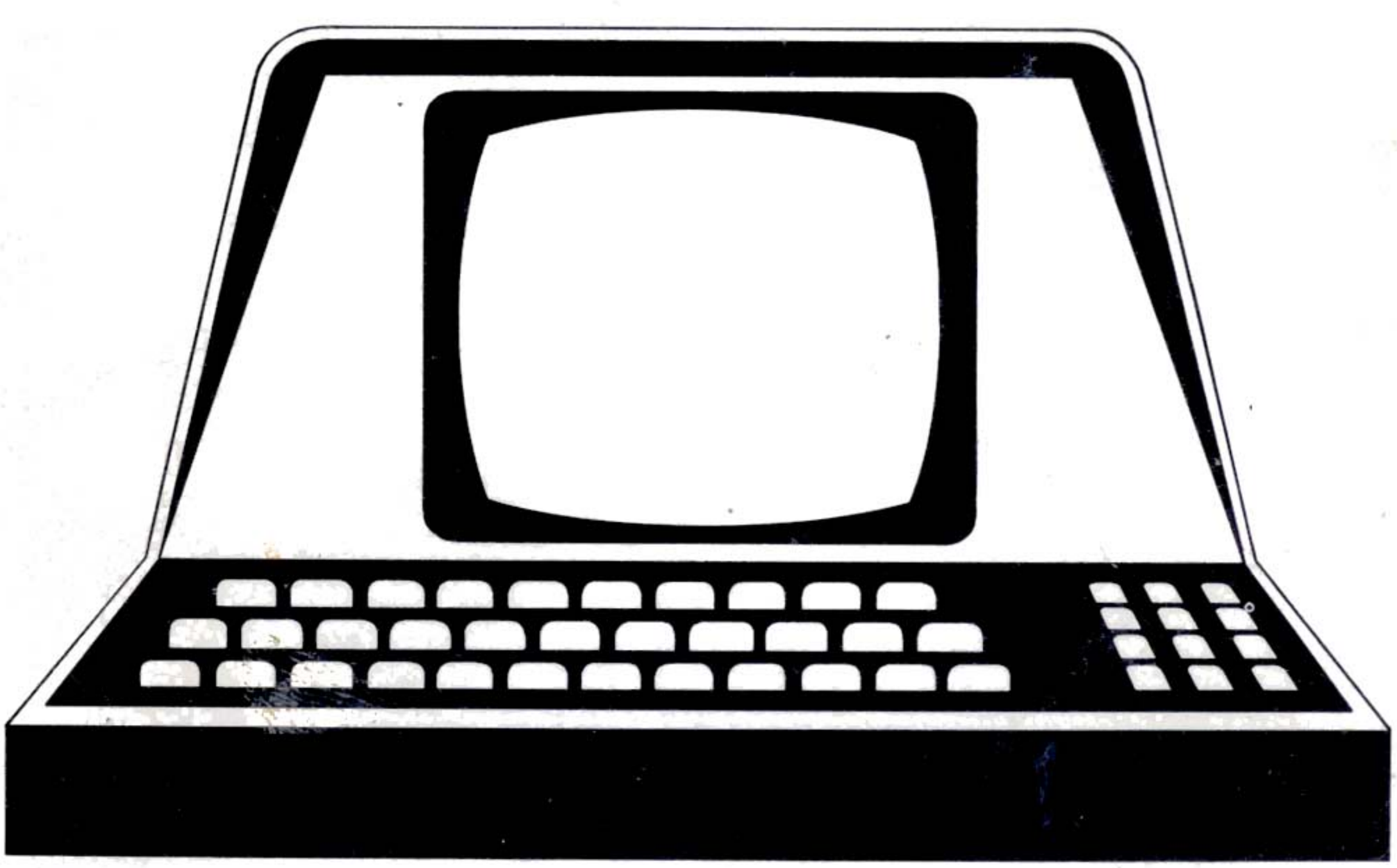
Small Computer Systems Handbook

SOL LIBES



HAYDEN

Small Computer Systems Handbook

The Hayden Microcomputer Series

CONSUMER'S GUIDE TO PERSONAL COMPUTING AND MICROCOMPUTERS*
Stephen J. Freiberger and Paul Chew, Jr.

THE FIRST BOOK OF KIM†
Jim Butterfield, Stan Ockers, and Eric Rehnke

GAME PLAYING WITH BASIC
Donald D. Spencer

SMALL COMPUTER SYSTEMS HANDBOOK†
Sol Libes

HOW TO BUILD A COMPUTER-CONTROLLED ROBOT†
Tod Lootbourrow

HOW TO PROFIT FROM YOUR PERSONAL COMPUTER*
Ted Lewis

THE MIND APPLIANCE: HOME COMPUTER APPLICATIONS*
Ted Lewis

THE 6800 MICROPROCESSOR: A SELF-STUDY COURSE WITH APPLICATIONS*
Lance A. Leventhal

**Consulting Editor: Ted Lewis, Oregon State University*

†*Consulting Editor: Sol Libes, Amateur Computer Group of New Jersey and Union Technical Institute*

Small Computer Systems Handbook

Sol Libes

President
Amateur Computer Group
of
New Jersey

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

Library of Congress Cataloging in Publication Data

Libes, Sol.
Small computer systems handbook.

Includes index.
1. Miniature computers. 2. Microcomputers.
I. Title.
QA76.5.L512 001.6'4 78-6983
ISBN 0-8104-5678-8

Printed in the United States of America

3 4 5 6 7 8 9 PRINTING

79 80 81 82 83 84 85 86 YEAR

Acknowledgments

This book could not have been written without the assistance of many people. It is not possible to mention them all here, but I would like to acknowledge those who helped most of all.

For being gracious enough to review the manuscript, and make critical recommendations and corrections, I am grateful to—Don Libes, Rutgers University (also my son), Lennie Libes, County College of Morris (also my wife), and Dr. Allen Katz, Trenton State College (also a very close friend).

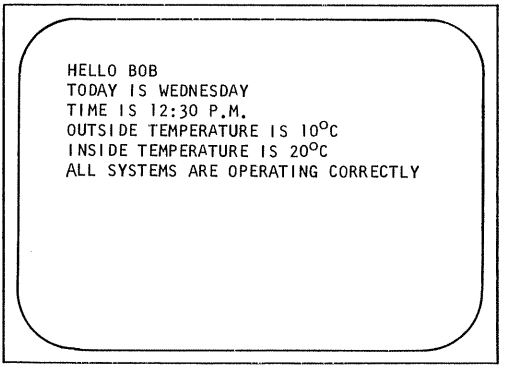
There were also those who furnished information and granted permission to use material. They are—Harry Garland, *Cromenco*; Robert Sumbs, *Intel Corp*; Don McClaughlin, *MOS Technology*; Martin Spergel, *M & R Enterprises*; Dan Meyers, *Southwest Technical Products*; and Dorothy Siegel, *Newtech Computer Systems.*

Contents

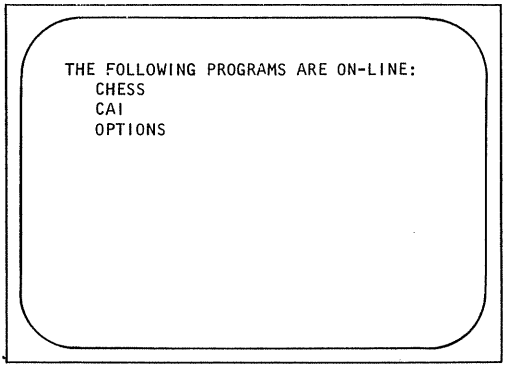
Introduction

THE ADVENTURE AND EXCITEMENT OF PERSONAL COMPUTING

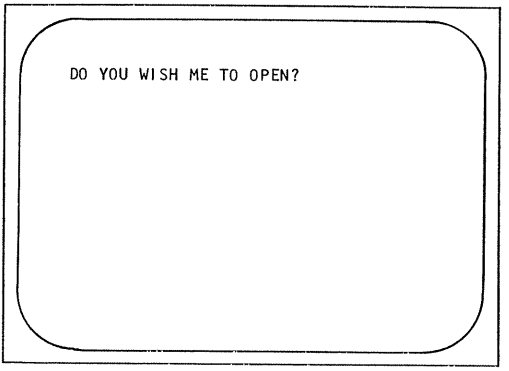
Bob Adams sits down in front of the computer terminal. He presses some keys on the keyboard and a message appears on the screen:



Bob hits another key. A new message appears on the screen.



Bob types "chess" on the keyboard. Immediately a chess board appears on the screen with a full array of chessmen and the message:



Bob types "yes" and the game is under way. Bob is playing chess with the computer.

Bob is doing all this at home. In his own study, with his own home-built computer. It may sound like science fiction from the year 2000, but it isn't. It's today and Bob is not unique. He is one of thousands of computer hobbyists who have built their own home personal computer systems. In fact, Bob found that building his computer was as exciting as using it.

Two years ago Bob was very ignorant about computers. Quite by chance one day, he passed a computer store. Being curious he went in and he found it to be a fascinating place. He spent 2 hours playing with several demonstration systems that were in operation. Bob learned of the kits that were available, and he bought some books on computer basics. He also went to some meetings of a local amateur computer club. He learned from his reading and from other hobbyists that computers, although very powerful machines, are not really that complicated.

As he came to know more about computers he realized that a computer is really nothing more than a machine, but it is the ultimate machine. Machines are designed and built by man to perform operations previously done by man himself. Machines can usually do these tasks better and faster. For example, the automobile performs the job of transporting us from one place to another, which was previously done only by our legs. But, in many cases, the automobile does it better and faster.

The computer is taking over operations previously done only by our brain. We could say that a computer is man's attempt to emulate the human mind. The mind is the central control center of our bodies. The computer is also a control center. Today's computers can perform many tasks performed previously by human minds. As such, it becomes an extension of the human mind, enabling us to do things better and faster.

It is these realizations that so excited Bob. Finally, Bob decided to build his own computer.

IT'S EASY TO BUILD A HOME COMPUTER SYSTEM

Bob consulted some of the computer club members and he got advice on which kits to buy for his intended system. Bob already had experience in assembling a kit—a Heathkit hi-fi receiver.

Much to his amazement he found that the computer kits were simpler than that of the hi-fi receiver. However, the instructions were not always clear and there were a great many unfamiliar terms. He proceeded slowly, consulting his computer club friends and the computer store owner. He found that many of the components were similar to his hi-fi kit, and assembling the electronics required basically the same tools he had used for his hi-fi kit.

He built one unit at a time. First he assembled the *TV terminal* and then the *keyboard.* He connected it to his TV receiver and was delighted when he saw the screen fill up with alphanumeric characters. Following the checkout procedure in the manual, he found that he could type on the keyboard and the characters appeared on the TV screen. He had, in effect, assembled a "TV typewriter."

He was eager now. He proceeded to assemble the *central processor unit,* called a *CPU* for short. Following the manual, he performed some simple electrical tests with an inexpensive voltmeter. It passed the electrical tests.

Now the day came. Bob connected the TV terminal to the CPU. He turned on power and pressed the reset switch. The computer responded by typing "ready" on the screen. Bob jumped for joy. The computer was communicating with him, telling him it was ready to be programmed.

Bob was now over the hardware phase of building his system, and he had done it in only a few weeks of spare-time activity. He felt a sense of accomplishment.

WHAT DO PEOPLE DO WITH HOME COMPUTERS

Now that Bob had his computer system working—or, as the pro's say, "his hardware was operational"—Bob found that a lot more work was required to make his computer really work. This is what the pro's called the *software* phase. Software is the program—the instructions, to tell the computer what to do and how to do it.

The computer brain, as created, is essentially empty . . . like a newborn baby. Just as it is the responsibility of parents to teach an infant what to do and how to do it, the computer must be trained for its tasks. This training is called programming.

Bob attended some classes in programming run by his computer club and a nearby college. He found that the computer understood a limited set of instructions, and that with these, a language could be constructed to make it easier to communicate with the computer.

Bob learned that there was a simple computer language, called *BASIC*, already available for his computer system. He soon learned how to educate his computer quickly by loading the BASIC language into the memory of his computer, using an ordinary audio cassette recorder. The entire process took only a few minutes, and Bob now had an intelligent computer system. He could now communicate with the computer by using ordinary English words like "let" and "print."

Soon Bob found that he could instruct the computer—in the BASIC language—to maintain his financial records, calculate things like interest and depreciation, and even help prepare his income tax return. As Bob came to have a better understanding of hardware and software, he found that his system was extremely versatile and could be expanded to handle the storage of records, control the operation of appliances, and even work as an intelligent typewriter (the pro's call this *word processing*). At computer club meetings he heard about other applications for which home computers are being used. He saw both a computer-controlled robot built by a young high school student and a computer-controlled amateur radio station. He learned of how one amateur uses his computer to assist him in playing the stock market; and he discovered many of the marvelously sophisticated games that could be played with computers.

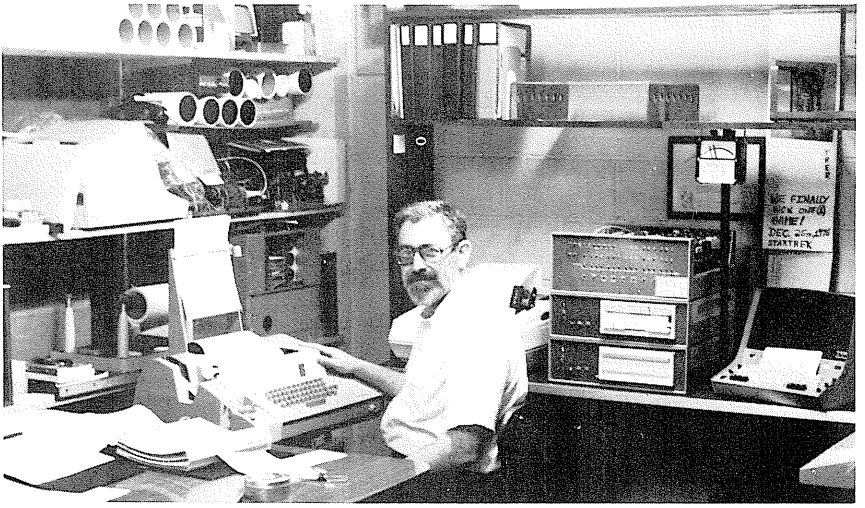
WHAT THIS BOOK IS ABOUT

This book is intended to be a handbook and primer for those new to the field of personal home computers. It provides the necessary background in digital logic fundamentals, number systems, computer hardware, and software basics. Only a minimal knowledge of electronics is required. The theory is presented in a straightforward manner without need to resort to complex theorems. The emphasis is on the important practical knowledge that the home computer user should have to be able to purchase components intelligently, assemble them, and interconnect them.

The material contained in this book has been organized into four basic sections: background information (Chapters 1 through 4), personal computer hardware (Chapters 5 through 9), personal computer software (Chapters 10 through 13), and personal computer applications (Chapter 14).

Further, it provides an introduction to the area of programming on both the lower machine level and the use of higher-level languages such as BASIC. This book will also serve as a reference handbook. Hence the reader can keep it handy to look up facts and definitions, as required, when reading other books.

This book is esentially an overview of the entire subject. The subject is far too extensive to be covered thoroughly in such a short book. Hence, many references are given, at the end of each chapter, for further in-depth reading. Additionally, the reader might also wish to study the author's other texts—one entitled *Fundamentals and Applications of Digital Logic Circuits,* also published by Hayden Book Company, Inc.; the other on microcomputer interfacing, which is now in preparation.



The author at work on a typical home personal computer system.

1.

Computer Codes, Bits, Bytes, and Arithmetic

When we count, we are using a code. We use a code based on the number 10. This code was adopted by man thousands of years ago, probably because man has ten fingers on his two hands. Thus we count 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, etc. We call this "base 10."

A computer uses only switches. Actually, transistors are used as electronic switches. A switch has two positions, open and closed. Hence, it has only two states and we say that it can count by two. Fig. 1-1A illustrates a simple switching circuit consisting of a switch, lamp, and power source. When the switch is open, the lamp is unlit, and when it is closed the lamp is lit.

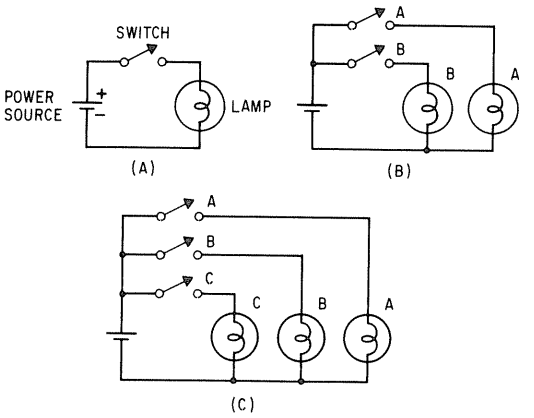


Fig. 1-1. Simple switch-lamp binary counting circuit.

BINARY NUMBERS

When the lamp is unlit we consider this a count of zero (0) and when it is lit, it is a count of one (1). There are only these two counts (or states) and hence it

is called a *binary* system, meaning two states. Therefore, computers that use switching circuits are called *binary computers*.

To count to higher numbers we can add switches and lamps, as shown in Fig. 1-1. Two switches (Fig. 1-1B) allow us to count four states:

Binary B	A	Decimal
0	0	0
0	1	1
1	0	2
1	1	3

Three switches (Fig. 1-1C) permit us to count eight states.

Binary C	B	A	Decimal
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

We are actually counting to the base 2. Hence the A column has a value of 1 (2^0), the B column is 2 (2^1), and the C column is 4 (2^2).

If we added a fourth switch and lamp, it would have a value of 8 (2^3), and we could count 16 states. ($2^3 + 2^2 + 2^1 + 2^0 = 8 + 4 + 2 + 1 = 15$, which when added to the zero state gives us the 16 states.)

Binary D	C	B	A	Decimal
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

We could continue increasing the number of switches to count to larger numbers. Our binary numbers use only the digits 0 and 1. Each digit is called a binary *bit,* and a group of bits is called a binary *word.* We usually group the bits into groups of 8 bits and refer to such a grouping as a *byte.* The A-bit would be called the *LSB* (*least significant bit*) and the D-bit would be called the *MSB* (*most significant bit*).

Converting from Binary to Decimal and Vice Versa

To convert from binary-to-decimal we sum the place values of the 1-bits. For example, convert the binary number 10010111 to decimal, as follows:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	←Base-2 values
128	~~64~~	~~32~~	16	~~8~~	4	2	1	←Decimal place values
1	0	0	1	0	1	1	1	←Binary value

$128 + 16 + 4 + 2 + 1 = 151$

Notice that we have summed the decimal place values for all the 1-bits. The decimal place values for the 0-bits are ignored.

Now to convert from decimal-to-binary. This is done by subtracting the largest base-2 value from the decimal number, then subtracting the next largest subtractable base-2 number from the remainder. A 1 is used for each base-2 value subtracted and a 0 for each value not subtractable. Here is an example—convert decimal 182 to binary as follows:

182	decimal number
−128	$2^7(128) \longrightarrow 1$
54	$2^6(64)$ not used→0
−32	$2^5(32) \longrightarrow 1$
22	
−16	$2^4(16) \longrightarrow 1$
6	$2^3(8)$ not used→0
−4	$2^2(4) \longrightarrow 1$
2	
−2	$2^1(2) \longrightarrow 1$
0	$2^0(1)$ not used→0

$182_{10} = 10110110_2$ ⟶↑

OCTAL AND HEX NUMBERS

The computer works with binary words. However, humans have difficulty working with larger binary numbers and have adopted other codes to simplify computer work. The two most used codes are the *octal* and *hexidecimal* (called

hex for short) codes. The octal code is based on grouping the bits into groups of threes. For example:

011	000	101	110	binary
↓	↓	↓	↓	
3	0	5	6	octal

It is much easier to remember octal 3056 than its binary equivalent of 011000101110. Thus, to convert from binary to octal, group the bits into groups of three each and convert each 3-bit group to its octal number. It is called octal since only eight numbers, 0–7, are used.

Here is another example: Convert 1234 (octal) to binary:

1	2	3	4_8
↓	↓	↓	↓
001	010	011	100_2

Notice the subscripts 8 and 2 are used to indicate the octal and binary numbers.

Hex code is also very popular because it is even easier to remember than octal. To convert from binary to hex, group the bits into groups of four. Note that to represent 16 states it is necessary to use letters as well as numbers as follows:

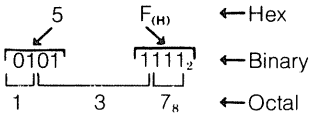
Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

To convert from binary to hex, group and convert as follows:

0110	0010	1110_2
↓	↓	↓
6	2	E_H

This is the same number we converted originally into octal 3056. Note now that only three hex digits are needed compared to the four octal digits. Fewer digits make the number easier to remember and cut down the chance for errors to be made. Observe that a subscript H is used to denote the hex number.

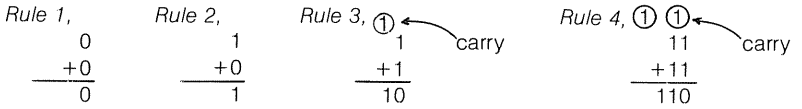
If we wish to convert from octal to hex or vice versa, an easy way to do it is to convert to binary first. For example, to convert from $5F_H$ to octal:



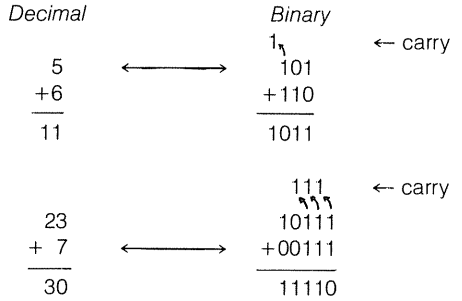
Several other computer codes are used, but octal and hex are the most popular. When communicating between the computer and a terminal, such as a Teletype*, an expanded code which can be used to denote all the printable characters and control codes is required. For this purpose the computer industry has standardized on the *ASCII code* (*American Society for Communications Interface and Interchange*). The ASCII (pronounced as-key) code uses an 8-bit byte and is included in Appendix A.

BINARY ARITHMETIC

We can add binary numbers in a manner similar to adding decimal numbers. Here are the four basic rules:



In Rules 3 and 4 note that a *carry* is generated from the previous column to the next column. Here are two examples:



Binary Subtraction

Most microprocessors use a technique called *2's complement subtraction*. In this way the basic adder circuit can also be used to perform subtraction. The technique requires that the number being subtracted be complemented and a 1 added to it. To complement a binary number means to change each 1 to 0 and

*Registered trademark of the Teletype Corp.

each 0 to a 1. Then adding a 1 to the complemented binary word makes it a 2's complement. For example:

Binary number	*Complement*		*2's Complement*
101 ⟶	010	+1 =	011
0001 ⟶	1110	+1 =	1111

Subtraction is performed by adding the 2's complement of the number being subtracted to the number from which it is being subtracted and disregarding the last carry. Example:

Decimal	*Binary*	*2's Complement*	
10	1010	⟶ 1010	(no change)
− 6	−0110 →	1001+1→+1010	(2's complement)
4		①0100	
		↖ disregard last carry	

Another example:

14	1110	⟶	1110
− 9	−1001 →	0110 + 1 →	+0111
5			①0101
			↖ disregard last carry

When the last carry = 1, the answer is positive. When the carry = 0 (no carry), the answer is negative and is the 2's complement of the result.

Binary Multiplication

Binary multiplication is similar to decimal multiplication. The technique is called *summing partial products*. The following rules apply:

Rule 1,	*Rule 2,*	*Rule 3,*	*Rule 4,*
0	0	1	1
×0	×1	×0	×1
0	0	0	1

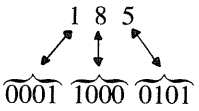
To multiply we find all the partial products, shifting each to the left one place and summing the partial products. In this way the operation can be done by the adder circuit in the computer. For example:

9	1001	
×5	× 101	
45	1001	
	0000	↔partial products
	1001	
	101101	←final product

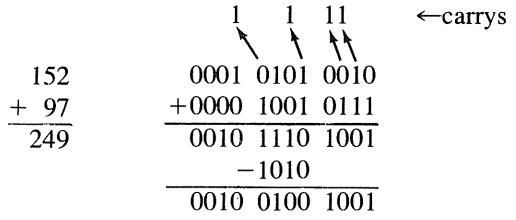
BCD NUMBERS

Most calculators and some computers use a *BCD* code (*b*inary-*c*oded-*d*ecimal). Although this code uses more bits it is easier to handle. The code

consists of using 4-bit binary groups to represent the decimal digits 0 through 9. Hence, there will be a 4-bit binary group for each decimal digit. For example, the decimal number 185 would be represented as follows:



BCD arithmetic is performed somewhat differently than binary arithmetic. For example, each 4-bit group is added in standard binary fashion. However, if the sum is greater than $1001_2(9_{10})$, then a carry is generated to the next group, and $1010_2(10_{10})$ is subtracted from the group. Here is an example.



Most microprocessors have instructions which facilitate BCD arithmetic operations.

Recommended Further Reading

1. Sol Libes, *Fundamentals and Applications of Digital Logic Circuits,* Revised Second Edition, Hayden Book Co., Inc., Rochelle Park, N.J., 1978.

2.

Digital Logic

The circuitry of a computer consists of millions of electronic switches. These switches are arranged to perform operations and make decisions and hence are called *logic* circuits. Present technology can house thousands of switching circuits in one case called an integrated circuit (Fig. 2-1).

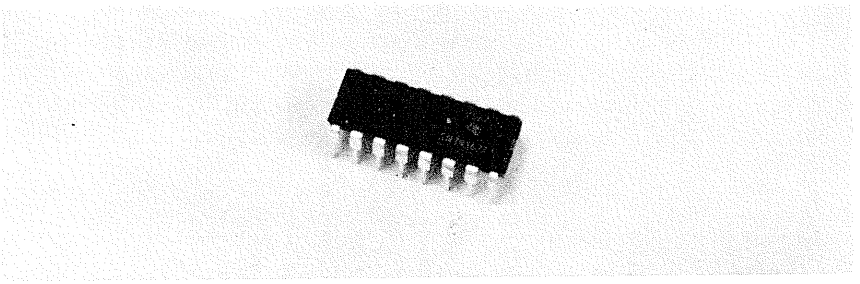


Fig. 2-1. Typical integrated circuit.

The switches are actually transistors which are caused to turn on or off. For short, we call an *integrated circuit* an *IC*.

LOGIC STATES

In our simple logic circuit (Fig. 2-2) we indicate that the switch is *ON* by a 1 and *OFF* by a 0. Often a +5 V (actually between +2.4 and +5 V) is usually taken to be a 1-logic level and 0 V (actually 0 to 0.4 V) a 0-logic level. We may also say that a 1 is a *high* or *hi* logic level and a 0 is a *low* or *lo* logic level.

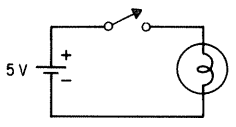


Fig. 2-2. Simple logic circuit.

DIGITAL LOGIC GATES

Logic circuits are electronic circuits that switch between logic-0 and 1 states. They are like switches that open and close and are, therefore, referred to as *gates*. A gate has two or more inputs and one output. This output occurs when certain conditions are met. The gate's operation is analyzed with a *truth table* which shows all input and output possibilities of the gate.

The very earliest gate circuits used relays. Later, vacuum tubes, diodes, and transistors were used. Today, gates are made from integrated circuits (ICs) almost exclusively. The IC is a plastic or ceramic package which contains transistors, diodes, and resistors assembled to form gates. Therefore, it will be better to concern ourselves with the gate as a functional block rather than the inner workings of the circuit.

There are three basic logic circuits. They are the *AND*, *OR*, and *NOT* (inverter). All other logic circuits are built from these basic logic circuits.

The AND Gate

The AND gate (Fig. 2-3A) has two inputs. When one or both inputs = 0, the output is 0. Only when both inputs = 1 will the output = 1. It operates in the same way as two switches connected in series (Fig. 2-3B). Only when switches A and B are closed (1 position) will the lamp light (1).

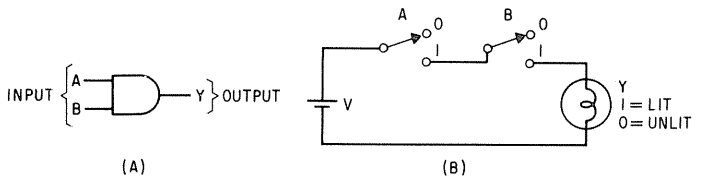


Fig. 2-3. A 2-input AND gate (A) and equivalent switch circuit (B).

Only when A *and* B are 1 will Y (output) be 1, hence the name *AND* gate. If either or both switches are open, or 0, no current will flow through the switches and there will be a 0 output (Y).

We can construct a truth table (Table 2-1) which describes the output Y of the AND gate. It shows that there are four possible combinations of input states

Table 2-1. 2-Input AND Gate

Inputs B A	Output Y
0 0	0
0 1	0
1 0	0
1 1	1

giving 1 and 0 at the output. In three of the combinations Y = 0. Only if A and B are 1 will Y = 1.

The number of gate inputs can be increased, just as the number of switches can be increased. Figure 2-4 is a 3-input AND gate with its switch equivalent circuit. The truth table is shown in Table 2-2. Notice that the 3-input gate has eight possible states, but again only when all inputs = 1 will the output = 1. Figure 2-5 shows the pin-out diagrams for three popular IC AND gates.

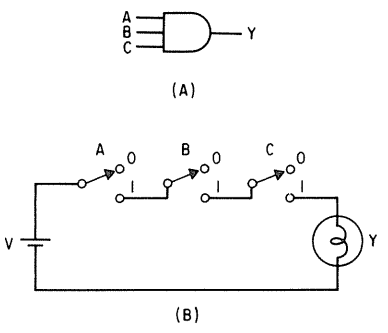


Fig. 2-4. A 3-input AND gate (A) and equivalent switch circuit.

Table 2-2. 3-Input AND Gate

Inputs C	B	A	Output Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

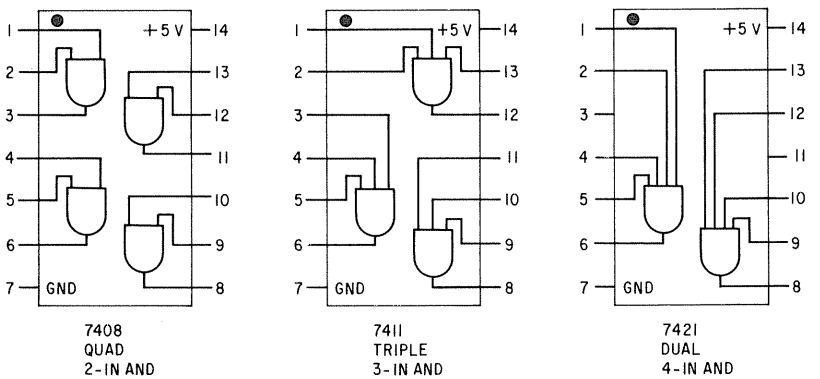


Fig. 2-5. Popular AND gate IC packages.

The diagram shows the pin connections within the IC. Note that a dot on the IC case usually indicates pin #1. On a 14-pin IC pins 7 and 14 are often used for the ground and +5 V connections, respectively.

Frequently when constructing circuits it is necessary to convert a gate with many inputs to one with fewer inputs. Two methods for doing this are shown in Fig. 2-6; in both, the 3-input AND gate will function as a 2-input AND gate.

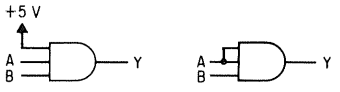


Fig. 2-6. Two methods of converting a 3-input AND gate to a 2-input AND gate.

The OR Gate

The schematic symbol for an OR gate is illustrated in Fig. 2-7A. The 2-input OR gate will have a 1 output when A or B = 1, hence the name *OR* gate. The equivalent switch circuit is shown in Fig. 2-7B. Note that the two switches are in parallel so that if switch A or B is closed, current flows and lights the lamp. The truth table is shown in Fig. 2-7C. The logic symbol, equivalent

Fig. 2-7. The 2-input OR gate.

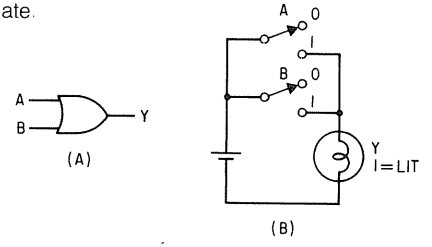


Fig. 2-8. The 3-input OR gate.

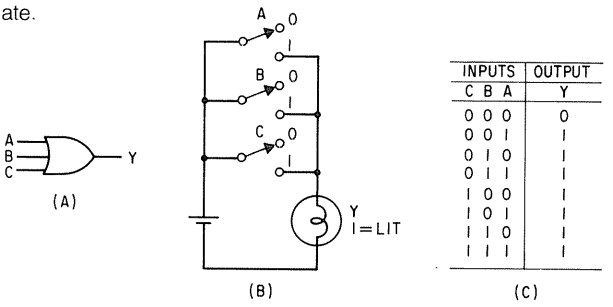
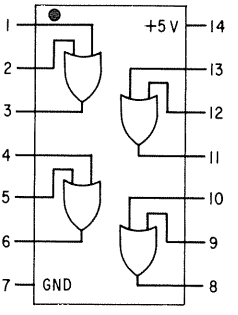


Fig. 2-9. Pin-out diagram for the 7432 quad 2-input OR gate IC.



switch circuit, and truth table for a 3-input OR gate are shown in Fig. 2-8. Figure 2-9 shows the pin-out diagram for a popular IC OR gate.

The NOT (Inverter) Gate

The NOT circuit, usually called an *inverter* (Fig. 2-10), inverts a logic level. This is indicated by a circle, usually called a *bubble,* at the gate's input or output. A bar is placed above the input or output letter ($\bar{A}$) to represent the opposite (inverted) logic state. It is said that the circle and bar indicate the action of *complementing* and *negating*. Hence, if the inverter input = 0, the output = 1; and if the input = 1, the output = 0 (Fig. 2-10A and B).

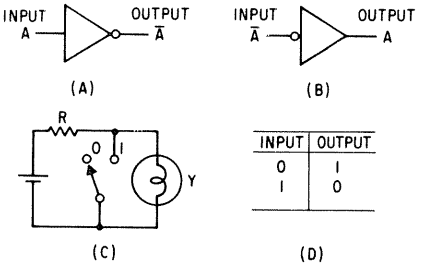


Fig. 2-10. The inverter or NOT gate.

In other words, the output is always the opposite state or complement of the input and hence, NOT the input. The equivalent switch circuit and the truth table are shown in Fig. 2-10C and D, respectively. Note that a resistor is used in the switch circuit in order to prevent the power source from being short-circuited when the switch is in the 1 position. Figure 2-11 illustrates the pin-out diagram for a popular IC inverter.

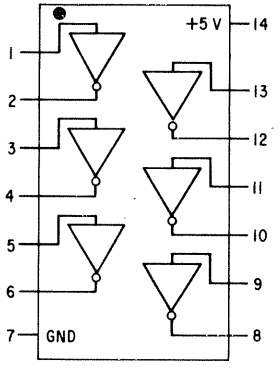
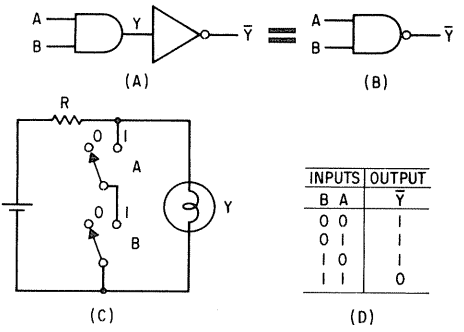


Fig. 2-11. Pin-out diagram for the 7404 hex-inverter IC.

The NAND and NOR Gates

The *NAND* and *NOR* gates are the most widely used logic gates, because any type of gate can be made from them. There have been entire systems built exclusively with NAND or NOR gates. Hence, they are considered to be *universal* gates.

The NAND is basically an AND gate followed by an inverter (Fig. 2-12A). Hence, the logic symbol is an AND gate symbol with a bubble on the output (Fig. 2-12B). The equivalent switch circuit and truth table for the NAND gate are shown in Fig. 2-12C and D. Only when A *and* B = 1 will the output = 0.

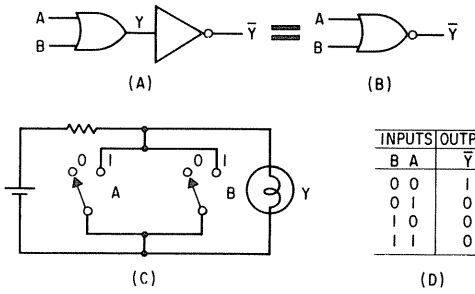


INPUTS		OUTPUT
B	A	$\bar{Y}$
0	0	1
0	1	1
1	0	1
1	1	0

(D)

Fig. 2-12. The NAND gate.

The NOR gate is essentially an OR gate followed by an inverter (Fig. 2-13A). The logic symbol is, therefore, an OR gate symbol with a bubble at the output (Fig. 2-13B). The equivalent switch circuit and truth table are shown in Fig. 2-13C and D. When A *or* B = 1 the output Y = 0.



INPUTS		OUTPUT
B	A	$\bar{Y}$
0	0	1
0	1	0
1	0	0
1	1	0

(D)

Fig. 2-13. The NOR gate.

It is possible using only NAND or only NOR gates to perform any type of gate operation. Figure 2-14 shows how NAND and NOR gates can be used to build all other types of logic gates. Figure 2-15 shows the pin-out diagrams of several popular NAND and NOR gate ICs.

The X-OR and X-NOR Gates

The *exclusive-OR*, popularly called *X-OR*, and the *exclusive-NOR*, or *X-NOR*, gates are widely used gates actually made from the previous gates. They compare inputs and are often called *comparators*.

The X-OR gate symbol is shown in Fig. 2-16A with its equivalent switch circuit (B) and truth table (C). The circuit compares switches A and B (inputs). When the input states are opposite, current flows through the lamp and it will light (=1); when the inputs are the same the lamp will not light. The X-OR gate can thus be considered an *inequality detector*.

The X-NOR gate symbol is shown in Fig. 2-17A with its equivalent switch circuit (B) and truth table (C). The circuit compares input A and B and when they

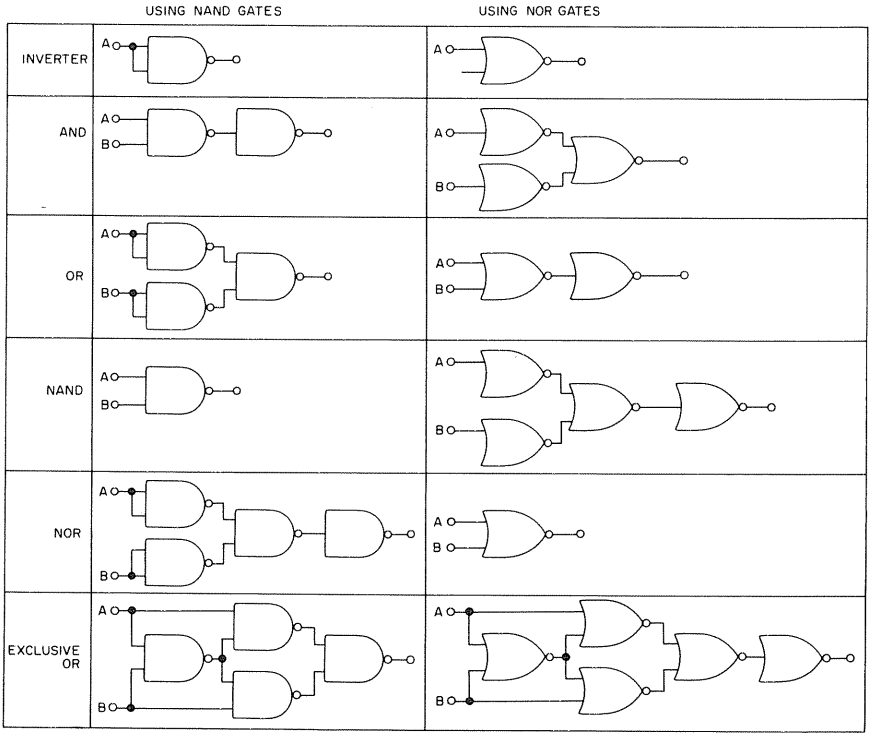


Fig. 2-14. Using NAND and NOR gates to perform basic logic functions.

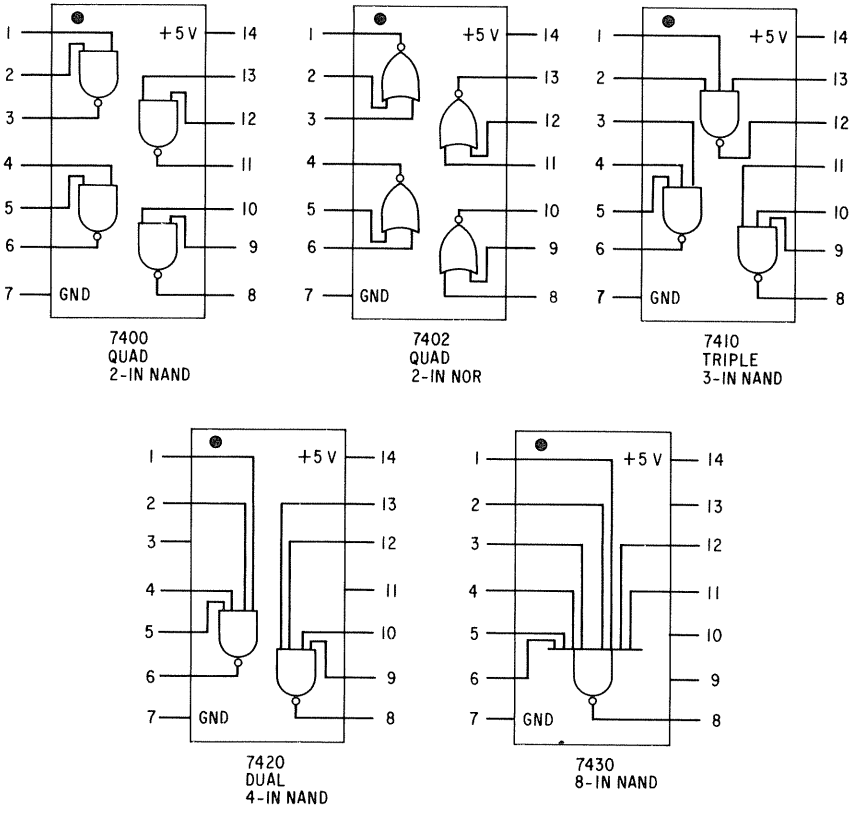
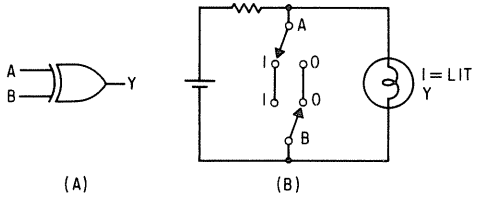


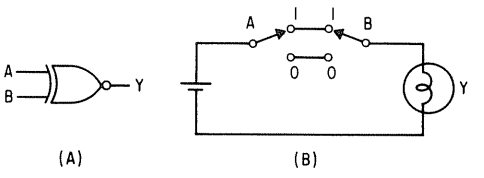
Fig. 2-15. Pin-out diagrams for popular NAND and NOR ICs.



INPUTS B	INPUTS A	OUTPUT Y
0	0	0
0	1	1
1	0	1
1	1	0

(C)

Fig. 2-16. The X-OR gate.



INPUTS B	INPUTS A	OUTPUT Y
0	0	1
0	1	0
1	0	0
1	1	1

(C)

Fig. 2-17. The X-NOR gate.

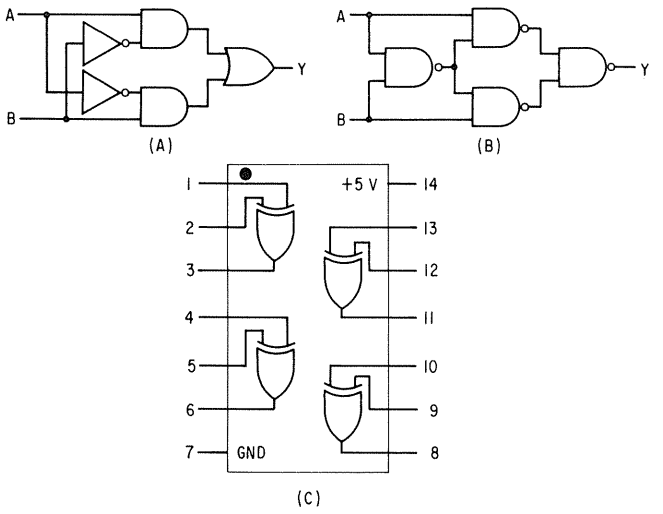


Fig. 2-18. Building X-OR gates using basic gates and the 7486 quad X-OR gate IC.

are identical Y = 1. The X-NOR gate can thus be considered an *equality detector*.

The X-OR gate can be built using basic gates (Fig. 2-18A and B), and an IC X-OR gate is shown in Fig. 2-18C.

THE PARITY-BIT AND ERROR-CHECKING

The X-OR and X-NOR gates find wide application in computer data transmission systems. Binary words are often checked as they are sent from one point to another to see that no error has occurred due to noise or equipment failure. This is done using X-NOR gates to generate a *parity-bit*.

The parity-bit is added to the data word and is used to check for errors. *Even-* and *odd-parity* systems are used. In the even-parity method, a 1-bit is added to the word if the number of 1-bits in the word is odd. In the odd-parity method, the 1-bit is added if an even number of 1-bits exist in the data word.

Figure 2-19 shows a system utilizing a 5-bit receiver input, where 4 bits are used for data and 1 bit is the parity-bit. The parity-bit generator circuit compares the 4-bit data word and generates the parity bit, so that the transmitted word consists of 5 bits. The parity-bit generator output = 0 for an odd number of 1s in the data word and the output = 1 for an even number of 1s in the data word. At the receiving point a parity-bit detector is used to compare the received word to the parity-bit. An error output is generated if they are not consistent.

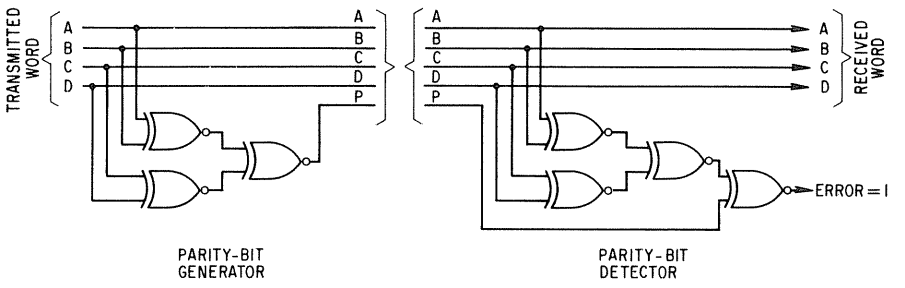


Fig. 2-19. Parity-bit method used in data transmission system.

ARITHMETIC CIRCUITS

Binary addition (discussed earlier) is accomplished using *half-adder* and *full-adder* circuits. The half-adder circuit (Fig. 2-20A) sums two binary bits and produces the sum and carry. The functional block symbol for the *half-adder (HA)* is shown in Fig. 2-20B.

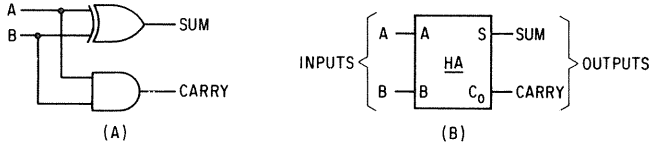


Fig. 2-20. The half-adder circuit.

The *full-adder (FA)* is shown in Fig. 2-21A. The FA sums two binary bits and the carry from a previous HA or FA. It produces a sum and carry. The functional symbol for the FA is shown in Fig. 2-21B.

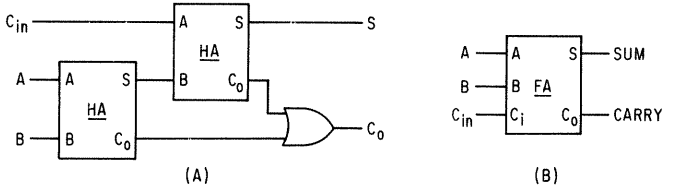


Fig. 2-21. The full-adder circuit.

A circuit to add binary words (Fig. 2-22) is called a *parallel-adder* circuit since it adds all bits at one time. The circuit will sum two 4-bit words. It consists of a HA and 3 FAs. The LSB (least significant bit) requires only a HA since there is no carry-in.

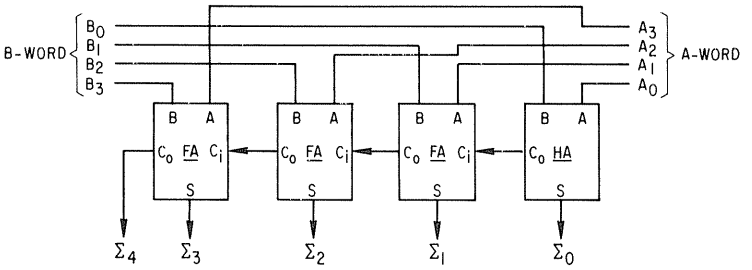
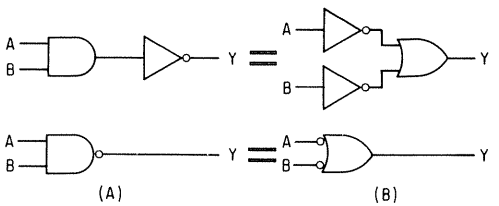


Fig. 2-22. A circuit to add two 4-bit words.

DEMORGAN'S THEOREMS

A very important facet of working with logic gates is referred to as *De-Morgan's Theorems*. These theorems are as follows:

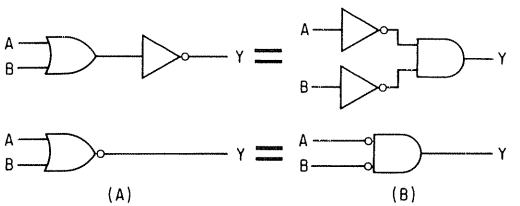
1. *A NAND gate performs the same logic function as an OR gate with negated inputs*. This is shown in Fig. 2-23A and B. Fig. 2-23C shows the truth table which applies to both circuits.
2. *A NOR gate performs the same logic function as an AND gate with negated inputs*. This is shown in Fig. 2-24A, B, and C.



INPUTS B	INPUTS A	OUTPUT Y
0	0	1
0	1	1
1	0	1
1	1	0

(C)

Fig. 2-23. The equivalency of NAND and negated input-OR gates.



INPUTS B	INPUTS A	OUTPUT Y
0	0	1
0	1	0
1	0	0
1	1	0

(C)

Fig. 2-24. The equivalency of NOR and negated input-AND gates.

The equivalency of these gates enable the use of NAND gates to be used as OR gates and the use of NOR gates to perform AND operations.

It is, therefore, very important to note where the bubbles appear on logic gate symbols. For example, a bubble at an input and no bubble at the output indicates that 0-logic levels at the input will produce a 1-logic at the output. The showing of a NAND gate on a schematic as a negated-input-OR gate often better describes to the reader the logic operation being performed.

BUILDING INVERTERS FROM OTHER GATES

Frequently inverters are made using NAND, NOR, and X-OR gates. This usually occurs because of unused gates in IC packages and the desire to use as few ICs as possible in a system. Figure 2-25 shows how an inverter is built using these gates. Note that the X-OR and X-NOR gates may also be used as controlled inverters if the input, which is shown = 0 (X-OR) or 1 (X-NOR), is used as an enable input. Note also that the arrangements shown in Fig. 2-25A and C are preferred to B and D since this method presents less of a load to the driving circuit.

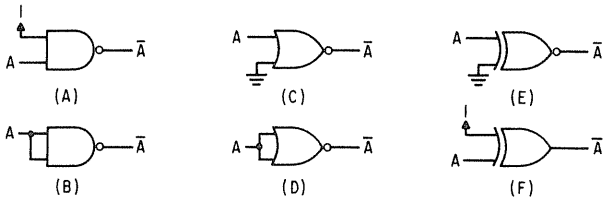


Fig. 2-25. Using NAND, NOR, X-NOR, and X-OR gates as inverters.

ENCODER AND DECODER CIRCUITS

Logic gates are employed to perform logic functions. In this introductory book it is not feasible to discuss all possible applications. One of the most common applications of logic gates is that of code conversion—converting from decimal-to-binary and from binary-to-decimal. The former is called *encoding* and the latter *decoding*.

A typical decimal-to-binary encoder circuit is shown in Fig. 2-26. It is typical of the circuit used to convert decimal keyboard output to binary. In other words, when a particular decimal switch is closed on the keyboard, the encoder circuit will develop the correct binary code for the computer.

Note that closing a keyboard switch places a 0 at the particular gate inputs to produce the desired 1 outputs. Hence, the bubbles appear at the inputs. In actuality the circuit is built using NAND gates but the logic functions performed are those of negated-input OR gates.

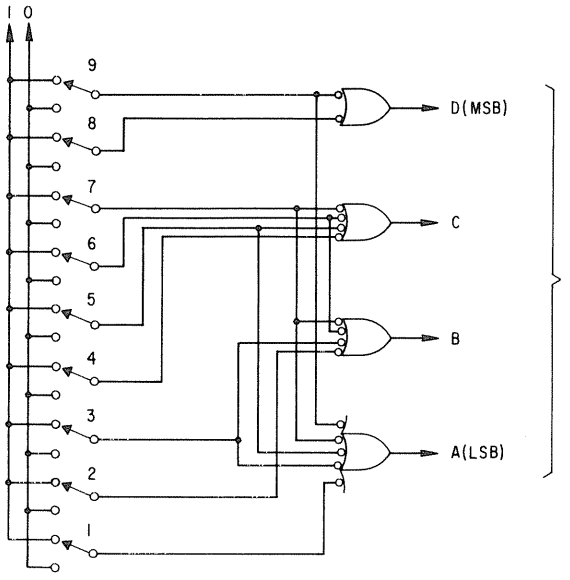
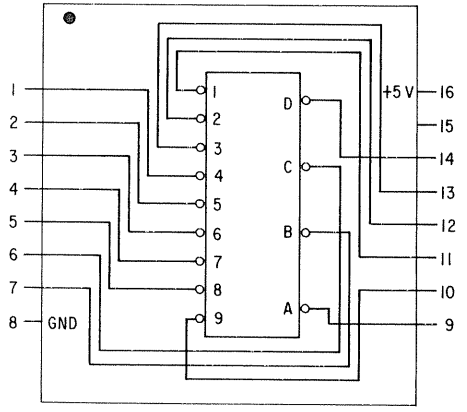


Fig. 2-26. A decimal-to-binary encoder circuit (10-line to 4-line).

A popular decimal-to-binary encoder IC, also called a 10-line to 4-line encoder, is shown in Fig. 2-27A and its truth table in 2-27B. Note that the desired inputs and outputs are 0-logic levels and, hence, bubbles are shown at both inputs and outputs. In the truth table the X indicates a "don't-care" state; in other words, it does not matter whether the input is 1 or 0 to achieve the desired output.



(A)

INPUTS									OUTPUTS			
1	2	3	4	5	6	7	8	9	D	B	C	A
1	1	1	1	1	1	1	1	1	1	1	1	1
X	X	X	X	X	X	X	X	0	0	1	1	0
X	X	X	X	X	X	X	0	1	0	1	1	1
X	X	X	X	X	X	0	1	1	1	0	0	0
X	X	X	X	X	0	1	1	1	1	0	0	1
X	X	X	X	0	1	1	1	1	1	0	1	0
X	X	X	0	1	1	1	1	1	1	0	1	1
X	X	0	1	1	1	1	1	1	1	1	0	0
X	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

(B)

Fig. 2-27. The 74147 10-line to 4-line encoder IC.

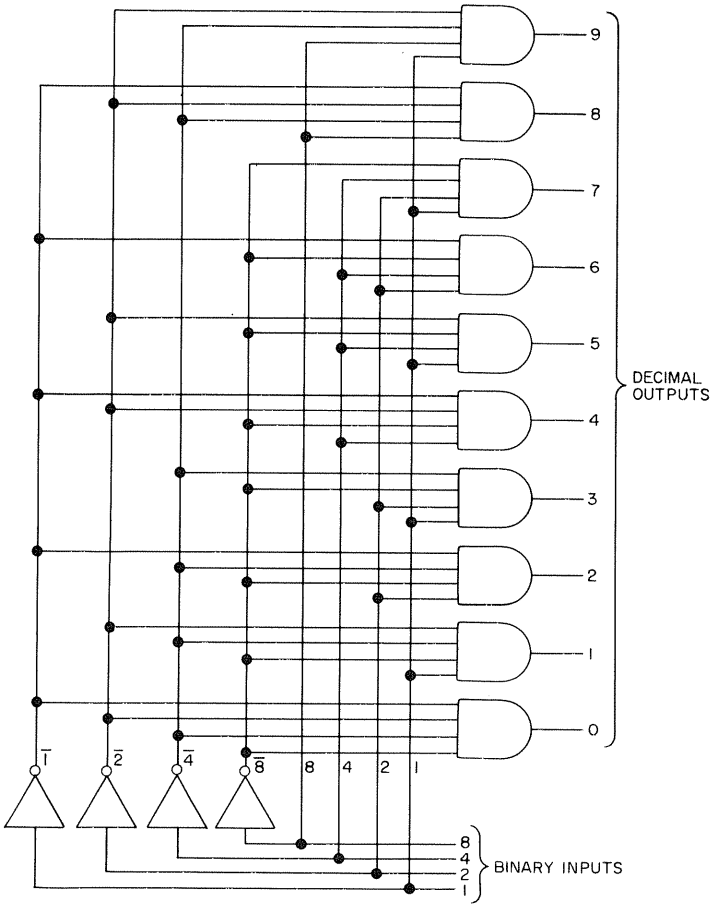


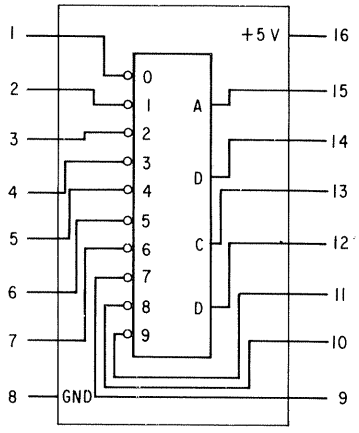
Fig. 2-28. A binary-to-decimal decoder circuit.

The decoder is the opposite of an encoder. For example, the circuit shown in Fig. 2-28 is a binary-to-decimal decoder circuit performing just the opposite code conversion of the previous circuit. Notice that the 4-bit binary word can produce up to 16 distinct outputs.

A popular binary-to-decimal decoder IC is shown in Fig. 2-29.

TTL, CMOS, AND MOS

So far, we have avoided any discussion of the internal circuitry of ICs. However, a few words are needed here to distinguish between the different types of ICs. Currently there are three basic technologies in use—TTL, CMOS, and MOS.



(A)

INPUTS				OUTPUTS									
D	B	C	A	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1

(B)

Fig. 2-29. The 7442 binary-to-decimal decoder IC.

TTL, as it is more commonly called, stands for *transistor-transistor logic.* For a detailed description of the circuit operation the reader is referred to the author's book *Fundamentals and Applications of Digital Logic Circuits.* TTL is very popular because of its low cost and good performance. All the ICs we have referred to previously are TTL ICs. TTL ICs always start with the designation 74 followed by two or three numbers designating the function. For example, the 7400 is a quad 2-input NAND IC. The 74xx family of ICs is furnished in plastic DIP cases and rated for operation from 0 to +70°C. The 54xx family is the same as the 74xx with the exception that it is furnished in ceramic and rated for −55 to +125°C operation.

Each TTL gate input presents a *unit load* and the gate output can drive up to 10 unit loads. TTL gates are available with high current outputs. They are called *buffers* and can drive 20 to 30 unit loads.

All TTL gates operate from a +5 V power supply, dissipate 10 MW per gate, and have a typical delay of 10 ns (nanoseconds). This means a maximum operating frequency of 35 MHz. High-speed TTL logic designated, with the letter H (i.e., 74H00), has only a 6-ns delay (maximum frequency = 50 MHz) but has an increased power dissipation of 22 MW/gate.

Low-power TTL, designated with the letter L (i.e., 74L00), has a power dissipation of only 1 MW/gate but is much slower (33-ns delay, 10-MHz maximum frequency). Higher frequencies are obtained using *Schottky* diodes with each transistor. These ICs are designated with the letter S (i.e., 74S00). They have a typical delay of only 3 ns (maximum frequency = 125 MHz).

It is also possible to obtain a low-power Schottky type TTL IC. This would be designated with the letters LS (i.e., 74LS00).

CMOS stands for *complementary metal-oxide-silicon* transistor. This is a newer technology than TTL and offers several advantages. It has very low power dissipation (2–3 MW/gate), very high noise immunity, draws very low current from the driving gate, and can drive up to 50 CMOS loads. However, it has a high delay time (25 ns).

MOS stands for *metal-oxide-silicon* transistor. ICs using this technology permit a very high density of electronic circuitry in a small piece of silicon. Most microprocessors and memory ICs are built using this technique. MOS ICs are limited in speed and power handling ability.

TRI-STATE AND OPEN-COLLECTOR ICs

Logic gates which are connected to a common line (we will discuss this in more detail later) usually employ a *tri-state* output circuit. This means that in addition to the 0- and 1-logic levels appearing at the output, a third state of infinite resistance (like an open switch) can exist. This is particularly useful when many gate outputs are connected to a common line called a *bus*.

The third state is controlled by a special input to the IC which in effect turns the buffer off. For example Fig. 2-30 shows a buffer IC with tri-state outputs. Each buffer has a control input which when 0 allows the buffer to function as a noninverting logic gate. When the control input = 1 the buffer turns off and the gate is effectively disconnected from the output.

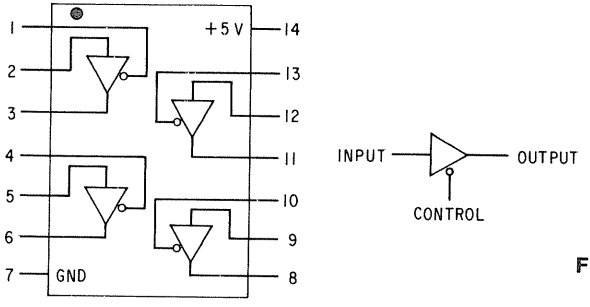


Fig. 2-30. The 74125 buffer with tri-state output.

Prior to the introduction of tri-state gates, bussing was accomplished using *open-collector* type gates. These gates are slower, less expensive, and more affected by noise than tri-state gates. However, they are in common use. In an open-collector type gate, a resistor which is normally internal to the IC-gate is omitted and instead is connected external to the gate. Using the external resistor permits wiring several open-collector type gate outputs together to share this resistor (Fig. 2-31).

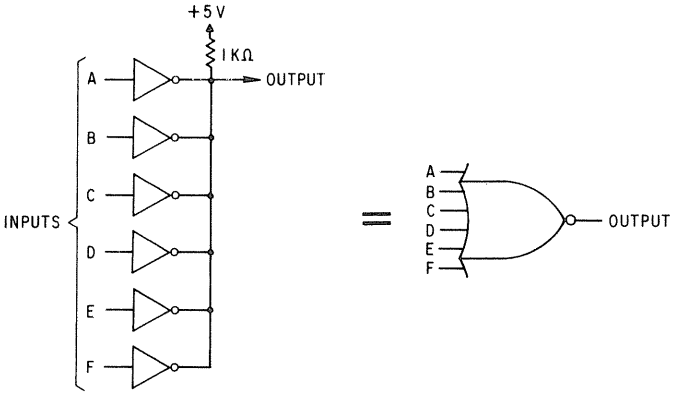


Fig. 2-31. The 7405 hex inverter with open collectors functioning as a NOR gate.

Notice that the output = 1 only when all inputs = 0 and that if any input = 1 the output = 0. Hence, the inverters have been wired to perform the function of a 6-input NOR gate.

Unfortunately, most schematic diagrams do not distinguish between an open-collector and standard gate. However, sometimes the letters o.c. or a slash are used (Fig. 2-32).

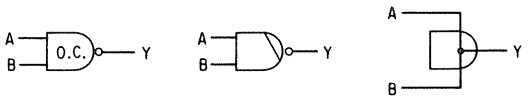


Fig. 2-32. Different ways of indicating an open-collector type gate on schematics.

Recommended Further Reading

1. Sol Libes, *Fundamentals and Applications of Digital Logic Circuits,* Revised Second Edition, Hayden Book Co., Inc., Rochelle Park, N.J., 1978.
2. Donald E. Lancaster, *TTL Cookbook,* Howard W. Sams & Co., Inc., Indianapolis, Ind., 1974.
3. Donald E. Lancaster, *CMOS Cookbook,* Howard W. Sams & Co., Inc., Indianapolis, Ind., 1977.
4. *Digital, Linear, MOS Data Book,* Signetics Corp., Sunnyvale, Calif., 1976.

3.

More about Digital Logic

Flip-flops are circuits made up of gates which have the ability to store data and to count. As such they are the basic building blocks of memory and control/timing systems.

THE BASIC FLIP-FLOP (R-S)

A simple flip-flop can be constructed from two NAND or NOR gates (Fig. 3-1). In Fig. 3-1A the resistors are called *pull-up resistors* since, when the gate input is not grounded (0 logic level) through the switch, the gate input is "pulled-up" to a logic-1 level through the resistor. On the other hand, in Fig. 3-1B, when NOR gates are used, the resistors pull the input down to ground (0-logic level) when the switch is open. Hence, the resistors are called *pull-down resistors*. Notice a pull-up resistor is connected from a gate input to +5 V (1) and a pull-down resistor is connected from a gate input to ground (0).

Since the gate inputs and outputs are cross-connected, one gate will always have a 1 output while the other will always have a 0 output. The outputs are labeled Q and $\overline{Q}$. The line above the Q indicates that it is the opposite logic state of Q. Hence if Q = 0 then $\overline{Q}$ = 1 and vice versa.

If Q = 0 and $\overline{Q}$ = 1 we call this the *reset* state. If Q = 1 and $\overline{Q}$ = 0 this is

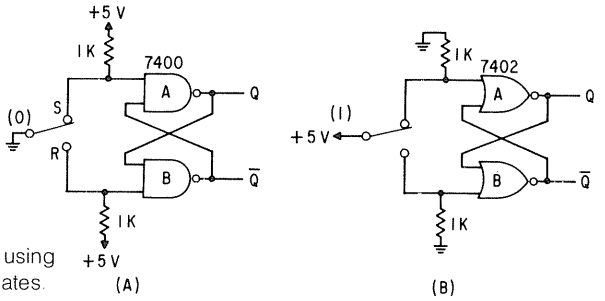


Fig. 3-1. The basic flip-flop using NAND and NOR gates.

called the *set* state. In Fig. 3-1A, if the flip-flop is reset, then moving the switch up will cause the flip-flop to set. This occurs because gate A has 0 inputs and output = 1 while gate B has 0 and 1 inputs and output = 0. Notice that the switch is labeled S on the side which will cause the flip-flop to set.

Moving the switch down to the R, or reset position, causes gate A to have 1 and 0 inputs and output = 0 and gate B to have 0 inputs and output = 1. Hence, the flip-flop resets.

Notice, that when the switch is between the S and R contacts no change occurs. The flip-flop remains in the same state it was in previously. The flip-flop "remembers" its last logic state. This is the way a memory stores logic state data.

There are only two logic states. If the switch should bounce as the contacts close or open, no change, other than the first change, will occur. Hence, no noise is generated. For this reason, this circuit is often employed to *de-bounce* switch contacts.

The circuit is commonly referred to as a *reset-set flip-flop* (*R-S flip-flop*). Although the circuit is usually constructed with 2-input NAND gates, it is often shown schematically as negated-input OR gates (Fig. 3-2A). This is because, although NAND gates are used, they are functioning as negated-input OR gates. Occasionally, the flip-flop will be shown as a functional logic block (Fig. 3-2C). In this case the bubbles at the R and S inputs indicate that a 0-logic level causes setting and resetting of the flip-flop. Notice that the R and S inputs are never both = 0. This is prohibited since it would cause the Q and $\overline{Q}$ outputs to = 1.

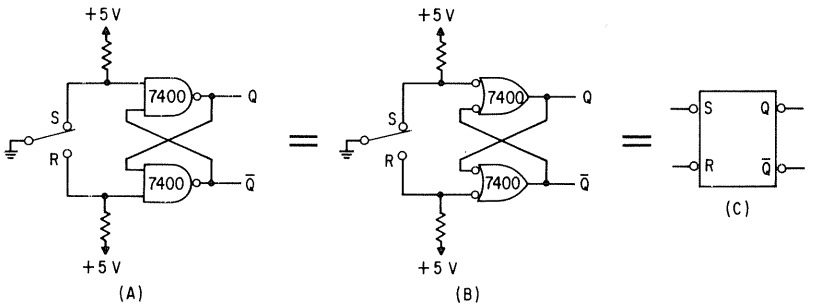


Fig. 3-2. The R-S flip-flop constructed with NAND gates is often shown functionally (B and C).

THE T FLIP-FLOP AND THE COUNTER

The *toggling flip-flop* (*T-flip-flop*) is shown in Fig. 3-3. It consists of a basic R-S flip-flop (gates C and D) and a set of control gates (A and B) which steer a pulse from the T input to the flip-flop to cause it to switch states, i.e., to *toggle*.

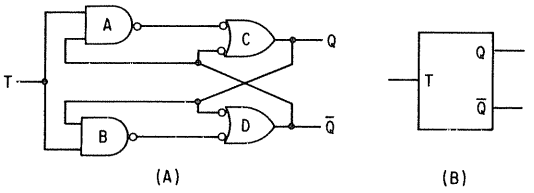


Fig. 3-3. The T flip-flop.

For example, if the flip-flop is reset, a positive pulse (1) at the T input will cause gate A's output to go to 0, setting the flip-flop. If the flip-flop is set, then gate B's output goes to 0 when a 1 occurs at the T input.

We can observe the action of the circuit with an oscilloscope, when a recurring pulse is fed to the T input. In this case we would observe the waveforms shown in Fig. 3-4. Notice that every time T goes to a 1 the flip-flop

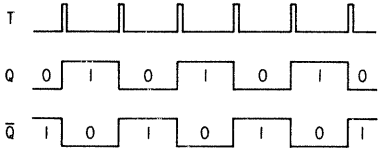


Fig. 3-4. Waveforms generated by a T flip-flop.

toggles. Further, the frequency of the Q and $\bar{Q}$ outputs is one-half that of the T input. In other words, the flip-flop divides the input frequency by 2. Note that the pulse duration time (1) must be very short for this circuit to operate properly.

If we connect two T flip-flops in series (Fig. 3-5) then we can divide the frequency by 4 (Fig. 3-6). We can continue to add flip-flops to increase the frequency division. For each flip-flop added, we divide the previous frequency by 2. Thus, three flip-flops divide by 8, four divide by 16, and so on.

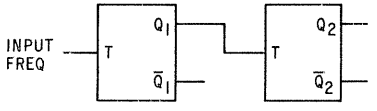


Fig. 3-5. Input freq ÷ 4 counter.

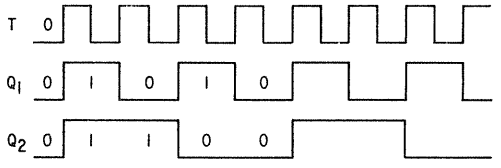


Fig. 3-6. Waveforms at Q outputs of a ÷ 4 counter.

A 5-flip-flop circuit will produce one output pulse for every 32 input pulses. Hence, we say that it counts 32 pulses. Thus, these circuits are more often referred to as *counters*.

THE CLOCKED R-S FLIP-FLOP

Most digital logic systems operate in a step-by-step manner. In other words, operations must be synchronized with one another. This method of operation is called *clocked* or *synchronous* logic. Nothing happens until the synchronizing clock pulse occurs.

An R-S flip-flop (Fig. 3-7) can be controlled by a clock pulse input to synchronize its operation with other activities occurring in the logic system.

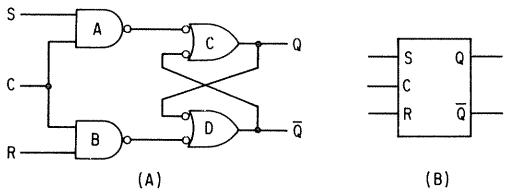


Fig. 3-7. The clocked R-S flip-flop.

Gates C and D form the R-S flip-flop. Gates A and B are clock control gates. When the C (clock) input is 1, the control gates are enabled and the gate is set or reset, depending on the S and R logic levels.

Figure 3-8 illustrates the operation of the clocked R-S flip-flop. Notice that if S and R = 1 both Q and $\overline{Q}$ = 1, and after the clock pulse we do not know what logic state will occur. This is called an *indeterminate condition.* It is undesirable and measures are usually taken to avoid this condition (usually using a J-K flip-flop, to be discussed shortly).

Notice that the Q and $\overline{Q}$ outputs do not change until the clock = 1. For this reason the clock pulse is made very narrow to avoid changing Q and $\overline{Q}$ levels, due to changing R and S control levels.

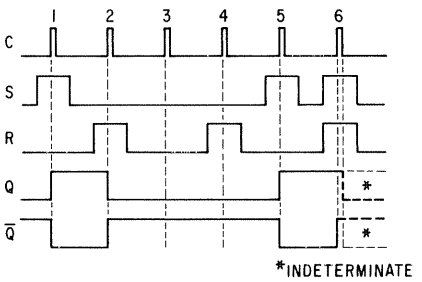


Fig. 3-8. Waveforms generated by a clocked R-S flip-flop.

THE D FLIP-FLOP

The *data flip-flop* (*D flip-flop*), frequently referred to as a *latch,* is shown in Fig. 3-9. The D flip-flop is used to temporarily store data. Gates C and D form

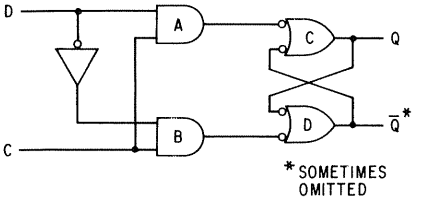


Fig. 3-9. The D flip-flop.

an R-S flip-flop, while gates A and B control the flip-flop. When the C (clock) input = 1, the control gates are enabled and the logic level at the D-input will cause the flip-flop to set or reset. When D = 1, the flip-flop will set (A = 1) with the occurrence of the clock pulse. When D = 0, the flip-flop will reset (Q = 0) with the clock pulse. This is shown in the timing diagram of Fig. 3-10.

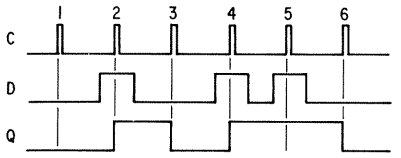
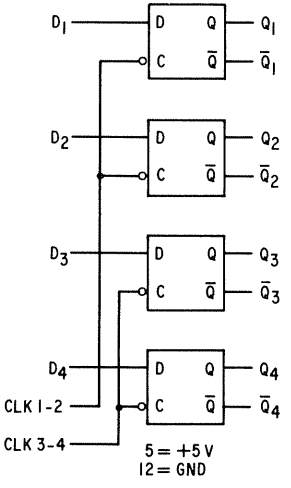
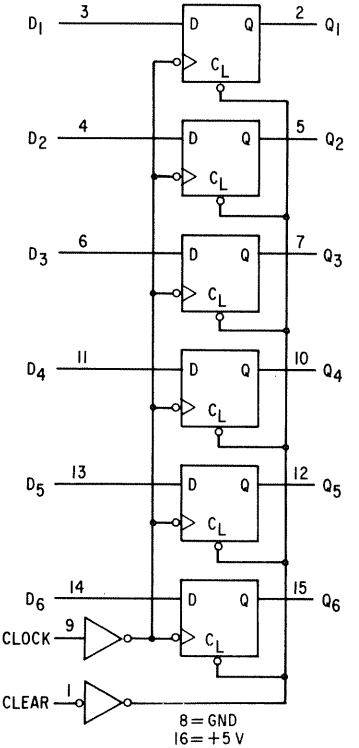


Fig. 3-10. Waveforms generated by a D flip-flop.

Fig. 3-11. The 7475 quad latch and 74174 hex D flip-flop.





Again, the D flip-flop temporarily stores data. For example, it is used at the input-output port of a computer to latch data from the bus to the port. Two widely used ICs for this purpose are the 7475 quad latch and 74174 hex D flip-flop (edge-triggered). The latter is often used in computer data bus applications. The reason for this will be discussed shortly. The pin-out diagrams are shown in Fig. 3-11.

Note that a D flip-flop is sometimes used as a T flip-flop by connecting the $\overline{Q}$ output back to the D input. This is shown in Fig. 3-12A; and a controllable T

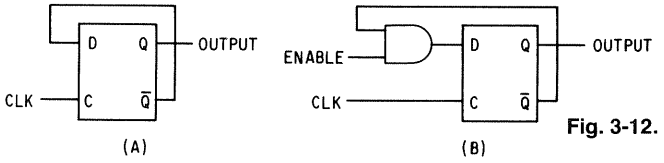


Fig. 3-12. Converting a D to a T flip-flop (A) and a controllable T (B).

flip-flop is shown in Fig. 3-12B. When the enable input = 1 the flip-flop will toggle on each clock pulse. When enable = 0 the flip-flop cannot get its feedback signal and hence cannot toggle.

THE J-K FLIP-FLOP

The J-K flip-flop is the most versatile and, hence, the most widely used type flip-flop. It has no indeterminate states and can be made to perform the functions of all the flip-flops already mentioned. A simple J-K flip-flop is shown in Fig. 3-13A. When J = 1 and K = 0 the flip-flop will set on the clock pulse.

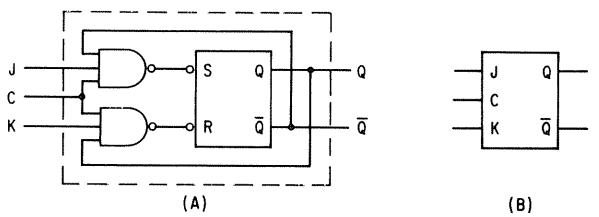


Fig. 3-13. The simple J-K flip-flop.

When J = 0 and K = 1 the flip-flop will reset on the clock pulse. When J and K = 1 it will toggle and when J and K = 0 it will not change. The functional symbol for a J-K flip-flop is shown in Fig. 3-13B.

The simple J-K flip-flop requires a very short clock pulse to prevent triggering problems and, hence, a more complex circuit is often used to prevent these problems. It is called the *Master-Slave J-K flip-flop* and is sometimes called an *M-S J-K flip-flop* (Fig. 3-14).

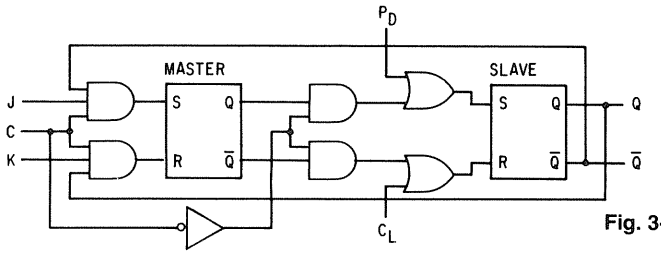


Fig. 3-14. The master-slave type J-K flip-flop.

The master-slave type has two R-S flip-flops coupled together. The controlling flip-flop is called the *master* and the controlled flip-flop is the *slave*. The master remembers the input control logic levels and, therefore, timing is not critical.

The J and K inputs are directly coupled to the master R-S flip-flop through AND gates. Therefore, when the clock goes from 0 to 1, the J-K control levels are transferred to the Q and $\bar{Q}$ of the master. The slave is *inhibited* (prevented from operating). The slave is enabled when the clock goes from 1 to 0 and the master now controls the slave. Hence, the flip-flop changes states only on the negative-going edge of the clock pulse.

In addition, the slave can be controlled by direct inputs to the slave which are not clocked. These inputs are labeled P_D (Preset data) and C_L (Clear). These inputs must = 0 for normal J-K operation.

The P_D and C_L inputs operate as follows: when $P_D = 1$ and $C_L = 0$ the flip-flop will set ($Q = 1$, $\bar{Q} = 0$). When $P_D = 0$ and $C_L = 1$ the flip-flop will reset ($Q = 0$, $\bar{Q} = 1$). These control inputs are not clocked and in fact override the J and K inputs. When P_D and $C_L = 0$ we get normal J-K operation. P_D and $C_L = 1$ is a prohibited condition and, therefore, must be avoided since it will cause indeterminate operation.

EDGE-TRIGGERED FLIP-FLOPS

Edge-triggered flip-flops operate on the *rising* or *falling edge* of a clock pulse (Fig. 3-15). The leading edge is often called the *positive edge* and the falling edge is often called the *trailing* or *negative edge* of the clock pulse. The upper Q output in Fig. 3-15 is produced by a positive edge-triggered flip-flop, while the lower Q output is produced by a negative edge-triggered flip-flop.

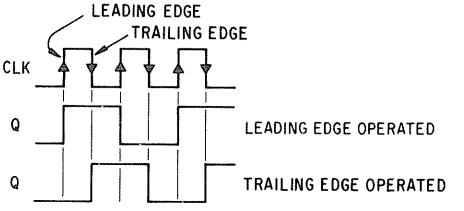


Fig. 3-15. Comparison of Q outputs from flip-flops operating on leading and trailing edges.

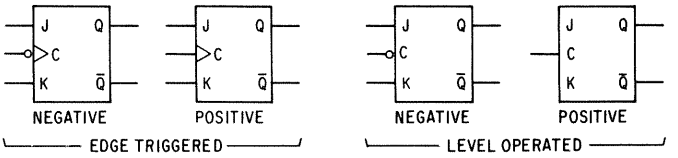


Fig. 3-16. Schematic symbols for edge- and level-operated flip-flops.

Edge triggering is indicated by a small arrow at the clock input (Fig. 3-16). A bubble and arrow indicates a flip-flop whose outputs change on the negative edge of the clock pulse. The arrow only indicates positive-edge operation; a bubble only indicates low-level operation; and neither bubble nor arrow indicates high-level operation.

Figure 3-17 illustrates some of the popular J-K flip-flops in current use. Note that J-K flip-flops are available with multiple J and K inputs and negated J or K inputs.

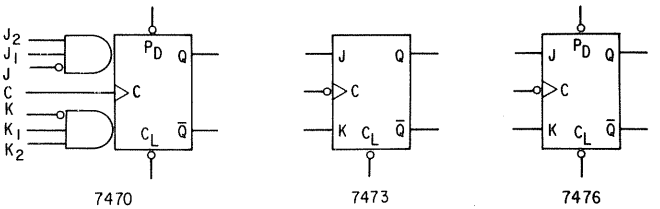


Fig. 3-17. Three widely used J-K type flip-flops.

USING J-K FLIP-FLOPS AS D AND T FLIP-FLOPS

The J-K flip-flop can be used as a D or T type flip-flop as shown in Fig. 3-18. The D flip-flop is accomplished by feeding complemented data to the J and K inputs. The T is accomplished by keeping the J and K inputs always = 1.

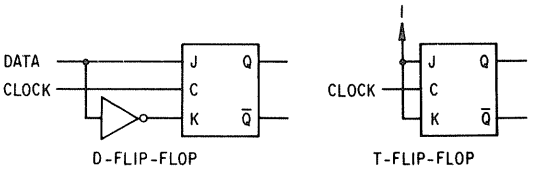


Fig. 3-18. Wiring J-K flip-flop to operate as D and T flip-flops.

ONE-SHOTS

The *one-shot (OS)* is really a flip-flop. However, it has only one stable state. It can be triggered into its opposite state but remains there only for a limited time, and then returns to its initial stable state. It is used primarily to create pulses of a known duration from pulses of an unknown duration.

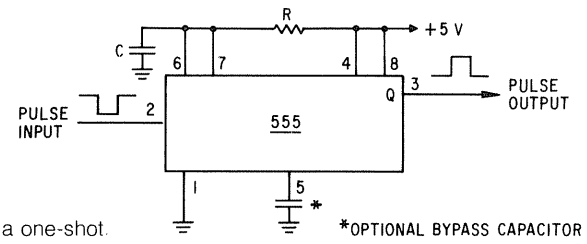


Fig. 3-19. 555 timer used as a one-shot.

The 74121, 74122, and 74123 are the most widely used OS ICs. Also, the 555 timer is often used as an OS. The 555 and 556 (a dual 555 IC) are used to develop pulse widths of microseconds to hours. The 74121, 74122, and 74123 are used for developing pulses of nanoseconds to microseconds.

Figure 3-19 shows the 555 wired as an OS. The output pulse width is determined by the values of R and C. The time of the output pulse = $1.1 \times R \times C$. Thus a 1-MΩ (megohm) resistor and 1-μF (microfarad) capacitor will cause a pulse width to be developed which is 1.1 seconds. This output pulse will be produced by a negative pulse at the input. The 555 is not suitable for developing short pulses because the triggering pulse must be significantly narrower than the output pulse for proper operation of the circuit. The 74121, 74122, and 74123 ICs are specifically designed to be used as OS devices. Figure 3-20 illustrates how each is wired to operate as a simple OS. The output pulse width = $0.7 \times R$

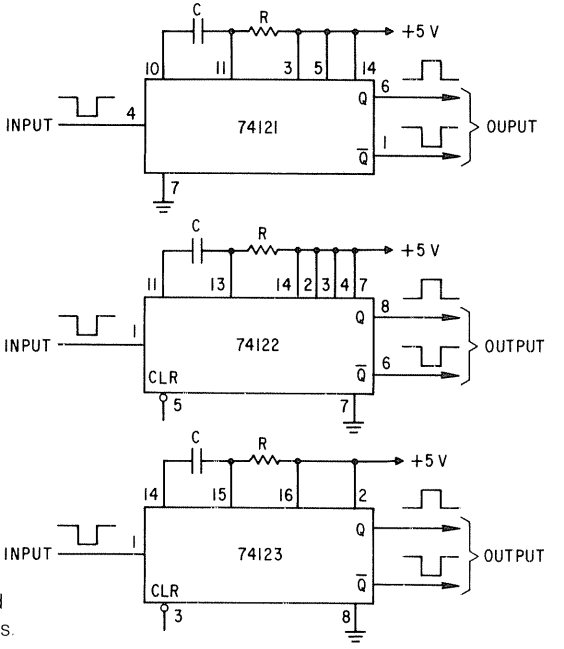


Fig. 3-20. The 74121, 74122, and 74123 TTL one-shot ICs.

× C. Thus a 1-μF capacitor and 10-K Ω (kilohm) resistor will provide a pulse = 7 ms (milliseconds). The 74121 and 74122 are single OS devices and the 74123 is a dual OS device. The 74122 and 74123 are retriggerable, before the output pulse ends, to extend the duration of the pulse.

CLOCK CIRCUITS

A *clock* circuit is actually an oscillator providing a continuous pulse signal. It is used to synchronize the operation of a computer or logic system. It sees that things happen at the right time, hence the name clock. It is usually a *free-running* type flip-flop in which feedback of an in-phase signal occurs from output to input.

An example is the clock circuit employed in the popular Altair 8800 CPU (Central Processor Unit) shown in Fig. 3-21. The clock oscillator is made up

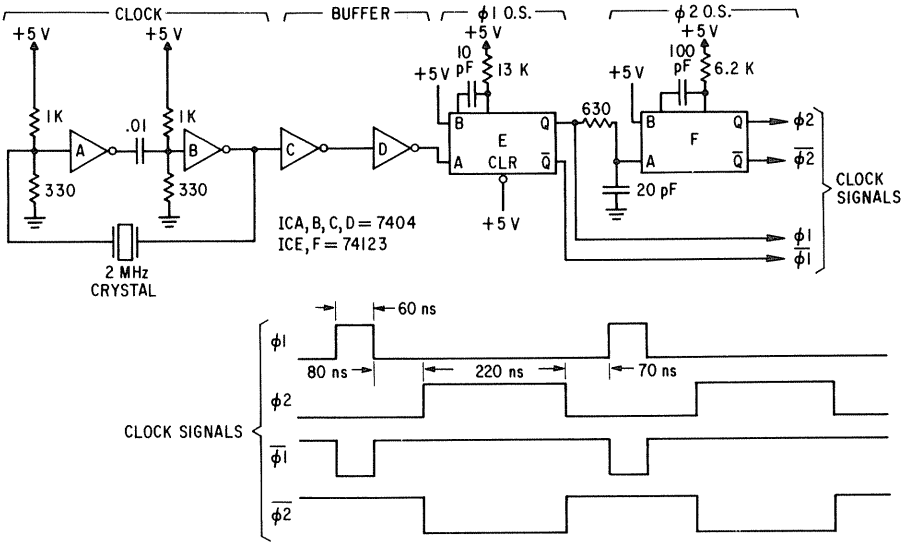


Fig. 3-21. The clock circuit used in the Altair-8800 central processor unit.

of two inverters (A and B) with the output of B fed back to A. Notice that B's output will be in phase with A's input; hence positive feedback occurs to sustain oscillation. A crystal, cut to resonate at 2 MHz, is in the feedback path and hence oscillation occurs only at this frequency. A crystal has excellent frequency stability under varying conditions of temperature and humidity. It ensures a constant clock frequency for the timing of the CPU. The resistors at the input of each inverter make the crystal operate like an amplifier instead of a gate, which is necessary for oscillation to occur.

Inverters C and D act as *buffers* for the oscillator so that a constant load always exist on the oscillator. This prevents frequency shifting due to varying loads.

Two one-shots (E and F) are used to develop the proper width of output clock signals. The microprocessor IC (8080) requires two clock signals (Ø1 and Ø2) which are of different pulse widths and spacing. OS-E develops a 60-ns pulse (Ø1) and triggers OS-F through a small resistor–capacitor delay circuit. OS-F produces a 220-ns pulse (Ø2).

The 555 timer is frequently used as a clock. A typical 555 clock circuit is shown in Fig. 3-22. The frequency of the clock is determined by the equation:

$$f = \frac{1.44}{(R_A + 2R_B) \times C}$$

The 555 clock is used in applications where frequency stability is not critical. It is capable of driving a heavy load.

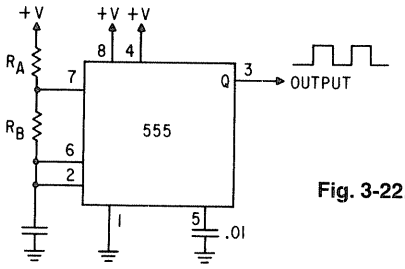


Fig. 3-22. A 555 timer used as a clock generator.

COUNTERS

Counters are circuits made up of flip-flops that are used to count pulses. The most widely used counter ICs are the 7490, 7492, and 7493. All consist of four flip-flops which are wired internally to count 10, 12, or 16 pulses. The 7490

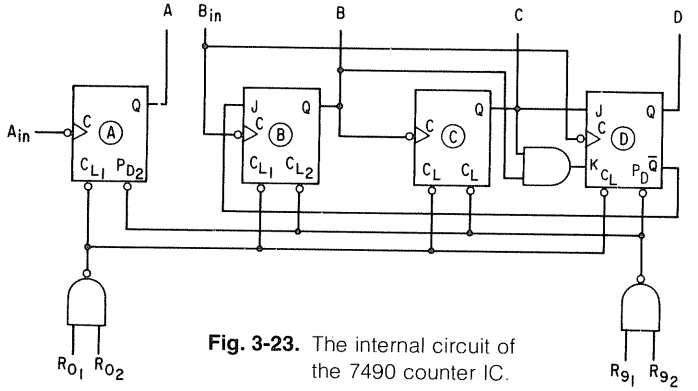


Fig. 3-23. The internal circuit of the 7490 counter IC.

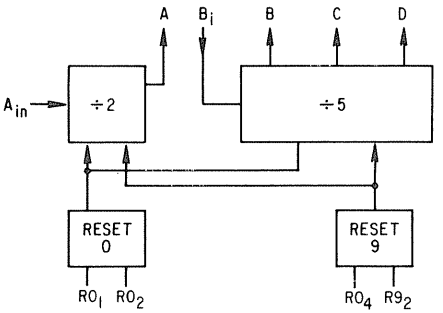


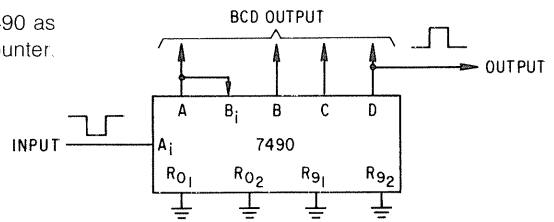
Fig. 3-24. Block diagram of 7490 circuit.

is shown in Fig. 3-23. A simplified block diagram of the circuit is shown in Fig. 3-24. Flip-flop A is a simple J-K flip-flop which divides the signal by 2. Flip-flops B, C, and D divide the B_{in} signal by 5. If the A output is connected to the B_{in} (Fig. 3-25), the 7490 will divide the A_{in} signal by 10. The 7490 is, therefore, referred to as a *divide-by-ten* or *decade* counter. For every 10 input pulses there will be one output pulse. A BCD count is produced at the ABCD outputs.

The R_0 and R_9 inputs are reset inputs. When both R_0 inputs = 1 the counter resets to a decimal count of 0 (0000 binary). When both R_9 inputs = 1 the counter resets to a decimal count of 9 (binary 1001).

The 7492 and 7493 are similar to the 7490 except that they provide counts of 12 and 16, respectively.

Fig. 3-25. Using the 7490 as a decade counter.



SHIFT REGISTERS

Shift registers (*SR*) are flip-flop circuits used for a wide variety of logic operations. They are used most frequently in computer systems for storing data, converting serial data to parallel data, and vice versa. A simple shift register using D flip-flops is shown in Fig. 3-26A. Data is shifted from one flip-flop to the next with each clock pulse. The shifting of pulses is shown in Fig. 3-26B. A reset input serves to clear the register (set all Qs = 0).

Notice that in Fig. 3-26 it takes four clock pulses to clock the serial word into the register. When the word is in the register, we may read it as a parallel word at the ABCD outputs. Likewise, it will take four clock pulses to clock the word out of the register. The serial output will be taken at the D output. Figure 3-26C shows the functional symbol for the shift register.

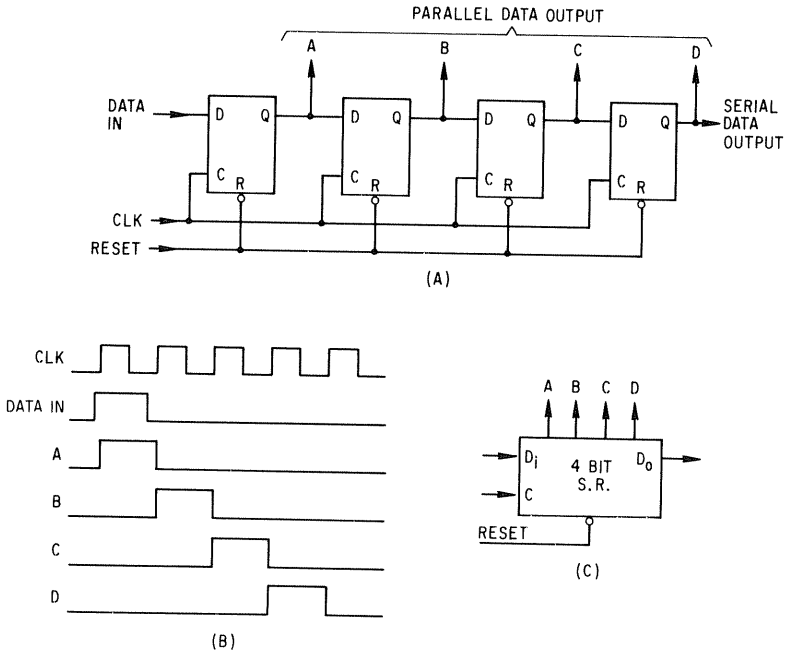


Fig. 3-26. A 4-bit shift register.

Shift registers that can be loaded in parallel, and hence can be used for parallel-to-serial conversion, are available (Fig. 3-27). Further, shift registers are available to store up to several thousand bits. These large shift registers are used as *buffer* (temporary) storage circuits when converting from one transmission rate to another.

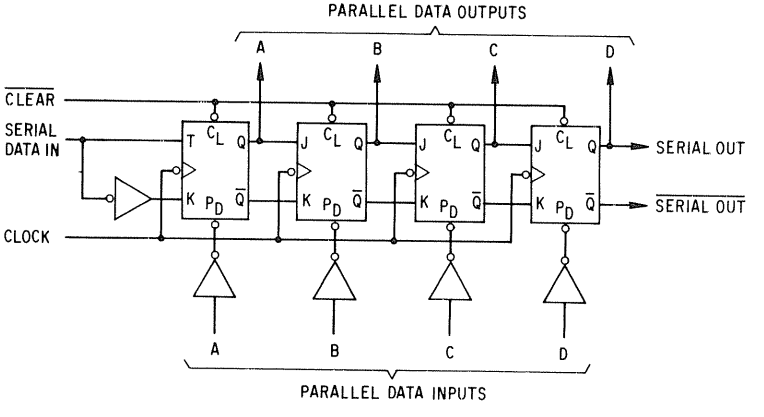


Fig. 3-27. Shift register with parallel I/O and serial I/O.

Recommended Further Reading

1. Sol Libes, *Fundamentals and Applications of Digital Logic Circuits,* Revised Second Edition, Hayden Book Co., Inc., Rochelle Park, N.J., 1978.
2. Donald E. Lancaster, *TTL Cookbook,* Howard W. Sams & Co., Inc., Indianapolis, Ind., 1974.
3. Donald E. Lancaster, *CMOS Cookbook,* Howard W. Sams & Co., Inc., Indianapolis, Ind., 1977.
4. *Digital, Linear, MOS Data Book,* Signetics Corp., Sunnyvale, Calif., 1976.

4.

An Introduction to Computer Systems

THE CPU—THE SYSTEM CONTROL CENTER

A computer is really a computing and processing system. It contains the basic components shown in Fig. 4-1. There is *input* and *output,* which permits instructions and data to enter the system and allows data and control to exit from the system. The *central processor (CPU)* is entirely electronic. It does arithmetic and logic operations on the data and controls the operation of the system. *Storage* is where we can store data and/or instructions when the computer is not being used.

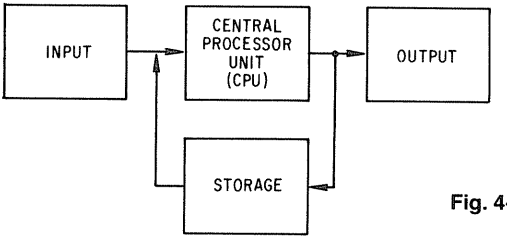


Fig. 4-1. Basic computer system—block diagram.

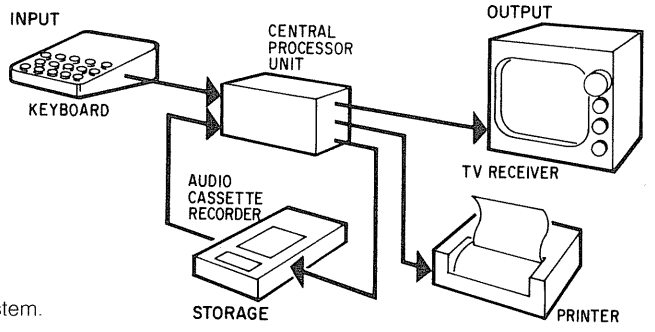


Fig. 4-2. Typical home computer system.

Figure 4-2 illustrates a typical home computer system. The input is from a typewriter style electronic keyboard. The output can go to either a TV display or printer. The storage is an audio cassette type tape recorder. The CPU is microprocessor based. In this chapter we examine the basic principles of the CPU. Later chapters will go into input-output and storage.

What Is a CPU?

The CPU is the electronic heart of a computer system. Most of the other parts of a computer system are electro-mechanical and are dependent on the CPU. The CPU processes data. In other words, it performs arithmetic and logic operations on data presented to it. In this book we are concerned with microprocessor-based CPUs. *Microprocessors* are called *MPUs* for short.

A basic MPU-based CPU is shown in Fig. 4-3. An MPU is a single IC which contains most of the circuitry of a CPU. Referring to the illustration, notice that the MPU has four basic sections: the *ALU* (*arithmetic logic unit*), the *registers,* the *instruction decoder/timing unit,* and the clock. The MPU communicates with *memory* (*RAM, random-access memory,* and *ROM, read-only memory*) and *I/O interface circuits* via *data* and *address buses* and control lines (R/W and interrupt).

Notice that the various sections communicate with one another via three paths, usually called *buses*. The three buses are the *data, address,* and *control buses*. Each bus consists of a set of parallel lines and signals that are transmitted on these buses in parallel fashion. Arrows are used to denote the direction of signal flow. In many systems, signals flow in both directions on the data bus but not at the same time.

The data bus transmits data and instructions. The address bus transmits addressing information from the MPU to memory (RAM and ROM) and I/O to select memory locations or input-output devices. The control bus in its very simplest form consists of lines which control memory and I/O transfers, interrupts (more of this in a later chapter), and resetting of the system.

Signals on the data bus may be either an instruction or data. The instruction decoder-timing section decides which it is. If it is an instruction, it controls the registers and ALU, routing the data to and from them, and causing the desired arithmetic or logic operation to be performed. A clock circuit, which is usually internal to the MPU, enables the timing circuit to synchronize the operation of the various sections.

The data and instructions on the data bus enter the MPU and are latched into a buffer register. From this register they are transferred to other registers in the MPU or processed through the ALU. A typical set of MPU registers is shown in Fig. 4-4.

Depending upon whether it is a data or instruction signal, data coming into the MPU is either directed into the data bus buffer or instruction registers. For example, data transferred from memory to the accumulator register will first

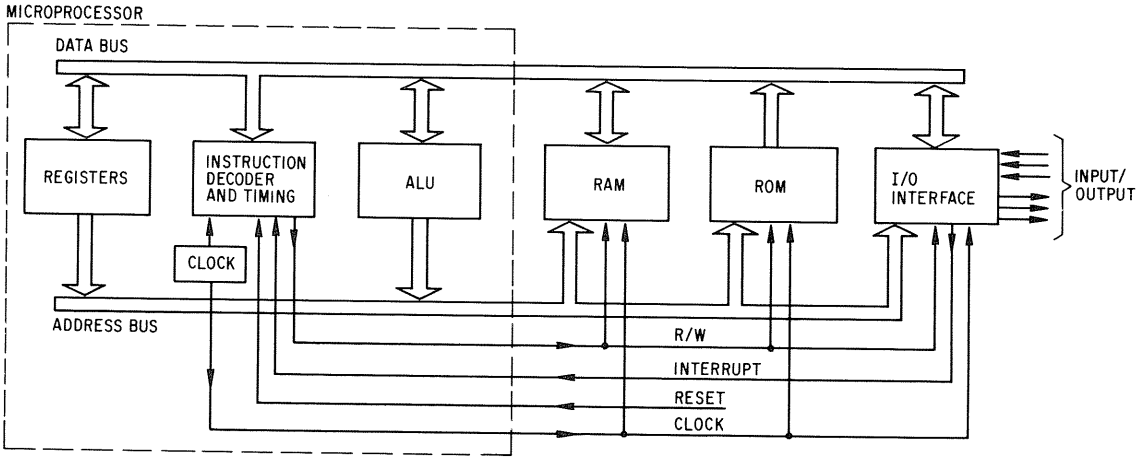


Fig. 4-3. CPU—block diagram.

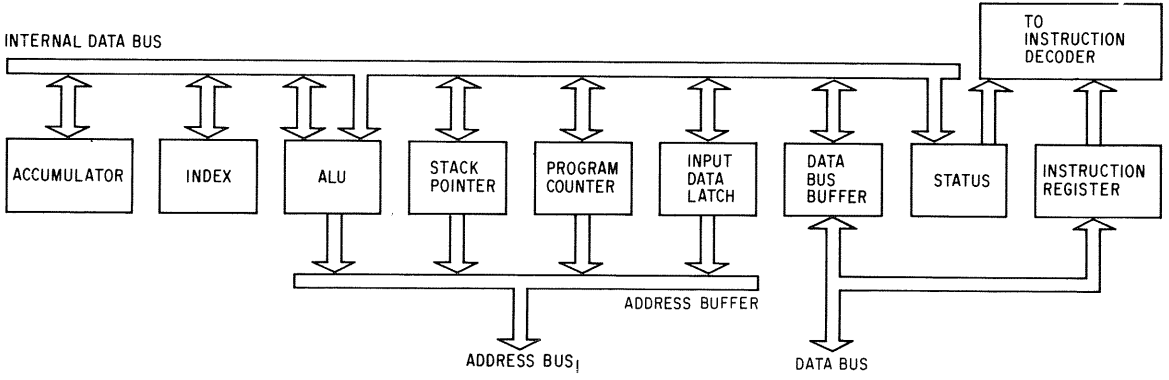


Fig. 4-4. Typical register and ALU organization of an MPU.

enter the input data latch and from there be transferred to the accumulator register. If an arithmetic or logic operation is to be performed using data from memory and the accumulator register, data is first transferred into the input data latch and from there to one input of the ALU. At the same time, the contents of the accumulator are transferred, via the bus, to the second input of the ALU. The results of the arithmetic or logic operation performed in the ALU are transferred back to the accumulator, via the data bus.

The program counter (PC) register provides the addresses which step the MPU through sequential instructions in the program stored in memory. Each time the MPU fetches an instruction from program memory, the contents of the PC register are placed on the address bus. This PC register is incremented each time an instruction is fetched from memory.

The accumulator (A) register is a general purpose register which stores the results of most arithmetic and logic operations. In addition, the accumulator usually contains one of the two words used in these operations.

Most logic and arithmetic operations take place in the ALU, which has two inputs (Fig. 4-5). These inputs come from internal registers or memory via an

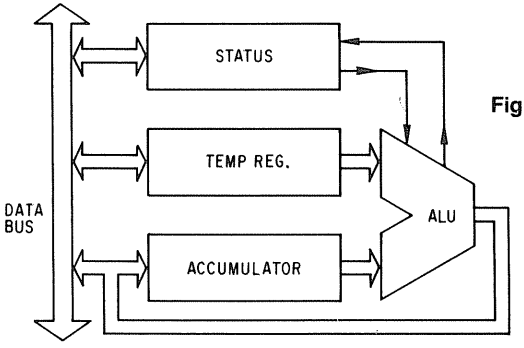


Fig. 4-5. ALU section—block diagram.

internal register and the output usually goes to the A register. The ALU performs arithmetic (addition and subtraction) or logic (AND, OR, NOT, X-OR, etc.) functions on all the bits at each input. For example, if the words at the input to the ALU are 01010101 and 00110011 and the ALU is instructed to perform the AND operation, the operation will be as shown in Fig. 4-6.

Notice in this hypothetical example that each register contains an 8-bit word and the status register contains a 5-bit word. The respective bits in the two

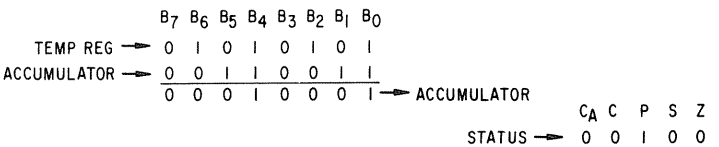


Fig. 4-6. Example of ALU performing an AND operation.

registers are ANDed together and the result returned to the accumulator. For example B_0 (shorthand notation for bit 0) is the ANDing of 1 and 1 = 1, B_1 is the ANDing of 0 and 1 = 0, and so on.

The *status register* is affected by the ALU operations. Typically the status register has five bits associated with ALU operations. They are zero, sign, parity, carry, and half carry, and are each represented by 1 bit in the status register. These bits are often called *flag* bits and are either set (1) or reset (0) as the result of ALU operations. They operate typically as follows:

Zero (Z): If ALU operation results in 0 in the accumulator, 0 flag is set, otherwise it is reset.
Sign (S): If ALU operation causes $B_7 = 1$, sign flag is set, otherwise it is reset.
Parity (P): If ALU result has even number of 1s in word, parity flag is set (even parity), otherwise it is reset (odd parity).
Carry (C): If ALU operation results in a carry or borrow out of B_7, carry flag is set, otherwise it is reset.
Half Carry (C_A): Sometimes called the *auxiliary carry*. If ALU operation causes a carry between bits B_3 and B_4, half carry flag is set, otherwise it is reset.

The *stack pointer* (*SP*) and *index* (*I*) registers store data used in calculating addresses in memory. The operation of these registers will be discussed later.

CPU Memory

The CPU contains memory to store program instructions and data. The memory is either a *read-only memory* (*ROM*) or *read-write memory*, usually referred to as *random-access memory* (*RAM*). In point of fact, both ROM and RAM can be randomly accessed and the misnomer RAM was given in the early days of computers and now it is too late to change.

ROM and RAM are electronic circuits that store data or instruction words (Fig. 4-7). The ROM is programmed permanently and cannot be changed (some

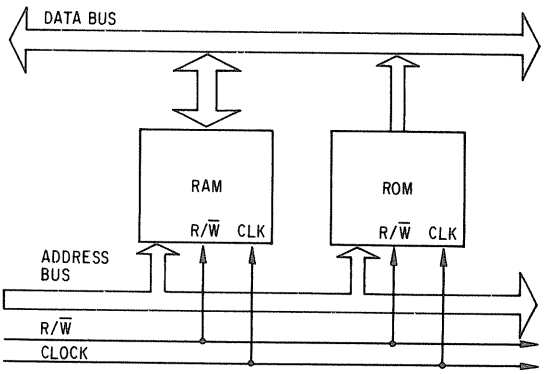


Fig. 4-7. The memory section of the CPU.

can, but not easily). Hence, we only read instruction words from ROM. RAM is easily changed by writing a new word into a specific memory location called an *address*. When communicating with ROM or RAM, the MPU addresses the memory via the address bus, indicates a read or write command via the R/$\overline{W}$ line, and sets the R/$\overline{W}$ time period via the clock line.

The data is transferred to and from the memory via the data bus. In most MPU-based CPU systems the data bus is 8 lines and the address bus is 16 lines. We will examine memory in greater detail later.

Input/Output

The CPU communicates with peripheral units such as a keyboard and display via appropriate interface circuits. These circuits match the characteristics of the CPU and I/O unit. For example, a *Teletype (TTY)* is a popular terminal for use with computer systems. The CPU operates in a parallel fashion, outputting and inputting all the bits in a word at the same time. A TTY operates in serial fashion, outputting and inputting one bit at a time. Further, the TTY operates at a fixed speed which is much slower than the CPU. These differences, as well as some others, which we will discuss in detail later, necessitate an interface circuit between the CPU and I/O unit.

As shown in Fig. 4-8, the I/O interface section contains addressing to select one I/O device and data bus communication. The R/$\overline{W}$ line is usually used to indicate input (read) or output (write).

Further, an interrupt circuit, which permits an I/O device to interrupt the operation of the CPU, is often provided. When an I/O device initiates an interrupt, the CPU finishes the instruction it was working on, saves the data in the MPU registers in a special area in memory (called the "stack" area), executes an interrupt program, and then returns to the original program. We discuss interrupts in more detail later.

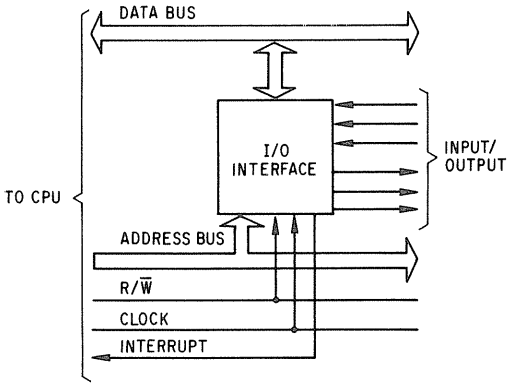


Fig. 4-8. The I/O section of the CPU.

HARDWARE VERSUS SOFTWARE

A computer, no matter how sophisticated, can only do what it is *told* to do. You *tell* the computer what to do via a series of coded instructions referred to as a *program*. The realm of the *programmer* is the writing of these coded instructions, referred to as *software*.

In contrast, *hardware* is the actual physical computer equipment, the CPU and associated peripheral devices such as terminals. The CPU has designed into it the ability to perform a particular set of operations. The CPU is designed such that a specific operation is performed when the CPU control logic decodes a particular instruction. Consequently, the operations that can be performed by a CPU define the computer's *instruction set*.

THE CPU INSTRUCTION SET

Each computer instruction allows the programmer to initiate the performance of a specific operation. All computers implement certain arithmetic operations in their instruction set, such as an instruction to add the contents of two registers. Often logic operations (e.g., ORing the contents of two registers) and register operation instructions (e.g., increment a register) are included in the instruction set. A computer's instruction set will also have instructions that move data between registers, between a register and memory, and between a register and an I/O device. Most instruction sets also provide *conditional instructions*. A conditional instruction specifies an operation to be performed only if certain conditions have been met, e.g., jump to a particular instruction if the result of the last operation was zero. Conditional intructions provide a computer with a decision-making capability.

COMPUTER PROGRAMMING

By logically organizing a sequence of instructions into a coherent program, the programmer can tell the computer to perform a very specific and useful function.

The computer, however, can only execute programs whose instructions are in a binary coded form (i.e., a series of 1s and 0s) that is called *machine code*. Because it would be extremely cumbersome to program in machine code, programming languages have been developed. There are programs available which convert the programming language instructions into machine code that can be understood by the processor.

One type of programming language is *assembly language*. A unique assembly language mnemonic is assigned to each of the computer's instructions. The programmer can write a program (called the *source program*) using these mnemonics and certain operands; the source program is then converted into

machine instructions (called the *object code*). Each assembly language instruction is converted into one machine code instruction (1 or more bytes) by an assembler program. Assembly languages are usually machine dependent (i.e., they are usually able to run on only one type of computer).

Higher level languages are available which enable the computer user to communicate with the computer with words very similar to spoken commands. The most popular such language is called *BASIC*. This language is machine independent; in other words it is essentially the same on all computer systems.

MICROCOMPUTERS VERSUS LARGE COMPUTERS

A *micro* is very small in physical size. In most respects it compares quite favorably with *mini* and *large* computer systems. Today's microcomputers are more powerful and faster than many of the large-scale computers that existed just a few years ago. In fact, some microprocessor-based computer systems are more powerful than some large computer systems. A single user may notice little, if any, difference when communicating with a micro, mini, or large computer system. The difference lies in the fact that large computers can handle many users while micro systems are seldom worthwhile for more than one or two users.

Table 4-1 gives a brief comparison between the 8080 microprocessor and an IBM-370 CPU.

Table 4-1.

	8080/8085	*IBM-370*
PC Register:	16 bit	24 bit
Op Code Register:	8 bit	48 bit
Accumulator:	8 bit	4–64 bit (floating point)
SP Register:	16 bit	—
Status Register:	5 bit	64 bit
General Purpose Registers:	6–8 bit byte	16–32 bit byte
Basic Word Size:	8 bit	8 bit
Op Code Format:	1, 2, or 3 bytes	2, 4, or 6 bytes
Maximum Memory Size:	65, 536 bytes	16,772,216 bytes
Memory Addressing:	Direct Register Indirect Immediate	Direct Indexed Relative Register Indirect
Instructions (basic):	78	143
Unique Op Codes:	244	Over 50,000 possible
I/O:	Up to 256 I/O ports	Up to 256 I/O channels with up to 256 devices per channel
Interrupts:	One mode up to 8 direct levels	Multiple mode priority direct vector to any location

It is also interesting to compare some of the characteristics of very large, large, medium, mini, and microcomputer CPUs. Here is such a comparison.

	Very Large CRAY-1	*Large* IBM 370/168	*Medium* IBM 370/115	*Mini* PDP-11/70	*Micro-Mini* PDP-11/03	*Micro* Altair 8800
No. of ICs:	278,000	20,000	1,800	600	400	200
Cycle time:	12½ ns	8 ns	480 ns	300 ns	750 ns	500 ns
RAM/ROM max words:	1,048,576	8,388,608	393,216	4,096,000	57,344	65,536
Word size:	64 bits	8–64 bits	8–64 bits	16 bits	16 bits	8 bits
Physical size:						
Width—	9 ft	13 ft	2½ ft	21 in.	19 in.	19 in.
Depth—	4½ ft	10 ft	5 ft	31 in.	13½ in.	17 in.
Height—	6½ ft	6½ ft	5 ft	6 ft	3½ in.	7 in.
Weight:	5¼ tons	5,100 lb	1,800 lb	500 lb	35 lb	22 lb
Cost (typ.):	$8,000,000	$4,500,000	$175,000	$63,000	$2,000	$1,600

Recommended Further Reading

1. Adam Osborne, *An Introduction to Microcomputers,* Adam Osborne & Assoc., Inc., Berkeley, Calif., 1975.

5.

To and From RAM and ROM

In the last chapter we learned that a computer cannot operate without a memory. The memory is that section of the CPU where we store data and instructions. The memory is basically an array of bi-stable storage elements. The elements can store either logic-0 or logic-1 states.

Earlier computers employed magnetic core read/write type memories. These have the advantage of retaining their states when power is shut down. These memories are, therefore, said to be *nonvolatile*. However, most computer systems today employ semiconductor memories, which unfortunately lose their stored data when the electric power is turned off. These memories are said to be *volatile*. This is true of RAMs; however, ROMs do not lose data when power is lost.

The semiconductor type memory offers the advantages of lower cost, smaller size, and lower power consumption. Hence, the likelihood is that semiconductor memories will eventually replace core memories entirely. Therefore, we will concentrate on semiconductor memories.

MEMORY BASICS

Memory may be either *RAM* or *ROM*. The *ROM* (*read only memory*) is permanently encoded and cannot be changed. It is used to store program instructions. The *RAM* (*random access memory*) can be changed. It is possible to write into any RAM location or to read what is in any RAM location. The RAM is used for storage of either program instructions or data.

In a memory, every binary word, consisting of a group of bits, has a unique address. Most microprocessor-based systems use an 8-bit word size. This 8-bit data unit is also referred to as a *byte*. If a word consists of 16 bits, it is said to contain 2 bytes.

MEMORY ADDRESSING

Each data byte stored in memory has a unique address. For example, a popular semiconductor memory IC is the 2102. The 2102 can store 1,024 bits. Therefore, a memory to store 1,024 8-bit words would require eight 2102s (Fig. 5-1). If the first word in the memory were selected, it would be at location 0, as shown in Fig. 5-1. Notice that there are 8 planes in the memory system with 1,024 bits per plane. When we address location 0 in the memory, we are addressing location 0 on each plane. The memory plane at the extreme right contains the B_0 data bits; the next plane contains the B_1 data bits, etc., until the highest order bit, B_7, is on the extreme left. Hence, there are 1,024 addressable locations of 8 bits each.

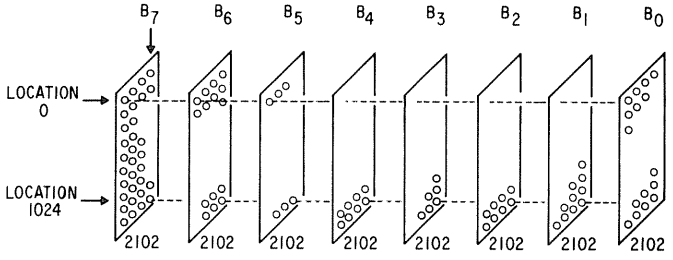


Fig. 5-1. A 1,024-word memory using 2102 ICs.

A 2102 IC is shown in Fig. 5-2. It is a 16-pin IC: 10 pins are used for addressing the 1,024 locations, 2 pins are used for data in and out, 1 pin is used for read/write control, and 1 pin is used for enabling the chip ($\overline{CE}$). The $\overline{CE}$ permits expansion of the memory system beyond the 1,024-bit size.

An example of a 4 K-RAM memory using the 2102 RAM IC is given in Figs. 5-3 through 5-6. This is a RAM board kit manufactured by Southwest Technical Products Co. for their 6800 microcomputer. The board is organized into four pages containing 1K words, hence 4K words total. As illustrated in Figs. 5-4 and 5-5, the 10 address lines A_0–A_9 and R/$\overline{W}$, via buffers, go to all RAM ICs in parallel. The buffers are used to reduce the effects of capacitance paralleling so many ICs, allowing the memory to operate at its rated speed.

The page addressing (one of four) is done by IC-23 (2- to 4- line decoder) decoding address bits A_{10}–A_{11}. Hence when A_{10}–A_{11}=00 the first page of 1,024 words is selected.

Address bits A_{12}–A_{15} are decoded by IC-22 to select RAM boards. Therefore, up to eight 4K-RAM boards may be used in the computer with a maximum memory of 32K. Notice that only the lower two pages are shown on the schema-

tic. The upper two pages are the same as the bottom two with the control lines as indicated.

Since the 8-bit data bus for the computer system is bi-directional, bi-directional transceiver/buffers IC-20 and IC-21 buffer the incoming and outgoing data to and from the memory board to the data bus. NOR gates IC-24 B, C, and D enable the outgoing sections of the bi-directional transceivers IC-20 and IC-21 at the appropriate times. The incoming sections of the bi-directional transceivers are enabled at all times since the memory ICs have separate input/output lines.

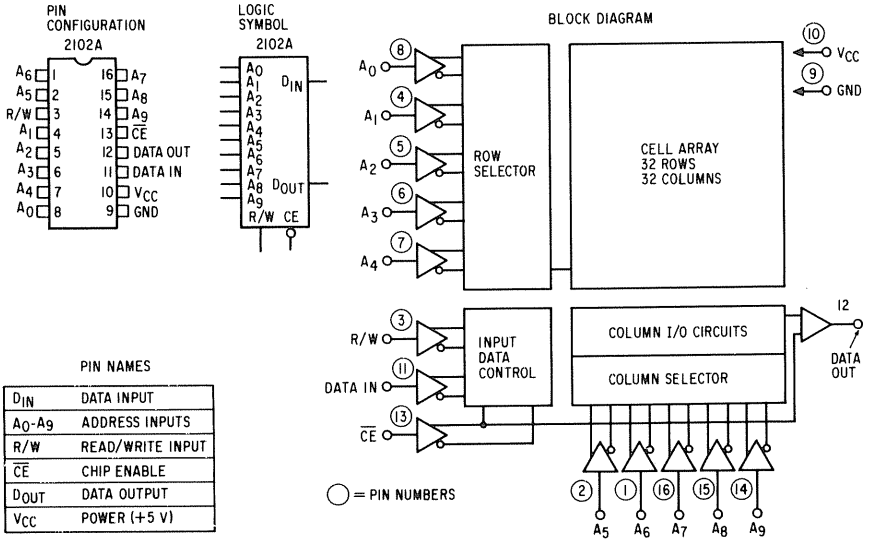


Fig. 5-2. The 1,024-bit RAM 2102 IC. (Courtesy Intel)

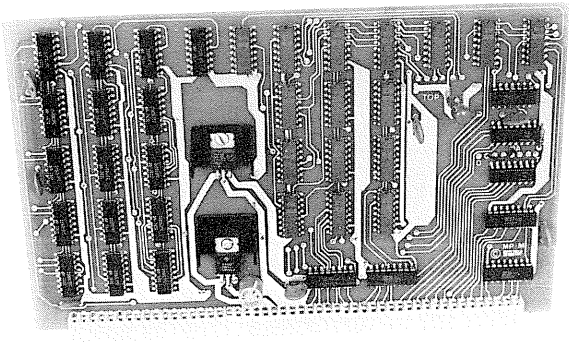


Fig. 5-3. SWTP MP 4K RAM board.

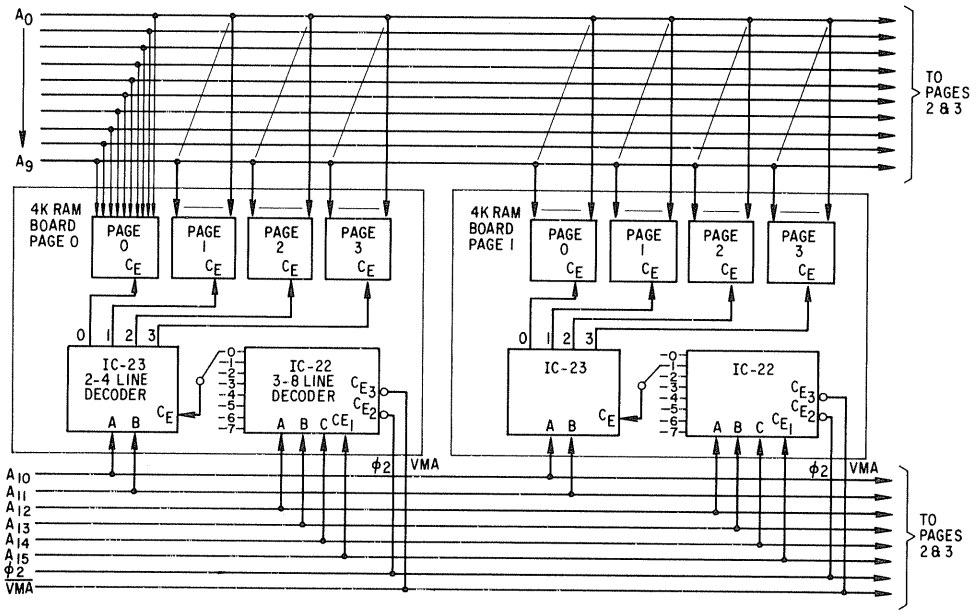


Fig. 5-4. Memory addressing by pages for the SWTP-6800 system.

Hex Memory Addressing

Most microcomputer systems employ a hex code for addressing (there are a few that use octal code). Table 5-1 can be used to convert from a decimal address to a hex address and vice versa; it shows the decimal equivalent for each hex number by place value. Most microcomputers can address up to 65,535 words using a 16-bit address word. Hence, four hex characters are used to represent the memory addresses.

The lowest address in memory would be:

Binary	*Hex*	*Decimal*
0000 0000 0000 0000	00 00	0

The highest address in memory would be:

Binary	*Hex*	*Decimal*
1111 1111 1111 1111	FF FF	65,535

Notice that the right-most hex digit indicates 1 of 16 possible memory locations. The next higher hex digit indicates 1 of 16 possible groups of 16 locations. The next higher hex digit indicates 1 of 16 possible groups of 256 locations. The highest hex digit indicates 1 of 16 possible groups of 4,096 locations.

Table 5-1. Hex-Decimal Memory Address Conversion Table

Hex	= *Dec*	*Hex*	= *Dec*	*Hex*	= *Dec*	*Hex*	= *Dec*
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

To find the decimal equivalent of a hex address, sum the decimal equivalents of each hex digit (from Table 5-1). For example, to find the decimal equivalent of $C31B_H$:

Hex

$$\begin{aligned} C_H &= 49{,}152_{10} \\ 3_H &= \quad 768_{10} \\ 1_H &= \quad 16_{10} \\ B_H &= \quad 11_{10} \\ &\quad 49{,}947_{10} \end{aligned}$$

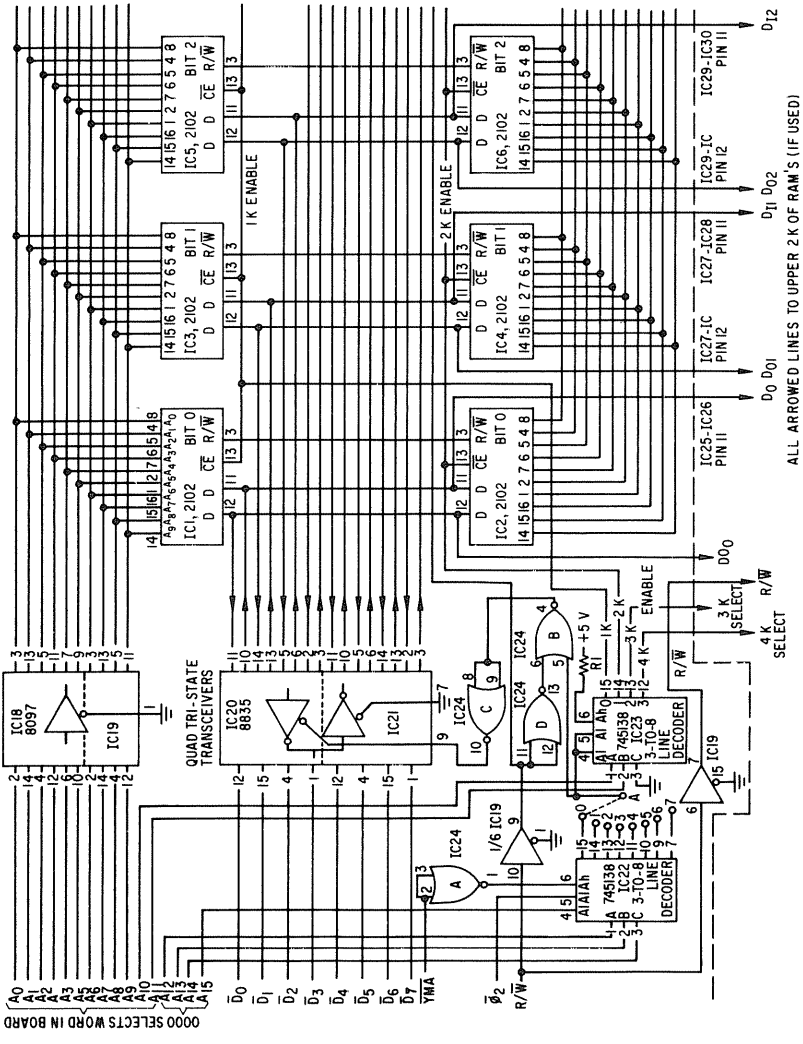
Hence $C31B_H = 49{,}947_{10}$.

To find the hex equivalent of a decimal address, subtract the decimal equivalent of highest hex value from the decimal number. Repeat the process for the decimal remainder. For example, to find the hex equivalent of $43{,}390_{10}$:

Hex

$$\begin{aligned} &43{,}390_{10} \\ -&40{,}960_{10} \leftrightarrow A \\ &2{,}430_{10} \\ -&2{,}304_{10} \leftrightarrow 9 \\ &126_{10} \\ -&112_{10} \leftrightarrow 7 \\ &14_{10} \leftrightarrow E \end{aligned}$$

Hence, $43{,}390_{10} = A97E_H$.


HEX TRI-STATE BUFFERS
IC18 8097
IC19
QUAD TRI-STATE TRANSCEIVERS
IC20 8835
IC21
IC1, 2102 BIT 0
IC2, 2102 BIT 0
IC3, 2102 BIT 1
IC4, 2102 BIT 1
IC5, 2102 BIT 2
IC6, 2102 BIT 2
1K ENABLE
2K ENABLE
IC24
1/6 IC19
74S138 IC22 3-TO-8 LINE DECODER
74S138 IC23 3-TO-8 LINE DECODER
A1A1Ah
3 K SELECT
4 K SELECT
R/W
ENABLE
+5 V
1K
2K
3K
4K
IC25-IC26 PIN 11
IC27-IC 28 PIN 12
IC27-IC28 PIN 11
IC29-IC30 PIN 12
IC29-IC30 PIN 11
ALL ARROWED LINES TO UPPER 2 K OF RAM'S (IF USED)
0000 SELECTS WORD IN BOARD
YMA

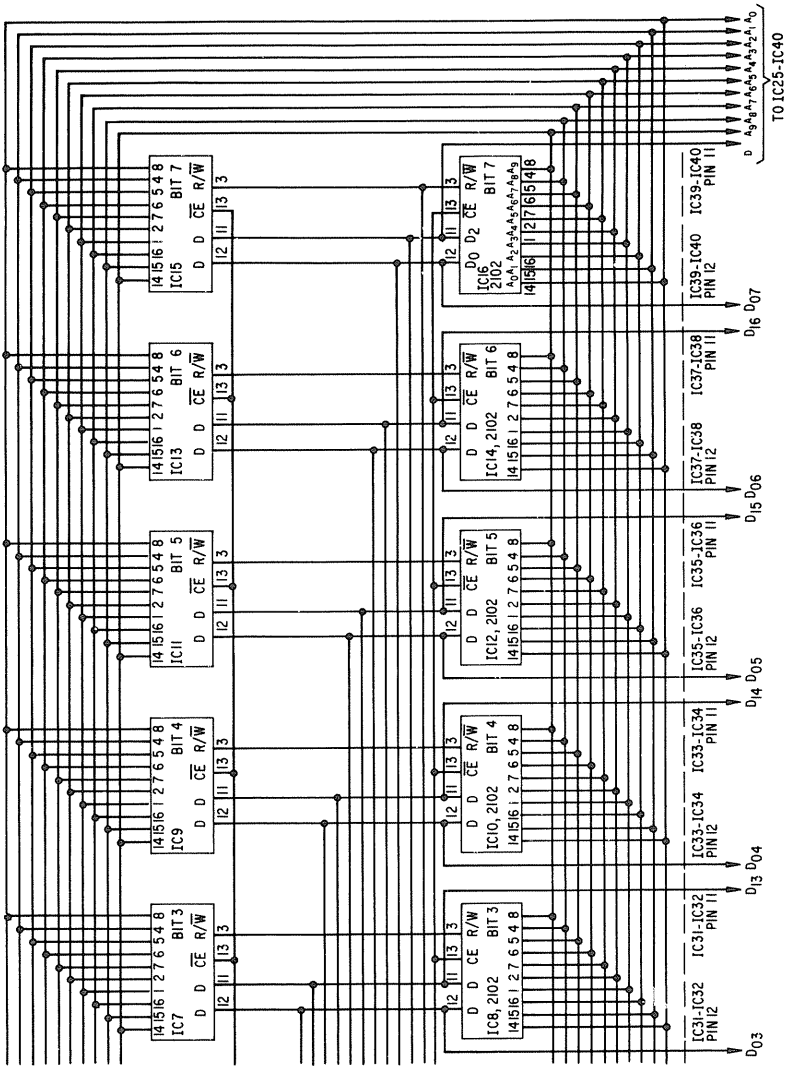


Fig. 5-5. Schematic of MP-M/MP-MX memory board (4K). (Courtesy SWTP)

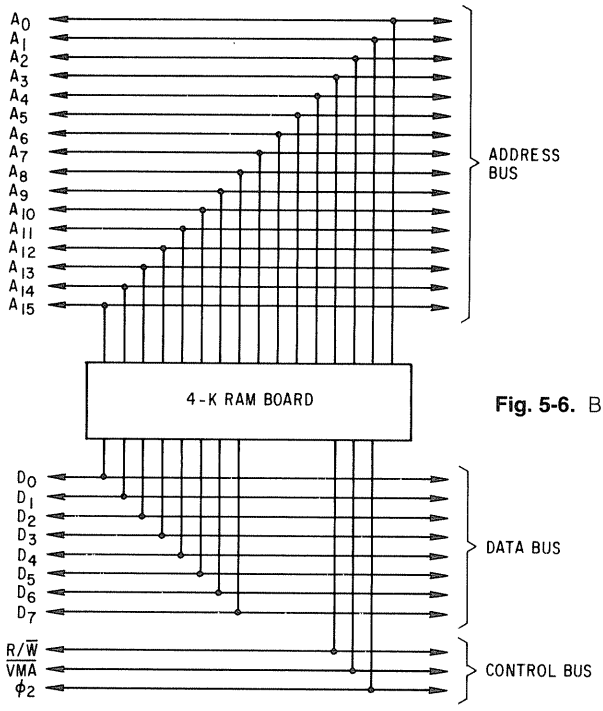


Fig. 5-6. Bus structure for SWTP CPU.

MEMORY TIMING

The signals on the address and data buses and control lines are not constant for the SWTP-4K RAM. There are 16 address lines, 8 data lines, and 3 control lines.

Figure 5-7 shows the signals that appear on the SWTP bus and control lines during a memory-read operation. Notice that the address word is placed on the address bus lines. In this example the memory address is $7E2F_H$—the 32,303rd decimal address in memory. At the same time, the control line logic levels are set up for the read operation ($R/\overline{W} = 1$, $\overline{VMA} = 0$ and $\phi 2 = 1$). Observe that the $\phi 2$ clock signal is much shorter than all the other signals. It is only when $\phi 2 = 1$ that the data from the RAM appears on the data bus. This is done to eliminate any critical timing which would otherwise occur. In other words, all the address and control signals have an adequate opportunity to settle down before the read or write operation occurs.

STATIC VERSUS DYNAMIC RAMS

The 2102 RAM IC cited previously is an example of a *static* type RAM IC. In other words, the storage element consists of a flip-flop type storage

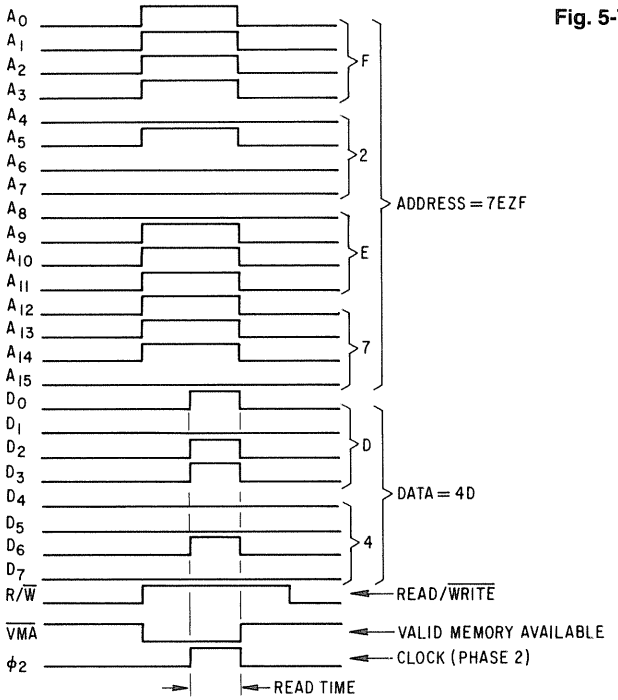


Fig. 5-7. Memory timing signals.

element. It is called static because it retains its 1 or 0 state, so long as power is applied.

Dynamic type RAM ICs store the logic state as an electrical charge between the gate and channel of a MOS-type transistor. This is a much simpler circuit requiring many fewer components than the static RAM. Therefore, it is possible to make much larger memories with fewer ICs using dynamic ICs. The disadvantage is that the charge leaks off and, hence, it is necessary to refresh the charge periodically (typically every 200 ms). In order to perform this task refresh circuitry is required. The newer dynamic RAM ICs include the refresh circuit on the chip and hence, as far as the user is concerned, there is little difference between the static and dynamic ICs. However, the older dynamic ICs require an external refresh circuit. These have been known to be critical and to sometimes create timing problems in the memory system.

ROMS AND PROMS

ROM (*read-only memory*) contains binary codes which cannot be altered. It is encoded during the manufacturing process. For example, a keyboard-to-ASCII code converter is actually a ROM. When a key is depressed it addresses one memory location which contains the ASCII coded word for the depressed

key. The ASCII code is then read out of the IC. The ROM is nonvolatile; it never loses the data placed in it.

A ROM is an array of selectively open and closed uni-directional circuits (diodes). A 16-bit ROM is depicted in Fig. 5-8. The address lines are decoded to select a pair of row and column select lines. Each diode has an open or closed contact. Hence, when a diode is addressed it passes current if there is a contact, or it does not pass current if there is no contact. These contacts are set selectively during the final phase of manufacture.

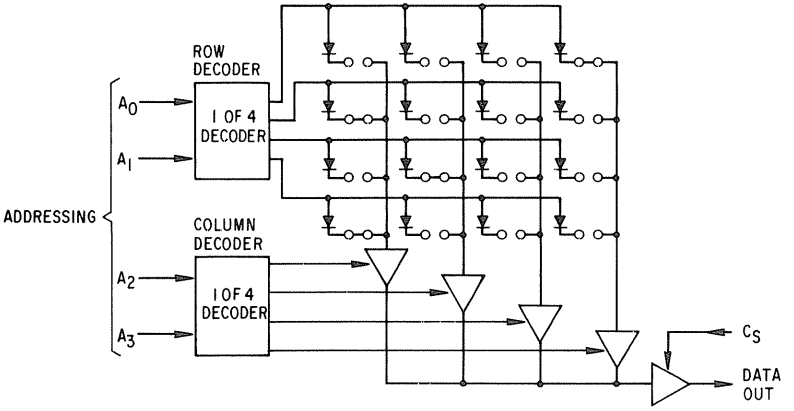


Fig. 5-8. A 16-bit ROM circuit.

PROMs (programmable ROMs) are made with a contact material that can be opened after manufacture by the user. The contacts are actually tiny fuses that are "blown" open selectively when programmed. The user can thus program the PROM. Once programmed, the PROM cannot be reprogrammed.
EPROMs (erasable PROMs) can be programmed by the user and then, if desired, the program can be erased and the EPROM reprogrammed. EPROMs (Fig.5-9) have a quartz lid to allow erasure by a high intensity ultraviolet light of the correct wavelength. An EPROM eraser unit is shown in Fig. 5-10.

ROMs, PROMs, and EPROMs are currently available in 1K, 2K, 4K, 8K, 16K, 32K, and 64K sizes. Many are packaged in 8-bit configurations. For example, the popular 2708 8K EPROM is configured as 1K words of 8 bits each. Hence, one 2708 will provide 1K of programmed memory.

A typical ROM memory is shown in Fig. 5-11. This unit, manufactured by Cromenco, is called the *Bytesaver*. It is a combination 8K EPROM memory and programmer. It can program either 2704 (512 × 8) or 2708 (1,024 × 8) EPROMs under control of the computer into which it is installed (uses S-100 bus).

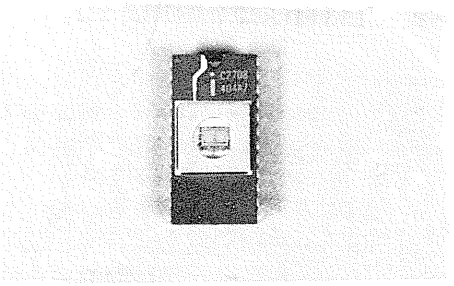


Fig. 5-9. Typical EPROM. Notice quartz lid.

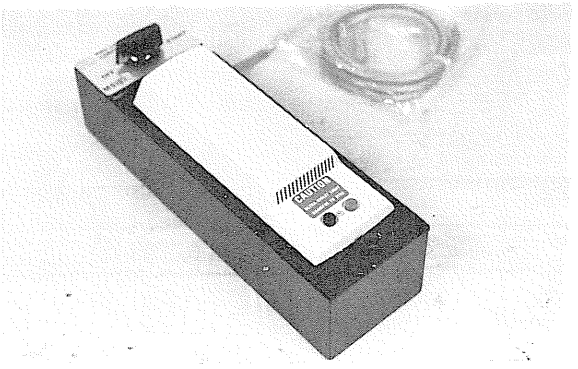


Fig. 5-10. An EPROM eraser unit.

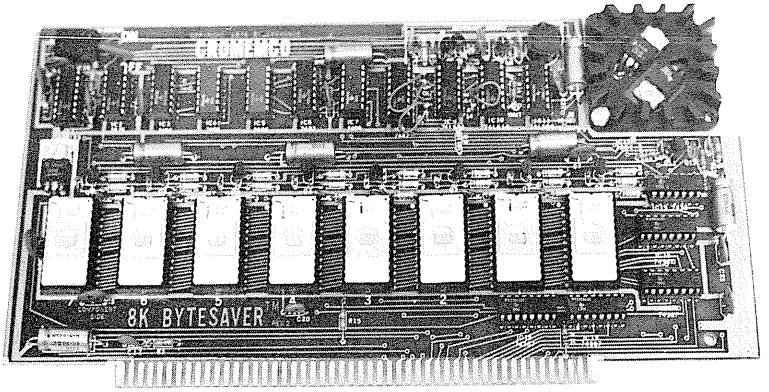


Fig. 5-11. The Cromenco ROM *Bytesaver* board.

MEMORY MAPPING

Memory space is allocated to RAM, ROM, and often I/O addressing. Note that microprocessors such as the 6800 and 6502 address I/O as if they were memory locations. These I/O addresses are called "ports."

An example of how memory space is allocated in a typical computer system is given in Fig. 5-12. This diagram shows where things are located in memory; hence it is called a *memory map*. The example shown is the memory map for the SWTP CPU. Notice that the memory is divided basically into 4K memory blocks. The first hex digit in the address word defines the block. Up to 64K of RAM space available. 32K is located from 0000 to 7FFF,8K is located from 8020 to 9FFF, 4K is located from A000 to A999, 16K is located from B000 to DFFF, and 4K from F000 to FFFF.

ROM is located in a 1K area from E000 to E3FF. This ROM contains two special programs which permit operation of the CPU with a terminal and does away with the need for a front panel (we discuss this later in detail). The program is referred to as a *debug* or *monitor* program. SWTP calls it *Mikbug/Minibug*.

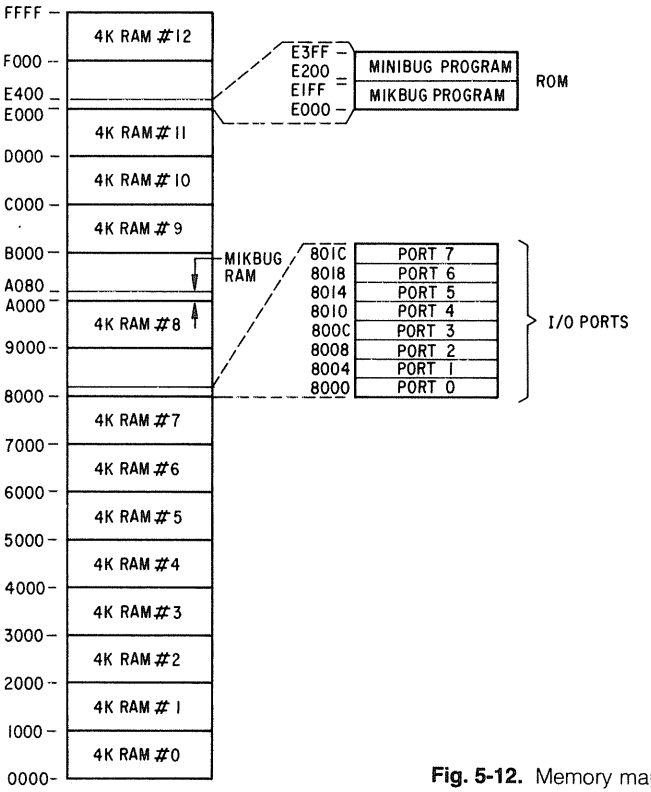


Fig. 5-12. Memory map of SWTP-6800 CPU.

Notice that the Mikbug program has a small, 128-byte RAM that it uses, which is located at A000 to A07F.

Lastly, the I/O ports are located at memory addresses 8000 to 801F. Each port uses four addresses, and, therefore, there are eight addressable I/O ports.

6.

Microprocessors

There are several dozen microcomputer systems sold in the personal computing area. With only minor exceptions, they are based on one of four MPUs (microprocessors)—the 8080/8085, Z-80, 6800, and 6502. In this chapter we examine these four microprocessors.

MICROPROCESSOR ARCHITECTURE

A typical microprocessor (Fig. 6-1) consists of three interconnected sections. They are registers, arithmetic logic unit (ALU), and control circuitry.

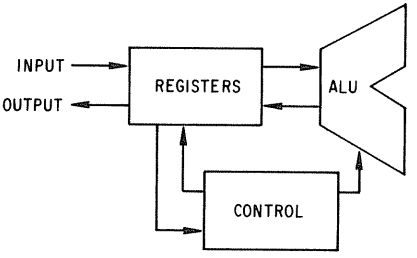


Fig. 6-1. The basic architecture of a microprocessor.

Registers

Registers are temporary storage circuits. All MPUs contain accumulator, program counter, stack pointer, and instruction registers. Some MPUs also contain additional general purpose registers.

The accumulator (the A-Register) stores one of the *operands* for the ALU. For example, an instruction might require the ALU to add the contents of some register to the accumulator and store the result in the accumulator (Fig. 6-2). The accumulator is both a source (operand) and destination register.

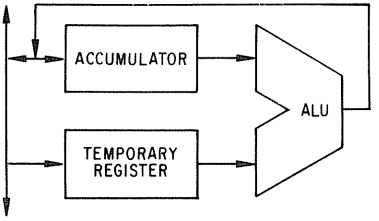


Fig. 6-2. Accumulator is both a source and destination register.

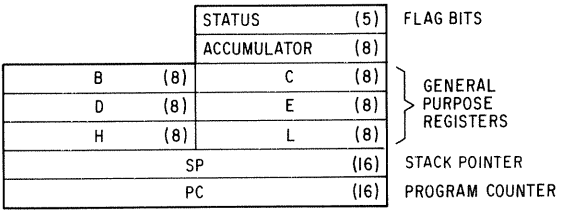


Fig. 6-3. The 8080/8085 registers.

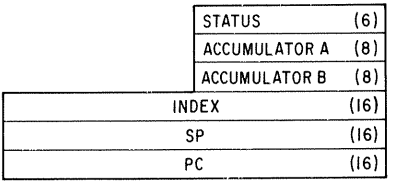


Fig. 6-4. The 6800 registers.

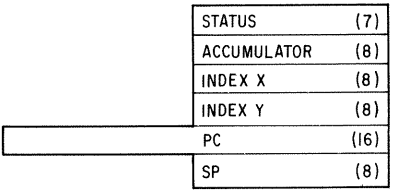


Fig. 6-5. The 6502 registers.

Often an MPU will include a number of additional general purpose registers to store operands or intermediate data. Other MPUs may use memory locations as registers. The 8080/8085 MPU has an accumulator and six general purpose registers (B, C, D, E, H, and L) (Fig. 6-3); the 6800 has two accumulator registers (Fig. 6-4); and the 6502 has one accumulator (Fig. 6-5) The 6800 and 6502 use memory locations as registers.

Program Counter and Stack Pointer Registers

The *PC* (*program counter register*) contains the *memory address* (*MA*) of the next program instruction. The MPU increments the PC every time it fetches

an instruction from memory. The program is placed in memory in sequential addresses. In this way the MPU sequentially fetches instructions from memory.

The exception to this procedure is a *jump* instruction. A jump instruction contains the MA of the next instruction to be fetched. When a jump instruction is executed, the MPU replaces the current MA with the MA in the jump instruction.

A special kind of jump occurs when a subroutine is called. A *subroutine* is an instruction sequence, located elsewhere in memory, which is *accessed* (used) a number of times. A *call* instruction is used to reference the subroutine. Most programs are nothing more than a sequence of calls to subroutines. The MPU handles the subroutine as follows:

1. Increments the PC and stores the PC word in a reserved memory area known as the *stack*. This saves the MA of the next instruction to be fetched after the subroutine is completed.
2. MPU loads the call MA into the PC. The next instruction will now be the first step in the subroutine.
3. The MPU now proceeds with the subroutine.
4. The last instruction in the subroutine must be a *Return* instruction. This causes the MPU to replace the MA from the stack into the PC and continue executing from that MA.

Subroutines can be *nested,* so that one subroutine calls another subroutine. The only requirement is that the stack area in memory must have enough space to store the return addresses.

The stack area in memory is referenced by the stack pointer (SP) register, which contains the MA of the most recent stack entry. The stack is also used when an MPU interrrupt occurs. In this case the contents of the accumulator and other general purpose registers are *pushed* onto the stack or *popped* off the stack via the MA stored in the SP.

Addressing Register

The MPU uses one or more registers to hold the MA that is to be accessed for data. For example, the 8080/8085 uses the H-L register pair as a 16-bit referenced MA register. Hence, an 8080/8085 instruction to move data from memory to the accumulator will move the data from the MA referenced by the H-L register pair. The H register contains the high part of the address and the L register the low part of the address.

The 6800 and 6502 use an *IR* (*index register*); the 6502 has two index registers. The second byte of an indexed addressing instruction is added to the contents of the IR to give the MA reference.

Status Register

The status register, often called the *flags* or *flag bits,* operates in conjunction with the ALU and specifies conditions that occur as arithmetic and logic

operations are performed. The typical flags (flag bits or status bits) are: carry, zero, sign, parity, and auxiliary carry. Program jumps are made conditionally dependent on one or more status flags. For example, a jump to a subroutine may be made if the carry bit is set by an addition operation.

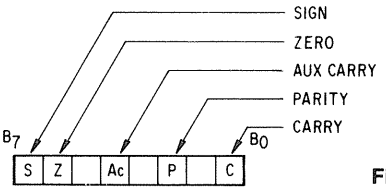


Fig. 6-6. The 8080/8085 status flag register.

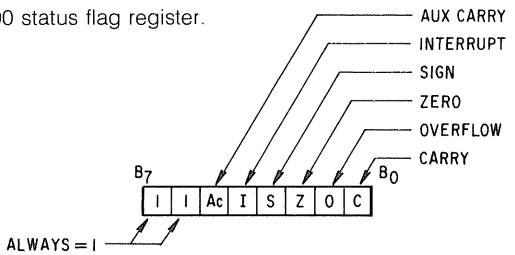
The 8080/8085 status flag register is shown in Fig. 6-6. Notice that it is an 8-bit register with 3 bits unused. The status bits operate as follows:

Zero: Set (1) if result of ALU operation is zero, otherwise it is reset (0).
Sign: Set if in the result of ALU operation the most significant bit = 1, otherwise reset. The most significant bit is used, in signed binary arithmetic, to indicate a negative number.
Parity: Set if result of ALU operation has even number of 1s in word, otherwise reset.
Carry: Set if result of addition or subtraction operation results in a carry or borrow out of the highest order bit, otherwise reset.
Auxiliary Carry: Set if a carry out results from B_3 into $_4$, otherwise reset. This is affected by the operations of addition, subtraction, increments, decrements, comparisons, and logic operations. This is used most for decimal adjusting operations.

The 6800 status flag register shown in Fig. 6-7 is very similar to the 8080/8085. However, there are two differences:

Overflow: Set when 2's complement arithmetic operation results in a carry, otherwise reset.
Interrupt: Set when it is desired that the MPU not respond to an interrupt request.

Fig. 6-7. The 6800 status flag register.

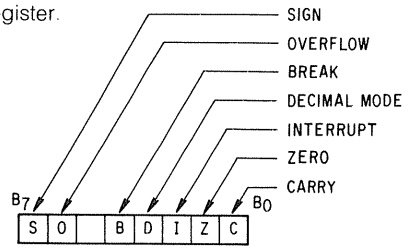


The 6502 is different from the 6800 and 8080/8085 in two status bits:

Decimal Mode: When this flag bit is set the MPU performs BCD arithmetic operations.
Break: Set when a software interrupt is executed.

Also note that the overflow status flag can also be used for control. The 6502 status flag register is shown in Fig. 6-8.

Fig. 6-8. The 6502 status flag register.



The Arithmetic Logic Unit (ALU)

The ALU is the section of the MPU which performs arithmetic and logic operations. It contains a parallel adder circuit which combines the contents of the accumulator and another register or memory location. The ALU can add, subtract, do Boolean logic operations (AND, OR, X-OR, NOT, compares), and shift operations. Figure 6-2 shows the relationship between the ALU and the registers.

The Control Section

The MPU performs operations identified by an 8-bit instruction word known as an *operation code* (*Op code*). Using 8 bits, it is possible to have as many as 256 distinct Op Codes.

The MPU fetches the Op Code from memory. The MA of the Op Code is in the PC. The Op Code is placed in the instruction register and decoded by the instruction decoder to control the activities of the registers, ALU, and peripheral ICs. The block diagram of the control section of the 8080/8085 is shown in Fig. 6-9.

The control section can respond to an *Interrupt* request which causes the control circuitry to temporarily interrupt the main program's execution, jump to a special routine to service the interrupting device, then return automatically to the main program. The control section can also respond to a *Wait* request from memory or a peripheral which causes the MPU to idle a specific period of time until the memory or peripheral is ready.

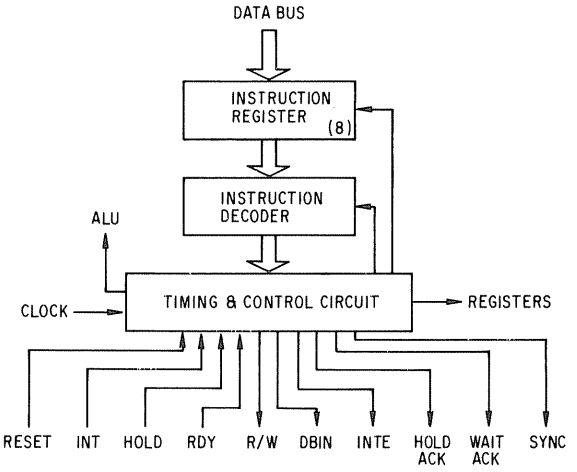


Fig. 6-9. 8080 control section.

The reset signal allows an external device to set the PC to zero and clears the registers.

COMPUTER OPERATIONS

Timing

An MPU goes through a cycle of events. It fetches an instruction, does the indicated operation, then fetches the next instruction, etc. These sequenced operations require synchronization of all circuits involved. This is accomplished by a free-running oscillator clock. All operations are synchronized to this clock and timing signals.

A complete fetch and execution of a single instruction is called an *instruction cycle*. It in turn is made up of *states* of clearly defined activity. Each state requires one or more clock periods.

Instruction Fetch

The MPU's first state(s) of the instruction cycle is an instruction fetch. The contents of the PC are placed on the address bus and the R/$\overline{W}$ line is set to read from memory. The memory responds by placing the contents of the selected MA on the data bus.

The MPU then inputs the data word on data bus into the instruction register. If the instruction is more than 1 byte long, the fetch state(s) is repeated with the PC incremented and the instruction decoded. The MPU then executes the operation in the remaining states of the instruction cycle.

Memory Read/Write

A *memory read* operation is similar to the instruction fetch with the exception that the data word transferred from the selected MA to the data bus is inputted to the accumulator. A *memory write* operation is the same as a memory read operation, except that the data word in the accumulator is placed on the data bus and then transferred to the selected MA.

Wait States

Some memories cannot supply data to the data bus during the short read time of the MPU. The speed with which the memory can output data is called the memory's *access time*. If a memory's access time is greater than the MPU read time, then the memory circuit can be designed to place a request signal at the MPU's *ready input*, causing the MPU to idle temporarily. Then the MPU frees the ready line and the instruction cycle continues.

INPUT/OUTPUT (I/O)

An input or output operation is similar to a memory read or write except that a peripheral device is addressed. The 8080/8085 can address up to 256 separate input and 256 separate output ports. The 8080/8085 issues separate I/O control signals. The 6800 and 6502 treat input/output in the exact same manner as memory read/write. Hence, the MPU does not distinguish between the two. The programmer writing the control program must be aware that certain memory addresses are really I/O addresses.

INTERRUPTS

In most applications the MPU is idle, waiting for an event to occur. Therefore, the MPU can handle more than one processing task in a given space of time. This is accomplished via an interrupt line(s). When an I/O device signals an interrupt, the MPU acknowledges the interrupt, suspends program execution, and branches to a routine which services the interrupt. When the interrupt service routine is completed the MPU returns to the previously interrupted program.

It is also possible to set up levels of interrupt where more than one device can initiate an interrupt. Such a system is called a *priority interrupt*.

DIRECT MEMORY ACCESS (DMA)

In normal operation, all data transfers to and from memory must pass through the MPU and the operation is controlled by the MPU. Memory transfers

may be performed at a much faster rate by using a *DMA* technique. This is accomplished via the *hold* line of the MPU. When the MPU hold line is enabled, the MPU suspends control over memory read/write operations and control is turned over to a special DMA controller circuit that performs memory read/write operations more quickly. The basic DMA configuration is shown in Fig. 6-10.

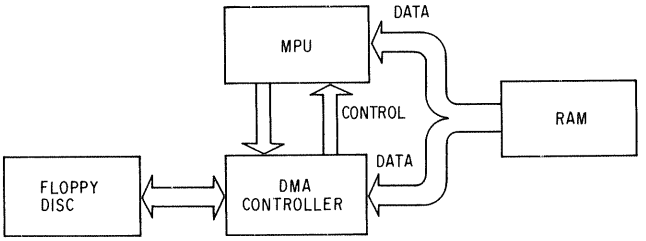


Fig. 6-10. DMA basic configuration.

THE 8080/8085

Intel was the first manufacturer to develop a microprocessor (the 8008 introduced in 1971). The 8080 MPU, introduced in 1973, became the most widely adopted MPU in the industry. In 1977 Intel introduced the 8085, an enhanced version of the 8080 (called the 8080A). In addition, Zilog manufactures the Z-80 which is also an 8080 enhancement (to be discussed later).

The 8080 and 8085 are similar yet they have the following differences. The 8085 has an on-chip clock circuit, operates from one +5 V source, is TTL compatible, has 4 vectored interrupts, and a serial-in/serial-out port. The 8080 lacks these features and requires separate power sources (+5 V, −5 V, +12 V), a separate clock circuit, system controller, and data bus buffers. Therefore, it is evident that the 8085 hardware is simpler than the 8080. All software for the 8080 will operate on the 8085.

The functional block diagram of the 8085 is shown in Fig. 6-11. Notice that this block diagram is very similar to the 8080, shown in Figs. 6-2, 6-3, and 6-9.

One additional difference between the 8080 and 8085 should be pointed out. At the begining of an instruction cycle the 8080 outputs a pulse on the sync line to indicate the beginning of the cycle. Also, the status control signals are outputted on the eight data lines. Hence, it is necessary to latch these signals and decode them to develop the CPU control signals. This is most often accomplished by a peripheral IC, the 8228. The 8228 also provides bi-directional data bus drivers. A typical 8080 system using the 8228 and also the 8224 clock generator-driver IC is shown in Fig. 6-12. These three ICs now make up a complete 8080 MPU system.

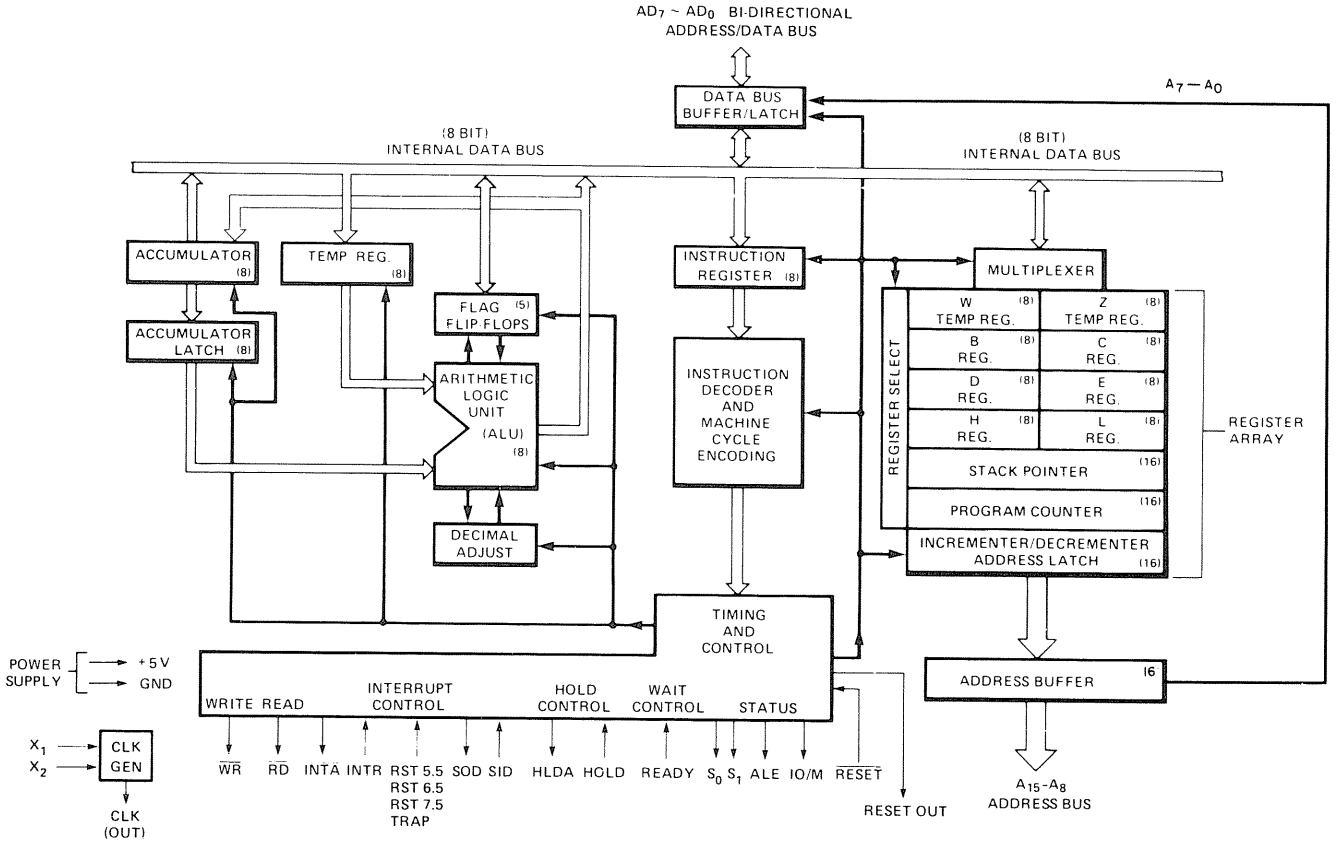


Fig. 6-11. 8085 MPU functional block diagram.

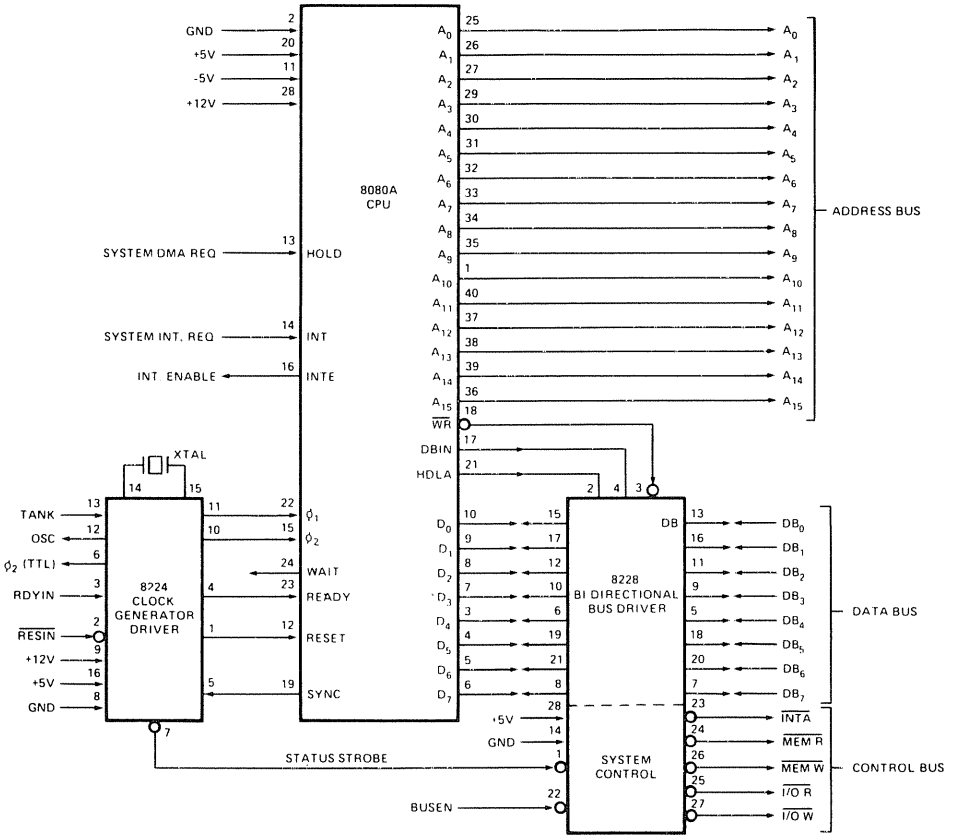


Fig. 6-12. 8080 typical CPU circuit.

The 8085 contains the clock, data bus drivers, and CPU control circuit and, hence, does not require any support ICs. However, it does multiplex the lower MA bits with the data bits on the eight data lines. Hence, it is necessary to provide a latch (eight D type flip-flops) to latch (temporarily store) the lower MA bits. Intel provides several memory-I/O ICs with this feature. Figure 6-13 shows a typical system using these ICs.

The system has one I/O port for a terminal, 5 parallel I/O ports, 2K bytes of ROM, and 256 bytes of RAM. The peripheral ICs are made by Intel specifically to work with the 8085. The 8155 IC contains 256 bytes of RAM, three parallel I/O ports, and a special timer circuit. It also contains an address latch to catch the MA word from the multiplexed address/data bus. The IO/M control line

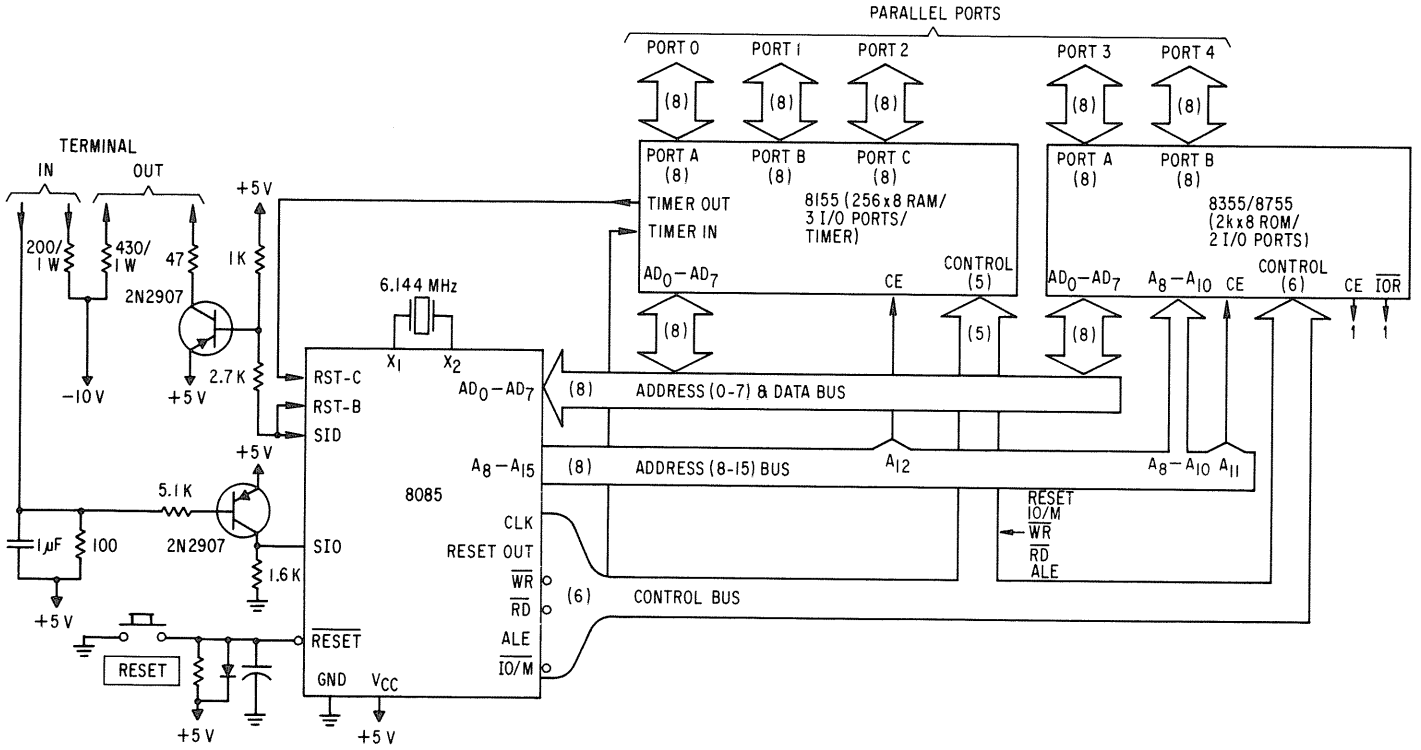


Fig. 6-13. Typical small 8085 CPU system.

selects either I/O or memory. The WR and RD lines determine whether data moves to or from the MPU to memory or I/O. ALE enables the address latch to catch the high order address word from the data bus.

The timer, under program control, divides the clock signal. It is used here to set the baud rate (to be discussed later) for the terminal. The MPU clock uses a 6.144 MHz crystal. The MPU clock out signal is one-half this, 3.072 MHz. The timer then divides this signal down to the desired rate and feeds it back to the MPU (RST-C) to operate the serial data input and output circuits (SDI and SDO) going to the terminal. Transistor interface circuits provide the terminal in and out circuits with a current loop signaling circuit.

The 8355/8755 (EPROM/ROM) provides the program to operate the terminal and the operating system program (to be discussed later). This does away with the need for a front panel on the CPU. All that is required is a *reset* switch which sets the PC to the first address in the operating system program and effectively restarts the CPU.

THE 6800

The 6800 was introduced by Motorola in 1973 at the same time that Intel introduced the 8080. The 6800 is a much simpler device than the 8080. It has simpler timing in which an instruction cycle takes only one clock period. The 6800 includes memory and I/O in one address space. Hence, all I/O is addressed as memory locations. The 6800 does not multiplex data and control signals on address lines; therefore the CPU control signals are simpler. Moreover, the 6800 operates from a single +5 V supply.

A typical 6800 CPU system is shown in Figs. 6-14, 6-16, and 6-17. This is the circuit for the popular SWTP-6800 CPU manufactured by Southwest Technical Products. Figure 6-14 shows the 2-phase clock circuit. IC-4 is an oscillator-counter IC. It functions as a crystal-controlled oscillator and counter to develop lower frequency clock signals used in the interface circuits, to be discussed later. The 2-phase clock signals are developed by inverting the clock signal. Two power amplifier circuits provide the drive to the MPU. The clock signals are shown in Fig. 6-15. Note that it takes two to eight machine cycles to execute instructions.

The CPU (Fig. 6-16) drives a bi-directional data bus thru bi-directional bus drivers. The bus has the following control signals.

Halt: When =0, CPU ceases execution and floats MPU address and data lines.

R/$\overline{W}$: When =1, MPU reads data from data bus; when =0, MPU is outputting data to data bus.

$\overline{VMA}$: (Valid memory address) =1 when a valid address is placed on address bus.

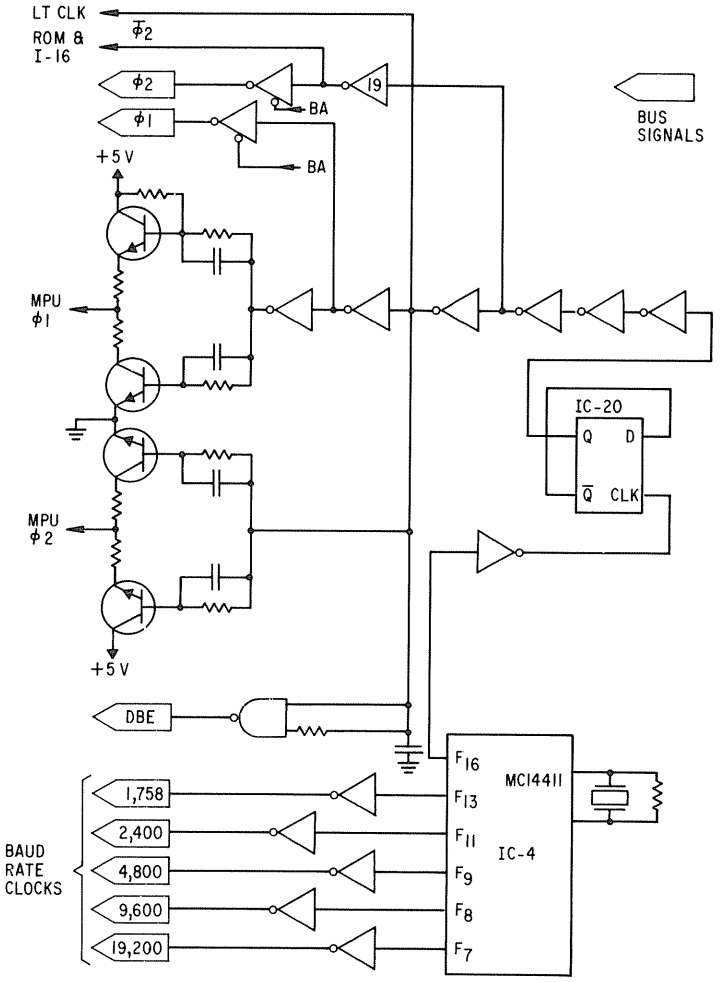


Fig. 6-14. SWTP-6800 CPU clock circuit.

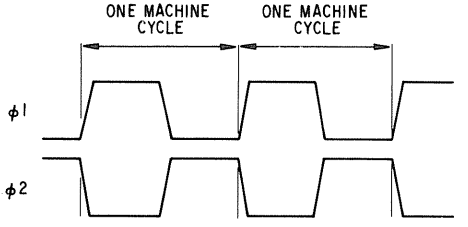


Fig. 6-15. 6800 two-phase clock signal.

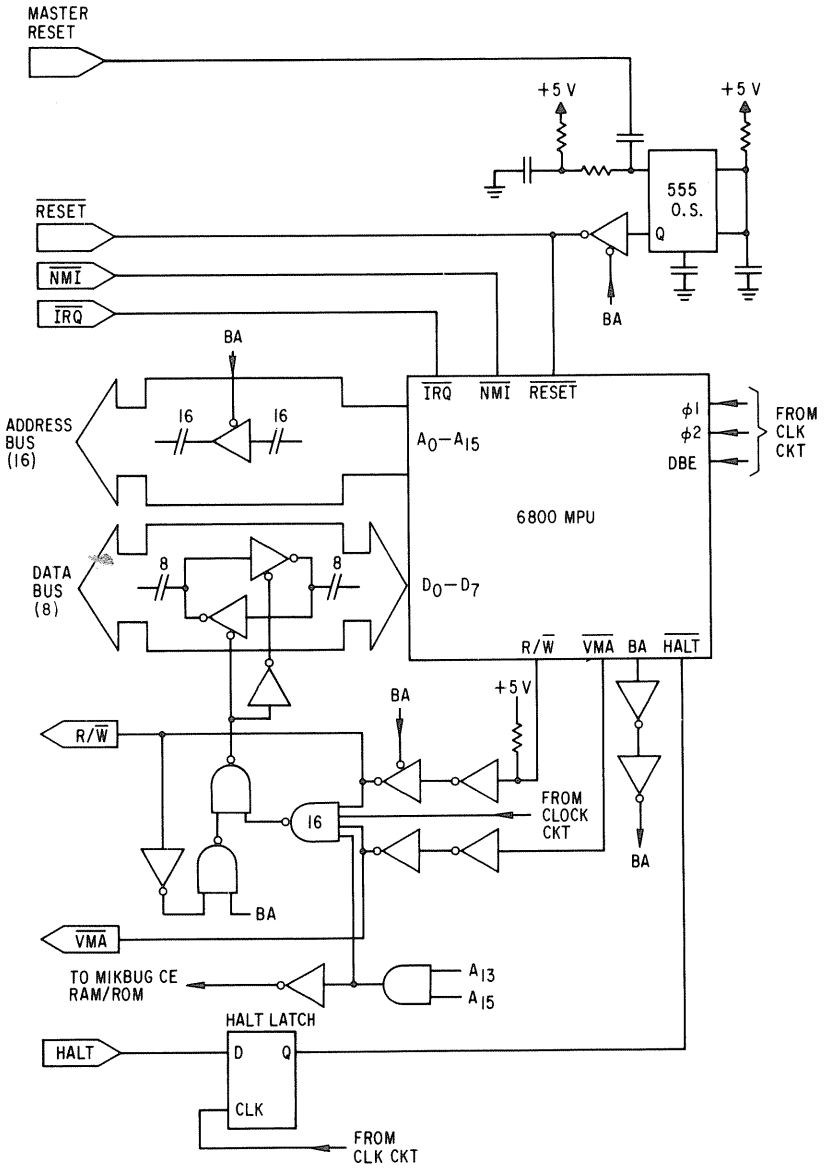


Fig. 6-16. SWTP-6800 CPU circuit.

IRQ: (Interrupt request) when =0, and MPU is not in halt state and interrupts have been enabled, will initiate an interrupt.
NMI: (Nonmaskable interrupt) same as IRQ but cannot be inhibited.
RESET: Resets MPU.

Notice that many buffering circuits are used since the 6800 can drive only one TTL load.

The memory circuits shown in Fig. 6-17 are on the same printed circuit board as the clock and MPU circuits. The ROM (1K bytes) contains a monitor-debug program (to be discussed later) and, hence, the only operating control necessary is a *reset* switch. The program allows a terminal to operate the CPU directly. The ROM program called Mikbug is in ROM IC No. 6830L7 and resides at memory location $E000_H$ to $E1FF_H$ (512 bytes). The Mikbug (Fig. 5-12) program utilizes a small RAM (128 bytes) area for location of the stack. This is provided by the 6810L-1 RAM also located on the CPU printed circuit board. This RAM resides at MA $A000_H$ to $A07F_H$ (Fig. 5-12).

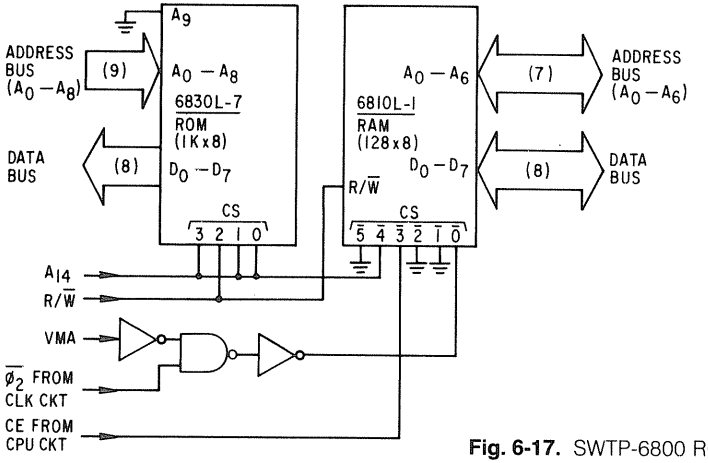


Fig. 6-17. SWTP-6800 ROM/RAM circuit.

THE 6502

The 6502 is just as popular as the 6800 in personal computing system use. Its popularity results from the fact that it is even easier to implement a 6502 hardware system than the 6800 (which is much simpler than the 8080). Also the 6502 offers some enhanced features over the 6800. They are: simpler control bus (only 9 control lines); on-chip clock (external crystal or R-C circuit required); operation at higher speeds (clock frequencies up to 4MHz); capability of performing BCD arithmetic directly. The functional block diagram of the 6502 is shown in Fig. 6-18.

The 6502 is the most widely used MPU among those who homebrew their own CPU systems. This is because the bus system and interfacing (to be discussed later) are so simple. Furthermore, MOS Technology, the manufacturer of the 6502, provides excellent, simple to use, and very powerful supporting ICs. For example, there is the 6530 IC which provides 1K bytes ROM, 64 bytes RAM, a

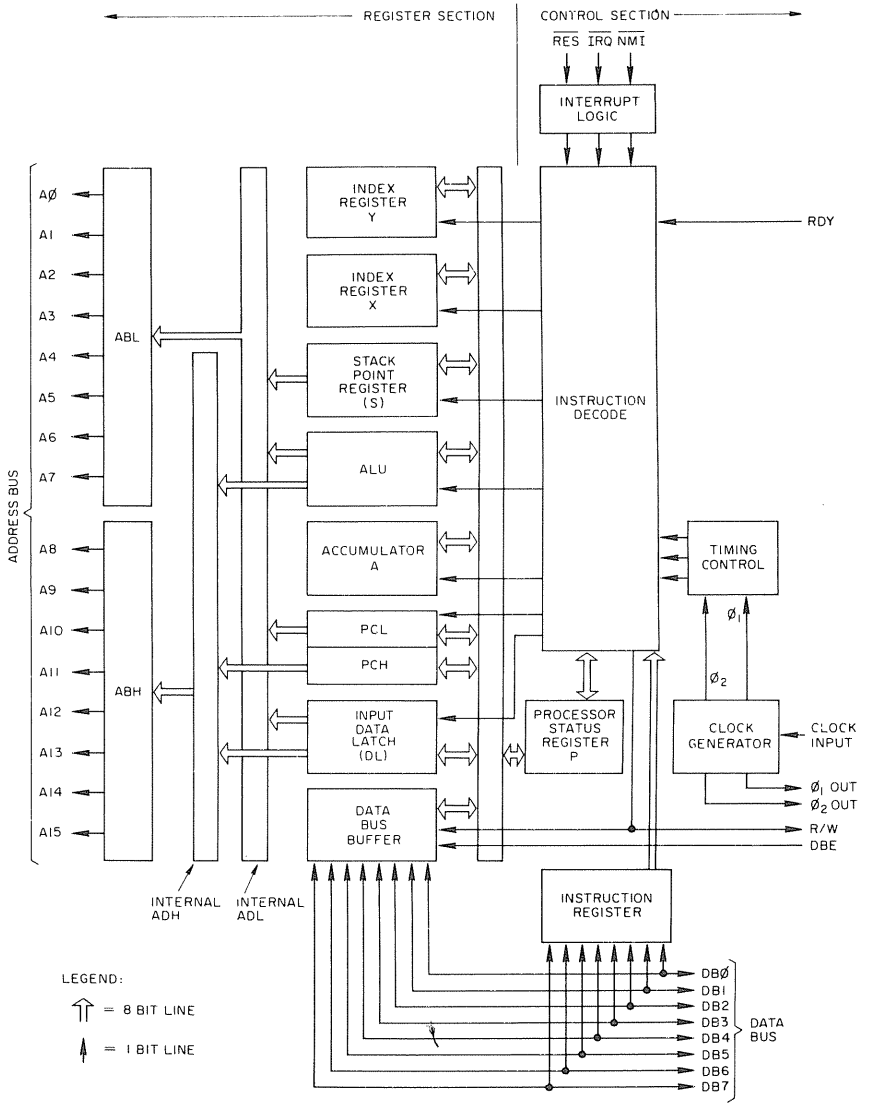


Fig. 6-18. 6502 functional block diagram. (Courtesy MOS Technology)

programmable timer, and two parallel I/O ports. The 6530 is available pre-programmed with a very good debug-monitor called *TIM* for *terminal interface monitor* (IC No. 6530-004). This same IC, with a different ROM program is employed in the popular KIM-1 CPU system to control a keyboard, 6-digit LED display, and cassette tape interface.

The TIM system is very popular among homebrewers. The reason why is seen in the example of a typical TIM CPU shown in Figs. 6-19 and 6-20. Using only 10 ICs, it provides a complete CPU system with 1K bytes ROM, 576 bytes RAM (directly expandable to 5K bytes RAM), one parallel I/O port, and one terminal interface circuit.

Four 2111 RAM ICs (256 × 4 bits each) provide 512 bytes of RAM which together with the 64 bytes in the 6530 provide a total of 576 bytes of RAM. The 7442 IC decodes the addressing to select either of two pairs of 2111 RAM ICs. Each pair provides 256 bytes of RAM. The control lines are very simple; only 2 are used, R/W and $\emptyset 2$. The 2111 RAM is addressed at 0000_H to $01FF_H$. Note that the 2111 ICs contain their own bi-directional bus drivers making interfacing to the data bus extremely simple.

The 6530-004 IC containing 1K bytes ROM, 64 bytes RAM, and I/O is addressed as follows: 6300_H to $630F_H$ for I/O, 7000_H to $73FF_H$ for ROM, and $7FA0_H$ to $7FFF_H$ for RAM. The 6530 actually has two parallel ports. Port A is used as a standard parallel port with 2 lines of Port B (PB2 and PB3) providing handshaking signals (see Chapter 7). Another 2 Port B lines are used to interface a serially operated terminal (PB1 and PB0). PB4 is fed back to control initial addressing of the ROM (CS1) so the system can be initialized by the terminal.

A manual reset switch, shown at the lower left of Fig. 6-20, is debounced by an R-S flip-flop circuit. It is used to reset the MPU and TIM ICs. The MPU clock is crystal controlled by the external crystal (1 MHz) connected between pins 37 and 39.

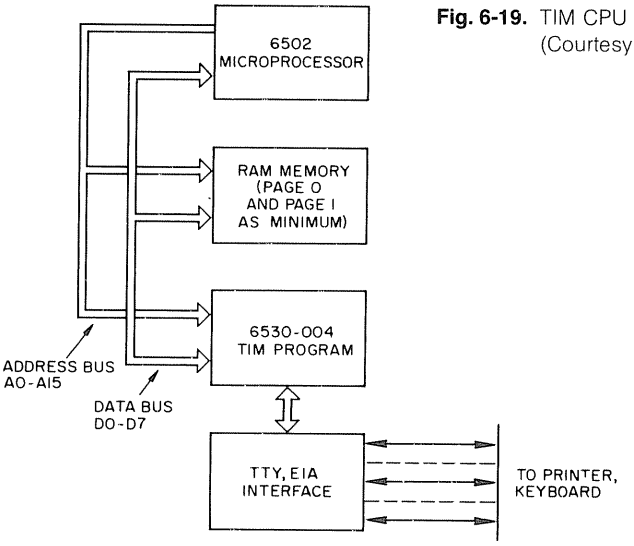


Fig. 6-19. TIM CPU system—block diagram. (Courtesy MOS Technology)

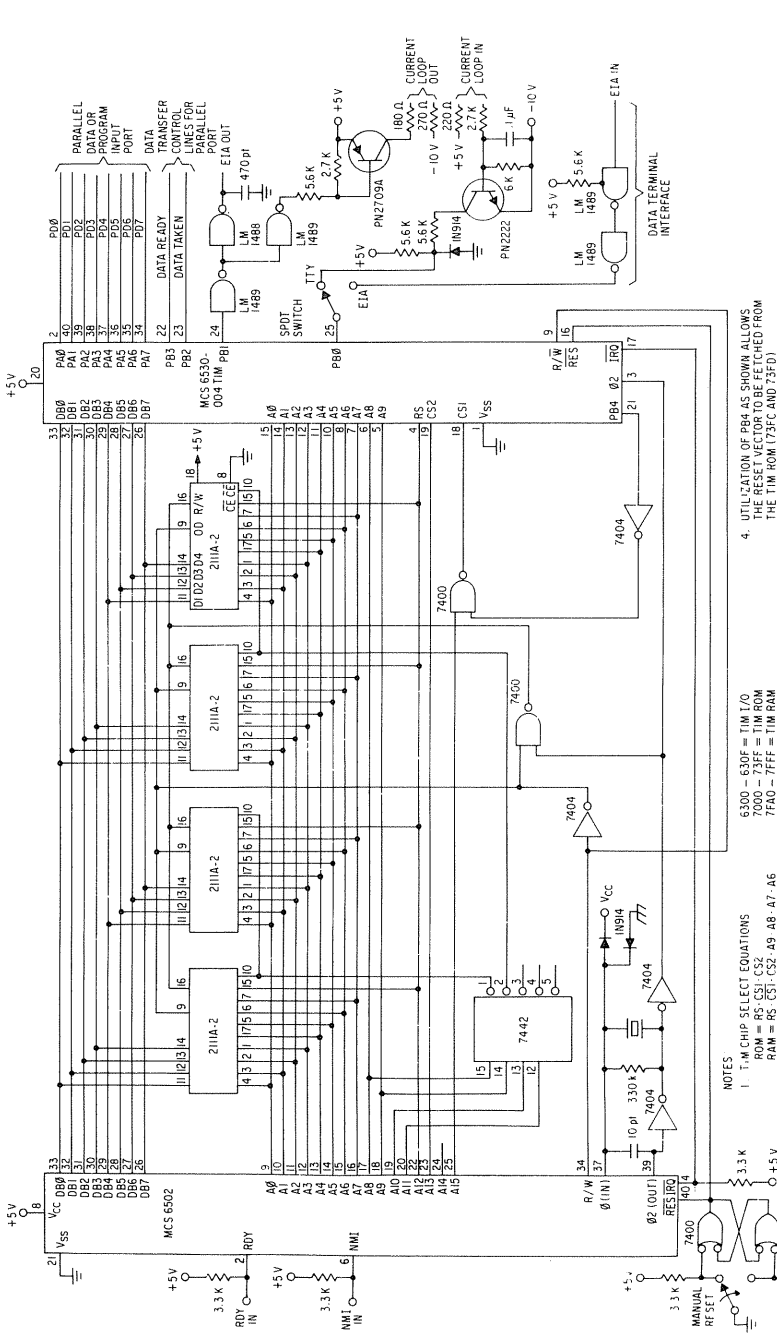


Fig. 6-20. TIM CPU system—schematic. (Courtesy MOS Technology)

As this book is being written, MOS Technology has announced the planned introduction of the 6509 IC which will combine the 6502 and 6530. This would reduce the IC count for the TIM system to only eight ICs. How simple can you get?

THE Z-80

Another MPU often used in personal computer systems is the Z-80. Introduced by Zilog in 1976, it is an enhanced version of the very popular Intel 8080 MPU. The Z-80 offers the following enhancements:

1. Clock and control circuitry on MPU chip. This does away with the 8224 clock and 8228 system controller ICs required with the 8080.
2. Operates from a single +5 V power supply.
3. Has additional interrupt input and clocking to refresh dynamic memories.
4. Can operate at higher clock rate (up to 4 MHz compared to 2 MHz for 8080A).
5. It has more than twice as many registers as the 8080. Figure 6-21 shows a comparison between the Z-80 and 8080 registers.
6. Has two index registers for indexed addressing (used in 6800 and 6502)
7. Has a greatly expanded instruction set.

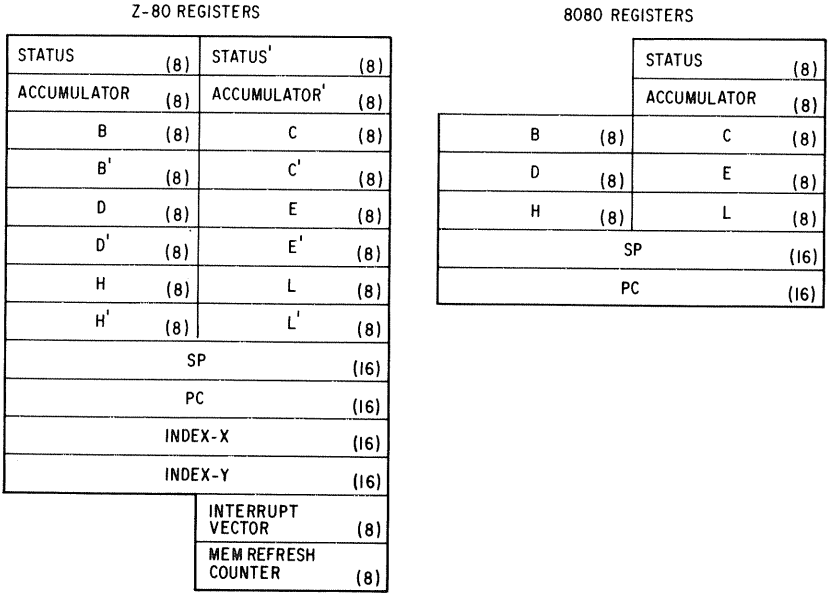


Fig. 6-21. Comparison of Z-80 and 8080 registers.

Recommended Further Reading

1. Adam Osborne, *An Introduction to Microcomputers,* Vol. II., Adam Osborne & Assoc., Inc., Berkeley, Calif., 1976.
2. *Intel 8080 Microcomputer Systems User's Manual,* Intel Corp., Santa Clara, Calif., 1975.
3. *6502 Hardware Manual,* MOS Technology, Norristown, Pa., 1976.
4. *6800 User's Manual,* Motorola, Phoenix, Arizona, 1975.

7.

The Ins and Outs of Interfacing

The interface section of a CPU is the circuitry necessary to select the peripheral devices outside the CPU and make the signals between the CPU and the peripheral compatible. The basic scheme is shown in Fig. 7-1. This interfac-

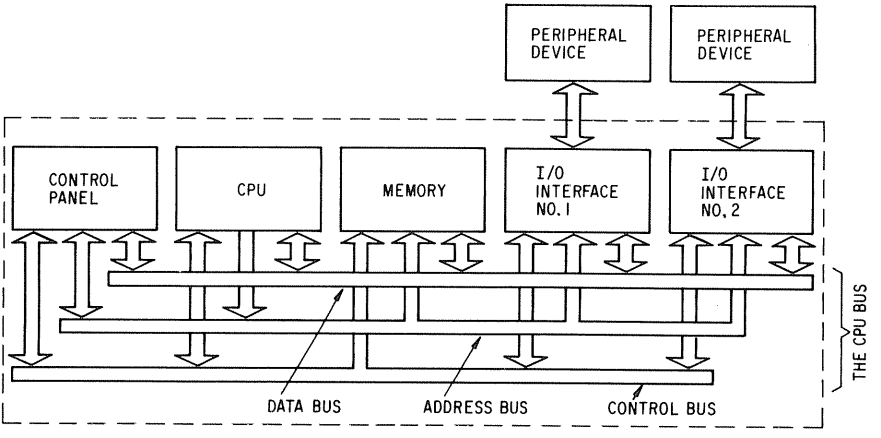


Fig. 7-1. CPU and peripherals.

ing task may be very complex. We may handle the signals between units one bit at a time—*serial I/O*, or transfer all bits at the same time—*parallel I/O*. Each separate grouping of input or output lines is called a *port*, and each has an address just like memory cells. We cover interfacing basics in this chapter with some additional points to be made in Chapters 8 and 9.

THE CPU BUS (S-100)

The CPU bus comprises the data transfer, addressing, and control lines. There is no standard CPU bus. However, the bus structure used on the Altair-8800 (manufactured by MITS) has also been adopted by several other manufacturers and is the most widely used in personal computers. The bus is also known as the *S-100 bus*. There are presently at least 15 manufacturers making CPU and other circuits on *printed circuit boards* (*PCBs*) that plug directly into the S-100 bus. Therefore, the S-100 bus may be considered as the de facto standard.

The Altair 8800 CPU uses the Intel 8080 MPU and, hence, most of the bus definitions come from the 8080 data sheet. Despite this, there are at least three Z-80, one 6800, and one 6502 CPU PCBs for the S-100 bus. It should be noted, however, that the Z-80, 6800, and 6502 have fewer control lines, more simple and straightforward timing sequences, and could easily operate with less than the number of bus lines of the S-100 bus. These manufacturers do utilize the S-100 bus so that their CPUs can be easily interfaced to the more than 100 S-100 compatible peripheral PCBs containing memory, I/O interface, etc.

The S-100 bus uses a 100-pin PCB edge connector with the pins allocated as follows:

TSL = Tri-State Logic
N.C. = No Connection

1. +8 V unregulated
2. +16 V unregulated
3. XRDY-1; ANDed with PRDY and goes to 8080 RDY; also used to cause MPU to enter a wait state.
4. VI-0
5. VI-1
6. VI-2
7. VI-3
8. VI-4
9. VI-5
10. VI-6
11. VI-7

Vectored Interrupt Request

12. XRDY-2; see pin # 3
13. N.C.
14. N.C.
15. N.C.
16. N.C.
17. N.C.
18. $\overline{\text{STA DSB}}$ Status buffer disable (TSL)
19. $\overline{\text{C/C DSB}}$ Command/control buffer disable (TSL)
20. UNPROT Input top memory unprotect circuit on selected RAM PCB
21. $\overline{\text{SS;}}$ Indicates machine is in single step mode
22. $\overline{\text{ADD DSB}}$ Address buffer disable (TSL)
23. $\overline{\text{DO DSB}}$ Phase out (from CPU) buffer disable (TSL)
24. $\phi 2$ Phase two clock, TTL levels
25. $\phi 1$ Phase one clock, TTL levels

26.	PHILDA	Hold acknowledge, buffered 8080 output
27.	PWAIT	Wait acknowledge, buffered 8080 output
28.	PINTE	Interrupt enable, buffered 8080 output
29.	A5	Buffered address lines
30.	A4	
31.	A3	
32.	A15	
33.	A12	
34.	A9	
35.	DO-1	Buffered data out lines
36.	DO-0	
37.	A10	Buffered address line
38.	DO-4	Buffered data out lines
39.	DO-5	
40.	DO-6	Buffered data out line
41.	DI-2	Data Input lines
42.	DI-3	
43.	DI-7	
44.	SM1	Latched 8080 MI status (MPU in fetch cycle)
45.	SOUT	Latched 8080 OUT status (MPU outputting in I/O port)
46.	SINP	Latched 8080 INP status (MPU inputting from I/O port)
47.	SMEMR	Latched 8080 MEMR status (MPU doing memory read)
48.	SHLTA	Latched 8080 HLTA status (acknowledges Halt)
49.	CLOCK	2-MHz clock
50.	GND	Circuit ground
51.	+8 V unregulated	
52.	−16 V unregulated	
53.	$\overline{\text{SSW DSB}}$	Sense switch disable (MPU inputs data from sense switches)
54.	$\overline{\text{EXT CLR}}$	Clear signal for I/O devices from front panel switch
55.	RTC	Real time clock
56.	$\overline{\text{STSTB}}$	Strobe signal (8800B only)
57.	$\overline{\text{DIG-1}}$	Enable signal for CPU DI drivers (8800B only)
58.	FRDY	Front panel ready signal (8800B only)
59.	N.C.	
60.	N.C.	
61.	N.C.	
62.	N.C.	
63.	N.C.	
64.	N.C.	
65.	N.C.	
66.	N.C.	
67.	N.C.	
68.	MWRT	Write enable signal for memory
69.	$\overline{\text{PS}}$	Indicates if addressed memory is protected
70.	PROT	Input to memory protect circuit on RAM PCB
71.	RUN	Indicates machine is in run mode
72.	PRDY	see XRDY
73.	$\overline{\text{PINT}}$	Input to 8080 interrupt request
74.	$\overline{\text{PHOLD}}$	Input to 8080 hold request
75.	PRESET	Clear signal for CPU
76.	PSYNC	Buffered 8080 SYNC signal (indicates beginning of machine cycle)
77.	$\overline{\text{PWR}}$	Buffered 8080 write enable signal

78.	PDBIN	Buffered 8080 BDIN signal (MPU data bus is in input mode)
79.	A0	Buffered Address lines
80.	A1	
81.	A2	
82.	A6	
83.	A7	
84.	A8	
85.	A13	
86.	A14	
87.	A11	
88.	DO-2	Buffered Data Out lines
89.	DO-3	
90.	DO-7	
91.	DI-4	Buffered Data Input lines
92.	DI-5	
93.	DI-6	
94.	DI-1	
95.	DI-0	
96.	SINTA	Latched 8080 INTA status (acknowledges interrupt)
97.	$\overline{\text{SWO}}$	Latched 8080 WO status (indicates memory write cycle)
98.	SSTACK	Latched 8080 STACK status (stack address is on address bus)
99.	$\overline{\text{POC}}$	Clear signal during power up
100.	GND	Circuit ground

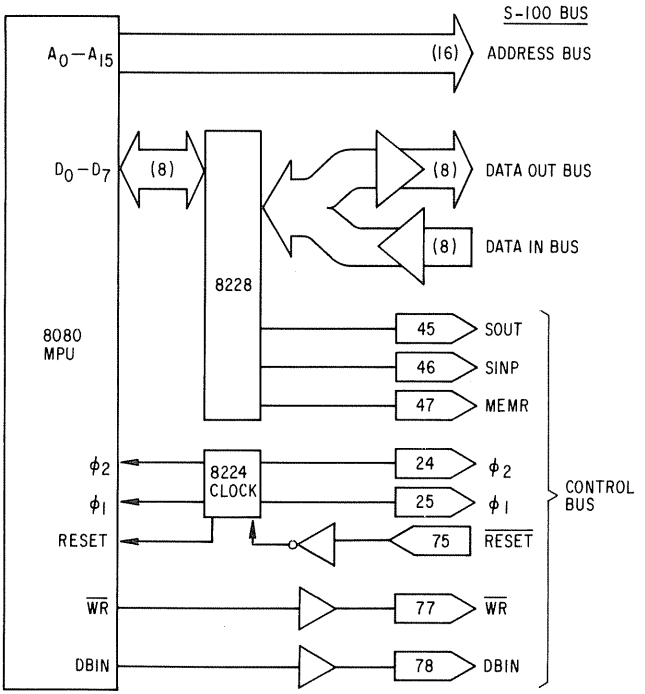


Fig. 7-2. Simplified diagram of 8080 and S-100 bus.

The S-100 bus structure is related to the MPU as shown in Fig. 7-2. SINP and SOUT differentiate I/O and memory operations. When either = 1, memory PCBs are disabled and I/O ports are enabled. An example of an S-100 compatible RAM is shown in Fig. 7-3.

The RAM has a *memory protect* R-S type flip-flop controlled by the Memory Protect (70) and Memory Unprotect (20) lines. A 1-level pulse will

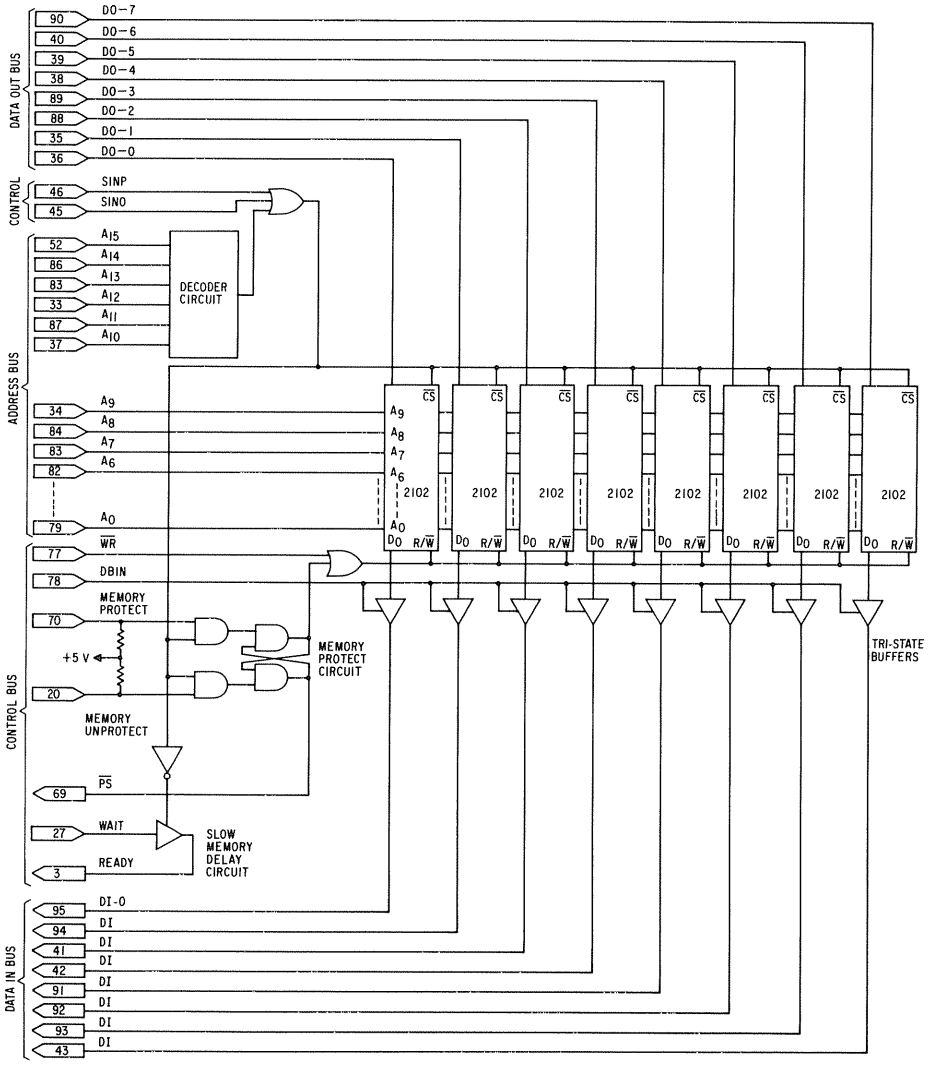


Fig. 7-3. S-100 compatible 1K × 8 RAM.

either set (pin 70) or reset (pin 20) the flip-flop. When the flip-flop is set the RAM cannot be written into.

If SINP or SINO (I/O in and out status lines) = 1 the memory is disabled. This occurs during I/O transfers. Notice that the MPU ready line (3) can be used to halt the MPU for a short period if slow memory ICs are used. The circuit shown generates a 1-cycle delay.

The data input (to MPU) bus has tri-state buffers controlled by the DBIN (data bus input) control lines.

TRI-STATE BUSING (TSL)

Tri-state logic (TSL) gates are presently the most widely used gates for bus structuring. TTL gates cannot be used since a short circuit condition is created when one TTL bus gate is on and another is off. Figure 7-4 shows the TTL equivalent output circuit. In order to maintain the high switching speeds of TTL logic, TSL was created. The TSL gate has an added control input. This control input effectively opens both switches at the output of the TSL gate causing the output to float in a high impedance condition (Fig. 7-5).

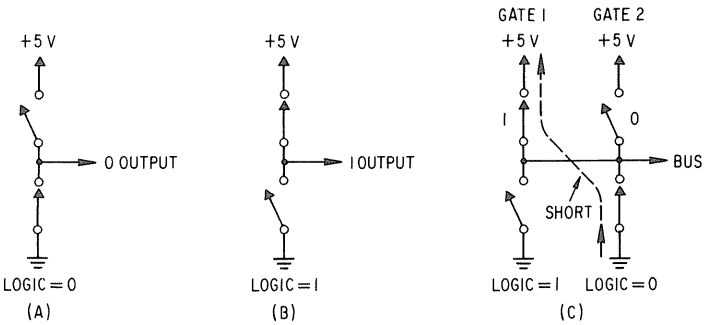


Fig. 7-4. (A and B) TTL equivalent output circuit; (C) short created by busing, TTL gates.

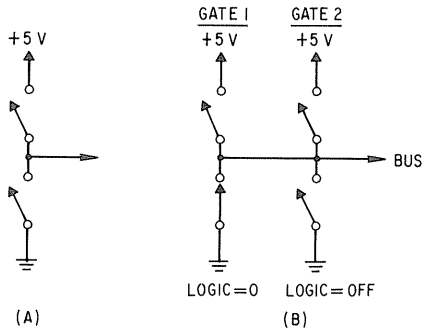


Fig. 7-5. (A) TSL gate shown in OFF state; (B) TSL bus.

When the control input = 1 or is disconnected, the TSL gates' output floats. When the control input =0, the TSL gate operates as a standard TTL gate. Two typical TSL ICs are shown in Fig. 7-6.

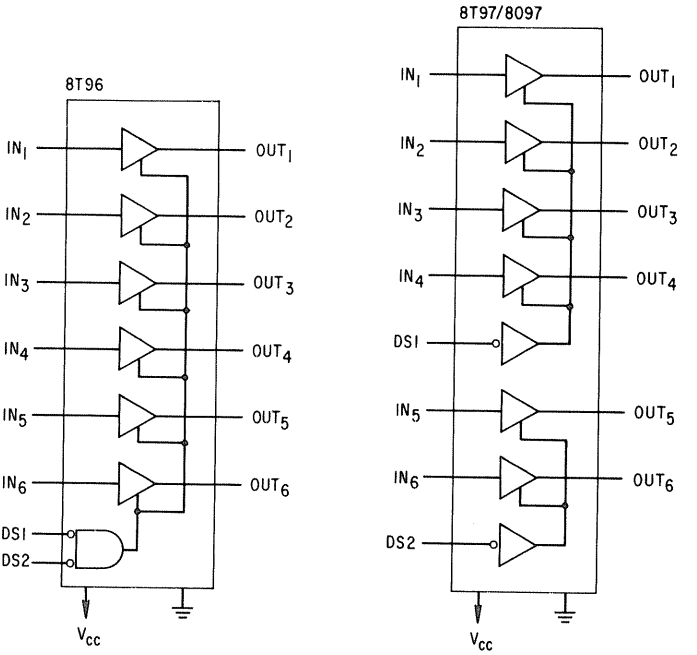


Fig. 7-6. Two typical TSL ICs.

MULTIPLEXING AND DEMULTIPLEXING

It is impractical to run separate lines for all signals present in a CPU. Frequently, two or more signals may be run on a given line. This is called

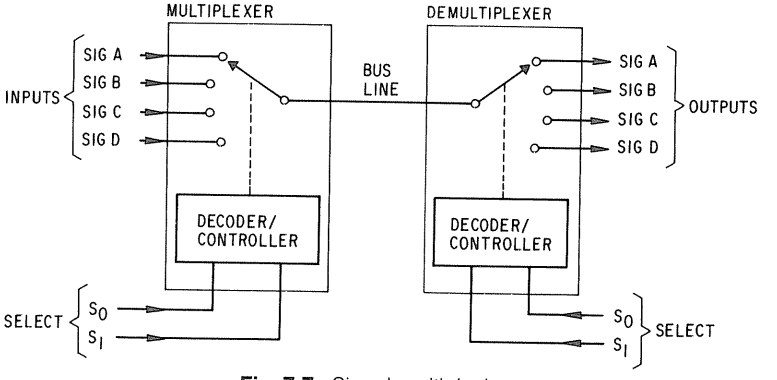


Fig. 7-7. Signal multiplexing.

multiplexing and the circuit that accomplishes this is known as a *multiplexer* (*mux*). The mux is used at the transmitting point (Fig. 7-7) and a *demultiplexer* (*demux*) is used at the receiving end.

The multiplexing and demultiplexing circuits are basically selectors with a decoder/controller circuit to determine which signal is on the line. Only one signal is on the line at a time. The select signals determine which signal is on the line, as follows:

S_0	S_1	Signal on Bus
0	0	A
0	1	B
1	0	C
1	1	D

The basic circuits of the mux and demux are shown in Fig. 7-8. The S_0-S_1 signals are decoded to enable one of the four AND gates routing one of the signals on to the bus. A typical mux IC is the 74151 shown in Fig. 7-9. It will select 1 of 8 signals and place the signal on the output (both true and complemented outputs available) when the strobe enable = 1.

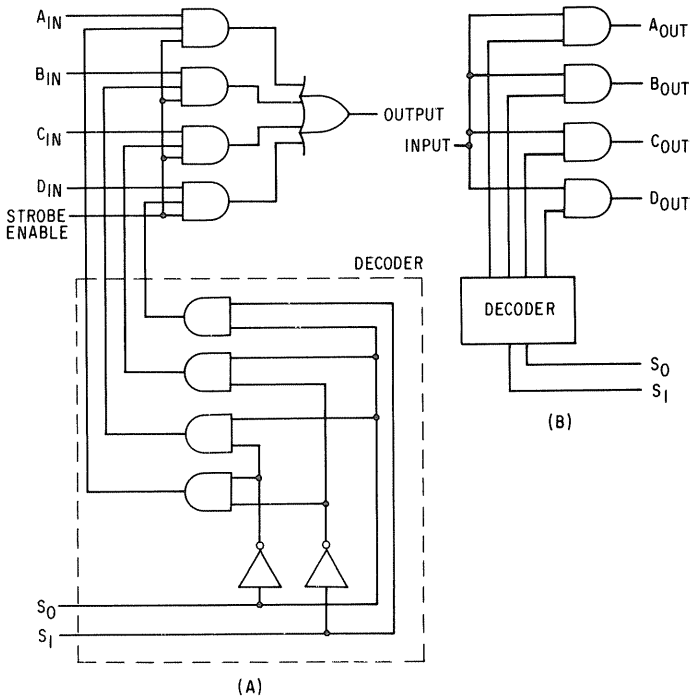


Fig. 7-8. (A) Basic multiplexer and (B) demultiplexer.

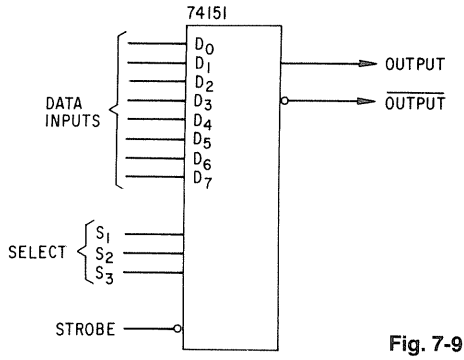


Fig. 7-9. The 74151 IC multiplexer.

SERIAL INTERFACING

The CPU bus operates in parallel. All data, address, and control signals appear on the bus lines at the same time. To economize on bus lines, data is often sent over only a single pair of lines, 1 bit at a time. In other words, the data is transmitted *serially*. The 74151 mux IC shown in Fig. 7-9 can be used to convert an 8-bit parallel data word to a serial word. If each input is selected in succession, then the data bits will appear at the output in succession (serially), 1 bit at a time. A typical serial transmission appears as shown in Fig. 7-10. A clock input determines the rate at which bits are transmitted and received. The number of bits per second transmitted is referred to as the *baud rate*. For example, 300 bits per second is equivalent to 300 baud.

A protocol has been adopted so that the receiving device will know when a serial word starts and ends. The word begins with a *start bit* (=0) followed by the data bits, a parity bit, and one or two *stop bits* (=1), as shown in Fig. 7-11. For example, if the data word is 0110101 (7 bits) and even parity is sent, the transmitted word will be 00110101111 and appear as shown in Fig. 7-12.

A Teletype, which uses 1 start bit, 7 data bits, 1 parity bit, and 2 stop bits, uses 11 bits to transmit each character. It operates at 10 characters per second (cps) and hence 110 baud. Most higher speed terminals use only 1 stop bit. Hence, a 30-cps terminal transmits at 300 baud.

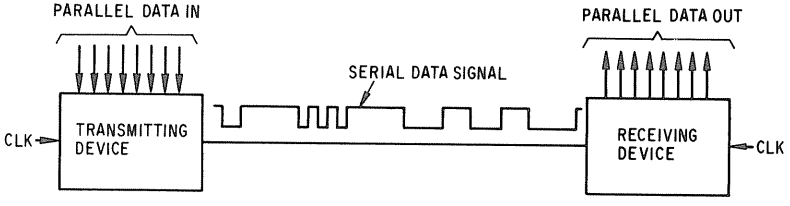


Fig. 7-10. Serial data transmission.

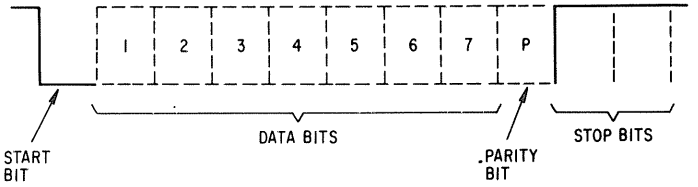


Fig. 7-11. Format for serial transmission.

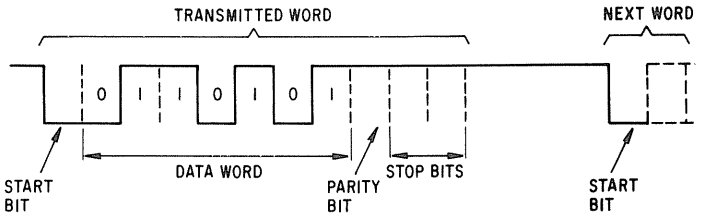


Fig. 7-12. Transmission of the word 0110101.

The UART

An IC has been developed to handle serial-to-parallel (and vice versa) data transmission and reception, and provide the proper protocol and interfacing signals to the CPU. This IC is called a *UART* (*universal asynchronous receiver/transmitter*). *Asynchronous transmission* means that the transmitter and receiver are not synchronized to the CPU. Instead, handshaking signals are used to indicate I/O operations.

A widely used UART IC is the AY-5-1013 shown in Fig. 7-13. It contains both a transmitter (parallel-to-serial shift register) and a receiver (serial-to-parallel shift register). Each is clocked separately and may be at different rates, although usually they are the same. The transmitter has a buffer register to latch the parallel input word and control logic to add start, stop, and parity bits. The UART is programmable for the number of data and stop bits and odd/even parity. In addition, there are *handshaking* signals: TBMT indicates that the input buffer is empty (ready to receive a new word) and TEDC indicates that the UART has completed transmitting a word.

The receiver has an output buffer register to hold the received word until the CPU is ready to take it. The receiver handshaking signals are *ODA* (*output data available*) and *RDA* (*reset data available*). In addition, the receiver can detect a framing error (improper start or stop bits), parity error (improper data), and overrun (character previously received has not been read by the CPU). These error outputs are labeled RFE, RPE, and ROR.

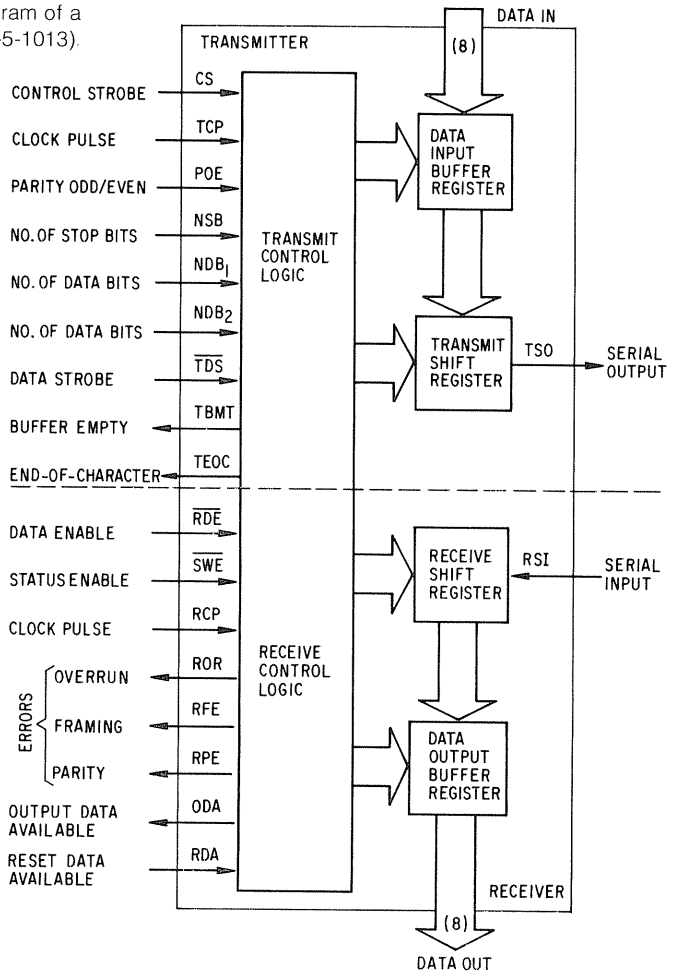


Fig. 7-13. Block diagram of a UART (AY-5-1013).

The ACIA

The *ACIA* (*asynchronous communications interface adapter* IC) is very similar to the UART. It differs in that it incorporates the multiplexing circuitry necessary for connection to a bi-directional data bus and a control bus. In other words it has the interface circuitry to make it directly compatible with a CPU bus.

A very popular ACIA is the Motorola 6850 (Fig. 7-14). It also features control circuitry that can be programmed by the CPU via the data bus during system initialization. This includes setting of word length, clock division ratio, and transmit, receive, and interrupt control.

The 6850 has four registers which may be addressed by the MPU. The MPU can read the contents of the status and receive data registers and can write

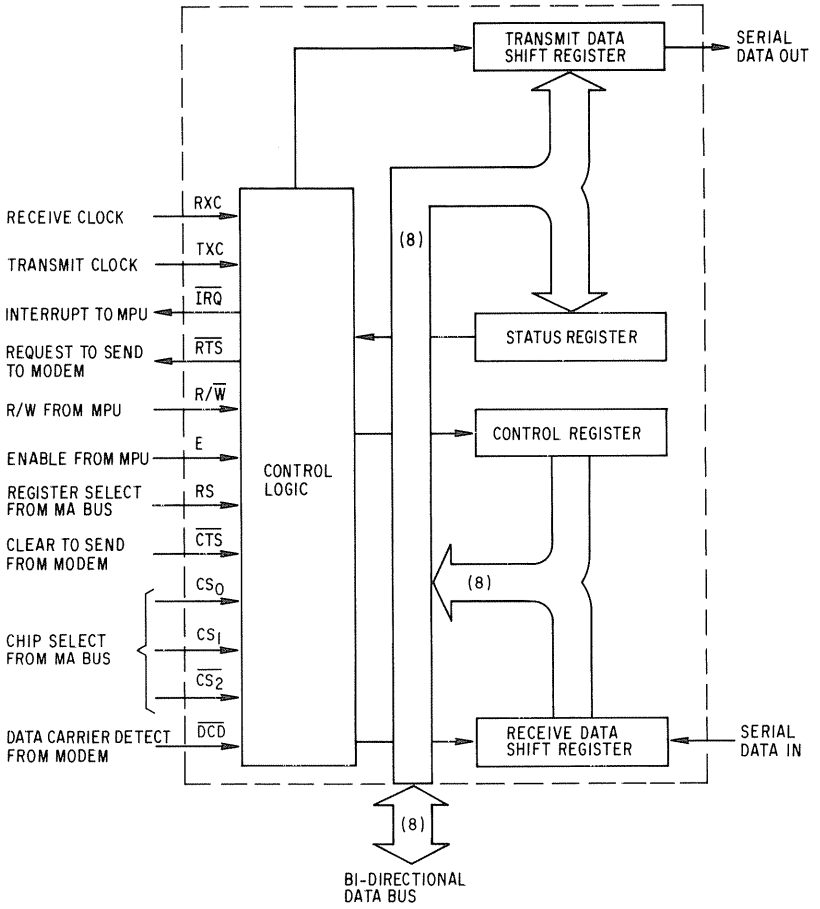


Fig. 7-14. The 6850 ACIA functional block diagram.

into the transmit and control registers. The 6850 has the additional feature of handshaking lines for direct connection to a Modem (to be discussed later).

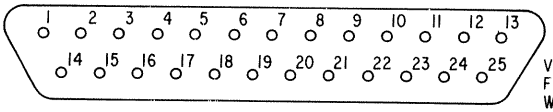
RS-232 INTERFACE

The serial in and out logic levels of the UART and ACIA are TTL logic levels, i.e., a logic-0 is between 0 and +0.8 V and a logic-1 is between +2.4 and +5 V. The maximum noise possible before interference occurs is 1.2 V (2.4−0.8 V). This generally limits line lengths between TTL devices to a few feet. This presents a serious problem when connecting to a printer or terminal where lines may be up to 50 ft or more.

Furthermore, there are the problems of line capacitance and induced surges from motors, lighting circuits, etc. It was for this reason that manufacturers got together through the EIA (Electrical Industries Association) and developed a standard, commonly referred to as RS-232, to define interconnect signals, voltages, and connectors. The latest version of this specification is RS-232C. It is the intent of RS-232 to allow line lengths of up to 50 ft without the use of a modem under any conditions of noise.

A control signal (e.g., request to send, data carrier detect, etc.) is defined as between +3 and +25 V in the one (1) state and between −3 and −25 V in the zero (0) state. Data is just the opposite: a logic-1 (called a *MARK*) is between −3 and −25 V and a logic-0 (called a *SPACE*) is between +3 and +25 V. Therefore, noise must swing at least 6 V before it will cause trouble. In fact, most manufacturers use output signals of +12 and −12 V, providing up to 24 V of noise immunity.

As a further aid to standardization, RS-232 specifies a connector type (the 25-pin D-connector made by Cinch and others) but also specifies pin assignment as shown in Fig. 7-15. Although not shown, all pins are assigned, and if you are



1. FRAME GND
2. TRANSMITTED DATA
3. RECEIVED DATA
4. REQUEST TO SEND (FROM TERMINAL/COMPUTER)
5. CLEAR TO SEND (FROM MODEM)
6. DATA SET READY (= 1 WITH MODEM POWER)
7. SIGNAL GND
12. DATA CARRIER DETECT (FROM MODEM)
13. DATA TERMINAL READY (TO MODEM)

Fig. 7-15. RS-232 connector pin assignments.

connecting to an unknown peripheral or modem, it is best to check which pins are used. Also, these pin assignments assume that *send* and *receive* are from a line that has a modem at each end; i.e., at each end of the line, the *send data* line of the data terminal goes to the *send data* terminal of the modem. However, when not using a modem and making direct connections between a CPU and a peripheral, the *send data* line of the CPU must be connected to the *receive data* line of the peripheral and vice versa (Fig. 7-16).

The conversion from TTL to RS-232 voltage levels, and vice versa, is easily accomplished with ICs. Most common are the 1488 (TTL-to-RS-232 quad driver) and 1489 (RS-232-to-TTL quad receiver) (Fig. 7-17).

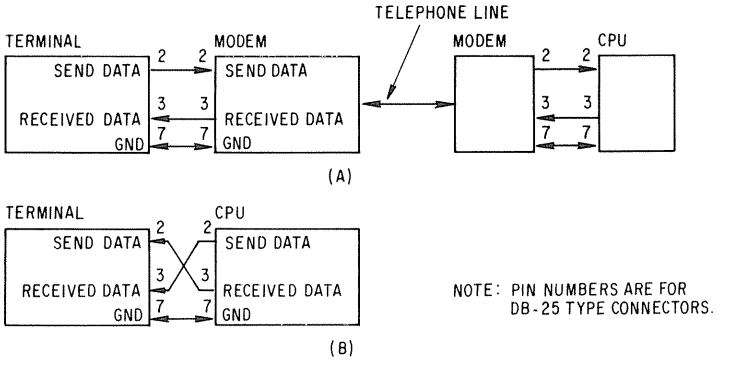


Fig. 7-16. Terminal/modem versus terminal/CPU RS-232 connections.

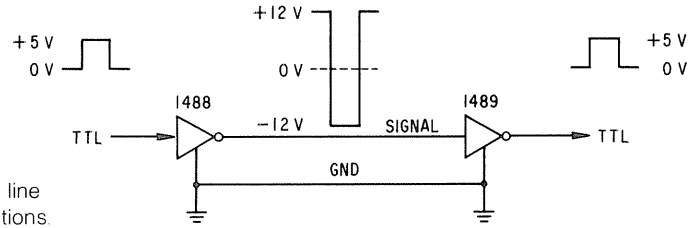


Fig. 7-17. RS-232 line connections.

CURRENT LOOP INTERFACE

The *receive data* line of a Teletype (TTY) is normally connected to the selector magnet through a resistor and draws either 20 mA (milliamperes) (330-Ω resistor) or 60 mA (160-Ω resistor) from the line. Likewise, the Teletype *send data* line expects the device at the far end to draw either 20 mA or 60 mA from the line. For this reason, most interface boards have a set of discrete transistors and resistors arranged in a configuration which will source 20 mA (send) or sink 20 mA (receive). The TTY should be checked to determine whether 20 or 60 mA is required. Most CPU circuits provide only 20-mA capability which a 60-mA TTY will not receive. Furthermore, the send circuit may burn out the receive circuit in the CPU interface.

A typical TTY (20-mA) interface circuit is shown in Fig. 7-18. Notice that an *opto-coupler* (LED and photo-transistor in one case) is used to completely isolate the TTL and 20-mA circuits. When the send switch of the TTY is closed, 20 mA flows from −12 to +12 V through the LED, causing it to light. The LED light turns on the photo-transistor, causing a TTL 0-logic level. When the send switch is open, the photo-transistor is off and the TTL logic level =1 (+5 V).

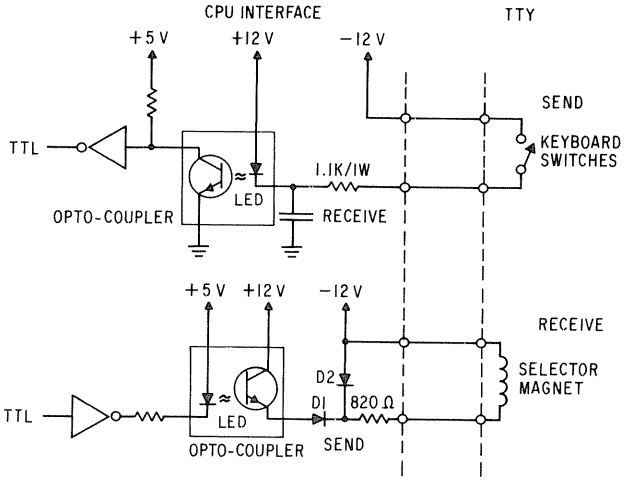


Fig. 7-18. A TTY 20-mA loop interface circuit.

The CPU send circuit causes 20 mA to flow when a TTL 0-logic level is present, and no current to flow when a 1-level is present. The two diodes protect the photo-transistor from current spikes induced by the selector magnet.

MODEMS

In noisy environments and where the distance between the CPU and terminal is great, modems must be used. Modem is a contraction of *modulator-demodulator,* a circuit which allows communication of digital data signals over telephone, radio, and other limited-bandwidth communications channels. Modems can operate *half-duplex* (one direction at a time) or *full-duplex* (both directions simultaneously). Modems can operate up to 9,600 baud over telephone circuits, although half-duplex usually limits maximum speed to 300 baud.

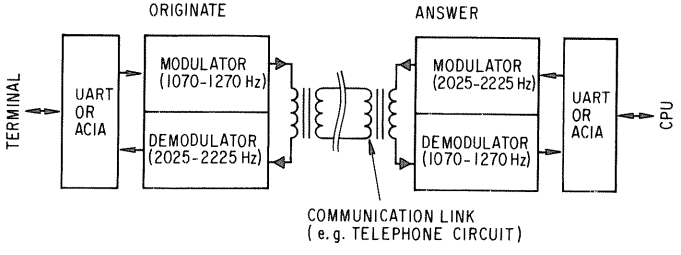


Fig. 7-19. A basic modem data communication system.

Modems in this speed range usually use an *FSK* (*frequency shift keying*) technique using four frequencies in two bands. The *low band* (1,070 to 1,270 Hz) is used for the *originate* (the terminal) modem, and the *high band* (2,025 to 2,225 Hz) is used for the *answer* (CPU) modem. The basic setup is shown in Fig. 7-19.

The originate modem generates a 1,270-Hz mark (1) or 1,070-Hz space (0) frequency. The answer modem generates a 2,225-Hz mark (1) or 2,025-Hz space (0) frequency.

Modems use either *acoustical couplers* (Fig. 7-20) to the telephone handset or direct-wire connection to the telephone line. The acoustical coupler avoids the necessity for the installation and rental charges for the special *DAA* (*data access arrangements*) which the phone company insists upon when a direct connection is made. Acoustic couplers do, however, have noise and distortion problems which can cause errors.

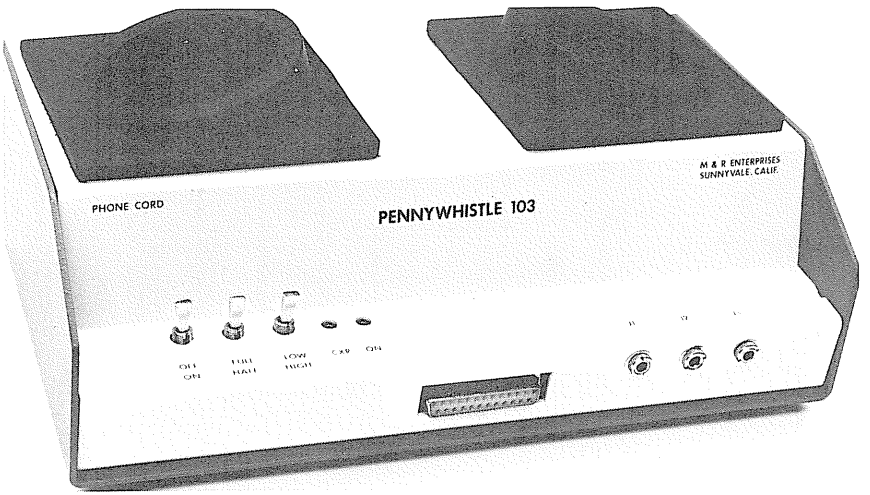


Fig. 7-20. A typical modem using an acoustical coupler.

The modem interfaces to the UART or ACIA of the terminal or CPU. The block diagram of the M & R Enterprises *Pennywhistle 103* acoustical coupler type modem is shown in Fig. 7-21 and the schematic in Fig. 7-22. The receiver contains a phase-locked loop (PLL) IC (A3) which contains a variable frequency oscillator and comparator. The IC is always trying to match the oscillator frequency to the incoming frequency. A correction voltage fed by the comparator to the oscillator is a measure of the difference between the incoming frequency and the preset center frequency. A three-stage active filter (A1-A2) ahead of the PLL prevents noise and harmonics from getting through.

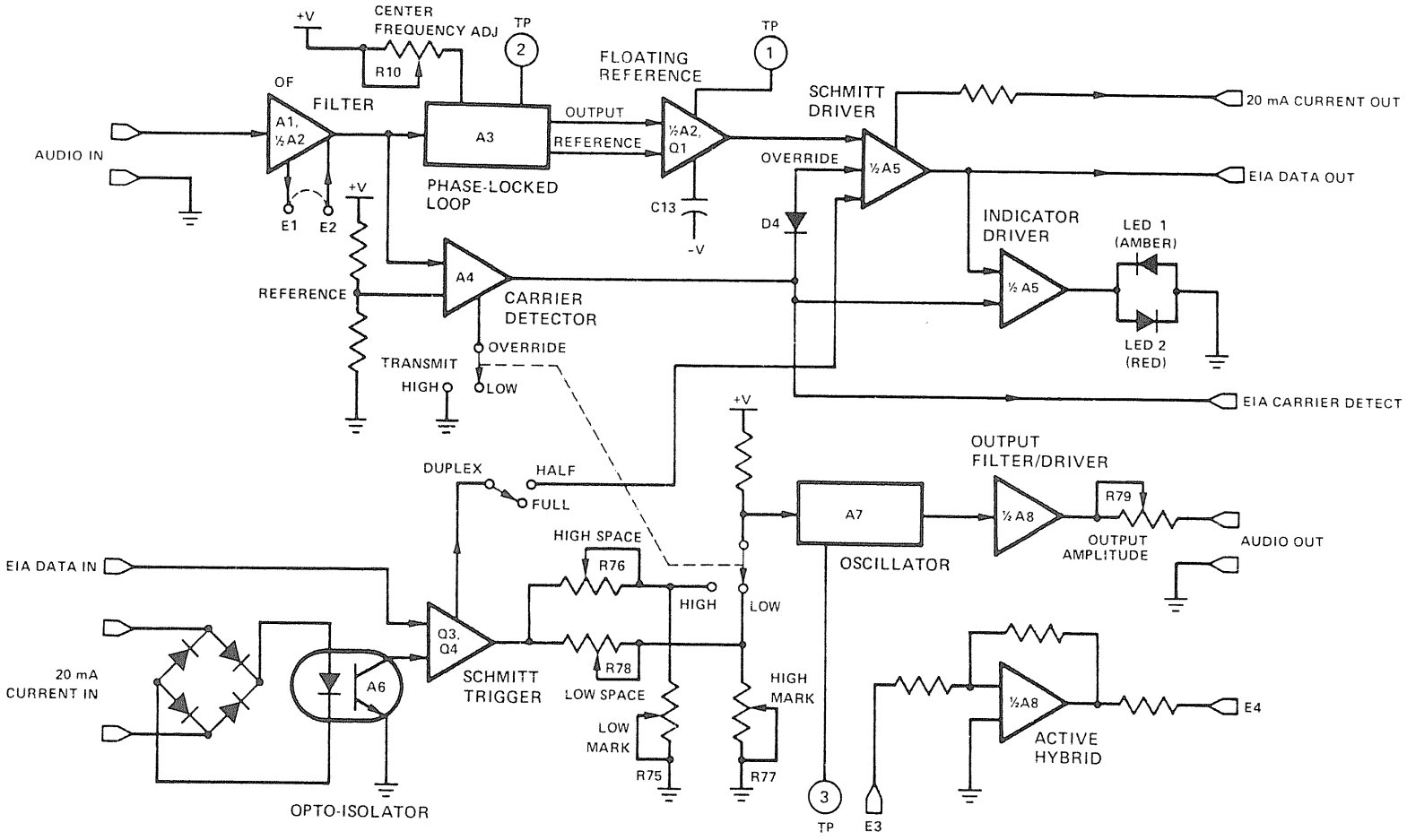


Fig. 7-21. Block diagram—Pennywhistle 103 modem. (Courtesy M & R Enterprises)

R10 sets the center frequency of the PLL. A floating reference circuit (Q1 and A4) stabilizes the circuit to prevent drift due to temperature changes. The reference voltage is fed to a Schmitt trigger (A5) to eliminate any noise and then to EIA and 20-mA loop output circuits. A4 detects when sufficient signal level exists for reliable operation and provides an EIA carrier detect signal to the terminals and holds the data at a mark when insufficient signal is present.

Two LEDs (one amber and the other red) are driven by A5 to indicate the data logic levels (1 = amber, 0 = red). The carrier detect signal turns on the transmitting oscillator through R59. Data is looped back from the transmitter to the receiver when the *half-full duplex switch* is in the half position. This provides an *echo* of the data originated at the terminal.

The transmitter uses a 555 IC (A7) oscillator followed by a filter/driver (A8) to reduce harmonics and provide the power to drive the small speaker in the acoustic coupler. The frequency is controlled by the voltage at the base of Q6. Q3 and Q4 form a Schmitt trigger circuit which takes data in from either an EIA or 20-mA current loop via an opto-isolator.

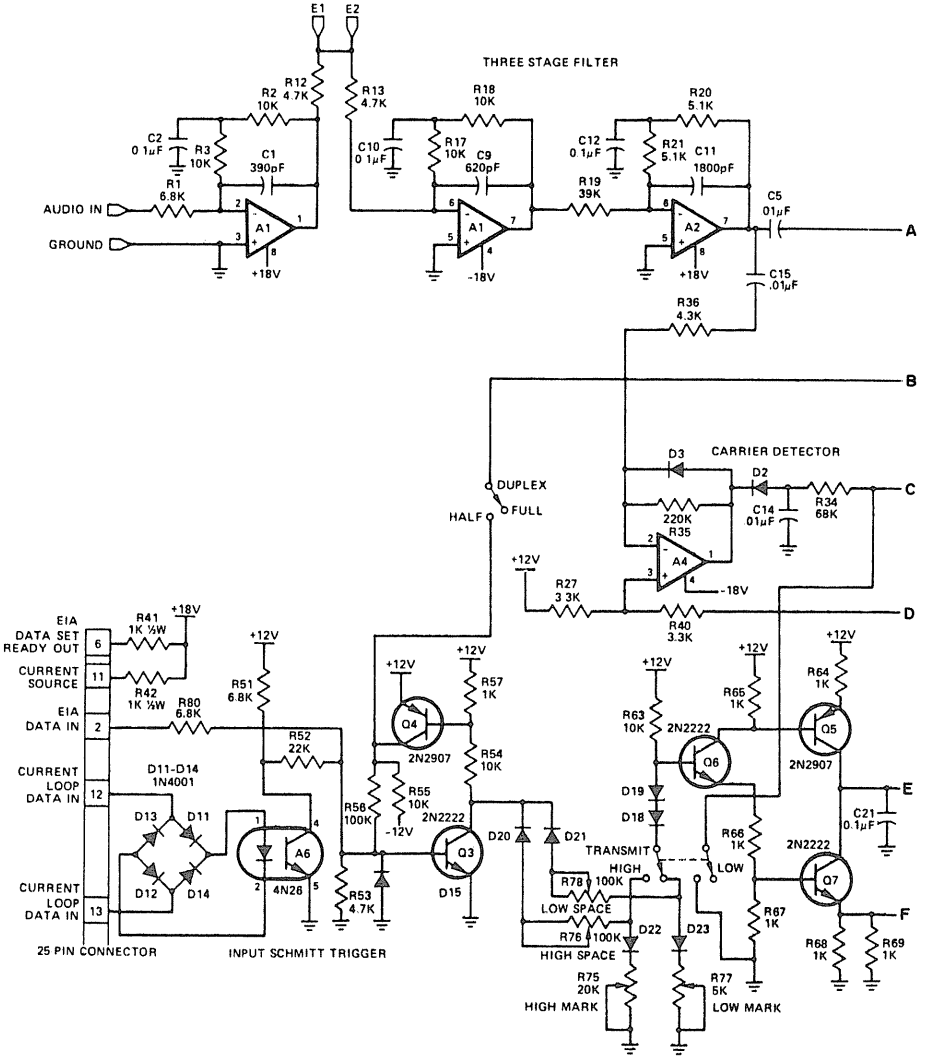
PARALLEL INTERFACING

The CPU can output or input all the data bits at one time via a *parallel port*. It is the job of the parallel interface to translate signal levels, synchronize signals, store data temporarily, and isolate the CPU from unwanted data. Figure 7-23 shows a simple parallel input port with TSL outputs. When the port is addressed and an *in* signal is given by the MPU, the input data word is placed on the data bus and inputted to the CPU.

A simple parallel output port is shown in Fig. 7-24. It consists of an 8-bit latch that latches the data on the CPU bus when the port is addressed and an *out* signal is given by the MPU.

Parallel Port ICs

A very popular IC for parallel ports is the Intel 8212. It can be used for input or output ports. As shown in Fig. 7-25 it consists of an 8-bit data latch with Q outputs connected to TSL buffers. Data flow is controlled by the MD, STB, DS1, and DS2 inputs. If MD = 1, the DS1/DS2 inputs will clock the latch and the buffers are enabled to provide output port operation. If MD = 0, the STB input clocks the latch to load data. DS1/DS2 then enable the buffers to transfer data to the CPU bus for input port operation. The 8212 may be used to initiate an interrupt (when INT output is connected to INT input of MPU). The INT output signals the MPU that data has been loaded into the data latch. Reading the port resets the INT output.


E1
E2
THREE STAGE FILTER
R12 4.7K
R2 10K
R13 4.7K
R18 10K
R20 5.1K
C2 0.1μF
R3 10K
C1 390pF
C10 0.1μF
R17 10K
C9 620pF
C12 0.1μF
R21 5.1K
C11 1800pF
R1 6.8K
R19 39K
C5 .01μF
AUDIO IN
GROUND
A1
A2
+18V
-18V
C15 .01μF
A
R36 4.3K
B
D3
CARRIER DETECTOR
D2
DUPLEX
HALF
FULL
220K
R35
C14 .01μF
R34 68K
C
A4
+12V
R27 3.3K
-18V
D
R40 3.3K
EIA DATA SET READY OUT
CURRENT SOURCE
EIA DATA IN
CURRENT LOOP DATA IN
CURRENT LOOP DATA IN
25 PIN CONNECTOR
R41 1K ½W
R42 1K ½W
R80 6.8K
R51 6.8K
R52 22K
D11-D14 1N4001
D13
D11
D12
D14
A6
4N26
INPUT SCHMITT TRIGGER
R53 4.7K
R56 100K
R55 10K
-12V
Q4
2N2907
R57 1K
R54 10K
2N2222
Q3
D15
D20
D21
R63 10K
R65 1K
Q6
2N2222
D19
D18
TRANSMIT
HIGH
LOW
R78 100K
LOW SPACE
R76 100K
HIGH SPACE
D22
D23
R75 20K
HIGH MARK
R77 5K
LOW MARK
R64 1K
Q5
2N2907
E
C21 0.1μF
R66 1K
2N2222
Q7
R67 1K
F
R68 1K
R69 1K

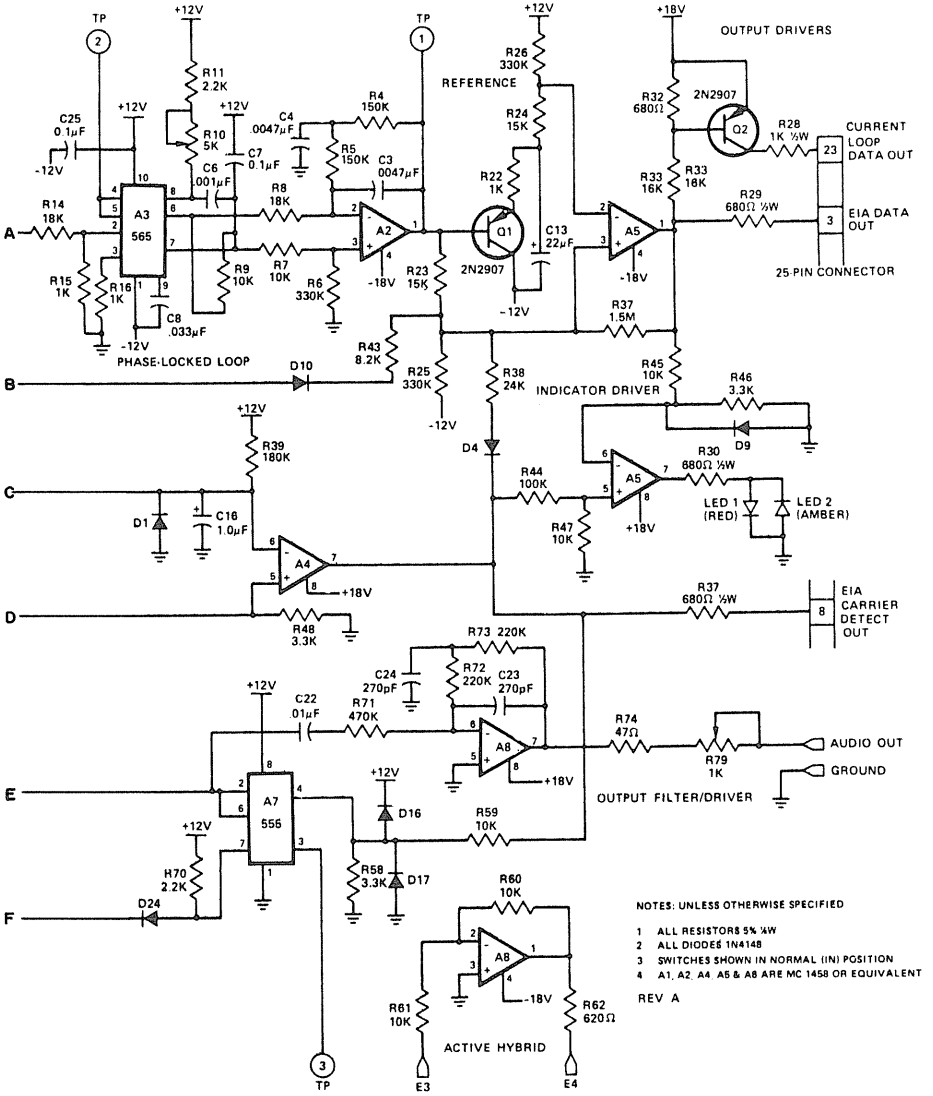


Fig. 7-22. Schematic diagram—Pennywhistle 103 modern. (Courtesy M & R Enterprises)

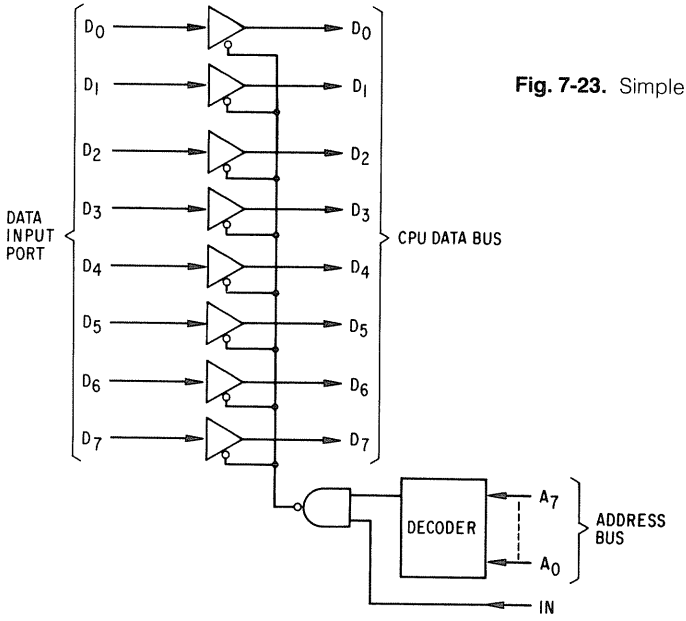


Fig. 7-23. Simple parallel input port.

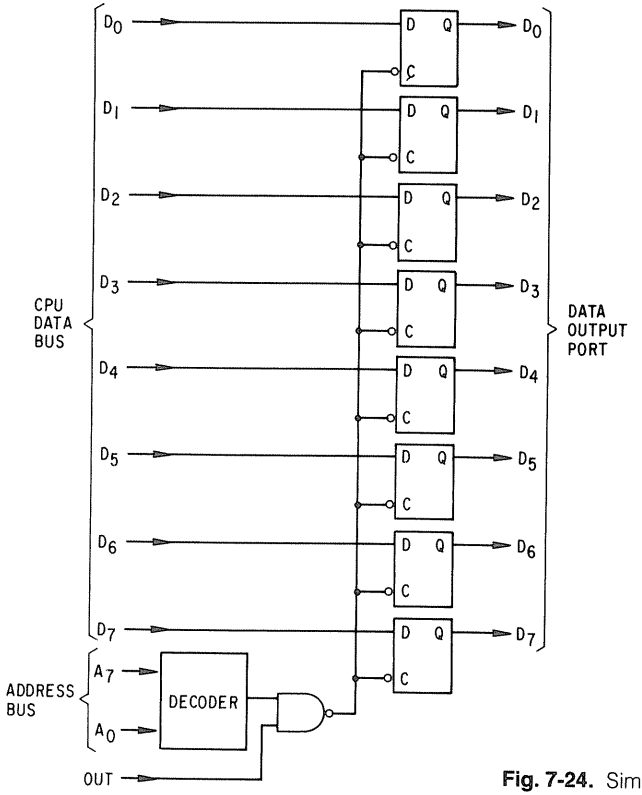


Fig. 7-24. Simple parallel output port.

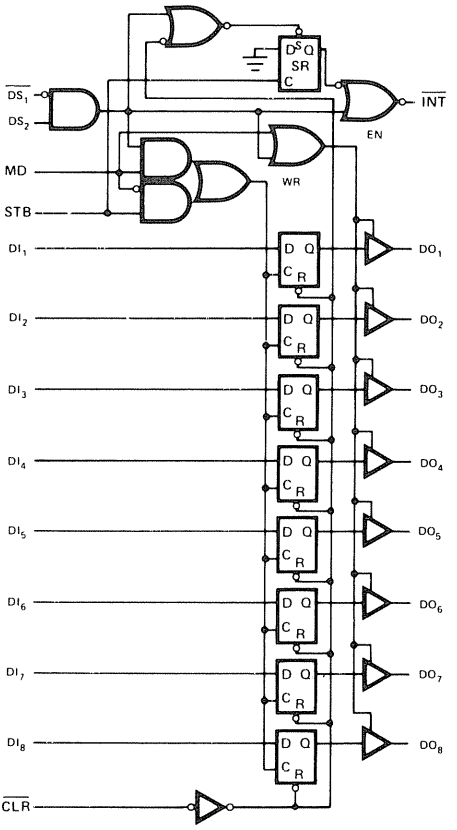


Fig. 7-25. 8212 I/O IC port schematic. (Courtesy Intel)

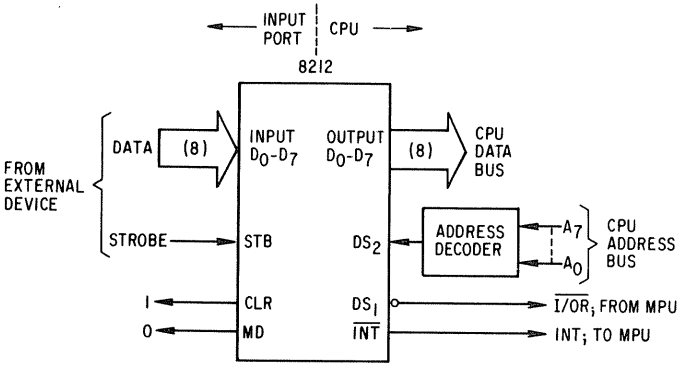


Fig. 7-26. 8212 used as an input port.

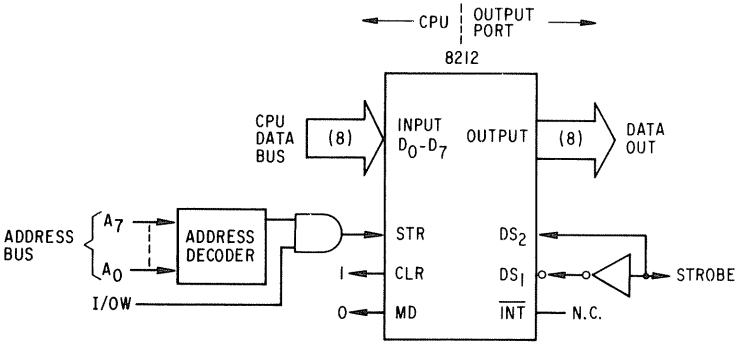


Fig. 7-27. 8212 used as an output port.

The use of the 8212 as an input port is shown in Fig. 7-26, and its use as an output port is shown in Fig. 7-27. Intel also makes a dual programmable peripheral interface (PPI) IC, the 8255. It may be configured as one, two, or three I/O ports.

A more popular programmable IC is the Motorola 6820 *PIA (peripheral interface adapter)*. The 6820 (Fig. 7-28), has two 8-bit bi-directional data registers to provide two I/O ports. Each port is programmable, via two control and data direction registers. Therefore the MPU during initialization may program the two ports as to their data direction, handshaking, and interrupt processing.

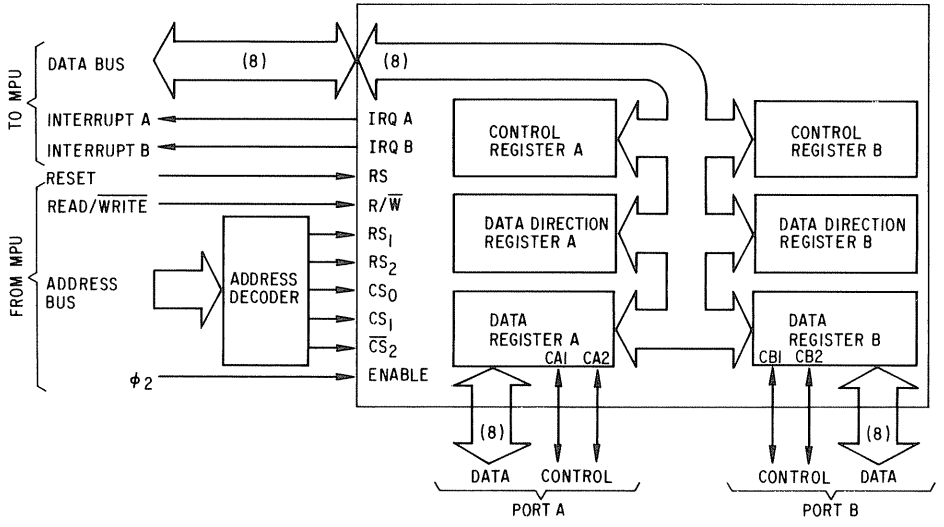


Fig. 7-28. The 6820 dual peripheral interface adapter (PIA) IC.

THE IEEE STD 488 BUS

In 1973 Hewlett-Packard standardized on a communications data bus to be used between their test instruments. In 1976 the IEEE (Institute for Electrical & Electronics Engineers) adopted it as a standard, labeling it Std-488. It permits the interfacing of five categories of instruments: stimulus, measurement, storage, displays, and control. As shown in Fig. 7-29, it contains 16 signal lines—8 bi-directional data lines, 5 general interface-management lines, and 3 byte-transfer-control lines.

Each station on the bus can work as a *controller,* a *talker,* or a *listener.* Some devices can assume multiple roles, such as a CPU (control, talk, and listen) or a digital multimeter (talk and listen). Other instruments, like counters and signal generators, are not as versatile.

Up to 15 stations are permitted on the bus and many stations may listen at the same time. One byte is transferred at a time, asynchronously, without strobes. First the control station sets the attention line $=1$, and sends out the addresses of all the stations to be involved in the upcoming data transfer. In turn, each station recognizes its address on the data bus and prepares to listen or talk. However, only one station may talk at a time. The byte-transfer control lines then support the per-byte handshake protocol.

ANALOG-TO-DIGITAL AND DIGITAL-TO-ANALOG CONVERSION

A digital signal has only two voltage levels. The industry has standardized on 0 V and +5 V. Ideally, there are no other voltage levels used in digital circuits. When we deal with non-digital circuits, e.g., varying the volume of an amplifier, we are dealing with *analog* signals that vary between a set of limits. For example, an analog signal may be 0 V, +5 V, −5 V or any voltage between the limits of +5 V and −5 V. A sine-wave signal is another very common example of an analog signal.

Very often we wish to interface analog devices and digital computers. For example, we may wish to connect a "joy-stick" (two-potentiometer X–Y position controller) to the input of the computer. Or, in another case, we might input the analog signal of a temperature transducer to the computer. In both of these cases we are inputting analog signals to the computer. This is accomplished by means of an *analog-to-digital converter* interface circuit, which is called an *ADC* for short.

On the other hand, we have *digital-to-analog* conversion, called *DAC* for short. For example, when the digital signals of the computer are used to generate music, we use DACs. Other examples include display of signals on an oscilloscope and the control of a motor.

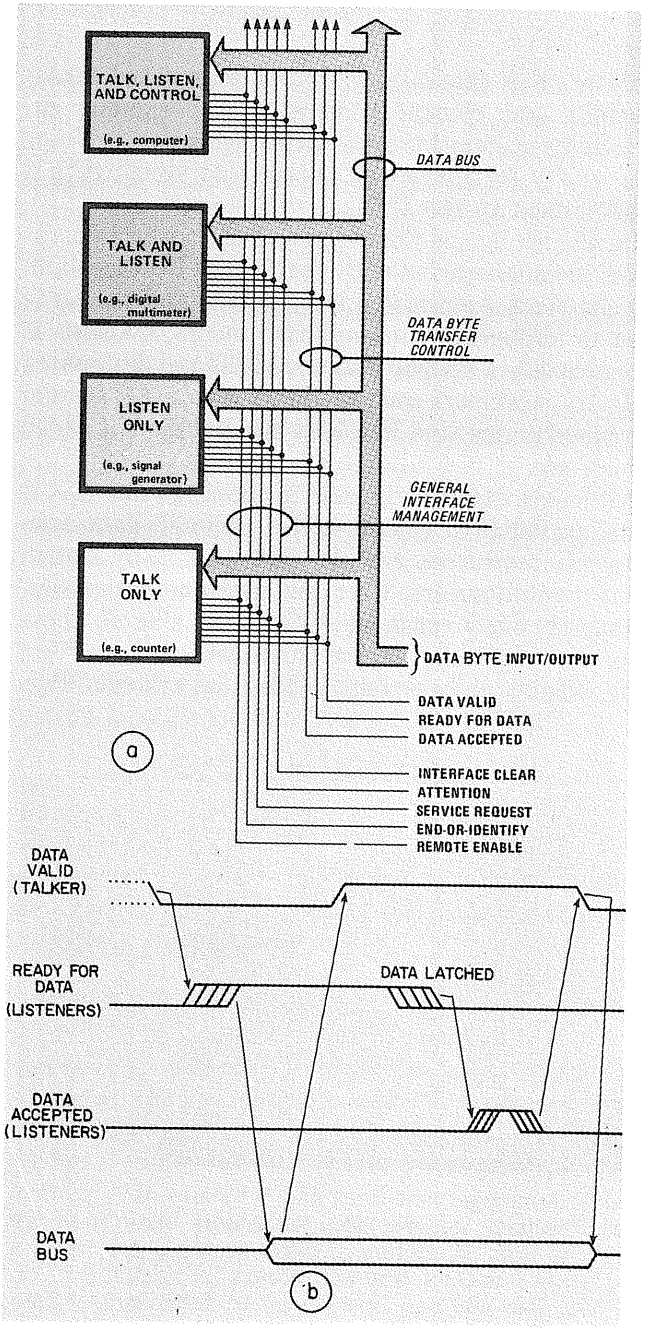


Fig. 7-29. The IEEE STD 488 bus and signals.

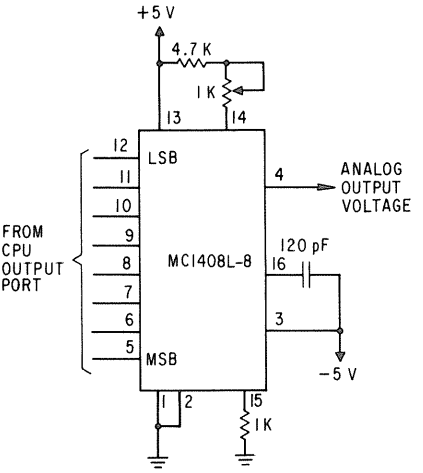


Fig. 7-30. An IC-DAC used with a microcomputer system.

Digital-to-Analog Conversion

Low-cost DACs are available in IC form. A very popular IC DAC is the Motorola MC1408L-8. This device is shown in Fig. 7-30. The DAC is driven from a standard output port latch such as the 8212 IC.

Analog-to-Digital Conversion

ADC circuits, in module form, are readily available which convert analog signals rapidly and accurately and require little or no programming in the microcomputer system. Unfortunately, these units tend to be expensive. In cases where speed and accuracy are needed they should be used. If conversion speed is not important, simpler ADCs can be used, which require program instructions in the CPU.

A simple ADC circuit is shown in Fig. 7-31. It is used to sense the position of a potentiometer (R1). It uses a 555 timer IC operating as a one-shot. The

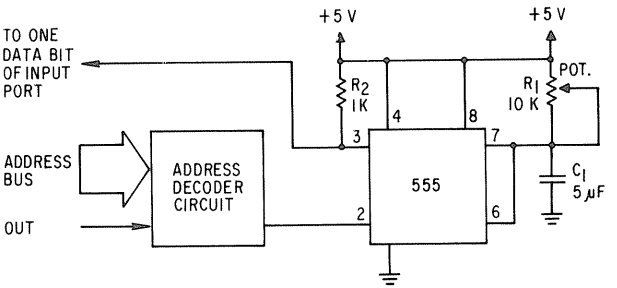


Fig. 7-31. Simple ADC to input potentiometer position (voltage) to CPU.

duration of the output pulses is controlled by R1 and capacitor C1 (T=1.1RC). The one-shot is triggered by an output pulse from the CPU. The CPU then executes a delay loop until the one-shot pulse ends. The CPU counts the number of cycles occurring during the delay loop and uses this as an indication of the position of R1. If R1 is a small resistance, the count will be low, and vice versa.

A more typical ADC circuit is shown in Fig. 7-32. It employs a DAC and comparator (form of operational amplifier) to make up a *successive-approximation* type ADC. The circuit uses a parallel output port and a serial input port.

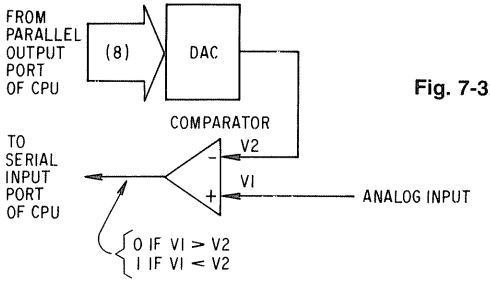


Fig. 7-32. A minimum component ADC system.

The CPU executes a program that successively approximates the analog input voltage. The DAC converts the digital output to an analog voltage (V2) and the comparator compares the DAC output to the analog input voltage (V1). The comparator output indicates if the binary approximation is too high or too low. The first approximation will be 1000 0000. If this is too low (comparator output = 1), the next binary approximation will be 1100 0000, and if too high (comparator output = 0), the next binary approximation will be 0100 0000. In this way nine successive approximations are needed to arrive at the correct binary value for the analog input.

A popular ADC/DAC for S-100 type CPUs is made by Cromenco Inc. It multiplexes seven analog inputs under CPU control to one ADC and also provides seven analog outputs. The ADC is a hardware type successive approximation circuit. A 2502 *successive approximation register* (*SAR*) together with the MC1408L-8 DAC and 710 comparator form the heart of the circuit. This is about 100 times faster than the software method described earlier.

Recommended Further Reading

1. Jean Daniel Nicoud, "Peripheral Interface Standards for Microprocessors," *Proceedings of the IEEE*, Vol. 64, No. 6, June 1976.

8.

Mass Storage Systems

The RAM of the CPU holds the program and data that is being worked on by the CPU. When not in the CPU, the program and/or data is stored in a mass storage medium. The long time-honored IBM *punched card* has been around for many years and was very low cost as a mass storage medium. However, the card readers and punches are expensive and very slow. *Magnetic tape drives* (*mag tape*) greatly improved the access time but increased the cost significantly. Teletypes with their *paper tape* reader/punch units also made possible very low cost, but slow, mass storage.

Presently, the fastest access mass storage medium is the *magnetic disc*. However, the cost is extremely high. A lower cost, but still expensive magnetic disc system has recently been introduced using *floppy discs*.

Personal computer users in nearly all cases use either paper tape, magnetic cassette tape, or floppy disc storage mediums. Also, a very new technology called *bubble* memories is being introduced for mass storage.

PAPER TAPE

Paper tape program storage is very popular among personal system users for several reasons. First, it is the only medium which is standardized. This means that a paper tape generated on one system can be used directly on virtually all other systems. This is not true for magnetic tape or disc.

Paper tape is low in cost. However, the machines for reading and generating paper tape can be expensive. An economical solution is the use of the Teletype (Fig. 8-1) which, as an adjunct, contains a paper tape reader and punch. Its disadvantage is that it is very slow, 10 cps (characters per second). This is because it is an electro-mechanical machine. There are high-speed readers and punches operating up to 400 cps, but they are very expensive.

Very low cost paper tape readers are available and one is shown in Fig. 8-2. To operate this unit a light source is provided directly over the reader (it uses

Fig. 8-1. Paper tape reader and punch on a model ASR-33 TTY.

photo-transistors to sense the holes in the tape) and the tape is pulled through the reader. It can read at high speeds.

A typical manually operated paper tape reader circuit is shown in Fig. 8-3. It uses a nine-photo-transistor sensing array to detect the 8 data holes and sprocket feed hole. The output of each sensor is fed to one-shot ICs to lengthen the pulse. A D flip-flop provides a *read data available* (RDA) signal to the CPU. This signal is derived from the sprocket-hole signal. The CPU's acknowledge (ACK) signal controls the RDA flip-flop so that no RDA signal will be provided unless an ACK signal has been provided by the CPU. The reader's output is connected to a parallel input port.

Paper tape can be corrected or edited with correction seals and splicing patches. A tape can be prepared or copied, off line, on a TTY without use of a computer. Also, the tape can be fan-folded and stored easily in a folder or book.

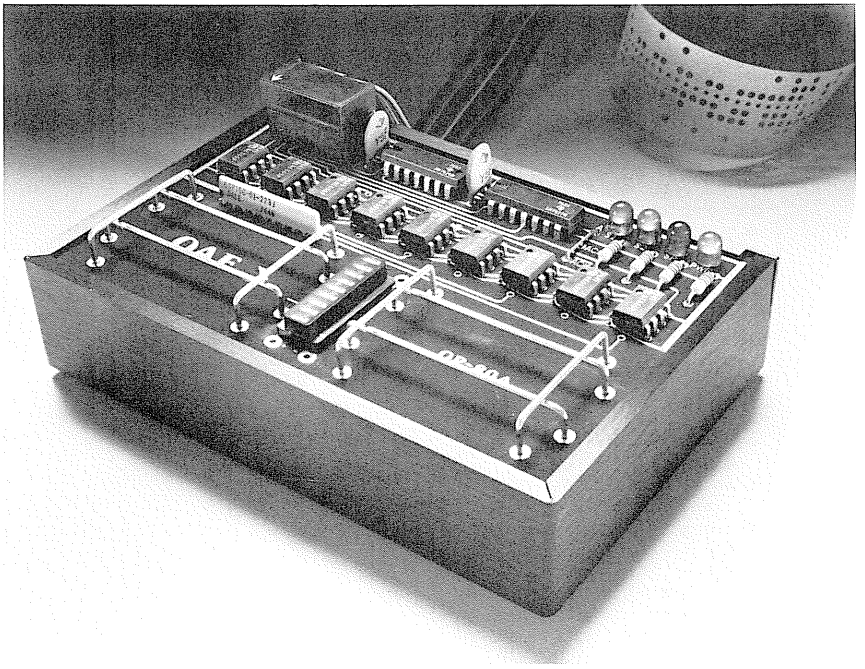


Fig. 8-2. Economical hand-operated paper tape reader

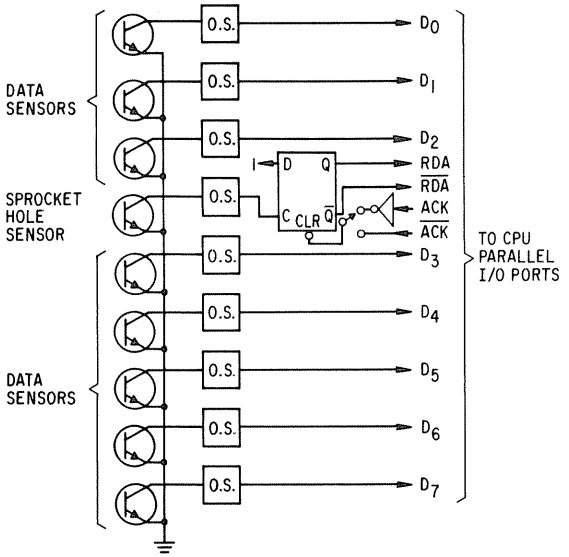


Fig. 8-3. Manual paper tape reader—schematic diagram. (Courtesy Oliver Audio Eng.)

The tape is 1 in. wide, usually oiled, and available in either 950-ft rolls or fan-folded 1,000-ft lengths. These will hold about 100,000 bytes of data. A typical tape punched with the standard 7-hole code is shown in Fig. 8-4.

The data on the tape is either *formatted* or *unformatted*. Unformatted tape contains only the program code characters. A formatted tape is structured in

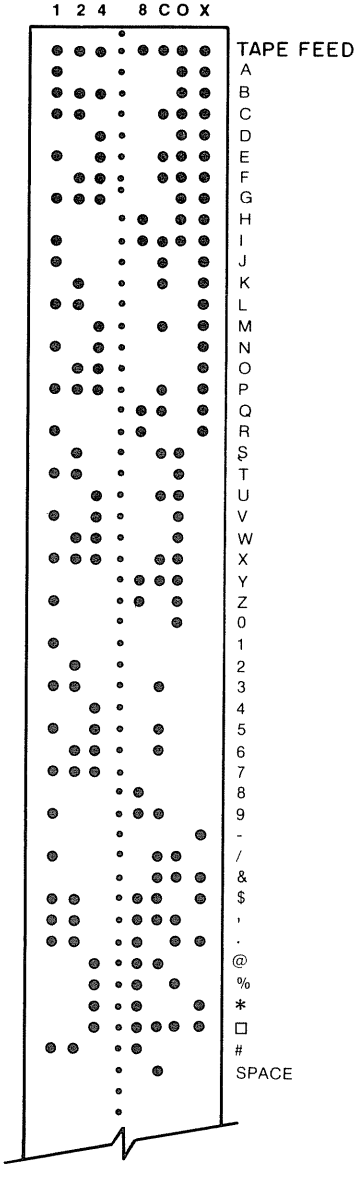


Fig. 8-4. Punched paper tape with the 7 hole code.

either a binary or hex format which provides a means of loading into memory with error checking.

For example, the Motorola 6800 (Mikbug) system employs a formatted binary tape (Fig. 8-5). The code is punched in blocks. Three different types of blocks can be recorded: header (program), data, and end-of-file. Each block starts with a hex 53 (ASCII S), followed by a hex character to indicate the type of record: header, data, or end-of-file. This is followed by a 2-binary-character byte count and a 4-binary-character loading address (tells CPU where in memory to load this block). Next the actual code to be loaded is given. The block is completed with a 2-binary-character checksum.

Using a formatted tape enables loading to be started and stopped and errors to be caught and easily and quickly corrected. For example, if an error is detected during loading, the tape can be stopped, backed up, and loading resumed. It is not necessary to load the tape from the beginning since each block has its own loading address.

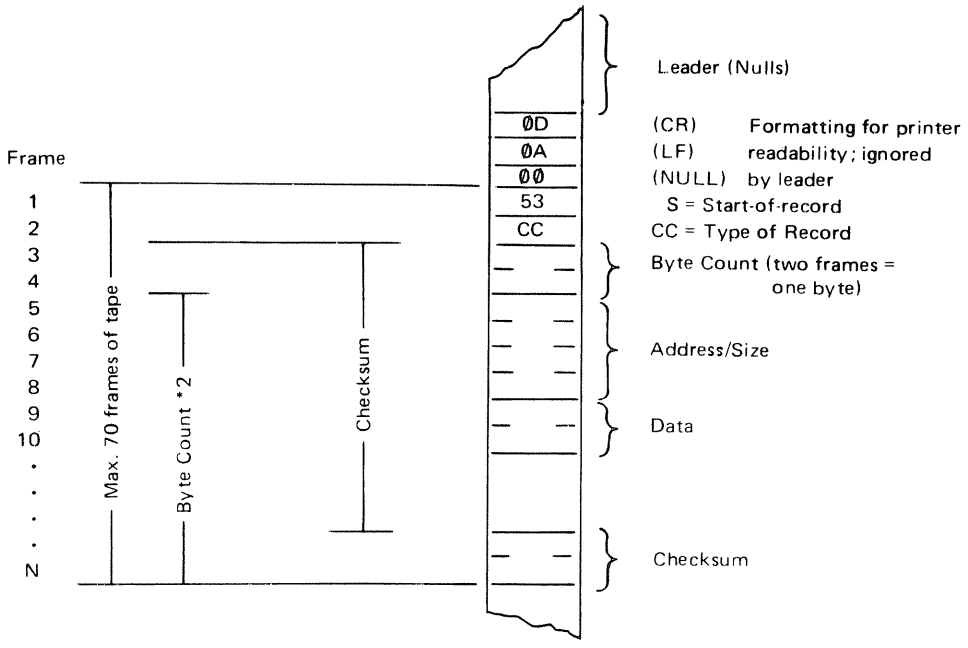
Paper tape punches, outside of the punch included on a TTY, are very expensive pieces of equipment and are generally beyond the pocketbook of a home user.

TAPE CASSETTE

The use of an ordinary audio type cassette recorder is very popular with personal system users. At a low cost it affords an easy to use, high speed, and high density mass storage system. The disadvantage is that no standard recording technique has been adopted. This, together with varying recorder characteristics, makes exchange of tapes much more difficult than with paper tape.

The data is most usually recorded on the tape in serial form as an *FSK* (*frequency shift keying*) modulated audio tone. Other recording techniques are used, but this is the most popular. This means that the recorder requires a serial interface to the CPU. The most popular recording system in use is the Kansas City standard (FSK) recorded at 300 baud. Using the KC standard about 100,000 bytes can be put on a C-30 tape cassette and the loading of an 8K program (formatted) would take about 5 minutes. This is three times the speed of a TTY using paper tape.

Another popular recording technique, developed by Don Tarbell, is the *Tarbell* standard. It uses a phase encoding technique and can be operated at speeds up to 1,800 baud. This reduces the loading time of an 8K program to under 1 minute. A third system, made by National Multiplex, uses a direct recording technique called *NRZ* (*nonreturn-to-zero*). This permits recording at rates up to 9,600 baud and can load an 8K program in less than 10 seconds. The disadvantage is that a special cassette recorder and cassettes are required, and thus the cost increases.



Frames 3 through N are hexadecimal digits (in 7-bit ASCII) which are converted to BCD. Two BCD digits are combined to make one 8-bit byte.

The checksum is the one's complement of the summation of 8-bit bytes.

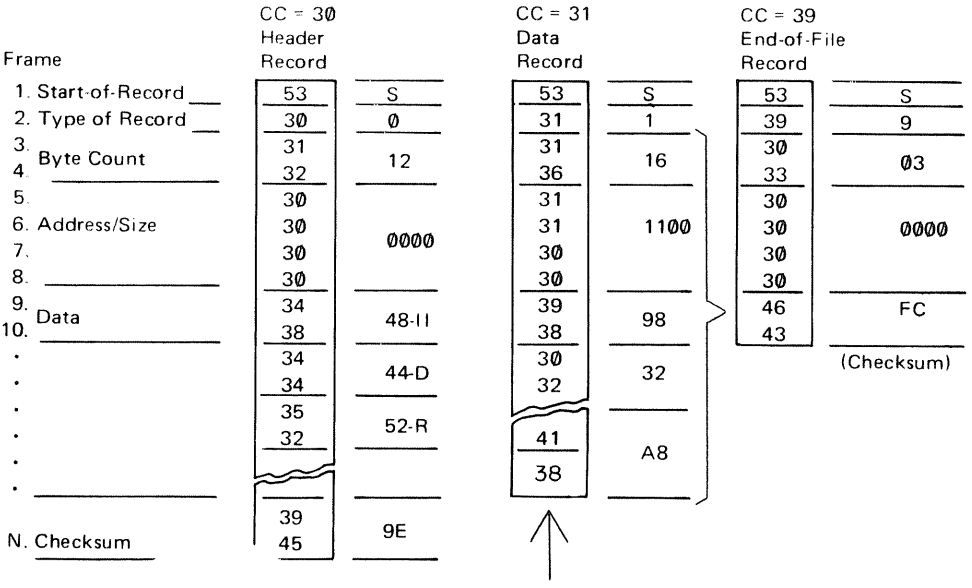


Fig. 8-5. Typical paper tape format. (Courtesy Motorola)

The Kansas City Standard

In November 1975, when personal computing was still very young, a number of manufacturers met in Kansas City, Kansas, and adopted a standard for exchange of programs on audio type cassettes. (The details of the KC standard can be found in the February 1976 issue of *Byte* magazine.)

The KC standard basically required the data to be recorded serially, using a standard UART format (1 start bit, 8 data bits, and 2 stop bits) at 300 baud. A logic-1 = 2,400-Hz sine-wave tone and a logic-0 = 1,200-Hz sine-wave tone; the clock pulses, are recorded on the tape with the data. The signal is then read from the tape and converted into a self-clocking signal that can tolerate 30% recorder speed variations.

A popular cassette interface unit using the KC standard is the SWTP model AC-30 shown in Fig. 8-6; a partial schematic is diagrammed in Fig. 8-7. It consists of an FSK modulator and demodulator. A 4,800-Hz clock is fed to the modulator which can be turned on or off by the *carrier enable* input controlling IC-5B. IC-5B divides the signal by 2 down to 2,400 Hz. IC-5A divides the clock signal by 2, reducing it to 1,200 Hz if data =1 or leaving it at 2,400 Hz if data =0. IC-4A is an active filter to reduce the harmonics of the signal so that it approximates a sine wave.

Fig. 8-6. A KC standard type tape cassette recording system.

The signal from the recorder is passed through a high-pass filter circuit to comparator IC-4B, which introduces hysteresis (delay) and eliminates false triggering. IC-3C/D generates a pulse every time the comparator changes states. Q2 and IC2B detect whenever several cycles of audio carrier are missing, and Q1

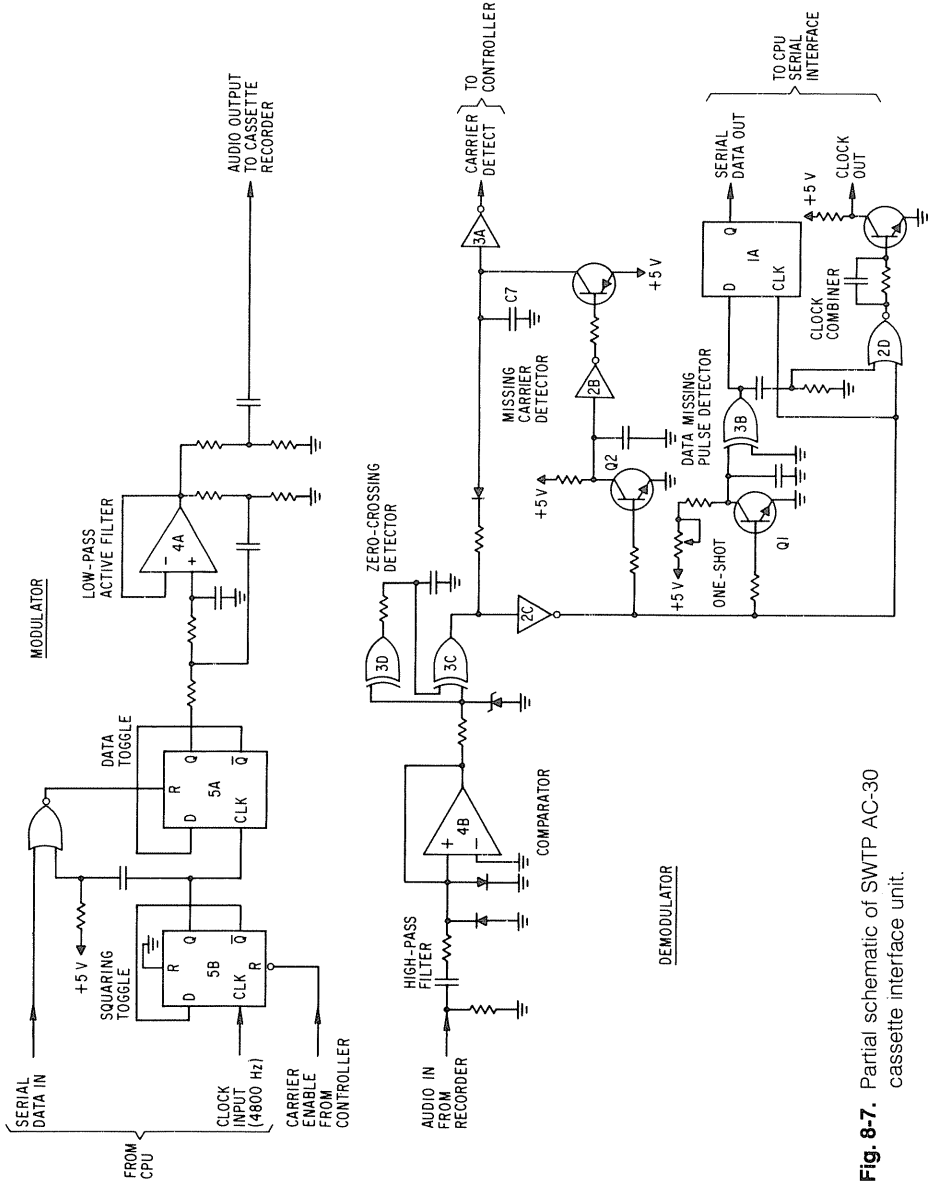


Fig. 8-7. Partial schematic of SWTP AC-30 cassette interface unit.

and IC-3B detect whether any data pulse is missing (times out when 1,200-Hz data exists). IC-2D synthesizes the clock pulse and IC-1A detects the logic levels of the data.

The AC-30 includes a basic controller circuit that permits starting and stopping of one or two recorders automatically. The unit is designed to work with the SWTP-6800 system.

The Tarbell System

The Tarbell system is designed to interface directly to the S-100 bus. It uses a sophisticated phase-encoding technique (used in many large scale computers) to encode the data on most standard audio cassette recorders. It can operate at up to 1,800 baud and, hence, is considerably faster than the KC standard. The interface unit is available on one PCB that plugs directly into any S/100 bus type CPU. In addition, it contains control circuitry to start and stop the recorder.

Cassette Recorders

The KC and Tarbell circuits will operate with most standard audio cassette recorders. However, for optimum, reliable performance, the cassette recorder should have a good high-end frequency response (e.g., 8,000 Hz), minimum speed fluctuation, and a good tape handling system to minimize tape wear. For example, Don Tarbell recommends a J.C.Penny recorder that costs about $40. The unit should also have a control input to start and stop the motor so that tape

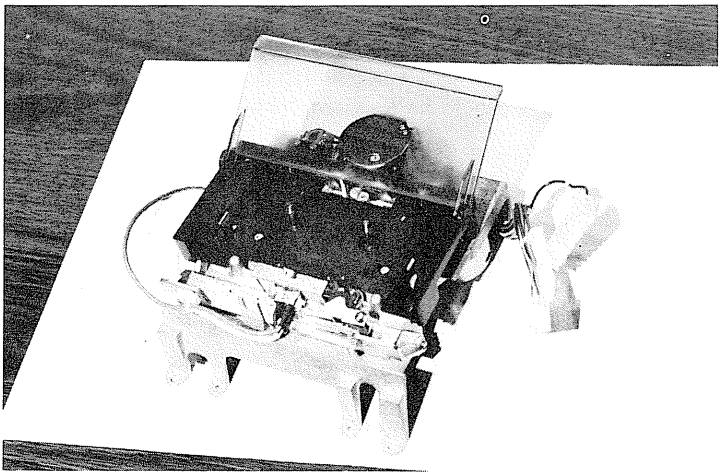


Fig. 8-8. A controllable cassette recorder. (Courtesy Economy Co.)

can be controlled via software. If planning to operate at high baud rates, it is recommended that a good quality recorder be used.

A fully controllable recorder, such as the popular Phi-deck unit (Fig. 8-8), is needed if data processing is to be done. This will allow block data to be read into the CPU. When used in conjunction with a sophisticated controller, and operating system software, it is possible to search the tape for blocks of data, update the data and hence handle applications such as mailing list maintenance, inventory control, etc. In fact, a system using multiple controllable recorders can be made to operate as well as a disc system but at lower speeds.

FLOPPY DISCS

The disadvantage of cassette tape storage is that of slow access time. The fast audio cassette units will take approximately 1 minute to load an 8K program, and added to this is the time to find the program; this can take many minutes. Controllable digital type recorders can search at high speeds and load quickly, reducing the search and load process to a matter of no more than 1 minute or so. To overcome this slow access time, the disc storage system was developed. However, it has the disadvantages of high cost and complexity. The recent introduction of *floppy discs* has reduced this significantly, but it is still considerably more costly than cassette storage.

The floppy disc permits having an immediately accessible program library, since a standard floppy can store up to 256K bytes and access any block of program in less than 1 second. Recently, a double-density floppy disc system has been introduced.

One of the biggest uses of disc storage is for information storage and retrieval. For example, a business may keep customer records, for direct access, on a disc. If each customer's data was 300 bytes and there were 800 customers, 240,000 bytes of storage would be required. This is easily handled on a floppy disc system.

The storage medium is a large round piece of Mylar 0.003 in. thick, covered with a thin layer of magnetic oxides. It is housed in an 8-in. square protective envelope (Fig. 8-9) with cutouts for the drive spindle, recording head, and index position hole. Two types of discs are used: a soft-sectored and a hard-sectored type. The soft-sectored, which is the standard, uses a single index hole and sectoring format information pre-recorded on the disc. The hard-sectored disc (Fig. 8-10) contains an index hole and holes to define the 32 sectors on the disc.

A typical floppy disc drive unit is shown in Fig. 8-11, and its internal construction in Fig. 8-12. The disc is inserted into the drive through a door. When the door closes, the disc is moved against a cone shaped spindle which pokes through the large center hole and clamps the disc. The spindle is driven at a

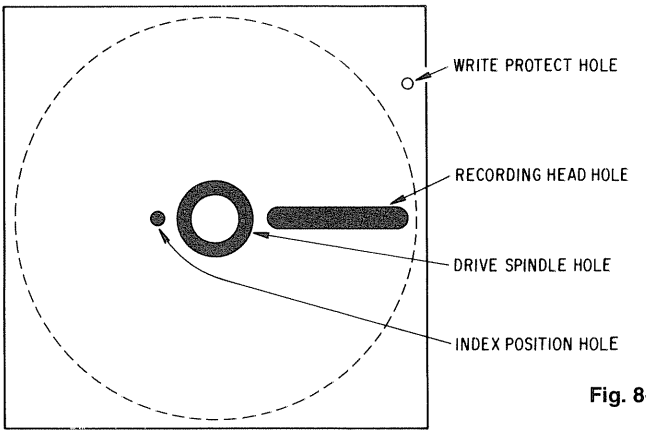


Fig. 8-9. Typical floppy disc in its envelope.

constant speed of 360 rpm by a synchronous motor and the disc rotates inside the felt-lined envelope.

The head is on a carriage, which moves radially in or out from the center of the disc by a stepping motor. The head can be positioned at any one of 77 points (tracks), with track 0 being the outermost track. The stepping ranges from 100 to 400 steps per second.

The soft-sectored disc contains 26 sectors and 73 tracks of data. Each sector contains 128 bytes. Hence, a total of 242,944 bytes can be stored on the disc. The hard-sectored disc contains 32 sectors (128 bytes per sector) and 77 tracks and thus 315,392 bytes can be stored.

The recording head protrudes through the slot in the envelope. When reading or writing occurs a pressure pad on the other side of the disc presses the

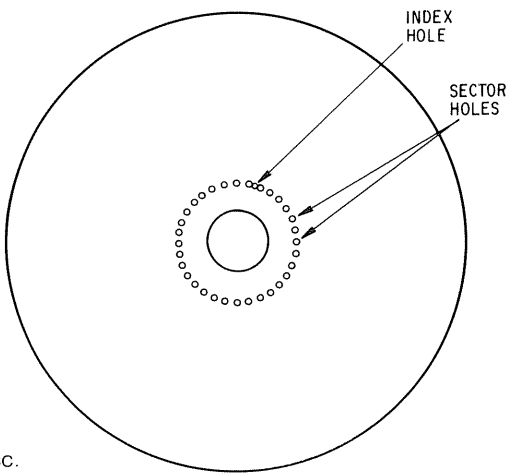


Fig. 8-10. Hard sectored disc.

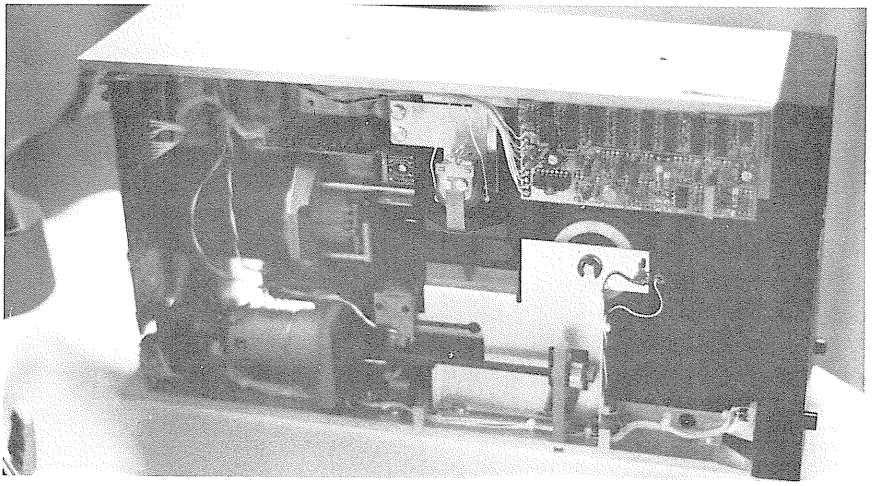


Fig. 8-11. A typical floppy disc drive unit.

disc against the head. A photo cell and lamp detect the index and sector holes. The record format on the disc consists of some leading zeros, a data ID pattern, the data (128 bytes), CRC characters, and some trailing zeros. The *CRC* (*cyclic redundancy check*) characters are used for error detection (to be explained later). The ID contains the track and sector addresses so that if the drive were to stop in the wrong position the controller could correct the error.

The disc unit contains the drive, controller, and interface electronics. They may be on one PCB, but are generally placed on separate PCBs to allow mating of different drives to controllers. The single PCB approach limits the user to the specific drive. The drive electronics contains the read, write, motor control, head positioning, and loading circuitry. The controller recognizes the CPU commands and data and provides the proper signals to the drive circuitry. Step counts are computed, sectors counted and recognized, data IDs recognized, CRCs computed and checked, data serialized and deserialized, and memory addresses counted. Also, circuitry for handling multiple drives may be included. The controller may contain as many as 200 ICs or use an MPU.

The interface circuitry provides for interfacing to two parallel CPU output ports and two or three input ports. Some of the faster units employ direct memory access (DMA, see Fig. 6-10) to read and write to the CPU's memory directly. Transfer rates of 30,000 bytes per second are typical.

Crucial to the optimal operation of the system is the supporting software to manage the operations. The software for this is referred to as *FDOS* (*floppy disc operating system*). FDOS keeps track of what is stored on the disc, and permits the user to call programs or data by name. A good FDOS will also enable the user to move programs from device to device. This is important when using an assembler or high level languages such as BASIC.

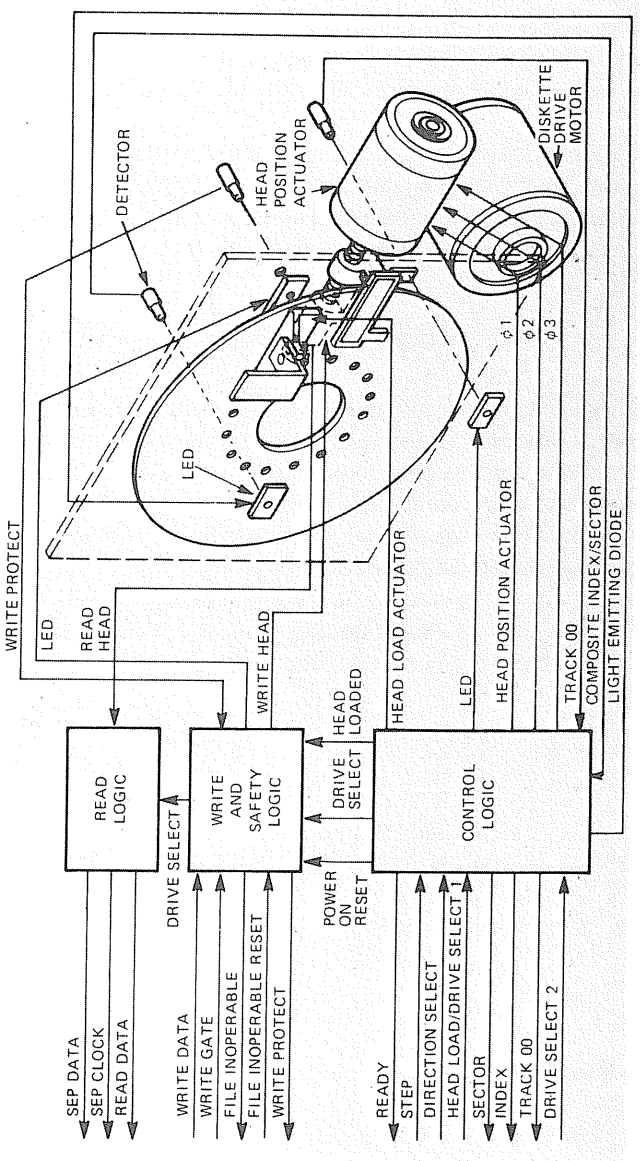


Fig. 8-12. Construction of a floppy disc drive. (Courtesy Shugart)

BUBBLE MEMORY

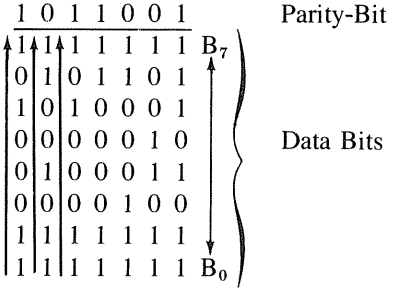
A new mass storage device is presently in the initial stages of reaching the personal computer market. It is *bubble memory*. It promises to be stiff competition for the floppy disc system. It is an all electronic system, compared to the electromechanical, motor-driven floppy disc system. It should prove to be more compact, require less power, be faster, quieter, and in the long run more economical.

Bubble memories consist of a thin orthoferrite film in which magnetic domains (bubbles) carry the digital data. Millions of bubbles can be contained in a square inch, and the memory contained in an IC package. It promises to be the next technological breakthrough in the computer field.

ERROR-CHECKING

When transferring data from one device to another, it is possible to introduce errors, e.g., a dropped bit. Hence, error-checking is performed as part of a data transfer to ensure that it is error free, relatively speaking. Three techniques are in wide use in personal computing systems. They are as follows:

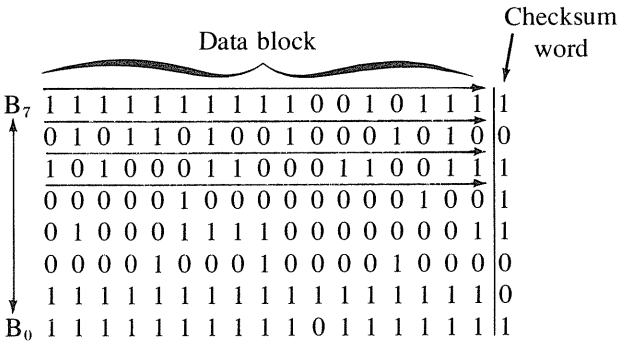
The Parity-Bit: This consists of the addition of a *parity-bit* to the data word. Both *even* and *odd parity* are employed. In odd parity a 1 is added to the word if an even number of 1s is in the data word, and a 0 is added if an odd number of 1s is in the word. This system is widely used in serial data transmission between a CPU and a terminal. Here is an example (odd parity); note that it uses a vertical redundancy check.



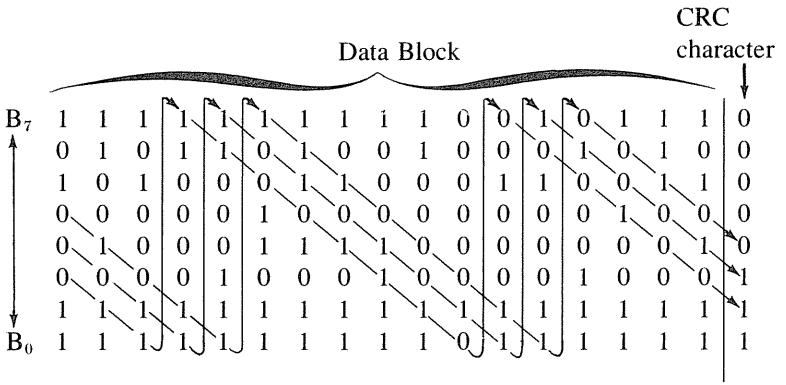
The Checksum: Here a block of data is checked horizontally for the number of 1s and 0s, and a data word is added to a block of data bytes. This technique is widely used when storing data on paper or cassette tape.

In the following example there are 17 data words and an 18th word (the checksum) is added to the block of data. The checksum word is derived by summing the 1s horizontally. A 1 is put in the appropriate bit position of the checksum word if there is an even number of 1s counted. Note that the format

used by Motorola (shown in Fig. 8-5) is based on summing hex digits and expressing the checksum word in the 1's complement.



The Cyclical Redundancy Check (CRC): This error check is much more rigorous than the preceding two. In addition it requires more circuitry to be accomplished. It is widely used in disc storage memory systems. It involves checking the data bits diagonally. Actually, it is an arithmetic operation performed on the data in a wraparound fashion. The check is performed as follows:



9.

Input/Output Devices

Input/output devices, often called peripherals, are the means by which the user communicates with the CPU. The most widely used I/O devices among personal systems users are the Teletype (TTY) and TV display/keyboard. CRT-type terminals and printers are also used frequently, but because of higher prices they are not as popular.

Terminals are usually divided into two classifications, the *hardcopy* and *softcopy* terminals. A hardcopy terminal is one which types characters on paper and, hence, leaves a permanent *hard copy* of the computer output. A softcopy terminal usually uses a TV display for output. Once the computer output to the TV screen passes from the screen, it is lost for all time and hence the designation *softcopy*.

TELETYPES

Teletypes, most affectionately called *TTYs,* predate the introduction of computers by more than 30 years. These machines were first produced for the transmission of messages via telephone lines. Their ready availability, when computers were born, made them a natural for computer I/O use.

Teletype is a trade name for teletypewriters manufactured by the Teletype Corp., Skokie, Illinois, the leading producer of these machines. A number of other manufacturers have made competitive machines but most of them have dropped out of the business.

Most personal computer users purchase their TTYs used, since there are large numbers of these machines available on the surplus market. The earliest models (11, 12, and 14) were manufactured in the 1920s and many are still in use, even though they could be classified as antiques. The models 15, 19, 28, and 32 followed the earlier models and some are still being manufactured. They are widely available used, for prices ranging from $25 to $400 depending on condition. They use the 5-level Baudot code and require the addition of interfac-

ing circuits. (Two excellent articles on interfacing these machines will be found in *Byte* magazine, April and May 1977, and *The Computer Hobbyist,* December 1974).

The models 33, 35, and 40 are ASCII-coded machines which can be directly interfaced to most CPUs, via 20-mA loop circuits with no modifications required. The model 33 (Fig. 9-1) is the most popular and is normally the machine referred to when one speaks of a TTY. It is still in current manufacture and more than 600,000 have reportedly been built. The machine is available in the following forms:

RO: Receive Only—consists of only a printer.
KSR: Send/Receive—includes a keyboard and a printer.
ASR: Automatic Send/Receive—includes keyboard, printer, tape reader, and punch.

The ASR-33 sells new for approximately $1,100 and used for $300–$900, depending on condition. It is the most popular hardcopy terminal in use by personal computer users.

The model 33 and its relatives, the 35 and 40, use the 8-level ASCII code shown in Fig. 8-4. Most personal computer users consider this tape format a standard for exchange of software. Also, most CPU serial I/O interfaces include the 20-mA current loop drive required by these machines.

The ASR-33 consists of five basic components within one enclosure (Fig. 9-2 and Fig. 9-3). The keyboard is the sending component of the TTY. Each of its keys controls a set of levers which position a set of electrical contacts to set up a parallel code (ASCII) for the character. This parallel output is fed to a motor driven distributor in the printer unit, which serializes the coded word and sends it to a selector magnet drive circuit in the call control unit from which it is sent to the CPU.

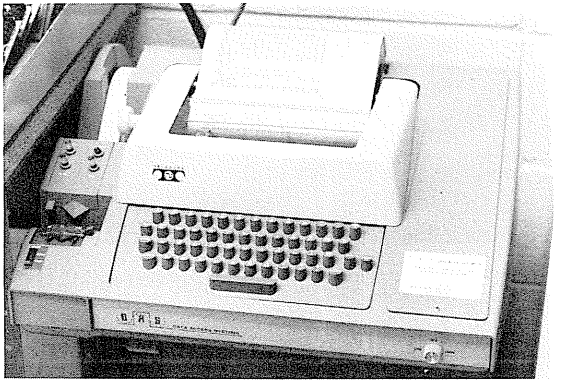


Fig. 9-1. The model ASR-33 TTY.

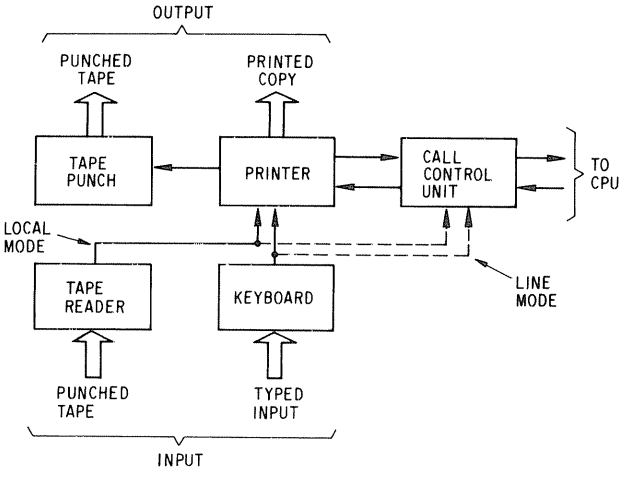


Fig. 9-2. Block diagram of a TTY (ASR).

Fig. 9-3. Interior view of ASR-33 TTY.

The printer receives the serial word from the call control interface unit and translates the signal into a mechanical arrangement of codebars. The positions of these codebars determine the position of the type wheel, on which characters are embossed, and the selection of functions such as *carriage return* (CR) and *line*

feed. A motor coupled to a main shaft drives all the printer functions. A standard friction feed (pin-fed option is available) advances the paper. The printer operates at 10 cps.

The tape reader has sensing pins which are driven upward for every cycle. Holes in the tape cause the sensing pins to close or not close a set of contacts which set up the parallel coded word. This is sent to the call control unit in the same manner as the keyboard input.

The punch is mechanically slaved off the printer codebars to set up the punch pins. The paper tape is advanced and punch pins are driven from the main shaft of the printer drive.

Model 33 comes supplied with an excellent set of maintenance manuals so that interested users can easily maintain their own machines. Replacement parts are readily available direct from Teletype and from numerous TTY repair shops around the country.

The most widely used model is the ASR-33, number 3320, 3JA. Also popular is the 5JA version with automatic tape reader (reader can be started and stopped under program control). A 3JA can be easily converted to a 5JA.

The ASR-33 operates at 10 cps with an 11 bit word (110 baud). The TTY's transmitted word consists of 1 start bit, 2 stop bits, and 8 data bits (8th bit is an even parity-bit).

TELETYPEWRITERS

TTYs are slow (10 cps), noisy, heavy, and yield only upper-case type. Where high quality typewritten copy is needed (e.g., word processing), typewriter type terminals can be used; they can operate at speeds of 15–45 cps.

It is possible to convert an office type IBM Selectric typewriter to serve as a printer output. It is usually not worthwhile to convert the keyboard for input. The conversion requires the addition of 5 solenoids to push the necessary levers in the Selectric mechanism. The conversion has been detailed in a number of computer hobbyist publications. However, be warned that the job is not easy. The IBM Selectric operates at 15 cps.

Many terminals employing the Selectric mechanism are available on the surplus market at prices ranging from $100 to $900. A typical surplus unit is shown in Fig. 9-4. These machines include the controls and often, but not always, the interfaces. The cheapest are the models 73, 731, 735, etc., which range in price from $250 to $900. These machines also require interfacing modifications, but they are considerably easier to interface than are the office Selectric (a typical conversion is described in *Byte* magazine, June 1977).

It should be pointed out that these machines do not use the standard ASCII code and, hence, code conversion software in the CPU is required. IBM uses either the *Correspondence* or *BCD codes*. The correspondence code permits selection of all the Selectric printable characters. However, the BCD code permits selection of only 48 characters and, hence, is limited to upper case only.

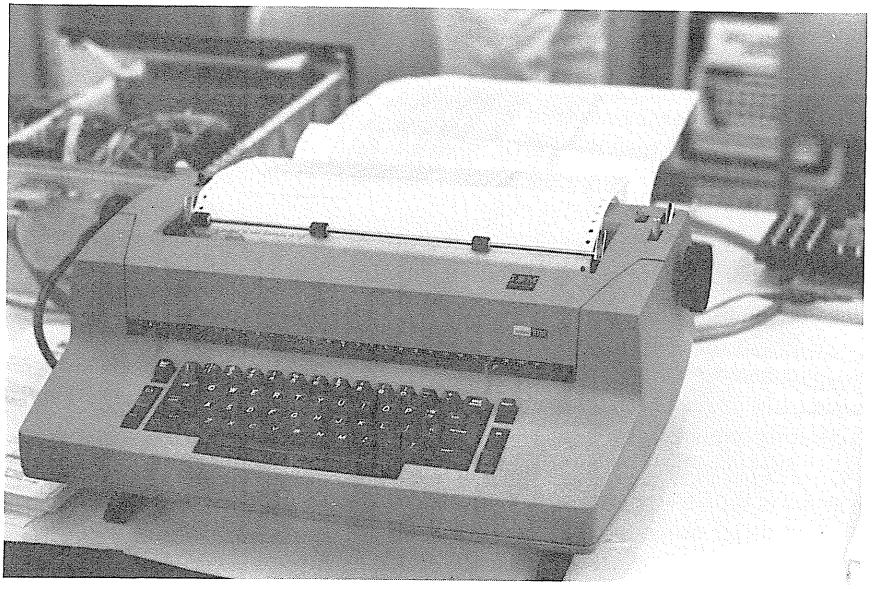


Fig. 9-4. A surplus terminal using an IBM Selectric printer mechanism.

Regretfully, most of the Selectrics available on the surplus market are BCD machines. It is possible to convert a BCD to a correspondence coded machine—refer to a *Byte* article June 1977.

For those not wishing to go through the conversion-interfacing problems of Selectrics, there are available hardcopy terminals which are ASCII-coded and include either RS-232 or 20-mA loop interfaces. They are, however, considerably more expensive. One can find terminals using Diablo and Queme printing mechanisms on the surplus market; their prices start at about $1,200. These machines are multispeed machines working at speeds of 10, 15, or 30 cps. Often the print mechanism alone is available. A typical unit is shown in Fig. 9-5. The newer versions of these printer mechanisms operate at 45 and 60 cps, but units will be extremely rare on the surplus scene for some time to come.

DOT MATRIX PRINTERS

In an attempt to increase the speed and reduce the cost, noise, and weight of typewriter type printers, a number of manufacturers have introduced dot matrix type printers of the thermal and impact type. A thermal printer employs a print head composed of an array of solid-state heating elements (actually transistors). A typical array is shown in Fig. 9-6. Each transistor is selectively energized so that the top surface becomes hot, producing a dot on the thermally

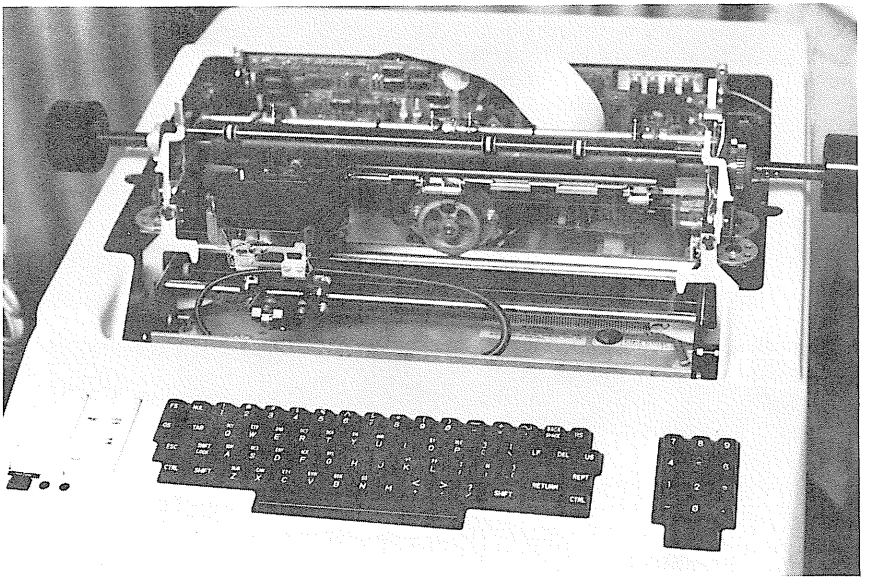


Fig. 9-5. A surplus printer using a Diablo printer mechanism.

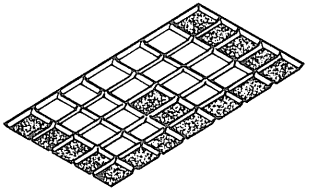
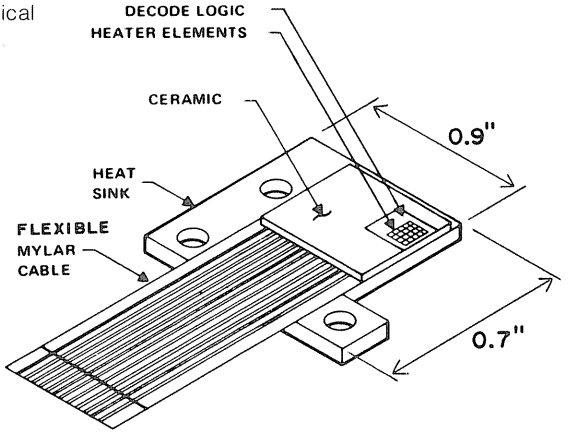


Fig. 9-6. Dot matrix print head energized to print E.

Fig. 9-7. Construction of a typical dot matrix print head.



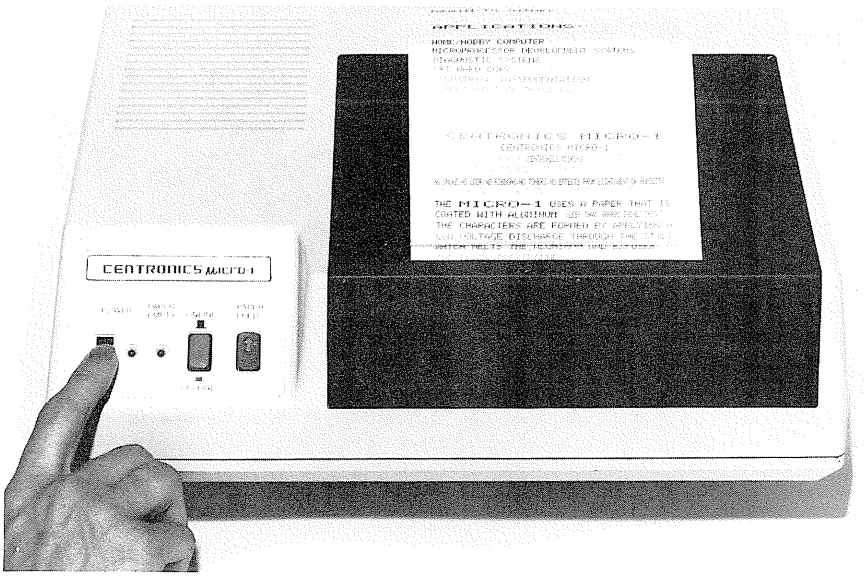


Fig. 9-8. Typical dot matrix type printer. (Courtesy Centronics)

sensitive paper pressed against it. Figure 9-6 illustrates the selection of elements in the array necessary to print the letter E (shaded). Note that special thermographic paper is required.

The print head (Fig. 9-7) usually includes the decoding and drive circuitry and, hence, the printer mechanism and electronics are simpler than an impact printer. Also, no ribbon is required.

An impact type dot-matrix printer uses solenoid actuate hammers that strike an inked ribbon against the paper. These printers operate at speeds up to 120 cps. A typical terminal using a dot matrix nonthermal printer is shown in Fig. 9-8. These units are often found on the surplus market starting at prices of $200 and up. New terminals start at about $400. Teletype recently introduced a terminal, model 43, using a nonthermal dot matrix printer which sells for approximately $1,100.

Note that the terminals described here are all KSR type machines. In other words, they do not include paper tape readers and punches. However, most operate at 10, 15, and 30 cps.

KEYBOARDS

Although often included in the same enclosure as the printer, a keyboard is usually electrically separate from the printer. A typical surplus keyboard is

Fig. 9-9. A typical surplus keyboard.

shown in Fig. 9-9. The keyboard generates a parallel ASCII-coded word (7 bits) and a keypressed pulse which may be sent to a parallel input port of the CPU or serialized by a UART and sent to the CPU.

A typical keyboard circuit is shown in Fig. 9-10. Most keyboards employ an encoder IC. This IC scans an array of crossing lines (X and Y matrix). The keyboard utilizes keyswitches at all the intersections of the matrix. In scanning the lines, the encoder enables one X and one Y line at a time. A pressed key causes a short between a given pair of X and Y lines. The keyboard encoder IC senses this contact closure and selects a code from a ROM in the IC. This code is placed on the seven output lines and a short pulse is generated as the keypressed output (sometimes called a *strobe*).

The scanning approach allows for more than one key to be pressed simultaneously and still be encoded properly as they are released. This is called *n-key rollover*. If no key is depressed, no keypressed pulse is generated.

THE TVT AND VDM

The lowest cost data output system from a CPU is the *TVT*, also known as a *TV typewriter*. This unit may also be called a *VDM* (*video display module*). The purpose is to display the alpha-numeric output on a TV screen in the same essential format as would appear on a TTY or teletypewriter. A typical TVT/VDM display is shown in Fig. 9-11.

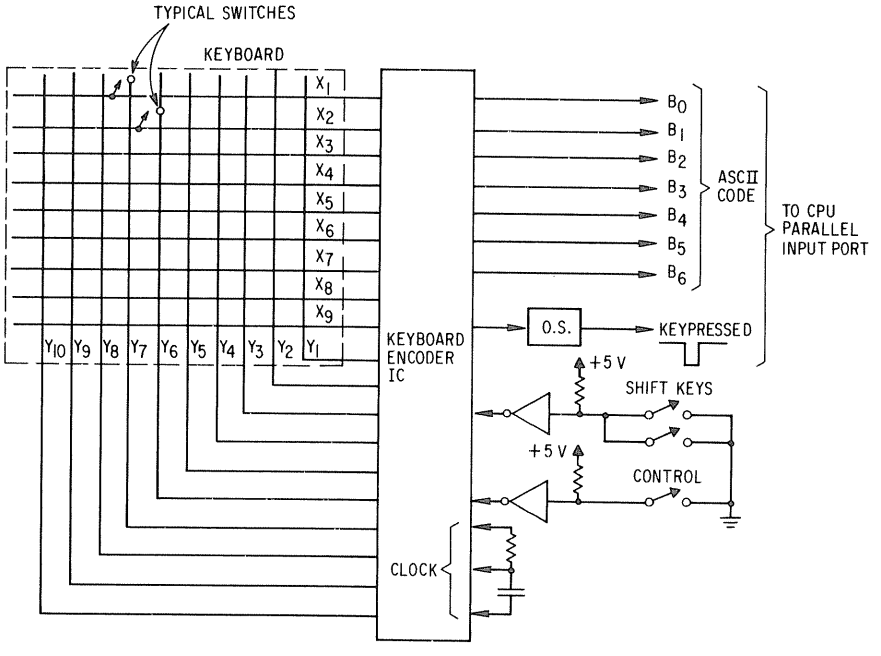


Fig. 9-10. Typical keyboard circuit.

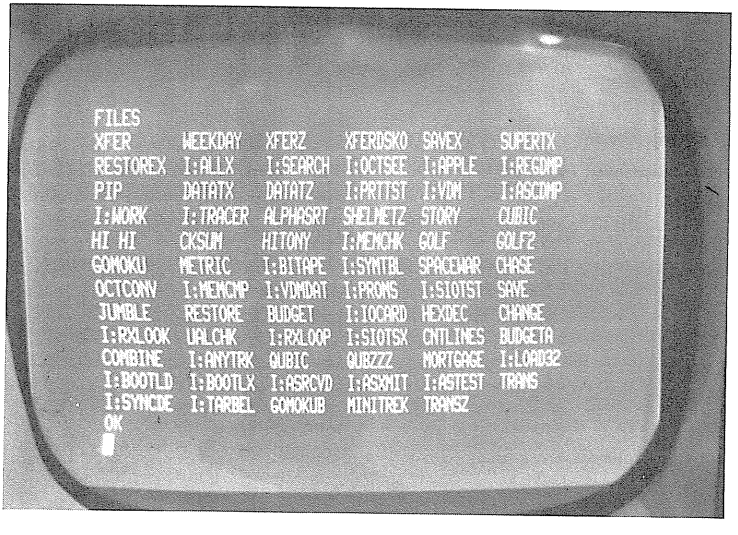


Fig. 9-11. A TVT/VDM display output.

The heart of the TVT/VDM is a ROM character generator (CG) IC. The IC is used to generate a dot matrix type character on the screen by outputting a horizontal line code. For example, the 2560 CG IC formats the 64-7×5 dot matrix characters shown in Fig. 9-12. The 2560 outputs a 5-bit line dot code when addressed by a 6-bit ASCII character code and a 3-bit line address code.

A typical TVT/VDM system block diagram is shown in Fig. 9-13. The unit is controlled by a *master clock* circuit which develops the horizontal and vertical sync pulses for the TV. The clock also drives a series of counters to derive the addressing for the CG and the RAM memory. The RAM stores the ASCII codes for every character displayed. The contents of the RAM are scanned by the address counters (A_0–A_8). The RAM output is then converted to a dot matrix output by the CG. The parallel CG output is serialized by a shift register and combined with the sync signals to form the composite video to drive the TV.

The data from the CPU is latched and then fed to the RAM. At the same time, a *cursor* circuit keeps tract of every character position on the screen. It permits changing character locations and moving of blocks of data (e.g., lines). The cursor may thus be under the control of the user or the CPU.

A typical TVT/VDM will display either 32 or 64 characters per line and typically 16 lines on a standard video monitor or modified TV. *Caution:* Transformerless type TV receivers should not be used as they can damage low-voltage CPU circuits. The displaying of 64 × 16 characters requires a TV display with good bandwidth characteristics (typically 6 MHz or better). TVT/VDM units vary in characteristics and features. For example, they may have as few as 6 and as many as 50 ICs. The fewer ICs employed, the more the TVT/VDM depends on the CPU to provide the hardware and software to store and control the display. This takes processing time and memory space away from the CPU. It also means that sophisticated driver software is required in the CPU.

CRT TERMINALS

A *CRT terminal* (Fig. 9-14) combines in one enclosure the display, TVT, keyboard, UART, and interface circuits. Hence, it connects to the CPU via an RS-232 or 20-mA loop circuit. A very popular CRT terminal is the ADM-3 (manufactured by Lear Siegler, Inc.) shown in Fig. 9-15. It displays up to 24 lines of 80 characters on a 12-in. TV screen. It has a 55-key keyboard and transmits and receives at baud rates from 75 up to 19,200.

The ADM-3 is known as a *dumb terminal*. This is because it is completely dependent upon the CPU to tell it what to do. An *intelligent* or *smart terminal* contains additional circuitry, often an MPU, to permit formatting and editing functions to be performed by the terminal. This reduces the programming load on the CPU. However, *smart terminals* are very expensive.

Used CRT terminals, usually the dumb type, are available on the surplus market. When purchasing a used CRT terminal, check to be sure that it uses the ASCII code and has either a standard 20-mA loop or RS-232 interface.

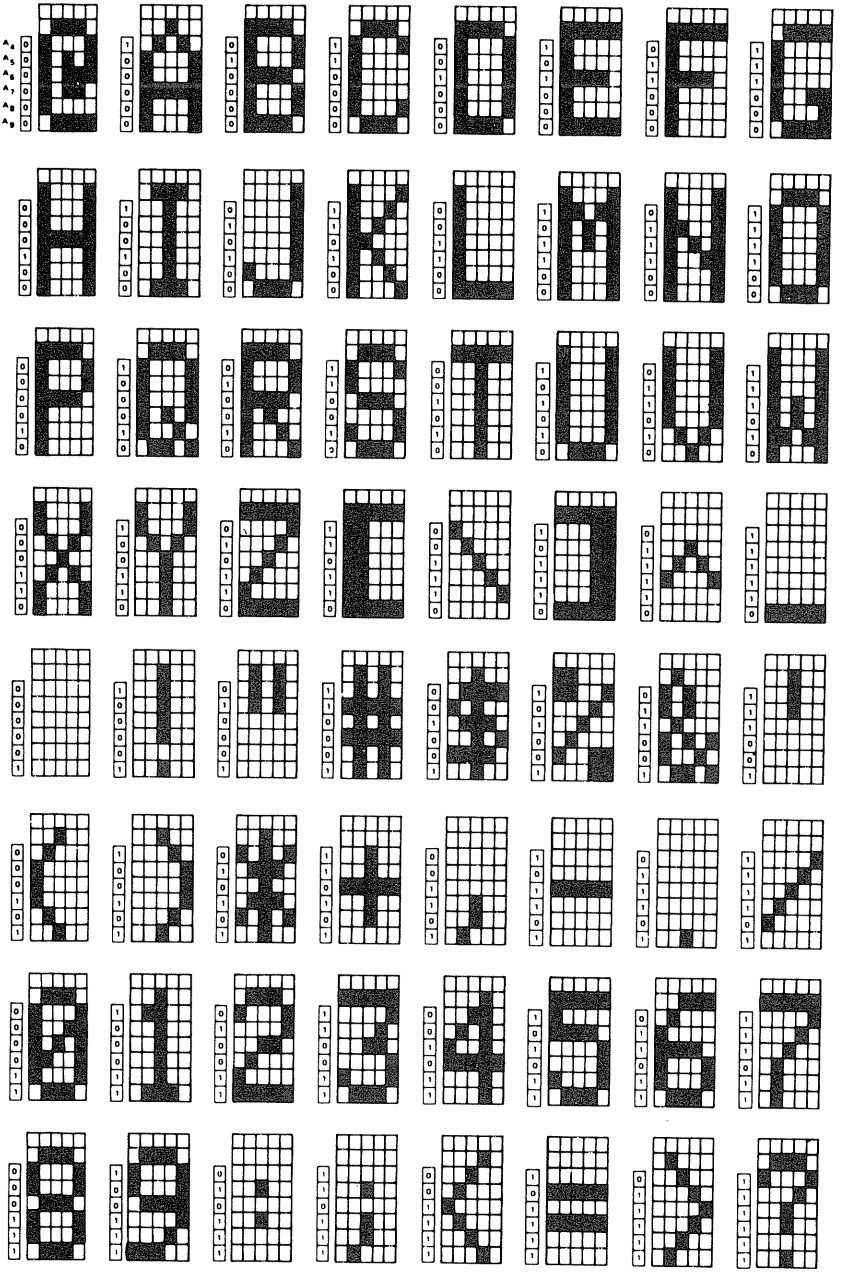


Fig. 9-12. Dot matrix format produced by the 2560 IC character generator. (Courtesy Signetics)

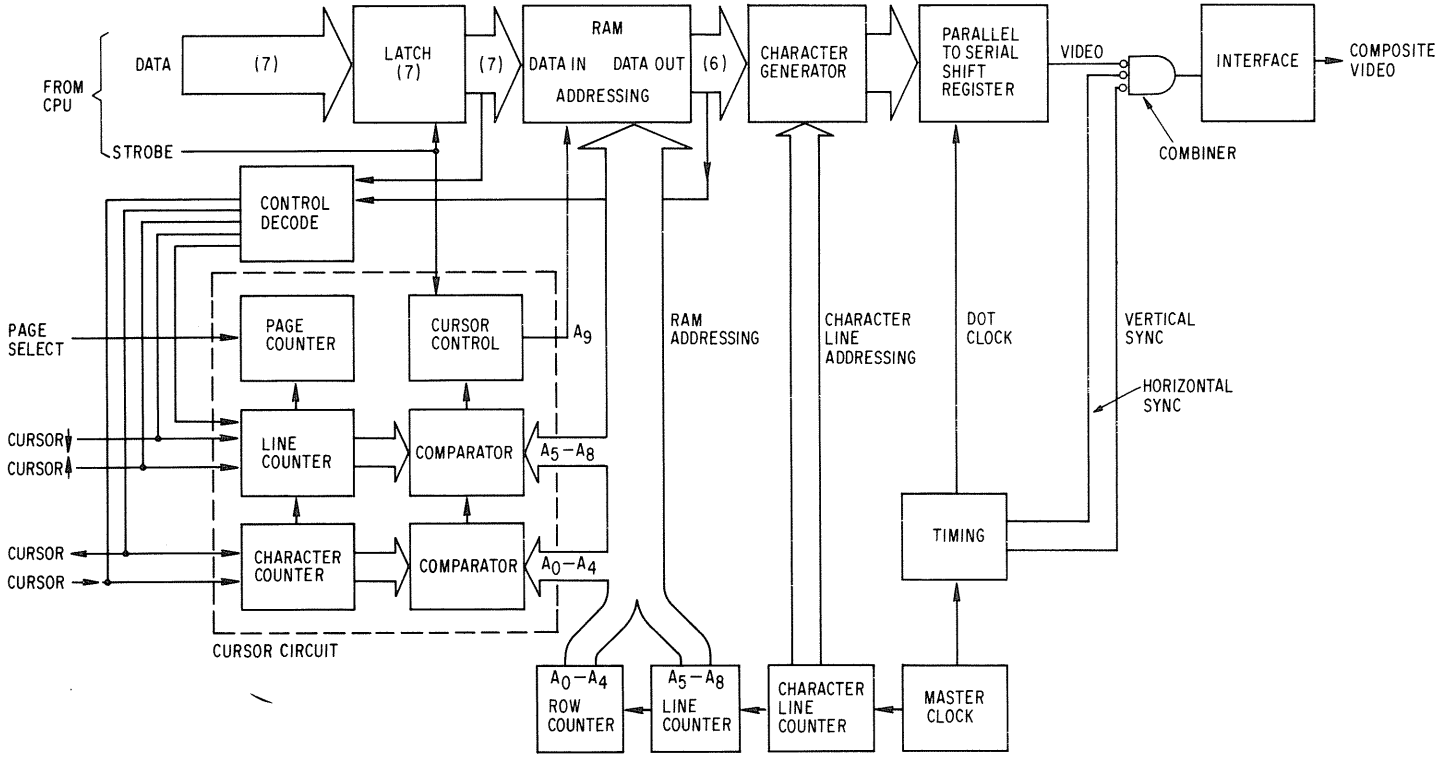


Fig. 9-13. Block diagram of a typical TVT/VDM unit.

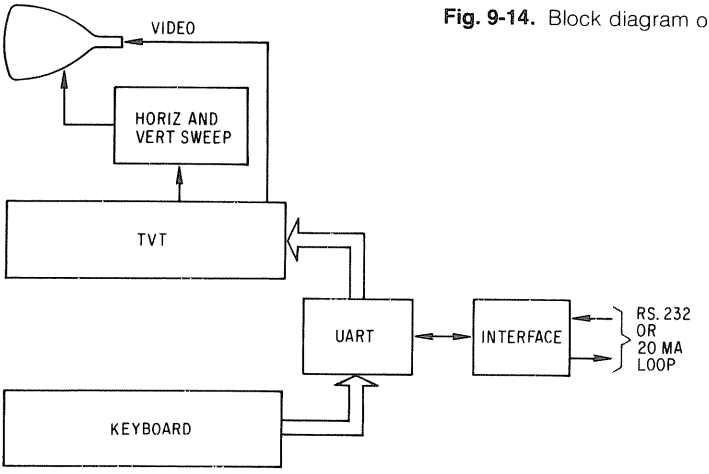


Fig. 9-14. Block diagram of a CRT terminal.

Fig. 9-15. The ADM-3 CRT terminal.

LINE PRINTERS

For those wishing hard copy output at a faster speed than that provided by teletypewriters, it is necessary to obtain a line printer. A typical line printer is shown in Fig. 9-16. These units are only output devices and are connected to the

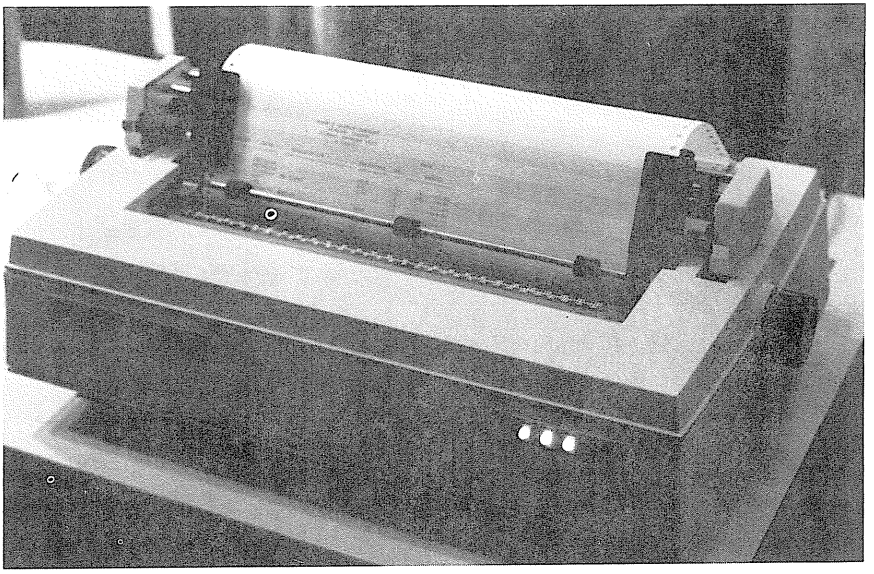


Fig. 9-16. An Oki-data line printer.

CPU via a serial (RS-232 or 20-mA loop) or parallel interface. These units print at speeds ranging from 45 cps on up. Most 45-cps printers employ a *daisy wheel* print head with each character on the *petals* of the head. Higher speed printers use a dot matrix type head made up of an array of solenoid operated pins which hit the ink ribbon against the paper. In order to save time line printers usually print in both directions. They may also calculate the most rapid way to print each line to avoid unnecessary carriage returns. Line printers also use special form paper with sprocket holes on both sides. A special tractor pin type feed mechanism is used to advance the paper at high speeds. Needless to say, line printers are considerably more expensive than character printers.

Recommended Further Reading

1. Donald Lancaster, *TV Typewriter Cookbook,* Howard W. Sams & Co., Inc., Indianapolis, Ind., 1976.

10.

Computer Software

WHAT IS SOFTWARE AND PROGRAMMING

Up until now we have been concerned with the computer's *hardware*. We will now turn our attention to the computer's *software*. Software refers to instructions to the CPU to *tell it what to do* and *how to do it*. It is also called *programming* and we tell the computer what to do by writing a *computer program,* placing it in the computer's memory, and then causing the computer to execute the program step by step.

When we use hardware to do a job we use gates, flip-flops, registers, and other IC logic elements. When we write programs we use instructions, subroutines, tables, and other standard software modules. A program can be defined as a means by which a computer is instructed to perform a given task. The program is written in a specific *language* which is designed to run on a specific computer.

Programming starts with a specific *algorithm* which turns the computer into an accountant, a game opponent, a process controller, and lots more. The algorithm is a precisely defined procedure which converts raw data input into processed data output.

PROGRAM CODES—FROM THE LOWEST TO HIGHEST LEVELS

We program a computer by entering a sequence of codes into the CPU's memory and then sequencing through these codes. The codes in turn cause the CPU to execute specific operations.

We may enter the code via a set of data and address switches and load the code into specific addresses of RAM. This method is called *machine level* programming and the codes are binary codes.

For example, the following machine level program, using 8080 instructions, adds two numbers:

Memory Address	*Instructions*
0000 0000	0011 1010
0000 0001	1000 0000
0000 0010	0000 0000
0000 0011	0100 0111
0000 0100	0011 1010
0000 0101	1000 0001
0000 0110	0000 0000
0000 0111	1000 0000
0000 1000	0011 0010
0000 1001	1000 0010
0000 1010	0000 0000
0000 1011	0111 0110

Notice that each instruction is shown as the 8-bit binary code that is stored at the memory addresses shown on the left. This is the program as the machine (CPU) works with it.

It is very difficult to write programs in binary code. It is very tedious and very prone to errors. Hence, programmers quickly devised simpler codes. The octal and hex codes, discussed in an earlier chapter, are both used; hex code is the preferred. Here is that same program written in hex code.

Memory Address	*Instructions*
00 00	3A
00 01	80
00 02	00
00 03	47
00 04	3A
00 05	81
00 06	00
00 07	80
00 08	32
00 09	82
00 0A	00
00 0B	76

As you can see, it is much easier to write and considerably less prone to errors. However, even hex is tedious and error prone for writing long programs. Therefore, programmers use hex coding only for short programs. Longer programs are written using special programs which are first loaded into the computer's memory. These programs are called an *editor* and an *assembler*. The editor permits the programmer to write the program in a *mnemonic code* which is close to the English instruction. The assembler can then translate the mnemonic

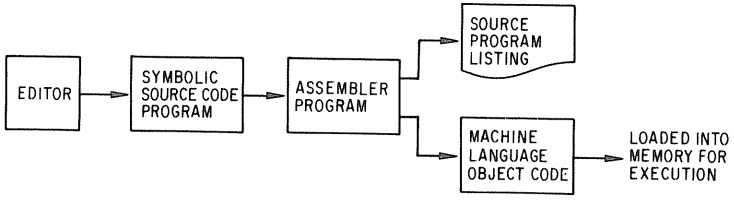


Fig. 10-1. Operation steps of an assembler program.

code into the machine level binary code for actual running on the computer or to produce a *listing* (print out) of the program (Fig. 10-1). Remember, the program in memory must be in binary form. Here is what the add program looks like in the mnemonic code understood by the assembler. Also note the comment statements which explain the operation of the program.

Instructions	*Comments*
LDA 0080H	Move contents of MA 0080_H to accumulator
MOV B,A	Move contents of accumulator to register B
LDA 0081H	Move contents of MA 0081_H to accumulator
ADD B	Add contents of register B to accumulator
STA 0082H	Store contents of accumulator in MA 0082_H
HLT	Halt

Observe that the mnemonic terms succinctly describe the machine operation to be performed. For example, LDA-0080 literally means *LoaD A register from memory address 0080.*

An assembler frees the programmer from the tedious mechanical details of machine-language programming. Besides freeing the programmer from the task of remembering all the machine codes, the assembler also keeps track of storage locations. *Labels* are used for *symbolic addressing,* and the assembler assigns a memory location to each label, where that label is defined.

For example, the label DELAY could be used to denote the memory address of a subroutine providing a time delay. Every time the instruction JMP DELAY (meaning "jump to delay") is given, the assembler assigns the MA where the delay subroutine is located. Thus if DELAY began at MA 0100, then the assembler would translate this instruction—"jump to MA 0100".

Symbols can also be used to define data constants. For example, the programmer can assign the symbol DATAOUT to port 10. When the program is assembled, all references to DATAOUT are assigned the instruction "data out to port 10".

Many computer systems have more powerful assembler programs, called *macroassemblers.* The macroassembler permits a single symbol to represent a group of machine instructions called a *macro instruction.* When a macroassem-

bler encounters such a symbol, it automatically inserts the proper group of instructions into the program.

Programming can be made even easier by using a language processor program that translates a more conversational language into machine code. Such programs are called *high-level languages*. The input to a language processor is called a *source program* and the translated program (actually run by the computer) is called an *object program*. The source program consists of statements that enable you to specify the program in a form you can easily understand. The object code is the machine language which the computer understands, and from which it executes the program.

High-level languages permit the programmer to use more natural instructions. For example, the high-level language called BASIC, which is the most popular with personal computerists, would require only a 1-line instruction to perform the addition operation shown previously in machine and assembler level forms. It would be the following:

```
10  LET C=A+B
```

Notice, that you now need not worry about instruction codes or memory locations. The language processor program takes care of all that. The disadvantage of this system is that the CPU's memory must be much larger to contain the language processor program as well as the instructions.

Higher level language processors are either *compilers* or *interpreters*. A compiler (Fig. 10-2) translates the entire source program at once to produce the entire object code. An interpreter (Fig. 10-3) translates each statement as it is encountered during program execution. Any high-level language can be im-

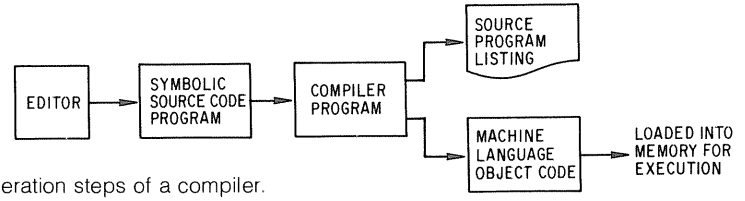


Fig. 10-2. Operation steps of a compiler.

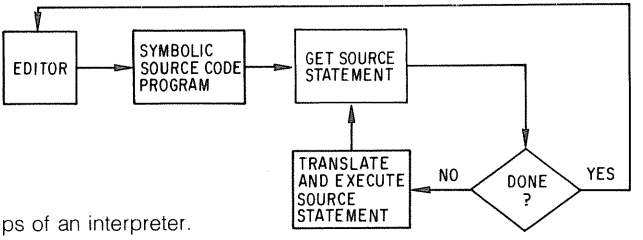


Fig. 10-3. Operation steps of an interpreter.

plemented as either a compiler or an interpreter. Compilers produce a more efficient code (program runs faster due to less program steps); however, they are harder to develop and require more memory storage areas. Interpreters have slower program execution time but require less memory and are easier to implement.

HIGH-LEVEL COMPUTER LANGUAGES

Many of the more popular high-level languages are available for microprocessor-based computer systems. Presently, the following are available:

BASIC (*Beginners All-purpose Symbolic Instruction Code*): BASIC is an algebraic programming language intended to make it easier for beginning programmers to use a computer. It is less extensive than FORTRAN, easier to learn, and has fewer error-prone factures. It is similar to FORTRAN but limited in scope.

FORTRAN (*FORmula TRANslator*): FORTRAN is used widely for mathematical type problems. It is very powerful in terms of handling formulas and mathematical procedures. Although most often used with IBM card input (*batch*) which is slow and tedious, there are some versions which use terminals (*interactive*).

COBOL (*Common Business Oriented Language*): COBOL is a procedure-oriented language designed for coding business data processing problems, i.e., those that use large files, a high volume of input and output, and production of reports requiring editing and formatting of output data.

FOCAL (*Formulating On-line Calculations in Algebraic Language*): FOCAL is an interactive language for performing mathematical calculations. It also has a powerful desk calculator operating mode. However, it is not as versatile as BASIC and very limited when compared to FORTRAN.

APL (*A Programming Language*): APL is a very powerful general purpose language which lends itself to complex mathematical programming. It employs conventional mathematical notation with a large set of unique symbols to form a kind of shorthand. Thus, it makes possible single line statements which process complex data manipulations.

In summary, machine level programs are more difficult to write, but take fewer steps, require a small fraction of memory, and run faster. In contrast, machine level programs cannot be transformed directly from one type of computer to another. Programs written in a high-level language are easier to understand and write, and can be run with little or no rewrite on all machines having the language process interpreter or compiler.

SYSTEMS SOFTWARE

The foregoing discussion covered the software used to execute a given application or solve a problem. Software is also needed to control the actual operation of the computer system. This type of software is called *systems software*. Actually, the editor program, discussed earlier, is a systems program. The following are the more common types of systems programs and their functions:

Monitor Programs: Also called *supervisors, executives,* and *operating systems* (*OS*), they enable the user to communicate with all of the system hardware and software. They allocate available resources as efficiently as possible, and range from simple single terminal monitors to complex multi-user, multi-task time-sharing systems.

I/O Driver Programs: They control the data transfer between the computer and its peripheral devices.

Data Management Programs: Also known as *file systems,* they enable the computer system to identify and organize individual blocks of data within the computer's memory. Few microprocessor-based computer systems have this software. Hence, the user must keep track of memory allocations.

Debugger Programs: Actually a type of monitor program, they are widely used to write machine level programs in hex or octal code and have features useful in tracking down errors in the program.

Simulator Programs: Help to evaluate a microprocessor's operation by simulating all the MPU's operations within the software of another computer. Hence a programmer may enter a program in 8080 code on an IBM-370 system and simulate the running of the 8080 program using the simulator program. Some microprocessor systems have resident simulators.

OTHER FORMS OF ASSEMBLERS AND DISASSEMBLERS

An assembler program is intended to translate a program written in mnemonic code into a machine coded program. The assembler program may run on the same system for which the software is intended. In this case the assembler program is called a *resident assembler*. When the program assembly is done on another computer, the program is called a *cross-assembler*. For example, cross-assemblers exist for assembling 6800 code on a DEC PDP-11 system.

Disassemblers do the opposite of assemblers. Given a machine-code program listing, they convert it into a mnemonic program listing. This is used for troubleshooting programs.

11.

The Computer's Instruction Set

MPUs differ in *architecture* and *instruction sets*. These are fixed in the MPU and the user must learn what they are and how to use them. An MPU's architecture consists of the functional blocks and the interaction between these blocks. In other words, it is hardware.

The computer's instruction set, on the other hand, is composed of the binary codes (octal or hex equivalents, can be used instead) which cause the MPU to execute specific operations.

In this chapter we discuss the more popular MPUs. In particular, we concentrate our attention on the most popular MPU, viz., the Intel 8080/8085.

MPU ARCHITECTURE

Figure 11-1 illustrates the architecture of the 8080/8085. It is characterized by a large number of registers within the MPU: accumulator, status, SP (stack pointer), PC (program counter), and six general purpose registers (B, C, D, E, H, and L). The Z-80, being an enhancement of the 8080, contains the same basic architecture with additional registers. The architecture of the Z-80 is shown in Fig. 11-2. In addition to all the registers of the 8080, the Z-80 contains a second accumulator, second set of general purpose registers, and two index registers.

The 6800 and 6502 MPUs are characterized by a minimum of general purpose registers. Instead, they have addressing registers (index) which facilitate the use of memory locations for register functions and I/O transfers. The architecture of the 6800 and 6502 is shown in Figs. 11-3 and 11-4,respectively.

When comparing MPUs note that extra registers permit complex operations with a minimum number of memory and register interchanges. General purpose registers are particularly valuable since they can be used to hold temporary results, addresses, and other data, without reference to memory, and so speed up program execution.

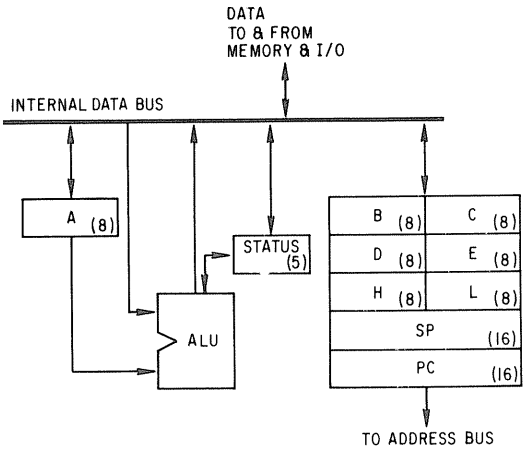


Fig. 11-1. Programming model of the 8080/8085 MPU.

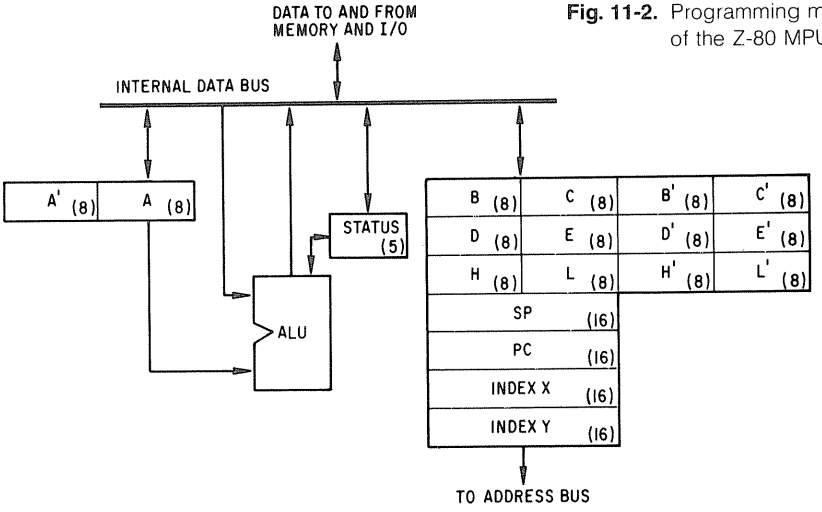
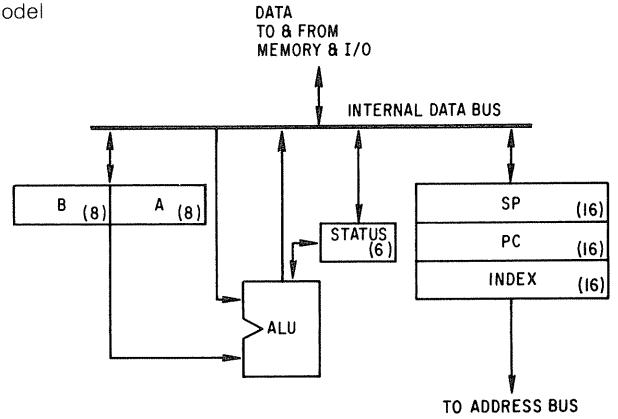


Fig. 11-2. Programming model of the Z-80 MPU.

Fig. 11-3. Programming model of the 6800.



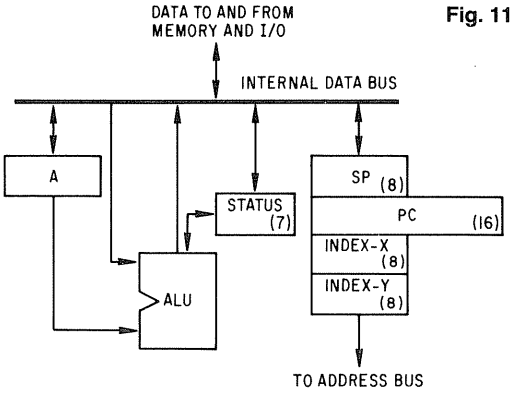


Fig. 11-4. Programming model of the 6502.

The ALU performs add, subtract, and logic operations. The ALU has two operands—one held in the accumulator and the other in one of the registers or memory. The result ends up in the accumulator. Note that the 6502 can also do BCD arithmetic directly.

The results of ALU operations are often indicated by the flag bits of the status register. The status register was discussed in Chapter 6 and shown in Figs. 6-6, 6-7, and 6-8. The flag bits can be tested to determine subsequent operations. Commonly used flags include carry, zero, sign, parity, interrupt status, cycle status, and I/O status. For example, the 8080 has five status flags. They are:

Zero: If the result of an operation = 0, then Z = 1; otherwise, Z = 0.
Sign: If the most significant bit of result of an operation = 1, then S = 1; otherwise, S = 0.
Parity: If there is an even number of 1s in result of an operation, P = 1; otherwise, P = 0.
Carry: If operation results in a carry or borrow out of bit B_7, C = 1; otherwise, C = 0.
Auxiliary Carry: If operation causes carry between bits B_3 and B_4, then AC = 1; otherwise, AC = 0.

MPU INSTRUCTIONS

MPU instruction words (bytes) are composed of 8 bits, the same as data words. The bits are B_0 through B_7 with the *least significant bit* (*LSB*) shown on the right and the *most significant bit* (*MSB*) shown on the left.

$$B_7\ B_6\ B_5\ B_4\ B_3\ B_2\ B_1\ B_0$$
MSB LSB

The instruction may consist of 1, 2, or 3 bytes. The first byte is always the *operation code* (*Op Code*). For example, the 8080 instruction MOV A,B (78_H) is

a 1-byte instruction directing the MPU to move the contents of the B-register to the A-register (accumulator). The hex code for this instruction is shown in parenthesis.

A 2-byte instruction has the first byte as the Op Code and the second byte as data or address. The second byte is called an *operand.* For example:

Hex Code	*Mnemonic Code*
06_H	MVI-B
00_H	00_H

This 2-byte instruction directs the MPU to move the operand 00_H into the B register. Here is another example:

Hex Code	*Mnemonic Code*
$D3_H$	OUT
01_H	01_H

This 2-byte instruction directs the MPU to output the contents of the accumulator to port 01.

A 3-byte instruction has its first byte as the Op Code and the second and third bytes as either an address or data operand. Here are two examples:

$3A_H$	LDA
20_H	20_H
31_H	31_H
21_H	LXI-H
20_H	20_H
31_H	31_H

The first instruction directs the CPU to load into the A register the contents of MA 3120. (Note that the 8080/8085 uses an MA format of low byte-high byte, and the 6800 and 6502 are just the opposite.) The second instruction directs the MPU to place the following 2 bytes (31_H and 20_H) into the H-L register pair.

ADDRESSING MODES

Memory addressing instructions can be of several types.

Direct Addressing: The memory address of the *operand* (data to be used by the op code when executing an instruction) is included as part of the instruction. For example, the program in Chapter 10 used direct addressing.

Immediate Addressing: The instruction contains the operand and the MA is contained in a register. For example, the following 8080 instruction moves the data (FF_H) to the MA contained in the H-L register pair.

36_H MVI-M
FF_H FF_H

Register Direct: The instruction specifies the register, or register-pair, in which the data is located.

Register Indirect: The instruction specifies a register-pair which contains the MA where the data is located.

Relative Addressing: Similar to indirect addressing, except that the address of the operand's MA is computed by adding the contents of the PC to the data in the next instruction. The 8080 does not have this addressing mode.

Indexed Addressing: Similar to relative addressing, except that an index register takes the place of the PC. This addressing mode is excellent for handling tables and arrays. The 8080 does not have this addressing mode.

Stack Addressing: The SP register contains the address of the stack. The stack is an area in memory used for temporary storage of the contents of the registers (i.e., when an interrupt is serviced, the contents of the registers are saved in the stack). The stack is a *last-in, first-out* memory (*LIFO*). Each time the MPU places data in or removes data from the stack, it increments or decrements the SP register. For example, the following 8080 instruction moves the contents of the accumulator to the MA that is 1 less than the SP. Also, the status flags are assembled into a word and moved to the MA that is 2 less than the SP. The contents of the SP is decremented by 2.

F5 PUSH-PSW

No single addressing method solves all programming tasks. Several methods are usually used in a given program. Direct addressing is convenient for single data items. Indirect, indexed, and stack addressing are used for arrays, lists, or tables. Immediate addressing is useful for constants, while relative addressing makes programs shorter and easier to move.

The 8080/8085 has direct, immediate, stack, register direct, and register indirect addressing. The 6800 and 6502 have direct, immediate, indexed, relative, and stack addressing. The Z-80 offers all these addressing methods.

I/O ADDRESSING

In some computers, such as the 6800 and 6502, the I/O devices are addressed in the same manner as memory locations. The I/O devices are then actually part of the system memory space and all the various memory addressing modes can be used for I/O transfers as well. Such *memory mapped* I/O systems offer significant advantages because memory addressing is more versatile than I/O addressing.

An example of this I/O addressing is seen in the SWTP-6800 CPU. Addresses 8000_H to $801F_H$ are used to address eight I/O ports (Fig. 11-5). Each port

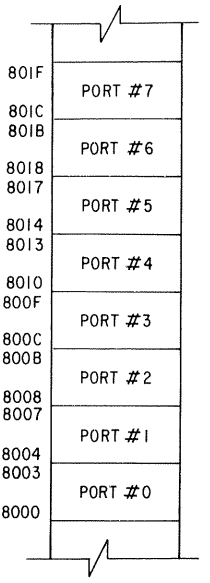


Fig. 11-5. Memory-map of I/O ports in SWTP-6800 CPU.

uses four MAs. If the 6820 PIA IC is used, then MA 8000 selects interface register A, MA 8001 selects control register A, MA 8002 selects interface register B, and MA 8003 selects control register B.

The 8080/8085 and Z-80 MPUs have separate addressing of I/O ports. Hence, an IN or OUT instruction, followed by a second byte for the port address, is used. This limits the MPU to a maximum of 256 input and 256 output ports.

THE INSTRUCTION SET

Computer program execution is a sequence of instruction executions. These instructions fall into four groups:

data-transfer
arithemtic/logic
control
transfer of control

The following is a brief description of these four groups of instructions and how they apply to the 8080/8085 MPU.

Data-transfer instructions move data from one location to another in the computer system. There are four types of transfer instructions: (1) move data to or from memory; (2) move data to or from I/O devices; (3) move data from one register to another; and (4) move data to or from the stack in memory (updating the SP).

Table 11-1. 8080/8085 Data Transfer Instructions

Instruction	*Operation*	*Bytes*
MDVr_1, r_2	Move register 2 to register 1 (e.g., MOV A,B)	1
MOV M,r	Move register to memory	1
MOV r,M	Move memory to register	1
MVI r	Move immediate to register	2
MVI M	Move immediate to memory	2
LXI rp	Load register pair immediate	3
LXI SP	Load SP register immediate	3
LDA	Load A direct	3
STA	Store A direct	3
LHLD	Load H-L direct	3
SHLD	Store H-L direct	3
LDAX	Load A indirect	1
STAX	Store A indirect	1
XCHG	Exchange H-L and D-E	1
PUSH rp	Move register pair to stack	1
PUSH PSW	Move A and status register to stack	1
POP rp	Move data from stack to register pair	1
POP PSW	Move data from stack to A and status registers	1
XTHL	Exchange H-L register with stack	1
SPHL	Move H-L contents to SP register	1
IN	Input from port to A register direct	2
OUT	Output from A register to port direct	2

Table 11-1 lists the 8080/8085 data transfer instructions. Note that "r" denotes a register. The registers can be A, B, C, D, E, H, and L. Then "rp" is a register pair (B-C, D-E, H-L). And "M" denotes an MA pointed to by H-L register pair.

Arithmetic/logic instructions modify the contents of an internal register or flag with an arithmetic or logic operation. The following types of operations are performed:

arithmetic
logic
shift
comparison
special purpose

All the instructions in this class affect the status bits, so these instructions may be used together with the transfer of control instructions to make decisions.

Table 11-2 gives the 8080/8085 arithmetic and logic instructions. Noted are the flag bits affected and the number of bytes.

Arithmetic instructions are straightforward. Note that *SUB* can also be used to determine if two quantities are equal. We subtract one from the other—if

Table 11-2. 8080/8085 Arithmetic and Logic Instructions

Instruction	*Operation*	*Affected Flags*	*Bytes*
ADD r	Add register to A	All	1
ADD M	Add memory to A	All	1
ADI	Add to A immediate	All	2
ADC r	Add register to A with carry	All	1
ADC M	Add memory to A with carry	All	1
ACI	Add to A immediate with carry	All	2
SUB r	Subtract register from A	All	1
SUB M	Subtract memory from A	All	1
SUI	Subtract from A immediate	All	2
SBBr	Subtract register from A with borrow	All	1
SBBM	Subtract memory from A with borrow	All	1
SBI	Subtract from A immediate with borrow	All	2
INR r	Increment register	Z, S, P, AC	1
INR M	Increment memory	Z, S, P, AC	1
DCR r	Decrement register	Z, S, P, AC	1
DCR M	Decrement memory	Z, S, P, AC	1
INX rp	Increment register pair	None	1
DCX rp	Decrement register pair	None	1
DAD rp	Add register pair to H-L	C	1
DAA	Decimal adjust accumulator	All	1
ANA r	AND register with A	All	1
ANA M	AND memory with A	All	1
ANI	AND immediate with A	All	1
XRA r	Exclusive OR register with A	All	1
XRA M	Exclusive OR memory with A	All	1
XRI	Exclusive OR immediate with A	All	1
ORA r	OR register with A	All	1
ORA m	OR memory with A	All	1
ORI	OR immediate with A	All	1
CMP r	Compare register with A	All	1
CMP M	Compare memory with A	All	1
CPI	Compare immediate with A	All	1
RLC	Rotate A left	C	1
RRC	Rotate A right	C	1
RAL	Rotate A left through carry	C	1
RAR	Rotate A right through carry	C	1
CMA	Complement A	None	1
CMC	Complement carry	C	1
STC	Set carry	C	1

the result is zero, the two are equal. ADC and SBB allow us to perform multiple-word arithmetic using the *carry* or *borrow* to transfer information between words. INR and DCR are used to increment and decrement counters, indexes, or indirect addresses. DAA is used to perform BCD rather than binary add operations.

The logic AND is used to *mask* bits, i.e., to remove one or more bits from a word. For example, assume that we want to see if a switch attached to line #3 of an input port is closed (0) or open (1). The procedure is to fetch the switch data from the port and AND it with a mask which has a 1 in bit position 3 and zeros elsewhere. Since anything ANDed with zero is zero, the result depends only on the status of the one switch; the result is zero if, and only if, that switch is closed. Logical OR can be used to combine fields and to set bits (i.e., by ORing with a 1 bit); logical Exclusive-OR can reverse bits; and complement is necessary for subtraction and for handling peripherals.

Shift operations allow the placement of bits or groups of bits where they can be easily handled. They are also used for multiplication and division, scaling, serial-to-parallel and parallel-to-serial conversions, and many mathematical functions. RLC and RRC move the data left or right and fill the empty bit with a

Table 11-3. 8080 Transfer Instructions

Instruction	*Operation*	*Bytes*
JMP	Jump direct	3
JNZ	Jump direct if Z = 0	3
JZ	Jump direct if Z = 1	3
JNC	Jump direct if C = 0	3
JC	Jump direct if C = 1	3
JPO	Jump direct if P = 0	3
JPE	Jump direct if P = 1	3
JP	Jump direct if S = 0	3
JM	Jump direct if S = 1	3
CALL	Call subroutine direct	3
CNZ	Call subroutine direct if Z = 0	3
CZ	Call subroutine direct if Z = 1	3
CNC	Call subroutine direct if C = 0	3
CC	Call subroutine direct if C = 1	3
CPO	Call subroutine direct if P = 0	3
CPE	Call subroutine direct if P = 1	3
CP	Call subroutine direct if S = 0	3
CM	Call subroutine direct if S = 1	3
RET	Return from subroutine	1
RNZ	Return from subroutine if Z = 0	1
RZ	Return from subroutine if Z = 1	1
RNC	Return from subroutine if C = 0	1
RC	Return from subroutine if C = 1	1
RPO	Return from subroutine if P = 0	1
RPE	Return from subroutine if P = 1	1
RP	Return from subroutine if S = 0	1
RM	Return from subroutine if S = 1	1
RSTn	Restart at MA=8 × n	1
PCHL	Jump H-L indirect	1

zero. They are equivalent to multiplying by 2 (left) or dividing by 2 (right). RAL and RAR preserves the sign bit.

Transfer instructions are used to transfer the program execution from the current PC location to some other location in memory. These instructions are either *returning* (*call*) or *nonreturning* (*jump*). A returning transfer saves the address from which it transfers; a nonreturning transfer does not. These instructions are available in conditional and unconditional forms, and all flags may be tested to determine if a transfer is to be made.

Subroutine calls are constructed as returning transfers. When the main program calls a subroutine, the processor saves (on the stack) the main program address where execution is to continue and transfers control to the subroutine. The subroutine performs its operation and its final instruction returns to the main program, using the saved address. Execution then continues.

Table 11-3 gives the 8080 transfer instructions. Note the flags are not affected.

Control instructions are usually used for enabling and disabling all or part of the I/O structure, conditioning the response to interrupts, and halting program execution. The *halt* instruction stops the PC from incrementing and allows the MPU to wait for an external signal. The *no operation* does nothing except increment the PC. It can provide a delay, equalize the execution time of alternate instruction sequences, replace erroneous instructions, or leave space for corrections or additions.

Table 11-4 gives the 8080 control instructions. Note the flags are not affected.

Table 11-4. 8080 Control Instructions

Instruction	*Operation*	*Bytes*
EI	Enable interrupts	1
DI	Disable interrupts	1
HLT	Halt	1
NOP	No operation	1

The appendix includes a complete list of all 8080 Op Codes and their respective hex codes.

IN CONCLUSION

Microprocessors have relatively small (typically 40–80) and simple instruction sets compared to minicomputers and large scale computers. Most lack

instructions for multiplication and division, floating point operations, multiword operations, bit manipulations, complex comparisons and conditional jumps, and block transfer I/O. However, today's MPUs do have more sophisticated instruction sets than older minicomputers (e.g., PDP-8) and large computers (e.g., IBM 1130). New MPUs, such as the Texas Instrument 9900 and Zilog Z-80, have the power of many minicomputers. The future promises MPUs with power comparable to minicomputers. Thus MPUs will be even easier to program and will be far more versatile.

12.

Introduction to Programming

A computer is a *useless* machine until provided with a *program*. The computer does only two things. It fetches an instruction from memory, or some storage medium, and then executes that instruction. The program is the group of instructions which the machine fetches and executes to achieve some desired objective. Programs operate on data. The computer program transforms the data from one form to another.

Writing a program involves designing a specialized instruction sequence that the computer follows. The instructions must be exact and unambiguous so that the computer has only one way of interpreting and therefore executing the program.

Writing a program begins by specifying precisely the task and how it is to be accomplished. This is called an *algorithm*. *Flowcharts* are most often used to graphically present the structure of the program. The flowchart uses differently shaped boxes for each function, with word descriptions inside and interconnections called *flowlines*. Figure 12-1 shows the standard flowchart symbols. Figure 12-2 shows the flowchart for a subroutine to get a character from a Teletype. After the data is transferred, control is returned to the main program.

The routine is called *Get char* and begins by checking the status port to see if the *data-available* bit has been set to 1. If it has, then the data word at the data port is transferred to the MPU accumulator and returns to the point, in the program, from which it left off. If the data-available bit has not been set, the MPU loops back and checks the status bit again.

WHICH LANGUAGE TO USE

After designing the program it must be coded into a computer language. It can be written in assembly language code or in a high-level language such as BASIC (see Chapter 13). If you understand the computer's architecture, as discussed in the preceding chapters, you will not find assembly language programs difficult to write.

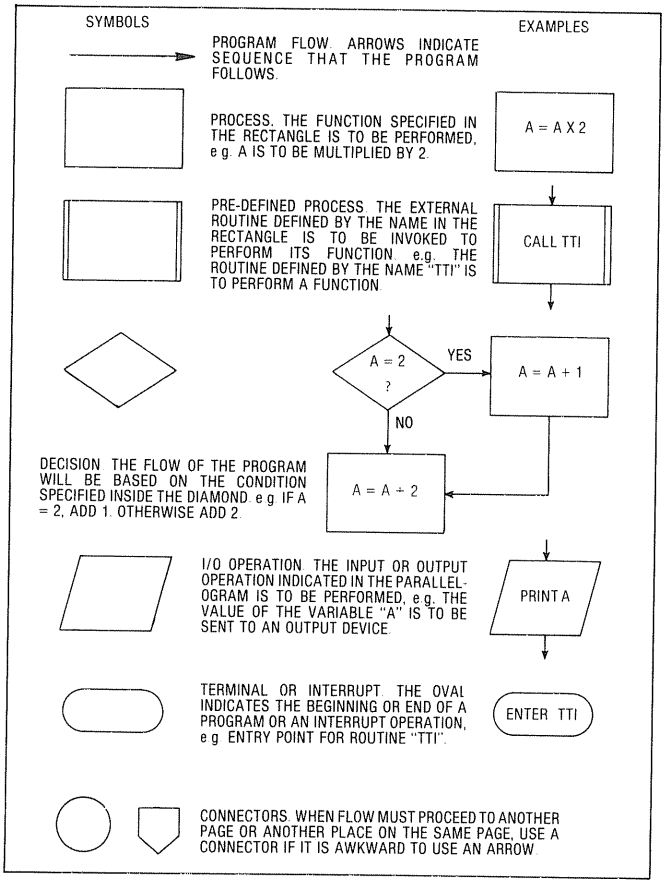


Fig. 12-1. Standard flowchart symbols. (Courtesy *Electronic Design* magazine)

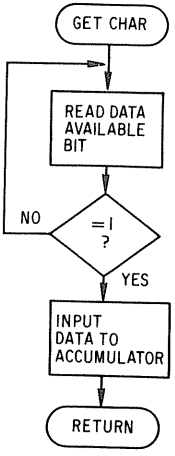


Fig. 12-2. Flowchart for "get character" subroutine.

Most high-level language processors produce one and one-half to two times as much machine code as programs written directly in assembly language. This means one and one-half to two times as much program storage memory is required. Also, the language processors for most high-level languages are much larger than assembler processors. Hence, as much as four to five times as much memory may be required to handle high-level language programs. Furthermore, high-level programs, being longer, take longer to execute. On the other hand, development time is less with a high-level language because you can write and debug programs much faster.

If the program you are writing is to be placed in ROM, assembly language programming is recommended, to economize on program size.

WRITING ASSEMBLY LEVEL PROGRAMS

Assembly level programs can be done by a hand assembly or machine assembly technique. Most experienced programmers will hand assemble programs of 100 steps or less and for larger program development use editor/assembler programs.

Hand assembly means that the programmer looks up the mnemonic instructions (see Chapter 11) and their corresponding hex codes and writes the program on paper. The program is then loaded into the CPU's memory using a monitor/debug program and run. For example, here is the program subroutine shown in Fig. 12-2 written in 8080/8085 mnemonic source code.

Instruction		*Comments*
CI	IN TTYS	Input TTY status byte to accumulator
	ANI TTYDA	Mask off DA (data available) bit
	JNZ CI	if B_0=0 jump back to input again
	IN TTYD	Input data byte to accumulator
	RET	Return to main program

The program would then be assembled into machine (object) code by looking up the hex code (see Appendix A) for the corresponding Op Codes, assigning memory, and port addresses. The assembled program would then look like this.

MA	*Object Code (hex)*	*Instruction*
CI		
2E48	DB 00	IN TTYS
2E4A	E6 01	ANI TTYDA
2E4C	C2 48 2E	JNZ CI
2E4F	IN 01	IN TTYD
2E51	C9	RET

The hex code could now be loaded into the indicated memory addresses of the CPU and used to load characters from a TTY. All that is necessary for this hand assembly is a small monitor-debug program. These programs range from 256 bytes to 2K bytes. The longer monitors having more features.

THE EDITOR/ASSEMBLER

Short routines and programs, such as the previous example, are easily hand assembled and debugged. However, a longer program of a few hundred or a few thousand steps is another matter. These require the use of editor and assembler software.

The programmer loads the editor program into the CPU and enters the program in mnemonic form. He would use a descriptive label to indicate a subroutine, all Op Codes, operands, and some helpful comments. Here is an example:

```
Label   Op Code   Operand        Comment
                  ; GET CHARACTER FROM TTY:
CI:     IN        TTYS           ; INPUT STATUS
        ANI       TTYDA          ; CHECK DA BIT
        JNZ       CI             ; IF=0 DO AGAIN
        IN        TTYD           ; INPUT DATA
        RET
        ;

                  ; PRINT CHARACTER ON TTY:
CO:     IN        TTYS           ; INPUT STATUS
        ANI       TTYBE          ; CHECK BE BIT
        JNZ       CO             ; IF=0 DO AGAIN
        MOV       A,C            ; MOVE DATA TO A
        OUT       TTYO           ; OUTPUT DATA
        RET
        ;

                  ; TYPE CARRIAGE RETURN-LINE FEED:
CRLF:   MVI       C, 'CR'        ; MOVE 'CR' TO C
        CALL      CO             ; CALL PRINT ROUTINE
        MVI       C, 'LF'        ; MOVE 'LF' TO C
        CALL      CO             ; CALL PRINT ROUTINE
        ORA       A              ; CLEAR A
        RZ                       ; IF A CLEAR RETURN
        ;
```

In addition, the programmer would have to define I/O ports, status words, and the starting address for the program in memory. This is done as follows:

```
TTYD    EQU    1
TTYO    EQU    1
TTYS    EQU    0
TTYDA   EQU    01H
TTYBE   EQU    80H
CR      EQU    0DH
LF      EQU    0AH
ORG     BASE   2E48
```

The source program created via the editor is now stored on magnetic or paper tape or on disc. The assembler program is now loaded and the source program is subsequently processed through the assembler. Of course, if the editor and assembler programs are coresident in memory the loading is not necessary.

The assembler now translates the MPU instruction mnemonics in the Op-Code field into machine code. The symbolic operand field is also translated into hex code. For example, in the CRLF subroutine MVI C, 'CR' instruction (move the ASCII character CR to register C) causes the translation of 0E (MVI C) and 0D (ASCII hex code for CR).

In the first pass through the assembler the user-created subroutine *labels* are identified and given memory addresses. The assembler thus builds a *symbol,* or *label, table.* A second pass is now made. The assembler matches up all entries in the symbol table with labels in the operand field and assigns the proper memory addresses.

The assembler selects the starting address from the *pseudo Op-Code* ORG (origin) and, hence, starts the program at MA 2E48. Also EQU (equivalence) directs the assembler to give a value to the symbol. For example TTYD (TTY input port) is assigned the value 01.

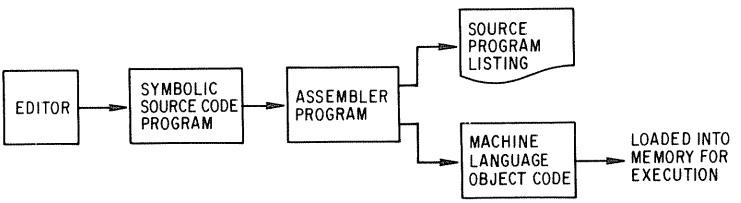


Fig. 12-3. Operation steps of an assembler program.

The assembler (Fig. 12-3) now can generate either an object code tape or a source listing. A typical source listing would appear as shown on the following page:

MACHINE LANGUAGE		ASSEMBLY LANGUAGE			
Location	*Object Code*	*Label*	*Op Code*	*Operand*	*Comments*
				; GET CHARACTER FROM TTY:	
2E48	DB 00	CI:	IN	TTYS	; INPUT STATUS
2E4A	E6 01		ANI	TTYDA	; CHECK DA BIT
2E4C	C2 48 2E		JNZ	CI	; IF=0 DO AGAIN
2E4F	DB 01		IN	TTYD	; INPUT DATA
2E51	C9		RET		
			;		
				; PRINT CHARACTER ON TTY:	
2E52	DB 00	CO:	IN	TTYS	; INPUT STATUS
2E54	E6 80		ANI	TTYBE	; CHECK BE BIT
2E56	C2 52 2E		JNZ	CO	; IF=0 DO AGAIN
2E59	79		MOV	A,C	; MOVE DATA TO A
2E5A	D3 01		OUT	TTYO	; OUTPUT DATA
2E5C	C9		RET		
			;		
				; TYPE CARRIAGE RETURN-LINE FEED:	
2E5D	0E 0D	CRLF:	MVI	C, 'CR'	; MOVE 'CR' TO C
2E5F	CD 52 2E		CALL	CO	; CALL PRINT ROUTINE
2E62	0E 0A		MVI	C, 'LF'	; MOVE 'LF' TO C
2E64	CD 52 2E		CALL	CO	; CALL PRINT ROUTINE
2E67	B7		ORA	A	; CLEAR A
2E68	C8		RZ		; IF A CLEAR RETURN
			;		
			END		

The assembler also prints a label table, as follows:

CO	2E52	TTYD	0001	TTYDA	0001
CI	2E48	TTYO	0001	TTYBE	0080
CRLF	2E5D	TTYS	0000	CR	000D
				LF	000A

MACROASSEMBLERS

Macroassemblers are available for many MPU systems. They are more powerful than nonmacro-type assemblers and greatly simplify program development. They permit the user to give a sequence of instructions a name which, when given in a program, causes the macroassembler to replace the name with the proper instruction sequence. Macros are thus used to define common sequences of code that are often used.

DEBUGGING

The assembler will detect syntax errors. For example, if a symbol appears in the symbol field but not in the operand field (and is thus undefined), an error message will be given. Likewise, if the assembler encounters a symbol more than once in the label field, it will put out an error message. Also, the assembler will detect illegal forms for symbols (for example, use of an Op-Code mnemonic for a symbol).

The programmer must now return to the editor and correct the program and reassemble it again. If only a small number of errors exist they can be easily corrected in the monitor phase to follow. When all errors have been removed the program can be saved on paper or magnetic tape or disc. It can then be loaded into the CPU via the monitor for running.

PROGRAM EXAMPLES

Here are some examples of short assembler level programs.

Program #1: This program clears memory from MA 0000_H to 1000_H. It resides in the last 4K block of RAM and checks that each MA is cleared. The flowchart is shown in Fig. 12-4.

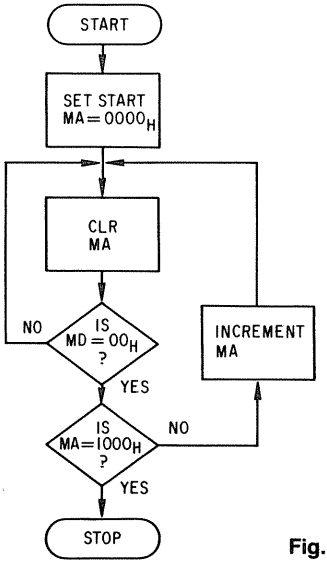


Fig. 12-4. Flowchart for clear memory program.

MA (hex)	Hex Code	Labels	Mnemonic Code
1FE9	21 00 00		LXI H-L
1FEC	3E 00	LOOP CLR	MVI A
1FEF	BE		CMP-M
1FF0	C2 EC 1F		JNZ LOOP CLR
1FF3	7C		MOV A,H
1FF4	FE 10		CPI 10
1FF6	CA FD 1F		JZ LOOP END
1FF9	2C		INCR L
1FFA	C3 EC 1F		JMP LOOP CLR
1FFD	C3 FD 1F	LOOP-END	LOOP END

Program #2: This program sorts data in locations 50_H up to MA-1 of last location of data. It sorts the data in increasing order. Figure 12-5 is the flowchart for the program.

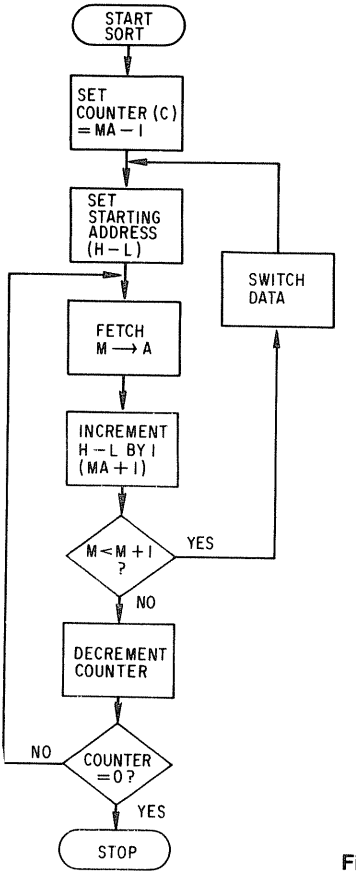


Fig. 12-5. Flowchart for sort program.

MA (hex)	*Hex Code*	*Labels*	*Mnemonic Code*
0000	0E MA-1		MVI-C MA – 1
0002	21 50 00	START	LXI-H 0050H
0004	7E	SORT	MOV A,M
0005	A7		ANA-A
0006	23		INX-H
0007	BE		CMP M
0008	DA 12 00		JC TEMP
000B	ϕD		DCR-C
000C	C2 04 00		JNZ SORT
000F	C3 1B 00		JMP END
0012	56	TEMP	MOV B,M
0013	2B		DCX-H
0014	7E		MOV A,M
0015	70		MOV M,B
0016	23		INX-H
0017	77		MOV M,A
0018	C3 02 00		JMP START
001B	76	END	HLT

Recommended Further Reading

1. Donald E. Knuth, *The Art of Computer Programming,* Vols. 1 and 2, Addison-Wesley, Reading, Mass., 1969.
2. *6502 Programming Manual,* MOS Technology, Norristown, Pa., 1975.
3. Robert Findley, *Scelbi's '8080' Software Gourmet Guide and Cook Book,* Scelbi Computer Consulting, Inc., Milford, Conn., 1976.
4. Adam Osborne, *8080 Programming For Logic Design,* Adam Osborne & Assoc., Inc., Berkeley, Calif., 1976.
5. Adam Osborne, *6800 Programming For Logic Design,* Adam Osborne & Assoc., Inc., Berkeley, Calif., 1977.

13.

Programming with BASIC

BASIC stands for *B*eginner's *A*ll-purpose *S*ymbolic *I*nstruction *C*ode. It is the most widely used high-level language. Programs written in BASIC will run on most computers with little or no change. This is not the case with the assembler level programs studied in the previous chapter. An 8080 program will not run on CPUs using other MPUs.

BASIC is also much easier to learn. In fact it is one of the easiest computer languages to learn. Hence, it is considered a "beginners" language. But do not be fooled; it is a full-fledged language suitable for most applications.

There are numerous versions of BASIC available: Business BASIC, Extended BASIC, Super BASIC, and even Tiny BASIC to run on CPUs with a "tiny" memory. All versions have much in common and the following discussion applies to most of them.

THE BASIC PROGRAM

A BASIC program consists of a series of lines, each beginning with a line number (integers only) followed by a command. Unless directed otherwise the computer then executes the commands one at a time by order of the ascending line number. Each line, or *statement,* is thus made up of a line number, capital letters, numbers, and a few special characters. Line numbers are usually assigned in multiples of ten to permit adding statements should the need arise.

Let's look at a simple program to solve the expression $E = I \times R$. This is a very fundamental formula in electronics that says that voltage is the product of current (amperes) and resistance (ohms). Here is the program as written in BASIC:

```
10  PRINT "WHAT IS CURRENT AND RESISTANCE?"
20  INPUT I,R
30  LET E=I*R
40  PRINT "VOLTAGE=";E
50  END
```

Every statement begins with a *key word,* which specifies the action desired. Line 10 specifies PRINT, telling the terminal to print the *character string* within the quotation marks. Line 20 tells the CPU to INPUT two pieces of data from the terminal (the values for I and R). The LET statement indicates the value on the right is to be assigned the value on the left of the equals sign. The asterisk indicates multiplication. Line 40 specifies the printing of a character string (within the quotes) and the value of E (computed in line 30). The key word END tells the computer to stop processing statements.

Note that the BASIC language program also contains an editor program that lets you add, delete, or change lines by number.

To execute the program the user types RUN and hits CR (Carriage Return) on the terminal. The program's execution is as follows (note that underlining indicates user-typed characters):

RUN (CR)
WHAT IS CURRENT AND RESISTANCE?
? 2,50
VOLTAGE=100

To correct a program, type the line number of the line to be changed and type in the new statement. This automatically erases the old statement. To delete a line (e.g., line 20) type:

DELETE 20

If while typing in a line you discover an error, backspace to the error and begin typing again. This replaces the old characters with the new characters.

SOME BASIC FUNDAMENTALS

Constants and variables are represented with a letter which closely symbolizes the quantity to be represented, i.e., E, I, and R. We can also follow the letter with a digit for multiple variables, i.e., V1, V2, V3, etc.

Quantities may be integers or real quantities. The following are formats accepted by BASIC.

50	2.6789	0.0000123
10.5	456.7	+1234.567
−8.07	0.1234	1.2E12

Notice that commas are not used to mark off thousands, etc. A "+" symbol, if not used, is assumed. Minus signs must precede the number to indicate a negative quantity. Scientific notation is represented by the letter E; for example, 1.2×10^{12} is typed as 1.2E12. The exponent must be an integer and can be either positive or negative.

Character strings (alpha-numeric text) are indicated by enclosing them in quotation marks. The final quotation mark must be on the same line as the beginning mark.

BASIC OPERATORS

BASIC has six comparative operators. They are:

Symbol	Example	Significance
=	B=A	B equal to A
<	B<A	B less than A
>	B>A	B greater than A
<=	B<=A	B less than or equal to A
>=	B>=A	B more than or equal to A
<>	B<>A	B not equal to A

BASIC has six mathematical operators. They are (shown with hierarchy of operations in descending order):

Symbol	Example	Significance
( )	8/(2*2)=2	Subexpression
↑	3↑2=9	Exponentiate
*	3*2=6	Multiply
/	3/2=1.5	Divide
+	3+2=5	Add
−	3−2=1	Subtract

BASIC FUNCTIONS

Most BASICs have a predefined set of functions. They are:

Function	Mathematical Notation	Description
ABS(x)	$\|x\|$	Absolute value of x
SIN(x)	sin x	Sine of x(radians)
COS(x)	cos x	Cosine of x(radians)
TAN(x)	tan x	Tangent of x(radians)
ATN(x)	$\tan^{-}x$	Arctangent of x(radians)
EXP(x)	e^x	Natural exponential of x
LOG(x)	ln x	Natural log of x
INT(x)	$[x]$	Integer part of x
SQR(x)	$\sqrt{x}$	Square root of x
RAND(x)		Random number (between 0 and 1)

BASIC STATEMENTS

Most BASICs have the following eleven statements. One statement does not cause any program execution. All the rest do. This one statement is the *REM statement*. It is used to insert remarks in the program. It is recommended that remarks be used liberally to explain the program's operation. This way other users will be better able to work with it and the originator can even use them as an aid to debugging, particularly after time has passed. The REM statements will be printed out during the listing of the program and will have no effect on the program's execution.

The *LET statement* causes the entire arithmetic expression to the right of the ''='' sign to be evaluated and then assigns the value to the named variable(s). Hence, the variable on the left of the ''='' sign may also appear on the right. Here are some examples of LET statements.

```
10  LET A=0
20  LET A=B+A
30  LET A2=(B+C)/D
40  LET D4=25+SIN(C)-D4
50  LET A$=''BOY''
```

Note that a variable followed by the symbol ''$'' denotes a character string variable.

The *PRINT statement* is used to print out the results of an arithmetic expression, print out a character string, or print out a constant. More than one element may be printed. This is indicated by commas or semicolons between the elements. The comma tells the computer to move to the next printing zone (usually 15 character positions in a printing zone) to print the next element. The semicolon tells the computer to print the element without skipping spaces. Here are some examples of *PRINT* statements.

```
10  PRINT A
20  PRINT A-1.2
30  PRINT A$
40  PRINT ''THE RESULT='';A
50  PRINT A,B,C
```

The *INPUT statement* is used for inputting data or character strings into the program. The INPUT statement may input one or more variables (denoted by commas between the variables). When running the program the INPUT statement causes a prompting ''?'' to be typed for each piece of data to be inputted.

The *STOP* and *END statements* are used to terminate execution of the program. The STOP statement may be located anywhere in the program while the END statement must be at the end of the program.

The *GO TO statement* causes an unconditional jump to a given statement number. For example:

```
100  GO TO 50
```

causes execution to return to line 50 and execute from that point onward.

The *IF . . . THEN statement* causes a conditional jump if a specific condition is met. If the condition is not met the program continues to the next statement. Here are some IF . . . THEN statements.

```
10  IF A>B THEN 150
20  IF A$="BOY" THEN 150
30  IF A<>B THEN 120
```

The *FOR* and *NEXT statements* permit easy programming of loops. The FOR statement defines the beginning of the loop and identifies the controlling parameters. For example:

```
10  FOR N=1 TO 5 STEP1
20  PRINT N,N+N
30  NEXT N
40  END
```

Line 10 establishes the variable N and initializes it = 1. It also defines the end-test value (5) for the loop and step size (1). The program will start with N = 1 and print N and N + N (line 20). Line 30 will loop back to line 10, test N to see if it is = 5, and if not, increments it by 1. If no step is specified, the computer assumes a step value of 1. The output will look like this:

```
RUN
 1          2
 2          4
 3          6
 4          8
 5          10
```

A negative step may be used. Note when using multiple loops that they must be nested one within the other.

The *GOSUB statement* permits the calling of a subroutine. Program control returns to the line after the GO SUB statement when referenced by a *RETURN statement* at the end of the subroutine. For example:

```
90  GOSUB 210
```

directs the computer to jump to line 210, the first line of the subroutine, and

```
350  RETURN
```

tells the computer to go back to the first line number greater than 90 and to continue the program from there.

The *READ* and *DATA statements* assign to the listed variables values obtained from the DATA statement. Neither statement is used without the other. A READ statement causes the variables listed to be given, in order, the next available quantity in the collection of DATA statements. The computer takes all of the DATA statements, in the order in which they appear, and creates a large data block. Each time a READ statement is encountered anywhere in the program, the data block supplies the next available number or numbers. If the data block runs out of data, with a READ statement still asking for more, the program is assumed to be done. Here is an example which uses these statements:

```
10  READ A,B,D,E
15  LET G=A*E-B*D
20  IF G=0 THEN 65
30  READ C,F
37  LET X=(C*E-B*F)/G
42  LET Y=(A*F-C*D)/G
55  PRINT X,Y
60  GO TO 30
65  PRINT ''NO UNIQUE SOLUTION''
70  DATA 1,2,4,2
80  DATA -7,5
85  DATA 1,3,4,-7
90  END
```

When this program is processed it produces the following output:

```
RUN
 4              -5.5
  .666667        .166667
-3.66667        3.83333
OUT OF DATA IN 30
```

Note that the first time through the program the following values were assigned to the variables: A=1, B=2, D=4, E=2, C=−7, and F=5. The second time around the computer found no values supplied for C and F and hence, stopped execution and printed the error message OUT OF DATA IN 30.

LISTS, TABLES, AND ARRAYS

We can enter a list (single dimension array) A(0),A(1) . . . A(10) into a program very simply by the lines:

```
10  FOR I=0 to 10
20  READ A(I)
30  NEXT I
40  DATA 2,3,-5,7,2.2,4,-9,123,4,-4,3
```

We need no special instruction to the computer if no subscript greater than 10 occurs. However, if we want larger subscripts, we must use a *DIM* (dimension) statement to define the space required for the list. When in doubt, indicate a larger dimension than you expect to use. Here is an example of a program to set up a 3 × 5 array (*matrix*).

```
10  DIM I(3), J(5), B(3,5)
15  FOR I=1 TO 3
20  FOR J=1 TO 5
30  READ B(I,J)
40  NEXT J
50  NEXT I
60  DATA 2,3,-5,-9,2
70  DATA 4, -7,3,4,-2
80  DATA 3,-3,5,7,8
```

} 3 × 5 array

FUN WITH BASIC

BASIC is a highly interactive language and hence, it can give the computer human characteristics. Here is a program segment which makes the computer seem human:

```
10  PRINT "WHAT IS YOUR NAME?"
20  INPUT N$
30  PRINT "HELLO";N$;".";"  ";"HOW ARE YOU?"
40  PRINT "GOOD, BAD, OR FAIR?"
50  INPUT M$
60  IF M$="GOOD" THEN 110
70  IF M$="BAD" THEN 130
80  IF M$="FAIR" THEN 150
90  PRINT "I DO NOT UNDERSTAND YOU";N$
100  GOTO 40
110  PRINT "I AM GOOD TOO";N$
120  GOTO 160
130  PRINT "SORRY TO HEAR IT";N$
140  GOTO 160
150  PRINT "ME TOO!"
160  PRINT "SO LONG NOW."
170  END
```

Here is what the program's execution will look like:

```
RUN
WHAT IS YOUR NAME?
? CHARLEY
```

```
HELLO CHARLEY. HOW ARE YOU?
GOOD, BAD, OR FAIR?
?SO SO
I DO NOT UNDERSTAND YOU CHARLEY
GOOD, BAD, OR FAIR?
?FAIR
ME TOO!
SO LONG NOW.
```

BASIC is also very popular for playing games. Here is a version of the popular program called *Lunar Lander*. Note that many statements have been condensed to one or two letters, e.g., PR for PRINT and L for LET. This is permitted on most BASICs and in fact saves memory space.

```
5  REM LUNAR LIFEBOAT
6  REM BY DAVID KRAUSS AND TOM MARTIN
7  REM MAR. 10, 1977
10 PR"DO YOU DESIRE INSTRUCTIONS? TYPE 'Y' FOR YES, 'N' FOR NO."
11 INPUT Z
12 IF Z=N GOTO 30
14 REM INSTRUCTION BLOCK FROM LINE 15 TO LINE 29
15 PR"WHILE FLYING A LOW ORBIT MAPPING MISSION ON THE MOON,"
16 PR"YOUR CRAFT HAS HIT A FLYING WOMBAT! (AN AVIAN MAMMAL"
17 PR"NATIVE TO THE AREA) YOU ARE SAFE INSIDE YOUR EJECTED"
18 PR"SURVIVAL CAPSULE WHEN YOU DISCOVER THAT YOUR AUTOMATIC"
19 PR"DESCENT COMPUTER IS JAMMED FULL OF WOMBAT FEATHERS"
20 PR"AND HAS FAILED. YOUR MAIN THRUST UNIT IS FALTERING BUT"
21 PR"MAY YET GET YOU DOWN SAFELY."
23 PR"TO SAVE THE CAPSULE YOU MUST LAND AT LESS THAN 2 FT/SEC."
24 PR"TO SURVIVE YOU MUST LAND AT LESS THAN 5 FT/SEC."
25 PR"CAPSULE INSTRUMENTATION IS OK AND WILL TELL YOU WHERE YOU ARE."
26 PR"REMEMBER, GRAVITY WILL ADD 5 FT/SEC. TO YOUR DESCENT."
27 PR"GOOD LUCK!"
30 REM L=LIMIT OF BURN
31 L=RND(10)+25
34 PR
35 PR"YOUR ENGINES ARE CAPABLE OF A MAX. BURN OF ";L;"FT/SEC."
40 REM INITIALIZE DATA: T=TIME, H=HEIGHT, V=VELOCITY, F=FUEL LEFT
41 T=1
42 V=RND(75)-75
43 H=RND(300)+200
44 F=120
45 PR
46 PR"MANUAL DESCENT MODE ENGAGED"
47 PR
55 PR"TIME   HEIGHT   VELOCITY   FUEL BURN"
60 PR"SEC.   (FEET)   (FT/SEC)   LEFT"
61 PR"  ";T,H,V,F,
65 INPUT B
66 IF B>L THEN B=L
67 REM L=LIMIT OF BURN
69 IF F<=0 THEN B=0
70 IF B+100<=100 THEN B=0
71 F=F-B
```

```
72  REM B=BURN
73  T=T+1
75  V=V-5+B
76  H=H+V
77  E=RND(12)
78  IF E<9 IF E >6 GOSUB 170
79  REM VARIABLE 'E' DETERMINES THRUST FAILURE
80  IF F<=0 GOTO 100
81  IF H<=0 THEN IF V+100<=95 GOTO 150
82  IF H<=0 THEN IF V+100<=98 GOTO 140
83  IF H>0 GOTO 61
91  PR"CONGRATULATIONS! YOU HAVE LANDED SAFELY."
92  PR"YOUR VELOCITY AT TOUCHDOWN WAS ";V;" FT/SEC."
93  PR"WITH ";F;" UNITS OF FUEL REMAINING"
94  GOTO 160
100  PR"!!!OUT OF FUEL!!!"
101  IF H>0 THEN GOTO 61
105  GOTO 150
140  PR"YOU HAVE MADE A CONTROLLED CRASH! YOU ARE ALIVE"
141  PR"BUT THE LANDER IS DAMAGED AND YOU ARE STRANDED!"
142  GOTO 92
150  PR"CRUNCH! YOU HAVE JUST BECOME THE MOON'S NEWEST CRATER"
151  PR"YOUR FLIGHT PAY WILL BE FORWARDED TO YOUR WIDOW."
142  GOTO 92
150  PR"CRUNCH! YOU HAVE JUST BECOME THE MOON'S
     NEWEST CRATER"
151  PR"YOUR FLIGHT PAY WILL BE FORWARDED TO YOUR
     WIDOW."
152  GOTO 92
160  PR
161  PR"LIKE TO TRY AGAIN? (Y/N)";
162  INPUT Z
163  IF Z=Y GOTO 30
165  END
169  REM DERIVE LEVEL OF THRUST FAILURE
170  L=L-(RND(10)+1)
172  IF L+100<=100 THEN GOTO 195
180  PR"DETERIORATION IN MAIN THRUST UNIT"
190  PR"YOUR MAX. BURN IS NOW ";L;" FT/SEC."
191  RETURN
195  PR"YOUR THRUST UNITS HAVE FAILED COMPLETELY!"
196  L=0
200  RETURN
```

Recommended Further Reading

1. James S. Coan, *Basic BASIC*, Second Edition, Hayden Book Co., Inc., Rochelle Park, N.J., 1978.
2. Robert E. Smith, *Discovering BASIC*, Hayden Book Co., Inc., Rochelle Park, N.J., 1970.
3. James S. Coan, *Advanced BASIC*, Hayden Book Co., Inc., Rochelle Park, N.J., 1976.
4. Byron S. Gottfried, *Programming with BASIC*, McGraw-Hill Book Co., New York, N.Y., 1975.

14.

Applications

Personal computer systems are used by individuals for a wide range of applications. It is not possible to cover all the applications, particularly since new ones are being created all the time. The purpose of this chapter is to pin-point the more common applications and give a general overview of each. Where possible, some references are given, to guide the reader to more in-depth material.

GAME PLAYING

There is no doubt that more time is spent playing games on computers than is spent on any other personal computer activity. This is because computers have brought a new level of sophistication to game playing. Games can be played with the computer as an opponent, and different levels of skill can be selected, as for example, in playing chess with the computer.

Computer games have been created which depend on the computer's ability to store considerable amounts of data and do complex calculations quickly. The most popular of these games is *Star Trek* (after the TV series), a computer game which places the operator in the role of Captain Kirk and has him piloting the starship "Enterprise" through galaxies and entering into combat with the infamous "Klingons." This game can go on for hours and calls for skill on the part of the user. A very impressive version of *Star Trek* is shown in Fig. 14-1. Here, three CRT displays are mounted in a simulated space-ship cockpit. The large center screen displays the universe while the small two side CRTs display status information. Information on building the unit can be obtained from: 2005 AD, Inc., 2005 Nandain Street, Philadelphia, Pa., 19146.

Most of these games are written in BASIC and, hence, require a BASIC interpreter in the system, and some mass storage device (e.g., cassette tape or disc) and typically 12–20K of memory. A CRT type terminal is preferred over a hardcopy terminal.

Fig. 14-1. A *Star-Trek* game display built into a simulated cabin by 2005 AD, Inc. Lance Strickler, one of the builders, dressed in a space suit, is seated next to the unit.

The following are some of the best source materials on computer games.

1. David Ahl, *101 Games in BASIC,* Digital Equipment Corp., Maynard, Mass.
2. *What To Do After You Hit Return,* Peoples Computer Co., Menlo Park, Calif.
3. Donald D. Spencer, *Game Playing with Computers, Second Edition,* Hayden Book Co., Inc., Rochelle Park, N.J. 1976.
4. Nat Wadsworth and Robert Findley, *Scelbi's First Book of Computer Games for the '8008/8080',* Scelbi Computer Consulting, Inc., Milford, Conn., 1976.
5. Robert Findley, *Scelbi's Galaxy Game for the '8008/8080',* Scelbi Computer Consulting, Inc., Milford, Conn., 1976.
6. Peter Jennings, *Microchess,* Peter Jennings, 1612-43 Thorncliffe Park Drive, Toronto, Ontario, Canada, M4H 1J4.
7. Donald D. Spencer, *Game Playing with BASIC,* Hayden Book Co., Inc., Rochelle Park, N.J., 1977.
8. *Metagaming Concepts,* Box 15346, Austin, Texas 78761.
9. *Simulation & Gaming News,* Box 3039, University Station, Moscow, Idaho 83843.

In addition, the following individuals can be contacted.

1. Gerald H. Herd, 3781 Whispering Pines Drive, Pensacola, Florida 32504. Information on *Star Trek* games.
2. Dr. Monroe Newborn, School of Computer Science, McGill University, Montreal, Quebec, Canada, H3C 3G1. Information on chess games.
3. Dr. Robert Suding, Box 6528, Denver, Colorado 80206. Information on educational games.

WORD PROCESSING

Word processing, also called *text editing,* is really the adding of intelligence to a typewriter. For example, it permits the typing of letters, correction of mistakes, and rearranging of sentences and paragraphs with just a few key strokes. When the composition of the letters is satisfactory the computer is told to type the letters. If desired, the computer will even justify the text (even right-hand margins). As many originals can be generated as desired, with headings changed on each letter.

Businesses have been using word processing machines for years, at a cost of $10K to $30K per machine. But now it is possible to do this at a small fraction of the cost.

The hardware is not difficult. In fact a 6-IC CPU for the system is fully described in the January 18, 1977, issue of *Electronic Design* magazine with full description of the hardware and software involved. The CPU can be easily built for under $100.

A standard general purpose CPU, such as most of the CPU kits on the market, can also be used. Usually 4K of RAM is adequate for a minimal system, but 8K and even 16K of RAM is desirable for more elaborate program and buffer text storage.

An upper/lower case type keyboard with a shift-lock key is required for input. The keyboard should have an ASCII output. Although Teletypes may be used for printing, they offer only upper-case letters and do not provide the best quality print. Selectric typewriters can be converted to interface to a CPU, but the conversion is not easy and the conversion kits presently available are expensive. The best machines are those that use Diablo or Queme printers and similar units that use "daisy wheel" printer heads. These units are becoming available on the used-equipment market.

Although the final output must be typed on a printer, it is more desirable to compose the text on a CRT type terminal or TV display. It is much faster and quieter than the printer. Insertions, deletions, movement of words, lines, paragraphs, etc., occur with virtually no delay. The best display to use generates the display from directly addressable computer RAM (e.g., the Processor Technology VDM-1 or Polymorphic Systems Video interface). In this type unit, as bytes

are moved around in RAM, they also move around on the display. The TV typewriter is not suited to this application since changes in text require retransmission of the whole displayed page.

A mass storage device is desirable. Without it you are limited to editing only the text available in RAM. This is all right for one page letters and reports. For longer text a mass storage device allows editing and updating. Furthermore, the text can be saved for reuse or updating at a later time.

Although one cassette recorder may be used for storage, a two-cassette system, employing high speed CPU controlled cassette decks, is preferable. Where it is anticipated that there will be frequent insertions, deletions, and updates, floppy discs are recommended.

Although a standard editor program, such as is used when writing programs, can be used as the editing software, it is far preferable to use a word processor text editor specifically designed for text editing. At this writing, Technical Design Labs, Princeton, N.J., offers a powerful word processor program (Z-80 based). Those wishing to write their own text editor program should read "Text Editing," by Hal Chamberlin, *Popular Electronics,* January 1977. A simple text editor, written in BASIC, can be found in "Computerized Typesetting" by Lee Wilkinson, *Kilobaud,* June 1977.

For those considering constructing their own word processing system, it would be wise to contact: Robert H. Edmonds, Box 464, Estudillo Station, San Leandro, Calif., 94577. Bob is very active in coordinating and stimulating homebrew word processing units. Also active in this area is Ward Christensen, 688 East 154 Street, Dolton, Illinois 60419.

COMPUTER MUSIC

This application of microcomputers is just starting to be explored. Already there have been demonstrations of simple computer music generated by reading a table in RAM to generate a square-wave frequency at bit B_1 position of a parallel port. This technique can play simple melodies. Another simple technique involves picking up the switching signals of the CPU (Steve Dompier, "Music of a Sort," *Dr. Dobbs Journal,* February 1976) on a nearby AM radio. This technique produces surprisingly good simple music with no interface required.

An improvement on the table technique was developed by Malcom Wright (*People's Computer Company Magazine,* January 1976; see his text, *Alpha-Numeric Music With Amplitude Control,* P.C.C., 1976). The music is encoded as ASCII characters. Hence, to play middle C, just type "4C"; for a sixteenth note of B flat, one octave above middle C, type "5SB". In this way the melody is loaded into memory. The user can program volume, tempo, time, rests, repeats, and even create envelopes (attack) over a six-octave range.

More elaborate systems, which provide multivoicing and precise tonal control, have been developed using a Fourier Synthesizing technique (Hal

Fig. 14-2. A music synthesizer board for the S-100 bus. (Courtesy Newtech Computer Systems)

Chamberlin, "Fourier Series Waveform Generator," *Electronotes Magazine,* May 1974).

Many companies are developing music synthesizer kits which either plug directly into the S-100 bus or interface very easily to most microcomputers. A typical music synthesizer board is shown in Fig. 14-2. Along with this is the development of software which enables a composer to write and edit pieces of music for multivoices and then to play the music instantly after completion on his computer.

Also, some companies are developing computer control hardware devices for musical instruments. Dr. Prentis Knowlton demonstrated a pipe organ controlled by a PDP-8 minicomputer on his LP record "Unplayed By Human Hands" (available from Computer Humanities, 2310 El Moreno St., La Crescenta, Calif., 91214).

For more information on computer music, the reader is referred to the following magazines:

1. *Electronotes*
2. *Journal of the Audio Engineering Society*
3. *Journal of the Audio Society of America*

4. *Gravesono Review*
5. *IEEE Transactions on Audio and Electroacoustics*
6. *Computer Music Journal*

Also recommended are the following books:

1. J. W. Beauchamp and H. Von Foerster, Eds., *Music Computer,* Wiley, New York, N.Y., 1969.
2. H. L. F. Helmholtz, *On the Sensations of Tone As a Physiological Basis for the Theory of Music,* translation (originally written in German in 1863) and additions by A. J. Ellis, Dover, N.Y., 1954.
3. D. Bohn, Ed., *Audio Handbook,* National Semiconductor Corp., Santa Clara, Calif., 1976.
4. H. Chamberlin, *Musical Applications of Microprocessors,* Hayden Book Co., Inc., Rochelle Park, N.J., in press.

AMATEUR RADIO

Amateur radio operators are veteran experimenters and seize upon each new electronic device to aid them in their hobby. So it should come as no surprise to find that amateur radio operators are big users of MPUs. AMSAT is presently orbiting a satellite built by amateurs for radio relay work that contains a microcomputer. Similarly, amateurs on the ground are using microcomputers to track the satellite and calculate its position.

However, the widest application is in control of morse code reception and transmission. The microcomputer is an ideal unit for converting morse code to ASCII code, automatically tracking varying received code and displaying it on either a printer or CRT/TVT display. A typical system is shown in Fig. 14-3. On the other hand, it makes transmission simpler and faster by permitting the amateur to use a standard keyboard, putting out ASCII code to convert it automatically to morse code for transmission. An excellent set of articles on these applications can be found in the October 1976 issue of *Byte* magazine.

The reader is also referred to the following amateur radio magazines which have carried numerous articles on the application of MPUs in amateur radio: *QST, CQ, Amateur Radio,* and *73*.

The following nonprofit amateur radio organization is devoted primarily to the application of microprocessors and other new devices in amateur radio applications. They publish a newsletter and maintain a RTTY repeater station (WR4APC, 147.81-MHz input and 147.21-MHz output) and hold regular meetings and training sessions.

Amateur Radio Research & Development Corp. (AMRAD)
1524 Springvale Ave.
McLean, Va., 22101

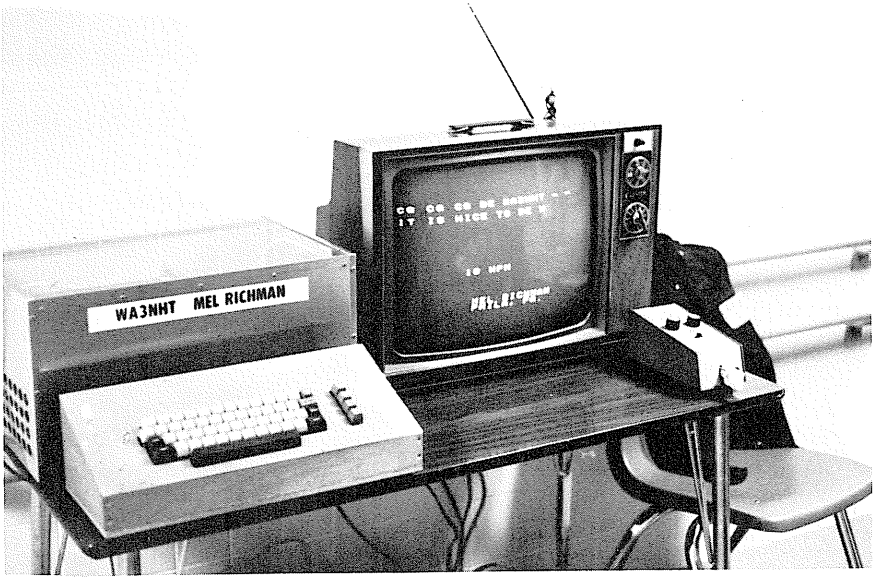


Fig. 14-3. Using a microcomputer to transmit and receive Morse code.

BUSINESS APPLICATIONS

It used to be that a business had to be grossing one million dollars a year or more to justify the purchase of even a minicomputer. After all, even small business-oriented minicomputer systems started at $75K and cost a fortune to operate. But the microcomputer, less expensive peripherals, and easy to use software have changed that radically. Microprocessor-based systems can now be assembled by the user and even serviced, to a great degree, by the user. Gone is that complete dependence on IBM.

Microcomputers are just what the small business user needs—a system scaled down to his needs. The applications are endless. Here are some of the more widely used applications: payroll, cost accounting, accounts payable, accounts receivable, general ledger, sales analysis, investment return, interest calculations, depreciation calculations, loan calculations, inventory control, preparation of financial reports (Balance sheets and Profit and Loss statements), and mailing list maintenance.

All of the foregoing applications are readily programmed in BASIC. In fact, there are several software packages already available for the applications listed. Two of the firms currently supplying such software are:

1. Scientific Research Inst., Box 3692, Crofton, Md., 21114
2. Osborne & Associates, Inc., Box 2036, Berkeley, Calif., 94702.

The following individuals can be contacted for information:

1. *Farm record keeping:* Frank H. Oemig, 501 Oak Park Avenue, Watertown, Wisconsin 53094.
2. *Stock Market Analysis:* Charles Pack, 25470 Elena Road, Los Altos Hills, California 94022.

The hardware necessary to run this software usually includes:

1. CPU with 16–32K of memory, including operating system in ROM and preferably the BASIC interpreter in ROM (particularly if a disc system is not used). Interfaces to the following peripherals should be included in the CPU.
2. CRT terminal
3. High-speed character printer or more preferably a line printer.
4. Dual cassette tape program/data storage system using high speed controllable cassette decks. More preferable is a dual floppy disc system. Either system should be provided with a suitable OS software for the applications for which they will be used.

ROBOTICS

The robots are coming. There are already an estimated 10,000 robots being used in manufacturing plants in this country. They are doing jobs which are either too difficult, too dangerous, or are otherwise undesirable for humans.

Amateur robotics is just beginning. At present there are only a few pioneers experimenting with robotic devices and even building a complete robot. An amateur-built robot is shown in Fig. 14-4. A robot is a vehicle system capable of interacting with its environment in a rational way and managing its own survival. It is a highly sophisticated system combining electronics, mechanics, computer design, computer programming, and artificial intelligence.

Articles describing amateur-built robots have appeared in the following publications:

1. Tod Loofbourrow, "A Computer-Controlled Robot," *Interface Age Magazine,* April 1977. Includes complete schematic diagrams and software for a basic robot controlled by a KIM-1 CPU (6502 MPU).
2. Ralph Hollis, "Newt: A Mobile Cognitive Robot," *Byte,* June 1977. Includes basic, but not complete schematic diagrams and design philosophy for an 8080 MPU controlled robot (still in an early design stage).

An interesting book on the subject is also available:

Tod Loofbourrow, *How to Build a Computer-Controlled Robot,* Hayden Book Co., Inc., Rochelle Park, N.J., 1978.

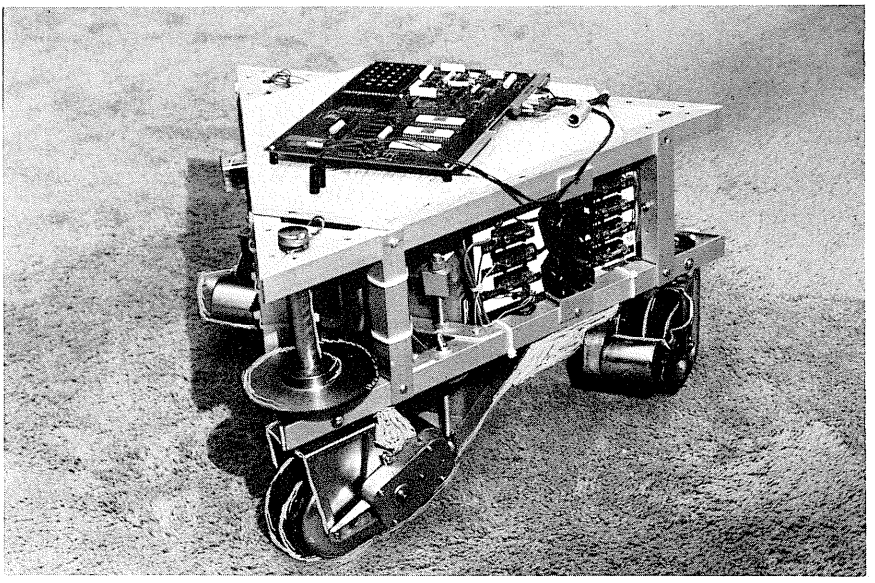


Fig. 14-4. A home-built robot designed and constructed by Tod Loofbourrow.

Although most present robots are programmed to perform a given series of tasks, some are equipped with a few low level sensors.

The future of robotics will probably change radically and quickly. It is expected that very soon robots will be constructed which have considerable ability to interpret sensory information and make decisions accordingly. They will be equipped with vision, including a ranging device, and a set of tactile (force, torque) sensors. They will be able to modify their tasks by making decisions based on information collected from their environment. The likelihood is that the robot will have a main central CPU to handle task planning and other local MPUs to attend to control of and interpretation of sensor functions.

For further information on robotics contact:

United States Robotics Society
Box 26484
Albuquerque, N. Mex. 87102

AUTOMATIC CONTROL

One of the things that computers do very well is controlling repetitive complex processes. Each day more and more of these applications are being introduced in consumer products. We are already acquainted with microproces-

sors in automobiles to optimize gas economy and reduce pollution, and even in microwave ovens to control the cooking cycles.

Here is an example of the automatic sequential control of a chemical production process using an 8080/8085 based CPU. The process is as follows: (1) two liquids are added to a mixing tank, (2) the solution is heated and stirred to allow the reaction to occur, and finally, (3) the tank contents are drained. The following devices are controlled by the CPU via relays:

	Device Codes	
	On	Off
Power switch	00	01
Water valve	02	03
Chemical valve	04	05
Drain valve	06	07
Water pump	08	09
Chemical pump	0A	0B
Stirrer	0C	0D
Heater	0E	0F

The devices must be sequenced for specific periods of time, as indicated by the following control program. Note that a delay subroutine (Wait 01) is used to generate a 1-second time delay and is called to generate the desired sequential time periods.

```
LXI SP, PGMEND+7    Initialize stack address
OUT     00          Apply power at t=0 sec
CALL    WAIT 01     Wait 1 sec
OUT     02          Open water valve at t=1 sec
CALL    WAIT 01     Wait 1 sec
OUT     08          Start water pump at t=2 sec
MVI-D   10          Set D up for18-sec interval
CALL    DELAYD      Wait 18 sec
OUT     09          Stop water pump at t=20 sec
CALL    WAIT 01     Wait 1 sec
OUT     03          Close water valve at t=21 sec
CALL    WAIT 01     Wait 1 sec
OUT     04          Open chemical valve at t=23 sec
MVI-D   06          Set up D for 7-sec interval
CALL    DELAYD      Wait 7 sec
OUT     0B          Stop chemical pump at t=30 sec
CALL    WAIT 01     Wait 1 sec
OUT     05          Close chemical valve at t=31 sec
CALL    WAIT 01     Wait 1 sec
OUT     05          Close chemical valuve at t=31 sec
CALL    WAIT 01     Wait 1 sec
OUT     0E          Start heater at t=32 sec
MVI-D   14          Set up D for 21-sec interval
CALL    DELAYD      Wait 21 sec
```

```
        OUT     0C          Start stirrer
        MVI-D   18          Set up D for 25 sec interval
        CALL    DELAYD      Wait 25 sec
        OUT     0D          Stop stirrer at t=78 sec
        CALL    WAIT 01     Wait 1 sec
        OUT     0F          Stop heater at t=79 sec
        CALL    WAIT 01     Wait 1 sec
        OUT     06          Open drain valve at t=80 sec
        MVI-D   27          Set up D for 40-sec interval
        CALL    DELAYD      Wait 40 sec
        OUT     07          Close drain valve at t=120 sec
        CALL    WAIT 01     Wait 1 sec
        OUT     01          Shut off power at t=121 sec
        HLT                 End
DELAY D CALL    WAIT 01     { Time delay routine WAIT-01 delays
        DCR-D               { 1-sec-DELAY-D delays number in
        JNZ     DELAY D     { Register D + 1 sec
WAIT 01 LXI-B   00 71       { B=113 decimal; C=00H;
NEXT    CALL    RETURN      { WAIT-01 delays 1.0000035 sec
        CALL    RETURN      { DELAY-D delays [(1.0000035)(D+1)
        DCR-C               { (1.6×10^-5)] sec
        JNZ     NEXT
        CMA
        CMA
        CMA
        CMA
        CMA
        DCR-B
        JNZ     NEXT
RETURN  RET
```

Two excellent examples of automatic control in the home will be found in the following two articles:

1. Robert W. Ulrickson, "The Design of a Home Security System," *Electronic Design,* April 26, 1977.
2. Hal Chamberlin, "Try Solar Energy," *Kilobaud,* June 1977.

Appendix A
Hex—ASCII Table

	Control	Numeric	Uppercase	Lowercase	Special
00	Null	30 0	41 A	61 a	20 Space
01	Start of Heading	31 1	42 B	62 b	21 !
02	Start of Text	32 2	43 C	63 c	22 "
03	End of Text	33 3	44 D	64 d	23 #
04	End of Transmission	34 4	45 E	65 e	24 $
05	Enquiry	35 5	46 F	66 f	25 %
06	Acknowledge	36 6	47 G	67 g	26 &
07	Bell	37 7	48 H	68 h	27 '
08	Backspace	38 8	49 I	69 i	28 (
09	Horizontal Tabulation	39 9	4A J	6A j	29)
0A	Line Feed		4B K	6B k	2A *
0B	Vertical Tabulation		4C L	6C l	2B +
0C	Form Feed		4D M	6D m	2C ,
0D	Carriage Return		4E N	6E n	2D –
0E	Shift Out		4F O	6F o	2E .
0F	Shift In		50 P	70 p	2F /
10	Data Link Escape		51 Q	71 q	3A :
11	X-on		52 R	72 r	3B ;
12	Tape		53 S	73 s	3C <
13	X-off		54 T	74 t	3D =
14	Device Control 4		55 U	75 u	3E >
15	Negative Acknowledge		56 V	76 v	3F ?
16	Synchronous Idle		57 W	77 w	40 @
17	End of Transmission Block		58 X	78 x	5B [
18	Cancel		59 Y	79 y	5C \
19	End of Medium		5A Z	7A z	5D]
1A	Substitute				5E ^ or ↑
1B	Escape				5F _ or ←
1C	File Separator				60
1D	Group Separator				7B {
1E	Record Separator				7C |
1F	Unit Separator				7D }
7F	Rub Out or Delete				7E ~

Appendix B
8080 Instruction Codes

JUMP

C3	JMP	Adr
C2	JNZ	
CA	JZ	
D2	JNC	
DA	JC	
E2	JPO	
EA	JPE	
F2	JP	
FA	JM	
E9	PCHL	

CALL

CD	CALL	Adr
C4	CNZ	
CC	CZ	
D4	CNC	
DC	CC	
E4	CPO	
EC	CPE	
F4	CP	
FC	CM	

RETURN

C9	RET
C0	RNZ
C8	RZ
D0	RNC
D8	RC
E0	RPO
E8	RPE
F0	RP
F8	RM

RESTART

C7	RST	0
CF	RST	1
D7	RST	2
DF	RST	3
E7	RST	4
EF	RST	5
F7	RST	6
FF	RST	7

MOVE IMMEDIATE

06	MVI	B D8
0E	MVI	C
16	MVI	D
1E	MVI	E
26	MVI	H
2E	MVI	L
36	MVI	M
3E	MVI	A

Acc IMMEDIATE*

C6	ADI	D8
CE	ACI	
D6	SUI	
DE	SBI	
E6	ANI	
EE	XRI	
F6	ORI	
FE	CPI	

LOAD IMMEDIATE

01	LXI	B D16
11	LXI	D
21	LXI	H
31	LXI	SP

DOUBLE ADD+

09	DAD	B
19	DAD	D
29	DAD	H
39	DAD	SP

STACK OPS

C5	PUSH	B
D5	PUSH	D
E5	PUSH	H
F5	PUSH	PSW
C1	POP	B
D1	POP	D
E1	POP	H
F1	POP	PSW*
E3	XTHL	
F9	SPHL	

INCREMENT**

04	INR	B
0C	INR	C
14	INR	D
1C	INR	E
24	INR	H
2C	INR	L
34	INR	M
3C	INR	A
03	INX	B
13	INX	D
23	INX	H
33	INX	SP

DECREMENT**

05	DCR	B
0D	DCR	C
15	DCR	D
1D	DCR	E
25	DCR	H
2D	DCR	L
35	DCR	M
3D	DCR	A
0B	DCX	B
1B	DCX	D
2B	DCX	H
3B	DCX	SP

LOAD/STORE

0A	LDAX	B
1A	LDAX	D
2A	LHLD	Adr
3A	LDA	Adr
02	STAX	B
12	STAX	D
22	SHLD	Adr
32	STA	Adr

SPECIALS

EB	XCHG
27	DAA*
2F	CMA
37	STC+
3F	CMC+

INPUT/OUTPUT

D3	OUT	D8
DB	IN	D8

ROTATE+

07	RLC
0F	RRC
17	RAL
1F	RAR

CONTROL

00	NOP
76	HLT
F3	DI
FB	EI

MOVE

40	MOV B,B	68	MOV L,B	90	SUB B	B8	CMP B
41	MOV B,C	69	MOV L,C	91	SUB C	B9	CMP C
42	MOV B,D	6A	MOV L,D	92	SUB D	BA	CMP D
43	MOV B,E	6B	MOV L,E	93	SUB E	BB	CMP E
44	MOV B,H	6C	MOV L,H	94	SUB H	BC	CMP H
45	MOV B,L	6D	MOV L,L	95	SUB L	BD	CMP L
46	MOV B,M	6E	MOV L,M	96	SUB M	BE	CMP M
47	MOV B,A	6F	MOV L,A	97	SUB A	BF	CMP A
48	MOV C,B	70	MOV M,B	98	SBB B		
49	MOV C,C	71	MOV M,C	99	SBB C		
4A	MOV C,D	72	MOV M,D	9A	SBB D		
4B	MOV C,E	73	MOV M,E	9B	SBB E		
4C	MOV C,H	74	MOV M,H	9C	SBB H		
4D	MOV C,L	75	MOV M,L	9D	SBB L		
4E	MOV C,M		.....	9E	SBB M		
4F	MOV C,A	77	MOV M,A	9F	SBB A		
50	MOV D,B	78	MOV A,B	A0	ANA B		
51	MOV D,C	79	MOV A,C	A1	AN/ C		
52	MOV D,D	7A	MOV A,D	A2	ANA D		
53	MOV D,E	7B	MOV A,E	A3	ANA E		
54	MOV D,H	7C	MOV A,H	A4	ANA H		
55	MOV D,L	7D	MOV A,L	A5	ANA L		
56	MOV D,M	7E	MOV A,M	A6	ANA M		
57	MOV D,A	7F	MOV A,A	A7	ANA A		

MOVE (cont)

58	MOV E,B	60	MOV H,B
59	MOV E,C	61	MOV H,C
5A	MOV E,D	62	MOV H,D
5B	MOV E,E	63	MOV H,E
5C	MOV E,H	64	MOV H,H
5D	MOV E,L	65	MOV H,L
5E	MOV E,M	66	MOV H,M
5F	MOV E,A	67	MOV H,A

ACCUMULATOR*

80	ADD	B	88	ADC	B	A8	XRA B
81	ADD	C	89	ADC	C	A9	XRA C
82	ADD	D	8A	ADC	D	AA	XRA D
83	ADD	E	8B	ADC	E	AB	XRA E
84	ADD	H	8C	ADC	H	AC	XRA H
85	ADD	L	8D	ADC	L	AD	XRA L
86	ADD	M	8E	ADC	M	AE	XRA M
87	ADD	A	8F	ADC	A	AF	XRA A

80	ADD	B	88	ADC	B	B0	ORA B
81	ADD	C	89	ADC	C	B1	ORA C
82	ADD	D	8A	ADC	D	B2	ORA D
83	ADD	E	8B	ADC	E	B3	ORA E
84	ADD	H	8C	ADC	H	B4	ORA H
85	ADD	L	8D	ADC	L	B5	ORA L
86	ADD	M	8E	ADC	M	B6	ORA M
87	ADD	A	8F	ADC	A	B7	ORA A

D8 = constant or logical arithmetic expression that evaluates to an 8-bit data quantity
D16 = constant or logical arithmetic expression that evaluates to a 16-bit data quantity
Adr = 16-bit address
\+ = only CARRY affected
* = all Flags (C.Z.S.P.) affected
** = all Flags except CARRY affected (exception, INX and DCX affect no Flags)

Appendix C
8080 Codes—Numerical Order

HEX	OCT	MNEM
00	000	NOP
01	001	LXI B
02	002	STAX B
03	003	INX B
04	004	INR B
05	005	DCR B
06	006	MVI B
07	007	RLC
08	010	Unimpl
09	011	DAD B
0A	012	LDAX B
0B	013	DCX B
0C	014	INR C
0D	015	DCR C
0E	016	MVI C
0F	017	RRC
10	020	Unimpl
11	021	LXI D
12	022	STAX D
13	023	INX D
14	024	INR D
15	025	DCR D
16	026	MVI D
17	027	RAL
18	030	Unimpl
19	031	DAD D
1A	032	LDAX D
1B	033	DCX D
1C	034	INR E
1D	035	DCR E
1E	036	MVI E
1F	037	RAR
20	040	Unimpl
21	041	LXI H
22	042	SHLD
23	043	INX H
24	044	INR H
25	045	DCR H
26	046	MVI H
27	047	DAA
28	050	Unimpl
29	051	DAD H
2A	052	LHLD
2B	053	DCX H
2C	054	INR L
2D	055	DCR L
2E	056	MVI L
2F	057	CMA
30	060	Unimpl
31	061	LXI SP
32	062	STA
33	063	INX SP
34	064	INR M
35	065	DCR M
36	066	MVI M
37	067	STC
38	070	Unimpl
39	071	DAD SP
3A	072	LDA
3B	073	DCX SP
3C	074	INR ACC
3D	075	DCR ACC
3E	076	MVI ACC
3F	077	CMC
40	100	MOV B,B
41	101	MOV B,C
42	102	MOV B,D
43	103	MOV B,E
44	104	MOV B,H
45	105	MOV B,L
46	106	MOV B,M
47	107	MOV B,ACC
48	110	MOV C,B
49	111	MOV C,C
4A	112	MOV C,D
4B	113	MOV C,E
4C	114	MOV C,H
4D	114	MOV C,L
4E	115	MOV C,M
4F	117	MOV C,ACC
50	120	MOV D,B
51	121	MOV D,C
52	122	MOV D,D
53	123	MOV D,E
54	124	MOV D,H
55	125	MOV D,L
56	126	MOV D,M
57	127	MOV D,ACC
58	130	MOV E,B
59	131	MOV E,C
5A	132	MOV E,D
5B	133	MOV E,E
5C	134	MOV E,H
5D	135	MOV E,L
5E	136	MOV E,M
5F	137	MOV E,ACC
60	140	MOV H,B
61	141	MOV H,C
62	142	MOV H,D
63	143	MOV H,E
64	144	MOV H,H
65	145	MOV H,L
66	146	MOV H,M
67	147	MOV H,ACC
68	150	MOV L,B
69	151	MOV L,C
6A	152	MOV L,D
6B	153	MOV L,E
6C	154	MOV L,H
6D	155	MOV L,L
6E	156	MOV L,M

HEX	OCT	MNEM
6F	157	MOV L,ACC
70	160	MOV M,B
71	161	MOV M,C
72	162	MOV M,D
73	163	MOV M,E
74	164	MOV M,H
75	165	MOV M,L
76	166	HLT
77	167	MOV M,ACC
78	170	MOV ACC,B
79	171	MOV ACC,C
7A	172	MOV ACC,D
7B	173	MOV ACC,E
7C	174	MOV ACC,H
7D	175	MOV ACC,L
7E	176	MOV ACC,M
7F	177	MOV ACC,A
80	200	ADD B
81	201	ADD C
82	202	ADD D
83	203	ADD E
84	204	ADD H
85	205	ADD L
86	206	ADD M
87	207	ADD ACC
88	210	ADC B
89	211	ADC C
8A	212	ADC D
8B	213	ADC E
8C	214	ADC H
8D	215	ADC L
8E	216	ADC M
8F	217	ADC ACC
90	220	SUB B
91	221	SUB C
92	222	SUB D
93	223	SUB E
94	224	SUB H
95	225	SUB L
96	226	SUB M
97	227	SUB ACC
98	230	SBB B
99	231	SBB C
9A	232	SBB D
9B	233	SBB E
9C	234	SBB H
9D	235	SBB L
9E	236	SBB M
9F	237	SBB ACC
A0	240	ANA B
A1	241	ANA C
A2	242	ANA D
A3	243	ANA E
A4	244	ANA H
A5	245	ANA L
A6	246	ANA M
A7	247	ANA ACC
A8	250	XRA B
A9	251	XRA C
AA	252	XRA D
AB	253	XRA E
AC	254	XRA H
AD	255	XRA L
AE	256	XRA M
AF	257	XRA ACC
B0	260	ORA B
B1	261	ORA C
B2	262	ORA D
B3	263	ORA E
B4	264	ORA H
B5	265	ORA L
B6	266	ORA M
B7	267	ORA ACC
B8	270	CMP B
B9	271	CMP C
BA	272	CMP D
BB	273	CMP E
BC	274	CMP H
BD	275	CMP L
BE	276	CMP M
BF	277	CMP ACC
C0	300	RNZ
C1	301	POP B
C2	302	JNZ
C3	303	JMP
C4	304	CNZ
C5	305	PUSH B
C6	306	ADI
C7	307	RST 0
C8	310	RZ
C9	311	RET
CA	312	JZ
CB	313	Unimpl
CC	314	CZ
CD	315	CALL
CE	316	ACI
CF	317	RST 1
D0	320	RNC
D1	321	POP D
D2	322	JNC
D3	323	OUT
D4	324	CNC
D5	325	PUSH D
D6	326	SUI
D7	327	RST 2
D8	330	RC
D9	331	Unimpl
DA	332	JC
DB	333	IN
DC	334	CC
DD	335	Unimpl
DE	336	SBI
DF	337	RST 3
E0	340	RPO
E1	341	POP H
E2	342	JPO
E3	343	XTHL
E4	344	CPO
E5	345	PUSH H
E6	346	ANI
E7	347	RST 4
E8	350	RPE
E9	351	PCHL
EA	352	JPE
EB	353	XCHG
EC	354	CPE
ED	355	Unimpl
EE	356	XRI
EF	357	RST 5
F0	360	RP
F1	361	POP PSW
F2	362	JP
F3	363	DI
F4	364	CP
F5	365	PUSH PSW
F6	366	ORI
F7	367	RST 6
F8	370	RM
F9	371	SPHL
FA	372	JM
FB	373	EI
FC	374	CM
FD	375	Unimpl
FE	376	CPI
FF	377	RST 7

Appendix D
Personal Computer Magazines

The following is a listing of some of the magazines referred to in this book. Additional and current information on the topics discussed in this book will be found in them. Note that *Electronic Design* and *EDN* magazines are sent free-of-charge to qualified readers.

Byte
70 Main St.
Peterborough NH 03458

Dr. Dobb's Journal
Box 310
Menlo Park, CA 94025

Electronic Design
Hayden Publishing Co., Inc.
50 Essex St.
Rochelle Park, NJ 07662

Kilobaud
1001001 Inc.
Peterborough, NH 03458

Personal Computing
Benwill Publishing Corp.
167 Corey Rd.
Brookline, MA 02146

Radio Electronics
Gernsback Publications, Inc.
200 Park Ave. So.
New York, NY 10003

Creative Computing
Box 789-M
Morristown, NJ 07960

EDN
Cahners Publishing Co.
Denver, CO 80206

Interface Age
McPheters, Wolfe & Jones
13913 Artesia Blvd.
Cerritos, CA 90701

People's Computer Co.
Box 310
Menlo Park, CA 94025

Popular Electronics
1 Park Ave.
New York, NY 10016

ROM
Route 97
Hampton, CT 06247

Index

SMALL COMPUTER SYSTEMS HANDBOOK

Sol Libes

You have in your hands the primer written for those new to the field of personal home computers. The emphasis throughout is on the important practical knowledge that the small computer user should have to be able to intelligently purchase, assemble, and interconnect components, and to program the microcomputer.

This guide provides the necessary background in digital logic fundamentals, number systems, and computer hardware and software basics. Only a minimal knowledge of electronics is required. Further, the book offers an introduction to programming on the machine level and with higher level languages such as BASIC.

Finally, the various applications of the small computer are described, such as maintaining financial records, storing records, controlling appliances, typewriting, sales analysis, and inventory control.

Other Books of Interest . . .

HOW TO BUILD A COMPUTER-CONTROLLED ROBOT

Tod Loofbourrow

Step-by-step directions for constructing a robot controlled by a KIM-1 microprocessor. The complete control programs are clearly written out. Photographs, diagrams, and tables direct you through the construction. #5681-8, paper, 144 pages.

BASIC BASIC: An Introduction to Computer Programming in BASIC Language, Second Edition and ADVANCED BASIC: Applications and Problems

Both by James S. Coan

The complete picture of the BASIC language. One introduces the language through an integration of programming and the teaching of mathematics. The other offers advanced techniques and applications. Basic BASIC, #5106-9, paper, #5107-7, cloth, 288 pages; Advanced BASIC, #5855-1, paper, #5856-X, cloth, 192 pages.

HOW TO PROFIT FROM YOUR PERSONAL COMPUTER Professional, Business, and Home Applications

T. G. Lewis

Shows you how to put your personal computer to work for you in common business applications, such as accounting, handling payrolls, inventory, and sorting mailing lists. Includes many BASIC language programs to illustrate the techniques. #5761-X, paper, 208 pages.

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

ISBN 0-8104-5678-8