Maria, CA 93454

Vector Electronics Company, Inc.
12460 Gladstone Avenue
Sylmar, CA 91342

Vector Graphic Inc.
717 Lakefield Road, Suite F
Westlake Village, CA 91361

Western Data Systems
3650 Charles Street, No. Z
Santa Clara, CA 95050

WIZARD Engineering
8205 Ronson Road, Suite C
San Diego, CA 92111

Xybek
P.O. Box 4925
Stanford, CA 94305

INDEX

S

MICROPROCESSOR BOOKS

BOOKS

C200	AN INTRODUCTION TO PERSONAL COMPUTING, by Rodnay Zaks
C201	MICROPROCESSORS, from chips to systems, by Rodnay Zaks
C4	LES MICROPROCESSEURS: du composant au systeme, par Rodnay Zaks et Pierre Le Beux
C207	MICROPROCESSOR INTERFACING TECHNIQUES, by Austin Lesea and Rodnay Zaks
MD	INTERNATIONAL MICROPROCESSOR DICTIONARY (10 languages)

CASSETTES (2 cassettes plus special book)

S1	INTRODUCTION TO MICROPROCESSORS
S2	PROGRAMMING MICROPROCESSORS

SEMINAR BOOKS

B1	MICROPROCESSORS
B2	PROGRAMMING AND MICROPROGRAMMING
B3	MILITARY MICROPROCESSOR SYSTEMS
B5	BIT-SLICE
B6	INDUSTRIAL MICROPROCESSOR SYSTEMS
B7	INTERFACING TECHNIQUES

IN HOUSE TRAINING AND SEMINARS

SYBEX offers over 12 different seminars which can be presented at your facility for a minimum group of 15 participants (world-wide). Please contact the nearest SYBEX office for full details.

INFORMATION REQUEST (see other side)

FIRST CLASS
Permit No. 2587
BERKELEY, CA

BUSINESS REPLY MAIL
No postage stamp necessary if mailed in the United States

Postage will be paid by

SYBEX INCORPORATED
2161 SHATTUCK AVENUE
BERKELEY, CA 94704

INFORMATION REQUEST

NAME_____ **POSITION**_____

COMPANY_____ **ADDRESS**_____

CITY_____ **STATE**_____ **ZIP**_____ **TEL:**_____

Send me information on:

- ☐ **BOOKS**
- ☐ **HOME STUDY WITH CASSETTES**
- ☐ **SEMINARS**
- ☐ **IN-HOUSE COURSES**
- ☐ **CONSULTING**
- ☐ **OTHER**

☐ **IMMEDIATELY**

FOLD HERE, THEN STAPLE

CUT HERE

C207

SYBEX

INTERFACING
is no longer an art, but a set of techniques and components. This book will teach you how to interconnect a complete system, and interface it to all the usual peripherals. It covers hardware and software skills and techniques, including the use and design of model buses such as the IEEE 488 or S100.

FROM KEYBOARD TO FLOPPY-DISK

SYBEX provides technical education and training in the microprocessor field. Write for a complete list of publications and for in-house programs.

INTERFACING

C207

SYBEX

Microprocessor
INTERFACING
Techniques